

```
1  /* *****
2      CUNY ACE UPSKILLING:  INTRODUCTION TO STRUCTURED QUERY LANGUAGE
3          SF21JOB#2, 2021/11/08 to 2021/12/13
4          https://folvera.commonsgc.cuny.edu/?cat=30
5  *****
6
7  SESSION #5 (2021/11/22): MANIPULATING DATA
8
9  1. Using clauses `BETWEEN`, `NOT`, `UNION`, `EXCEPT` and `INTERSECT`
10 2. Understanding function `FORMAT()` for dates and currencies including
11   culture codes
12 *****
13
14 1. Before you continue learning about SQL
15   (https://searchsqlserver.techtarget.com/definition/SQL) syntax
16   (https://whatis.techtarget.com/definition/syntax), we should cover some
17   important theory, which you will need whether you need to learn SQL to run
18   queries at work and/or you decide to become a database administrator (DBA).
19
20 1.1. SQL (Structured Query Language) is a standardized programming language
21   used for managing relational databases and performing various
22   operations on the data in them. Initially created in the 1970s, SQL
23   is regularly used by database administrators, as well as by
24   developers writing data integration scripts and data analysts looking
25   to set up and run analytical queries.
26   https://searchsqlserver.techtarget.com/definition/SQL
27
28 1.2. ISO/IEC 9075-1:2016 [SQL:2016] describes the conceptual framework used
29   in other parts of ISO/IEC 9075 to specify the grammar of SQL and the
30   result of processing statements in that language by an
31   SQL-implementation.
32   ISO/IEC 9075-1:2016 also defines terms and notation used in the other
33   parts of ISO/IEC 9075.
34   https://www.iso.org/standard/63555.html
35
36 1.3. T-SQL (Transact-SQL) is a set of programming extensions from Sybase
37   and Microsoft that add several features to the Structured Query
38   Language (SQL), including transaction control, exception and error
39   handling, row processing and declared variables.
40   https://searchsqlserver.techtarget.com/definition/T-SQL
41
42 1.4. A relational database is a set of tables containing data fitted into
43   predefined categories. Each table (which is sometimes called a
44   relation) contains one or more data categories in columns. Each row
45   contains a unique instance of data for the categories defined by the
46   columns.
47   http://searchsqlserver.techtarget.com/definition/relational-database
48
49 1.5. Microsoft SQL Server is a relational database management system, or
50   RDBMS, that supports a wide variety of transaction processing,
51   business intelligence and analytics applications in corporate IT
52   environments. It's one of the three market-leading database
```

53 technologies, along with Oracle Database and IBM's DB2.
54 Like other RDBMS software, Microsoft SQL Server is built on top of
55 SQL, a standardized programming language that database administrators
56 (DBAs) and other IT professionals use to manage databases and query
57 the data they contain. SQL Server is tied to Transact-SQL (T-SQL), an
58 implementation of SQL from Microsoft that adds a set of proprietary
59 programming extensions to the standard language.
60 The original SQL Server code was developed in the 1980s by the former
61 Sybase Inc., which is now owned by SAP. Sybase initially built the
62 software to run on Unix systems and minicomputer platforms. It,
63 Microsoft and Ashton-Tate Corp., then the leading vendor of PC
64 databases, teamed up to produce the first version of what became
65 Microsoft SQL Server, designed for the OS/2 operating system and
66 released in 1989.
67 <https://searchsqlserver.techtarget.com/definition/SQL-Server>
68

69 1.6. Another form of flat file is one in which table data is gathered in
70 lines of ASCII text with the value from each table cell separated by
71 a comma and each row represented with a new line. This type of flat
72 file is also known as a comma-separated values file (CSV) file.
73 <http://searchsqlserver.techtarget.com/definition/flat-file>
74

75 1.7. A hierarchical database is a design that uses a one-to-many
76 relationship for data elements. Hierarchical database models use a
77 tree structure that links a number of disparate elements to one
78 `owner,` or `parent,` primary record.
79 <https://www.techopedia.com/definition/19782/hierarchical-database>
80

81 1.8. Data Manipulation Language (DML) is the ``vocabulary used to retrieve
82 and work with data... to add, modify, query, or remove data``
83 (<https://msdn.microsoft.com/en-us/library/ff848766.aspx>).
84

85 1.8.1. SELECT to retrieve records from one or more tables
86 <https://techonthenet.com/sql/select.php>
87

88 1.8.2. INSERT to insert a one or more records into a table
89 <https://techonthenet.com/sql/insert.php>
90

91 1.8.3. UPDATE to update existing records in the tables
92 <https://techonthenet.com/sql/update.php>
93

94 1.8.4. DELETE to delete a one or more records from a table
95 <https://techonthenet.com/sql/delete.php>
96

97 1.8.5. MERGE to insert, update, or delete operations on a target
98 table based on the results of a join with a source
99 table
100 <https://msdn.microsoft.com/en-us/library/bb510625.aspx>
101

102 1.9. Data Definition Language (DDL) is the ``vocabulary used to define data
103 structures... to create, alter, or drop data structures``
104 (<https://msdn.microsoft.com/en-us/library/ff848799.aspx>).

105
106 1.9.1. USE to select any existing database in SQL schema [or
107 output from another query]
108 <http://tutorialspoint.com/sql/sql-select-database.htm>
109
110 1.9.2. CREATE to create and define a table [or other database
111 object]
112 https://techonthenet.com/sql/tables/create_table.php
113
114 1.9.3. ALTER to add a column, modify a column, drop a column,
115 rename a column or rename a table [or other database
116 object]
117 https://techonthenet.com/sql/tables/alter_table.php
118
119 1.9.4. DROP to remove or delete a table [or other database
120 object]
121 https://techonthenet.com/sql/tables/drop_table.php
122
123 1.9.5. TRUNCATE to remove all records from a table
124 <https://techonthenet.com/sql/truncate.php>
125
126 1.9.6. DELETE to delete a one or more records from a table
127 <https://techonthenet.com/sql/delete.php>
128
129 1.10. Note that some of these statements can do more than what is covered
130 in these notes for our first sessions.
131
132 For example, the `CREATE` statement is also used to create other
133 database objects as well as access management, but we will not cover
134 these other statements yet. Refer to
135 <https://msdn.microsoft.com/en-us/library/cc879262.aspx> for more
136 information on the `CREATE` statement.
137
138 On a personal note, when looking for information and/or explanation
139 on how to use Microsoft technologies, in this case SQL Server, go to
140 <https://techonthenet.com/> or <http://tutorialspoint.com/> as
141 <https://msdn.microsoft.com/> and other Microsoft websites often seem
142 to be written for advanced users.
143
144 We will use DML and DDL in detail later in the course.
145
146 2. There are several data types
147 (<https://msdn.microsoft.com/en-us/library/ms187752.aspx>) that you need to
148 know if you are interested in taking the certification exam for Database
149 Fundamentals. In everyday use, these are the most often used data types in
150 T-SQL (<http://searchsqlserver.techtarget.com/definition/T-SQL>) -- the
151 version of SQL (<http://searchsqlserver.techtarget.com/definition/SQL>) used
152 in SQL Server (<http://searchsqlserver.techtarget.com/definition/SQL-Server>)
153 -- are the following.
154
155 2.1. INT -2³¹ (-2,147,483,648) to 2³¹-1 (2,147,483,647)
156 <https://technet.microsoft.com/en-us/library/ms187745.aspx>

157
 158 2.2. DECIMAL fixed precision and scale numbers, 10³⁸+1 through 10³⁸-1
 159 <https://msdn.microsoft.com/en-us/library/ms187746.aspx>
 160 * instead of DOUBLE or FLOAT, indicating the whole value
 161 followed by the number of decimals where pi(1,10) can
 162 hold 3.1415926536, but not 3.14159265359 for eleven (11)
 163 decimal spaces
 164
 165 2.3. VARCHAR(n) 2³¹-1 bytes (2 GB); variable-length, ASCII
 166 (<http://whatis.techtarget.com/definition/ASCII-American-Standard-Code-for-Information-Interchange>)
 167 string data
 168 <https://technet.microsoft.com/en-us/library/ms176089.aspx>
 169 not to be confused with NVARCHAR(n) -- variable-length,
 170 2³¹-1 bytes (2 GB), Unicode
 171 (<http://whatis.techtarget.com/definition/Unicode>) string
 172 data, not part of most relational database management
 173 systems (RDBMS)
 174 <https://technet.microsoft.com/en-us/library/ms186939.aspx>
 175
 176 2.4. DATE date
 177 <https://technet.microsoft.com/en-us/library/bb630352.aspx>
 178
 179 2.5. TIME time
 180 <https://technet.microsoft.com/en-us/library/bb677243.aspx>
 181
 182 2.6. DATETIME defines a date that is combined with a time of day with
 183 fractional seconds that is based on a 24-hour clock
 184 <https://technet.microsoft.com/en-us/library/ms187819.aspx>
 185
 186 2.7. MONEY money, not part of most relational database management
 187 systems (RDBMS)
 188 <https://technet.microsoft.com/en-us/library/ms179882.aspx>
 189
 190 2.8. 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)

208	DATETIME to DATE	date only; time dropped	
209	+-----+-----+-----+		
210	DATETIME to TIME	time only; date dropped	
211	+-----+-----+-----+		
212	DATE to DATETIME	date with default value of	
213		`00:00.00.000`	
214	+-----+-----+-----+		
215	TIME to DATETIME	time with default value of	
216		`1900/01/01`	
217	+-----+-----+-----+		
218	INT	converted to text; no longer	
219	DECIMAL	numeric data and cannot be used	
220	DATETIME to VARCHAR	in mathematical calculations	
221	DATE	NVARCHAR	
222	TIME		
223	+-----+-----+-----+		
224		INT	straight conversion to proper
225		DECIMAL	data type as long as the string
226	VARCHAR to DATETIME	field only has numbers and	
227	NVARCHAR	DATE	structure is correct (for
228		TIME	example, text with value of
229			`2019/03/11` to DATE); no
230			conversion if the string has
231			letters or special characters
232	+-----+-----+-----+		
233	VARCHAR to NVARCHAR	straight conversion; no data	
234		loss	
235	+-----+-----+-----+		
236	NVARCHAR to VARCHAR	straight conversion if string is	
237		encoded as ACIII or UTF-8;	
238		possible data loss if string is	
239		encoded as Unicode or no	
240		conversion at all	
241	+-----+-----+-----+		

242

243 2.9. Refer to <https://technet.microsoft.com/en-us/library/ms187912.aspx> for
 244 information on approximate numeric data types -- FLOAT and REAL. If
 245 you are considering taking the certification, you should know the
 246 concept below and why Microsoft recommends not using approximate
 247 numeric data types.

248

249 ``The float and real data types are known as approximate data
 250 types. The behavior of float and real follows the IEEE 754
 251 specification on approximate numeric data types. Approximate
 252 numeric data types do not store the exact values specified for many
 253 numbers; they store an extremely close approximation of the value.
 254 For many applications, the tiny difference between the specified
 255 value and the stored approximation is not noticeable. At times,
 256 though, the difference becomes noticeable. Because of the
 257 approximate nature of the float and real data types, do not use
 258 these data types when exact numeric behavior is required, such as
 259 in financial applications, in operations involving rounding, or in

260 equality checks. Instead, use the integer, decimal, money, or
 261 smallmoney data types.
 262 Avoid using float or real columns in WHERE clause search
 263 conditions, especially the = and <> operators. It is best to limit
 264 float and real columns to > or < comparisons. The IEEE 754
 265 specification provides four rounding modes: round to nearest, round
 266 up, round down, and round to zero. Microsoft SQL Server uses round
 267 up. All are accurate to the guaranteed precision but can result in
 268 slightly different floating-point values. Because the binary
 269 representation of a floating-point number may use one of many legal
 270 rounding schemes, it is impossible to reliably quantify a
 271 floating-point value.``
 272 <https://technet.microsoft.com/en-us/library/ms187912.aspx>

273
 274 Note that FLOAT is commonly used in other relational database
 275 management systems (RDBMS) like Oracle (<http://oracle.com/>) and in
 276 most programming languages including those distributed by Microsoft.
 277

278 3. As we start, we keep in mind that the most basic structure of a `SELECT`
 279 statement (<https://techonthenet.com/sql/select.php>) is the following.

```
280
281     SELECT field1, field2...
282     FROM   table1
```

283
 284 From the previous structure, you can add clauses in the following order.
 285 If you organize the clauses any other order, the query will not work.

```
286
287     SELECT table1.field1,      -- 1. calling columns/fields
288           table1.field2,      --   (data)
289           ...
290           table2.field1,
291           table2.field2,
292           ...
293           table3.field1,
294           table3.field2,
295           ...
296
297     FROM table1                -- 2. where to find data
298                                --   (tables/views)
299     INNER|LEFT|RIGHT JOIN table2
300         ON table1.shared_field1 = table2.shared_field1
301        AND table1.shared_field2 = table2.shared_field2
302         ...
303     INNER|LEFT|RIGHT JOIN table3
304         ON table1.shared_field1 = table3.shared_field1
305        AND table1.shared_field2 = table3.shared_field2
306         ...
307
308     WHERE condition1          -- 3. filtering output, what
309           AND|OR condition2  --   rows/records you want to
310           AND|OR condition3  --   retrieve
311           ...
```

```
312
313         GROUP BY table1.field1,      -- 4. grouping fields not in an
314                table1.field2,      --   aggregate function
315                ...
316                table2.field1,
317                table2.field2,
318                ...
319                table3.field1,
320                table3.field2,
321                ...
322
323         ORDER BY                    -- 5. organizing rows/records
324                table1.field1 ASC|DESC, -- (output) in ascending
325                table1.field2 ASC|DESC, -- (`ASC`) or descending
326                ...                  -- (`DESC`) order
327                table2.field1 ASC|DESC,
328                table2.field2 ASC|DESC,
329                ...
330                table3.field1 ASC|DESC,
331                table3.field2 ASC|DESC,
332                ...
333
334 3. In the example below, we retrieve all (`*`) columns from table
335   `AP1.Vendors`.
336 ***** */
337
338 SELECT *
339 FROM AP1.Vendors;                -- retrieves all values from
340                                  -- table `AP1.Vendors`
341
342
343 /* *****
344    3.1. The only time you can use `SELECT` without `FROM` is when you want the
345        machine to return a value, similar to `PRINT`.
346 ***** */
347
348 SELECT 9 * 8;                    -- returns integer 72 (a
349                                  -- mathematical equation)
350
351 SELECT 'Hello there';           -- returns string `Hello there`
352                                  -- (a simple string)
353
354
355 /* *****
356    3.2. As you can see in the examples above, we are not retrieving data from
357        any table. You can get the same results using `PRINT`.
358 ***** */
359
360 PRINT 9 * 8;                    -- prints integer 72 (a
361                                  -- mathematical equation)
362
363 PRINT 'Hello there';           -- prints string `Hello there`
```

```
364 -- (a simple string)
365
366
367 /* *****
368 4. We have covered built-in functions that affect strings.
369
370 4.1. CONCAT() allows you to concatenate strings together
371 https://techonthenet.com/sql_server/functions/concat.php
372
373 '+' allows you to concatenate 2 or more strings together
374 https://techonthenet.com/sql_server/functions/concat2.php
375
376 4.2. LEFT() allows you to extract a substring from a string, starting
377 from the left-most character
378 https://techonthenet.com/sql_server/functions/left.php
379
380 4.3. LEN() returns the length of the specified string... does not
381 include trailing space characters at the end the string
382 when calculating the length
383 https://techonthenet.com/sql_server/functions/len.php
384
385 4.4. LTRIM() removes all space characters from the left-hand side of a
386 string
387 https://techonthenet.com/sql_server/functions/ltrim.php
388
389 4.5. LOWER() converts all letters in the specified string to lowercase
390 https://techonthenet.com/sql_server/functions/lower.php
391
392 4.6. REPLACE() replaces a sequence of characters in a string with another
393 set of characters, not case-sensitive
394 https://techonthenet.com/sql_server/functions/replace.php
395
396 4.7. RIGHT() allows you to extract a substring from a string, starting
397 from the right-most character
398 https://techonthenet.com/sql_server/functions/right.php
399
400 4.8. RTRIM() removes all space characters from the right-hand side of a
401 string
402 https://techonthenet.com/sql_server/functions/rtrim.php
403
404 4.9. SUBSTRING allows you to extract a substring from a string
405 https://techonthenet.com/sql_server/functions/substring.php
406
407 4.10. UPPER() converts all letters in the specified string to uppercase
408 https://techonthenet.com/sql_server/functions/upper.php
409
410 5. Now we will see functions used with numeric values.
411
412 5.1. AVG() returns the average value of an expression
413 https://techonthenet.com/sql_server/functions/avg.php
414
415 5.2. CEILING() returns the smallest integer value that is greater than or
```




```

468                                     --          1/10/2012 ...
469  FORMAT(InvoiceDueDate, 'D', 'en-us') -- `D` (upper case) for long
470  AS InvoiceDueDate,                  -- date returning full day of
471                                     -- the week, full month, no
472                                     -- leading zeros with culture
473                                     -- `en-us` (English US);
474                                     -- returns
475                                     --    Sunday, January 8, 2012
476                                     --    Tuesday, January 10, 2012
477                                     --    ...
478  FORMAT(InvoiceDueDate, 'MM/dd/yyyy', 'en-us') -- custom date using format
479  AS InvoiceDueDate                    -- `MM/dd/yyyy` which overrides
480                                     -- culture `en-us` (English
481                                     -- US); returns 01/08/2012
482                                     --          01/10/2012 ...
483 FROM AP1.Invoices
484 GROUP BY InvoiceTotal,
485          AP1.Invoices.InvoiceDueDate
486
487
488 /* *****
489  6.1. When using an aggregate function, we must use `GROUP BY` and list all
490  columns not in affected by any aggregate function.
491
492  In the example below, we retrieve `VendorState` plus the count of
493  column `VendorState` for each `VendorState` (`COUNT(VendorState)`).
494
495  We can use `DISTINCT` to make sure that duplicate values (rows) are
496  not included in the output of a query.
497
498  We can use `ORDER BY` to organize output by a specific column or list
499  of columns.
500
501  The default option for `ORDER BY` is ascending (`ASC`), which can be
502  omitted (1, 2, 3... a, b, c...).
503
504  The opposite option for `ORDER BY` is descending (`DESC`), which must
505  be used if needed (...3, 2, 1 ...c, b, a).
506  ***** */
507
508 SELECT DISTINCT                       -- 1. to avoid duplicates
509    VendorState,                       -- 2. column not in aggregate
510                                     -- function
511    COUNT(VendorState)                 -- 3. column in aggregate
512                                     -- function (calculation)
513 FROM AP1.Vendors                      -- 4. from table `AP1.Vendors`
514 GROUP BY VendorState                 -- 5. must use `GROUP BY` when
515                                     -- using any aggregate
516                                     -- function, listing all
517                                     -- columns not in the
518                                     -- aggregate function
519 ORDER BY VendorState ASC;            -- 6. organizing results by


```

```
520 -- column `VendorState` in
521 -- ascending order
522
523
524 /* *****
525 6.2. In the example below, we retrieve `VendorID` plus the sum of column
526 `PaymentTotal` for each `VendorID` (`SUM(PaymentTotal)`).
527 ***** */
528
529 SELECT DISTINCT -- 1. to avoid duplicates
530 VendorID, -- 2. column not in aggregate
531 -- function
532 SUM(PaymentTotal) -- 3. column in aggregate
533 -- function (calculation)
534 FROM AP1.Invoices -- 4. from table `AP1.Invoices`
535 GROUP BY VendorID -- 5. must use `GROUP BY` when
536 -- using any aggregate
537 -- function, listing all
538 -- columns not in the
539 -- aggregate function
540 ORDER BY VendorID DESC; -- 6. organizing results by
541 -- column `VendorID` in
542 -- descending order
543
544
545 /* *****
546 7. In the example below, the query returns all values from the `AP1.Vendors`
547 table with all related values from table `AP1.Invoices`,
548 `AP1.InvoiceLineItems` and `AP1.Terms`.
549
550 7.1. The relation between related tables `AP1.Invoices`,
551 `AP1.InvoiceLineItems` and `AP1.Terms` is `INNER JOIN` since the value
552 (row ID) of one table is referenced in another.
553
554 7.2. Dollar amounts are formatted as `c` (currency) with culture `en-us`
555 (English-United States). Dates are formatted as `MM/dd/yyyy` (two
556 digits for month and day, four digits for year) and culture `en-us`
557 (English-United States). Refer to
558 https://msdn.microsoft.com/en-us/library/hh213506.aspx for more
559 information. Note that formatting a numeric value changes it to an
560 alpha-numeric value -- change in data type.
561
562 7.3. To include the average value of `InvoiceTotal` of all records from
563 table `AP1.Invoices`, we use a sub-query (also referred to as nested
564 query, http://tutorialspoint.com/sql/sql-sub-queries.htm). We use
565 alias `AvgInvoiceTotal` to refer to this new column.
566
567 (
568 SELECT FORMAT(AVG(AP1.Invoices.InvoiceTotal),'c','en-us')
569 FROM AP1.Invoices
570 )
571 AS AvgInvoiceTotal
```

572
 573 There are various values for culture (one per language and country
 574 combination). The following are just a few, probably the most common
 575 in American businesses. Refer to
 576 [http://sql-server-helper.com/sql-server-2012/format-string-function-
 culture.aspx](http://sql-server-helper.com/sql-server-2012/format-string-function-culture.aspx) 
 577 for a more detailed list of cultures.

CULTURE	LANGUAGE	COUNTRY	RESULT
en-us	English	USA	dollar
en-gb	English	Great Britain	pound
de-de	German	Germany	euro
zh-cn	Simplified Chinese	China	yuan
jp-jp	Japanese	Japan	yen

593
 594 Refer to <https://www.iso.org/iso-4217-currency-codes.html> for
 595 more information on currency codes (ISO 4217).
 596

597 When formatting DATETIME fields, you can use any of the formats below
 598 and the culture (`en-us`). The default format in data type DATETIME
 599 is `yyyy-MM-dd hh:mm:ss.nnnnnn`. Refer to
 600 [https://docs.microsoft.com/en-us/sql/t-sql/functions/datetime-transact-
 sql](https://docs.microsoft.com/en-us/sql/t-sql/functions/datetime-transact-sql) 
 601 for more information about dates.

OPTION	OUTPUT	FORMAT
c	currency depending on culture (`\$`)	`c`, `en-us`
d	day without leading zero, day without leading zero and complete year (11/23/2021)	`d`, `en-us`
D	whole day of the week, first letter capitalized;	`D`, `en-us`

622		whole month,	
623		first letter	
624		capitalized;	
625		day without	
626		leading zero	
627		and complete	
628		year (Monday,	
629		November 22,	
630		2021)	
631	+-----+		
632	+-----+		
633			
634	DATEPART	OUTPUT	FORMAT
635	+-----+		
636	dw	whole day of	`dw MMMM dd, yyyy`
637		the week,	`dw MMMM d, yyyy`
638		first letter	`dw MMMM dd, yy`
639		capitalized	`dw MMMM d, yy`
640		(Monday)	
641	+-----+		
642	MMMM	whole month,	`MMMM dd, yyyy`
643		first letter	`MMMM d, yyyy`
644		capitalized	`MMMM dd, yy`
645		(November)	`MMMM d, yy`
646	+-----+		
647	MMM	month in	`MMM dd, yyyy`
648		abbreviation,	`MMM d, yyyy`
649		first letter	`MMM dd, yy`
650		capitalized	`MMM d, yy`
651		(Nov)	`dd-MMM-yy` (default Oracle)
652			`d-MMM-yy` (default Oracle)
653	+-----+		
654	MM	month number	`MM/dd/yyyy`
655		with leading	`MM/d/yyyy`
656		zero (11)	`MM/dd/yy`
657			`MM/d/yy`
658	+-----+		
659	M	month number	`M/dd/yyyy`
660		without	`M/d/yyyy`
661		leading zero	`M/dd/yy`
662		(22)	`M/d/yy`
663	+-----+		
664	dddd	day of week	`dddd, MMM d, yyyy`
665		(Monday)	`dddd, MMMM d, yyyy`
666	+-----+		
667	ddd	day of week	`ddd, MMM d, yyyy`
668		abbreviation	} `ddd, MMMM d, yyyy`
669		(Mon)	
670	+-----+		
671	dd	day with	`MM/dd/yyyy`
672		leading zero	`M/dd/yyyy`
673		(23)	`MM/dd/yy`

674			`M/dd/yy`
675			
676	d	day without	`MM/d/yyyy`
677		leading zero	`M/d/yyyy`
678		(23)	`MM/d/yy`
679			`M/d/yy`
680			
681	yy	last two	`M/dd/yy`
682		digits of year	`M/d/yy`
683		(21)	`MM/d/yy`
684			`M/d/yy`
685			
686	yyyy	complete year	`M/dd/yyyy`
687		(2019)	`M/d/yyyy`
688			`MM/d/yyyy`
689			`M/d/yyyy`
690			
691	HH	24-hour,	`HH:mm:ss`
692		military time	
693		with leading	
694		zero (20)	
695			
696	H	24-hour,	`H:mm:ss`
697		military time	
698		without	
699		leading zero	
700		(20)	
701			
702	hh	12-hour	`hh:mm:ss`
703		(AM/PM), with	
704		leading zero	
705		(08 PM)	
706			
707	h	12-hour	`h:mm:ss`
708		(AM/PM),	
709		without	
710		leading zero	
711		(8 PM)	
712			
713	mm	minutes (13)	`HH:mm:ss`
714			`H:mm:ss`
715	ss	seconds (58)	`hh:mm:ss`
716			`h:mm:ss`
717			
718	nnnnnn	six decimal	`HH:mm:ss.nnnnnn`
719		spaces,	`H:mm:ss.nnnnnn`
720		fractions of	`hh:mm:ss.nnnnnn`
721		a second	`h:mm:ss.nnnnnn`
722			

723

724 Although we are using aggregate function `AVG()`, we do not need to
 725 use `GROUP BY` since the function is inside the sub-query.

```

726
727         Go to https://docs.microsoft.com/en-us/sql/t-sql/functions/format-
728         transact-sql
729         for more information on `FORMAT()`.
730 ***** */
731 SELECT DISTINCT AP1.Vendors.VendorID,
732 AP1.Vendors.VendorName,
733 CONCAT (
734     AP1.Vendors.VendorAddress1,
735     ' ',
736     AP1.Vendors.VendorAddress2
737 ) AS VendorAddress,
738 AP1.Vendors.VendorCity,
739 AP1.Vendors.VendorState,
740 CONCAT (
741     AP1.Vendors.VendorZipCode,
742     '-0000'
743 ) AS VendorZipCode,
744 CONCAT (
745     '(',
746     LEFT(AP1.Vendors.VendorPhone, 3),
747
748     ') ',
749     SUBSTRING(AP1.Vendors.VendorPhone, 4, 3),
750
751     '-',
752     RIGHT(AP1.Vendors.VendorPhone, 4)
753
754     ') ',
755     LTRIM(RTRIM(
756         CONCAT(AP1.Vendors.VendorContactLName,
757             ', ',
758             AP1.Vendors.VendorContactFName))
759     ) AS VendorContactName,
760
761     AP1.Vendors.DefaultAccountNo,
762     AP1.Invoices.InvoiceID,
763     AP1.Invoices.InvoiceNumber,
764     FORMAT(AP1.Invoices.InvoiceDate,
765         'MM/dd/yyyy', 'en-us')
766
767     AS InvoiceDate,
768     FORMAT(AP1.Invoices.InvoiceTotal,
769         'MM/dd/yyyy', 'en-us')
770
771     AS InvoiceTotal,
772
773     AS InvoiceDate,
774     FORMAT(AP1.Invoices.InvoiceTotal,
775         'MM/dd/yyyy', 'en-us')
776
777     AS InvoiceTotal,

```

```
777 AS InvoiceTotal, -- `InvoiceTotal`
778 (
779     SELECT -- 7. embedded query calling
780     FORMAT(AVG(AP1.Invoices.InvoiceTotal), -- `AVG(InvoiceTotal)`
781     'c', 'en-us') -- formatted as `c`
782     -- (currency) with culture
783     -- `en-us`
784     FROM AP1.Invoices -- from all values in table
785     -- `AP1.Invoices` as
786 ) AS AvgInvoiceTotal, -- `AvgInvoiceTotal`
787 FORMAT(AP1.Invoices.PaymentTotal, -- 8. formatting column as `c`
788 'c', 'en-us') -- (currency) with culture
789 AS PaymentTotal, -- `en-us` as `PaymentTotal`
790 FORMAT(AP1.Invoices.CreditTotal, -- 9. formatting column as `c`
791 'c', 'en-us') -- (currency) with culture
792 AS CreditTotal, -- `en-us` as `CreditTotal`
793 FORMAT(AP1.Invoices.InvoiceDueDate, -- 10. formatting column as
794 'MM/dd/yyyy', 'en-us') -- `MM/dd/yyyy` (date) with
795 -- culture `en-us` as
796 AS InvoiceDueDate, -- `InvoiceDueDate`
797 FORMAT(AP1.Invoices.PaymentDate, -- 11. formatting column as
798 'MM/dd/yyyy', 'en-us') -- `MM/dd/yyyy` (date) with
799 -- culture `en-us` as
800 AS PaymentDate, -- `PaymentDate`
801 AP1.InvoiceLineItems.InvoiceSequence,
802 AP1.InvoiceLineItems.AccountNo,
803 FORMAT(AP1.InvoiceLineItems.InvoiceLineItemAmount,
804 'c', 'en-us') -- 12. formatting column as
805 -- `c` (currency) with
806 -- culture `en-us` as
807 AS InvoiceLineItemAmount, -- `InvoiceLineItemAmount`
808 AP1.InvoiceLineItems.InvoiceLineItemDescription,
809 AP1.Terms.TermsDescription,
810 AP1.Terms.TermsDueDays
811 FROM AP1.InvoiceLineItems -- 13. from
812 -- `AP1.InvoiceLineItems`
813 INNER JOIN AP1.Invoices -- using `INNER JOIN` to
-- connect to
814 -- `AP1.Invoices` to get
815 -- all shared values from
816 --
817 ON AP1.InvoiceLineItems.InvoiceID = AP1.Invoices.InvoiceID
818 -- `AP1.InvoiceLineItems`
819 -- and `AP1.Invoices`
820 INNER JOIN AP1.Terms -- using `INNER JOIN` to
-- connect to `AP1.Terms`
821 -- to get all shared values
822 -- from
823 --
824 ON AP1.Invoices.TermsID = AP1.Terms.TermsID -- (`AP1.InvoiceLineItems`
-- and `AP1.Invoices`) and
825 -- `AP1.Terms` using
826 --
827 RIGHT JOIN AP1.Vendors -- `RIGHT JOIN` to connect
-- to `AP1.Vendors` to get
828
```



```

829 -- values from
830 -- `AP1.Vendors` and
831 -- related data from
832 ON AP1.Invoices.VendorID=AP1.Vendors.VendorID -- (`AP1.InvoiceLineItems`
833 -- and `AP1.Invoices` and
834 -- `AP1.Terms`)
835 ORDER BY -- 14. ordering results by
836 AP1.Vendors.VendorName, -- `VendorName` first and
837 AP1.Invoices.InvoiceID; -- then by `InvoiceID`
838
839
840 /* *****
841 8. To get the difference between two dates, we use `DATEDIFF()`, which
842 ``returns the difference between two date values, based on the interval
843 specified`` (https://techonthenet.com/sql\_server/functions/datediff.php).
844
845 We also call functions `DAY()`
846 (https://techonthenet.com/sql\_server/functions/day.php), `MONTH()`
847 (https://techonthenet.com/sql\_server/functions/month.php) and `YEAR()`
848 (https://techonthenet.com/sql\_server/functions/year.php).
849
850 8.1. In the example below, we use `01/01/2017` as the starting date and
851 `11/22/2021` as the end date.
852 ***** */
853
854 SELECT DATEDIFF(DAY, '01/01/2017', '11/22/2021') AS DatediffDays, -- 1,786 days
855 DATEDIFF(MONTH, '01/01/2017', '11/22/2021') AS DatediffMonths, -- 58 months
856 DATEDIFF(YEAR, '01/01/2017', '11/22/2021') AS DatediffYears; -- 4 years
857
858
859 /* *****
860 8.2. Instead of hard-coding today's date, we can use function `GETDATE()`
861 to retrieve the local system datetime.
862 ***** */
863
864 SELECT DATEDIFF(DAY, '01/01/2017', GETDATE()) AS DatediffDays, -- 1,786 days
865 DATEDIFF(MONTH, '01/01/2017', GETDATE()) AS DatediffMonths, -- 58 months
866 DATEDIFF(YEAR, '01/01/2017', GETDATE()) AS DatediffYears; -- 4 years
867
868
869 /* *****
870 9. LAB #4
871 Write a query without duplicate rows (`SELECT DISTINCT`)
872 9.1. to get all fields from `AP1.Invoices` and `AP1.InvoiceLineItems` to
873 retrieve shared data (`INNER JOIN`) removing all duplicate columns
874 (`AP1.Invoices.InvoiceID` or `AP1.InvoiceLineItems.InvoiceID`),
875 9.2. to format dates as `MMM d, yyyy` (first three letters of the month,
876 the day without leading zeros and the full year)
877 9.3. and to format money (`c`) as `en-us` (`$`).
878 ***** */
879
880 SELECT DISTINCT

```

```

881 AP1.Invoices.InvoiceID,
882 AP1.Invoices.InvoiceNumber,
883 FORMAT(AP1.Invoices.InvoiceDate,           -- 1. formatting column as
884         'MM/dd/yyyy', 'en-us')           -- `MM/dd/yyyy` (date) with
885                                           -- culture `en-us` as
886 AS InvoiceDate,                           -- `InvoiceDate`
887 FORMAT(AP1.Invoices.InvoiceTotal,         -- 2. formatting column as
888         'MM/dd/yyyy', 'en-us')           -- `MM/dd/yyyy` (date) with
889                                           -- culture `en-us` as
890 AS InvoiceTotal,                           -- `InvoiceTotal`
891 (
892     SELECT                                 -- 3. embedded query calling
893         FORMAT(AVG(AP1.Invoices.InvoiceTotal), -- `AVG(InvoiceTotal)`
894               'c', 'en-us')               -- formatted as `c`
895                                           -- (currency) with culture
896                                           -- `en-us`
897     FROM AP1.Invoices                     -- from all values in table
898                                           -- `AP1.Invoices` as
899 ) AS AvgInvoiceTotal,                     -- `AvgInvoiceTotal`
900 FORMAT(AP1.Invoices.PaymentTotal,         -- 4. formatting column as `c`
901         'c', 'en-us')                     -- (currency) with culture
902 AS PaymentTotal,                         -- `en-us` as `PaymentTotal`
903 FORMAT(AP1.Invoices.CreditTotal,         -- 5. formatting column as `c`
904         'c', 'en-us')                     -- (currency) with culture
905 AS CreditTotal,                          -- `en-us` as `CreditTotal`
906 FORMAT(AP1.Invoices.InvoiceDueDate,     -- 6. formatting column as
907         'MM/dd/yyyy', 'en-us')           -- `MM/dd/yyyy` (date) with
908                                           -- culture `en-us` as
909 AS InvoiceDueDate,                        -- `InvoiceDueDate`
910 FORMAT(AP1.Invoices.PaymentDate,        -- 7. formatting column as
911         'MM/dd/yyyy', 'en-us')           -- `MM/dd/yyyy` (date) with
912                                           -- culture `en-us` as
913 AS PaymentDate,                          -- `PaymentDate`
914 AP1.InvoiceLineItems.InvoiceSequence,
915 AP1.InvoiceLineItems.AccountNo,
916 FORMAT(AP1.InvoiceLineItems.InvoiceLineItemAmount,
917         'c', 'en-us')                     -- 8. formatting column as `c`
918                                           -- (currency) with culture
919                                           -- `en-us` as
920 AS InvoiceLineItemAmount,                -- `InvoiceLineItemAmount`
921 AP1.InvoiceLineItems.InvoiceLineItemDescription
922 FROM AP1.Invoices                        -- 9. from `AP1.Invoices` using
923 INNER JOIN AP1.InvoiceLineItems         -- `INNER JOIN` to connect
924                                           -- to `AP1.InvoiceLineItems`
925                                           -- to get all shared values
926 ON AP1.Invoices.InvoiceID = AP1.InvoiceLineItems.InvoiceID
927                                           -- in `AP1.InvoiceLineItems`
928                                           -- and `AP1.Invoices`
929
930 /* *****
931 https://folvera.commons.gc.cuny.edu/?p=1024 (2^10)
932 ***** */

```