

SESSION #10 (2025/12/17): FINAL EXAMINATION

1. Final examination

\*\*\*\*\*

1. Change the tables in schema `AP3` using the following structure.

SCHEMA & TABLE	FIELD	DATA TYPE	NULL
AP3.Employees	EmpID	VARCHAR(11)	NOT NULL
	FName	VARCHAR(100)	NOT NULL
	MName	VARCHAR(100)	
	LName	VARCHAR(100)	NOT NULL
	Gender	VARCHAR(1)	NOT NULL
	Address1	VARCHAR(100)	
	Address2	VARCHAR(100)	
	City	VARCHAR(100)	
	State	VARCHAR(2)	
	PhoneNumber	VARCHAR(10)	
	ZipCode	VARCHAR(10)	
	JobDeptID	INT	
	JobTitleID	INT	
	JobSalary	FLOAT	
StartDate	DATE	NOT NULL	
AP3.JobDepts	JobDeptID	INT	NOT NULL
	JobDept	VARCHAR(100)	NOT NULL
	Comments	VARCHAR(255)	
AP3.JobTitles	JobTitleID	INT	NOT NULL
	JobTitle	VARCHAR(100)	NOT NULL
	JobDept	INT	NOT NULL

2. Update (overwrite) the `AP3.Employees` table.

\*\*\*\*\* \*/

```

45 ALTER TABLE AP3.Employees          -- 01. beginning of table
46     ALTER COLUMN EmpID VARCHAR(11) NOT NULL;  -- `AP3.Employees`
47
48 ALTER TABLE AP3.Employees
49     ALTER COLUMN FName VARCHAR(50) NOT NULL;
50
51 ALTER TABLE AP3.Employees
52     ALTER COLUMN MName VARCHAR(50);
53
54 ALTER TABLE AP3.Employees
55     ALTER COLUMN LName VARCHAR(50) NOT NULL;
56
57 ALTER TABLE AP3.Employees
58     ALTER COLUMN Address1 VARCHAR(100);
59
60 ALTER TABLE AP3.Employees
61     ALTER COLUMN Address2 VARCHAR(100);
62
63 ALTER TABLE AP3.Employees
64     ALTER COLUMN City VARCHAR(50);
65
66 ALTER TABLE AP3.Employees
67     ALTER COLUMN State VARCHAR(2);
68
69 ALTER TABLE AP3.Employees

```

```

70     ALTER COLUMN ZipCode VARCHAR(10);
71
72 ALTER TABLE AP3.Employees
73     ALTER COLUMN PhoneNumber VARCHAR(10);
74
75 ALTER TABLE AP3.Employees
76     ALTER COLUMN JobDeptID INT;
77
78 ALTER TABLE AP3.Employees
79     ALTER COLUMN JobTitleID INT;
80
81 ALTER TABLE AP3.Employees
82     ALTER COLUMN JobSalary FLOAT;
83
84 ALTER TABLE AP3.JobDepts                -- 02. beginning of table
85     ALTER COLUMN JobDeptID INT NOT NULL; --      `AP3.JobDepts`
86
87 ALTER TABLE AP3.JobDepts
88     ALTER COLUMN JobDept VARCHAR(100) NOT NULL;
89
90 ALTER TABLE AP3.JobDepts
91     ALTER COLUMN Comments VARCHAR(255);
92
93 ALTER TABLE AP3.JobTitles              -- 03. beginning of table
94     ALTER COLUMN JobTitleID INT NOT NULL; --      `AP3.JobTitles`
95
96 ALTER TABLE AP3.JobTitles
97     ALTER COLUMN JobTitle VARCHAR(100) NOT NULL;
98
99 ALTER TABLE AP3.JobTitles
100    ALTER COLUMN JobDeptID INT NOT NULL;
101
102
103 /* *****
104    2.1. Remove all empty spaces in fields `FName`, `LName`, `MName`, `City`
105        and `State`.
106
107    2.2. Use proper case in fields `FName`, `LName`, `MName` and `City`.
108
109        2.2.1. Since `MName` is only a character long, there is no need for
110              `LEFT()` or `SUBSTRING()`.
111
112        2.2.2. Since the only value in `City` is `new york` in lower case, we
113              can simply replace the original value for `New York` with the
114              correct case.
115    ***** */
116
117
118 UPDATE AP3.Employees
119 SET FName = CONCAT (
120     UPPER(LEFT(LTRIM(RTRIM(FName)), 1)),
121     LOWER(SUBSTRING(LTRIM(RTRIM(FName)), 2, LEN(LTRIM(RTRIM(FName))) - 1))
122 );
123
124 UPDATE AP3.Employees
125 SET LName = CONCAT (
126     UPPER(LEFT(LTRIM(RTRIM(LName)), 1)),
127     LOWER(SUBSTRING(LTRIM(RTRIM(LName)), 2, LEN(LTRIM(RTRIM(LName))) - 1))
128 );
129
130
131 /* *****
132     The latter can be combined setting values to multiple fields in
133     one `SET` clause.
134    ***** */
135
136 UPDATE AP3.Employees
137 SET FName = CONCAT (
138     UPPER(LEFT(LTRIM(RTRIM(FName)), 1)),

```

```

139     LOWER (SUBSTRING (LTRIM (RTRIM (FName)), 2, LEN (LTRIM (RTRIM (FName))) - 1))
140     ), -- 01. if not done in line #178
141     MName = UPPER (LTRIM (RTRIM (MName))), -- 02. if not done in line #180
142     LName = CONCAT (
143         UPPER (LEFT (LTRIM (RTRIM (LName)), 1)),
144         LOWER (SUBSTRING (LTRIM (RTRIM (LName)), 2, LEN (LTRIM (RTRIM (LName))) - 1))
145     ); -- 03. if not done in line #182
146
147
148 /* *****
149     2.2.3. Note that we cannot make `AP3.Employees.StartDate` `NOT NULL`
150         till we push values into the column if any field is NULL.
151     ***** */
152
153 UPDATE AP3.Employees -- 01. passing values of to
154 SET StartDate = GETDATE() -- columns where
155 WHERE StartDate IS NULL -- `StartDate` is NULL or
156 OR StartDate = ''; -- has no value, in this
157 -- case passing the value
158 -- retrieved from
159 -- `GETDATE()`
160
161 ALTER TABLE AP3.Employees -- 02. making column `NOT NULL`
162 ALTER COLUMN StartDate DATE NOT NULL; -- now that `StartDate` has
163 -- values from `GETDATE()`
164
165
166 /* *****
167     2.3. The updates above (#2.1 & #2.2) to `AP3.Employees` (same table) can
168     also be done in one (1) `UPDATE` statement with only one (1) `SET` and
169     commas separating each column.
170
171     2.3.1. Note that functions are called in the same order (innermost to
172     outermost) as specified in each section (#2.1 & #2.2).
173
174     2.3.2. Since `MName` is only a character long, there is no need for
175     `LEFT()` or `SUBSTRING()`.
176     ***** */
177
178 UPDATE AP3.Employees
179 SET FName = RTRIM (LTRIM (FName)), -- 01. if not done in lines
180 -- #138 to #140
181     MName = UPPER (RTRIM (LTRIM (MName))), -- 02. if not done in lines
182 -- #141
183     LName = RTRIM (LTRIM (LName)), -- 03. if not done in line
184 -- #142 to #144
185     City = RTRIM (LTRIM (City)),
186     State = UPPER (RTRIM (LTRIM (State)));
187
188
189 /* *****
190     2.3.3. Since the only value in `City` is `new york` in lower case, we
191     can simply overwrite the original value for `New York` with the
192     correct case.
193     ***** */
194
195 UPDATE AP3.Employees
196 SET City = 'New York'
197 WHERE City = 'new york';
198
199
200 /* *****
201     2.3. Capitalize States.
202     ***** */
203
204 UPDATE AP3.Employees
205 SET State = UPPER (RTRIM (LTRIM (State)));
206
207

```

```

208 /* *****
209 2.4. Change data type of `JobSalary` from FLOAT to MONEY.
210 ***** */
211
212 ALTER TABLE AP3.Employees
213 ALTER COLUMN JobSalary MONEY;
214
215
216 /* *****
217 3. Add column `LastUpdateDate` to table `AP3.Employees` with data type DATE
218 and NOT NULL with today's DATE.
219
220 3.1. When you UPDATE a table, you cannot assign `NOT NULL`. There is
221 always a way to do things in SQL and other programming. You might
222 have to first create a new table and then populate it with the data
223 from the original table.
224
225 3.2. One alternative is using DATE function `GETDATE()`.
226
227 3.3. After adding column `LastUpdateDate` to table `AP3.Employees`, we pass
228 the value of the result of built-in function `GETDATE()`.
229
230 3.4. `GETDATE()` must be followed by an opening and closing parenthesis
231 (without parameters) to tell the system that `GETDATE()` is a function
232 and not confuse the system into believing that `GETDATE()` is a column
233 in a table).
234 ***** */
235
236 ALTER TABLE AP3.Employees
237 ADD LastUpdateDate DATE;
238
239 UPDATE AP3.Employees
240 SET LastUpdateDate = GETDATE();
241
242
243 /* *****
244 4. Create a view named `AP3.EmployeesVW` with all employee information with
245 the following fields without extra spaces.
246
247 +-----+-----+
248 | FIELD(S) | WHAT TO DISPLAY |
249 +-----+-----+
250 | EmpID | first seven (7) characters masked with |
251 | | `XXX-XX-` |
252 +-----+-----+
253 | LName, FName MName | returned in one field with a single |
254 | | space between them, avoid NULLs if |
255 | | `MName` is NULL; no multiple spaces; |
256 | | trimmed if needed; capitalized with a |
257 | | comma between `LName` and `FName` |
258 | | replacing empty spaces |
259 +-----+-----+
260 | Gender | capitalized |
261 +-----+-----+
262 | Address1 Address2 | returned in one field with a single |
263 | | space between them; trimmed if needed |
264 +-----+-----+
265 | City | first letter capitalized; trimmed if |
266 | | needed |
267 +-----+-----+
268 | State | capitalized; trimmed if needed |
269 +-----+-----+
270 | ZipCode | with missing Plus 4 as `-0001` |
271 +-----+-----+
272 | PhoneNumber | formatted as `(XXX) XXX-XXXX` |
273 +-----+-----+
274 | JobDept | using corresponding value from table |
275 | | `JobDept`, not `JobDeptID` |
276 +-----+-----+

```

277	JobTitle	using corresponding value from table	
278		`JobTitle`, not `JobTitleID`	
279	+-----+-----+-----+		
280	JobSalary	formatted as currency (`c`)	
281	+-----+-----+-----+		
282	StartDate	formatted as date (`d`)	
283	+-----+-----+-----+		
284	LastUpdateDate	new column updated with today's date;	
285		formatted as date (`d`)	
286	+-----+-----+-----+		
287	Comments	no formatting	
288	+-----+-----+-----+		
289			

290 4.1. When creating the view, only show employees hired after 12/30/1999 and  
 291 before 6/1/2015 sorting the output by last name, first name, middle  
 292 name and employee ID.

293  
 294 4.1.1. When using an alias after concatenating fields, you would use  
 295 the alias in `ORDER BY` (no `ORDER BY` in views).

296  
 297 4.1.2. Before creating the view, it is good practice to first create  
 298 the query.

299  
 300 4.1.3. Add `XXX-XX-` to the last four characters in `EmpID`.

301  
 302 4.1.3.1. The latter can be done using a user-defined function.

303  
 304 4.1.4. Concatenate `LName`, `FName` and `MName` with spaces in between  
 305 if we have a value for `MName` or `LName` and `FName` with a  
 306 space in between if we have no value for `MName`.

307  
 308 4.1.4.1. We can use a `CASE` clause to add added logic.

309  
 310 4.1.4.2. The latter can be done using a user-defined function.

311  
 312 4.1.5. Concatenate (put strings together) with space (` `) between  
 313 `Address1` and `Address2` if we have a value in `Address2` or  
 314 only call `Address1` if we have no value in `Address2`.

315  
 316 4.1.5.1. We can use a `CASE` clause for added logic.

317  
 318 4.1.5.2. The latter can be done using a user-defined function.

319  
 320 4.1.6. Once again we have to concatenate strings. In this case, we  
 321 add dummy Plus 4 `-0001` (non-existent string in the database,  
 322 hard-coded) before field `ZipCode`.

323  
 324 4.1.6.1. The latter can be done using a user-defined function.

325  
 326 4.1.7. We have to separate the parts of the phone number logically and  
 327 present the phone number as `(XXX) XXX-XXXX`.

328  
 329 4.1.7.1. We add `(` to surround the area code part of  
 330 `PhoneNumber`.

331  
 332 4.1.7.2. We call the first three (3) characters of  
 333 `PhoneNumber` using `LEFT()` calling three (3) spaces.

334  
 335 4.1.7.3. Then we add `)` to close the first parenthesis and  
 336 complete the way area codes are commonly displayed.

337  
 338 4.1.7.4. Now we have to call the inside of `PhoneNumber` using  
 339 `SUBSTRING()` calling first where we want to start  
 340 (fourth characters) and how many characters from the  
 341 first value we want to call -- next three (3)  
 342 characters.

343  
 344 4.1.7.5. We add a hyphen (`-`) to continue the way how phone  
 345 numbers are normally presented.

346  
 347 4.1.7.6. We finish by calling the last four (4) characters  
 348 using `RIGHT()` and calling four (4) places.  
 349  
 350 4.1.7.7. The latter can be done using a user-defined function.  
 351  
 352 4.1.8. Format all monetary values (`c`) with culture `en-us`  
 353 (<https://msdn.microsoft.com/en-us/library/hh213525.aspx>).  
 354  
 355 4.1.8.1. Note that formatting any numeric value returns a  
 356 string.  
 357  
 358 4.1.8.2. The latter can be done using a user-defined function.  
 359  
 360 4.1.9. Format all dates (`d`) with culture `en-us`  
 361 (<https://msdn.microsoft.com/en-us/library/hh213525.aspx>).  
 362  
 363 4.1.9.1. Note that formatting any numeric value returns a  
 364 string.  
 365  
 366 4.1.9.2. The latter can be done using a user-defined function.  
 367  
 368 4.1.10. Retrieve values for `AP3.Employees.StartDate` greater or equal  
 369 to (`>=`) to `12/30/1999` and less than or equal to (`<=`) to  
 370 `6/1/2015` (range of values).

```
WHERE AP3.Employees.StartDate >= `12/30/1999`
      AND AP3.Employees.StartDate <= `6/1/2015`
```

The better option is using `BETWEEN`, which is cleaner and easier to read.

```
WHERE AP3.Employees.StartDate BETWEEN `12/30/1999`
      AND `6/1/2015`
```

It also avoids the possibility of errors when using operators `>=` and `<=`.

\*\*\*\*\* \*/

```
385 SELECT DISTINCT -- 01. start of query (#4.1)
386 REPLACE(AP3.Employees.EmpID, LEFT(AP3.Employees.EmpID, 7), 'xxx-xx-')
387 AS EmpID,
388 CASE -- 02. concatenating last,
389 WHEN AP3.Employees.MName IS NOT NULL -- first and middle names;
390 OR AP3.Employees.MName <> '' -- if not concatenating
391 THEN CONCAT ( -- only first and last
392 AP3.Employees.LName, -- names
393 ' ',
394 AP3.Employees.FName,
395 ' ',
396 AP3.Employees.MName
397 )
398 ELSE CONCAT (
399 AP3.Employees.LName,
400 ' ',
401 AP3.Employees.FName
402 )
403 END AS Name,
404 AP3.Employees.Gender,
405 CASE
406 WHEN AP3.Employees.Address2 IS NOT NULL -- 03. concatenating `Address1`
407 OR AP3.Employees.Address2 <> '' -- and `Address2` when
408 THEN CONCAT ( -- `Address2` is NOT NULL
409 AP3.Employees.Address1, -- or an empty string; if
410 ' ', -- not showing only the
411 AP3.Employees.Address2 -- original value of
412 ) -- `Address1`
413 ELSE AP3.Employees.Address1
414 END AS Address,
```

```

415 AP3.Employees.City,
416 CASE
417     WHEN AP3.Employees.ZipCode IS NULL
418          OR AP3.Employees.ZipCode = ''
419          OR LEN(AP3.Employees.ZipCode) <> 5
420     THEN ''
421     WHEN LEN(AP3.Employees.ZipCode) = 5
422     THEN CONCAT (
423         AP3.Employees.ZipCode,
424         '-0001'
425     )
426     ELSE AP3.Employees.ZipCode
427     END AS ZipCode,
428 CASE
429     WHEN PhoneNumber <> ''
430          OR PhoneNumber <> 10
431     THEN CONCAT (
432         '(',
433         LEFT(PhoneNumber, 3),
434         ') ',
435         SUBSTRING(PhoneNumber, 4, 3),
436         '-',
437         RIGHT(PhoneNumber, 4)
438     )
439     ELSE PhoneNumber,
440     END AS PhoneNumber,
441
442 AP3.JobDepts.JobDept,
443 AP3.JobTitles.JobTitle,
444 FORMAT(AP3.Employees.JobSalary, 'c', 'en-us') -- 06. formatting dollar amounts
445     AS JobSalary, -- as currency ('c') with
446 -- culture 'en-us'
447 FORMAT(AP3.Employees.StartDate, 'd', 'en-us') -- 07. formatting dates as
448     AS StartDate, -- short date ('d') with
449 -- culture 'en-us'
450 FORMAT(AP3.Employees.LastUpdateDate, -- 08. formatting dates as
451     'd', 'en-us') AS LastUpdateDate, -- short date ('d') with
452 -- culture 'en-us'
453
454 AP3.JobDepts.Comments
455 FROM AP3.Employees
456 LEFT JOIN AP3.JobDepts
457     ON AP3.Employees.JobDeptID = AP3.JobDepts.JobDeptID
458 LEFT JOIN AP3.JobTitles
459     ON AP3.Employees.JobTitleID = AP3.JobTitles.JobTitleID
460 WHERE AP3.Employees.StartDate BETWEEN '12/30/1999'
461     AND '6/1/2015'
462 ORDER BY Name, -- 09. end of query (#4.1)
463     EmpID;
464
465 /* *****
466     4.2. Now we can create view `final.EmployeesVW` in database `labs` using
467         the query above.
468
469         4.2.1. Do not forget to create schema `final` in in database `labs`
470             and to add database `SF25SQL6172025` before schema `AP3` and
471             the name of the tables.
472
473         4.4.2. Note that views in T-SQL cannot be created using `ORDER BY` so
474             we have to remove the clause.
475
476         4.2.3. Therefore we are forced to remove the `ORDER BY` clause from
477             our query.
478     ***** */
479
480 CREATE SCHEMA final; -- 01. creating schema `final`
481 -- in database `labs` where
482 -- we are creating view
483 -- `EmployeesVW`

```

```

484
485 CREATE VIEW final.EmployeesVW -- 02. start of view
486 AS -- `final.EmployeesVW`
487 (
488 SELECT DISTINCT -- 03. start of query (#4.1)
489 REPLACE(SF25SQL6172025.AP3.Employees.EmpID,
490 LEFT(SF25SQL6172025.AP3.Employees.EmpID, 7), 'xxx-xx-') AS EmpID,
491 CASE
492 WHEN SF25SQL6172025.AP3.Employees.MName IS NOT NULL
493 OR SF25SQL6172025.AP3.Employees.MName <> ''
494 THEN CONCAT (
495 SF25SQL6172025.AP3.Employees.LName,
496 ', ',
497 SF25SQL6172025.AP3.Employees.FName,
498 ', ',
499 SF25SQL6172025.AP3.Employees.MName
500 )
501 ELSE CONCAT (
502 SF25SQL6172025.AP3.Employees.LName,
503 ', ',
504 SF25SQL6172025.AP3.Employees.FName
505 )
506 END AS Name,
507 SF25SQL6172025.AP3.Employees.Gender,
508 CASE
509 WHEN SF25SQL6172025.AP3.Employees.Address2 IS NOT NULL
510 OR SF25SQL6172025.AP3.Employees.Address2 <> ''
511 THEN CONCAT (
512 SF25SQL6172025.AP3.Employees.Address1,
513 ', ',
514 SF25SQL6172025.AP3.Employees.Address2
515 )
516 ELSE SF25SQL6172025.AP3.Employees.Address1
517 END AS Address,
518 SF25SQL6172025.AP3.Employees.City,
519 CASE
520 WHEN SF25SQL6172025.AP3.Employees.ZipCode IS NULL
521 OR SF25SQL6172025.AP3.Employees.ZipCode = ''
522 OR LEN(SF25SQL6172025.AP3.Employees.ZipCode) <> 5
523 THEN ''
524 WHEN LEN(SF25SQL6172025.AP3.Employees.ZipCode) = 5
525 THEN CONCAT (
526 SF25SQL6172025.AP3.Employees.ZipCode,
527 '-0001'
528 )
529 ELSE SF25SQL6172025.AP3.Employees.ZipCode
530 END AS ZipCode,
531 CASE
532 WHEN PhoneNumber IS NOT NULL -- 04. checking for NULL
533 OR PhoneNumber <> '' -- 05. checking that it is not
534 -- an empty string
535 OR PhoneNumber <> 10 -- 06. checking for length
536 OR PhoneNumber NOT LIKE ('(%)%-%') -- 07. checking for format
537 THEN CONCAT (
538 '(',
539 LEFT(PhoneNumber, 3),
540 ') ',
541 SUBSTRING(PhoneNumber, 4, 3),
542 '-',
543 RIGHT(PhoneNumber, 4)
544 )
545 ELSE PhoneNumber
546 END AS PhoneNumber,
547 SF25SQL6172025.AP3.JobDepts.JobDept,
548 SF25SQL6172025.AP3.JobTitles.JobTitle,
549 FORMAT(SF25SQL6172025.AP3.Employees.JobSalary,
550 'c', 'en-us') AS JobSalary,
551 FORMAT(SF25SQL6172025.AP3.Employees.StartDate,
552 'd', 'en-us') AS StartDate,

```

```

553     FORMAT (SF25SQL6172025.AP3.Employees.LastUpdateDate,
554             'd', 'en-us') AS LastUpdateDate,
555     SF25SQL6172025.AP3.JobDepts.Comments
556 FROM SF25SQL6172025.AP3.Employees
557 LEFT JOIN SF25SQL6172025.AP3.JobDepts
558     ON SF25SQL6172025.AP3.Employees.JobDeptID =
559     SF25SQL6172025.AP3.JobDepts.JobDeptID
560 LEFT JOIN SF25SQL6172025.AP3.JobTitles
561     ON SF25SQL6172025.AP3.Employees.JobTitleID =
562     SF25SQL6172025.AP3.JobTitles.JobTitleID
563 WHERE SF25SQL6172025.AP3.Employees.StartDate BETWEEN '12/30/1999'
564        AND '6/1/2015'
565 -- ORDER BY Name, -- 08. no `ORDER BY` in views
566 -- EmpID -- 09. end of query (#4.1)
567 ); -- 10. end of view (#4.2)
568
569
570 /* *****
571     As an alternative, we can use an alias (`AS`) for each table to
572     make our code tidier and to avoid repeating the database and
573     schema before each table (`<database>.<schema>.<table>` or
574     `<schema>.<table>`) throughout the query.
575
576     `emp` for `SF25SQL6172025.AP3.Employees`
577     `jd` for `SF25SQL6172025.AP3.JobDepts`
578     `jt` for `SF25SQL6172025.AP3.JobTitles`
579 ***** */
580
581 CREATE VIEW final.EmployeesVW
582 AS
583 (
584     SELECT DISTINCT REPLACE (emp.EmpID, LEFT (emp.EmpID, 7), 'xxx-xx-') AS EmpID,
585     CASE
586         WHEN emp.MName IS NOT NULL
587             OR emp.MName <> ''
588             THEN CONCAT (
589                 emp.LName,
590                 ' ',
591                 emp.FName,
592                 ' ',
593                 emp.MName
594             )
595         ELSE CONCAT (
596             emp.LName,
597             ' ',
598             emp.FName
599         )
600     END AS Name,
601     emp.Gender,
602     CASE
603         WHEN emp.Address2 IS NOT NULL
604             OR emp.Address2 <> ''
605             THEN CONCAT (
606                 emp.Address1,
607                 ' ',
608                 emp.Address2
609             )
610         ELSE emp.Address1
611     END AS Address,
612     emp.City,
613     CASE
614         WHEN emp.ZipCode IS NULL
615             OR emp.ZipCode = ''
616             OR LEN (emp.ZipCode) <> 5
617             THEN ''
618         WHEN LEN (emp.ZipCode) = 5
619             THEN CONCAT (
620                 emp.ZipCode,
621                 '-0001'

```

```

622         )
623     ELSE emp.ZipCode
624     END AS ZipCode,
625     CASE
626     WHEN PhoneNumber IS NOT NULL
627     OR PhoneNumber <> ''
628     OR PhoneNumber <> 10
629     OR PhoneNumber NOT LIKE ('(%)%-')
630     THEN CONCAT (
631         '(',
632         LEFT(PhoneNumber, 3),
633         ') ',
634         SUBSTRING(PhoneNumber, 4, 3),
635         '-',
636         RIGHT(PhoneNumber, 4)
637     )
638     ELSE PhoneNumber
639     END AS PhoneNumber,
640     jd.JobDept,
641     jt.JobTitle,
642     FORMAT(emp.JobSalary, 'c', 'en-us') AS JobSalary,
643     FORMAT(emp.StartDate, 'd', 'en-us') AS StartDate,
644     FORMAT(emp.LastUpdateDate, 'd', 'en-us') AS LastUpdateDate,
645     jd.Comments
646 FROM SF25SQL6172025.AP3.Employees AS emp
647 LEFT JOIN SF25SQL6172025.AP3.JobDeptsAS jd
648     ON emp.JobDeptID = jd.JobDeptID
649 LEFT JOIN SF25SQL6172025.AP3.JobTitles AS jt
650     ON emp.JobTitleID = jt.JobTitleID
651 WHERE emp.StartDate BETWEEN '12/30/1999'
652     AND '6/1/2015'
653 );
654
655
656 /* *****
657 4.3. To make sure that the view displays the correct data, we can call all
658 the values referred in view `final.EmployeesVW`.
659 ***** */
660
661 SELECT *
662 FROM final.EmployeesVW;
663
664
665 /* *****
666 5. As a bonus, you can use functions to format, concatenate and other tasks
667 when writing the query that will be part of your view.
668
669 5.1. First we write the function to add `XXX-XX-` to the last four
670 characters in `EmpID` (#4.1.3).
671 ***** */
672
673 CREATE FUNCTION final.EmpID_udf (@in_empid VARCHAR(15))
674 RETURNS VARCHAR(15) -- 01. returning data type
675 AS
676 BEGIN
677     DECLARE @out_empid VARCHAR(15) = REPLACE(@in_empid,
678         LEFT(@in_empid, 7),
679         'xxx-xx-')
680 -- 02. declaring output and
681 -- passing value of
682 -- `REPLACE` in-line, not
683 -- using `SET`
684     RETURN @out_empid -- 03. returning value of
685 -- output parameter
686 END;
687
688
689 /* *****
690 5.2. Then we write the function to concatenate `LName`, `FName` and `MName`

```

```

691         with spaces in between if we have a value for `MName` or `LName` and
692         `FName` with a space in between if we have no value for `MName`
693         (#4.1.4).
694     /* ***** */
695
696     CREATE FUNCTION final.fullname_udf (
697         @in_fname VARCHAR(50),
698         @in_mname VARCHAR(1),
699         @in_lname VARCHAR(50)
700     )
701
702     RETURNS VARCHAR(101)
703
704     AS
705     BEGIN
706         DECLARE @out_fullname VARCHAR(101) = CASE
707             WHEN @in_mname IS NOT NULL
708                 OR @in_mname <> ''
709             THEN CONCAT (
710                 @in_lname,
711                 ', ',
712                 @in_fname,
713                 ', ',
714                 @in_mname
715             )
716             ELSE CONCAT (
717                 @in_lname,
718                 ', ',
719                 @in_fname
720             )
721             END
722         RETURN @out_fullname
723     END;
724
725
726     /* *****
727     5.3. Then we write the function to Concatenate with space between
728     `Address1` and `Address2` if we have a value in `Address2` or only
729     call `Address1` if we have no value in `Address2` (#4.1.5).
730     ***** */
731
732     CREATE FUNCTION final.address2_udf (
733         @in_address1 VARCHAR(100),
734         @in_address2 VARCHAR(100)
735     )
736
737     RETURNS VARCHAR(201)
738
739     AS
740     BEGIN
741         DECLARE @out_address VARCHAR(201) = CASE
742             WHEN @in_address2 IS NOT NULL
743                 OR @in_address2 <> ''
744             THEN CONCAT (
745                 @in_address1,
746                 ', ',
747                 @in_address2
748             )
749             ELSE @in_address1
750             END
751         RETURN @out_address
752     END;
753
754
755     /* *****
756     5.4. Then we write the function to add dummy Plus 4 `-0001` (non-existent
757     string in the database, hard-coded) before field `ZipCode` (#4.1.6).
758     ***** */
759

```

```

760 CREATE FUNCTION final.zipcode_udf (@in_zipcode VARCHAR(10))
761 RETURNS VARCHAR(10) -- 01. returning data type
762 AS
763 BEGIN
764 DECLARE @out_zipcode VARCHAR(10) = CASE
765 WHEN @in_zipcode IS NULL
766 OR @in_zipcode = ''
767 OR LEN(@in_zipcode) <> 5
768 THEN ''
769 WHEN LEN(@in_zipcode) = 5
770 THEN CONCAT (
771 @in_zipcode,
772 '-0001'
773 )
774 ELSE @in_zipcode
775 END -- 02. closing `CASE` clause
776 RETURN @out_zipcode
777 END; -- 03. closing function
778
779
780 /* *****
781 5.4. Then we write the function to to separate the parts of the phone
782 number logically and present the phone number as `(XXX) XXX-XXXX`
783 (#4.1.7).
784 ***** */
785
786 CREATE FUNCTION final.phones_udf (@in_phone VARCHAR(15))
787 RETURNS VARCHAR(15) -- 01. returning data type
788 AS
789 BEGIN
790 DECLARE @out_phone VARCHAR(15) = CASE
791 WHEN @in_phone IS NOT NULL
792 OR @in_phone <> ''
793 OR @in_phone <> 10
794 OR @in_phone NOT LIKE ('(%)%-%')
795 THEN CONCAT (
796 '(',
797 LEFT(@in_phone, 3),
798 ') ',
799 SUBSTRING(@in_phone, 4, 3),
800 '-',
801 RIGHT(@in_phone, 4)
802 )
803 ELSE @in_phone
804 END -- 02. closing `CASE` clause
805 RETURN @out_phone
806 END; -- 03. closing function
807
808
809 /* *****
810 5.5. Now we can re-write the view.
811 ***** */
812
813 CREATE VIEW final.EmployeesVW
814 AS
815 (
816 SELECT DISTINCT
817 final.EmpID_udf(SF25SQL6172025.AP3.Employees.EmpID) AS EmpID,
818 final.fullname_udf(SF25SQL6172025.AP3.Employees.LName,
819 SF25SQL6172025.AP3.Employees.FName,
820 SF25SQL6172025.AP3.Employees.MName
821 ) AS Name,
822 SF25SQL6172025.AP3.Employees.Gender,
823 final.address2_udf(
824 SF25SQL6172025.AP3.Employees.Address1,
825 SF25SQL6172025.AP3.Employees.Address2
826 ) AS Address,
827 SF25SQL6172025.AP3.Employees.City,
828 final.zipcode_udf(SF25SQL6172025.AP3.Employees.ZipCode) AS ZipCode,

```

```

829     final.phones_udf(PhoneNumber) AS PhoneNumber,
830     SF25SQL6172025.AP3.JobDepts.JobDept,
831     SF25SQL6172025.AP3.JobTitles.JobTitle,
832     FORMAT(SF25SQL6172025.AP3.Employees.JobSalary,
833             'c', 'en-us') AS JobSalary,
834     FORMAT(SF25SQL6172025.AP3.Employees.StartDate,
835             'd', 'en-us') AS StartDate,
836     FORMAT(SF25SQL6172025.AP3.Employees.LastUpdateDate,
837             'd', 'en-us') AS LastUpdateDate,
838     SF25SQL6172025.AP3.JobDepts.Comments
839 FROM SF25SQL6172025.AP3.Employees
840 LEFT JOIN SF25SQL6172025.AP3.JobDepts
841     ON SF25SQL6172025.AP3.Employees.JobDeptID =
842     SF25SQL6172025.AP3.JobDepts.JobDeptID
843 LEFT JOIN SF25SQL6172025.AP3.JobTitles
844     ON SF25SQL6172025.AP3.Employees.JobTitleID =
845     SF25SQL6172025.AP3.JobTitles.JobTitleID
846 WHERE SF25SQL6172025.AP3.Employees.StartDate BETWEEN '12/30/1999'
847        AND '6/1/2015'
848 );
849
850
851 /* *****
852     As an alternative, we can use an alias (`AS`) for each table to make
853     our code tidier and to avoid repeating the database and schema before
854     each table (`<database>.<schema>.<table>` or `<schema>.<table>`)
855     throughout the query.
856
857         `emp` for `SF25SQL6172025.AP3.Employees`
858         `jd`  for `SF25SQL6172025.AP3.JobDepts`
859         `jt`  for `SF25SQL6172025.AP3.JobTitles`
860 ***** */
861
862 CREATE VIEW final.EmployeesVW
863 AS
864 (
865     SELECT DISTINCT final.EmpID_udf(emp.EmpID) AS EmpID,
866     final.fullname_udf(emp.LName, emp.FName, emp.MName) AS Name,
867     emp.Gender,
868     final.address2_udf(emp.Address1, emp.Address2) AS Address,
869     emp.City,
870     final.zipcode_udf(emp.ZipCode) AS ZipCode,
871     final.phones_udf(PhoneNumber) AS PhoneNumber,
872     jp.JobDept,
873     jt.JobTitle,
874     FORMAT(emp.JobSalary, 'c', 'en-us') AS JobSalary,
875     FORMAT(emp.StartDate, 'd', 'en-us') AS StartDate,
876     FORMAT(emp.LastUpdateDate, 'd', 'en-us') AS LastUpdateDate,
877     jp.Comments
878 FROM SF25SQL6172025.AP3.Employees AS emp
879 LEFT JOIN SF25SQL6172025.AP3.JobDeptsAS jd
880     ON emp.JobDeptID = jd.JobDeptID
881 LEFT JOIN SF25SQL6172025.AP3.JobTitles AS jt
882     ON emp.JobTitleID = jt.JobTitleID
883 WHERE emp.StartDate BETWEEN '12/30/1999'
884        AND '6/1/2015'
885 );
886
887
888 /* *****
889     5.6. As a bonus, there are other objects that can be added to tables.
890     Constraints like keys restrict the possibility of losing data by
891     dropping the wrong object or even inserting the wrong data into a
892     field.
893
894     ``A primary key, also called a primary keyword, is a key in a
895     relational database that is unique for each record. It is a unique
896     identifier, such as a driver license number, telephone number
897     (including area code), or vehicle identification number (VIN). A

```

898 relational database must always have one and only one primary key.  
899 Primary keys typically appear as columns in relational database  
900 tables.  
901 The choice of a primary key in a relational database often depends  
902 on the preference of the administrator. It is possible to change  
903 the primary key for a given database when the specific needs of the  
904 users changes. For example, the people in a town might be uniquely  
905 identified according to their driver license numbers in one  
906 application, but in another situation it might be more convenient to  
907 identify them according to their telephone numbers.`  
908 <https://searchsqlserver.techtarget.com/definition/primary-key>  
909

910 The syntax is similar to that of any structure change of a table. In  
911 this case, we add a primary key (PK) constraint.

```
912 ALTER TABLE table_name  
913 ADD CONSTRAINT pk_name  
914 PRIMARY KEY (column_name)
```

915  
916  
917 ``A foreign key is a column or columns of data in one table that  
918 connects to the primary key data in the original table.  
919 To ensure the links between foreign key and primary key tables  
920 aren't broken, foreign key constraints can be created to prevent  
921 actions that would damage the links between tables and prevent  
922 erroneous data from being added to the foreign key column.`  
923 <https://searchoracle.techtarget.com/definition/foreign-key>  
924

925 Just like in the case of primary keys, we need to alter the table to  
926 add a foreign key (FK) constraint. The big difference is that we need  
927 to indicate the primary key (PK), on which the FK depends on.

```
928  
929 ALTER TABLE child_table  
930 ADD CONSTRAINT fk_name  
931 FOREIGN KEY child_table_column  
932 REFERENCES parent_table (parent_column)
```

933  
934 These two constraints are different as noted above. The main  
935 difference is that a table can have only one primary key (PK) making  
936 it the identifier for the row and/or multiple foreign keys (FKs)  
937 referencing other PKs in other tables in order to maintain  
938 dependency to other tables.

939  
940 ``A primary key in the original table, or parent table, can be  
941 targeted by multiple foreign keys from other `child` tables. But a  
942 primary key does not necessarily have to be the target of any  
943 foreign keys. A primary key is a column or a set of columns that  
944 identify a row in a table. A foreign key, however, is in a table  
945 that is different from the table that the primary key must match.`  
946 <https://searchoracle.techtarget.com/definition/foreign-key>  
947

948 In the example below, we make a PK from `AP1.Vendors.VendorID` and a  
949 FK from `AP1.ContactUpdates.ContactUpdateID`.

950 \*\*\*\*\* \*/

```
951  
952 ALTER TABLE AP1.Vendors -- 01. must make column  
953 ALTER COLUMN VendorID INT NOT NULL; -- `VendorID` NOT NULL  
954 -- before making it a key  
955  
956 ALTER TABLE AP1.Vendors -- 02. altering table by adding  
957 ADD CONSTRAINT VendorID_PK -- constraint PRIMARY KEY  
958 PRIMARY KEY (VendorID); -- (PK) with `VendorID` in  
959 -- parenthesis  
960  
961  
962 ALTER TABLE AP1.ContactUpdates -- 03. must make column  
963 ALTER COLUMN ContactUpdateID INT NOT NULL; -- `ContactUpdateID` NOT  
964 -- NULL before making it a  
965 -- key  
966
```

```

967 ALTER TABLE AP1.ContactUpdates -- 04. altering table by adding
968 ADD CONSTRAINT ContactUpdateID_FK -- constraint FOREIGN KEY
969 FOREIGN KEY (VendorID) -- (FK) with `VendorID` in
970 REFERENCES AP1.Vendors (VendorID); -- parenthesis indicating
971 -- the reference to the PK
972 -- in parent table
973
974
975 /* *****
976 7. What do you do now that the `Introduction to SQL` course has ended?
977
978 7.1. Go to Microsoft Virtual Academy (https://mva.microsoft.com/) and
979 continue learning including application to write reports
980 (https://docs.microsoft.com/en-us/sql/reporting-services/), visualize
981 and/or analyze data (https://powerbi.microsoft.com/).
982
983 7.2. Register for the `Intermediate to SQL` course
984 (https://www.campusce.net/bmcc/course/course.aspx?catId=328).
985
986 7.3. If you are interested in a career in data science, learn Python
987 (https://python.org/) and the multiple libraries available for data
988 analysis (https://folvera.commons.gc.cuny.edu/?s=python).
989
990 7.4. Contact me if you ever need help.
991
992 8. Before you leave, figure out what the following prints.
993
994 8.1. Change the value `your_name` with your name before running the code
995 below to create the procedure. Then execute it.
996
997 8.2. If curious, visit http://ascii.cl/ for more information on ASCII
998
999 (http://whatis.techtarget.com/definition/ASCII-American-Standard-Code-for-Information-Interchange).
1000 ***** */
1001 CREATE PROCEDURE final.message_sp
1002 AS
1003 BEGIN
1004 DECLARE @yourName VARCHAR(50)='your_name', -- 01. param for your name
1005 @courseCd VARCHAR(15)='SF25SQL6172025'; -- 02. code for the course
1006 PRINT CONCAT (CHAR(084),CHAR(104),CHAR(097),CHAR(110),CHAR(107),CHAR(032),
1007 CHAR(121),CHAR(111),CHAR(117),CHAR(044),CHAR(032),@yourName,CHAR(044),
1008 CHAR(032),CHAR(102),CHAR(111),CHAR(114),CHAR(032),CHAR(116),CHAR(097),
1009 CHAR(107),CHAR(105),CHAR(110),CHAR(103),CHAR(032),CHAR(099),CHAR(108),
1010 CHAR(097),CHAR(115),CHAR(115),CHAR(032),@courseCd,CHAR(046),CHAR(013),
1011 CHAR(083),CHAR(101),CHAR(101),CHAR(032),CHAR(121),CHAR(111),CHAR(117),
1012 CHAR(032),CHAR(105),CHAR(110),CHAR(032),CHAR(116),CHAR(104),CHAR(101),
1013 CHAR(032),CHAR(105),CHAR(110),CHAR(116),CHAR(101),CHAR(114),CHAR(109),
1014 CHAR(101),CHAR(100),CHAR(105),CHAR(097),CHAR(116),CHAR(101),CHAR(032),
1015 CHAR(099),CHAR(108),CHAR(097),CHAR(115),CHAR(115),CHAR(046),CHAR(013),
1016 CHAR(013),CHAR(070),CHAR(046),CHAR(079),CHAR(108),CHAR(118),CHAR(101),
1017 CHAR(114),CHAR(097),CHAR(032),CHAR(040),CHAR(102),CHAR(111),CHAR(108),
1018 CHAR(118),CHAR(101),CHAR(114),CHAR(097),CHAR(064),CHAR(098),CHAR(109),
1019 CHAR(099),CHAR(099),CHAR(046),CHAR(099),CHAR(117),CHAR(110),CHAR(121),
1020 CHAR(046),CHAR(101),CHAR(100),CHAR(117),CHAR(041),CHAR(013),CHAR(104),
1021 CHAR(116),CHAR(116),CHAR(112),CHAR(058),CHAR(047),CHAR(047),CHAR(102),
1022 CHAR(111),CHAR(108),CHAR(118),CHAR(101),CHAR(114),CHAR(097),CHAR(046),
1023 CHAR(099),CHAR(111),CHAR(109),CHAR(109),CHAR(111),CHAR(110),CHAR(115),
1024 CHAR(046),CHAR(103),CHAR(099),CHAR(046),CHAR(099),CHAR(117),CHAR(110),
1025 CHAR(121),CHAR(046),CHAR(101),CHAR(100),CHAR(117),CHAR(047));
1026 END;
1027
1028 EXEC final.message_sp; -- 03. executing final message
1029
1030 /* *****
1031 https://folvera.commons.gc.cuny.edu/?p=1429
1032 ***** */

```