```
/* ************************************************************************
        INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
                  WS24SQL10001, 2024/03/11 - 2024/04/10
               https://folvera.commons.gc.cuny.edu/?cat=34
 ************************************************************************

  SESSION #10 (2024/04/10): 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.
 ************************************************************************ */

ALTER TABLE AP3.Employees                       -- 01. beginning of table
  ALTER COLUMN EmpID VARCHAR(11) NOT NULL;      --     `AP3.Employees`

ALTER TABLE AP3.Employees
  ALTER COLUMN FName VARCHAR(50) NOT NULL;

ALTER TABLE AP3.Employees
  ALTER COLUMN MName VARCHAR(50);

ALTER TABLE AP3.Employees
  ALTER COLUMN LName VARCHAR(50) NOT NULL;

ALTER TABLE AP3.Employees
  ALTER COLUMN Address1 VARCHAR(100);

ALTER TABLE AP3.Employees
  ALTER COLUMN Address2 VARCHAR(100);

ALTER TABLE AP3.Employees
  ALTER COLUMN City VARCHAR(50);

ALTER TABLE AP3.Employees
  ALTER COLUMN State VARCHAR(2);

ALTER TABLE AP3.Employees
```

```sql
    ALTER COLUMN ZipCode VARCHAR(10);

ALTER TABLE AP3.Employees
    ALTER COLUMN PhoneNumber VARCHAR(10);

ALTER TABLE AP3.Employees
    ALTER COLUMN JobDeptID INT;

ALTER TABLE AP3.Employees
    ALTER COLUMN JobTitleID INT;

ALTER TABLE AP3.Employees
    ALTER COLUMN JobSalary FLOAT;

ALTER TABLE AP3.JobDepts                    -- 02. beginning of table
    ALTER COLUMN JobDeptID INT NOT NULL;    --     `AP3.JobDepts`

ALTER TABLE AP3.JobDepts
    ALTER COLUMN JobDept VARCHAR(100) NOT NULL;

ALTER TABLE AP3.JobDepts
    ALTER COLUMN Comments VARCHAR(255);

ALTER TABLE AP3.JobTitles                   -- 03. beginning of table
    ALTER COLUMN JobTitleID INT NOT NULL;   --     `AP3.JobTitles`

ALTER TABLE AP3.JobTitles
    ALTER COLUMN JobTitle VARCHAR(100) NOT NULL;

ALTER TABLE AP3.JobTitles
    ALTER COLUMN JobDeptID INT NOT NULL;


/* ****************************************************************************
    2.1. Remove all empty spaces in fields `FName`, `LName`, `MName`, `City`
        and `State`.

    2.2. Use proper case in fields `FName`, `LName`, `MName` and `City`.

        2.2.1. Since `MName` is only a character long, there is no need for
            `LEFT()` or `SUBSTRING()`.

        2.2.2. Since the only value in `City` is `new york` in lower case, we
            can simply replace the original value for `New York` with the
            correct case.
 **************************************************************************** */


UPDATE AP3.Employees
SET FName = CONCAT (
    UPPER(LEFT(LTRIM(RTRIM(FName)), 1)),
    LOWER(SUBSTRING(LTRIM(RTRIM(FName)), 2, LEN(LTRIM(RTRIM(FName))) - 1))
    );

UPDATE AP3.Employees
SET LName = CONCAT (
    UPPER(LEFT(LTRIM(RTRIM(LName)), 1)),
    LOWER(SUBSTRING(LTRIM(RTRIM(LName)), 2, LEN(LTRIM(RTRIM(LName))) - 1))
    );


/* ****************************************************************************
            The latter can be combined setting values to multiple fields in
            one `SET` clause.
 **************************************************************************** */

UPDATE AP3.Employees
SET FName = CONCAT (
    UPPER(LEFT(LTRIM(RTRIM(FName)), 1)),
```

```
    LOWER(SUBSTRING(LTRIM(RTRIM(FName)), 2, LEN(LTRIM(RTRIM(FName))) - 1))
    ),                                                  -- 01. if not done in line #178
  MName = UPPER(LTRIM(RTRIM(MName))),                   -- 02. if not done in line #180
  LName = CONCAT (
    UPPER(LEFT(LTRIM(RTRIM(LName)), 1)),
    LOWER(SUBSTRING(LTRIM(RTRIM(LName)), 2, LEN(LTRIM(RTRIM(LName))) - 1))
    );                                                  -- 03. if not done in line #182


/* ****************************************************************************
        2.2.3. Note that we cannot make `AP3.Employees.StartDate` `NOT NULL`
               till we push values into the column if any field is NULL.
 **************************************************************************** */

UPDATE AP3.Employees                                    -- 01. passing values of to
SET StartDate = GETDATE()                               --     columns where
WHERE StartDate IS NULL                                 --     `StartDate` is NULL or
  OR StartDate = '';                                    --     has no value, in this
                                                        --     case passing the value
                                                        --     retrieved from
                                                        --     `GETDATE()`

ALTER TABLE AP3.Employees                               -- 02. making column `NOT NULL`
ALTER COLUMN StartDate DATE NOT NULL;                   --     now that `StartDate` has
                                                        --     values from `GETDATE()`


/* ****************************************************************************
    2.3. The updates above (#2.1 & #2.2) to `AP3.Employees` (same table) can
         also be done in one (1) `UPDATE` statement with only one (1) `SET` and
         commas separating each column.

         2.3.1. Note that functions are called in the same order (innermost to
                outermost) as specified in each section (#2.1 & #2.2).

         2.3.2. Since `MName` is only a character long, there is no need for
                `LEFT()` or `SUBSTRING()`.
**************************************************************************** */

UPDATE AP3.Employees
SET FName = RTRIM(LTRIM(FName)),                        -- 01. if not done in lines
                                                        --     #138 to #140
  MName = UPPER(RTRIM(LTRIM(MName))),                   -- 02. if not done in lines
                                                        --     #141
  LName = RTRIM(LTRIM(LName)),                          -- 03. if not done in line
                                                        --     #142 to #144
  City = RTRIM(LTRIM(City)),
  State = UPPER(RTRIM(LTRIM(State)));


/* ****************************************************************************
        2.3.3. Since the only value in `City` is `new york` in lower case, we
               can simply overwrite the original value for `New York` with the
               correct case.
 **************************************************************************** */

UPDATE AP3.Employees
SET City = 'New York'
WHERE City = 'new york';


/* ****************************************************************************
    2.3. Capitalize States.
 **************************************************************************** */

UPDATE AP3.Employees
SET State = UPPER(RTRIM(LTRIM(State)));
```

```
/* ************************************************************************
    2.4. Change data type of `JobSalary` from FLOAT to MONEY.
 ************************************************************************ */

ALTER TABLE AP3.Employees
  ALTER COLUMN JobSalary MONEY;


/* ************************************************************************
 3. Add column `LastUpdateDate` to table `AP3.Employees` with data type DATE
    and NOT NULL with today's DATE.

    3.1. When you UPDATE a table, you cannot assign `NOT NULL`.  There is
         always a way to do things in SQL and other programming.  You might
         have to first create a new table and then populate it with the data
         from the original table.

    3.2. One alternative is using DATE function `GETDATE()`.

    3.3. After adding column `LastUpdateDate` to table `AP3.Employees`, we pass
         the value of the result of built-in function `GETDATE()`.

    3.4. `GETDATE()` must be followed by an opening and closing parenthesis
         (without parameters) to tell the system that `GETDATE()` is a function
         and not confuse the system into believing that `GETDATE()` is a column
         in a table).
 ************************************************************************ */

ALTER TABLE AP3.Employees
  ADD LastUpdateDate DATE;

UPDATE AP3.Employees
SET LastUpdateDate = GETDATE();


/* ************************************************************************
 4. Create a view named `AP3.EmployeesVW` with all employee information with
    the following fields without extra spaces.

            +----------------------+-----------------------------------------+
            | FIELD(S)             | WHAT TO DISPLAY                         |
            +----------------------+-----------------------------------------+
            | EmpID                | first seven (7) characters masked with  |
            |                      | `XXX-XX-`                               |
            +----------------------+-----------------------------------------+
            | LName, FName MName   | returned in one field with a single     |
            |                      | space between them, avoid NULLs if      |
            |                      | `MName` is NULL;  no multiple spaces;   |
            |                      | trimmed if needed;  capitalized with a  |
            |                      | comma between `LName` and `FName`       |
            |                      | replacing empty spaces                  |
            +----------------------+-----------------------------------------+
            | Gender               | capitalized                             |
            +----------------------+-----------------------------------------+
            | Address1 Address2    | returned in one field with a single     |
            |                      | space between them;  trimmed if needed  |
            +----------------------+-----------------------------------------+
            | City                 | first letter capitalized;  trimmed if   |
            |                      | needed                                  |
            +----------------------+-----------------------------------------+
            | State                | capitalized; trimmed if needed          |
            +----------------------+-----------------------------------------+
            | ZipCode              | with missing Plus 4 as `-0001`          |
            +----------------------+-----------------------------------------+
            | PhoneNumber          | formatted as `(XXX) XXX-XXXX`           |
            +----------------------+-----------------------------------------+
            | JobDept              | using corresponding value from table    |
            |                      | `JobDept`, not `JobDeptID`              |
            +----------------------+-----------------------------------------+
```

JobTitle	using corresponding value from table `JobTitle`, not `JobTitleID`
JobSalary	formatted as currency (`c`)
StartDate	formatted as date (`d`)
LastUpdateDate	new column updated with today's date; formatted as date (`d`)
Comments	no formatting

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
                4.1.7.6. We finish by calling the last four (4) characters
                         using `RIGHT()` and calling four (4) places.

                4.1.7.7. The latter can be done using a user-defined function.

          4.1.8. Format all monetary values (`c`) with culture `en-us`
                 (https://msdn.microsoft.com/en-us/library/hh213525.aspx).

                4.1.8.1. Note that formatting any numeric value returns a
                         string.

                4.1.8.2. The latter can be done using a user-defined function.

          4.1.9. Format all dates (`d`) with culture `en-us`
                 (https://msdn.microsoft.com/en-us/library/hh213525.aspx).

                4.1.9.1. Note that formatting any numeric value returns a
                         string.

                4.1.9.2. The latter can be done using a user-defined function.

          4.1.10. Retrieve values for `AP3.Employees.StartDate` greater or equal
                  to (`>=`) to `12/30/1999` and less than or equal to (`<=`) to
                  `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 `<=`.
 ************************************************************************** */

SELECT DISTINCT                                     -- 01. start of query (#4.1)
  REPLACE(AP3.Employees.EmpID, LEFT(AP3.Employees.EmpID, 7), 'xxx-xx-')
    AS EmpID,
  CASE                                              -- 02. concatenating last,
    WHEN AP3.Employees.MName IS NOT NULL            --     first and middle names;
      OR AP3.Employees.MName <> ''                  --     if not concatenating
      THEN CONCAT (                                 --     only first and last
          AP3.Employees.LName,                      --     names
          ', ',
          AP3.Employees.FName,
          ' ',
          AP3.Employees.MName
          )
    ELSE CONCAT (
        AP3.Employees.LName,
        ', ',
        AP3.Employees.FName
        )
    END AS Name,
  AP3.Employees.Gender,
  CASE
    WHEN AP3.Employees.Address2 IS NOT NULL         -- 03. concatenating `Address1`
      OR AP3.Employees.Address2 <> ''               --     and `Address2` when
      THEN CONCAT (                                 --     `Address2` is NOT NULL
          AP3.Employees.Address1,                   --     or an empty string;  if
          ' ',                                      --     not showing only the
          AP3.Employees.Address2                    --     original value of
          )                                         --     `Address1`
    ELSE AP3.Employees.Address1
    END AS Address,
```

```
    AP3.Employees.City,
    CASE                                                -- 04. concatenating `ZipCode`
      WHEN AP3.Employees.ZipCode IS NULL                --     and dummy Plus 4 `-0001`
        OR AP3.Employees.ZipCode = ''                   --     is NOT NULL, an empty
        OR LEN(AP3.Employees.ZipCode) <> 5              --     string or longer than 5;
        THEN ''                                         --     otherwise showing only
      WHEN LEN(AP3.Employees.ZipCode) = 5               --     the original value of
        THEN CONCAT (                                   --     `ZipCode`
             AP3.Employees.ZipCode,
             '-0001'
             )
      ELSE AP3.Employees.ZipCode
      END AS ZipCode,
    CASE                                                -- 05. concatenating an opening
      WHEN PhoneNumber <> ''                            --     parenthesis (`(`), three
        OR PhoneNumber <> 10                            --     characters from the left
        THEN CONCAT (                                   --     of `PhoneNumber`, the
             '(',                                       --     SUBSTRING starting from
             LEFT(PhoneNumber, 3),                      --     the 4th character taking
             ') ',                                      --     3 characters (4th to
             SUBSTRING(PhoneNumber, 4, 3),              --     6th), a closing
             '-',                                       --     parenthesis and space
             RIGHT(PhoneNumber, 4)                      --     (`( `), a hyphen and 4
             )                                          --     characters from the
      ELSE PhoneNumber,                                 --     right;  if not showing
      END AS PhoneNumber,                               --     only the original value
                                                        --     of `PhoneNumber`
    AP3.JobDepts.JobDept,
    AP3.JobTitles.JobTitle,
    FORMAT(AP3.Employees.JobSalary, 'c', 'en-us')       -- 06. formatting dollar amounts
      AS JobSalary,                                     --     as currency (`c`) with
                                                        --     culture `en-us`
    FORMAT(AP3.Employees.StartDate, 'd', 'en-us')       -- 07. formatting dates as
      AS StartDate,                                     --     short date (`d`) with
                                                        --     culture `en-us`
    FORMAT(AP3.Employees.LastUpdateDate,                -- 08. formatting dates as
      'd', 'en-us') AS LastUpdateDate,                  --     short date (`d`) with
                                                        --     culture `en-us`
    AP3.JobDepts.Comments
  FROM AP3.Employees
  LEFT JOIN AP3.JobDepts
    ON AP3.Employees.JobDeptID = AP3.JobDepts.JobDeptID
  LEFT JOIN AP3.JobTitles
    ON AP3.Employees.JobTitleID = AP3.JobTitles.JobTitleID
  WHERE AP3.Employees.StartDate BETWEEN '12/30/1999'
      AND '6/1/2015'
  ORDER BY Name,
    EmpID;                                              -- 09. end of query (#4.1)


  /* ***************************************************************************
      4.2. Now we can create view `final.EmployeesVW` in database `labs` using
           the query above.

           4.2.1. Do not forget to create schema `final` in in database `labs`
                  and to add database `WS24SQL10001` before schema `AP3` and the
                  name of the tables.

           4.4.2. Note that views in T-SQL cannot be created using `ORDER BY` so
                  we have to remove the clause.

           4.2.3. Therefore we are forced to remove the `ORDER BY` clause from
                  our query.
   *************************************************************************** */

  CREATE SCHEMA final;                                  -- 01. creating schema `final`
                                                        --     in database `labs` where
                                                        --     we are creating view
                                                        --     `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(WS24SQL10001.AP3.Employees.EmpID,
490         LEFT(WS24SQL10001.AP3.Employees.EmpID, 7), 'xxx-xx-') AS EmpID,
491       CASE
492         WHEN WS24SQL10001.AP3.Employees.MName IS NOT NULL
493           OR WS24SQL10001.AP3.Employees.MName <> ''
494           THEN CONCAT (
495                WS24SQL10001.AP3.Employees.LName,
496               ', ',
497                WS24SQL10001.AP3.Employees.FName,
498               ' ',
499                WS24SQL10001.AP3.Employees.MName
500               )
501         ELSE CONCAT (
502              WS24SQL10001.AP3.Employees.LName,
503             ', ',
504              WS24SQL10001.AP3.Employees.FName
505             )
506         END AS Name,
507       WS24SQL10001.AP3.Employees.Gender,
508       CASE
509         WHEN WS24SQL10001.AP3.Employees.Address2 IS NOT NULL
510           OR WS24SQL10001.AP3.Employees.Address2 <> ''
511           THEN CONCAT (
512                WS24SQL10001.AP3.Employees.Address1,
513               ' ',
514                WS24SQL10001.AP3.Employees.Address2
515               )
516         ELSE WS24SQL10001.AP3.Employees.Address1
517         END AS Address,
518       WS24SQL10001.AP3.Employees.City,
519       CASE
520         WHEN WS24SQL10001.AP3.Employees.ZipCode IS NULL
521           OR WS24SQL10001.AP3.Employees.ZipCode = ''
522           OR LEN(WS24SQL10001.AP3.Employees.ZipCode) <> 5
523           THEN ''
524         WHEN LEN(WS24SQL10001.AP3.Employees.ZipCode) = 5
525           THEN CONCAT (
526                WS24SQL10001.AP3.Employees.ZipCode,
527               '-0001'
528               )
529         ELSE WS24SQL10001.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       WS24SQL10001.AP3.JobDepts.JobDept,
548       WS24SQL10001.AP3.JobTitles.JobTitle,
549       FORMAT(WS24SQL10001.AP3.Employees.JobSalary,
550         'c', 'en-us') AS JobSalary,
551       FORMAT(WS24SQL10001.AP3.Employees.StartDate,
552         'd', 'en-us') AS StartDate,
```

```
      FORMAT(WS24SQL10001.AP3.Employees.LastUpdateDate,
        'd', 'en-us') AS LastUpdateDate,
      WS24SQL10001.AP3.JobDepts.Comments
    FROM WS24SQL10001.AP3.Employees
    LEFT JOIN WS24SQL10001.AP3.JobDepts
      ON WS24SQL10001.AP3.Employees.JobDeptID =
        WS24SQL10001.AP3.JobDepts.JobDeptID
    LEFT JOIN WS24SQL10001.AP3.JobTitles
      ON WS24SQL10001.AP3.Employees.JobTitleID =
        WS24SQL10001.AP3.JobTitles.JobTitleID
    WHERE WS24SQL10001.AP3.Employees.StartDate BETWEEN '12/30/1999'
        AND '6/1/2015'
    -- ORDER BY Name,                          -- 08. no `ORDER BY` in views
      -- EmpID                                 -- 09. end of query (#4.1)
    );                                         -- 10. end of view (#4.2)


/* ***************************************************************************
                As an alternative, we can use an alias (`AS`) for each table to
                make our code tidier and to avoid repeating the database and
                schema before each table (`<database>.<schema>.<table>` or
                `<schema>.<table>`) throughout the query.

                    `emp` for `WS24SQL10001.AP3.Employees`
                    `jd`  for `WS24SQL10001.AP3.JobDepts`
                    `jt`  for `WS24SQL10001.AP3.JobTitles`
 *************************************************************************** */

CREATE VIEW final.EmployeesVW
AS
(
    SELECT DISTINCT REPLACE(emp.EmpID, LEFT(emp.EmpID, 7), 'xxx-xx-') AS EmpID,
      CASE
        WHEN emp.MName IS NOT NULL
          OR emp.MName <> ''
          THEN CONCAT (
              emp.LName,
              ', ',
              emp.FName,
              ' ',
              emp.MName
              )
        ELSE CONCAT (
            emp.LName,
            ', ',
            emp.FName
            )
        END AS Name,
      emp.Gender,
      CASE
        WHEN emp.Address2 IS NOT NULL
          OR emp.Address2 <> ''
          THEN CONCAT (
              emp.Address1,
              ' ',
              emp.Address2
              )
        ELSE emp.Address1
        END AS Address,
      emp.City,
      CASE
        WHEN emp.ZipCode IS NULL
          OR emp.ZipCode = ''
          OR LEN(emp.ZipCode) <> 5
          THEN ''
        WHEN LEN(emp.ZipCode) = 5
          THEN CONCAT (
              emp.ZipCode,
              '-0001'
```

```
            )
        ELSE emp.ZipCode
        END AS ZipCode,
      CASE
        WHEN PhoneNumber IS NOT NULL
          OR PhoneNumber <> ''
          OR PhoneNumber <> 10
          OR PhoneNumber NOT LIKE ('(%)%-%')
          THEN CONCAT (
            '(',
            LEFT(PhoneNumber, 3),
            ') ',
            SUBSTRING(PhoneNumber, 4, 3),
            '-',
            RIGHT(PhoneNumber, 4)
            )
        ELSE PhoneNumber
        END AS PhoneNumber,
      jd.JobDept,
      jt.JobTitle,
      FORMAT(emp.JobSalary, 'c', 'en-us') AS JobSalary,
      FORMAT(emp.StartDate, 'd', 'en-us') AS StartDate,
      FORMAT(emp.LastUpdateDate, 'd', 'en-us') AS LastUpdateDate,
      jd.Comments
    FROM WS24SQL10001.AP3.Employees AS emp
    LEFT JOIN WS24SQL10001.AP3.JobDeptsAS jd
      ON emp.JobDeptID = jd.JobDeptID
    LEFT JOIN WS24SQL10001.AP3.JobTitles AS jt
      ON emp.JobTitleID = jt.JobTitleID
    WHERE emp.StartDate BETWEEN '12/30/1999'
        AND '6/1/2015'
    );


/* ***************************************************************************
    4.3. To make sure that the view displays the correct data, we can call all
         the values referred in view `final.EmployeesVW`.
 *************************************************************************** */

SELECT *
FROM final.EmployeesVW;


/* ***************************************************************************
 5. As a bonus, you can use functions to format, concatenate and other tasks
    when writing the query that will be part of your view.

    5.1. First we write the function to add `XXX-XX-` to the last four
         characters in `EmpID` (#4.1.3).
 *************************************************************************** */

CREATE FUNCTION final.EmpID_udf (@in_empid VARCHAR(15))
RETURNS VARCHAR(15)                                  -- 01. returning data type
AS
  BEGIN
    DECLARE @out_empid VARCHAR(15) = REPLACE(@in_empid,
                                             LEFT(@in_empid, 7),
                                             'xxx-xx-')
                                                     -- 02. declaring output and
                                                     --     passing value of
                                                     --     `REPLACE` in-line, not
                                                     --     using `SET`
    RETURN @out_empid                                -- 03. returning value of
                                                     --     output parameter
  END;


/* ***************************************************************************
    5.2. Then we write the function to concatenate `LName`, `FName` and `MName`
```

```
        with spaces in between if we have a value for `MName` or `LName` and
        `FName` with a space in between if we have no value for `MName`
        (#4.1.4).
 ************************************************************************** */

CREATE FUNCTION final.fullname_udf (
  @in_fname VARCHAR(50),
  @in_mname VARCHAR(1),
  @in_lname VARCHAR(50)
  )                                                 -- 01. accepting three (3)
                                                    --     values
RETURNS VARCHAR(101)                                -- 02. returning same data type
                                                    --     as output parameter
AS
  BEGIN
    DECLARE @out_fullname VARCHAR(101) = CASE
        WHEN @in_mname IS NOT NULL
          OR @in_mname <> ''
          THEN CONCAT (
              @in_lname,
              ', ',
              @in_fname,
              ' ',
              @in_mname
              )
        ELSE CONCAT (
            @in_lname,
            ', ',
            @in_fname
            )
        END
    RETURN @out_fullname
  END;


/* **************************************************************************
    5.3. Then we write the function to Concatenate with space between
        `Address1` and `Address2` if we have a value in `Address2` or only
        call `Address1` if we have no value in `Address2` (#4.1.5).
 ************************************************************************** */

CREATE FUNCTION final.address2_udf (
  @in_address1 VARCHAR(100),
  @in_address2 VARCHAR(100)
  )                                                 -- 01. accepting three (3)
                                                    --     values
RETURNS VARCHAR(201)                                -- 02. returning same data type
                                                    --     as output parameter
AS
  BEGIN
    DECLARE @out_address VARCHAR(201) = CASE
        WHEN @in_address2 IS NOT NULL
          OR @in_address2 <> ''
          THEN CONCAT (
              @in_address1,
              ' ',
              @in_address2
              )
        ELSE @in_address1
        END                                         -- 03. closing `CASE` clause
    RETURN @out_address
  END;                                              -- 04. closing function


/* **************************************************************************
    5.4. Then we write the function to add dummy Plus 4 `-0001` (non-existent
        string in the database, hard-coded) before field `ZipCode` (#4.1.6).
 ************************************************************************** */

```

```
CREATE FUNCTION final.zipcode_udf (@in_zipcode VARCHAR(10))
RETURNS VARCHAR(10)                                    -- 01. returning data type
AS
  BEGIN
    DECLARE @out_zipcode VARCHAR(10) = CASE
        WHEN @in_zipcode IS NULL
          OR @in_zipcode = ''
          OR LEN(@in_zipcode) <> 5
          THEN ''
        WHEN LEN(@in_zipcode) = 5
          THEN CONCAT (
              @in_zipcode,
              '-0001'
              )
        ELSE @in_zipcode
        END                                            -- 02. closing `CASE` clause
    RETURN @out_zipcode
  END;                                                 -- 03. closing function


/* ***************************************************************************
    5.4. Then we write the function to to separate the parts of the phone
         number logically and present the phone number as `(XXX) XXX-XXXX`
         (#4.1.7).
 *************************************************************************** */

CREATE FUNCTION final.phones_udf (@in_phone VARCHAR(15))
RETURNS VARCHAR(15)                                    -- 01. returning data type
AS
  BEGIN
    DECLARE @out_phone VARCHAR(15) = CASE
        WHEN @in_phone IS NOT NULL
          OR @in_phone <> ''
          OR @in_phone <> 10
          OR @in_phone NOT LIKE ('(%)%-%')
          THEN CONCAT (
              '(',
              LEFT(@in_phone, 3),
              ') ',
              SUBSTRING(@in_phone, 4, 3),
              '-',
              RIGHT(@in_phone, 4)
              )
        ELSE @in_phone
        END                                            -- 02. closing `CASE` clause
    RETURN @out_phone
  END;                                                 -- 03. closing function


/* ***************************************************************************
    5.5. Now we can re-write the view.
 *************************************************************************** */

CREATE VIEW final.EmployeesVW
AS
(
    SELECT DISTINCT
      final.EmpID_udf(WS24SQL10001.AP3.Employees.EmpID) AS EmpID,
      final.fullname_udf(WS24SQL10001.AP3.Employees.LName,
              WS24SQL10001.AP3.Employees.FName,
              WS24SQL10001.AP3.Employees.MName
              ) AS Name,
      WS24SQL10001.AP3.Employees.Gender,
      final.address2_udf(
              WS24SQL10001.AP3.Employees.Address1,
              WS24SQL10001.AP3.Employees.Address2
              ) AS Address,
      WS24SQL10001.AP3.Employees.City,
      final.zipcode_udf(WS24SQL10001.AP3.Employees.ZipCode) AS ZipCode,
```

```
        final.phones_udf(PhoneNumber) AS PhoneNumber,
        WS24SQL10001.AP3.JobDepts.JobDept,
        WS24SQL10001.AP3.JobTitles.JobTitle,
        FORMAT(WS24SQL10001.AP3.Employees.JobSalary,
          'c', 'en-us') AS JobSalary,
        FORMAT(WS24SQL10001.AP3.Employees.StartDate,
          'd', 'en-us') AS StartDate,
        FORMAT(WS24SQL10001.AP3.Employees.LastUpdateDate,
          'd', 'en-us') AS LastUpdateDate,
        WS24SQL10001.AP3.JobDepts.Comments
    FROM WS24SQL10001.AP3.Employees
    LEFT JOIN WS24SQL10001.AP3.JobDepts
      ON WS24SQL10001.AP3.Employees.JobDeptID =
        WS24SQL10001.AP3.JobDepts.JobDeptID
    LEFT JOIN WS24SQL10001.AP3.JobTitles
      ON WS24SQL10001.AP3.Employees.JobTitleID =
        WS24SQL10001.AP3.JobTitles.JobTitleID
    WHERE WS24SQL10001.AP3.Employees.StartDate BETWEEN '12/30/1999'
        AND '6/1/2015'
    );


/* ****************************************************************************
        As an alternative, we can use an alias (`AS`) for each table to make
        our code tidier and to avoid repeating the database and schema before
        each table (`<database>.<schema>.<table>` or `<schema>.<table>`)
        throughout the query.

                    `emp` for `WS24SQL10001.AP3.Employees`
                    `jd`  for `WS24SQL10001.AP3.JobDepts`
                    `jt`  for `WS24SQL10001.AP3.JobTitles`
 **************************************************************************** */

CREATE VIEW final.EmployeesVW
AS
(
    SELECT DISTINCT final.EmpID_udf(emp.EmpID) AS EmpID,
      final.fullname_udf(emp.LName, emp.FName, emp.MName) AS Name,
      emp.Gender,
      final.address2_udf(emp.Address1, emp.Address2) AS Address,
      emp.City,
      final.zipcode_udf(emp.ZipCode) AS ZipCode,
      final.phones_udf(PhoneNumber) AS PhoneNumber,
      jp.JobDept,
      jt.JobTitle,
      FORMAT(emp.JobSalary, 'c', 'en-us') AS JobSalary,
      FORMAT(emp.StartDate, 'd', 'en-us') AS StartDate,
      FORMAT(emp.LastUpdateDate, 'd', 'en-us') AS LastUpdateDate,
      jp.Comments
    FROM WS24SQL10001.AP3.Employees AS emp
    LEFT JOIN WS24SQL10001.AP3.JobDeptsAS jd
      ON emp.JobDeptID = jd.JobDeptID
    LEFT JOIN WS24SQL10001.AP3.JobTitles AS jt
      ON emp.JobTitleID = jt.JobTitleID
    WHERE emp.StartDate BETWEEN '12/30/1999'
        AND '6/1/2015'
    );


/* ****************************************************************************
    5.6. As a bonus, there are other objects that can be added to tables.
        Constraints like keys restrict the possibility of losing data by
        dropping the wrong object or even inserting the wrong data into a
        field.

            ``A primary key, also called a primary keyword, is a key in a
          relational database that is unique for each record.  It is a unique
          identifier, such as a driver license number, telephone number
          (including area code), or vehicle identification number (VIN).  A
```

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

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

                      ALTER TABLE table_name
                      ADD CONSTRAINT pk_name
                        PRIMARY KEY (column_name)

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

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

                      ALTER TABLE child_table
                      ADD CONSTRAINT fk_name
                        FOREIGN KEY child_table_column
                        REFERENCES parent_table (parent_column)

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

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

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

ALTER TABLE AP1.Vendors                         -- 01. must make column
  ALTER COLUMN VendorID INT NOT NULL;           --     `VendorID` NOT NULL
                                                --     before making it a key

ALTER TABLE AP1.Vendors                         -- 02. altering table by adding
  ADD CONSTRAINT VendorID_PK                    --     constraint PRIMARY KEY
  PRIMARY KEY (VendorID);                       --     (PK) with `VendorID` in
                                                --     parenthesis


ALTER TABLE AP1.ContactUpdates                  -- 03. must make column
  ALTER COLUMN ContactUpdateID INT NOT NULL;    --     `ContactUpdateID` NOT
                                                --     NULL before making it a
                                                --     key

```

```
ALTER TABLE AP1.ContactUpdates                  -- 04. altering table by adding
  ADD CONSTRAINT ContactUpdateID_FK             --     constraint FOREIGN KEY
  FOREIGN KEY (VendorID)                        --     (FK) with `VendorID` in
  REFERENCES AP1.Vendors (VendorID);            --     parenthesis indicating
                                                --     the reference to the PK
                                                --     in parent table


/* ***************************************************************************
 7. What do you do now that the `Introduction to SQL` course has ended?

    7.1. Go to Microsoft Virtual Academy (https://mva.microsoft.com/) and
         continue learning including application to write reports
         (https://docs.microsoft.com/en-us/sql/reporting-services/), visualize
         and/or analyze data (https://powerbi.microsoft.com/).

    7.2. Register for the `Intermediate to SQL` course
         (https://www.campusce.net/bmcc/course/course.aspx?catId=328).

    7.3. If you are interested in a career in data science, learn Python
         (https://python.org/) and the multiple libraries available for data
         analysis (https://folvera.commons.gc.cuny.edu/?s=python).

    7.4. Contact me if you ever need help.

 8. Before you leave, figure out what the following prints.

    8.1. Change the value `your_name` with your name before running the code
         below to create the procedure.  Then execute it.

    8.2. If curious, visit http://ascii.cl/ for more information on ASCII
         (http://whatis.techtarget.com/definition/ASCII-American-Standard-Code-for-Information-Interchange).
 *************************************************************************** */

CREATE PROCEDURE final.message_sp
AS
BEGIN
  DECLARE @yourName VARCHAR(50)='your_name',    -- 01. param to hold your name
          @courseCd VARCHAR(15)='WS24SQL10001'; -- 02. code for the SQL course
  PRINT CONCAT (CHAR(084),CHAR(104),CHAR(097),CHAR(110),CHAR(107),CHAR(032),
       CHAR(121),CHAR(111),CHAR(117),CHAR(044),CHAR(032),@yourName,CHAR(044),
       CHAR(032),CHAR(102),CHAR(111),CHAR(114),CHAR(032),CHAR(116),CHAR(097),
       CHAR(107),CHAR(105),CHAR(110),CHAR(103),CHAR(032),CHAR(099),CHAR(108),
       CHAR(097),CHAR(115),CHAR(115),CHAR(032),@courseCd,CHAR(046),CHAR(013),
       CHAR(083),CHAR(101),CHAR(101),CHAR(032),CHAR(121),CHAR(111),CHAR(117),
       CHAR(032),CHAR(105),CHAR(110),CHAR(032),CHAR(116),CHAR(104),CHAR(101),
       CHAR(032),CHAR(105),CHAR(110),CHAR(116),CHAR(101),CHAR(114),CHAR(109),
       CHAR(101),CHAR(100),CHAR(105),CHAR(097),CHAR(116),CHAR(101),CHAR(032),
       CHAR(099),CHAR(108),CHAR(097),CHAR(115),CHAR(115),CHAR(046),CHAR(013),
       CHAR(013),CHAR(070),CHAR(046),CHAR(079),CHAR(108),CHAR(118),CHAR(101),
       CHAR(114),CHAR(097),CHAR(032),CHAR(040),CHAR(102),CHAR(111),CHAR(108),
       CHAR(118),CHAR(101),CHAR(114),CHAR(097),CHAR(064),CHAR(098),CHAR(109),
       CHAR(099),CHAR(099),CHAR(046),CHAR(099),CHAR(117),CHAR(110),CHAR(121),
       CHAR(046),CHAR(101),CHAR(100),CHAR(117),CHAR(041),CHAR(013),CHAR(104),
       CHAR(116),CHAR(116),CHAR(112),CHAR(058),CHAR(047),CHAR(047),CHAR(102),
       CHAR(111),CHAR(108),CHAR(118),CHAR(101),CHAR(114),CHAR(097),CHAR(046),
       CHAR(099),CHAR(111),CHAR(109),CHAR(109),CHAR(111),CHAR(110),CHAR(115),
       CHAR(046),CHAR(103),CHAR(099),CHAR(046),CHAR(099),CHAR(117),CHAR(110),
       CHAR(121),CHAR(046),CHAR(101),CHAR(100),CHAR(117),CHAR(047));
END;

EXEC final.message_sp;                          -- 03. executing final message

/* ***************************************************************************
 https://folvera.commons.gc.cuny.edu/?p=1298
 *************************************************************************** */
```