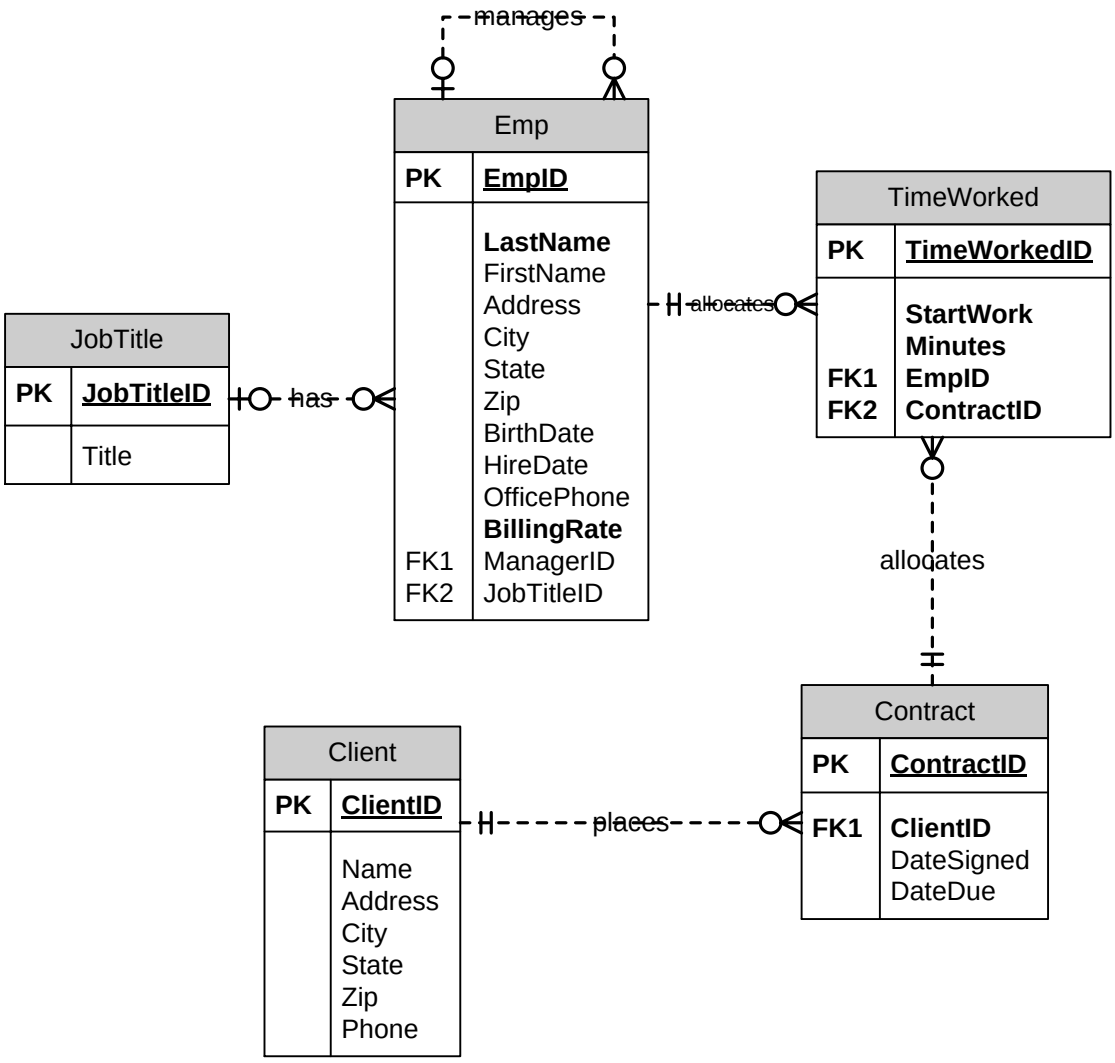
IS 475/675 SQL Extract, Transform and Load (ETL) Lab

The goals of this exercise are: (1) to review how to use the SQL Server import/export utility, (2) to populate multiple tables from a single table in SQL Server, and (3) to gain introductory knowledge of how to write a stored procedure and a function.

We are going to import data into a single table in SQL Server from an Excel worksheet that contains all the data needed for the five tables below. We are doing this because it is frequently easier to manipulate large data sets in SQL than it is in Excel. The data in the Excel worksheet has many redundant rows and we are going to "clean up" or "transform" that data in SQL. We are doing a simplified ETL (extract, transform and load) process in this lab. We will be "extracting" the data from the worksheet and importing it into SQL Server. Then we will "transform" the data into **five tables and "load" those five tables** with data.

The "database" used for this exercise is for a consulting company that keeps track of the time that an employee spends working on a contract for a client. Employees keep track of their time in minute increments to allocate their time that is chargeable to a given contract. The data stored in this database contains both current and historical rows.

Here is the ERD of the tables you will create and populate. All tables will have the prefix "el" to separate them from other tables in your database.



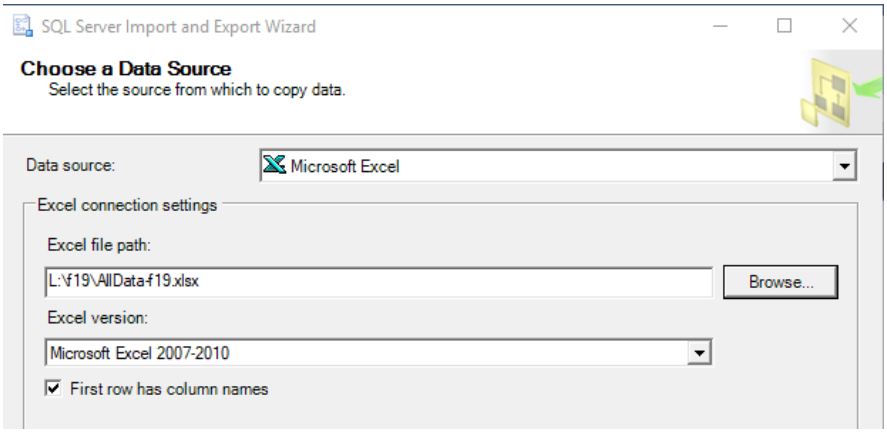
Task 1. Import data from Excel into an existing table in SQL Server

1) We are going to import an Excel worksheet that encompasses all the data in this database. Before we import the data, first copy the Excel workbook to your own data storage area on the "u:" drive. The worksheet is located in the workbook called k:\IS475\labfiles\AllData-f19.xlsx. The worksheet in that workbook that we are going to import is called "AllDataExcel". Open it up in Excel and take a look to become familiar with it. This worksheet represents the result table of a full join between all five tables in the database. Close it and Excel after you have checked out the worksheet.

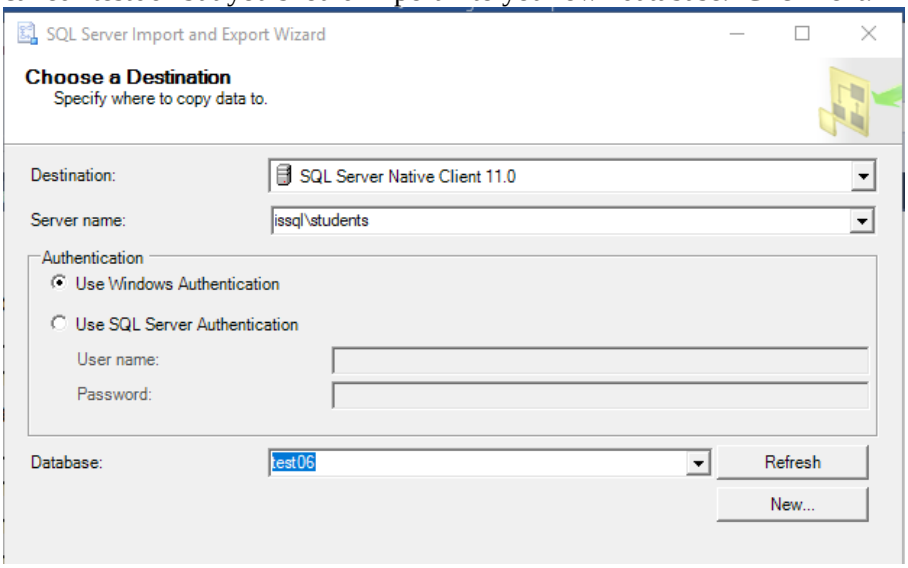
2) Go to SQL Server Management Studio. Create a table for the data to reside in after it is imported. The script file for this table is located in k:\IS475\labfiles\AllDataCreateTable-f19.sql. Open that file and execute it. It creates a table called "AllData" without any constraints (i.e. no primary key). Take a look at the data structure in Object Explorer.

3) Execute the SQL Server Import and Export Wizard by right clicking on the name of your database in ObjectExplorer, selecting the "Tasks" option, and selecting "Import Data".

4) The data source is Excel, and the workbook is the AllData workbook you copied from the k: drive. It is an Excel 2007-2010 workbook and the first row has column names. The screen shot below shows the location where I put the Excel workbook, but you need to browse for the location of your copy of the Excel workbook. Click next.



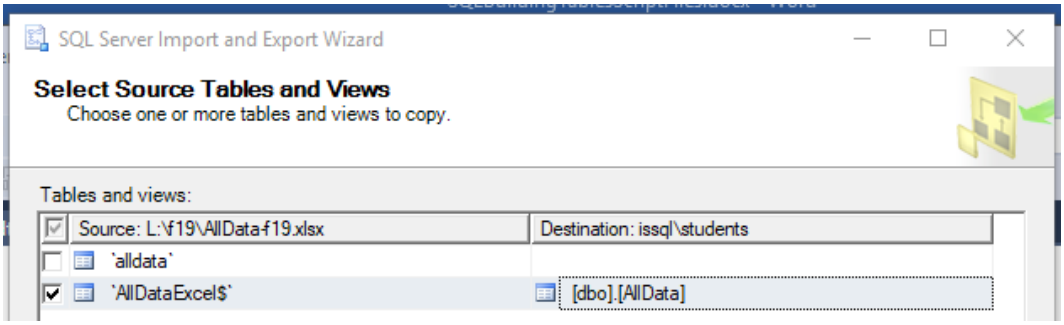
5) Select the destination SQL Server Native Client 11.0 from the drop-down list. The server name is issql\students, use Windows authentication and the database is your own. The screen shot below shows that the database I'm using is called "test06" but you should import into your own database. Click next.



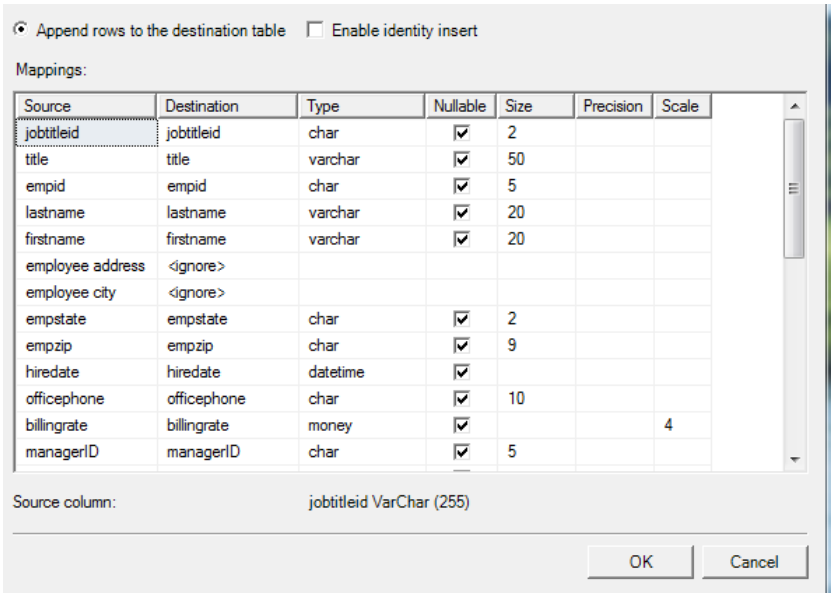
6) Select the radio button for “Copy data from one or more tables or views.” Click next.

7) Select the AllDataExcel$ worksheet selection as the source.

8) Select the name of the AllData table as the destination. SQL will show you a table called [dbo].[AllDataExcel$] as a possibility, but that means SQL will create the table for you. We have already created a table in step 2, and that is the table I want you to use. You will need to click the Destination column where it says "AllDataExcel$" to get the utility to show you a drop down box. Once the drop down box displays, then select the AllData table created in your database as shown at the top of the next page. Do NOT let the Import utility create the table for you and call that table AllDataExcel$. We do NOT want to let Excel dictate our data types for the table for this lab.



9) Check the mappings by clicking the “edit mappings” button. You should see mappings similar to that shown in the screen shot below. Because we created the table prior to importing the data, SQL should use the data types that we want when importing the data. Notice that there are two mappings in the screen below where the destination says “<ignore>”. We don’t want them to be ignored. We want the employee address source to go to the empaddres destination, and the employee city source to go to the empcity destination.

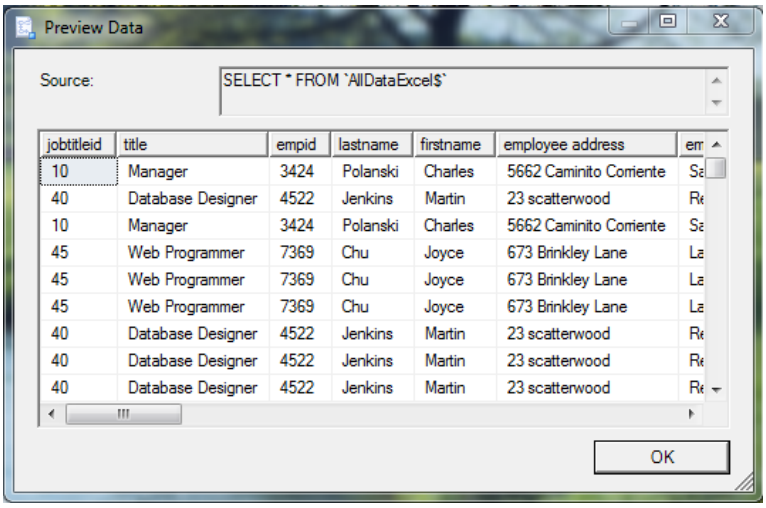


Click on the destination field name. A drop down box will be displayed. Change the two destinations as shown on the top of the next page.

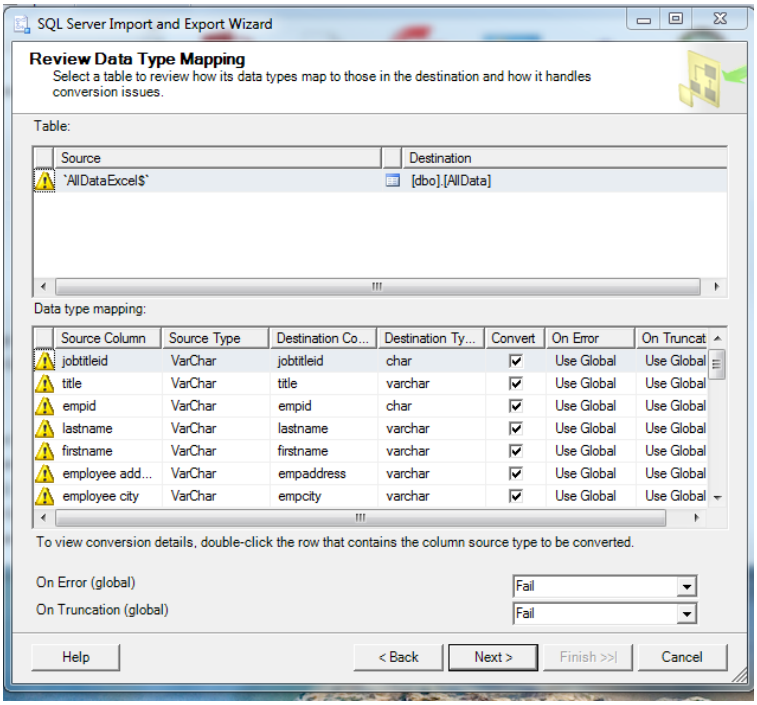
lastname	lastname	varchar	☑	20
firstname	firstname	varchar	☑	20
employee address	empaddress	varchar	☑	30
employee city	empcity	varchar	☑	20
empstate	empstate	char	☑	2
empzip	empzip	char	☑	9

Scroll through the rest of the mappings to make sure that none of the others are ignored. Click OK once you see that all the mapping are correct.

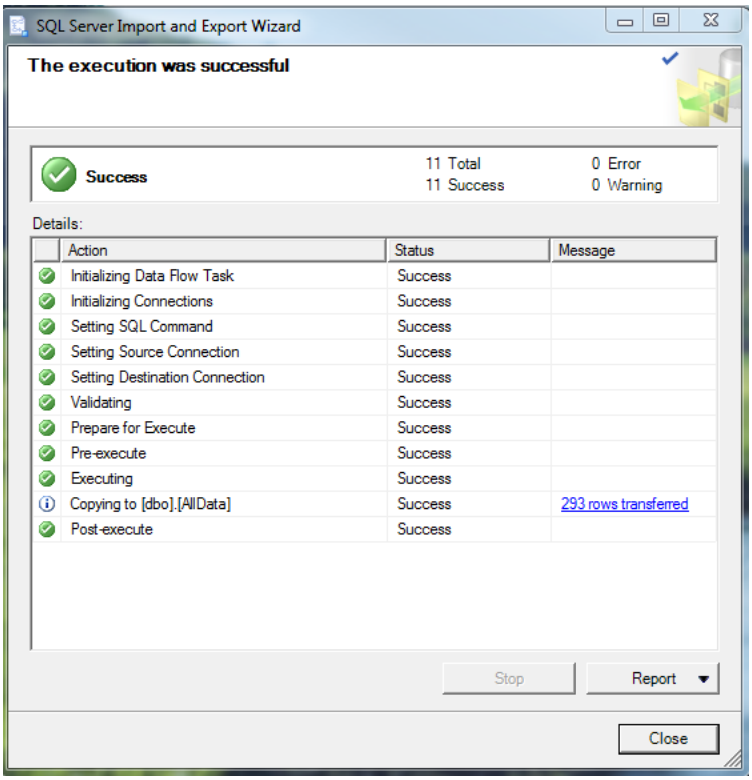
10) Preview the data by clicking on the Preview button. You should see the screen shot below. Click OK. Click next.



11) Run the package. There will be flagged errors that SQL Server is telling you may be of concern before you start the run as shown below. Click next. Click next when you get to the next page that says "run immediately". Then click "finish".



12) You should see the screen below telling you that 293 rows were transferred to the AllData table. Click close.



13) Go to SQL Server Management Studio. Look at the contents of the AllData table with the SQL command:

```
SELECT *
FROM AllData;
```

This table represents raw data transferred from Excel. There is no primary key, there are many rows with null values, and there is much redundant data. This data is not in a usable, non-redundant, well-designed database form. We are going to "transform" and "load" the data from this table into the five tables shown on the first page. We are moving data from an "unnormalized" table into a "normalized" database as depicted with the ERD on the first page.

Task 2. Moving data from one table to another in SQL Server

1) Create the five tables shown on the ERD on the first page of this lab by executing this script file: k:\IS475\LabFiles\Create-5-SQLTables. All tables will have the prefix 'el' to differentiate them from other tables in your database. (The "el" means "extract/load" for this lab.) These tables are created without any referential integrity constraints in order to make it easier to create/drop/re-create them during the transformation and loading process. The only constraint in some of the tables right now is a primary key constraint as shown on the ERD.

2) Transform and load the data into the JobTitle table. We want to create a JobTitle table that looks like this:

	jobtitleid	title
1	10	Manager
2	20	Business Analyst
3	40	Database Designer
4	45	Web Programmer
5	50	Interface Programmer
6	55	Graphics Designer
7	57	SAP Analyst

The data for the JobTitle table is in AllData. We are going to insert data from the AllData table directly into the JobTitle table through the INSERT INTO/SELECT FROM statement. The INSERT INTO/SELECT FROM statement takes data from a table via the SELECT FROM part of the statement and inserts it into another table via the INSERT INTO part of the statement. We are going to INSERT data, delete it again, and then INSERT it again a few times to practice seeing the results of different ways of SELECTing the data we want to insert.

To INSERT multiple rows from the Alldata table created and populated in task 1 into the elJobTitle table you just created with the Create-5-SQLTables script file, type the SQL code below.

```
INSERT INTO elJobTitle (jobtitleid, title)
SELECT      jobtitleid,
            title
FROM        alldata
```

3) Look at the contents of the elJobTitle table.

```
SELECT      *
FROM        elJobTitle
ORDER BY    jobtitleID;
```

How many rows were inserted into elJobTitle? Was that the correct INSERT/SELECT statement? The INSERT statement is being fed from the SELECT statement. How would you write an INSERT statement that would produce only the unique values for the jobtitle? Notice that the data loaded into elJobTitle contains many duplicate rows. These can be fixed by using the SELECT DISTINCT statement, so let's remove all those rows you just added and try it again.

4) Remove all rows from the eljobtitle table with the following command:

```
TRUNCATE TABLE elJobtitle
```

5) Try to add rows again:

```
INSERT INTO elJobTitle (jobtitleid, title)
SELECT DISTINCT jobtitleid,
       title
FROM alldata
```

How many rows this time? We want a total of 7 rows without any null values to be inserted into the table. How would you write a SELECT statement that excludes the null values in the jobtitleID?

6) Remove all rows from the eljobtitle table again:

```
TRUNCATE TABLE elJobTitle
```

We are doing this so we can add the rows again, but this time without the NULL values.

7) Try to add the rows again:

```
INSERT INTO elJobTitle (jobtitleid, title)
SELECT DISTINCT jobtitleid,
            title
FROM        alldata
WHERE jobtitleid is not null
```

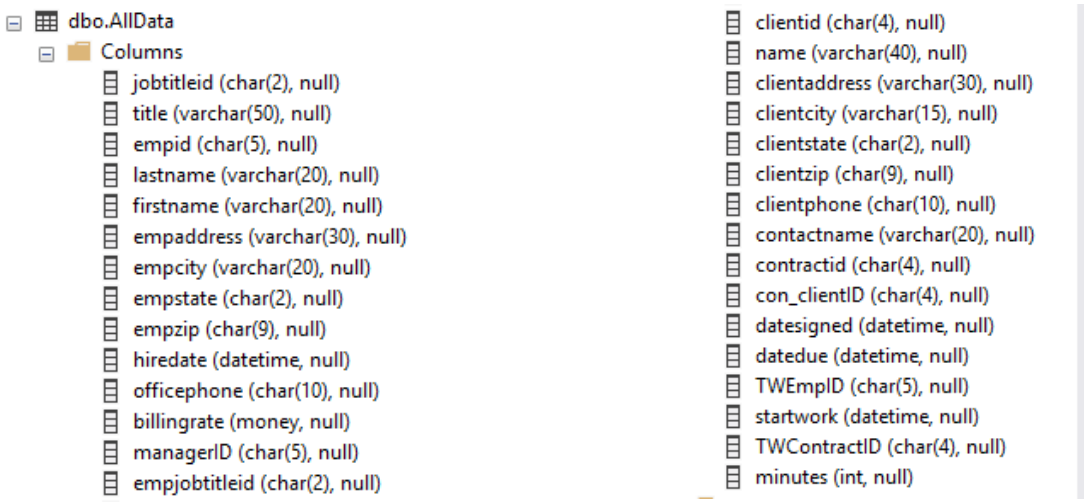
8) Look at the contents of the elJobTitle table. It should look like the sample table provided in Task 2, Step 2:

	jobtitleid	title
1	10	Manager
2	20	Business Analyst
3	40	Database Designer
4	45	Web Programmer
5	50	Interface Programmer
6	55	Graphics Designer
7	57	SAP Analyst

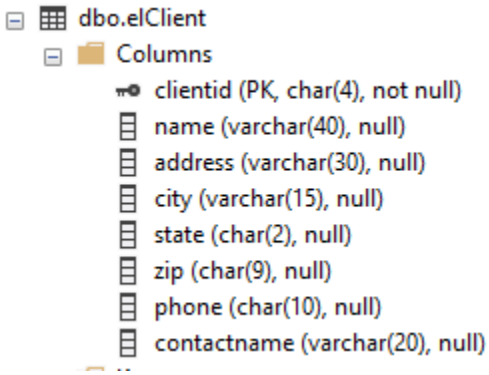
Save your SQL code as you populate each table. It may be necessary as the exercise continues to "re-populate" a table after initially populating a table.

Task 3. Populate the elClient table from the AllData table.

1) Use Object Explorer to look at the field names in the Alldata table and the elClient tables. Notice that the field names are not the same in the two tables. When taking data from the AllData table, you need to know the field names so that you can transform a field in AllData to a field in elClient. The field names are seldom exactly the same in your source table as they will be in the destination table, so you need to be familiar with both source and destination table descriptions. You can do this by using Object Explorer in SQL Server to look at the name of the columns in both tables. Here are the column names in AllData as displayed in Object Explorer. I put the AllData column names next to each other in pictures below so that they will take up less space on this document:



Here are elClient column names in ObjectExplorer:



2) Populate the elClient table from the AllData table. There should be 6 rows in the client table when you are done:

	clientid	name	address	city	state	zip	phone	contactname
1	1200	Cancer Research Society	76 Bell Parkway	las vegas	nv	89661	7023145617	mark jones
2	4900	Best Industries	214 Arrow Dr.	reno	nv	89510	7754456771	Nancy Ng
3	5600	Sobret Manufacturing	661 Terminal Way	reno	NV	89523	7752215661	candice guest
4	5700	Springwater Hotel/Casino	778 Springwater Drive	san diego	ca	92124	8584556991	Lori Mosqueda
5	8900	Coldcreek Cavern Club	781 Coldwater St.	san diego	ca	92128	8586433554	Bill Worth
6	9700	Crestwave Supply Co.	562 S. Park Place.	san diego	ca	92126	8583667889	Fred Lane

3) Here is the SQL statement to populate the elClient table from the AllData table. Note that you can include the field names in both the INSERT statement and the SELECT/FROM statement. SQL will "match up" the names based on sequential placement in the statements. So, the first field goes into the first field, the second into the second, etc.

```
INSERT INTO elclient (clientid, name, address, city, state, zip, phone, contactname)
SELECT distinct clientid, name, clientaddress, clientcity, clientstate, clientzip, clientphone,
contactname
FROM alldata
WHERE clientid is not null;
```

Task 4. Populate the elEmp table from the AllData table.

1) Use Object Explorer to look at the field names in the Alldata table and the elEmp tables. Notice that the field names are not the same in the two tables. Notice that the elEmp table has a birthdate while the Alldata table does not have a birthdate. This is not a problem because you can use the INSERT INTO/SELECT FROM with just the field names that you want to use to transfer data.

2) Populate the elEmp table from the AllData table. If you need help with the SQL code, look at the Appendix to the lab, but try and do it yourself first. Notice that there is no birthdate in AllData, but a birthdate is part of the elEmp table. If there is no data for a particular field, then don't reference that field in either the INSERT INTO or SELECT FROM statements. If the field is not referenced, it will be populated with a NULL value. There should be 8 rows in the elEmp table when you are done:

	empid	lastname	firstname	address	city	state	zip	birthdate	hiredate	officephone	billingrate	managerID	jobtitleID
1	2412	Perez	Martina	8372 Via Coronado	San diego	Ca	92126	NULL	2017-12-20 00:00:00.000	8582123848	89.00	3424	22
2	3411	Tristan	Elliott	344 Crestview Ct.	RENO	NV	89502	NULL	2013-05-25 00:00:00.000	7757843221	225.00	7819	55
3	3424	Polanski	Charles	5662 Caminito Corriente	San Diego	ca	92128	NULL	2019-01-12 00:00:00.000	8582339001	95.00	7819	10
4	4522	Jenkins	Martin	23 scatterwood	Reno	nv	89508	NULL	2017-01-14 00:00:00.000	7754639001	90.00	3424	40
5	5633	Flanders	Janice	P.O. Boz 445	Sparks	Nv	89431	NULL	2009-02-12 00:00:00.000	7756351441	65.00	3424	45
6	7369	Chu	Joyce	673 Brinkley Lane	Las Vegas	NV	89111	NULL	2018-08-23 00:00:00.000	7023456771	95.00	7819	45
7	7715	Kendall	Brent	662 westwood	san diego	CA	92128	NULL	2019-01-12 00:00:00.000	5802389912	75.00	3424	50
8	7819	Martinson	Cassandra	123 Main St.	Reno	nv	89557	NULL	2017-10-06 00:00:00.000	7757812334	125.00	NULL	10

Task 5. Populate the elContract table from the AllData table.

1) Use Object Explorer to look at the field names in the Alldata table and the elContract tables. Notice that the field names are not the same in the two tables.

2) Populate the elContract table from the Alldata table. There should be 7 rows in the elContract table when you are done, as shown below. If you can't figure out the SQL code, it is available in the Appendix.

	contractid	Clientid	datesigned	datedue
1	333	9700	2019-12-18 00:00:00.000	2019-11-18 00:00:00.000
2	444	1200	2019-07-12 00:00:00.000	2019-12-01 00:00:00.000
3	555	5600	2019-07-10 00:00:00.000	2019-12-15 00:00:00.000
4	662	4900	2019-07-13 00:00:00.000	2019-11-01 00:00:00.000
5	666	4900	2019-04-03 00:00:00.000	2019-08-22 00:00:00.000
6	777	9700	2018-12-08 00:00:00.000	2019-09-15 00:00:00.000
7	781	5700	2018-12-12 00:00:00.000	2019-09-02 00:00:00.000

3) Look at the data in the elContract table (on the previous page). Notice that one of the date signed values is later than the date due value for the row (see contractID 333). That data might be invalid. If you were moving that data in "real life" would you want to put it into the contract table? How would you check for that invalid data before putting it into the contract table?

4) Truncate the elContract table to delete all the data from table. We are going to populate it again, but this time we will check for this "dirty" data.

5) Populate the elContract table from the Alldata table, but do not include any rows where the datesigned is less than the datedue. The elContract table should have 6 rows. If you can't figure out how to do it, the SQL code is in the appendix.

	contractid	Clientid	datesigned	datedue
1	444	1200	2019-07-12 00:00:00.000	2019-12-01 00:00:00.000
2	555	5600	2019-07-10 00:00:00.000	2019-12-15 00:00:00.000
3	662	4900	2019-07-13 00:00:00.000	2019-11-01 00:00:00.000
4	666	4900	2019-04-03 00:00:00.000	2019-08-22 00:00:00.000
5	777	9700	2018-12-08 00:00:00.000	2019-09-15 00:00:00.000
6	781	5700	2018-12-12 00:00:00.000	2019-09-02 00:00:00.000

In "real-life" if any rows are going to be rejected from placement in a table, those rows will be placed in another table to show that they are "rejects". Let's do that task.

6) If you want to create a table and populate it at the same time, a slightly different syntax is used. The SELECT/INTO statement creates and populates a table with a single statement. Here are the two statements together:

```
SELECT distinct contractID, con_clientID, datesigned, datedue
INTO elContractReject
FROM alldata
WHERE contractID is not null and
       datesigned > datedue;
```

This statement will create the elContractReject table and populate it with the rejected contract. Look at the contents of the elContractReject table you just created and populated. The result table produced is below.

```
SELECT *
FROM elContractReject
```

	contractID	con_clientID	datesigned	datedue
1	333	9700	2019-12-18 00:00:00.000	2019-11-18 00:00:00.000

Note that the field names used in the AllData table are the field names that were used to create the table. The data types will be the same, too. That is OK for a table that contains "reject" data because the data will most likely be individually viewed and processed by a person, rather than by a computer. We don't use the SELECT/INTO/FROM statement to create tables that will be used exclusively by a computer because the data types are not usually defined enough to protect the integrity of the data.

Task 6. Populate the elTimeWorked table from the AllData table.

1) Use Object Explorer to look at the field names in the Alldata table and the elTimeWorked tables. Notice that the field names are not the same in the two tables. Also notice that the primary key in elTimeWorked is a surrogate (an identity type) so you do not put any data in that field when populating the table. A final thing to notice is that there is a worktypeID in the elTimeWorked table, but there is no worktypeID in the Alldata table. In addition, think about which field in the Alldata table you want to check to see whether it is NULL when inserting data into the elTimeWorked table.

2) Populate the elTimeWorked table from the AllData table. There should be 288 rows in the elTimeWorked table when you are done. I have provided the first 50 rows below for your information. If you can't figure out how to write the code, it is available in the Appendix.

	TimeWorkedID	EmpID	startwork	ContractID	Worktypeid	minutes
1	1	2412	2018-11-03 13:00:00.000	777	NULL	225
2	2	2412	2018-11-04 13:00:00.000	777	NULL	25
3	3	2412	2018-11-11 11:30:00.000	777	NULL	225
4	4	2412	2018-11-12 11:30:00.000	777	NULL	25
5	5	2412	2018-11-13 13:00:00.000	777	NULL	270
6	6	2412	2018-11-14 13:00:00.000	777	NULL	70
7	7	2412	2018-11-16 12:00:00.000	666	NULL	45
8	8	2412	2018-11-21 11:30:00.000	777	NULL	270
9	9	2412	2018-11-22 11:30:00.000	777	NULL	70
10	10	2412	2018-11-24 10:30:00.000	666	NULL	45
11	11	2412	2018-11-25 12:00:00.000	777	NULL	25
12	12	2412	2018-11-25 19:00:00.000	777	NULL	225
13	13	2412	2018-11-26 12:00:00.000	666	NULL	90
14	14	2412	2018-12-03 10:30:00.000	777	NULL	25
15	15	2412	2018-12-03 17:30:00.000	777	NULL	225
16	16	2412	2018-12-04 10:30:00.000	666	NULL	90
17	17	2412	2018-12-05 12:00:00.000	777	NULL	70
18	18	2412	2018-12-05 20:00:00.000	777	NULL	270
19	19	2412	2018-12-13 10:30:00.000	777	NULL	70
20	20	2412	2018-12-13 18:30:00.000	777	NULL	270
21	21	3411	2018-11-27 12:00:00.000	444	NULL	405
22	22	3411	2018-11-27 20:00:00.000	444	NULL	45
23	23	3411	2018-12-01 12:00:00.000	444	NULL	625
24	24	3411	2018-12-05 10:30:00.000	444	NULL	405
25	25	3411	2018-12-05 18:30:00.000	444	NULL	45

	TimeWorkedID	EmpID	startwork	ContractID	Worktypeid	minutes
26	26	3411	2018-12-07 12:00:00.000	444	NULL	450
27	27	3411	2018-12-07 20:00:00.000	444	NULL	90
28	28	3411	2018-12-09 10:30:00.000	444	NULL	625
29	29	3411	2018-12-11 12:00:00.000	444	NULL	670
30	30	3411	2018-12-15 10:30:00.000	444	NULL	450
31	31	3411	2018-12-15 18:30:00.000	444	NULL	90
32	32	3411	2018-12-19 10:30:00.000	444	NULL	670
33	33	3411	2019-07-31 09:30:00.000	444	NULL	480
34	34	3411	2019-07-31 19:30:00.000	444	NULL	120
35	35	3411	2019-08-04 09:30:00.000	444	NULL	700
36	36	3411	2019-08-08 08:00:00.000	444	NULL	480
37	37	3411	2019-08-08 18:00:00.000	444	NULL	120
38	38	3411	2019-08-10 09:30:00.000	444	NULL	525
39	39	3411	2019-08-10 19:30:00.000	444	NULL	165
40	40	3411	2019-08-12 08:00:00.000	444	NULL	700
41	41	3411	2019-08-14 09:30:00.000	444	NULL	745
42	42	3411	2019-08-18 08:00:00.000	444	NULL	525
43	43	3411	2019-08-18 18:00:00.000	444	NULL	165
44	44	3411	2019-08-22 08:00:00.000	444	NULL	745
45	45	3424	2018-10-29 13:00:00.000	662	NULL	45
46	46	3424	2018-11-06 11:30:00.000	662	NULL	45
47	47	3424	2018-11-08 13:00:00.000	662	NULL	90
48	48	3424	2018-11-16 11:30:00.000	662	NULL	90
49	49	3424	2018-11-22 12:00:00.000	777	NULL	405
50	50	3424	2018-11-27 12:00:00.000	444	NULL	405

Task 7. Put your code to Create and Populate the Reject Contract Table into a Stored Procedure.

In Task #5, you created and populated a table to store rejected contracts with the following statement. We will now turn the statement below into a stored procedure.

```
SELECT distinct contractID, con_clientID, datesigned, datedue
INTO elContractReject
FROM alldata
WHERE contractID is not null and
       datesigned > datedue;
```

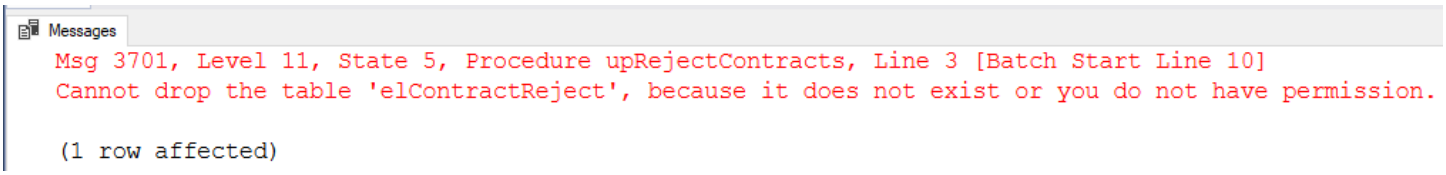
1) Here is a stored procedure with that statement. Type and execute this code to create the stored procedure called "upRejectContracts."

```
CREATE PROCEDURE upRejectContracts
AS
DROP TABLE elContractReject
SELECT distinct contractID, con_clientID, datesigned, datedue
INTO elContractReject
```

```
FROM alldata
WHERE contractID is not null and
       datesigned > datedue;
```

Type EXEC upRejectContracts to execute the stored procedure.

Notice that there is first a DROP TABLE statement in the stored procedure. What happens if the table doesn't exist and you execute the stored procedure? The stored procedure will still run and create the elContractReject table, but you will get the following result that includes an error message:



2) Add in an IF statement to avoid the error.

```
ALTER PROCEDURE upRejectContracts
AS
IF object_id ('elContractReject') is not null
BEGIN
   DROP TABLE elContractReject
END
SELECT distinct contractID, con_clientID, datesigned, datedue
INTO elContractReject
FROM alldata
WHERE contractID is not null and
       datesigned > datedue;
```

The BEGIN and END allow a programmer to identify multiple lines of code that must be done if the condition in the IF statement is true.

Note that we did not have an ELSE to match up with the IF statement in this situation. In addition, there is a BEGIN and END to separate the task done for the IF statement with the SELECT/INTO statement used to create and populate the table. The BEGIN and END can be used with any IF statement to help identify conditional code, but are only necessary where multiple lines of code are conditional to the IF statement.

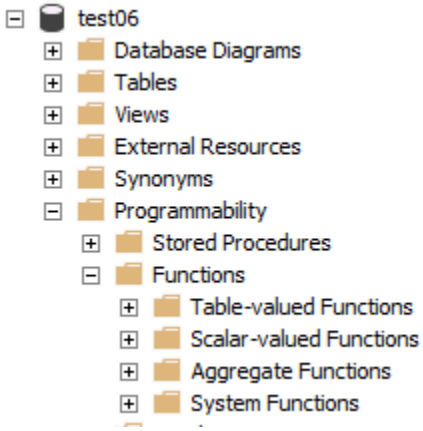
Task 8. Create a custom function.

The purpose of this function is to format a 10 byte character field into a phone number formatted in a U.S. standard format.

1) Before we create the function write a SQL SELECT statement to generate the following result table from the elEmp table. The OfficePhone field is a character data type, so don't use the FORMAT function, format the phone number with SUBSTRING functions that are then concatenated together. You have done this task in prior homework assignments so it should be familiar to you. If you don't remember how to write it, the code is available in the appendix.

	lastname	firstname	OfficePhoneNumber
1	Perez	Martina	(858) 212-3848
2	Tristan	Elliott	(775) 784-3221
3	Polanski	Charles	(858) 233-9001
4	Jenkins	Martin	(775) 463-9001
5	Flanders	Janice	(775) 635-1441
6	Chu	Joyce	(702) 345-6771
7	Kendall	Brent	(580) 238-9912
8	Martinson	Cassandra	(775) 781-2334

2) We will use the SUBSTRING concatenation in that query to create a user-defined function. In the Object Explorer window, drill down to Programmability|Functions|Scalar-valued Functions.



Right click on Scalar-valued Function and select the option for New Scalar-valued function

A new query window will open filled with code and lots of comments but here are the important parts of the function's framework:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:           <Author,,Name>
-- Create date: <Create Date, ,>
-- Description:      <Description, ,>
-- =============================================
CREATE FUNCTION <Scalar_Function_Name, sysname, FunctionName>
(
       -- Add the parameters for the function here
       <@Param1, sysname, @p1> <Data_Type_For_Param1, , int>
)
RETURNS <Function_Data_Type, ,int>
AS
BEGIN
       -- Declare the return variable here
       DECLARE <@ResultVar, sysname, @Result> <Function_Data_Type, ,int>

       -- Add the T-SQL statements to compute the return value here
       SELECT <@ResultVar, sysname, @Result> = <@Param1, sysname, @p1>

       -- Return the result of the function
       RETURN <@ResultVar, sysname, @Result>

END
GO
```

3) Delete all the code from where it starts with CREATE FUNCTION. Replace all that code with this code:

```
CREATE FUNCTION PrettyPhone
 (
      @phonein char(10)
 )
RETURNS char(15)
BEGIN
```

```
  RETURN '(' + SUBSTRING(@phonein,1,3) + ') ' + SUBSTRING(@phonein,4,3) + '-' +
SUBSTRING(@phonein,7,4)
END
GO
```

A function usually has arguments and then usually returns a value. In the statement above, the argument that we are passing our function is called @phonein and is a char(10) field (that means it is a fixed character data type of 10 bytes in length). The data that will be returned from the function is a char(15) field that has been broken down and recreated with SUBSTRING and concatenation.

Execute the code. It should say "command(s) completed successfully".

4) Now use the function within a query, execute the query and look at the results.

```
SELECT lastname,
       firstname,
       dbo.prettyphone(officephone) OfficePhoneNumber
FROM   elEmp;
```

The function accepts a char field as input. In the query above, the function is accepting the argument of the officephone field. It then modifies it with SUBSTRING and concatenation and returns a formatted phone number.

Task 9. Create a function by yourself.

The purpose of this function is to calculate how long a person has worked for the consulting company. The DATEDIFF function does not actually calculate the length of time correctly and there is no standard function in SQL Server that will do it accurately for you.

1) To see what I mean, try typing in your birthdate and the current date to see what year result you get. If you don't want to use your own birthdate, then here is an example:

```
SELECT DATEDIFF(yy, '12-29-1988', getdate())
```

	(No column name)
1	31

How old is a person today who was born on December 29, 1988? Does the DATEDIFF function calculate the age correctly?

2) Let's try it in days instead of years and then divide it by 365.25 to turn it into years. Now what do you get? A much more accurate number, right? (Your number will be a little different. I ran this code on 10/30/2019)

```
SELECT DATEDIFF(dd, '12-29-1988', getdate())/ 365.25
```

	(No column name)
1	30.833675

3) How do you see just the "characteristic" part rather than the characteristic part plus the mantissa?

```
SELECT FLOOR(DATEDIFF(dd, '12-29-1988', getdate())/ 365.25)
```

	(No column name)
1	30

4) Create a function called dbo.YearsCalc so that you can generate a result table that looks like the one below from the following SELECT statement:

```
SELECT lastname,
       firstname,
      hiredate,
      dbo.YearsCalc(hiredate) YearsHired
FROM   elEmp;
```

	lastname	firstname	hiredate	YearsHired
1	Perez	Martina	2017-12-20 00:00:00.000	1
2	Tristan	Elliott	2013-05-25 00:00:00.000	6
3	Polanski	Charles	2019-01-12 00:00:00.000	0
4	Jenkins	Martin	2017-01-14 00:00:00.000	2
5	Flanders	Janice	2009-02-12 00:00:00.000	10
6	Chu	Joyce	2018-08-23 00:00:00.000	1
7	Kendall	Brent	2019-01-12 00:00:00.000	0
8	Martinson	Cassandra	2017-10-06 00:00:00.000	2

If you can't figure out how to write the function, then look at the appendix for the solution.

Task 10. Put your functions and Stored Procedure knowledge to use.

Create a new version of the elEmp table called elEmpVer2. This version of the elEmp table should have the fields shown in the result table below and you should load the data from elEmp using the functions you created. Use the SELECT/INTO/FROM so that the elEmpVer2 table is created when you populate it. The solution is in the appendix.

After you create the new table called elEmpVer2, look at the contents of the table. The result table from the statement `SELECT * FROM elEmpVer2` is shown below.

	lastname	firstname	hiredate	YearsHired	OfficePhoneNumber
1	Perez	Martina	2017-12-20 00:00:00.000	1	(858) 212-3848
2	Tristan	Elliott	2013-05-25 00:00:00.000	6	(775) 784-3221
3	Polanski	Charles	2019-01-12 00:00:00.000	0	(858) 233-9001
4	Jenkins	Martin	2017-01-14 00:00:00.000	2	(775) 463-9001
5	Flanders	Janice	2009-02-12 00:00:00.000	10	(775) 635-1441
6	Chu	Joyce	2018-08-23 00:00:00.000	1	(702) 345-6771
7	Kendall	Brent	2019-01-12 00:00:00.000	0	(580) 238-9912
8	Martinson	Cassandra	2017-10-06 00:00:00.000	2	(775) 781-2334

Appendix: Code Required For Lab

Task 4: Populate elEmp table

```
INSERT INTO elemp
       ([empid],[lastname],[firstname],[address],[city],[state],[zip],[hiredate],[officephone],
       [billingrate],[managerID],[jobtitleID])
SELECT DISTINCT empid, lastname, firstname, empaddress, empcity, empstate, empzip, hiredate,
officephone, billingrate, managerid, empjobtitleid
FROM AllData
WHERE empid is not null;
```

Task 5: Populate elContract table

```
INSERT INTO elContract (contractid,clientid, datesigned, datedue)
SELECT distinct contractid, con_clientID, datesigned, datedue
FROM alldata
WHERE contractid is not null

INSERT INTO elContract (contractID, clientID, datesigned, datedue)
SELECT distinct contractID, con_clientID, datesigned, datedue
FROM alldata
WHERE contractid is not null and
datesigned <= datedue;
```

Task 6: Populate elTimeWorked table

```
INSERT INTO elTimeWorked (empid, startwork, contractid, minutes)
SELECT distinct twempid, startwork, twcontractid, minutes
FROM alldata
WHERE twempid is not null
```

Task 7: Format the telephone number using the SUBSTRING

```
SELECT lastname,
       firstname,
       '(' + SUBSTRING(officephone,1,3) + ') ' + SUBSTRING(officephone,4,3) + '-' +
SUBSTRING(officephone,7,4) OfficePhoneNumber
FROM  elEmp
```

Task 8: Create a Function to Display data in Years

```
CREATE FUNCTION [dbo].[YearsCalc]
 ( @datein datetime
 )
RETURNS int
BEGIN
  RETURN floor(datediff(dd, @datein, getdate())/365.25)
END
GO
```

Task 10: Create a Stored Procedure to create and populate a new table

```
CREATE PROCEDURE upEmpVer2 AS
SELECT lastname,
       firstname,
       hiredate,
       dbo.YearsCalc(hiredate) YearsHired,
       dbo.prettyphone(officephone) OfficePhoneNumber
INTO elEmpVer2
FROM   elEmp
```