In the simplest ETL scenarios, you use an SSIS Data Flow to extract data from a single source table and populate the corresponding destination table. In practice, though, you usually won’t see such trivial scenarios: the more common ETL scenarios will require you to access two or more data sources simultaneously and merge their results together into a single destination structure. For instance, you may have a normalized source system that uses three or more tables to represent the product catalog, whereas the destination represents the same information using a single denormalized table (perhaps as part of a data warehouse schema). In this case you would need to join the multiple source tables together in order to present a unified structure to the destination table. This joining may take place in the source query in the SSIS package or when using a Lookup Transform in an SSIS Data Flow.

Another less obvious example of joining data is loading a dimension that would need to have new rows inserted and existing rows updated in a data warehouse. The source data is coming from an OLTP database and needs to be compared to the existing dimension to find the rows that need updating. Using the dimension as a second source, you can then join the data using a Merge Join Transformation in your Data Flow. The joined rows can then be compared to look for changes.

The Lookup Transformation in SSIS enables you to perform the similar relational inner and outer

hash-joins. The main difference is that the operations occur outside the realm of the database engine and in the SSIS Data Flow. Typically, you would use this component within the context of an integration process, such as the ETL layer that populates a data warehouse from source systems. For example, you may want to populate a table in a destination system by joining data from two separate source systems on different database platforms.

The component can join only two data sets at a time, so in order to join three or more data sets, you would need to chain multiple Lookup Transformations together, using an output from one Lookup Transformation as an input for another. Compare this to relational join semantics, whereby in a similar fashion you join two tables at a time and compose multiple such operations to join three or more tables.

The transformation is written to behave in a synchronous manner, meaning it does not block the pipeline while it is doing its work. While new rows are entering the Lookup Transformation, rows that have already been processed are leaving through one of four outputs. However, there is a catch here: in certain caching modes (discussed later in this chapter) the component will initially block the package’s execution for a period of time while it loads its internal caches with the Lookup data.

The component provides several modes of operation that enable you to compare performance and resource usage.

In full-cache mode, one of the tables you are joining is loaded in its entirety into memory, and then the rows from the other table are flowed through the pipeline one buffer at a time, and the selected join operation is performed.

With no up-front caching, each incoming row in the pipeline is compared one at a time to a specified relational table. Between these two options is a third that combines their behavior. Each of these modes is (“Full-Cache Mode,” “No-Cache Mode,” and “Partial-Cache Mode”).

Example

Table 1. Name of Employee

|  |  |  |
| --- | --- | --- |
| Id | firstName | LastName |
| 1 | Alex | Binuro |
| 2 | Amy | Hurnandez |
| 3 | Adam | Smith |
| 4 | Ben | Tabab |
| 5 | Tommy | Dilleno |
| 6 | Mark | Antony |
| 7 | Paul | Mpemba |

Table 2. Employee Address

|  |  |  |
| --- | --- | --- |
| ID | StreetNumber | StreetName |
| 1 | 1245 | Gaucher blvd |
| 2 | 3564 | Putty Hill ave |
| 3 | 7586 | Park ave |
| 4 | 15863 | Parker st. |
| 5 | 47821 | Plano pkwy |
| 6 | 45879 | Martin blvd |
| 7 | 666842 | Park ave |

Now we are joining those two tables using lookup transformation. To do that, first load those two tables into the SQL tables.

create table Employee (id int, firstName varchar (15), LastName varchar(15))

insert into Employee (Id, firstName, LastName)

values

(1, 'Alex','Binuro')

,(2, 'Amy' ,'Hurnandez')

,(3, 'Adam','Smith')

,(4, 'Ben' ,'Tabab')

,(5, 'Tommy', 'Dilleno')

,(6, 'Mark', 'Antony')

,(7, 'Paul', 'Mpemba')

create table addresses (id int, StreetNumber int, StreetName varchar(25))

insert into addresses (ID, StreetNumber, StreetName)

values

(1, 1245, 'Gaucher blvd.')

,(2, 3564, 'Putty Hill ave.')

,(3, 7586, 'Park ave.')

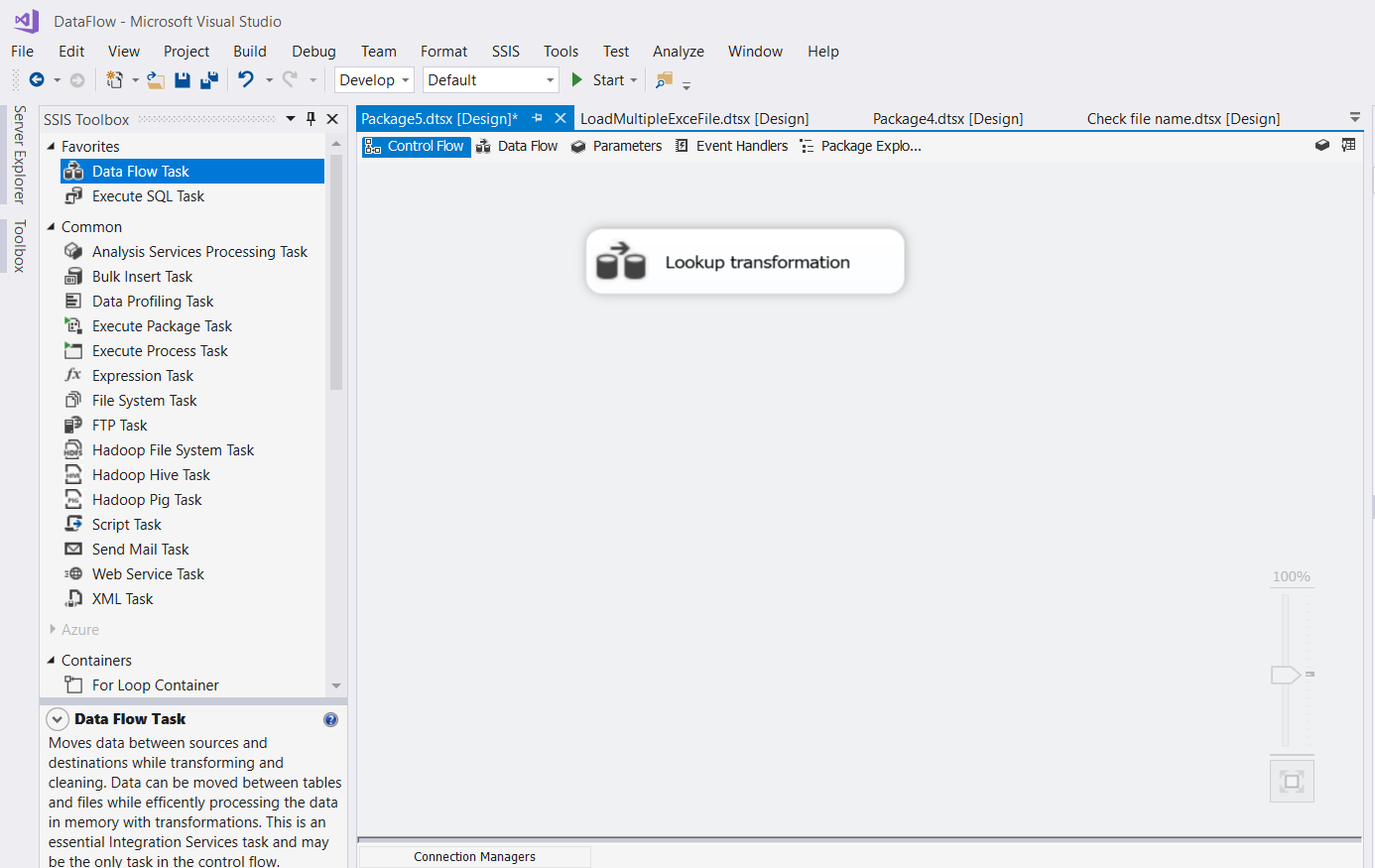
,(4, 15863, 'Parker st.')

,(5, 47821, 'Plano pkwy')

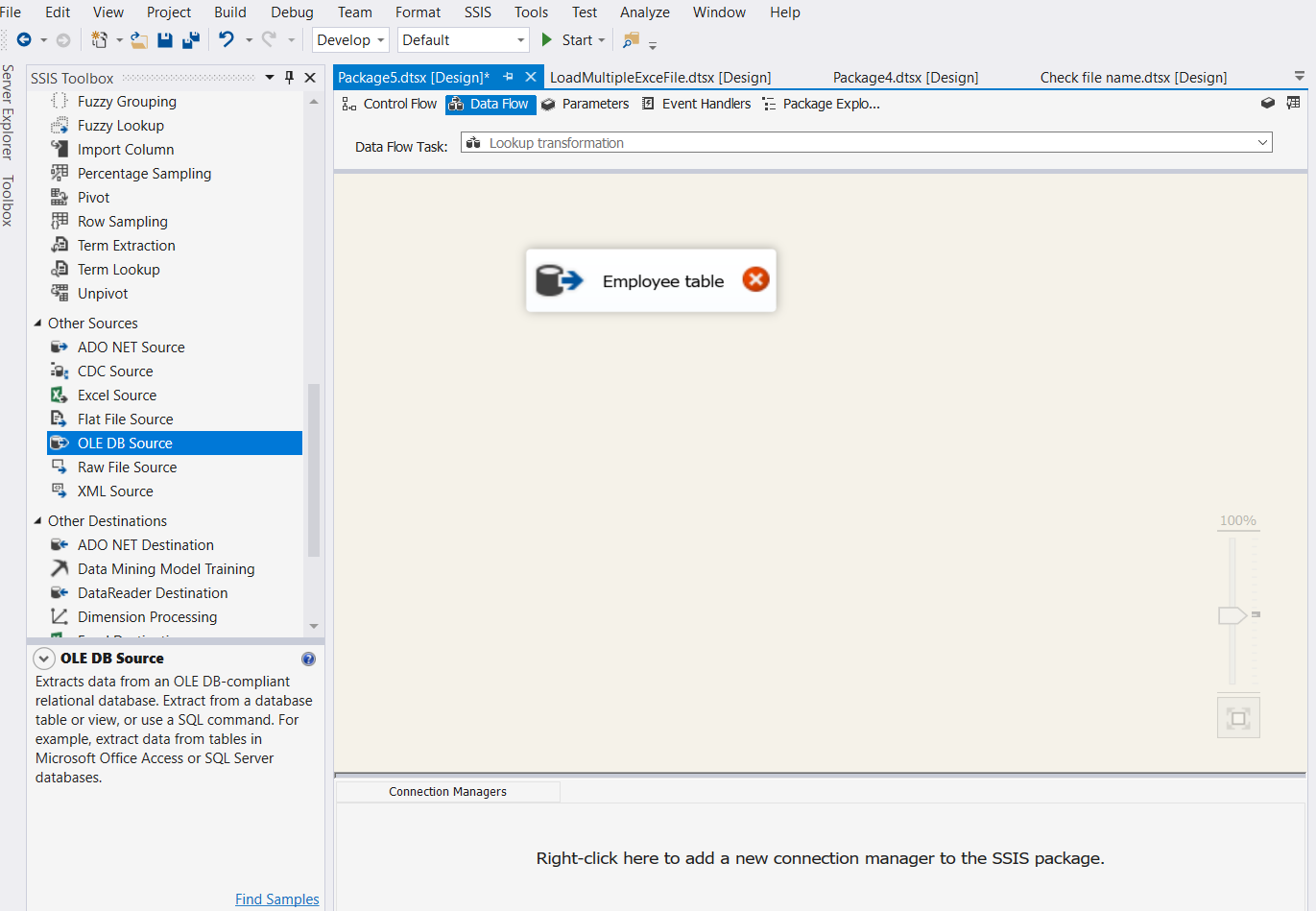
,(6, 45879, 'Martin blvd.')

,(7, 666842, 'Park ave')

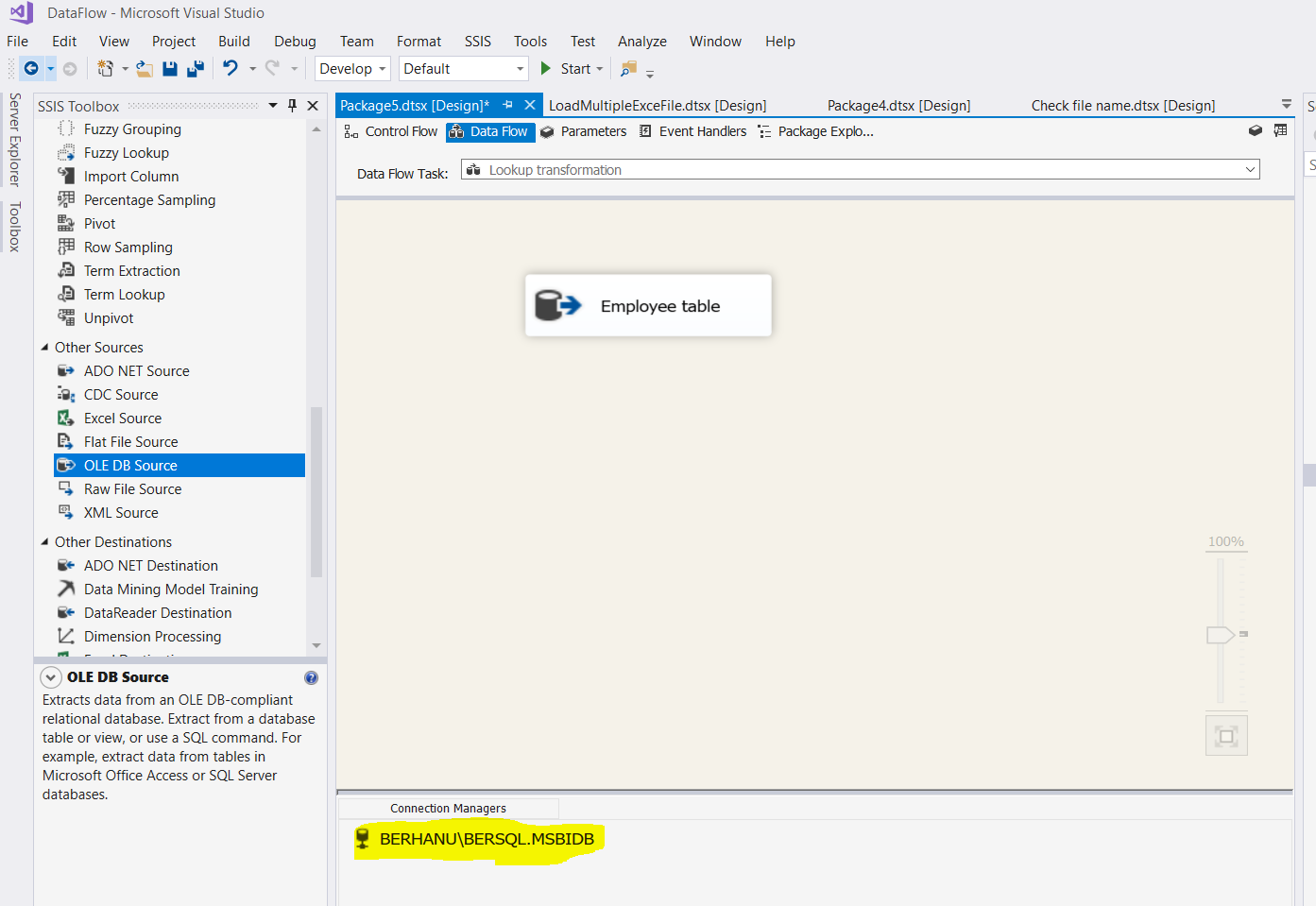
Open visual studio and create new project and add new package into the solution explorer. Then in the control flow add new data flow task and rename it as lookup transformation



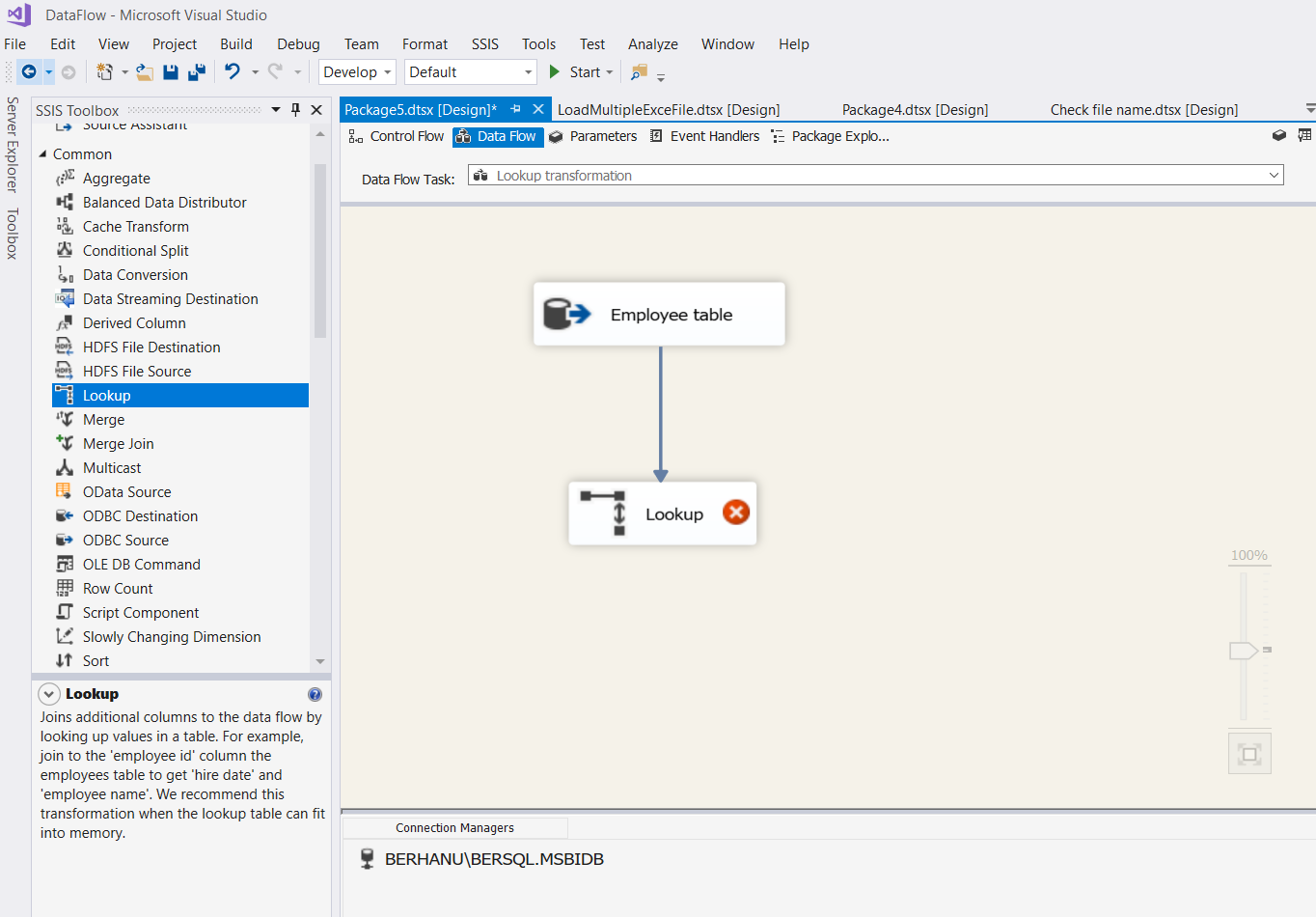
Then open the data flow task editor then it will take you to the dataflow window and then drag and drop the OLE DB source, as depicted as follows.



Then configure the OLE DB data source.



Then drag and drop lookup transformation to the data flow window and connect with the OLE DB data source. Right click on the lookup transformation and open the lookup transformation editor as shows in the following figure.

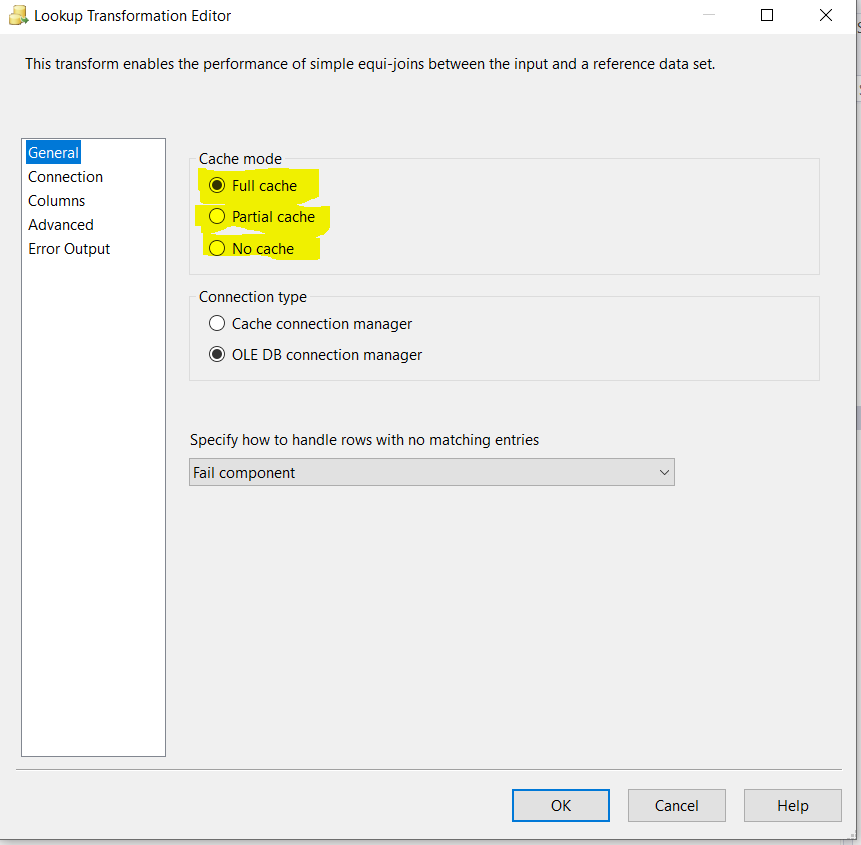


When you open the lookup transformation editor in the general window there are three cache mode 1. Full cache, 2. Partial cache, 3 No cache

1. Full cache mode: one of the tables you are joining is loaded in its entirety into memory, and then the rows from the other table are flowed through the pipeline one buffer at a time, and the selected join operation is performed.
2. No cache mode: each incoming row in the pipeline is compared one at a time to a specified relational table.
3. Partial cache mode: Between full cache mode and no cache mode is partial cache mode that combines their behavior.

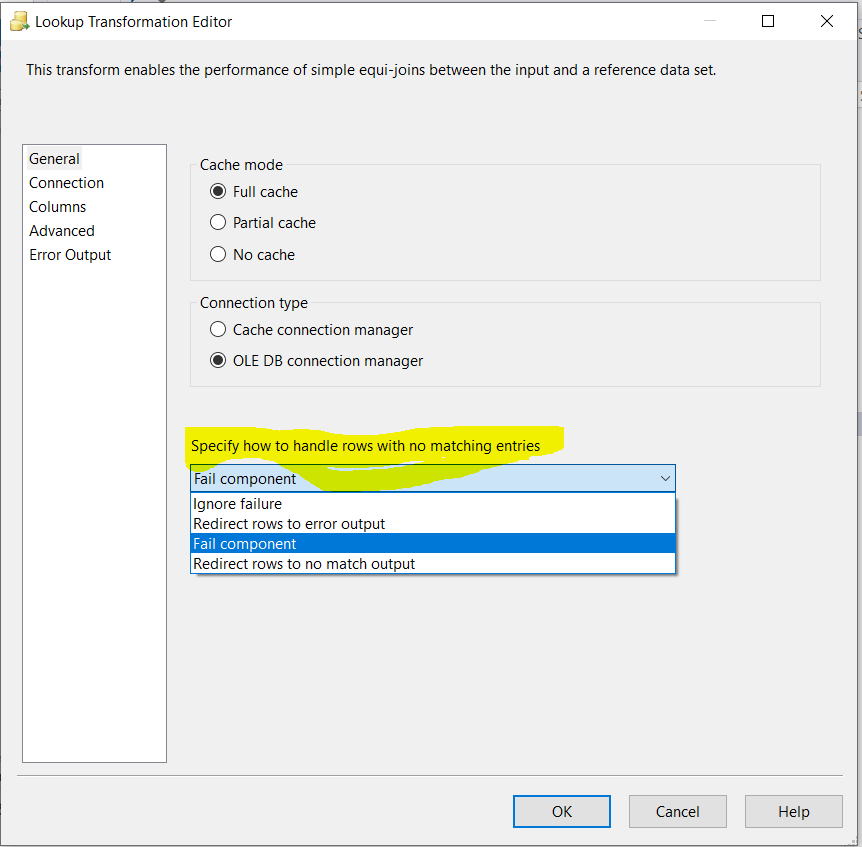
When we are using full cache mode in the lookup transformation, we have two different connection types. Cache connection manager and OLE DB connection manager.

The Cache Connection Manager (CCM) is a separate component that is essential when creating advanced Lookup operations. The CCM enables you to populate the Lookup cache from an arbitrary source; for instance, you can load the cache from a relational query, an Excel file, a text file, or a Web service. You can also use the CCM to persist the Lookup cache across iterations of a looping operation. You can still use the Lookup Transformation without explicitly using the CCM, but you would then lose the resource and performance gains in doing so.

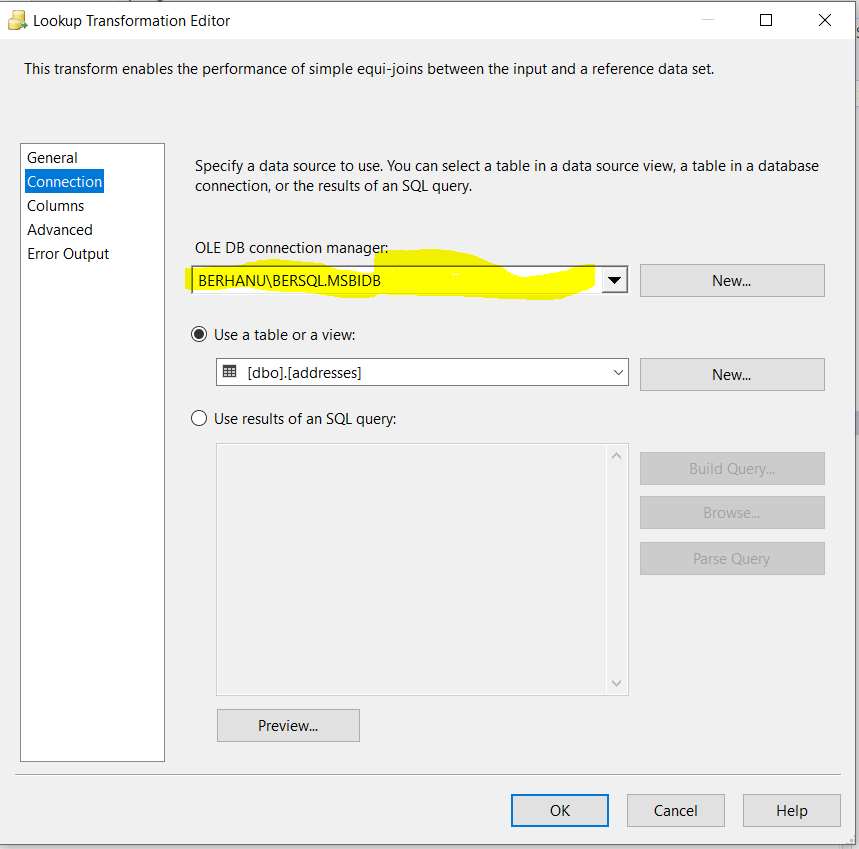


In this case we will use full cache transformation and OLE DB connections because both tables are residing in the relational database and they are connected via id.

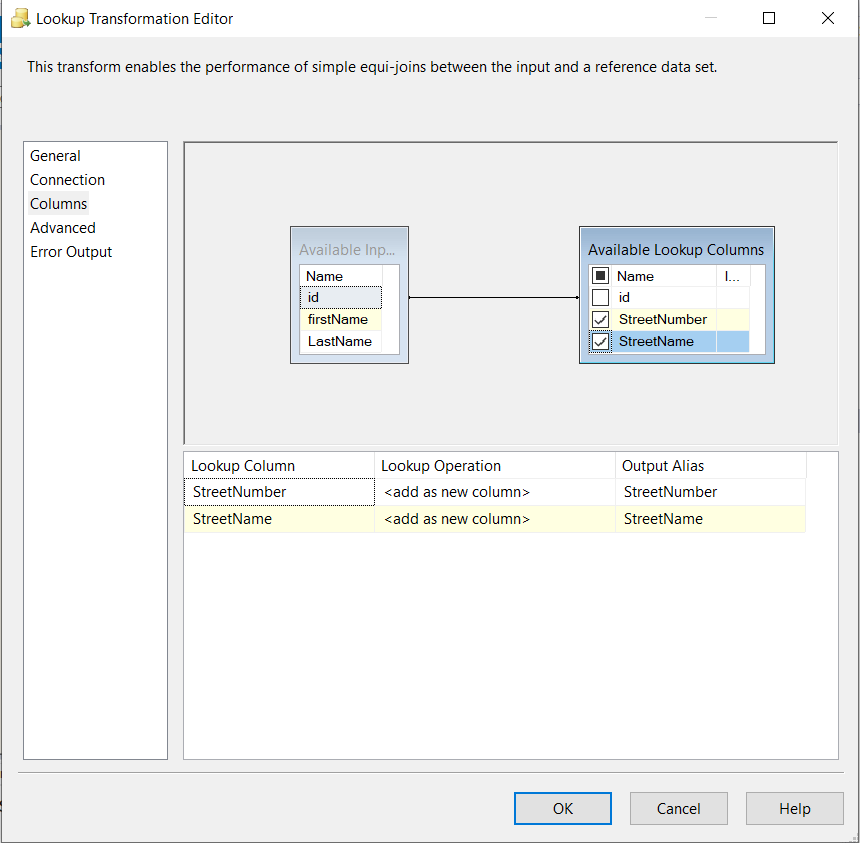
Then you can specify how to handle rows with no matching entries. In the lookup transformation we have four different choices as depicted in the following figure. This time let us do fail component.



Then in the next step we have to open up the connection and provide the connection manager and pick the second table that we want to join. (we can also use a query to build the inline table as a second table)

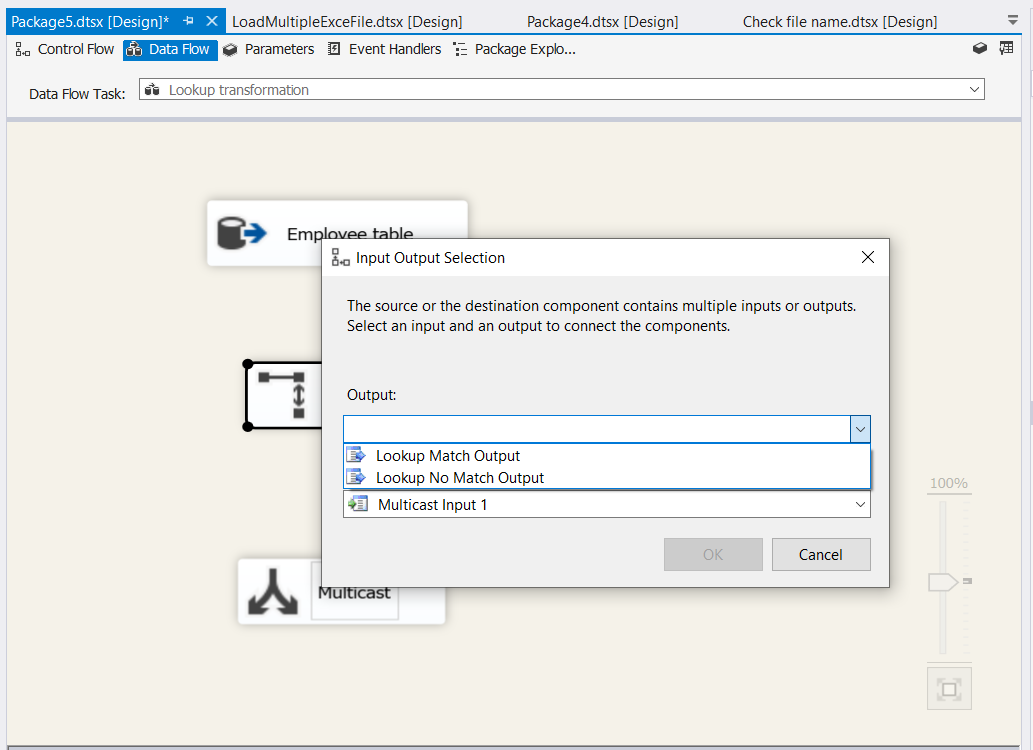


In the next step provide the column that the two tables match. So go to the column. In the column window you will see available input and available lookup columns. Then connect the column that connects the two tables. This time Id is a column that exists in both tables and connect it as shown in the following picture.

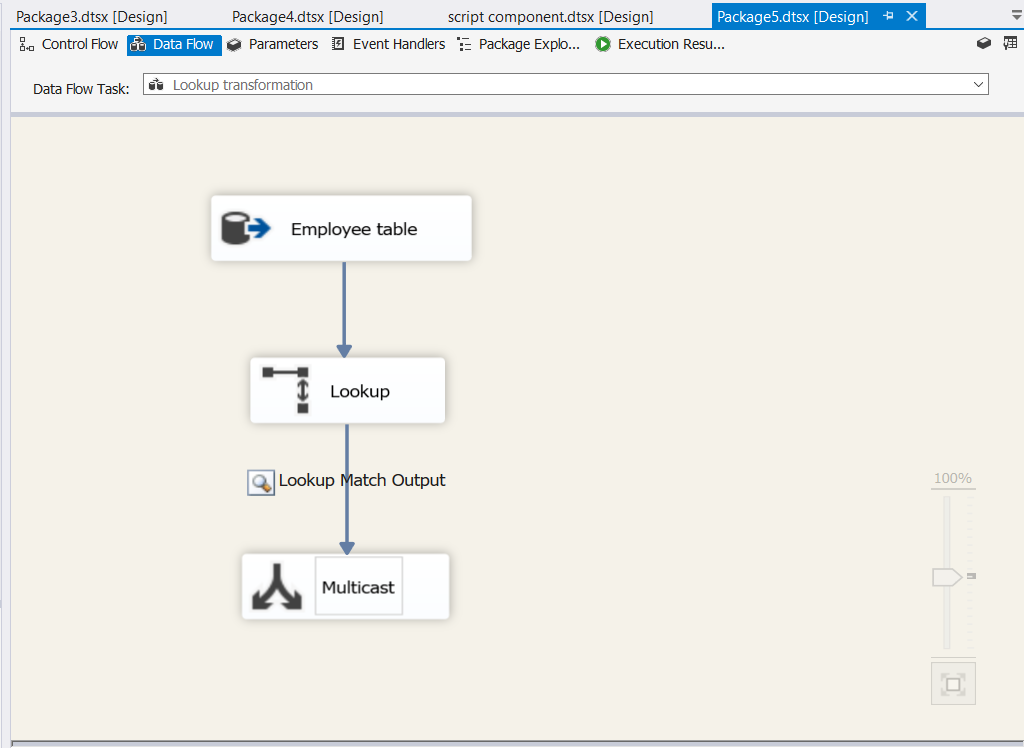


Once we connect the column then we can pick the columns we want to add on the first table. This time we can pick street number and street name.

To test the result, drag and drop multicast transformation then connect lookup transformation with multicast transformation and pick lookup match output as shown in the following picture.



Then on the connection string right click and enable the data viewer as follows.



Then execute the task and you will see the join result (full employee table and address)

