



Working with Object Explorer

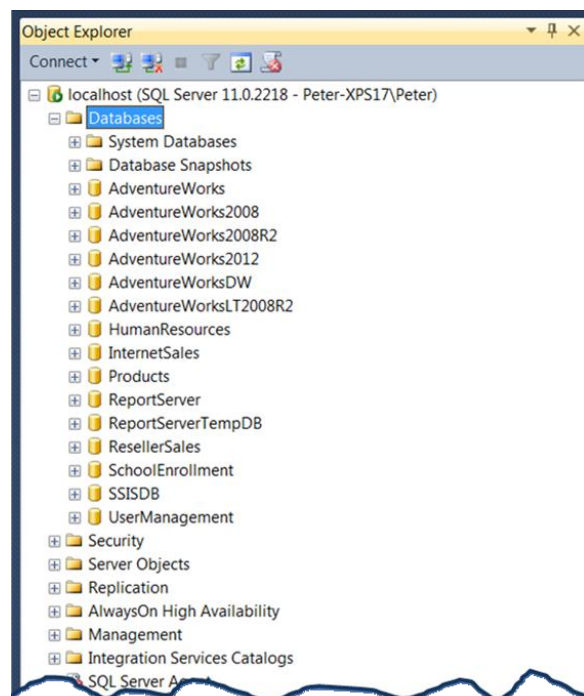
Demo Overview

- A. Explore a Database
- B. Explore Schemas
- C. Explore Constraints
 - a. Foreign Key Constraints
 - b. Unique Constraints

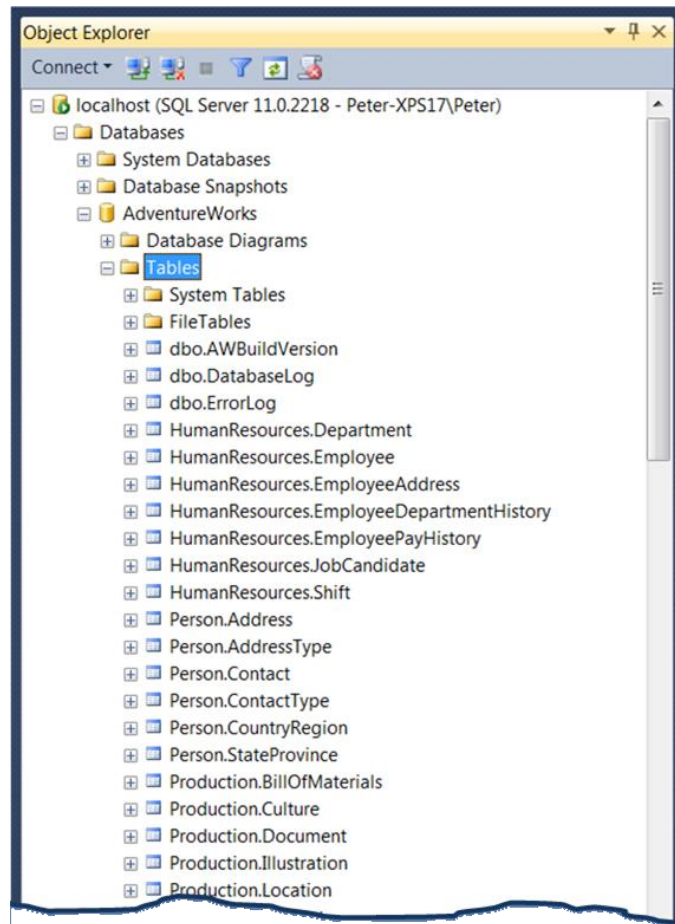
A. Explore a Database

In Demo 01, you saw how to connect to a server. The contents of that server is shown in Object Explorer as a hierarchy of folders that can be expanded and collapsed at several levels. In this course, we will be working only with the Databases folder.

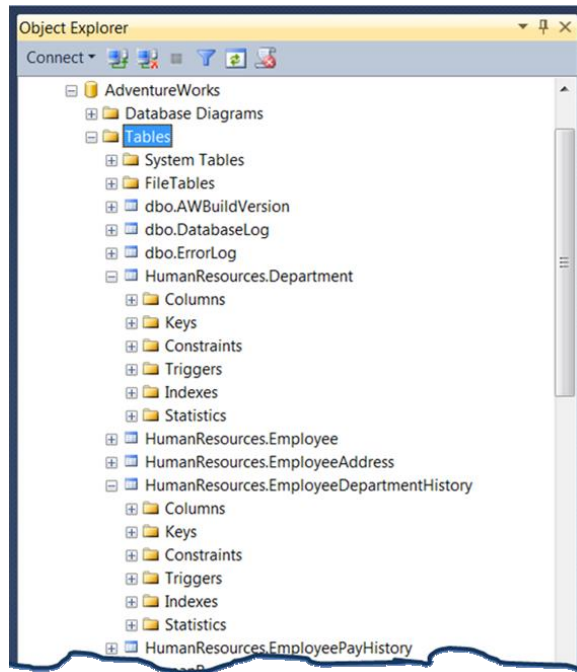
1. If necessary, start SQL Server and connect to localhost as you did in Demo 1.
2. Click the expand button  just to the left of the **Databases** folder and notice that the button changes to a collapse button . You can also expand a folder by double-clicking on it. Once the folder has been expanded, notice the list of databases (the databases on your screen may be different from the one shown here) in addition to a folder called **System Databases** (system databases are beyond the scope of this course; but, in general they are databases used internally by SQL Server) and another folder called **Database Snapshots** (also beyond the scope of this course, database snapshots are copies of a database created at certain points in time).






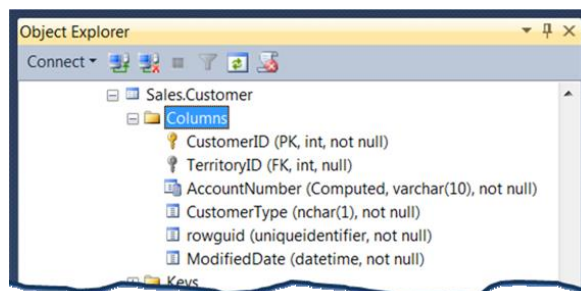
3. Expand the **AdventureWorks** database and then the **Tables** folder. Notice the list of tables. Every table has a two-part name in which the first part is the schema to which the table belongs, and the second part is the name of that table. We will explore schemas shortly. Notice also the expand buttons next to each table; tables can be further expanded.



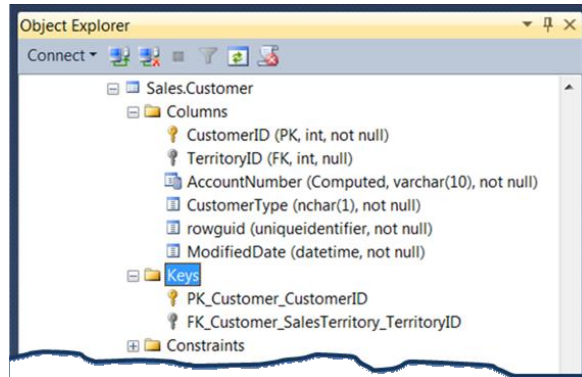
4. In the **AdventureWorks** database's **Tables** folder, expand a few tables and notice that they all have the same six folders (**Columns**, **Keys**, **Constraints**, **Triggers**, **Indexes**, and **Statistics**). Every table stores its definition in those six folders.





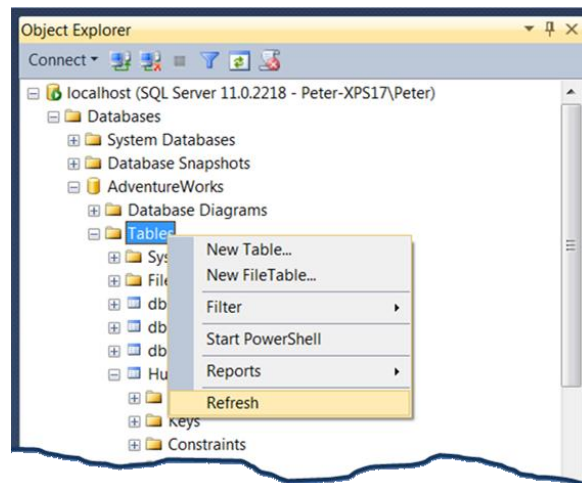
5. Expand the **Sales.Customer** table and then its **Columns** folder and notice the columns listed. Observe the symbols for primary key , foreign key , and computed column . Notice also the data types (in this case **int**, **varchar**, **nchar**, **uniqueidentifier**, and **datetime**) and whether the column accepts nulls (**null**) or not (**not null**). Columns that accept nulls are optional; no data are required in that column for a row to exist. Columns that do not accept nulls are mandatory; without a value in that column, a row cannot be created or modified.



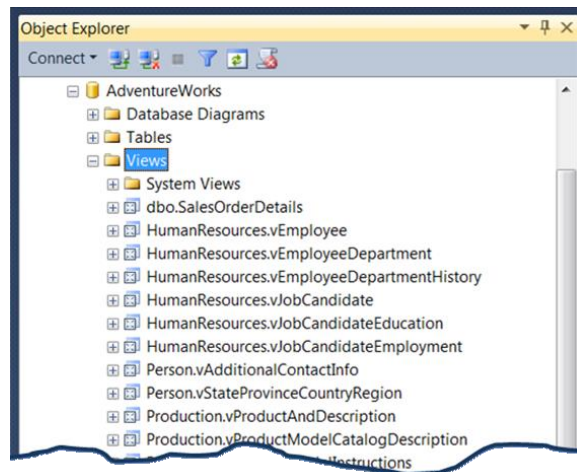
- Expand the **Keys** folder. You can see the two keys (primary and foreign) defined here. Notice that the names are not the same as the column names! These are names of constraints. A key is a type of constraint in the table and has its own name. We will explore this in more detail soon.



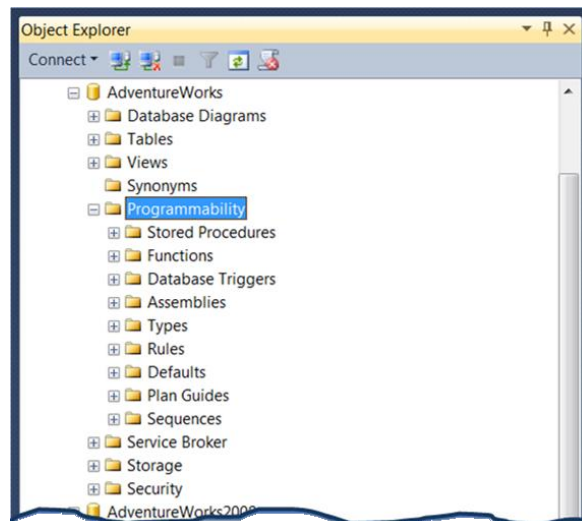
- Collapse all the tables that you expanded. You can do this individually by clicking the collapse button  next to each of the expanded tables (or double-clicking on each table), or you can collapse all expanded tables at once by selecting the **Tables** folder and clicking the **Refresh** button  on the **Object Explorer** toolbar. You can also refresh a folder by right-clicking on it and selecting **Refresh**. (This works for any folder.)



8. Collapse the **Tables** folder and then expand the **Views** folder. Views are query definitions that have been stored as objects in the database. We will be examining views later in this course. Notice that, like tables, views can also be expanded further.



9. Collapse the **Views** folder and expand the **Programmability** folder. The Programmability folder contains objects used by developers, such as stored procedures, functions, and others that we will be working with later in this course. Take a moment to explore these as well.



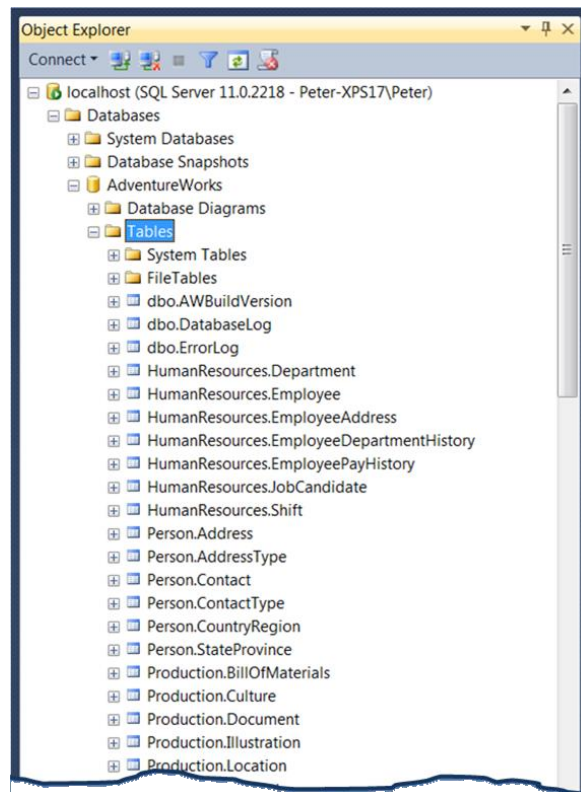
B. Explore Schemas

Schemas are logical groupings of database objects. Every table, view, stored procedure, and others belong to a schema. This can be useful when you want to group database objects by department, project, function, and so on. Schemas are also security boundaries, so all objects in a schema can operate under the same security structure, making it easier to secure a number of

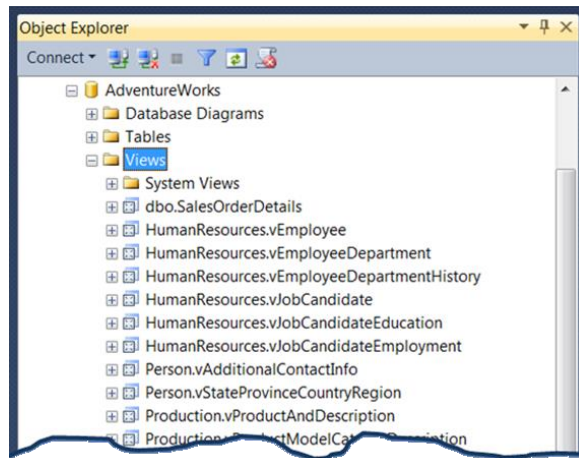
objects in the same way at once.

1. Browse the names of the tables in the AdventureWorks database again. Remember those two-part object names? The first part is the schema. Notice that a number of tables belong to the **dbo** schema, the **HumanResources** schema, the **Production** schema, as well as others (scroll down to see also the **Purchasing** and **Sales** schemas).

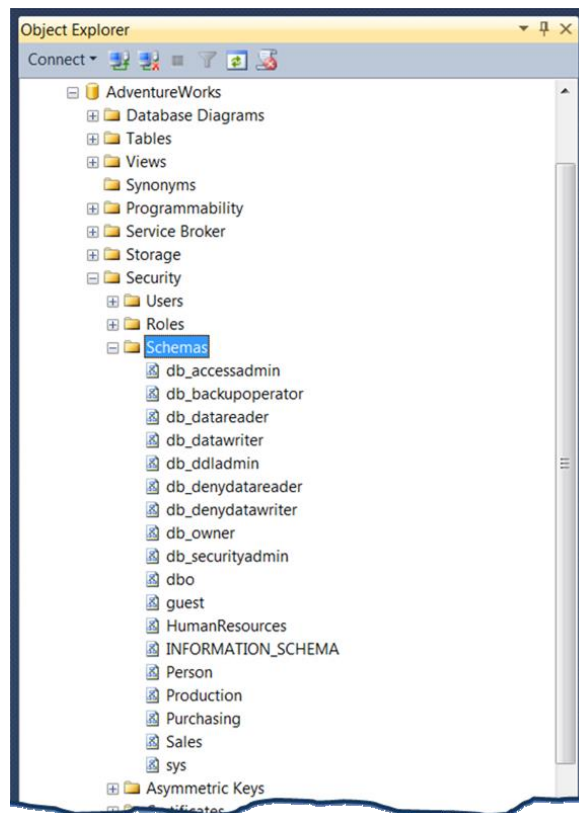
The dbo schema is the default schema in SQL Server. If a schema is not specified for an object when it is created, SQL Server assigns it to the dbo schema. Some users of SQL Server ignore schemas, and so all of their objects end up in the dbo schema.



2. Collapse the **Tables** folder and expand the **Views** folder. Notice that views are also grouped into the same schemas.



3. Take a moment to explore other objects, such as stored procedures, functions, and others to see schemas in their names, too.
4. To see a comprehensive list of the schemas in a database, expand the database's **Security** folder and then the **Schemas** folder. Notice that the same schemas we saw earlier are contained in this list.



C. Explore Table Constraints

In this part of the demo, we will explore four types of constraints: Foreign key (a.k.a. referential integrity), check, default, and unique.

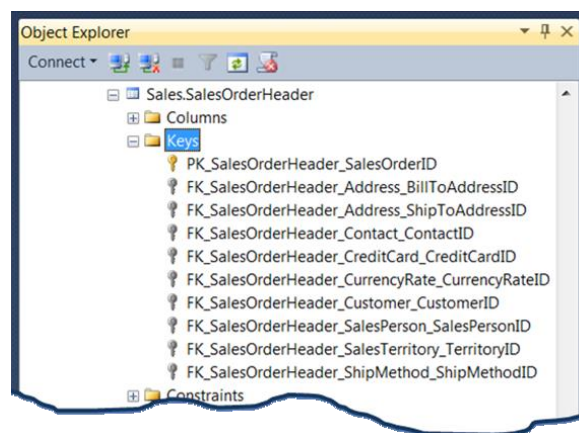
Foreign Key (Referential Integrity) Constraints

A foreign key constraint enforces a relationship between two tables. It ensures that the values in a foreign key column must either exist in the primary key it references or be null. Foreign key constraints are also known as referential integrity constraints.

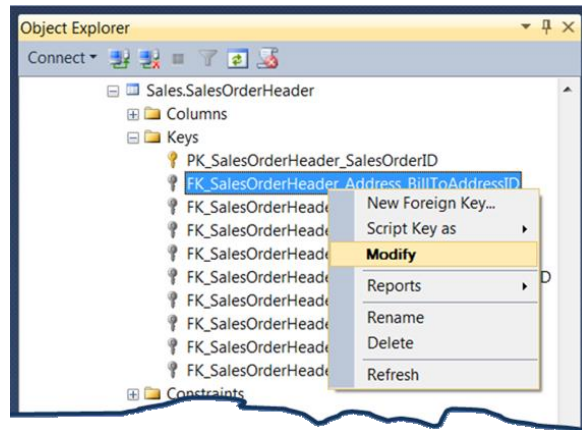
In Module 1, we saw that there were two one-to-many relationships between the Sales.SalesOrderHeader table and the Person.Address table, with the Sales.SalesOrderHeader table on the many side of that relationship. We also found two columns that looked suspiciously like the foreign keys of those relationships in that table; one was the ShipToAddressID and the other one was the BillToAddressID.

We can usually tell the table and primary key that a foreign key references by its name; but, on its own, the name is not conclusive. Let's confirm our suspicions.

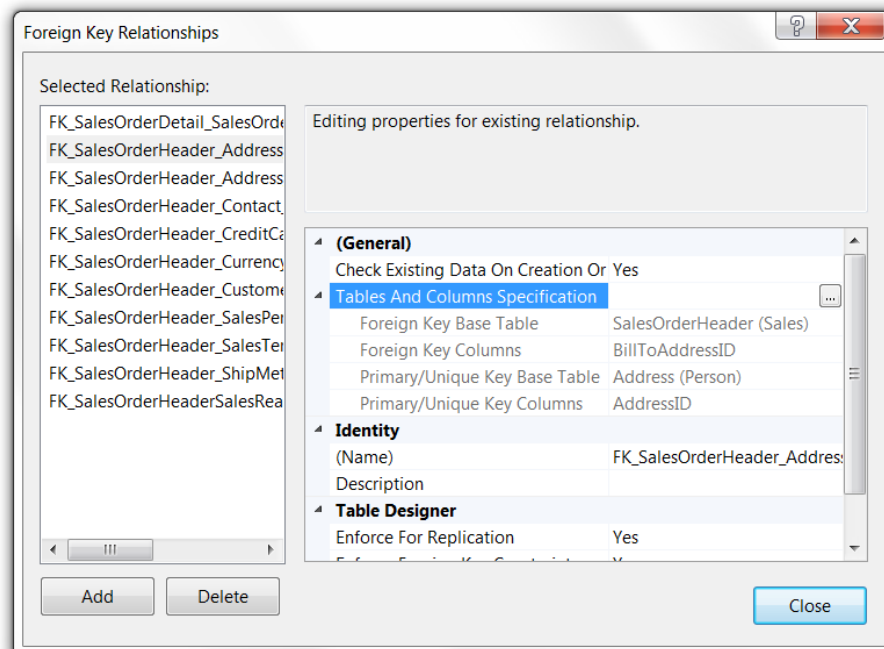
1. Expand the **Keys** folder of the **Sales.SalesOrderHeader** table. Notice the single primary key constraint and all the referential integrity constraints (**FK_**). The first two referential integrity constraints, **FK_SalesOrderHeader_Address_BillToAddressID** and **FK_SalesOrderHeader_Address_ShipToAddressID**, are the ones we want to explore.



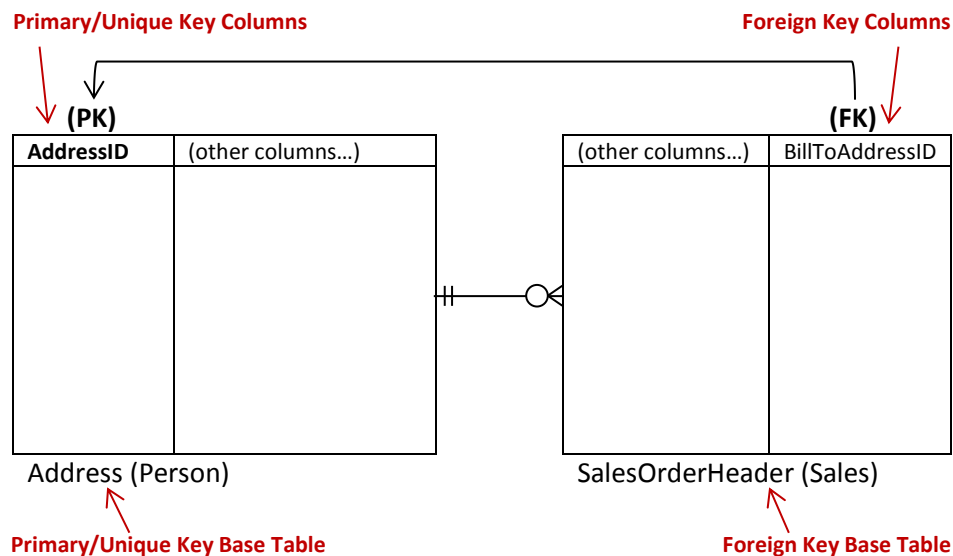
2. To examine the details of a foreign key constraint, either double-click on it or right-click on it and select **Modify**.



3. The table design screen opens and a **Foreign Key Relationships** window appears. Expand **Tables and Columns Specifications** and notice the four fields that appear.




Foreign Key Base Table identifies the table that holds the foreign key; in this case, that's the **Sales.SalesOrderHeader** table. **Foreign Key Columns** identifies the column(s) that make up the foreign key in the foreign key base table; in this case, that's the **BillToAddressID**. **Primary/Unique Key Base Table** identifies the table with the primary key (notice that this can also be a unique key that is not a primary key) that the foreign key references; in this case, that's the **Person.Address** table. **Primary/Unique Key Column** identifies the column(s) in the primary/unique key base table that the foreign key references; in this case, that's the **AddressID**. We can confirm our suspicions!

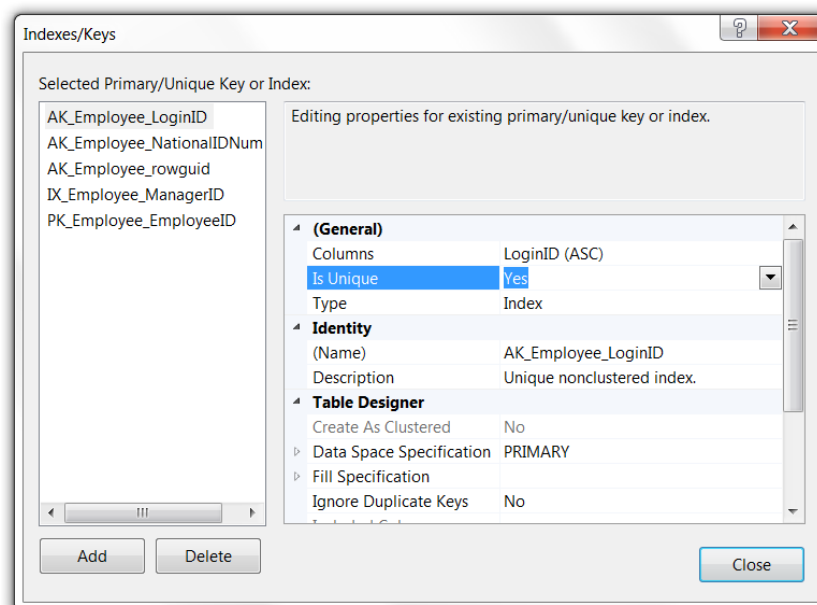


4. In the **Foreign Key Relationships** window, click on the next foreign key constraint, **FK_SalesOrderHeader_Address_ShipToAddressID** and confirm that it also references the **AddressID** primary key in the **Person.Address** table. When you are done, close the **Foreign Key Relationships** window and then the design screen.

Unique constraints

Certain columns need to be unique even if they are not the primary key. Consider the Name column in a Course table. No two courses should ever have the same name. To enforce uniqueness in a column without making it a primary key, create a unique constraint (a.k.a. unique key) on that column.

1. In the design screen of the HumanResources.Employee table, click the **Manage Indexes and Keys** icon  in the toolbar. In the **Indexes/Keys** window, notice the alternate keys (**AK_**) listed on the left side. Notice that they all have a **Yes** in their **Is Unique** properties. Notice also that the primary key constraint is also listed in this window. (The remaining item is a non-unique index (**IX_**) that is useful when searching data, in this case, the **ManagerID**.)



2. In the **Indexes/Keys** window, click **Close**, and then close the design screen.
3. When a unique constraint is created, SQL Server automatically creates an index for it. To see the indexes of the **HumanResources.Employee** table, expand its **Indexes** folder. Notice that there is an index for every constraint we saw in the **Indexes/Key** window.

