SQL 250 Transact-SQL for Developers – Module 02, Demo 02

Working with Object Explorer

Demo Overview

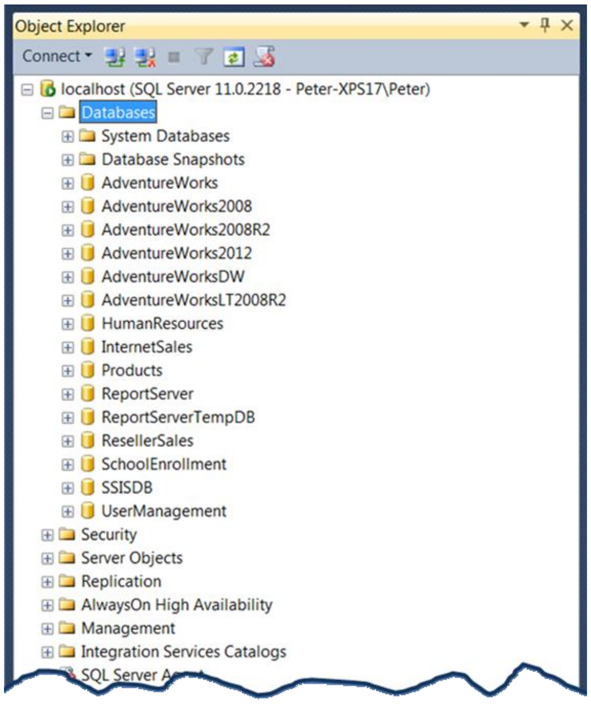
A. Explore a Database
B. Explore Schemas
C. Explore Constraints
    a. Foreign Key Constraints
    b. Check Constraints
    c. Default Constraints
    d. Unique Constraints

A. Explore a Database

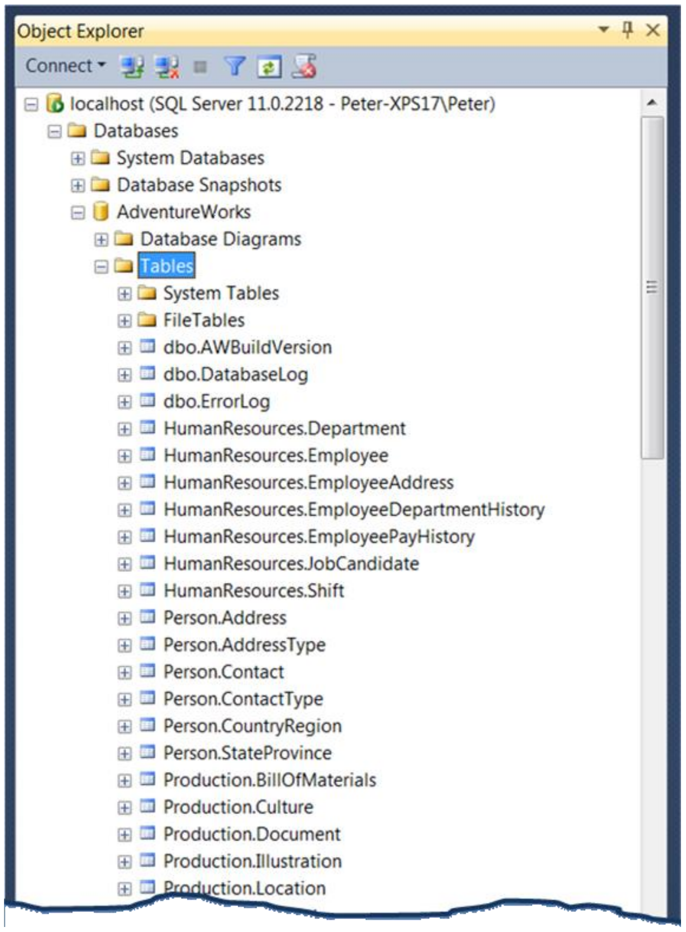
In Demo 01, you saw how to connect to a server. The contents of that server is shown in Object Explorer as a hierarchy of folders that can be expanded and collapsed at several levels. In this course, we will be working only with the Databases folder.

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).

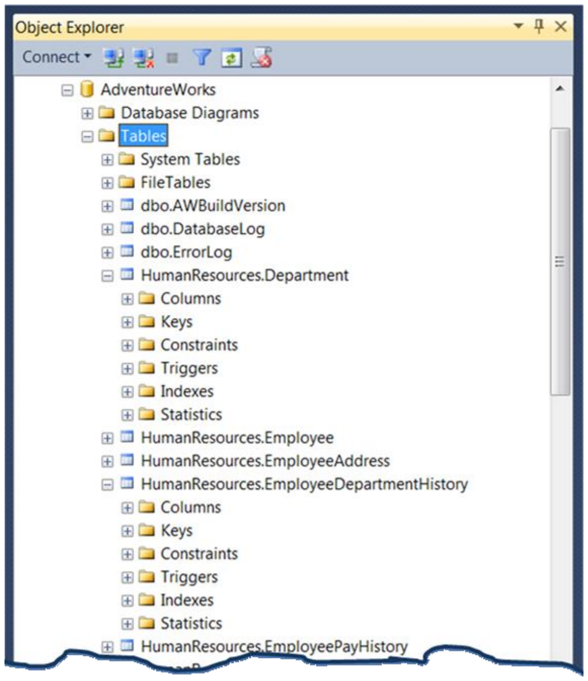


3. Expand the **AdventureWorks** database and then the **Tables** folder. Notice the list of tables. Every table has a two-part name in which the first part is the schema to which the table belongs, and the second part is the name of that table. We will explore schemas shortly. Notice also the expand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



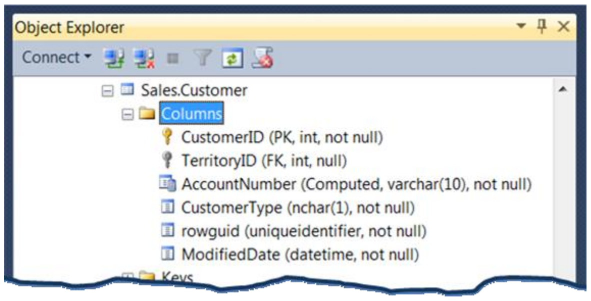
```
localhost (SQL Server 11.0.2218 - Peter-XPS17\Peter)
  Databases
    System Databases
    Database Snapshots
    AdventureWorks
      Database Diagrams
      Tables
        System Tables
        FileTables
        dbo.AWBuildVersion
        dbo.DatabaseLog
        dbo.ErrorLog
        HumanResources.Department
        HumanResources.Employee
        HumanResources.EmployeeAddress
        HumanResources.EmployeeDepartmentHistory
        HumanResources.EmployeePayHistory
        HumanResources.JobCandidate
        HumanResources.Shift
        Person.Address
        Person.AddressType
        Person.Contact
        Person.ContactType
        Person.CountryRegion
        Person.StateProvince
        Production.BillOfMaterials
        Production.Culture
        Production.Document
        Production.Illustration
        Production.Location
```

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.



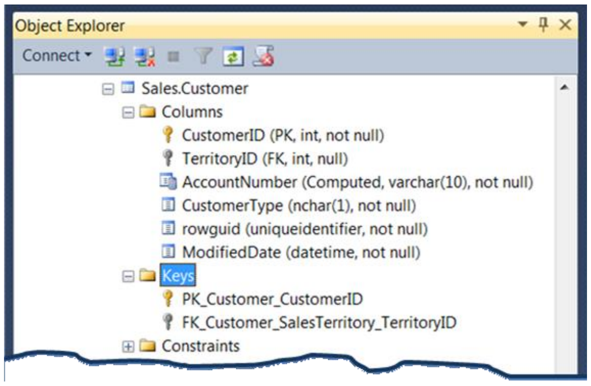
```
Object Explorer
Connect
⊟ AdventureWorks
  ⊞ Database Diagrams
  ⊟ Tables
    ⊞ System Tables
    ⊞ FileTables
    ⊞ dbo.AWBuildVersion
    ⊞ dbo.DatabaseLog
    ⊞ dbo.ErrorLog
    ⊟ HumanResources.Department
      ⊞ Columns
      ⊞ Keys
      ⊞ Constraints
      ⊞ Triggers
      ⊞ Indexes
      ⊞ Statistics
    ⊞ HumanResources.Employee
    ⊞ HumanResources.EmployeeAddress
    ⊟ HumanResources.EmployeeDepartmentHistory
      ⊞ Columns
      ⊞ Keys
      ⊞ Constraints
      ⊞ Triggers
      ⊞ Indexes
      ⊞ Statistics
    ⊞ HumanResources.EmployeePayHistory
```

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.



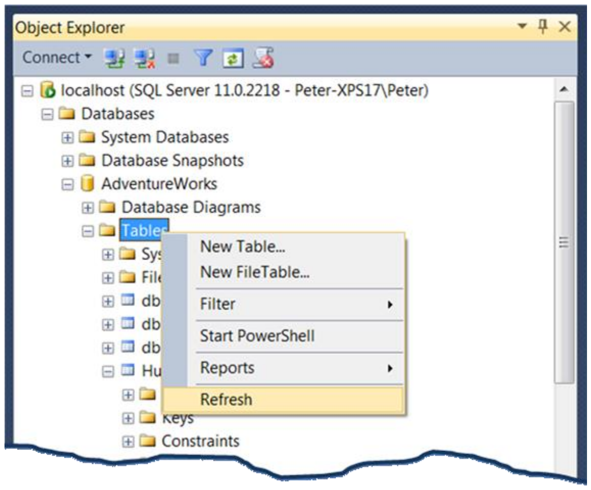
```
Object Explorer
Connect
⊟ Sales.Customer
  ⊟ Columns
      CustomerID (PK, int, not null)
      TerritoryID (FK, int, null)
      AccountNumber (Computed, varchar(10), not null)
      CustomerType (nchar(1), not null)
      rowguid (uniqueidentifier, not null)
      ModifiedDate (datetime, not null)
  ⊞ Keys
```

6. 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.

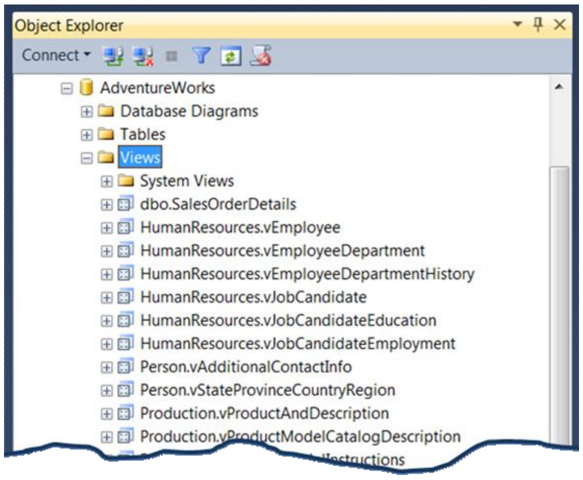


Sales.Customer
Columns
CustomerID (PK, int, not null)
TerritoryID (FK, int, null)
AccountNumber (Computed, varchar(10), not null)
CustomerType (nchar(1), not null)
rowguid (uniqueidentifier, not null)
ModifiedDate (datetime, not null)
Keys
PK_Customer_CustomerID
FK_Customer_SalesTerritory_TerritoryID
Constraints

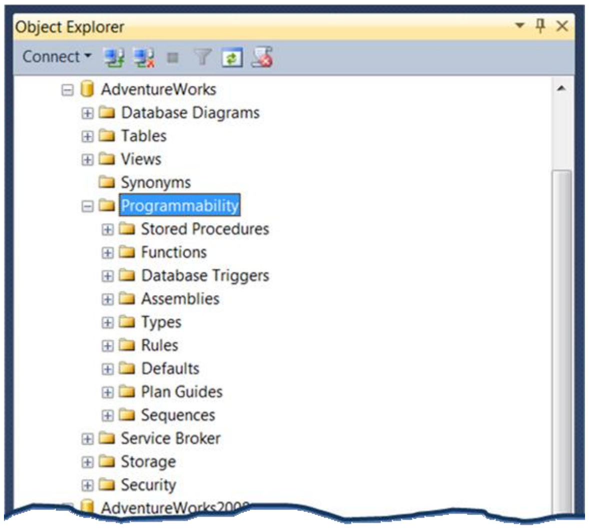
7. Collapse all the tables that you expanded. You can do this individually by clicking the collapse button ⊟ next to each of the expanded tables (or double-clicking on each table), or you can collapse all expanded tables at once by selecting the **Tables** folder and clicking the **Refresh** button on 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.


Object Explorer
Connect
AdventureWorks
Database Diagrams
Tables
Views
System Views
dbo.SalesOrderDetails
HumanResources.vEmployee
HumanResources.vEmployeeDepartment
HumanResources.vEmployeeDepartmentHistory
HumanResources.vJobCandidate
HumanResources.vJobCandidateEducation
HumanResources.vJobCandidateEmployment
Person.vAdditionalContactInfo
Person.vStateProvinceCountryRegion
Production.vProductAndDescription
Production.vProductModelCatalogDescription

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.

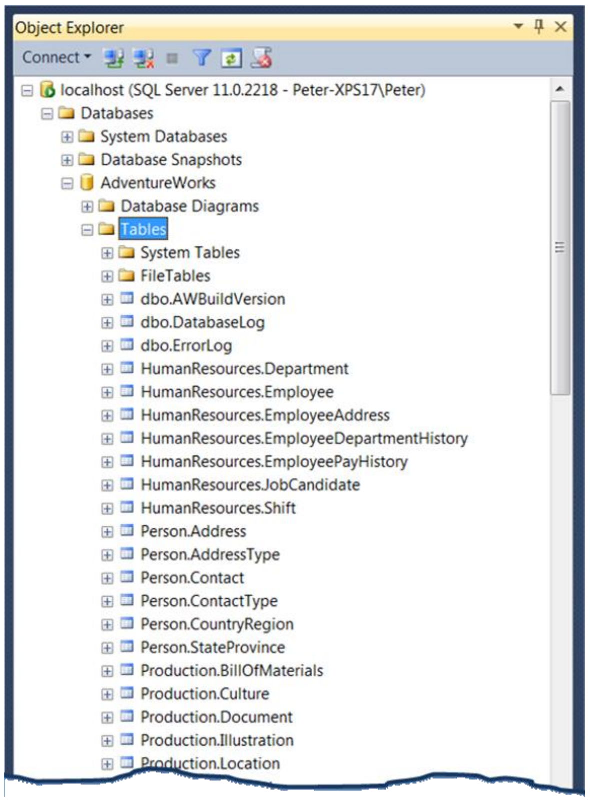


B. Explore Schemas

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.

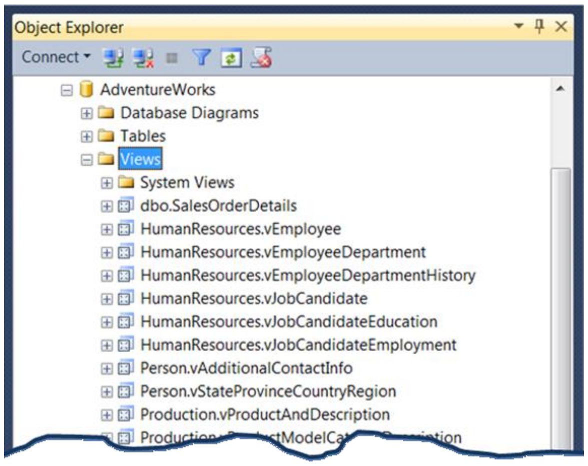
1. Browse the names of the tables in the AdventureWorks database again. Remember those two-part object names? The first part is the schema. Notice that a number of tables belong to the **dbo** schema, the **HumanResources** schema, the **Production** schema, as well as others (scroll down to see also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



```
localhost (SQL Server 11.0.2218 - Peter-XPS17\Peter)
  Databases
    System Databases
    Database Snapshots
    AdventureWorks
      Database Diagrams
      Tables
        System Tables
        FileTables
        dbo.AWBuildVersion
        dbo.DatabaseLog
        dbo.ErrorLog
        HumanResources.Department
        HumanResources.Employee
        HumanResources.EmployeeAddress
        HumanResources.EmployeeDepartmentHistory
        HumanResources.EmployeePayHistory
        HumanResources.JobCandidate
        HumanResources.Shift
        Person.Address
        Person.AddressType
        Person.Contact
        Person.ContactType
        Person.CountryRegion
        Person.StateProvince
        Production.BillOfMaterials
        Production.Culture
        Production.Document
        Production.Illustration
        Production.Location
```

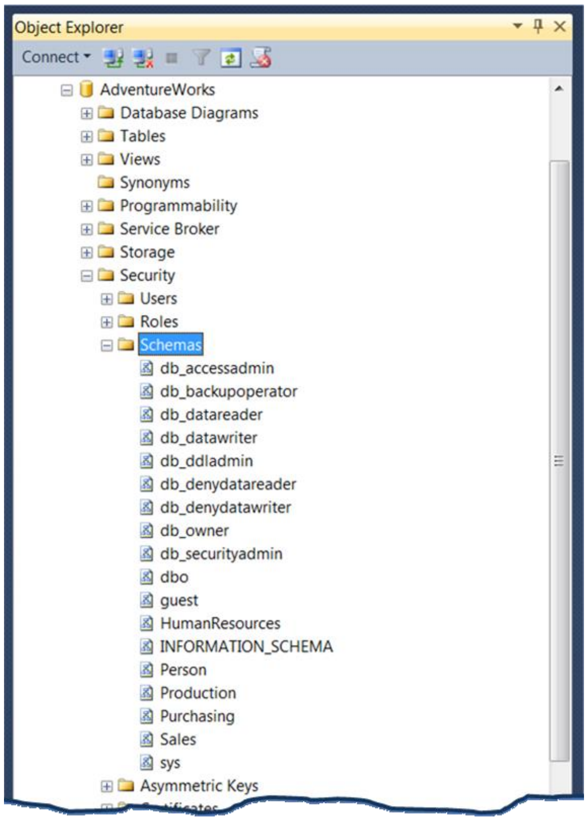
2. Collapse the **Tables** folder and expand the **Views** folder. Notice that views are also grouped into the same schemas.



```
AdventureWorks
  Database Diagrams
  Tables
  Views
    System Views
    dbo.SalesOrderDetails
    HumanResources.vEmployee
    HumanResources.vEmployeeDepartment
    HumanResources.vEmployeeDepartmentHistory
    HumanResources.vJobCandidate
    HumanResources.vJobCandidateEducation
    HumanResources.vJobCandidateEmployment
    Person.vAdditionalContactInfo
    Person.vStateProvinceCountryRegion
    Production.vProductAndDescription
```

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.



```
AdventureWorks
  Database Diagrams
  Tables
  Views
  Synonyms
  Programmability
  Service Broker
  Storage
  Security
    Users
    Roles
    Schemas
      db_accessadmin
      db_backupoperator
      db_datareader
      db_datawriter
      db_ddladmin
      db_denydatareader
      db_denydatawriter
      db_owner
      db_securityadmin
      dbo
      guest
      HumanResources
      INFORMATION_SCHEMA
      Person
      Production
      Purchasing
      Sales
      sys
    Asymmetric Keys
```

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.

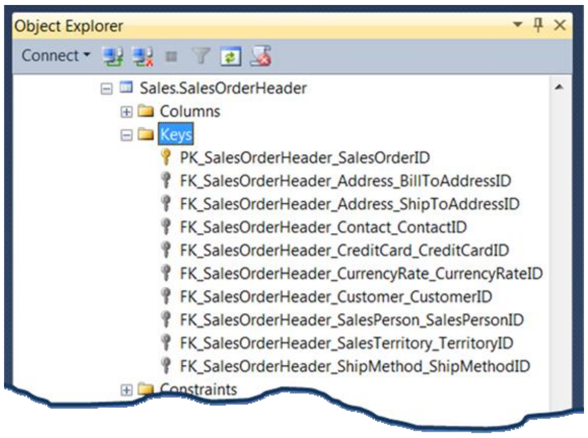
Foreign Key (Referential Integrity) Constraints

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.

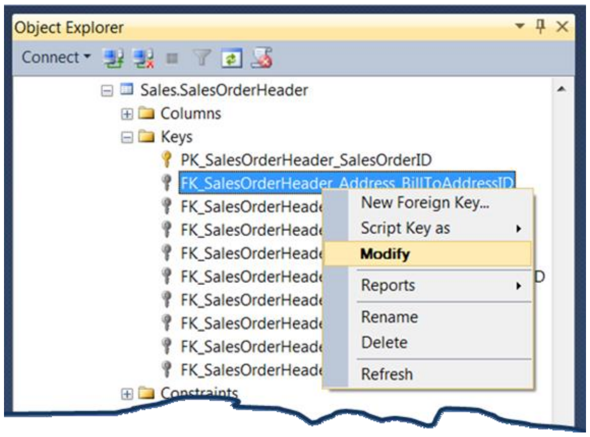
In Module 1, we saw that there were two one-to-many relationships between the Sales.SalesOrderHeader table and the Person.Address table, with the Sales.SalesOrderHeader table on the many side of that relationship. We also found two columns that looked suspiciously like the foreign keys of those relationships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

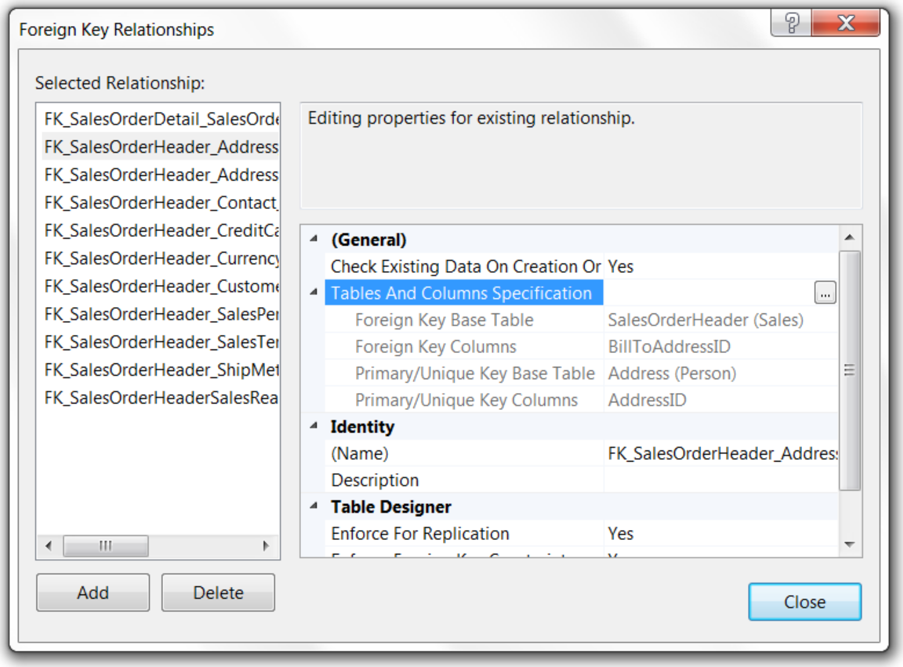
1. Expand the **Keys** folder of the **Sales.SalesOrderHeader** table. Notice the single primary key constraint and all the referential integrity constraints (**FK_**). The first two referential integrity constraints, **FK_SalesOrderHeader_Address_BillToAddressID** and **FK_SalesOrderHeader_Address_ShipToAddressID**,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


Object Explorer
Connect
Sales.SalesOrderHeader
Columns
Keys
PK_SalesOrderHeader_SalesOrderID
FK_SalesOrderHeader_Address_BillToAddressID
FK_SalesOrderHeader_Address_ShipToAddressID
FK_SalesOrderHeader_Contact_ContactID
FK_SalesOrderHeader_CreditCard_CreditCardID
FK_SalesOrderHeader_CurrencyRate_CurrencyRateID
FK_SalesOrderHeader_Customer_CustomerID
FK_SalesOrderHeader_SalesPerson_SalesPersonID
FK_SalesOrderHeader_SalesTerritory_TerritoryID
FK_SalesOrderHeader_ShipMethod_ShipMethodID
Constraints

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



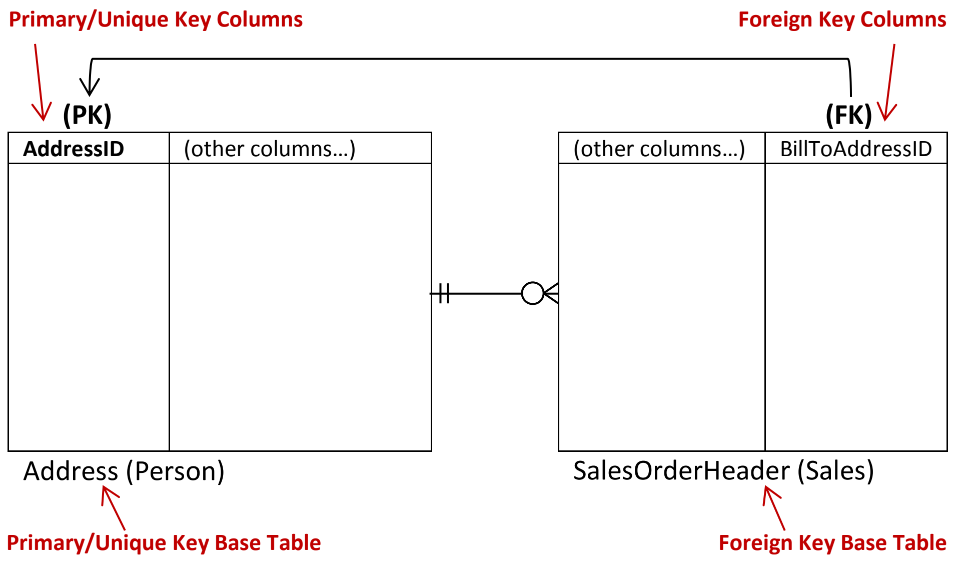
3. The table design screen opens and a **Foreign Key Relationships** window appears. Expand **Tables and Columns Specifications** and notice the four fields that appear.


Foreign Key Relationships
Selected Relationship:
FK_SalesOrderDetail_SalesOrde
FK_SalesOrderHeader_Address
FK_SalesOrderHeader_Address
FK_SalesOrderHeader_Contact_
FK_SalesOrderHeader_CreditCa
FK_SalesOrderHeader_Currency
FK_SalesOrderHeader_Custome
FK_SalesOrderHeader_SalesPer
FK_SalesOrderHeader_SalesTer
FK_SalesOrderHeader_ShipMet
FK_SalesOrderHeaderSalesRea
Add
Delete
Close

Editing properties for existing relationship.

(General)	
Check Existing Data On Creation Or	Yes
Tables And Columns Specification	
Foreign Key Base Table	SalesOrderHeader (Sales)
Foreign Key Columns	BillToAddressID
Primary/Unique Key Base Table	Address (Person)
Primary/Unique Key Columns	AddressID
Identity	
(Name)	FK_SalesOrderHeader_Addres
Description	
Table Designer	
Enforce For Replication	Yes

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!

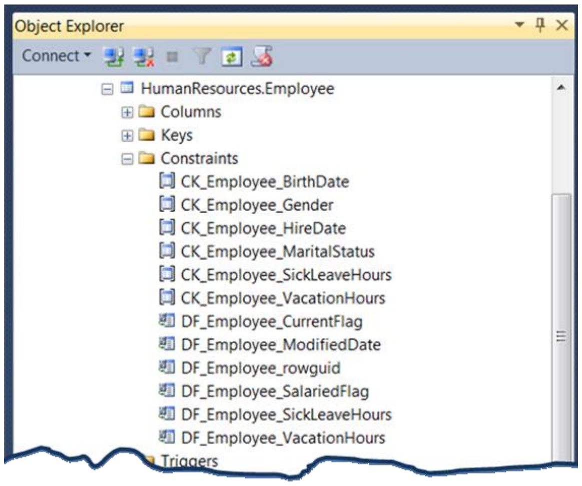


4. In the **Foreign Key Relationships** window, click on the next foreign key constraint, **FK_SalesOrderHeader_Address_ShipToAddressID** and confirm that it also references the AddressID primary key in the Person.Address table. When you are done, close the **Foreign Key Relationships** window and then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

Check Constraints

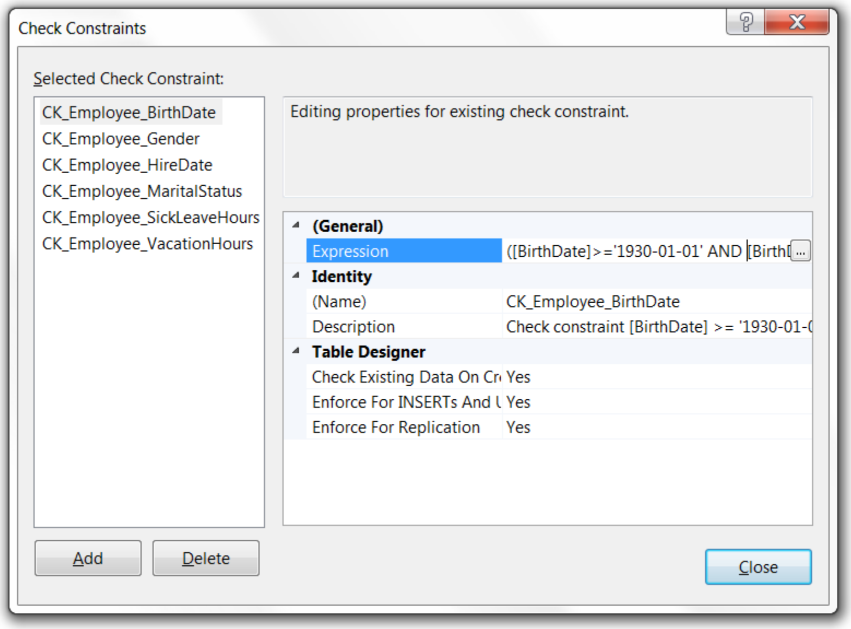
A check constraint is a rule that must be passed before data can be entered into a column. (Check constraints can also be created as table check constraints where there is a dependency between columns, such as when the ReviewDate must be greater than the HireDate).

1. Expand the **HumanResources.Employee** table and then its **Constraints** folder and notice that there are two types of constraints in this folder: Check constraints (**CK_**) and default constraints (**DF_**).



HumanResources.Employee
Columns
Keys
Constraints
CK_Employee_BirthDate
CK_Employee_Gender
CK_Employee_HireDate
CK_Employee_MaritalStatus
CK_Employee_SickLeaveHours
CK_Employee_VacationHours
DF_Employee_CurrentFlag
DF_Employee_ModifiedDate
DF_Employee_rowguid
DF_Employee_SalariedFlag
DF_Employee_SickLeaveHours
DF_Employee_VacationHours
Triggers

2. To see the rule behind a check constraint, first double-click on it or right-click on it and select **Modify**. Then select the Expression property in the **Check Constraints** window.

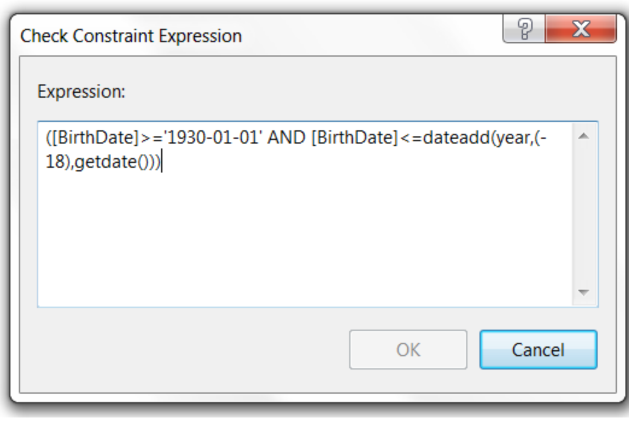


CK_Employee_BirthDate
CK_Employee_Gender
CK_Employee_HireDate
CK_Employee_MaritalStatus
CK_Employee_SickLeaveHours
CK_Employee_VacationHours

Editing properties for existing check constraint.

(General)	
Expression	([BirthDate]>='1930-01-01' AND [Birth
Identity	
(Name)	CK_Employee_BirthDate
Description	Check constraint [BirthDate] >= '1930-01-(
Table Designer	
Check Existing Data On Cr	Yes
Enforce For INSERTs And U	Yes
Enforce For Replication	Yes

3. Next, in the **Check Constraints** window, click the ellipses button ... on the right side of the **Expression** property. Notice in the **Check Constraint Expression** window that valid birthdates must be after 1929 and no one under 18 can be an employee.



```
([BirthDate]>='1930-01-01' AND [BirthDate]<=dateadd(year,(-
18),getdate()))
```

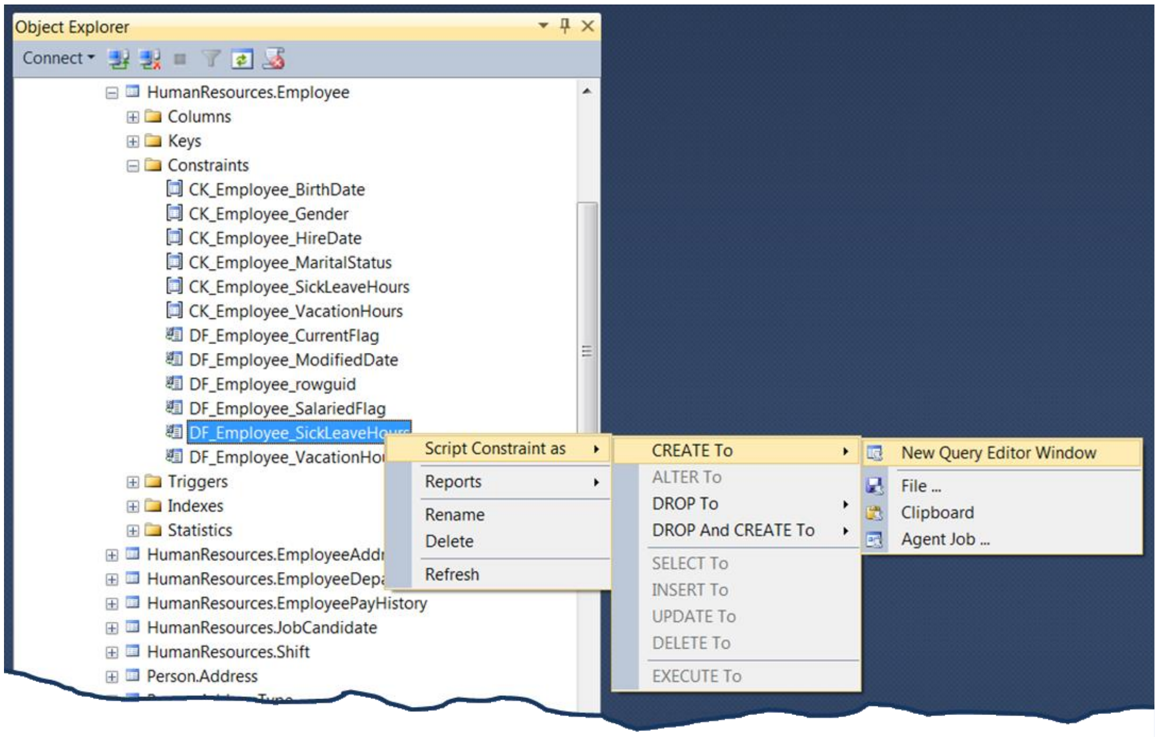
4. Click **Cancel** in the **Check Constraint Expression** window and then **Close** in the **Check Constraints** window.

Default Constraints

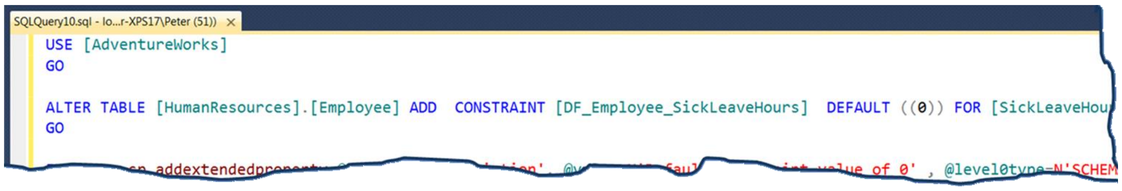
A default constraint puts a value into a column when a row is *created* if no other value is provided.

1. In the **HumanResources.Employee** table's **Constraints** folder, double-click on any of the default constraints and notice that nothing happens. Default constraints cannot be viewed in the same way as check constraints.

2. One way to see a default constraint is to script it. Right-click on the **DF_Employee_SickLeaveHours** default constraint, then select **Script Constraint as**, then **CREATE to**, and finally **New Query Editor Window**.



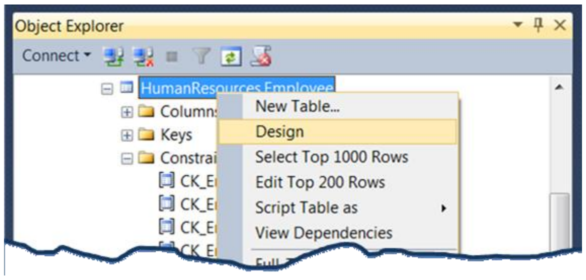
3. In the query window that appears to the right of the **Object Explorer** panel, observe the **ALTER TABLE** statement. Just after the **DEFAULT** clause, you will see **((0))**. Just after that, you will see a **FOR** clause followed by the name of the column to which the constraint applies. The statement says that, if no value is supplied for the SickLeaveHours column of the Employee table, SQL Server will supply the value 0 for that column. Remember that this only happens when a row is created!



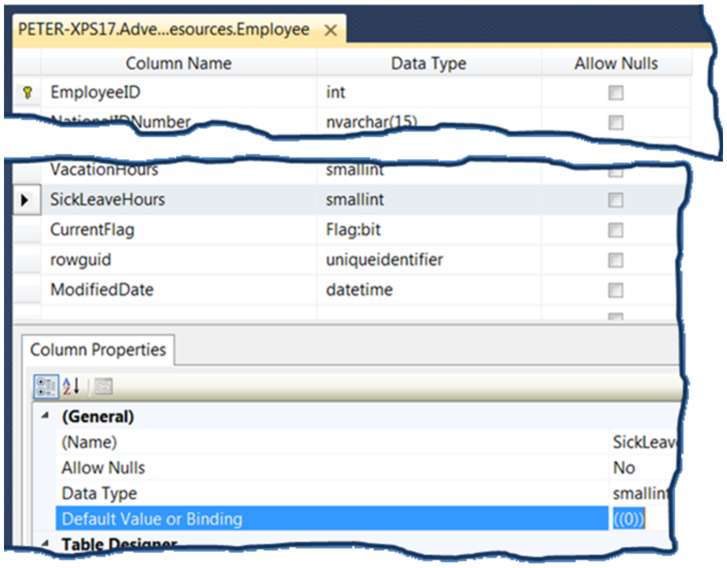
```
USE [AdventureWorks]
GO

ALTER TABLE [HumanResources].[Employee] ADD  CONSTRAINT [DF_Employee_SickLeaveHours]  DEFAULT ((0)) FOR [SickLeaveHou
GO
```

4. Another way to see a default constraint is in the design screen of a table. Right-click on the **HumanResources.Employee** table and select **Design.**



5. Under **Column Name**, select **SickLeaveHours** and notice in the **Column Properties** panel in the bottom portion of the screen the **Default Value or Binding** property. It also shows the same default value of **0**.



Column Name	Data Type	Allow Nulls
EmployeeID	int	☐
NationalIDNumber	nvarchar(15)	☐
VacationHours	smallint	☐
SickLeaveHours	smallint	☐
CurrentFlag	Flag:bit	☐
rowguid	uniqueidentifier	☐
ModifiedDate	datetime	☐

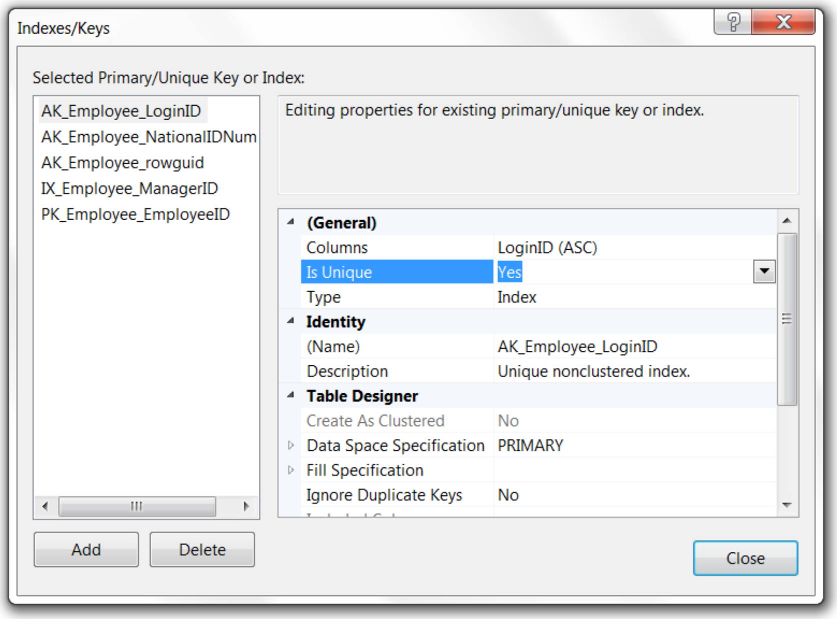
(General)	
(Name)	SickLeav
Allow Nulls	No
Data Type	smallin
Default Value or Binding	((0))
Table Designer	

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 there **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**.)


Indexes/Keys

Selected Primary/Unique Key or Index:

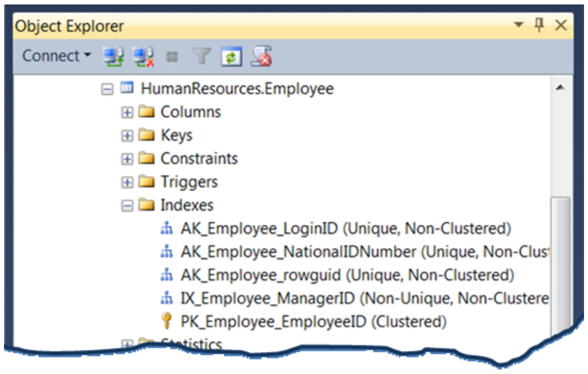
AK_Employee_LoginID
AK_Employee_NationalIDNum
AK_Employee_rowguid
IX_Employee_ManagerID
PK_Employee_EmployeeID

Editing properties for existing primary/unique key or index.

(General)	
Columns	LoginID (ASC)
Is Unique	Yes
Type	Index
Identity	
(Name)	AK_Employee_LoginID
Description	Unique nonclustered index.
Table Designer	
Create As Clustered	No
Data Space Specification	PRIMARY
Fill Specification	
Ignore Duplicate Keys	No

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.



HumanResources.Employee
Columns
Keys
Constraints
Triggers
Indexes
AK_Employee_LoginID (Unique, Non-Clustered)
AK_Employee_NationalIDNumber (Unique, Non-Clus
AK_Employee_rowguid (Unique, Non-Clustered)
IX_Employee_ManagerID (Non-Unique, Non-Clustere
PK_Employee_EmployeeID (Clustered)
Statistics