CS108: Advanced Database

- Database Programming

Lecture 05:

Creating and Altering Tables

Overview

- Create a database using a script
- Create a table using a script
- Alter and drop objects

Object Names in SQL Server

- There are four levels in the naming for any SQL Server table.
- A full qualified name is as follows:

```
[ServerName.[DatabaseName.[SchemaName.]]]ObjectName
```

- We must provide an object name whenever we are performing an operation on that object, but
- All parts of the name to the left of the object name are optional

Schema Name

- The Schema Name can help us to separate our database objects into logical groups.
- For example

```
SELECT e.NationalIDNumber, p.FirstName, p.LastName, City
FROM HumanResources.Employee e
INNER JOIN
     Person.Person p ON p.BusinessEntityID = e.BusinessEntityID
INNER JOIN
     Person.BusinessEntityAddress a
        ON p.BusinessEntityID = a.BusinessEntityID
INNER JOIN
     Person.Address pa ON pa.AddressID = a.AddressID
```

- This example makes use of four tables spread across two schemas.

Schema Name

- A database schema is a way to logically group objects such as tables, views, stored procedures etc.
- The default schema is dbo (database owner), any objects that a dbo creates within that database shall be listed with a schema of dbo.
- It’s worth pointing out that members of the sysadmin role (including the *sa* login) always alias to the dbo.

The CREATE Statement

- The CREATE DATABASE statement is used to create a database
- The CREATE statement will always look like this:

```
CREATE <object type> <object name>
```

- The most basic syntax for the CREATE DATABASE statement looks like this:

```
CREATE DATABASE <database name>
```

- A fuller syntax of CREATE DATABASE is

```
CREATE DATABASE <database name>
[ON [PRIMARY]
    ([NAME = <'logical file name'>,]
      FILENAME = <'file name'>
      [, SIZE = <size in kilobytes, megabytes, gigabytes, or terabytes>]
      [, MAXSIZE = size in kilobytes, megabytes, gigabytes, or terabytes>]
      [, FILEGROWTH = <kilobytes, megabytes, gigabytes, or
                                                terabytes|percentage>])]
  [LOG ON
    ([NAME = <'logical file name'>,]
      FILENAME = <'file name'>
     [, SIZE = <size in kilobytes, megabytes, gigabytes, or terabytes>]
     [, MAXSIZE = size in kilobytes, megabytes, gigabytes, or terabytes>]
     [, FILEGROWTH = <kilobytes, megabytes, gigabytes, or
                                                terabytes|percentage>])]
  [ CONTAINMENT = OFF|PARTIAL ]
  [ COLLATE <collation name> ]
  [ FOR ATTACH [WITH <service broker>]| FOR ATTACH_REBUILD_LOG|
      WITH DB_CHAINING ON|OFF | TRUSTWORTHY ON|OFF]
  [AS SNAPSHOT OF <source database name>]
  [;]
```

The CREATE DATABASE Statement

- ***ON*** - ON is used in two places:
  - to *define the location of the file* where the data is stored, and
  - to *define the location of the file* where the log is stored
- ***NAME*** - It is only a logical name for the file use in SQL Server. We use this name when we want to resize (expand or shrink) the database and/or file.
- ***FILENAME*** - The physical name on the disk of the actual operating system file in which the data and log will be stored. The extension of the file typically is mdf for database and ldf for log file.

- ***SIZE*** - The size of the database. By default, the size is in megabytes.
- ***MAXSIZE*** - It is the maximum size to which the database can grow.
- ***FILEGROWTH*** - It essentially determines just how fast the database gets to the maximum size. We can provide a value that indicates by how many bytes (in KB, MB, GB, or TB) or percentage at a time want the file to be enlarged.
- ***LOG ON*** - The option allows us to establish that we want our log to go to a specific set of files and where exactly those files are to be located.

Example: Building a Database

- The following is the statement to create a database

```
CREATE DATABASE Accounting
ON
   (NAME = 'Accounting',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\AccountingData.mdf',
    SIZE = 10,
    MAXSIZE = 50,
    FILEGROWTH = 5)
LOG ON
   (NAME = 'AccountingLog',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\AccountingLog.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB);
GO
```

Check the database: sp_helpdb

- ‘sp_helpdb’ stored procedure especially tailored for database structure information

```
EXEC sp_helpdb 'Accounting'
```

- The result sets are:

	name	db_size	owner	dbid	created	status	compatibility_level
1	Accounting	15.00 MB	A212-SLLUO\Administrator	8	Sep 8 2015	Status=ONLINE, Updateability=READ_WRITE, UserAccess=MU...	120

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	Accounting	1	D:\AccountingData.mdf	PRIMARY	10240 KB	51200 KB	5120 KB	data only
2	AccountingLog	2	D:\AccountingLog.ldf	NULL	5120 KB	25600 KB	5120 KB	log only

Create Table

- The CREATE DATABASE statement is used to create a database
- The CREATE DATABASE syntax is:

```
CREATE TABLE [database_name.[owner].]table_name
(<column name> <data type>
[[DEFAULT <constant expression>]
   |[IDENTITY [(seed, increment) [NOT FOR REPLICATION]]]]
   [ROWGUIDCOL]
   [COLLATE <collation name>]
   [NULL|NOT NULL]
   [<column constraints>]
   |[column_name AS computed_column_expression]
   |[<table_constraint>]
   [,...n]
)
[ON {<filegroup>|DEFAULT}]
[TEXTIMAGE_ON {<filegroup>|DEFAULT}]
```

Create Table

- ***Table and Column Names*** - The rules for naming tables and columns are, in general, the same rules that apply to all database objects
- ***Data Types*** - We need to provide a data type immediately following the column name - there is no default data type
- ***DEFAULT*** - This is the value we want to use for any rows that are inserted without a user-supplied value for this particular column.
- ***IDENTITY*** - When we make a column an identity column, SQL Server automatically assigns a sequenced number to this column with every row we insert.

Create Table

- ***NULL/NOT NULL*** - It states whether or not the column in question accepts NULL values. The default is NOT NULL.
- ***Column Constraints*** - They are restrictions and rules that we place on individual columns about the data that can be inserted into that column.
- ***Computed Columns*** - We can also have a column that doesn't have any data of its own, but whose value is derived on the fly from other columns in the table. The specific syntax:

```
<column name> AS <computed column expression>
```

- Example:

```
ExtendedPrice AS Price * Quantity
```

Create Table - Computed Columns

- ***Computed Columns:***

```
<column name> AS <computed column expression>
```

- For the computed columns:
  - Cannot use a subquery, and the values cannot come from a different table.
  - Cannot directly specify the data type of a computed column; it is implicitly of whatever type the expression produces (can use CAST or CONVERT as part of our expression to explicitly impose a type on the result).
  - Could not use a computed column as any part of any key (primary, foreign, or unique).
  - Special steps must be taken if we want to create indexes on computed columns.

Create Table

- Table constraints are quite similar to column constraints, in that they place restrictions on the data that can be inserted into the table.
- Example:

```
USE Accounting;
CREATE TABLE Customers
(
    CustomerNo   INT           IDENTITY NOT NULL,
    CustomerName VARCHAR(30)   NOT NULL,
    Address1     VARCHAR(30)   NOT NULL,
    Address2     VARCHAR(30)   NOT NULL,
    City         VARCHAR(20)   NOT NULL,
    State        CHAR(2)       NOT NULL,
    Zip          VARCHAR(10)   NOT NULL,
    Contact      VARCHAR(25)   NOT NULL,
    Phone        CHAR(15)      NOT NULL,
    FedIDNo      VARCHAR(9)    NOT NULL,
    DateInSystem DATE          NOT NULL
)
```

Create Table (cont.)

- Example:

```
USE Accounting
CREATE TABLE Employees
(
   EmployeeID       INT          IDENTITY   NOT NULL,
   FirstName        VARCHAR(25)             NOT NULL,
   MiddleInitial    CHAR(1)                 NULL,
   LastName         VARCHAR(25)             NOT NULL,
   Title            VARCHAR(25)             NOT NULL,
   SSN              VARCHAR(11)             NOT NULL,
   Salary           MONEY                   NOT NULL,
   PriorSalary      MONEY                   NOT NULL,
   LastRaise AS Salary - PriorSalary,
   HireDate         DATE                    NOT NULL,
   TerminationDate DATE                     NULL,
   ManagerEmpID     INT                     NOT NULL,
   Department       VARCHAR(25)             NOT NULL
)
```

The ALTER Statement

- The ALTER statement change a table/DB rather than re-create it.

```
ALTER DATABASE <database name>
   ADD FILE
        ([NAME = <'logical file name'>,]
         FILENAME = <'file name'>
       [, SIZE = <size in KB, MB, GB or TB>]
       [, MAXSIZE = < size in KB, MB, GB or TB >]
       [, FILEGROWTH = <No of KB, MB, GB or TB |percentage>]) [,...n]
            [ TO FILEGROUP filegroup_name]
[, OFFLINE ]
  |ADD LOG FILE
       ([NAME = <'logical file name'>,]
        FILENAME = <'file name'>
        [, SIZE = < size in KB, MB, GB or TB >]
        [, MAXSIZE = < size in KB, MB, GB or TB >]
        [, FILEGROWTH = <No KB, MB, GB or TB |percentage>])
  |REMOVE FILE <logical file name> [WITH DELETE]
  |ADD FILEGROUP <filegroup name>
  |REMOVE FILEGROUP <filegroup name>
  |MODIFY FILE <filespec>
  |MODIFY NAME = <new dbname>
  |MODIFY FILEGROUP <filegroup name> {<filegroup property>|NAME =
          <new filegroup name>}
  |SET <optionspec> [,...n ][WITH <termination>]
  |COLLATE <collation name>
```

Example: ALTER DATABASE

- We want to expand our DB to 100MB, before expand:

```
EXEC sp_helpdb 'Accounting'
```

	name	db_size	owner	dbid	created	status	compatibility_level
1	Accounting	15.00 MB	A212-SLLUO\Administrator	8	Sep 8 2015	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	120

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	Accounting	1	D:\AccountingData.mdf	PRIMARY	10240 KB	51200 KB	5120 KB	data only
2	AccountingLog	2	D:\AccountingLog.ldf	NULL	5120 KB	25600 KB	5120 KB	log only

- Using ALTER Statement to modify our DB

```
ALTER DATABASE Accounting
MODIFY FILE
 ( NAME = Accounting, SIZE = 100MB )
```

Results Messages

	name	db_size	owner	dbid	created	status	compatibility_level
1	Accounting	105.00 MB	A212-SLLUO\Administrator	8	Sep 8 2015	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	120

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	Accounting	1	D:\AccountingData.mdf	PRIMARY	102400 KB	102400 KB	5120 KB	data only
2	AccountingLog	2	D:\AccountingLog.ldf	NULL	5120 KB	25600 KB	5120 KB	log only

The ALTER Statement

- The ALTER statement change a table/DB rather than re-create it.

```
ALTER TABLE table_name
   {[ALTER COLUMN <column_name>
        { [<schema of new data type>].<new_data_type>
            [(precision [, scale])] max |
<xml schema collection>
        [COLLATE <collation_name>]
        [NULL|NOT NULL]
        |[{ADD|DROP} ROWGUIDCOL] | PERSISTED}]
    |ADD
        <column name> <data_type>
        [[DEFAULT <constant_expression>]
        |[IDENTITY [(<seed>, <increment>) [NOT FOR REPLICATION]]]
        [ROWGUIDCOL]
        [COLLATE <collation_name>]
           [NULL|NOT NULL]
        [<column_constraints>]
        |[<column_name> AS <computed_column_expression>]
    |ADD
```

```
  [CONSTRAINT <constraint_name>]
  {[{PRIMARY KEY|UNIQUE}
      [CLUSTERED|NONCLUSTERED]
      {(<column_name>[ ,...n ])}
      [WITH FILLFACTOR = <fillfactor>]
      [ON {<filegroup> | DEFAULT}]
      ]
      |FOREIGN KEY
         [(<column_name>[ ,...n])]
         REFERENCES <referenced_table> [(<referenced_column>[ ,...n])]
         [ON DELETE {CASCADE|NO ACTION}]
         [ON UPDATE {CASCADE|NO ACTION}]
         [NOT FOR REPLICATION]
      |DEFAULT <constant_expression>
         [FOR <column_name>]
      |CHECK [NOT FOR REPLICATION]
         (<search_conditions>)
   [,...n][ ,...n]
      |[WITH CHECK|WITH NOCHECK]
| { ENABLE | DISABLE } TRIGGER
    { ALL | <trigger name> [ ,...n ] }
|DROP
  {[CONSTRAINT] <constraint_name>
     |COLUMN <column_name>}[ ,...n]
  |{CHECK|NOCHECK} CONSTRAINT
     {ALL|<constraint_name>[ ,...n]}
  |{ENABLE|DISABLE} TRIGGER
     {ALL|<trigger_name>[ ,...n]}
  | SWITCH [ PARTITION <source partition number expression> ]
      TO [ schema_name. ] target_table
      [ PARTITION <target partition number expression> ]
}
```

Example: ALTER TABLE

- Add several additional columns to Employees Table

```
EXEC sp_helpdb Accounting
```

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	EmployeeID	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	FirstName	varchar	no	25			no	no	no	Chinese_PRC_CI_AS
3	MiddleInitial	char	no	1			yes	no	yes	Chinese_PRC_CI_AS
4	LastName	varchar	no	25			no	no	no	Chinese_PRC_CI_AS
5	Title	varchar	no	25			no	no	no	Chinese_PRC_CI_AS
6	SSN	varchar	no	11			no	no	no	Chinese_PRC_CI_AS
7	Salary	money	no	8	19	4	no	(n/a)	(n/a)	NULL
8	PriorSalary	money	no	8	19	4	no	(n/a)	(n/a)	NULL
9	LastRaise	money	yes	8	19	4	yes	(n/a)	(n/a)	NULL
10	HireDate	date	no	3	10	0	no	(n/a)	(n/a)	NULL
11	Termination...	date	no	3	10	0	yes	(n/a)	(n/a)	NULL
12	ManagerEm...	int	no	4	10	0	no	(n/a)	(n/a)	NULL
13	Department	varchar	no	25			no	no	no	Chinese_PRC_CI_AS

Example: ALTER TABLE (cont.)

```
ALTER TABLE Employees
  ADD
      PreviousEmployer  VARCHAR(30) NULL

ALTER TABLE Employees
  ADD
      DateOfBirth       DATE         NULL,
      LastRaiseDate     DATE         NOT NULL  DEFAULT '2008-01-01'
```

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	EmployeeID	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	FirstName	varchar	no	25			no	no	no	Chinese_PRC_CI_AS
3	MiddleInitial	char	no	1			yes	no	yes	Chinese_PRC_CI_AS
4	LastName	varchar	no	25			no	no	no	Chinese_PRC_CI_AS
5	Title	varchar	no	25			no	no	no	Chinese_PRC_CI_AS
6	SSN	varchar	no	11			no	no	no	Chinese_PRC_CI_AS
7	Salary	money	no	8	19	4	no	(n/a)	(n/a)	NULL
8	PriorSalary	money	no	8	19	4	no	(n/a)	(n/a)	NULL
9	LastRaise	money	yes	8	19	4	yes	(n/a)	(n/a)	NULL
10	HireDate	date	no	3	10	0	no	(n/a)	(n/a)	NULL
11	Termination...	date	no	3	10	0	yes	(n/a)	(n/a)	NULL
12	ManagerEm...	int	no	4	10	0	no	(n/a)	(n/a)	NULL
13	Department	varchar	no	25			no	no	no	Chinese_PRC_CI_AS
14	PreviousEm...	varchar	no	30			yes	no	yes	Chinese_PRC_CI_AS
15	DateOfBirth	date	no	3	10	0	yes	(n/a)	(n/a)	NULL
16	LastRaiseDate	date	no	3	10	0	no	(n/a)	(n/a)	NULL

The DROP Statement

- The DROP statement allows we to remove or delete a table/DB from the SQL database.
- The DROP statement will always look like this:

```
DROP <object type> <object name> [, ...n]
```

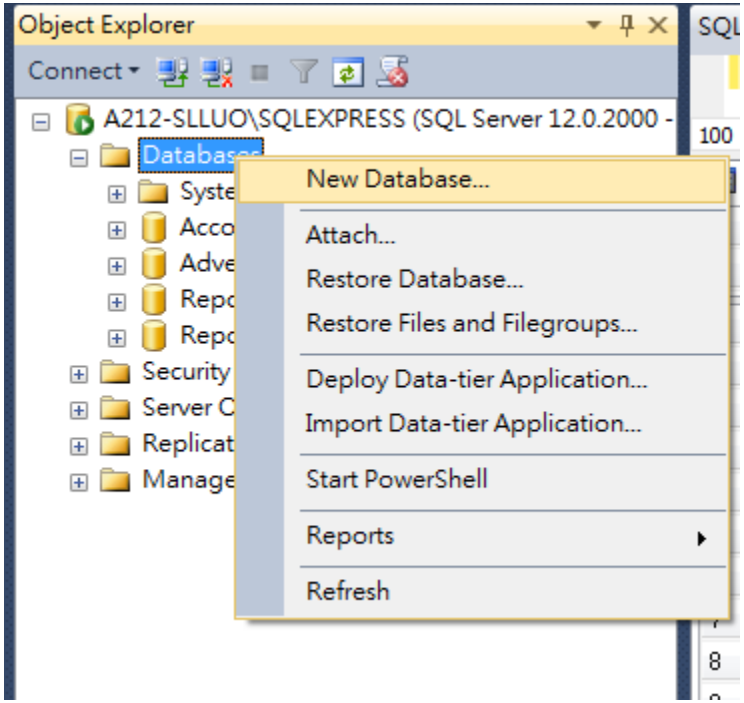
- Example: Drop both of two tables at the same time:

```
DROP TABLE Customers, Employees
```

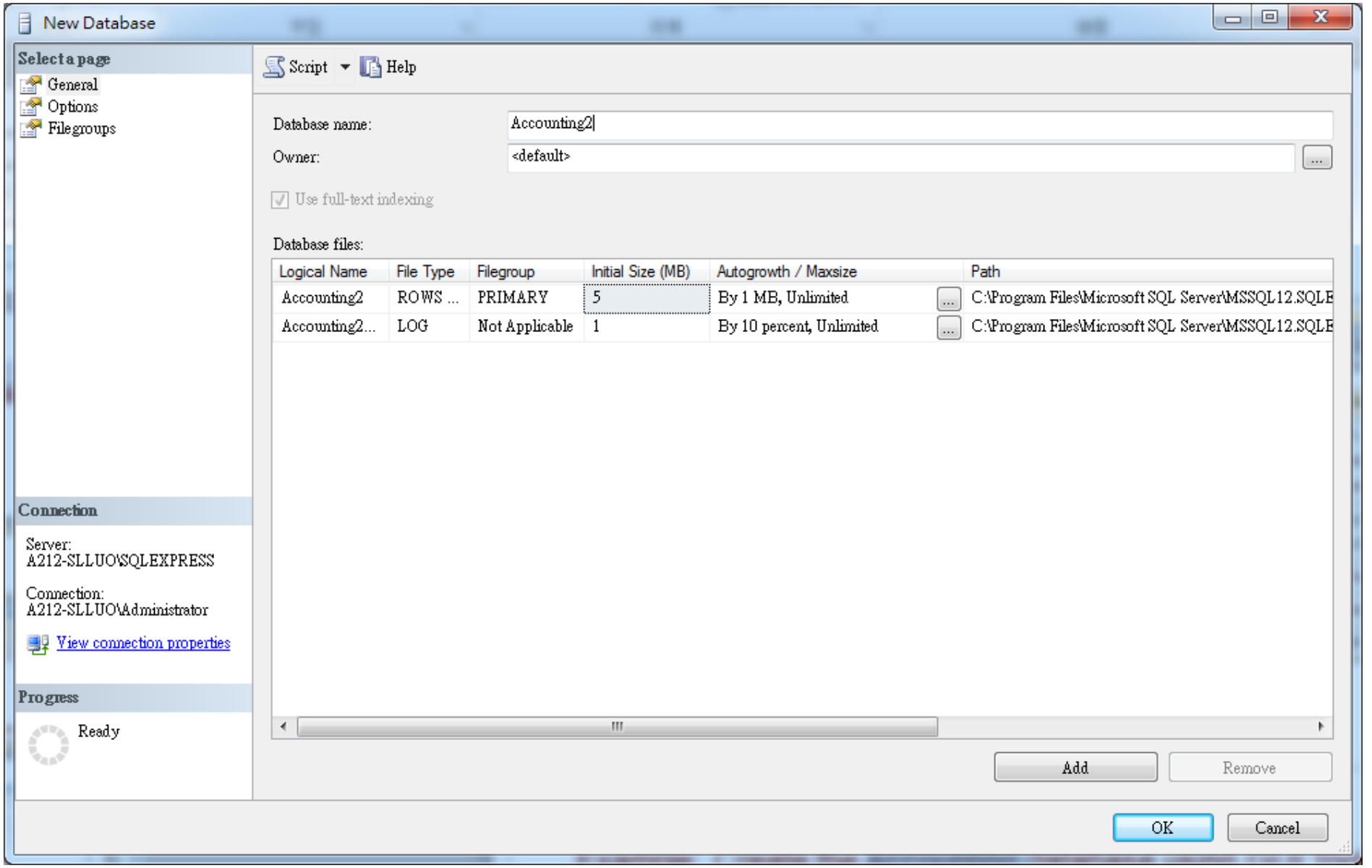
Using GUI Tools

Creating a Database

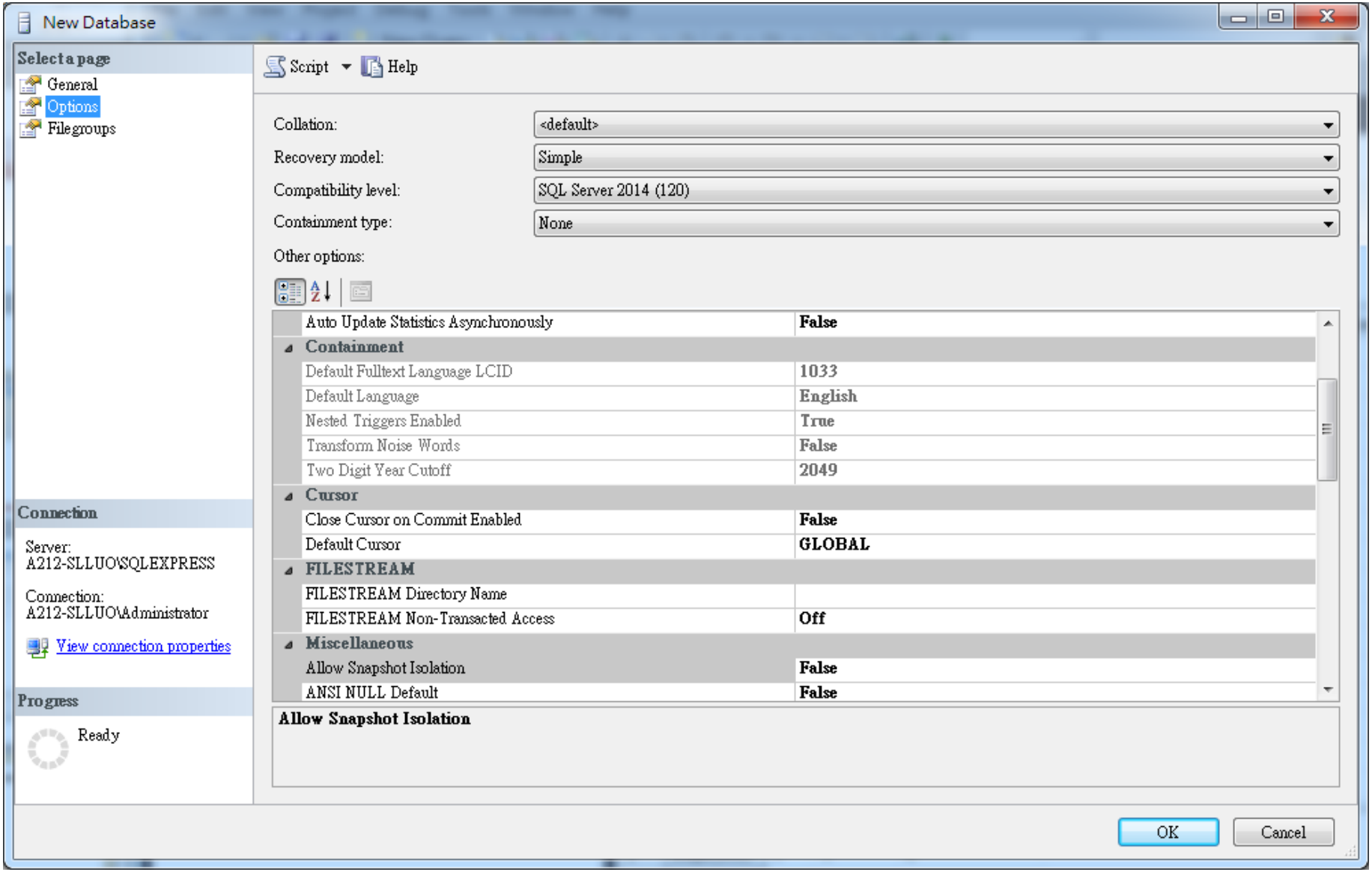
- Follow these steps to create a database using Management Studio:
  - Right-click the Databases node, and choose the New Database option. This pulls up the Database Properties dialog box.
  - We can fill in the information on how we want our database created.



Example: Create the database using GUI Tools

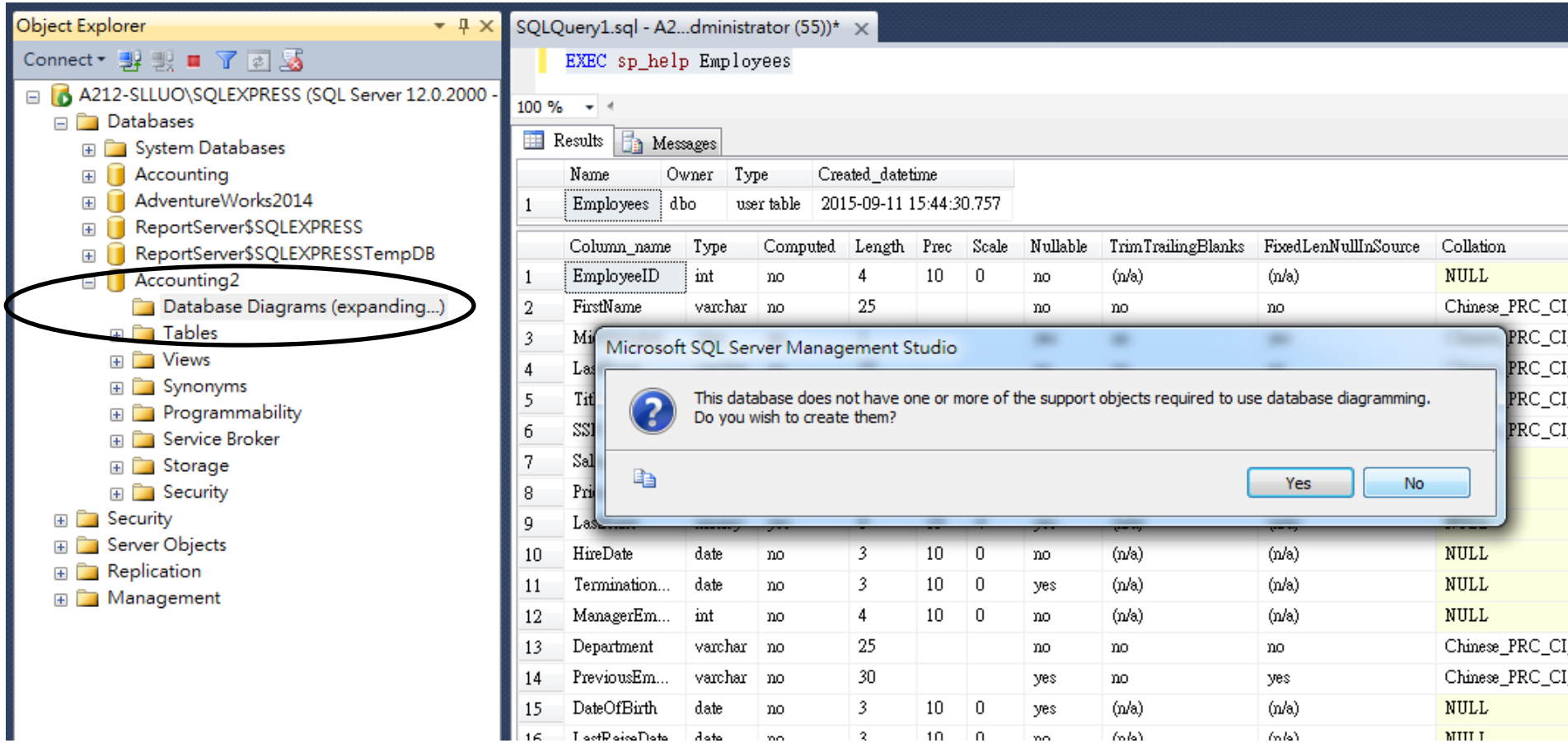

New Database
Select a page
General
Options
Filegroups
Script
Help
Database name:
Accounting2
Owner:
<default>
Use full-text indexing
Database files:
Logical Name
File Type
Filegroup
Initial Size (MB)
Autogrowth / Maxsize
Path
Accounting2
ROWS ...
PRIMARY
5
By 1 MB, Unlimited
C:\Program Files\Microsoft SQL Server\MSSQL12.SQLE
Accounting2...
LOG
Not Applicable
1
By 10 percent, Unlimited
C:\Program Files\Microsoft SQL Server\MSSQL12.SQLE
Add
Remove
Connection
Server:
A212-SLLUO\SQLEXPRESS
Connection:
A212-SLLUO\Administrator
View connection properties
Progress
Ready
OK
Cancel

- Click Options tab, which contains a host of additional settings

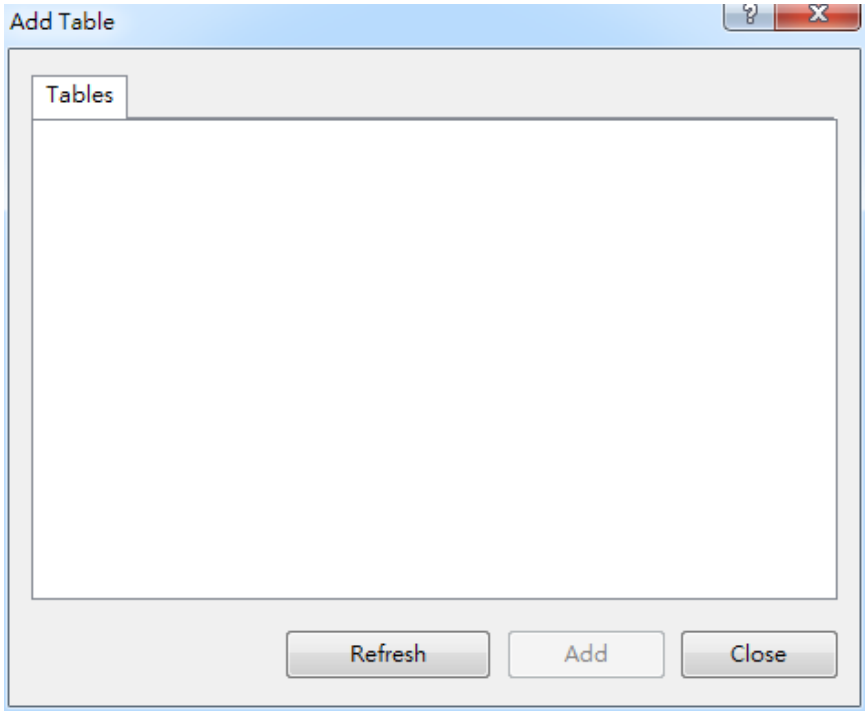


- Using default setting, and click OK

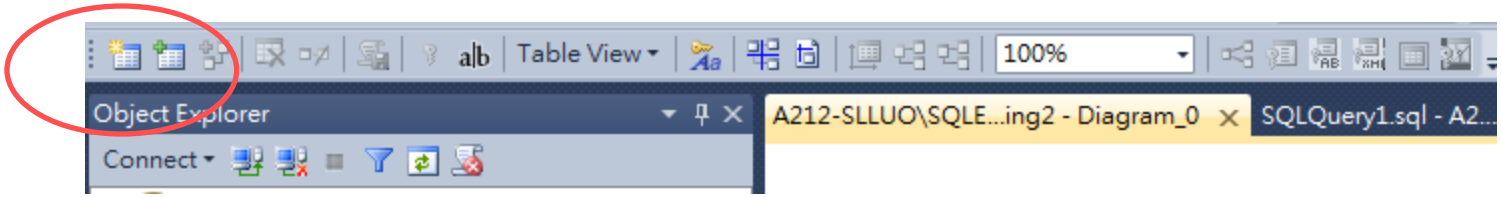
- Expand the tree to show the various items underneath the Accounting node, and select the Database Diagrams node. Right-click it, and we'll get a dialog indicating that the database is missing some objects it needs to support database diagramming. Click Yes.



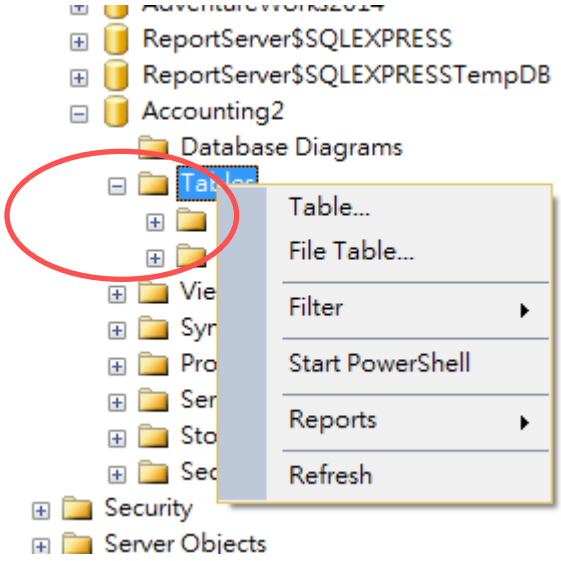
- With that, we'll get an Add Table dialog, just close it in current (we do not have any table currently).



- We can add a table either by right-clicking and choosing the appropriate option, or by clicking the New Table icon in the toolbar.

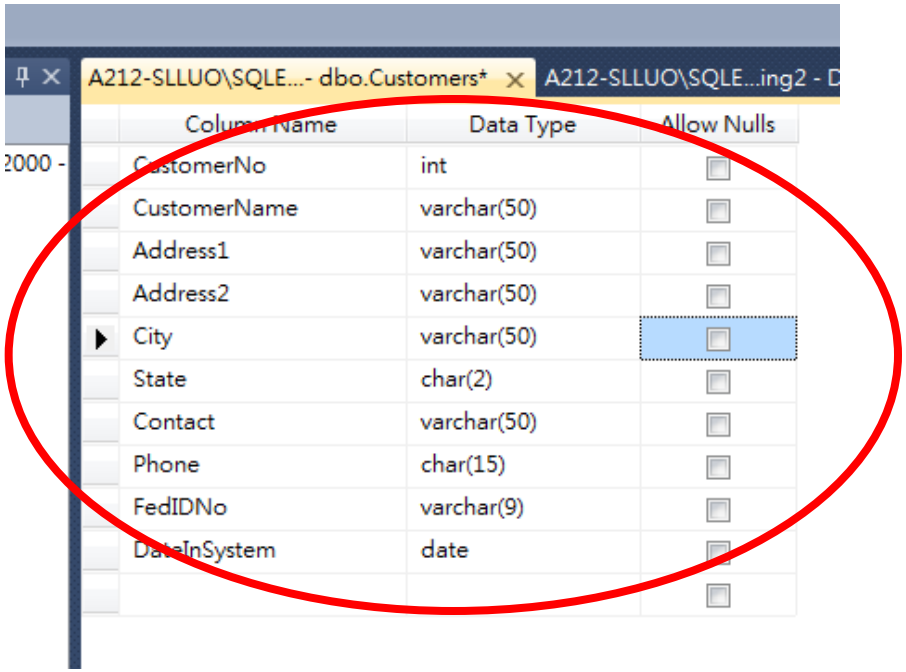


- OR

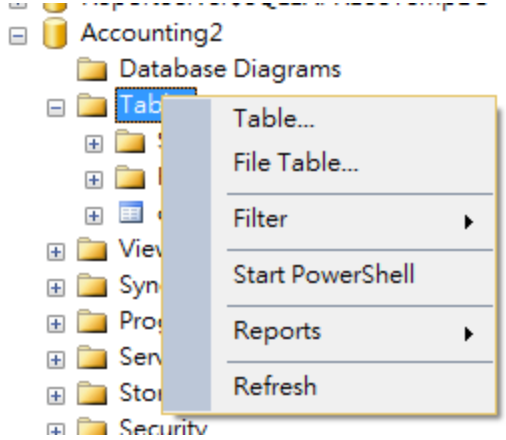


Create Table in Design Windows

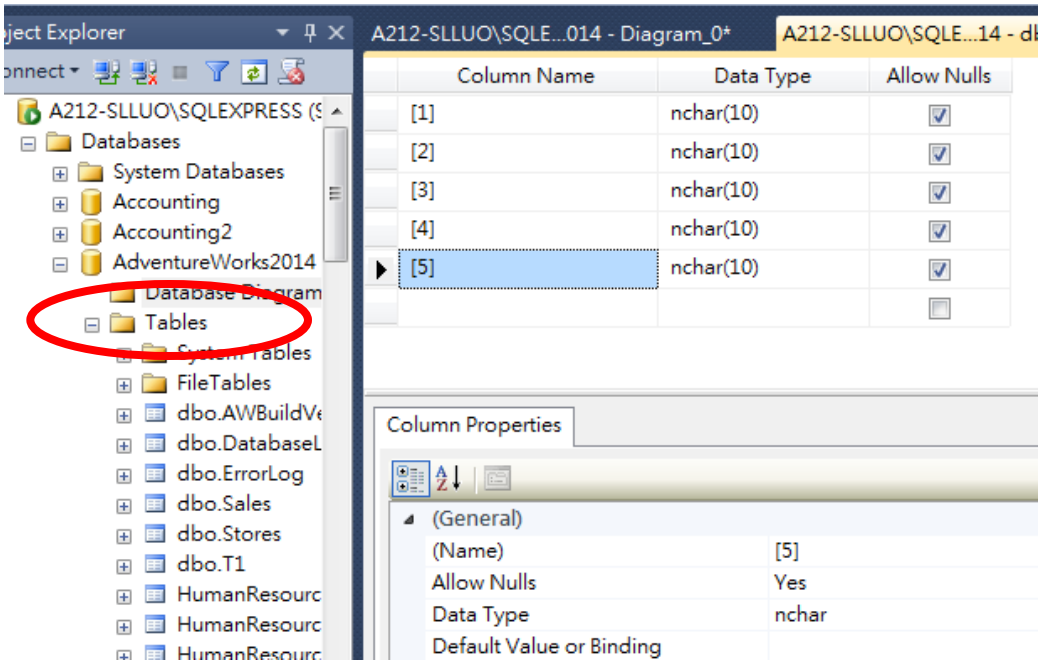
- Fill in the table one piece at a time



- To refresh the list to see the change

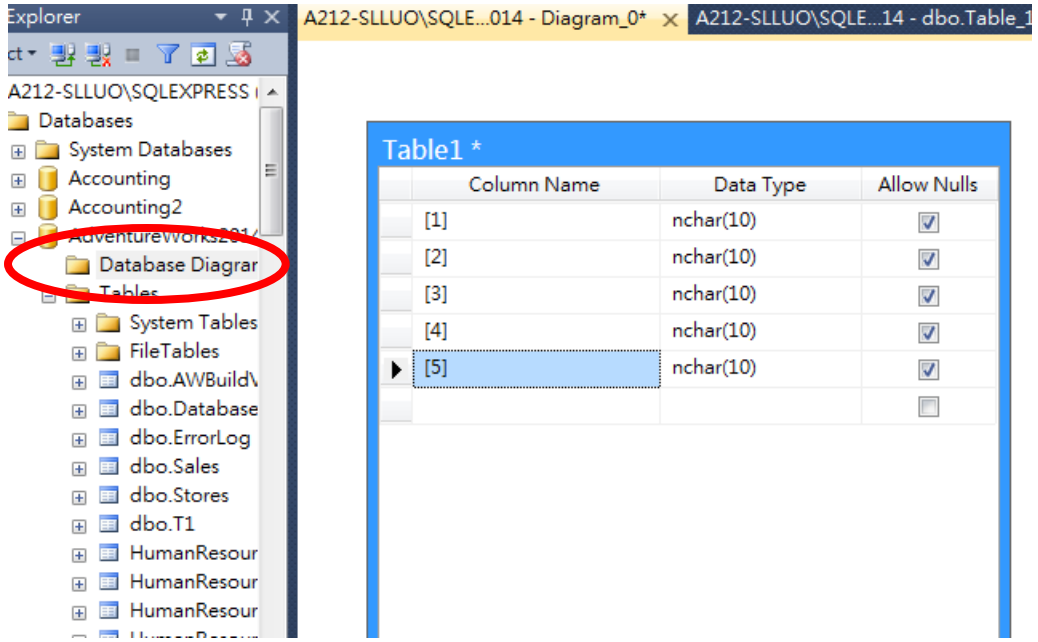


Create Table in Design Windows

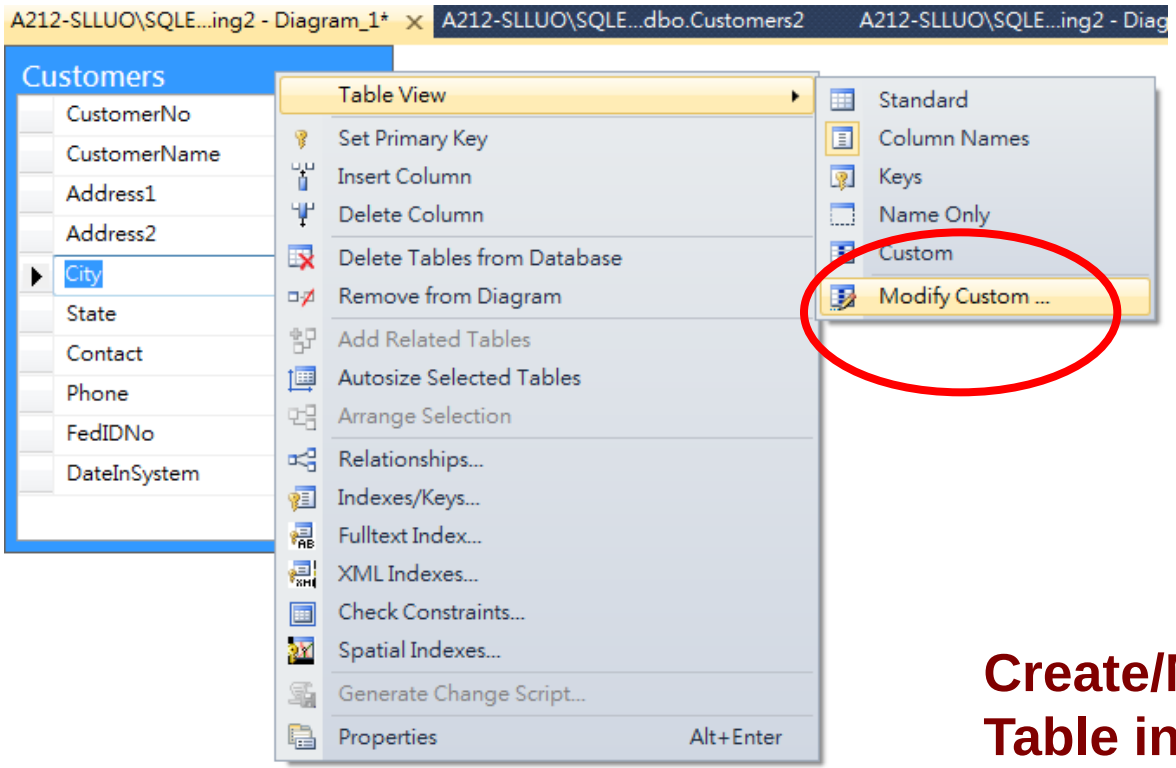


Create Table in Design Windows

Create Table in SQL Diagram

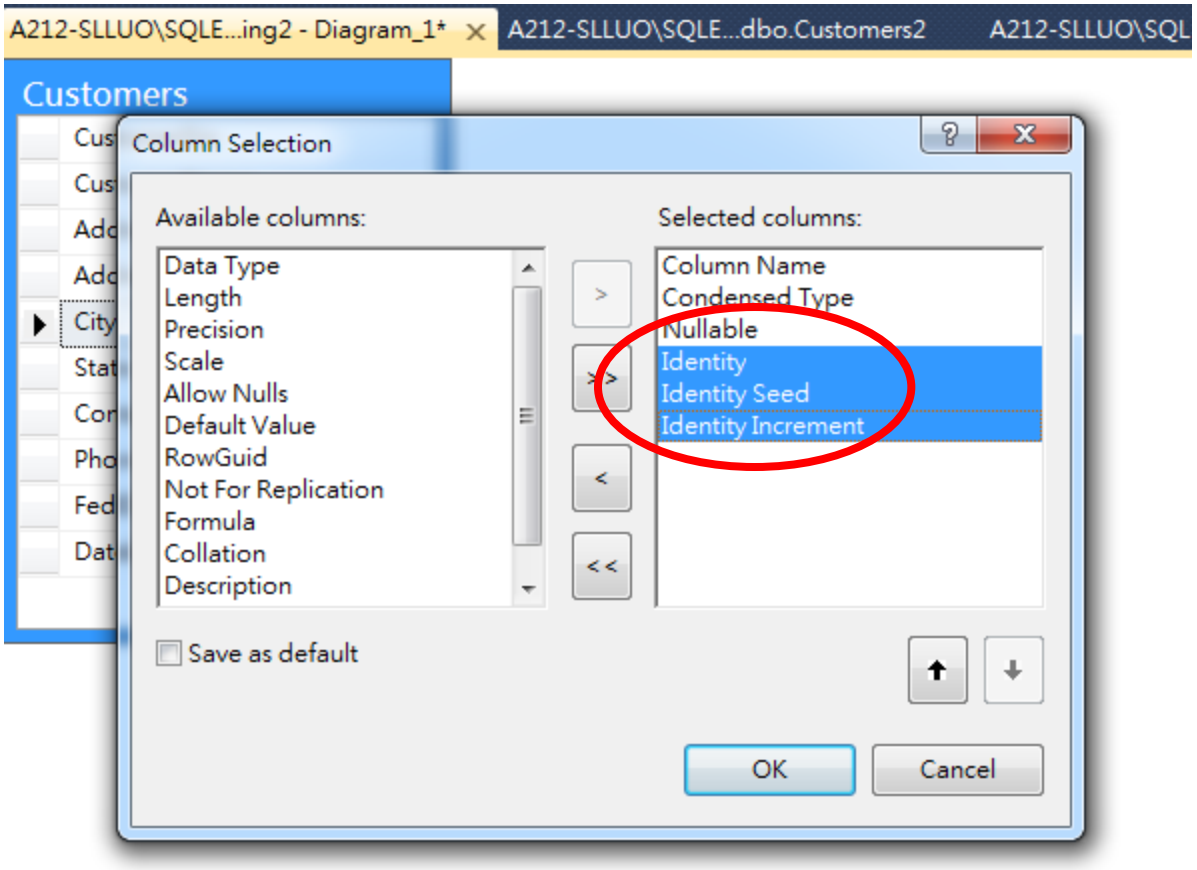


- We also need to define our first column as being an identity column. Unfortunately, we don't appear to have any way of doing that with the default grid here.
- So, the following step to define first column and identity column "In ***SQL Diagram***, selects the table ➪ Right Click ➪ Table View ➪ Modify Custom"



Create/Modify Table in SQL Diagram

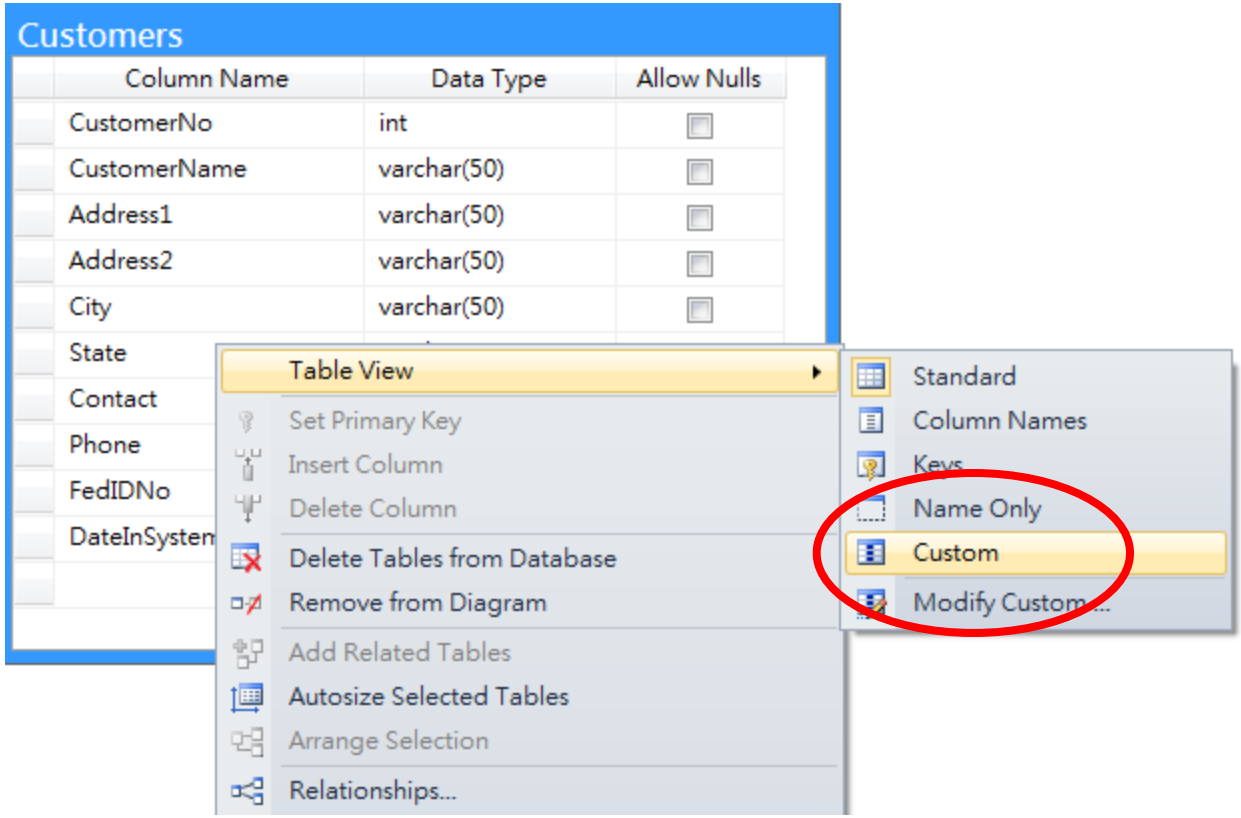
- So, the following step to define first column and identity column “In SQL Diagram, selects the table ➪ Right Click ➪ Table View ➪ Modify Custom”



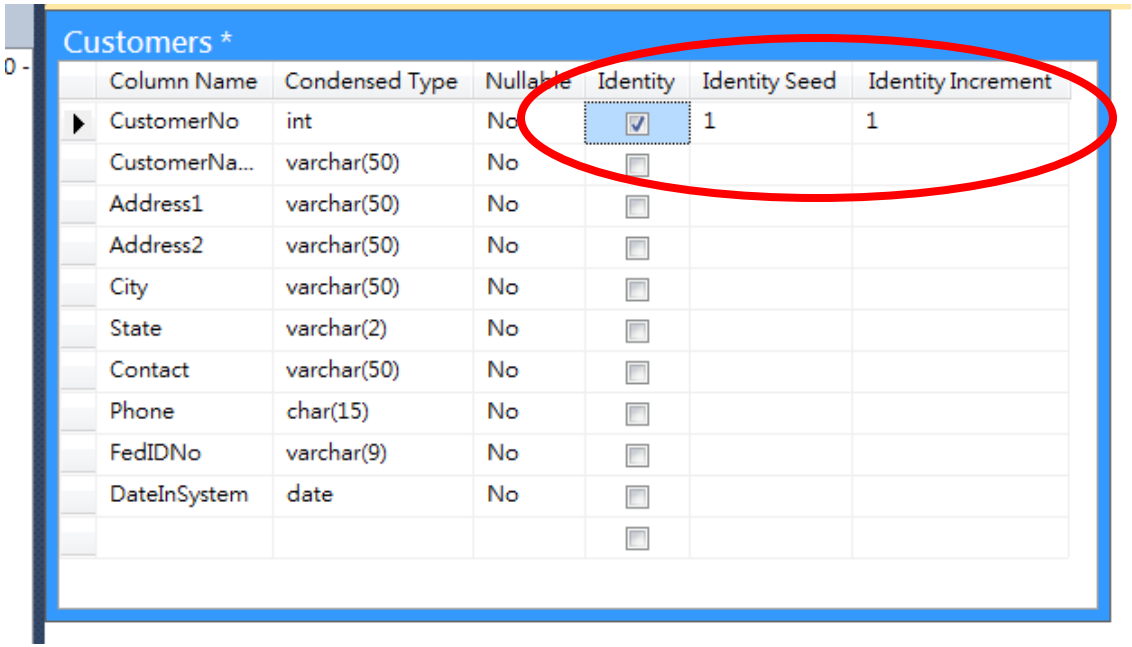
- Then click ‘OK’

Create/Modify Table in SQL Diagram

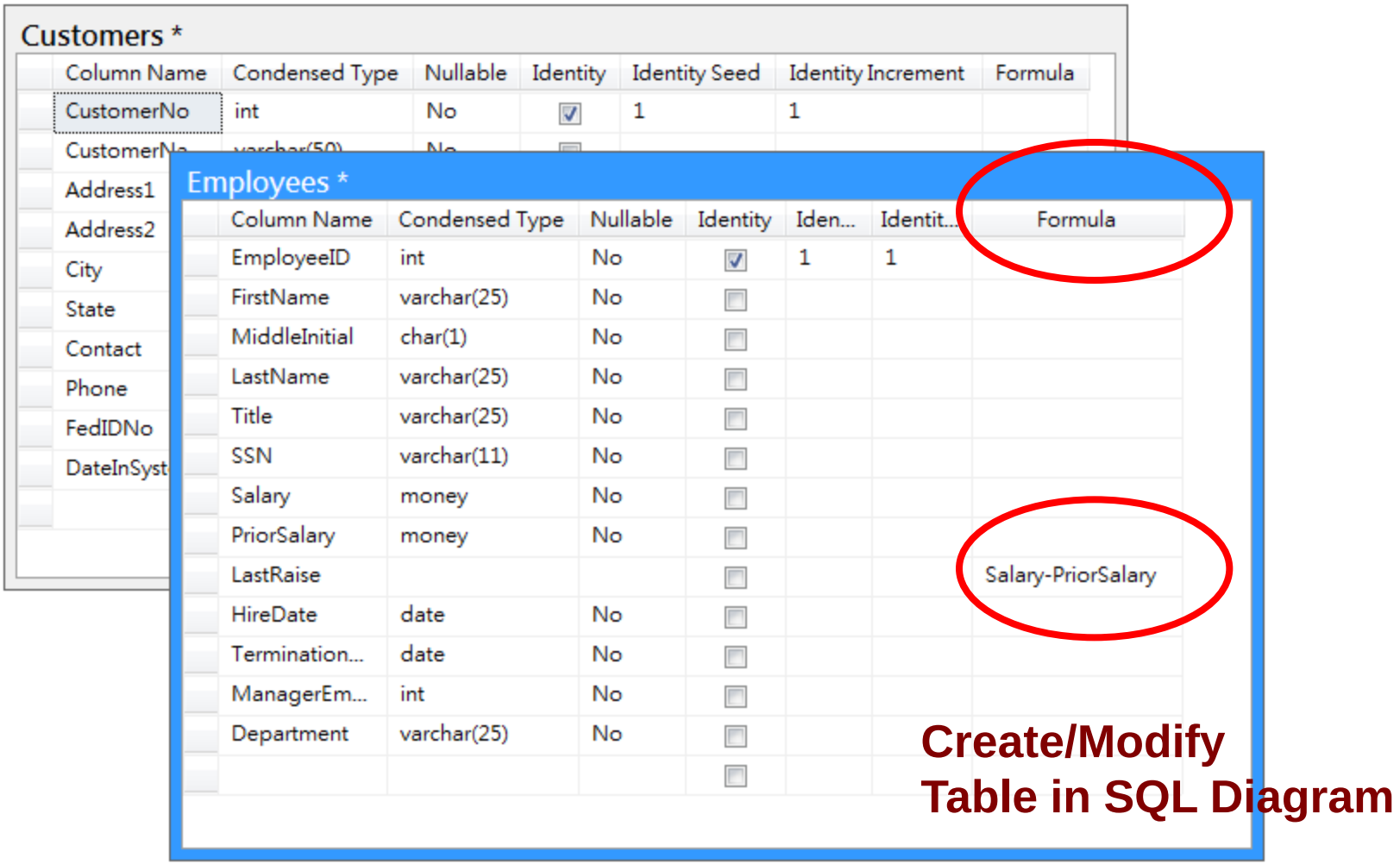
- Go back to the editing dialog box and select Table View ➪ Custom to view the identity column



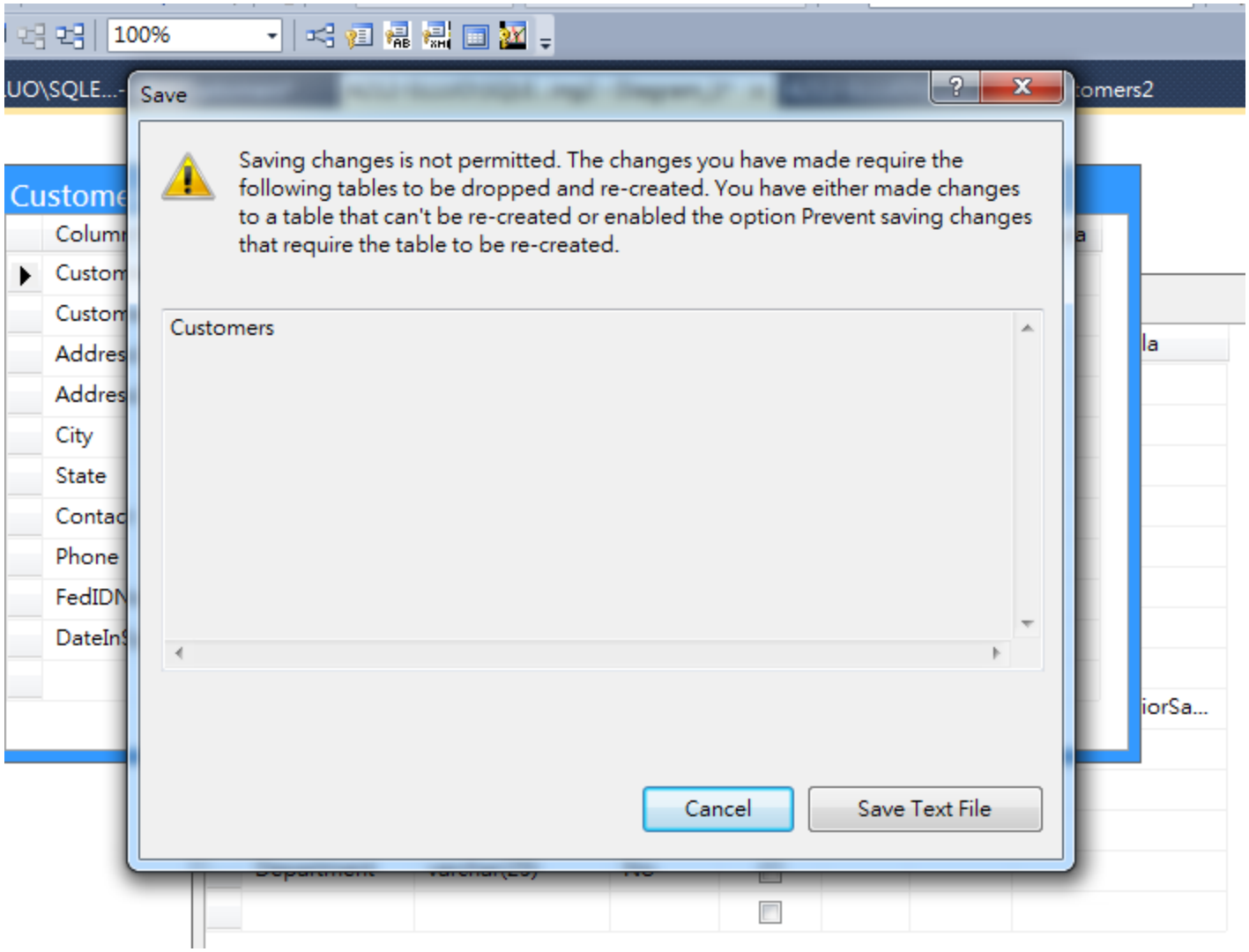
- Go back to the editing dialog box and select Table View ➪ Custom to view the identity column



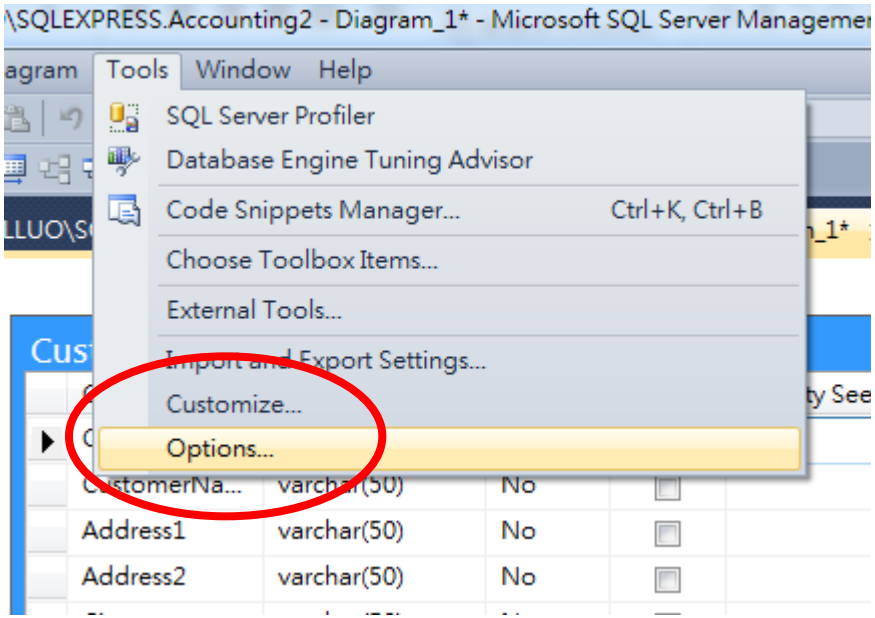
- Add the Employees table.
- To deal with the computed column, just select Modify Custom again (from the right-click menu), and add the Formula column. Then, simply add the proper formula (in this case, Salary-PriorSalary).



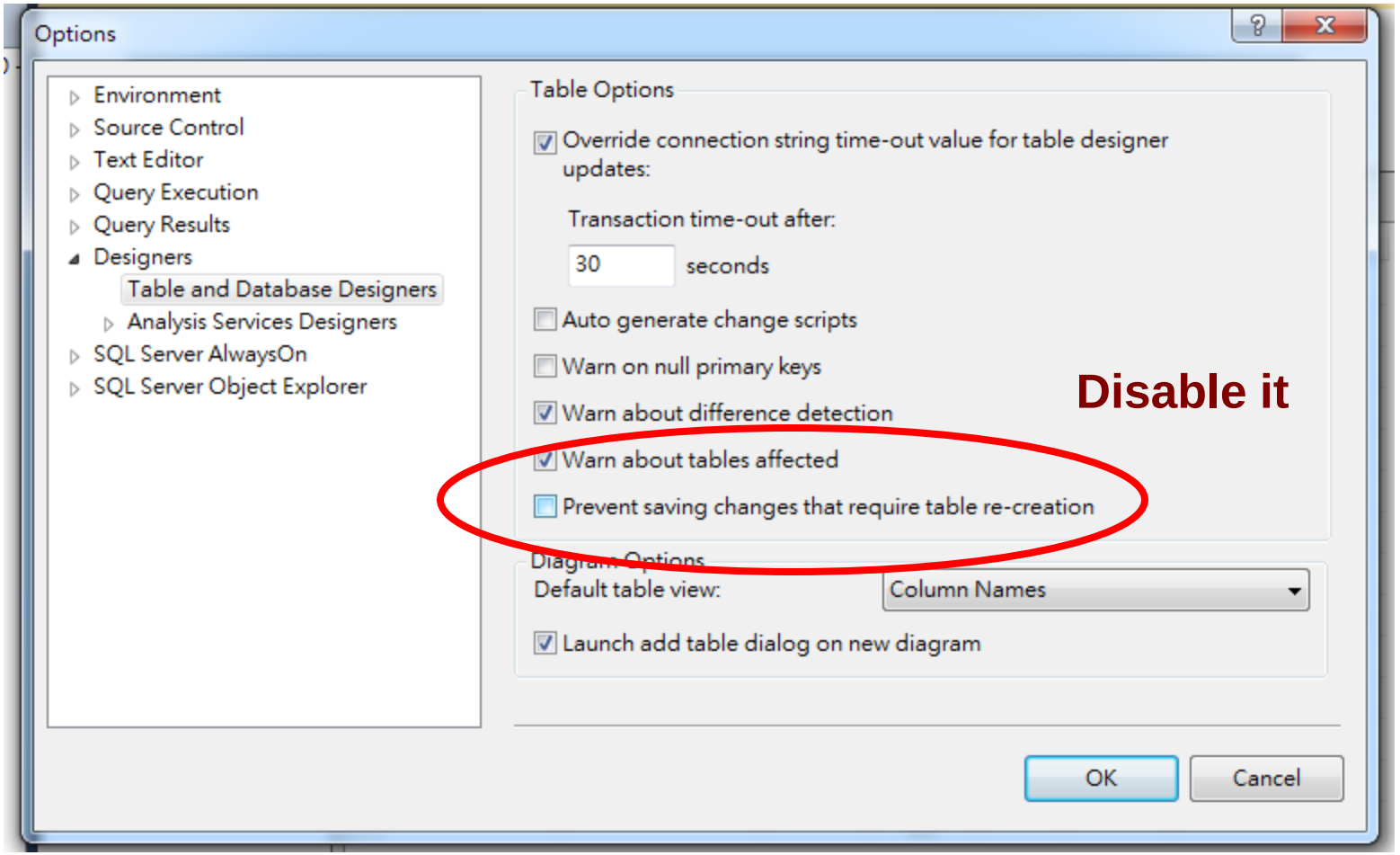
- ISSUE: Saving is not permitted.



- SOULTION: Saving is not permitted.

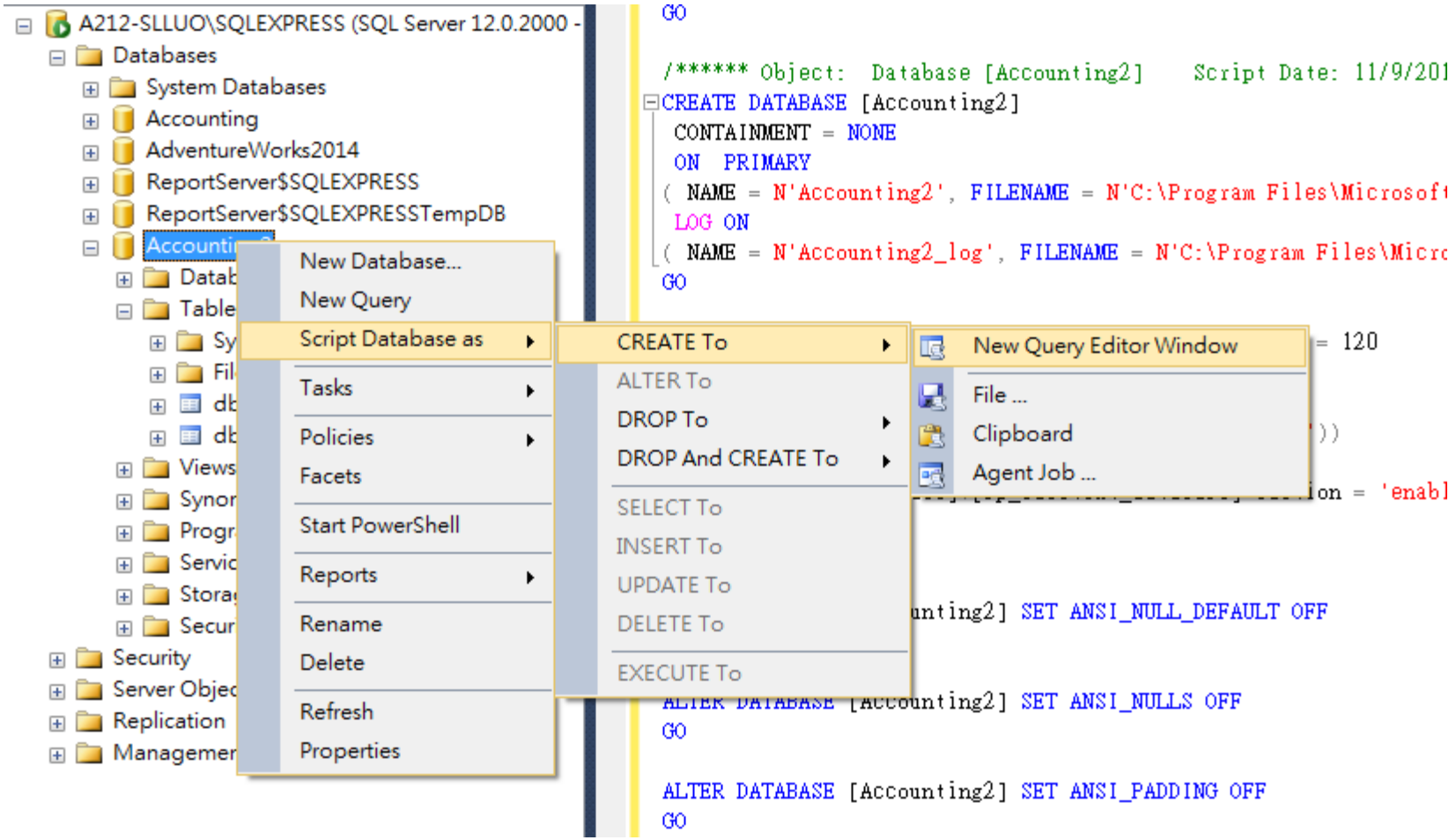


- SOULTION: Saving is not permitted.



Creating Scripts with the Management Studio

- Management Studio can write our scripts for us



Summary

TOPIC	CONCEPT
CREATE DATABASE	The container for all your data, the database object is created using the CREATE DATABASE statement. You can specify the size and location of the data and log fi les, spread the data across multiple files, and set up a data growth strategy.
CREATE TABLE	Tables are the basic unit of data storage. CREATE TABLE starts similarly to CREATE DATABASE, but the rest of the syntax centers on specifying column names and their data types, keys, and constraints, as well as some storage options.
ALTER DATABASE	Functions similarly to CREATE DATABASE, but applies its changes to a database that already exist. ALTER DATABASE will throw an error if it’s run against a nonexistent database.
ALTER TABLE	Can be used to add or change columns, constraints, or storage on an existing table.
DROP DATABASE DROP TABLE	Used to remove an object entirely from the system. Dropping a database or table results in loss of any data stored within.

Exercises

1. Using the Management Studio's script generator, generate SQL for both the Customers and the Employees tables.
2. Without using the Management Studio, script a database called MyDB with a starting database size of 17MB and a starting log size of 5MB - set both the log and the database to grow in 5MB increments.
3. Create a table called Foo with a single variable length character field called Col1 - limit the size of Col1 to 50 characters.

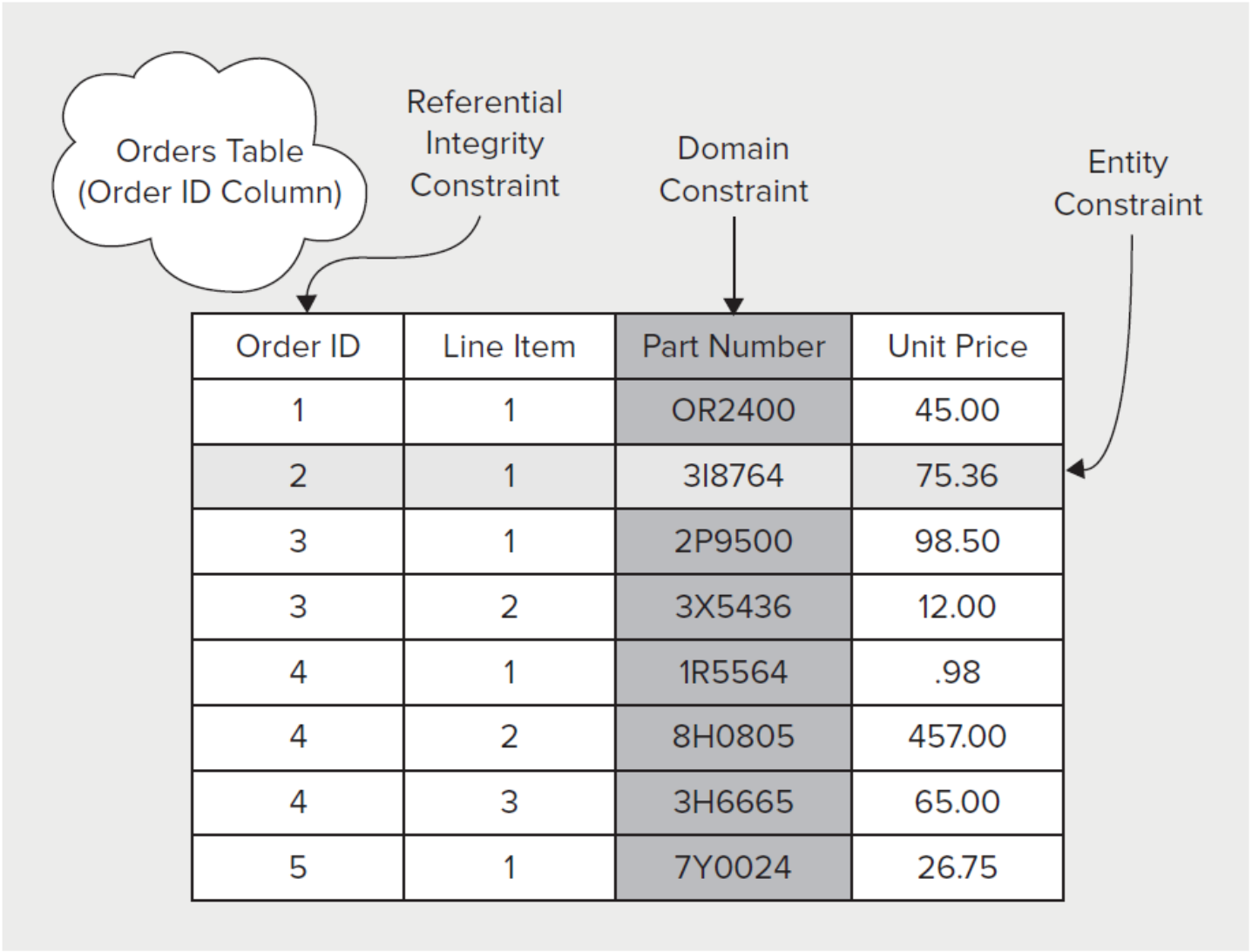
Keys and Constraints

Constraints

- The DBMS controls the interaction of the user with the database to ensure that a number of data integrity constraints are observed.
- Three integrity constraints are used to achieve data integrity:
  - Entity constraints
  - Domain constraints
  - Referential integrity constraints
- The methods of implementing each of these types of constraints at a more specific level are
  - Primary key constraints, foreign key constraints, unique constraints, check constraints, default constraints, etc.

Types of Constraints

- T e
  - d
  - c
- T
  - d



Order ID	Line Item	Part Number	Unit Price
1	1	OR2400	45.00
2	1	3I8764	75.36
3	1	2P9500	98.50
3	2	3X5436	12.00
4	1	1R5564	.98
4	2	8H0805	457.00
4	3	3H6665	65.00
5	1	7Y0024	26.75

Types of Constraints

- ***Domain constraints*** deal with one or more columns.
  - Ensuring that a particular column or set of columns meets particular criteria.
  - For example, UnitPrice should be ≥ 0
- ***Entity constraints*** are all about comparing rows.
  - Entity constraints are best exemplified by a constraint that requires every row to have a unique value for a column or combination of columns
  - For example, primary key and unique constraints
- ***Referential integrity constraints*** are created when a value in one column must match the value in another column.
  - For example, foreign key constraints

Constraints Name

- SQL Server generates name for a primary key on the table might be something like PK__Table1__145C0A3F.
- But sometime we can not get help for this generated name on other constraints.
  - A CHECK constraint, might generate something like CK__Table2__22AA2996.
- Also, when we have system-generated names, our scripts will not work on other server when we need to edit our scripts
  - SQL Server creates a slightly different system generated name each time it creates a constraint name
- It it better to beware of the system generated constraint name

Primary Key Constraints

- ***Primary keys*** are the unique identifiers for each row. They must contain unique values (and hence cannot be NULL).
- Example, creating the primary key at table creation

```
CREATE TABLE Customers
(
    CustomerNo     INT             IDENTITY NOT NULL PRIMARY KEY,
    CustomerName   VARCHAR(30)     NOT NULL,
    Address1       VARCHAR(30)     NOT NULL,
    Address2       VARCHAR(30)     NOT NULL,
...
);
```

	index_name	index_description	index_keys
1	PK__Customer__A4AFBF63BBBCE00C	clustered, unique, primary key located on PRIMARY	CustomerNo

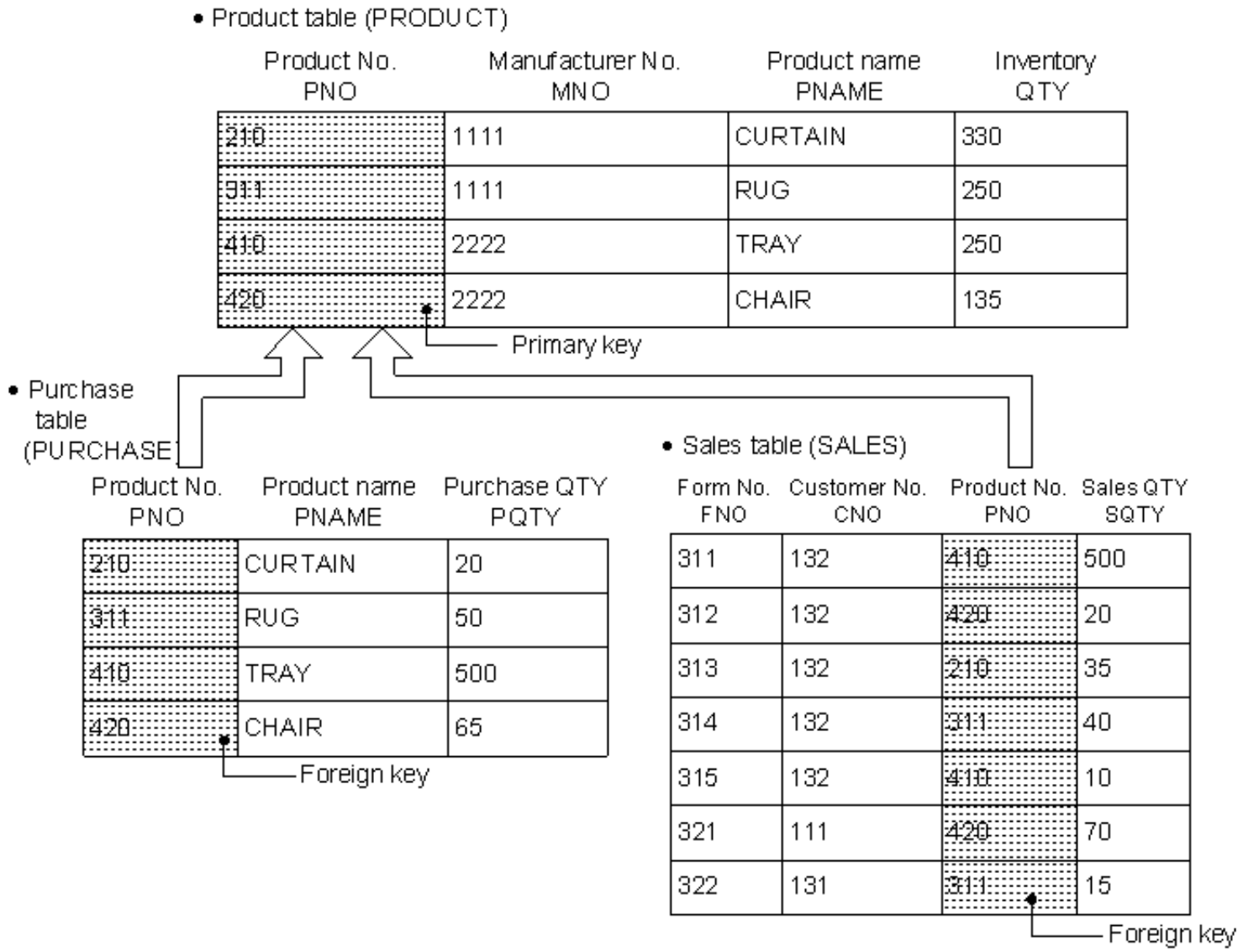
- Example, creating a primary key on an existing table

```
ALTER TABLE Employees
  ADD CONSTRAINT PK_Employees
      PRIMARY KEY (EmployeeID);
```

Foreign Key Constraints

- ***Foreign keys*** are both a method of ensuring data integrity and a manifestation of the relationships between tables.
- When we add a foreign key to a table, we are creating a dependency between the tables
  - we define the foreign key (***the referencing table***), and
  - the table our foreign key references (***the referenced table***).
- After adding a foreign key:
  - any record we insert into ***the referencing table*** must have a matching record in the referenced column(s) of ***the referenced table***,
  - or the value of the foreign key column(s) must be set to NULL.

Foreign Keys Constraints



- The syntax goes on the column or columns that we are placing our FOREIGN KEY constraint on like this:

```
{ CREATE TABLE <table name> | ALTER TABLE <table name> }
(
   <column name> <data type> <nullability>
   [FOREIGN KEY] REFERENCES <table name>(<column name>)
       [ON DELETE {CASCADE|NO ACTION|SET NULL|SET DEFAULT}]
       [ON UPDATE {CASCADE|NO ACTION|SET NULL|SET DEFAULT}]
);
```

- Example: creating a table with a foreign key

```
CREATE TABLE Orders
(
   OrderID      INT    IDENTITY NOT NULL PRIMARY KEY,
   CustomerNo   INT    NOT NULL
                       FOREIGN KEY REFERENCES Customers(CustomerNo),
   OrderDate    DATE   NOT NULL,
   EmployeeID   INT    NOT NULL
);
```

- Example: adding a foreign key to an existing table

```
ALTER TABLE Orders
  ADD CONSTRAINT FK_EmployeeCreatesOrder
      FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID);
```

Making a Table Self-Referencing

- A table can be both the referencing and the referenced table.
- For the ***self-referencing constraint*** that references a required (***non-nullable***) field that's based on ***an identity column*** - we need at least one row in the table. For example,

```
CREATE TABLE Customers
(
    CustomerNo     INT             IDENTITY PRIMARY KEY,
    CustomerName   VARCHAR(30)     NOT NULL,
    FriendNo       INT             NOT NULL REFERENCES Customers(CustomerNo)
);

INSERT Customers(CustomerName, FriendNo) VALUES('Athena', ??????);
```

- The problem stems from the fact that the identity value is chosen and filled in after the foreign key has already been checked and enforced - we don't have a value yet for that first row to reference when the check happens.

- For example,

```
INSERT Customers(CustomerName, FriendNo)
   VALUES('Athena', 1);--How do you know the system generated value is '1'?

--Use IDENT_CURRENT to return the last identity value generated for a
--specified table or view.
INSERT Customers(CustomerName, FriendNo)
   VALUES('Athena', IDENT_CURRENT('Customers'));
```

- Another option is to go ahead and create the foreign key but then disable it when adding the first row.

```
CREATE TABLE Customers (
    CustomerNo     INT             IDENTITY PRIMARY KEY,
    CustomerName   VARCHAR(30)     NOT NULL,
    FriendNo       INT             NOT NULL
                                   CONSTRAINT FK_100
                                   REFERENCES Customers(CustomerNo) );
```

```
ALTER TABLE Customers NOCHECK CONSTRAINT FK_100;

INSERT Customers(CustomerName, FriendNo) VALUES('Athena', 10);

ALTER TABLE Customers CHECK CONSTRAINT FK_100;
```

Cascading Actions

- One important difference between foreign keys and other kinds of keys is that foreign keys are ***bidirectional***.
  - not only restrict the child table (referencing table) to values that exist in the parent (referenced table),
  - but they also check for child rows whenever we do something to the parent (referenced table).
- The default behavior is for SQL Server to “restrict” the parent row from being deleted if any child rows exist.
- Sometimes, we would rather automatically delete any dependent records rather than prevent the deletion.
  - The process of making such automatic deletions and updates is known as ***cascading***.

- All we need is a modification to the syntax we use when declaring our foreign key just by adding the ON clause.
- For example, create a new table called OrderDetails

```
CREATE TABLE OrderDetails
(
   OrderID      INT            NOT NULL,
   PartNo       VARCHAR(10)    NOT NULL,
   Description  VARCHAR(25)    NOT NULL,
   UnitPrice    MONEY          NOT NULL,
   Qty          INT            NOT NULL,
   CONSTRAINT   PK_OrderDetails PRIMARY KEY (OrderID, PartNo),
   CONSTRAINT   FK_OrderContainsDetails FOREIGN KEY (OrderID)
                                        REFERENCES Orders(OrderID)
                                        ON UPDATE NO ACTION
                                        ON DELETE CASCADE
);
```

- Note: Instead of placing the primary key declaration immediately after the key, we declared it as a separate constraint item.
- We've declared the OrderID as being dependent on a "foreign" column (Orders(OrderID)).

```
...
   CONSTRAINT   PK_OrderDetails PRIMARY KEY (OrderID, PartNo),
   CONSTRAINT   FK_OrderContainsDetails FOREIGN KEY (OrderID)
                                        REFERENCES Orders(OrderID)
                                        ON UPDATE NO ACTION
                                        ON DELETE CASCADE);
...
```

- At the ON clauses:

```
ON UPDATE NO ACTION
ON DELETE CASCADE
```

- We've defined two referential integrity actions.
- A referential integrity action is what we want to have happen whenever the referential integrity rule we've defined is invoked.
- **ON UPDATE NO ACTION**, mean, the parent record (in the Orders table) is updated, we've said that we do not want that update to be cascaded to the child table (OrderDetails), so the UPDATE is rolled back.

- Example 1: Dependency chains. If we try an insert into the OrderDetails table:

```
INSERT INTO OrderDetails
VALUES (1, '4X4525', 'This is a part', 25.00, 2);
```

```
The INSERT statement conflicted with the FOREIGN KEY constraint
"FK_OrderContainsDetails". The conflict occurred in database
"AdventureWorks2014", table "Orders", column 'OrderID'.
```

- The error is because the Orders table is empty in currently.
- In order to get the row into the OrderDetails table:
  - We must also have a record in the Orders table, and getting a row into the Orders table requires that we have one in the Customers table.
- A dependency chain exists when we have something that is, in turn, dependent on something else, which may yet be dependent on something else, and so on.
- We have to start at the top of the chain and work our way down to what we need inserted.

- Example 1: Dependency chains.

```
INSERT INTO Customers
VALUES ('Billy Bob''s Shoes', '123 Main St.', ' ', 'Vancouver', 'WA', ... );

INSERT INTO Orders (CustomerNo, OrderDate, EmployeeID)
VALUES (1, GETDATE(), 1);

INSERT INTO OrderDetails
VALUES (1, '4X4525', 'This is a part', 25.00, 2)

INSERT INTO OrderDetails
VALUES (1, '0R2400', 'This is another part', 50.00, 2);
```

- As a result:

```
SELECT OrderID, PartNo FROM OrderDetails;
```

OrderID	PartNo
1	0R2400
1	4X4525

```
SELECT OrderID, CustomerNo FROM Orders;
```

OrderID	CustomerNo
1	1

```
SELECT CustomerNo, CustomerName FROM Customers;
```

CustomerNo	CustomerName
1	Billy Bob's...

- Example 2: UPDATE - we can not update the value

```
UPDATE Orders
SET    OrderID = 2
WHERE  OrderID = 1;
```

```
The UPDATE statement conflicted with the REFERENCE constraint
"FK_OrderContainsDetails". The conflict occurred in database
"AdventureWorks2014", table "OrderDetails", column 'OrderID'.
```

- Example 3: DELETE - Cascaded to our matching records in the OrderDetails table – records in both tables were deleted.

```
DELETE Orders
WHERE  OrderID = 1;
```

```
SELECT OrderID, PartNo FROM OrderDetails;
```

```
OrderID      PartNo
-----------  ----------
```

```
SELECT OrderID, CustomerNo FROM Orders;
```

```
OrderID      CustomerNo
----------   -----------
```

```
SELECT CustomerNo, CustomerName FROM Customers;
```

```
CustomerNo   CustomerName
-----------  --------------
1            Billy Bob's...
```

ACTIONS	DESCRIPTIONS
ON DELETE NO ACTION	Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE statement is rolled back.
ON UPDATE NO ACTION	Specifies that if an attempt is made to update a key value in a row whose key is referenced by foreign keys in existing rows in other tables, an error is raised and the UPDATE is rolled back.
ON DELETE CASCADE	All rows that contain those foreign keys are also deleted.
ON UPDATE CASCADE	All the values that make up the foreign key are also updated to the new value specified for the key.
ON DELETE SET NULL	All the values that make up the foreign key in the rows that are referenced are set to NULL. All foreign key columns of the target table must be nullable.
ON UPDATE SET NULL	All the values that make up the foreign key in the rows that are referenced are set to NULL. All foreign key columns of the target table must be nullable.
ON DELETE SET DEFAULT	All the values that make up the foreign key in the rows that are referenced are set to their default value. If a column is nullable, and there is no explicit default value set, NULL becomes the implicit default value of the column.
ON UPDATE SET DEFAULT	All the values that make up the foreign key in the rows that are referenced are set to their default value.

UNIQUE Constraints

- ***UNIQUE constraints*** are essentially the younger sibling of primary keys in that they require a unique value throughout the named column(s) in the table.
  - Often hear UNIQUE constraints referred to as alternate keys
- Once we establish a UNIQUE constraint, every value in the named columns must be unique.
  - If we try to update or insert a row with a value that already exists in a column with a UNIQUE constraint, SQL Server will raise an error and reject the record.

UNIQUE Constraints

- Unlike a primary key, a UNIQUE constraint does not automatically prevent us from having a NULL value.
  - Whether NULLs are allowed or not depends on how we set the NULL option for that column in the table.
  - If we do allow NULLs, we will be able to insert **ONLY ONE** of them.
- To create UNIQUE constraint, use the UNIQUE cluase:

```
CREATE TABLE Shippers
(
     ShipperID   INT          IDENTITY NOT NULL PRIMARY KEY,
     ShipperName VARCHAR(30) NOT NULL,
     Address     VARCHAR(30) NOT NULL,
     City        VARCHAR(25) NOT NULL,
     State       CHAR(2)     NOT NULL,
     Zip         VARCHAR(10) NOT NULL,
     PhoneNo     VARCHAR(14) NOT NULL UNIQUE
);
```

CHECK Constraints

- CHECK constraints are **NOT** restricted to a particular column.
  - They can be **table** related
  - They can check one column against another (in a single table) and the values are for the same row being updated or inserted.
  - They can check that any ***combination of column values***.
- The constraint is defined using the same rules in WHERE.

GOAL	SQL
Limit Month column to appropriate numbers	BETWEEN 1 AND 12
Proper SSN formatting	LIKE '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]'
Limit to a specifi clist	IN ('UPS', 'Fed Ex', 'USPS')
Price must be positive	UnitPrice >= 0
Reference another column in the same row	ShipDate >= OrderDate

- Example: Adding a modification to the Customers table to check for a valid date in the DateInSystem field (we can't have a date in the system that's in the future):

```
ALTER TABLE Customers
  ADD CONSTRAINT CN_CustomerDateInSystem
  CHECK ( DateInSystem <= GETDATE() );
```

- Now try to insert a record that violates the CHECK constraint:

```
INSERT INTO Customers
  (CustomerName, Address1, Address2, City, State, Zip, Contact,
   Phone, FedIDNo, DateInSystem)
VALUES
  ('Customer1', 'Address1', 'Add2', 'MyCity', 'NY', '55555',
   'No Contact', '553-1212', '930984954', '20491231');
```

- We will get an error:

```
The INSERT statement conflicted with the CHECK constraint
"CN_CustomerDateInSystem". The conflict occurred in database
"AdventureWorks2014", table "Customers", column 'DateInSystem'.
```

DEFAULT Constraints

- A DEFAULT constraint defines what to do when a new row is inserted that doesn't include data for the column.
- We can define it either as a literal value or as one of several system values such as GETDATE().
- The main points to understand about a DEFAULT constraint are that:
  - Defaults are used only in INSERT statements.
  - If any value is supplied in the INSERT, the default is not used.
  - If no value is supplied, the default will always be used.

- For example, a simpler version of shippers

```
CREATE TABLE SimpleShippers
(
   ShipperID     INT              IDENTITY NOT NULL PRIMARY KEY,
   ShipperName   VARCHAR(30)      NOT NULL,
   DateInSystem  SMALLDATETIME    NOT NULL DEFAULT GETDATE()
);
```

- Test how the default works by inserting a new record:

```
INSERT INTO SimpleShippers(ShipperName)
  VALUES('United Parcel Service');
```

```
ShipperID   ShipperName                    DateInSystem
----------- ------------------------------ -----------------------
1           United Parcel Service          2015-09-20 19:07:00
```

- Example: adding a DEFAULT constraint to an existing table

```
ALTER TABLE Customers
  ADD CONSTRAINT CN_CustomerAddress
  DEFAULT 'UNKNOWN' FOR Address1;
```

Disabling Constraints

- Sometimes we want to eliminate constraints, either just for a time or permanently.
- SQL Server also allows us to just deactivate a FOREIGN KEY or CHECK constraint while otherwise leaving it intact.
- The usual reason to disable a data integrity rule is because we already have bad data.
  - Data that's already in our database when we create the constraint
  - Data that we want to add after the constraint is already built
- SQL Server allows us to turn the integrity check off long enough to deal with the bad data we want to make an exception for, and then re-enable the integrity check later

Ignoring Bad Data (Create Constraint)

- Quite often, data rules are established after create the table.
- To add a constraint that **WON'T** apply to ***existing data***, we make use of the **WITH NOCHECK** option when we perform the ALTER TABLE statement that adds our constraint.
- For example,

```
SELECT CustomerName, Phone
FROM   Customers;
```

```
CustomerName          Phone
--------------------- ---------------
Billy Bob's Shoes     (360) 555-1234
MyCust                555-1212
```

- We want to add a constraint to control the Phone field with a length of 15.
- But there are some bad data alreay in the table.

- Add a constraint to control the formatting of the Phone field:

```
ALTER TABLE Customers
ADD CONSTRAINT CN_CustomerPhoneNo
CHECK
(Phone LIKE '([0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]');
```

```
The ALTER TABLE statement conflicted with the CHECK constraint
"CN_CustomerPhoneNo". The conflict occurred in database
"AdventureWorks2014", table "Customers", column 'Phone'.
```

- SQL Server does not create the constraint unless the existing data meets the constraint criteria.
- To get around this long enough to install the constraint:
  - We need to correct the existing data or
  - We must make use of the **WITH NOCHECK** option in the ALTER statement

- Add a constraint to control the formatting of the Phone field:

```
ALTER TABLE Customers
WITH NOCHECK
ADD CONSTRAINT CN_CustomerPhoneNo
CHECK (Phone LIKE '([0-9][0-9][0-9]) ...
```

```
Command(s) completed successfully.
```

- If we run the INSERT statement, the constraint works and the wrong data will be rejected:

```
INSERT INTO Customers
VALUES('MyCust', ..., '555-1212', GETDATE());
```

```
The INSERT statement conflicted with the CHECK constraint "CN_CustomerPhoneNo".
The conflict occurred in database "AdventureWorks2014", table "Customers",
column 'Phone'.
```

- The right format data will not be rejected:

```
INSERT INTO Customers
VALUES('MyCust', ..., '(800)555-1212', GETDATE());
```

The old data is retained for back reference, but any new data is restricted to meeting the new criteria.

```
CustomerName          Phone
--------------------- ---------------
Billy Bob's Shoes     (360) 555-1234
MyCust                555-1212
MyCust                (800) 555-1212
```

Temporarily Disabling an Existing Constraint

- The most common reason for which we make use of the WITH NOCHECK option - old data.
- Old data may also be data that we are importing from a legacy database or some other system.
- Certainly one way to do this would be to drop the constraint, add the desired data, and then add the constraint back using a WITH NOCHECK.
- Another option is we can run an ALTER statement with an option called NOCHECK that turns off the constraint in question

- Example: Temporarily disabling an existing constraint
- Disables the CHECK constraint:

```
ALTER TABLE Customers
  NOCHECK
  CONSTRAINT CN_CustomerPhoneNo;
```

- Now we can run that INSERT statement

```
INSERT INTO Customers
VALUES('MyCust', ..., '555-1212', GETDATE());
```

```
(1 row(s) affected)
```

- Active the constraint

```
ALTER TABLE Customers
  CHECK
  CONSTRAINT CN_CustomerPhoneNo;
```

```
CustomerName          Phone
--------------------- ---------------
Billy Bob's Shoes     (360) 555-1234
MyCust                555-1212
MyCust                (800) 555-1212
MyCust                555-1212
```

- SQL Server provides a procedure to indicate the status (enable/disable) of a constraintd, sp_helpconstraint.

The Identity Property

- SQL Server supports two built-in solutions to automatically generate keys: *the identity column property* and *the sequence object*.
- SQL Server allows us to define a property called identity for a column with any numeric type with a scale of zero (no fraction).
- This property generates values automatically upon INSERT based on a seed (first value) and an increment (step value) that are provided in the column's definition.
- Typically, we would use this property to generate surrogate keys, which are keys that are produced by the system and are not derived from the application data.

- Example: The Identity Property

```
CREATE TABLE T1
(
   Keycol      INT           NOT NULL IDENTITY(1, 1)
                             CONSTRAINT PK_T1 PRIMARY KEY,
   Datacol     VARCHAR(10)   NOT NULL
                             CONSTRAINT CHK_T1
                             CHECK (Datacol LIKE '[A-Za-z]%')
);
```

- The table contains a column called Keycol that is defined with an identity property using 1 as the seed and 1 as the increment.
- SQL Server produced the values for Keycol automatically.
- SQL Server allow only one identity column can be created per table.

```
INSERT INTO T1(Datacol) VALUES('AAAAA');
INSERT INTO T1(Datacol) VALUES('CCCCC');
INSERT INTO T1(Datacol) VALUES('BBBBB');
```

Keycol	Datacol
1	AAAAA
2	CCCCC
3	BBBBB

- When we query the table, naturally we can refer to the identity column by its name.
- SQL Server also provides a way to refer to the identity column by using the more generic form *$identity*.

```
SELECT $identity FROM T1;
```

```
Keycol
---------
1
2
3
```

- When we insert a new row into the table, SQL Server generates a new identity value based on the current identity value in the table and the increment.
- If we need to obtain the newly generated identity value - for example, to insert child rows into a referencing table - we query one of two functions called *@@identity* and *SCOPE_IDENTITY*.

- The *@@identity* function is an old feature that returns the last identity value generated by the session, regardless of scope.
- *SCOPE_IDENTITY* returns the last identity value generated by the session in ***the current scope***. For example,

```
INSERT INTO T1(Datacol) VALUES('AAAAA');
SELECT SCOPE_IDENTITY() AS NewKey
```

```
NewKey
----------
4
```

- *IDENT_CURRENT* provide the table name as input to obtain the current identity value in a table (regardless of session).
- Run the following code from a ***new session***:

```
SELECT
  SCOPE_IDENTITY()        AS [SCOPE_IDENTITY],
  @@identity              AS [@@identity],
  IDENT_CURRENT('T1')     AS [IDENT_CURRENT];
```

```
SCOPE_IDENTITY  @@identity      IDENT_CURRENT
--------------- --------------- ----------------
NULL            NULL            4
```

- The change to the current identity value in a table is not undone if the INSERT that generated the change fails or the transaction in which the statement runs is rolled back.
- For example, we insert two data in the T1 table, the first one fails, the second success

```
INSERT INTO T1(Datacol) VALUES('12345');
```

```
The INSERT statement conflicted with the CHECK constraint
"CHK_T1". The conflict occurred in database
"AdventureWorks2014", table "T1", column 'Datacol'.
```

```
INSERT INTO T1(Datacol) VALUES('EEEEE');
```

The current identity value in the table changed from 4 to 5, and this change was not undone because of the failure. This means that the next insert will produce the value 6.

Keycol	Datacol
1	AAAAA
2	CCCCC
3	BBBBB
4	AAAAA
6	EEEEE

- Another important aspect of the identity property is that we *cannot add* it to an existing column or *remove* it from an existing column. (define it by CREATE TABLE or ALTER TABLE)
- SQL Server does allow we to explicitly specify our own values for the identity column in INSERT statements.

```
SET IDENTITY_INSERT T1 ON;
INSERT INTO T1(Keycol, Datacol) VALUES(5, 'FFFFF');
  --Will not chage the current identity value (5 < 6 = current value)
SET IDENTITY_INSERT T1 OFF;
```

- SQL Server changes the current identity value in the table only if the explicit value provided for the identity column is higher than the current identity value in the table.

```
INSERT INTO T1(datacol) VALUES('GGGGG');
```

Keycol	Datacol
1	AAAAA
2	CCCCC
3	BBBBB
4	AAAAA
5	FFFFF
6	EEEEE
7	GGGGG

Sequence

- The sequence object is an alternative key-generating mechanism for identity.
- The sequence object is an independent object in the database, unlike identity which tied to a particular column.
- To create a sequence object, use the CREATE SEQUENCE.
  - The Sequence name
  - The data type, BIGINT by default
  - A minimum value (MINVALUE <val>) and a maximum value (MAXVALUE <val>) within type
  - Cycling or not (using the CYCLE option)
  - The starting value (START WITH <val>) and the increment (INCREMENET BY <val>).

- Example: An INT type, have a minimum value of 1 and a maximum value that is the maximum supported by the type, start with 1, increment by 1, and allow cycling.

```
CREATE SEQUENCE SeqOrderIDs AS INT
   MINVALUE 1
   CYCLE;
```

- We can change any of the other options with an ALTER SEQUENCE:
  - MINVAL <val>
  - MAXVAL <val>
  - INCREMENT BY <val>
  - CYCLE | NO CYCLE.
- Example:

```
ALTER SEQUENCE SeqOrderIDs
  NO CYCLE;
```

- To generate a new sequence value, we need to invoke the function NEXT VALUE FOR <sequence name>. For example,

```
SELECT NEXT VALUE FOR SeqOrderIDs;
```

```
-----------
1
```

- Unlike with identity, we didn't need to insert a row into a table in order to generate a new value.
- With sequences, we can store the result of the function in a variable, and then use it wherever we like.

```
DECLARE @NewID AS INT = NEXT VALUE FOR SeqOrderIDs;
INSERT INTO T1(Keycol, Datacol) VALUES(@NewID, 'a');
```

- Or we can specify the NEXT VALUE FOR function directly as part of our INSERT statement

```
INSERT INTO T1(Keycol, Datacol)
  VALUES(NEXT VALUE FOR SeqOrderIDs, 'b');
```

```
Keycol      Datacol
--------    --------
2           a
3           b
```

- Unlike with identity, we can generate new sequence values in an UPDATE statement

```
UPDATE T1
SET    Keycol = NEXT VALUE FOR SeqOrderIDs;
```

Keycol	Datacol
4	a
5	b

- To get information about the sequences, query a view called sys.sequences:

```
SELECT Current_Value
FROM   sys.sequences
WHERE  OBJECT_ID = OBJECT_ID('SeqOrderIDs');
```

Current_Value
5

- SQL Server extends its support for allows the use of the NEXT VALUE FOR function in a default constraint.

```
ALTER TABLE T1
  ADD CONSTRAINT DFT_T1_Keycol
      DEFAULT (NEXT VALUE FOR SeqOrderIDs)
      FOR Keycol;
```

```
INSERT INTO T1(Datacol) VALUES('c');
```

Keycol	Datacol
4	a
5	b
6	c

Sequence

- Note that like identity, the sequence object does not guarantee that we will have no gaps.
- If a new sequence value was generated by a transaction that failed, the sequence change is not undone.
- We can delete the sequence using DROP SEQUENCE
- For example,

```
DROP SEQUENCE SeqOrderIDs;
```

Summary

TOPIC	CONCEPT
Types of constraints	SQL Server provides entity constraints (which refer to columns within a single row), domain constraints (which compare columns across different rows), and referential integrity constraints (which compare columns between related tables).
Key constraints	Primary keys, foreign keys, and unique constraints form the building blocks of referential integrity.
CHECK constraints	Prevent bad data from entering your database from any source by placing CHECK constraints to reject rows that violate the rules.
DEFAULT constraints	Replace values not specified on INSERT with a given default.
Disabling constraints	Allow non-compliant data that either predates the constraint or must be inserted later.

Exercises

1. In the Accounting database, create a constraint making the FedIDNo column of the Customers table an alternate key.
2. In your business, an customer can never be her own friend. Create a constraint on the Customers table (Self-Referencing table) to prevent this.