MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 5: Implementing Data Integrity

Contents

Microsoft

Project Lead: Rich Rose
Instructional Designers: Rich Rose, Cheryl Hoople, Marilyn McGill
Instructional Software Design Engineers: Karl Dehmer, Carl Raebler, Rick Byham
Technical Lead: Karl Dehmer
Subject Matter Experts: Karl Dehmer, Carl Raebler, Rick Byham
Graphic Artist: Kirsten Larson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Copy Editor: Edward McKillop (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Support: Lori Walker (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: TestingTesting123
Classroom Automation: Lorrin Smith-Bates
Creative Director, Media/Sim Services: David Mahlmann
Web Development Lead: Lisa Pease
CD Build Specialist: Julie Challenger
Online Support: David Myka (S&T Consulting)
Localization Manager: Rick Terek
Operations Coordinator: John Williams
Manufacturing Support: Laura King; Kathy Hershey
Lead Product Manager, Release Management: Bo Galford
Lead Product Manager, Data Base: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
45 Minutes

Lab:
30 Minutes

This module provides students with an introduction to data integrity concepts, including the methods available for enforcing data integrity. The module then introduces constraints, which are the key method for ensuring data integrity. Also illustrated are various types of constraints. The module discusses the creation and implementation of constraints in detail, as well as the means to disable constraints when necessary.

The module discusses defaults and rules as an alternate way to enforce data integrity, although the emphasis remains on constraints. The module concludes with a comparison of the different data integrity methods.

In the lab, students define DEFAULT, CHECK, PRIMARY KEY, and FOREIGN KEY constraints.

After completing this module, students will be able to:

- Describe the types of data integrity.
- Describe the methods to enforce data integrity.
- Determine which constraint to use, and create constraints.
- Define and use DEFAULT, CHECK, PRIMARY KEY, UNIQUE, and FOREIGN KEY constraints.
- Disable constraints.
- Describe and use defaults and rules.
- Determine which data integrity enforcement methods to use.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2073A_05.ppt
- The C:\Moc\2073A\Demo\Demo05\D05_Ex.sql example file, which contains all of the example scripts from the module, unless otherwise noted in the module.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the lab.

Customization Information

This section identifies the lab setup requirements for a module and the configuration changes that occur on student computers during the labs. This information is provided to assist you in replicating or customizing Microsoft Official Curriculum (MOC) courseware.

Important The lab in this module is dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Classroom Setup Guide* for course 2073A, *Programming a Microsoft SQL Server 2000 Database*.

Lab Setup

The following section describes the setup requirement for the lab in this module.

Setup Requirement

The lab in this module requires the **ClassNorthwind** database to be in a state required for this lab. To prepare student computers to meet this requirement, perform one of the following actions:

- Complete the prior lab
- Execute the C:\Moc\2073A\Batches\Restore05.cmd batch file.

Warning If this course has been customized, students must execute the C:\Moc\2073A\Batches\Restore05.cmd batch file to ensure that the lab will function properly.

Lab Results

There are no configuration changes on student computers that affect replication or customization.

Module Strategy

Use the following strategy to present this module:

- Types of Data Integrity

  This topic should serve as a review for experienced students and as an introduction for less experienced students. Data integrity is defined, and the types of data integrity are introduced. Keep the teaching points general.

- Enforcing Data Integrity

  This topic describes the two methods to enforce data integrity: declarative data integrity and procedural data integrity. Point out that only the features of the declarative method—constraints, defaults, and rules—are covered in this module.

- Defining Constraints

  Emphasize that constraints are the preferred method of enforcing data integrity because they are ANSI-compliant, and describe how constraints can be used. Introduce the different types of constraints.

- Types of Constraints

  This section describes the types of constraints. Syntax, examples, and considerations for use define each constraint. Make sure that you describe the implications of the indexes that are created when users define constraints.

- Disabling Constraints

  This section describes how to disable constraint checking, whether you are creating a new constraint or disabling an existing one.

- Using Defaults and Rules

  This section discusses creating and implementing defaults and rules to enforce data integrity. However, it is important to make the point that using constraints is preferable to using defaults and rules, because defaults and rules are not ANSI-compliant.

- Deciding Which Enforcement Method to Use

  This section compares the methods of enforcing data integrity in terms of functionality and overhead costs. Point out that because triggers allow data to be changed before integrity has been verified, they can be resource intensive if problems are encountered. You should only use triggers when constraints alone do not provide the necessary functionality.

Overview

Topic Objective
To provide an overview of the module topics and objectives.

Lead-in
In this module, you will learn about the different types of data integrity and the features that ensure it.

- **Types of Data Integrity**
- **Enforcing Data Integrity**
- **Defining Constraints**
- **Types of Constraints**
- **Disabling Constraints**
- **Using Defaults and Rules**
- **Deciding Which Enforcement Method to Use**

This module begins with an introduction to data integrity concepts, including the methods available for enforcing data integrity. The module then introduces constraints, which are the key method for ensuring data integrity, and the various types of constraints. The module discusses the creation and implementation of constraints in detail, as well as the means to disable constraints when necessary.

The module discusses defaults and rules as an alternate way to enforce data integrity, although the emphasis remains on constraints. The module concludes with a comparison of the different data integrity methods.

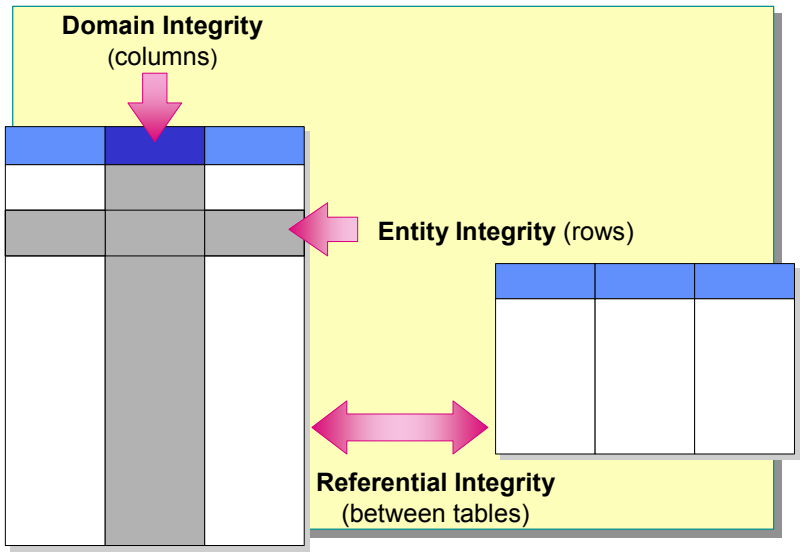
After completing this module, you will be able to:

- Describe the types of data integrity.
- Describe the methods to enforce data integrity.
- Determine which constraint to use and create constraints.
- Define and use DEFAULT, CHECK, PRIMARY KEY, UNIQUE, and FOREIGN KEY constraints.
- Disable constraint checking.
- Describe and use defaults and rules.
- Determine which data integrity enforcement methods to use.

Types of Data Integrity

Topic Objective
To introduce the different types of data integrity.

Lead-in
An important step in database planning is deciding the best way to enforce integrity of the data. Data integrity falls into these three categories.



An important step in database planning is deciding the best way to enforce the integrity of the data. Data integrity refers to the consistency and accuracy of data that is stored in a database. The different types of data integrity are as follows.

Delivery Tip
The types of data integrity referenced here represent basic relational database design. Keep the teaching points general. Consider giving an example from the **Northwind** database.

Domain Integrity

Domain (or column) integrity specifies a set of data values that are valid for a column and determines whether null values are allowed. Domain integrity is often enforced through the use of validity checking and can also be enforced by restricting the data type, format, or range of possible values allowed in a column.

Entity Integrity

Entity (or table) integrity requires that all rows in a table have a unique identifier, known as the *primary key value*. Whether the primary key value can be changed, or whether the whole row can be deleted, depends on the level of integrity required between the primary key and any other tables.

Referential Integrity

Referential integrity ensures that the relationships among the primary keys (in the referenced table) and *foreign keys* (in the referencing tables) are always maintained. A row in a referenced table cannot be deleted, nor the primary key changed, if a foreign key refers to the row, unless the cascade action is permitted. You can define referential integrity relationships within the same table or between separate tables.

Enforcing Data Integrity

Topic Objective
To introduce how SQL Server implements data integrity.

Lead-in
You can enforce data integrity through two methods.

- **Declarative Data Integrity**
  - Criteria defined in object definitions
  - SQL Server enforces automatically
  - Implement by using constraints, defaults, and rules
- **Procedural Data Integrity**
  - Criteria defined in script
  - Script enforces
  - Implement by using triggers and stored procedures

You can enforce data integrity through two methods: declarative data integrity or procedural data integrity.

Declarative Data Integrity

With *declarative* integrity, you define the criteria that the data must meet as part of an object definition, and then Microsoft® SQL Server™ 2000 automatically ensures that the data conforms to the criteria. The preferred method of implementing basic data integrity is to use declarative integrity. Consider the following facts about the declarative method:

- Declarative integrity is declared as part of the database definition, by using declarative constraints that you define directly on tables and columns.
- Implement declarative integrity by using constraints, defaults, and rules.

Delivery Tip
Mention that this module covers only declarative integrity features: constraints, defaults, and rules. Triggers and stored procedures are covered in later modules.

Procedural Data Integrity

With *procedural* integrity, you write scripts that define both the criteria that data must meet and enforce the criteria. You should limit your use of procedural integrity to more complicated business logic and exceptions. For example, use procedural integrity when you want to have a cascading delete. The following facts apply to procedural integrity:

- Procedural integrity can be implemented on the client or the server by using other programming languages and tools.
- Implement procedural integrity by using triggers and stored procedures.

◆ Defining Constraints

Topic Objective
To introduce enforcing data integrity by using constraints.

Lead-in
Constraints are the preferred method for enforcing data integrity.

- **Determining Which Type of Constraint to Use**
- **Creating Constraints**
- **Considerations for Using Constraints**

Constraints are the preferred method of enforcing data integrity. This section discusses how to determine the type of constraint to use, what type of data integrity that each type of constraint enforces, and how to define constraints.

Determining Which Type of Constraint to Use

Topic Objective
To introduce the different types of constraints and how to use them to implement data integrity.

Lead-in
Different types of constraints ensure that valid data values are entered in columns and that relationships are maintained between tables.

Type of integrity	Constraint type
Domain	DEFAULT
	CHECK
	REFERENTIAL
Entity	PRIMARY KEY
	UNIQUE
Referential	FOREIGN KEY
	CHECK

Key Point
Emphasize that constraints are ANSI-compliant.

Constraints are an ANSI-standard method of enforcing data integrity. Each type of data integrity — domain, entity, and referential — is enforced with separate types of constraints. Constraints ensure that valid data values are entered in columns and that relationships are maintained between tables. The following table describes the different types of constraints.

Type of integrity	Constraint type	Description
Domain	DEFAULT	Specifies the value that will be provided for the column when a value has not been explicitly supplied in an INSERT statement.
	CHECK	Specifies data values that are acceptable in a column.
	REFERENTIAL	Specifies the data values that are acceptable to update, based on values in a column in another table.
Entity	PRIMARY KEY	Uniquely identifies each row—ensures that users do not enter duplicate values and that an index is created to enhance performance. Null values are not allowed.
	UNIQUE	Prevents duplication of alternate (non-primary) keys, and ensures that an index is created to enhance performance. Null values are allowed.
Referential	FOREIGN KEY	Defines a column or combination of columns with values that match the primary key of the same or another table.
	CHECK	Specifies the data values that are acceptable in a column based on values in other columns in the same table.

Creating Constraints

Topic Objective
To introduce the syntax for defining constraints.

Lead-in
Constraints are implemented by using the CREATE TABLE or ALTER TABLE statement.

- **Use CREATE TABLE or ALTER TABLE**
- **Can Add Constraints to a Table with Existing Data**
- **Can Place Constraints on Single or Multiple Columns**
  - Single column, called column-level constraint
  - Multiple columns, called table-level constraint

You create constraints by using the CREATE TABLE or ALTER TABLE statement.

Delivery Tip
Point out that the term table-level constraint refers to any multicolumn constraint.

You can add constraints to a table with existing data, and you can place constraints on single or multiple columns:

- If the constraint applies to a single column, it is called a *column-level* constraint.
- If a constraint references multiple columns, it is called a *table-level* constraint, even if it does not reference all columns in the table.

Partial Syntax

```
CREATE TABLE table_name
    ( { < column_definition >
     | < table_constraint > } [ ,...n ])

< column_definition > ::= { column_name data_type }
    [ [ DEFAULT constant_expression ]
    [ < column_constraint > ] [ ,..n]

< column_constraint > ::=
  [ CONSTRAINT constraint_name ]
        | [ { PRIMARY KEY | UNIQUE }
            [ CLUSTERED | NONCLUSTERED ] ]

    | [ [ FOREIGN KEY ]
            REFERENCES ref_table [ ( ref_column ) ]
            [ ON DELETE { CASCADE | NO ACTION } ]
            [ ON UPDATE { CASCADE | NO ACTION } ]]
        | CHECK ( logical_expression ) }
```

Delivery Tips
Point out that the syntax is split into column-level and table-level constraints.

Advise students to create the base table first and then add constraints later, which simplifies the process of defining a table.

```
< table_constraint > ::=
   [ CONSTRAINT constraint_name ]
    { [ { PRIMARY KEY | UNIQUE }
        [ CLUSTERED | NONCLUSTERED ]
        { ( column [ ASC | DESC ] [ ,...n ] ) } ]
    | FOREIGN KEY
        [ ( column [ ,...n ] ) ]
        REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
        [ ON DELETE { CASCADE | NO ACTION } ]
        [ ON UPDATE { CASCADE | NO ACTION } ]
    | CHECK ( search_conditions ) }
```

Example

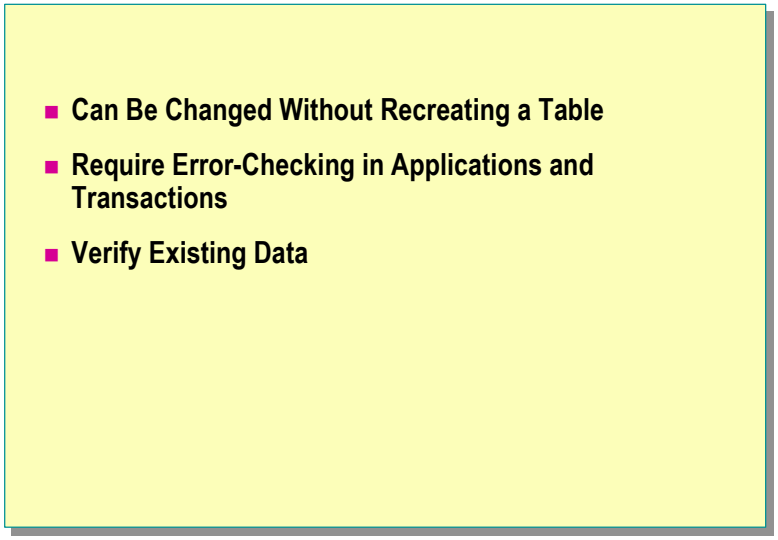
This example creates the **Products** table, defines columns, and defines constraints at both the column and table level.

```
USE Northwind
CREATE TABLE dbo.Products
(
   ProductID      int IDENTITY (1,1) NOT NULL,
   ProductName    nvarchar (40) NOT NULL,
   SupplierID     int      NULL,
   CategoryID     int      NULL,
   QuantityPerUnit nvarchar (20) NULL,
   UnitPrice      money    NULL     CONSTRAINT DF_Products_UnitPrice  DEFAULT(0),
   UnitsInStock   smallint NULL     CONSTRAINT DF_Products_UnitsInStock DEFAULT(0),
   UnitsOnOrder   smallint NULL     CONSTRAINT DF_Products_UnitsOnOrder DEFAULT(0),
   ReorderLevel   smallint NULL     CONSTRAINT DF_Products_ReorderLevel DEFAULT(0),
   Discontinued   bit      NOT NULL CONSTRAINT DF_Products_Discontinued DEFAULT(0),

   CONSTRAINT PK_Products PRIMARY KEY CLUSTERED (ProductID),

   CONSTRAINT FK_Products_Categories FOREIGN KEY (CategoryID)
      REFERENCES dbo.Categories (CategoryID) ON UPDATE CASCADE,
   CONSTRAINT FK_Products_Suppliers  FOREIGN KEY (SupplierID)
      REFERENCES dbo.Suppliers  (SupplierID) ON DELETE CASCADE,

   CONSTRAINT CK_Products_UnitPrice CHECK (UnitPrice >= 0),
   CONSTRAINT CK_ReorderLevel       CHECK (ReorderLevel >= 0),
   CONSTRAINT CK_UnitsInStock       CHECK (UnitsInStock >= 0),
   CONSTRAINT CK_UnitsOnOrder       CHECK (UnitsOnOrder >= 0)
)
GO
```

Considerations for Using Constraints

Topic Objective
To describe some of the considerations for using constraints.

Lead-in
Consider these facts when you implement or modify constraints.



Delivery Tip
Demonstrate that SQL Server creates complicated, system-generated constraint names.

Consider the following facts when you implement or modify constraints:

- You can create, change, and drop constraints without having to drop and recreate a table.
- You must build error-checking logic into your applications and transactions to test whether a constraint has been violated.
- SQL Server verifies existing data when you add a constraint to a table.

You should specify names for constraints when you create them, because SQL Server provides complicated, system-generated names. Names must be unique to the database object owner and follow the rules for SQL Server identifiers.

For help with constraints, execute the **sp_helpconstraint** or **sp_help** system stored procedure, or query information schema views, such as **check_constraints**, **referential_constraints**, and **table_constraints**.

The following system tables store constraint definitions: **syscomments**, **sysreferences**, and **sysconstraints**.

Types of Constraints

Topic Objective
To describe the types of constraints.

Lead-in
This section describes the types of constraints.

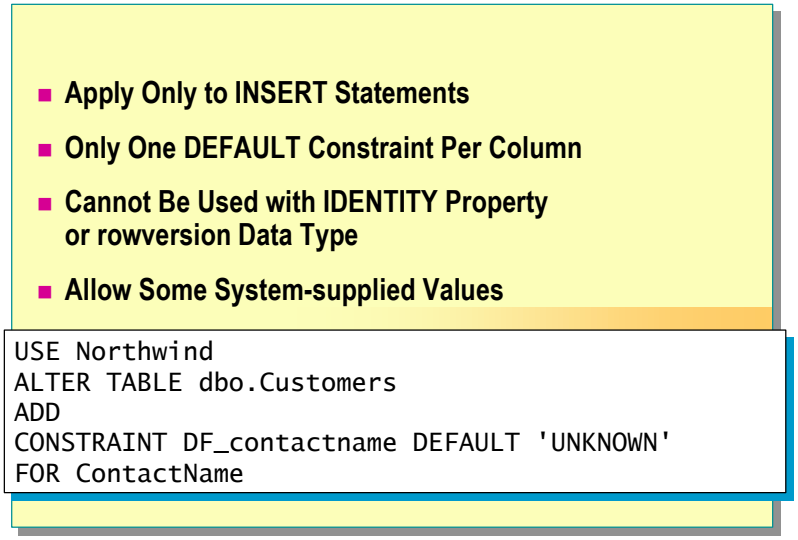
- DEFAULT Constraints
- CHECK Constraints
- PRIMARY KEY Constraints
- UNIQUE Constraints
- FOREIGN KEY Constraints
- Cascading Referential Integrity

This section describes the types of constraints. Syntax, examples, and considerations for use define each constraint.

DEFAULT Constraints

Topic Objective
To introduce the DEFAULT constraint.

Lead-in
The DEFAULT constraint enforces domain integrity.



A DEFAULT constraint enters a value in a column when one is not specified in an INSERT statement. DEFAULT constraints enforce domain integrity.

Partial Syntax

[CONSTRAINT *constraint_name*]
DEFAULT *constant_expression*

Example

This example adds a DEFAULT constraint that inserts the UNKNOWN value in the **dbo.Customers** table if a contact name is not provided.

```
USE Northwind
ALTER TABLE dbo.Customers
ADD
CONSTRAINT DF_contactname DEFAULT 'UNKNOWN' FOR ContactName
```

Consider the following facts when you apply a DEFAULT constraint:

- It verifies existing data in the table.
- It applies only to INSERT statements.
- Only one DEFAULT constraint can be defined per column.
- It cannot be placed on columns with the **Identity** property or on columns with the **rowversion** data type.
- It allows some system-supplied values—USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_TIMESTAMP—to be specified rather than user-defined values. These system-supplied values can be useful in providing a record of the users who have been inserting data.

CHECK Constraints

Topic Objective
To introduce the CHECK constraint.

Lead-in
A CHECK constraint restricts the data entered into a column to specific values.

- **Are Used with INSERT and UPDATE Statements**
- **Can Reference Other Columns in the Same Table**
- **Cannot:**
  - Be used with the **rowversion** data type
  - Contain subqueries

```
USE Northwind
ALTER TABLE dbo.Employees
ADD
CONSTRAINT CK_birthdate
CHECK (BirthDate > '01-01-1900' AND BirthDate <
getdate())
```

**********ILLEGAL FOR NON-TRAINER USE**********

A CHECK constraint restricts the data that users can enter into a particular column to specific values. CHECK constraints are similar to WHERE clauses in that you can specify the conditions under which data will be accepted.

Partial Syntax

[CONSTRAINT *constraint_name*]
CHECK **(***logical_expression***)**

Example

This example adds a CHECK constraint to ensure that a birth date conforms to an acceptable range of dates.

```
USE Northwind
ALTER TABLE dbo.Employees
ADD
CONSTRAINT CK_birthdate
CHECK (BirthDate > '01-01-1900' AND BirthDate < getdate())
```

Consider the following facts when you apply a CHECK constraint:

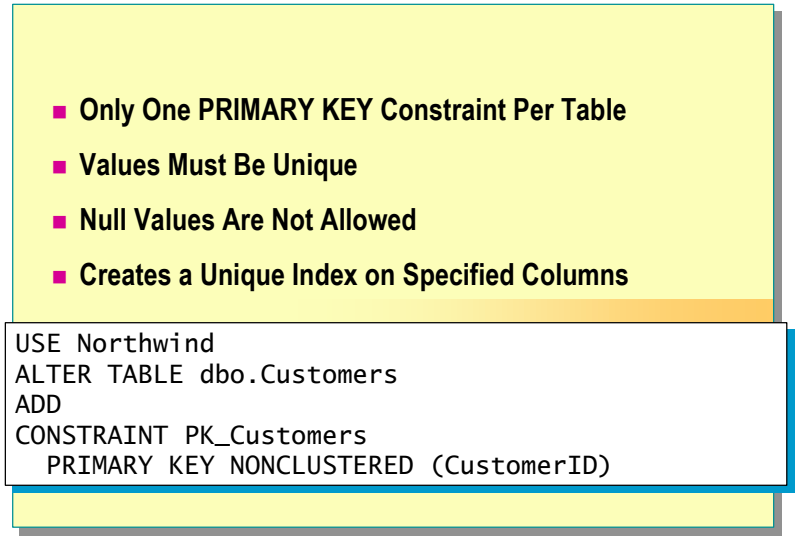
- It verifies data every time that you execute an INSERT or UPDATE statement.
- It can reference other columns in the same table.

  For example, a **salary** column could reference a value in a **job_grade** column.
- It cannot be placed on columns with the **rowversion** data type.
- It cannot contain subqueries.
- If any data violates the CHECK constraint, you can execute the DBCC CHECKCONSTRAINTS statement to return the violating rows.

PRIMARY KEY Constraints

Topic Objective
To introduce PRIMARY KEY constraints.

Lead-in
PRIMARY KEY constraints enforce entity integrity.



**********ILLEGAL FOR NON-TRAINER USE**********

A PRIMARY KEY constraint defines a primary key on a table that uniquely identifies a row. It enforces entity integrity.

Partial Syntax

[CONSTRAINT *constraint_name*]
PRIMARY KEY [CLUSTERED | NONCLUSTERED]
{ **(** *column*[**,**...*n*] **)** }

Example

This example adds a constraint that specifies that the primary key value of the **dbo.Customers** table is the customer identification and indicates that a nonclustered index will be created to enforce the constraint.

```
USE Northwind
ALTER TABLE dbo.Customers
ADD
CONSTRAINT PK_Customers
   PRIMARY KEY NONCLUSTERED (CustomerID)
```

Key Points
The PRIMARY KEY constraint is always unique, and it does not allow null values.

The PRIMARY KEY constraint always creates an index.

Consider the following facts when you apply a PRIMARY KEY constraint:

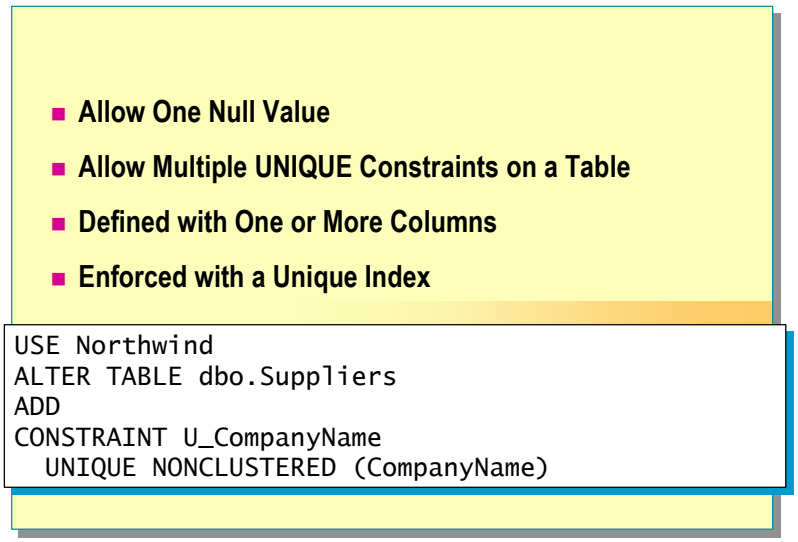
- Only one PRIMARY KEY constraint can be defined per table.
- The values entered must be unique.
- Null values are not allowed.
- It creates a unique index on the specified columns. You can specify a clustered or nonclustered index (clustered is the default if it does not already exist).

Note The index created for a PRIMARY KEY constraint cannot be dropped directly. It is dropped when you drop the constraint.

UNIQUE Constraints

Topic Objective
To introduce UNIQUE constraints.

Lead-in
A UNIQUE constraint specifies that two rows in a column cannot have the same value.



**********ILLEGAL FOR NON-TRAINER USE**********

A UNIQUE constraint specifies that two rows in a column cannot have the same value. This constraint enforces entity integrity with a unique index.

A UNIQUE constraint is helpful when you already have a primary key, such as an employee number, but you want to guarantee that other identifiers, such as an employee's driver's license number, are also unique.

Partial Syntax

[CONSTRAINT *constraint_name*]
UNIQUE [CLUSTERED | NONCLUSTERED]
{ (*column*[,...*n*]) }

Example

This example creates a UNIQUE constraint on the company name in the **dbo.Suppliers** table.

```
USE Northwind
ALTER TABLE dbo.Suppliers
ADD
CONSTRAINT U_CompanyName
   UNIQUE NONCLUSTERED (CompanyName)
```

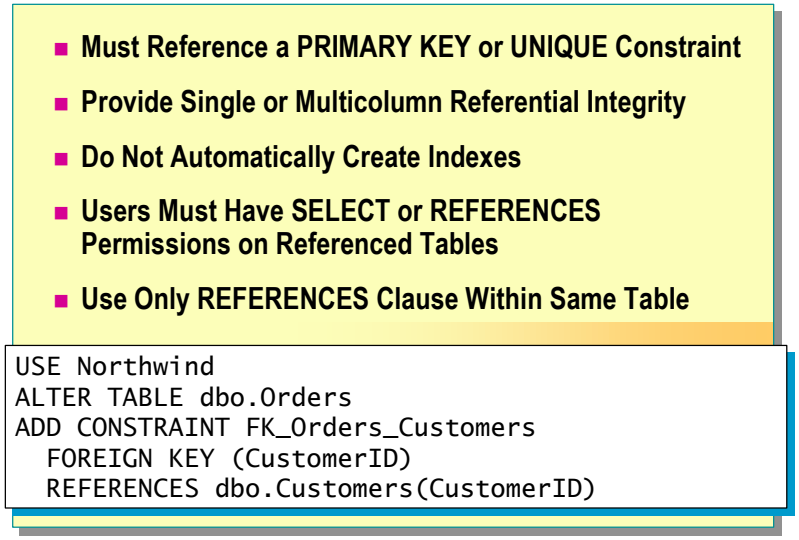
Consider the following facts when you apply a UNIQUE constraint:

- It can allow one null value.
- You can place multiple UNIQUE constraints on a table.
- You can apply the UNIQUE constraint to one or more columns that must have unique values, but are not the primary key of a table.
- The UNIQUE constraint is enforced through the creation of a unique index on the specified column or columns.

FOREIGN KEY Constraints

Topic Objective
To introduce the FOREIGN KEY constraint.

Lead-in
A FOREIGN KEY constraint enforces referential integrity.



**********ILLEGAL FOR NON-TRAINER USE**********

A FOREIGN KEY constraint enforces referential integrity. The FOREIGN KEY constraint defines a reference to a column with a PRIMARY KEY or UNIQUE constraint in the same, or another table.

Partial Syntax

[CONSTRAINT *constraint_name*]
[FOREIGN KEY] [**(***column*[,…*n*]**)**]
REFERENCES *ref_table* [**(***ref_column* [,…*n*]**)**].

Example

This example uses a FOREIGN KEY constraint to ensure that customer identification in the **dbo.Orders** table is associated with a valid identification in the **dbo.Customers** table.

```
USE Northwind
ALTER TABLE dbo.Orders
ADD CONSTRAINT FK_Orders_Customers
   FOREIGN KEY (CustomerID)
   REFERENCES dbo.Customers(CustomerID)
```

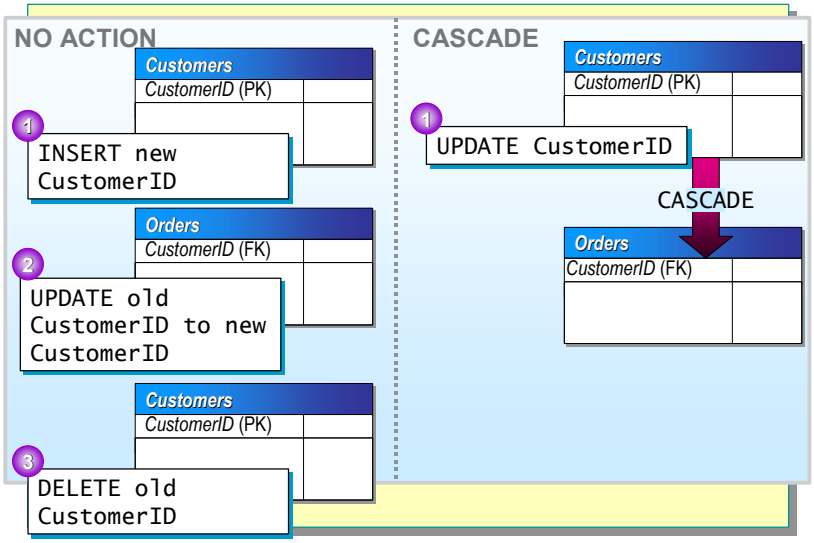
Consider the following facts and guidelines when you apply a FOREIGN KEY constraint:

- It provides single or multicolumn referential integrity. The number of columns and data types that are specified in the FOREIGN KEY statement must match the number of columns and data types in the REFERENCES clause.
- Unlike PRIMARY KEY or UNIQUE constraints, FOREIGN KEY constraints do not create indexes automatically. However, if you will be using many joins in your database, you should create an index for the FOREIGN KEY to improve join performance.
- To modify data, users must have SELECT or REFERENCES permissions on other tables that are referenced with a FOREIGN KEY constraint.
- You can use only the REFERENCES clause without the FOREIGN KEY clause when you reference a column in the same table.

Cascading Referential Integrity

Topic Objective
To describe how to cascade referential integrity.

Lead-in
Cascading referential integrity automatically propagates changes to the database.



The FOREIGN KEY constraint includes a CASCADE option that allows any change to a column value that defines a UNIQUE or PRIMARY KEY constraint to automatically propagate the change to the foreign key value. This action is referred to as *cascading referential integrity*.

The REFERENCES clauses of the CREATE TABLE and ALTER TABLE statements support ON DELETE and ON UPDATE clauses. These clauses allow you to specify the CASCADE or NO ACTION option.

Partial Syntax

```
[CONSTRAINT constraint_name]
    [FOREIGN KEY] [(column[,…n])]
        REFERENCES ref_table [(ref_column [,…n])].
        [ ON DELETE { CASCADE | NO ACTION } ]
        [ ON UPDATE { CASCADE | NO ACTION } ]
```

NO ACTION specifies that any attempt to delete or update a key referenced by foreign keys in other tables raises an error and the change is rolled back. NO ACTION is the default.

If CASCADE is defined and a row is changed in the parent table, the corresponding row is then changed in the referencing table.

For example, in the **Northwind** database, the **Orders** table has a referential relationship with the **Customers** table; specifically, the **Orders.CustomerID** foreign key references the **Customers.CustomerID** primary key.

If an UPDATE statement is executed on **CustomerID** in the **Customers** table, and an ON UPDATE CASCADE action is specified for **Orders.CustomerID**, SQL Server checks for one or more dependent rows in the **Orders** table. If any exist, it updates the dependent rows in the **Orders** table, as well as the row referenced in the **Customers** table.

Consider these factors when applying the CASCADE option:

- It is possible to combine CASCADE and NO ACTION on tables that have referential relationships with one another. If SQL Server encounters NO ACTION, it terminates and rolls back related CASCADE actions.

  When a DELETE statement causes a combination of CASCADE and NO ACTION actions, all the CASCADE actions are applied before SQL Server checks for any NO ACTION.

- CASCADE cannot be specified for any foreign key or primary key columns that are defined with a **rowversion** column.

Disabling Constraints

Topic Objective
To describe the ways to disable constraints.

Lead-in
For reasons of performance, it is sometimes advisable to disable constraints.

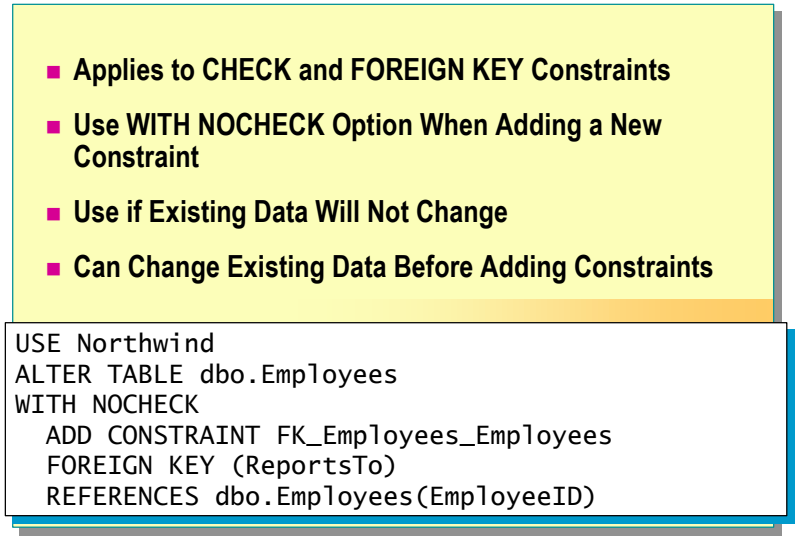
- Disabling Constraint Checking on Existing Data
- Disabling Constraint Checking When Loading New Data

For reasons of performance, it is sometimes advisable to disable constraints. For example, it is more efficient to allow large batch operations to process before enabling constraints. This section describes how to disable constraint checking, whether you are creating a new constraint or disabling an existing one.

Disabling Constraint Checking on Existing Data

Topic Objective
To introduce how to disable constraints.

Lead-in
You can disable constraint checking when you add a constraint to a table.



When you define a constraint on a table that already contains data, SQL Server checks the data automatically to verify that it meets the constraint requirements. However, you can disable constraint checking on existing data when you add a constraint to the table.

Consider the following guidelines for disabling constraint checking on existing data:

- You can disable only CHECK and FOREIGN KEY constraints. Other constraints must be dropped and then added again.
- To disable constraint checking when you add a CHECK or FOREIGN KEY constraint to a table with existing data, include the WITH NOCHECK option in the ALTER TABLE statement.
- Use the WITH NOCHECK option if existing data will not change. Data must conform to CHECK constraints if the data is updated.
- Be certain that it is appropriate to disable constraint checking. You can execute a query to change existing data before you decide to add a constraint.

Partial Syntax

ALTER TABLE *table*
[WITH CHECK | WITH NOCHECK]
ADD CONSTRAINT *constraint*

[FOREIGN KEY] [**(***column*[,…*n*]**)**]
REFERENCES *ref_table* [**(***ref_col* [,…*n*]**)**]
[CHECK **(***search_conditions***)**]

Example

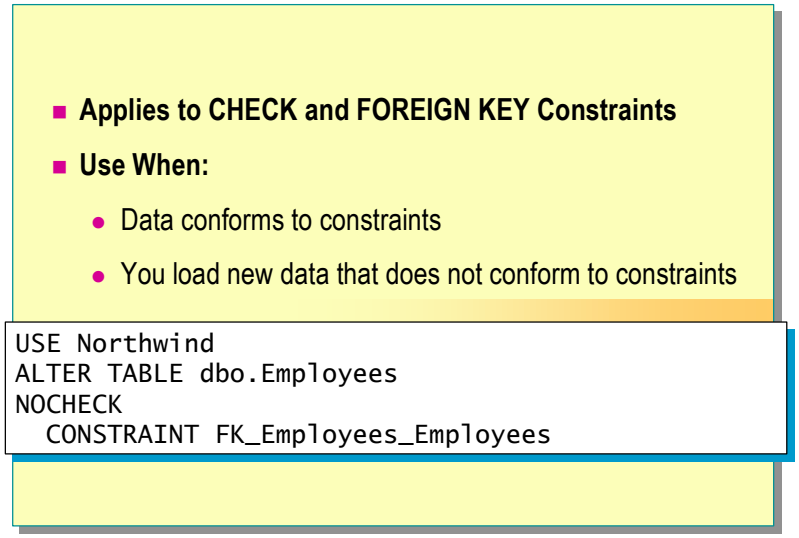
In this example, you add a FOREIGN KEY constraint that verifies that all employees are associated with a valid manager. The constraint is not enforced on existing data at the time that the constraint is added.

```
USE Northwind
ALTER TABLE dbo.Employees
WITH NOCHECK
   ADD CONSTRAINT FK_Employees_Employees
   FOREIGN KEY (ReportsTo)
   REFERENCES dbo.Employees(EmployeeID)
```

Disabling Constraint Checking When Loading New Data

Topic Objective
To describe how to disable constraint checking when you load new data.

Lead-in
This feature is limited to CHECK and FOREIGN KEY constraints.



**********ILLEGAL FOR NON-TRAINER USE**********

You can disable constraint checking on existing CHECK and FOREIGN KEY constraints so that any data that you modify or add to the table is not checked against the constraint.

To avoid the costs of constraint checking, you might want to disable constraints when:

- You already have ensured that the data conforms to the constraints.
- You want to load data that does not conform to the constraints. Later, you can execute queries to change the data and then re-enable the constraints.

Important Disabling constraints on one table does not affect constraints on other tables that reference the original table. Updates to a table still can generate constraint violation errors.

Enabling a constraint that has been disabled requires executing another ALTER TABLE statement that contains either a CHECK or CHECK ALL clause.

Partial Syntax

ALTER TABLE *table*
{CHECK | NOCHECK} CONSTRAINT
{ALL | *constraint*[,...*n*]}

Example

This example disables the **FK_Employees_Employees** constraint. It can be re-enabled by executing another ALTER TABLE statement with the CHECK clause.

```
USE Northwind
ALTER TABLE dbo.Employees
NOCHECK
   CONSTRAINT FK_Employees_Employees
```

To determine whether a constraint is enabled or disabled on a table, execute the **sp_help** system stored procedure, or use the **CnstIsDisabled** property in the OBJECTPROPERTY function.

Using Defaults and Rules

Topic Objective
To specify how to create defaults and rules.

Lead-in
Defaults and rules are two additional methods of enforcing data integrity.

- **As Independent Objects They:**
  - Are defined once
  - Can be bound to one or more columns or user-defined data types

```
CREATE DEFAULT phone_no_default
  AS '(000)000-0000'
GO
EXEC sp_bindefault phone_no_default,
  'Customers.Phone'
```

```
CREATE RULE regioncode_rule
  AS @regioncode IN ('IA', 'IL', 'KS', 'MO')
GO
EXEC sp_bindrule regioncode_rule,
  'Customers.Region'
```


Defaults and rules are objects that can be bound to one or more columns or user-defined data types, making it possible to define them once and use them repeatedly. A disadvantage to using defaults and rules is that they are not ANSI-compliant.

Creating a Default

If a value is not specified when you insert data, a default specifies one for the column to which the object is bound. Consider these facts before you create defaults:

Key Point
You cannot use a default constraint on a column with a user-defined data type if a default is already bound to the data type or column.

- Any rules that are bound to the column and the data types validate the value of a default.
- Any CHECK constraints on the column must validate the value of a default.
- You cannot create a DEFAULT constraint on a column that is defined with a user-defined data type if a default is already bound to the data type or column.

Syntax

CREATE DEFAULT *default*
  AS *constant_expression*

Binding a Default

After you create a default, you must bind it to a column or user-defined data type by executing the **sp_bindefault** system stored procedure. To detach a default, execute the **sp_unbindefault** system stored procedure.

Example

This example inserts a placeholder phone number in the correct format until the actual phone number can be supplied.

```
USE Northwind
GO
CREATE DEFAULT phone_no_default
 AS '(000)000-0000'
GO
EXEC sp_bindefault phone_no_default, 'Customers.Phone'
```

Creating a Rule

Rules specify the acceptable values that you can insert into a column. They ensure that data falls within a specified range of values, matches a particular pattern, or matches entries in a specified list. Consider these facts about rules:

- A rule definition can contain any expression that is valid in a WHERE clause.
- A column or user-defined data type can have only one rule that is bound to it.

Syntax

CREATE RULE *rule*
  AS *condition_expression*

Binding a Rule

After you create a rule, you must bind it to a column or user-defined data type by executing the **sp_bindrule** system stored procedure. To detach a rule, execute the **sp_unbindrule** system stored procedure.

Example

In this example, the rule ensures that only specified states are allowed.

```
USE Northwind
GO
CREATE RULE regioncode_rule
 AS @regioncode IN ('IA', 'IL', 'KS', 'MO')
GO
EXEC sp_bindrule regioncode_rule, 'Customers.Region'
```

Dropping a Default or Rule

The DROP statement removes a default or rule from the database.

Syntax

DROP DEFAULT *default* [,*...n*]

Syntax

DROP RULE *rule* [, *...n*]

Deciding Which Enforcement Method to Use

Topic Objective
To show the advantages and disadvantages of various data integrity components.

Lead-in
You should consider functionality and performance costs when you determine which methods to use to enforce data integrity.

Data integrity components	Functionality	Performance costs	Before or after modification
Constraints	Medium	Low	Before
Defaults and rules	Low	Low	Before
Triggers	High	Medium-High	After
Data types, Null/Not Null	Low	Low	Before

**********ILLEGAL FOR NON-TRAINER USE**********

Key Points
Use constraints if possible. Use defaults and rules if you need independent objects. Use triggers only when complex business logic is required.

You should consider functionality and performance costs when you determine which methods to use to enforce data integrity:

- It is best to use declarative integrity for fundamental integrity logic, such as when enforcing valid values and maintaining the relationships between tables.
- If you want to maintain complex redundant data that is not part of a primary or foreign key relationship, you must use triggers or stored procedures.

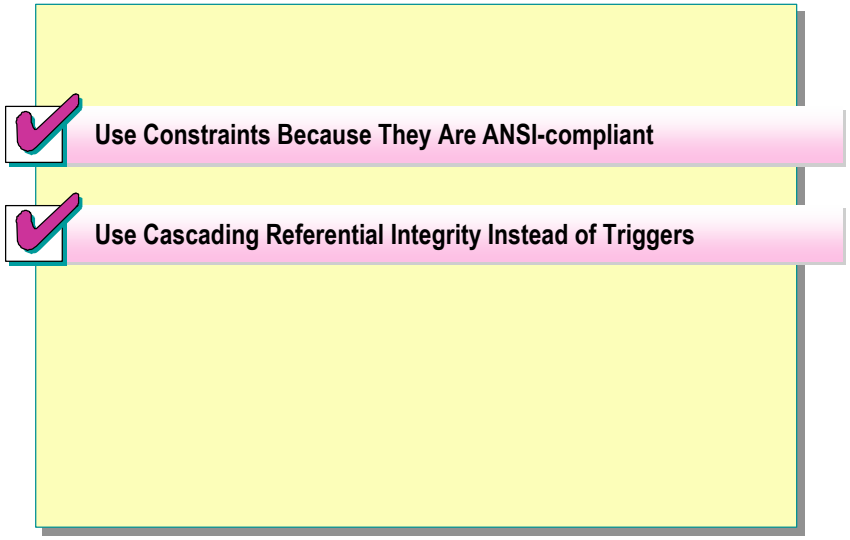
  However, because triggers do not fire until a modification occurs, error checking happens after the statement is completed. When a trigger detects a violation, it must undo the changes.

Data integrity component	Impact	Functionality	Performance costs	Before or after modification
Constraints	Define with a table and validate the data before a transaction begins, resulting in better performance.	Medium	Low	Before
Defaults and rules	Implement data integrity as separate objects that can be associated with one or more tables.	Low	Low	Before
Triggers	Provide additional functionality, such as cascading and complex application logic. Any modifications must be rolled back.	High	Medium-High	After (except for INSTEAD OF triggers)
Data types, Null/Not Null	Provides the lowest level of data integrity. Implemented for each column when the table is created. Data is validated before a transaction begins.	Low	Low	Before

Recommended Practices

Topic Objective
To present the recommended practices for implementing data integrity.

Lead-in
These recommended practices should help you when implementing data integrity.



The following recommended practices should help you implement data integrity:

- Use constraints because they are ANSI-compliant and are supported by third-party development tools.
- Use cascading referential integrity instead of triggers.

Additional information on the following topics is available in SQL Server Books Online.

Topic	Search on
Developing databases	"data integrity"
Clustered and nonclustered indexes	"creating an index"
Cascading referential integrity	"cascading referential integrity restraints"
Triggers	"triggers"

Lab A: Implementing Data Integrity

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will add DEFAULT, CHECK, PRIMARY KEY, and FOREIGN KEY constraints to certain tables.

**********ILLEGAL FOR NON-TRAINER USE**********

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Define and use DEFAULT and CHECK constraints to enforce domain integrity.
- Define and use PRIMARY KEY and FOREIGN KEY constraints to enforce entity and referential integrity.
- Create and use Microsoft SQL Server 2000 rules and defaults.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2073A\Labfiles\L05.
- Answer files for this lab, which are located in C:\Moc\2073A\Labfiles\L05\Answers.

Lab Setup

To complete this lab, you must have either:

- Completed the prior lab, or
- Executed the C:\Moc\2073A\Batches\Restore05.cmd batch file.

  This command file restores the **ClassNorthwind** database to a state required for this lab.

For More Information

If you require help with executing files, search SQL Query Analyzer Help for "Execute a query".

Other resources that you can use include:

- The **Northwind** database schema.
- SQL Server Books Online.

Scenario

The organization of the classroom is meant to simulate that of a worldwide trading firm named Northwind Traders. Its fictitious domain name is nwtraders.msft. The primary DNS server for nwtraders.msft is the instructor computer, which has an Internet Protocol (IP) address of 192.168.*x*.200 (where *x* is the assigned classroom number). The name of the instructor computer is London.

The following table provides the user name, computer name, and IP address for each student computer in the fictitious **nwtraders.msft** domain. Find the user name for your computer, and make a note of it.

User name	Computer name	IP address
SQLAdmin1	Vancouver	192.168.*x*.1
SQLAdmin2	Denver	192.168.*x*.2
SQLAdmin3	Perth	192.168.*x*.3
SQLAdmin4	Brisbane	192.168.*x*.4
SQLAdmin5	Lisbon	192.168.*x*.5
SQLAdmin6	Bonn	192.168.*x*.6
SQLAdmin7	Lima	192.168.*x*.7
SQLAdmin8	Santiago	192.168.*x*.8
SQLAdmin9	Bangalore	192.168.*x*.9
SQLAdmin10	Singapore	192.168.*x*.10
SQLAdmin11	Casablanca	192.168.*x*.11
SQLAdmin12	Tunis	192.168.*x*.12
SQLAdmin13	Acapulco	192.168.*x*.13
SQLAdmin14	Miami	192.168.*x*.14
SQLAdmin15	Auckland	192.168.*x*.15
SQLAdmin16	Suva	192.168.*x*.16
SQLAdmin17	Stockholm	192.168.*x*.17
SQLAdmin18	Moscow	192.168.*x*.18
SQLAdmin19	Caracas	192.168.*x*.19
SQLAdmin20	Montevideo	192.168.*x*.20
SQLAdmin21	Manila	192.168.*x*.21
SQLAdmin22	Tokyo	192.168.*x*.22
SQLAdmin23	Khartoum	192.168.*x*.23
SQLAdmin24	Nairobi	192.168.*x*.24

Estimated time to complete this lab: 30 minutes

Exercise 1
Defining DEFAULT Constraints

In this exercise, you will add DEFAULT constraints to the **Employees** table in the **ClassNorthwind** database.

► To define a DEFAULT constraint

In this procedure, you will execute a script that creates a default for the **Region** column in the **Employees** table, and then you will modify the same script to change the default region.

1. Log on to the **NWTraders** classroom domain by using the information in the following table.

Option	Value
User name	**SQLAdmin*x*** (where *x* corresponds to your computer name as designated in the **nwtraders.msft** classroom domain)
Password	**password**

2. Open SQL Query Analyzer and, if requested, log in to the (local) server with Microsoft Windows® authentication.

   You have permission to log in to and administer SQL Server because you are logged as **SQLAdmin*x***, which is a member of the Microsoft Windows 2000 local group, Administrators. All members of this group are automatically mapped to the SQL Server **sysadmin** role.

3. In the **DB** list, click **ClassNorthwind**.

4. Open Labfiles\L05\DefConst.sql, and then review and execute it.

   This script will create a DEFAULT constraint that adds NY (New York) as the default for the **Region** column in the **Employees** table.

5. Execute **sp_helpconstraint** ***tablename*** and **sp_help** ***constraintname*** to view information on the DEFAULT constraint that you just created.

```
EXEC SP_HELPCONSTRAINT Employees

EXEC SP_HELP DF_Region
```

6. Modify Labfiles\L05\DefConst.sql to replace the DEFAULT constraint created in step 4 with a constraint that makes WA (Washington) the default for the **Region** column in the **Employees** table. L05\Answers\DefCons2.sql is a completed script for this step.

```
USE ClassNorthwind
GO

ALTER TABLE Employees DROP CONSTRAINT DF_Region
GO

ALTER TABLE Employees
  ADD CONSTRAINT DF_Region DEFAULT 'WA' FOR Region
GO
```

Exercise 2
Defining CHECK Constraints

In this exercise, you will add two CHECK constraints to the tables in the **ClassNorthwind** database.

► To define a CHECK constraint

In this procedure, you will execute a script to add a *title of courtesy* constraint to the **Employees** table, and then you will write and execute a statement to add a birth date constraint to the **Employee** table. Finally, you will write and execute a statement to test the new constraints.

1. Open Labfiles\L05\ChkConst.sql, and then review and execute it.

   This script will add a title of courtesy CHECK constraint to the **Employees** table of the **ClassNorthwind** database.

2. Write and execute a statement that adds a constraint to the **BirthDate** column in the **Employees** table called **CK_BirthDate**. The value in the **BirthDate** column must be earlier than today's date. L05\Answers\BirthDate.sql is a completed script for this step.

```
USE ClassNorthwind
ALTER TABLE Employees
  ADD CONSTRAINT CK_BirthDate
  CHECK (BirthDate < GETDATE())
GO
```

3. Execute statements that violate each constraint.

   Use the following examples as a template.

```
USE ClassNorthwind
GO
UPDATE Employees SET TitleOfCourtesy = 'None'
WHERE EmployeeID = 1
GO
UPDATE Employees SET BirthDate = (GETDATE()+1)
WHERE EmployeeID = 1
GO
```

   What happens?

   In each case, the command is stopped, with the message that the UPDATE statement conflicted with the constraint.

   _____

   _____

4. Execute **sp_helpconstraint** ***tablename*** and **sp_help** ***constraintname*** to view information on the CHECK constraints that you created.

```
EXEC SP_HELPCONSTRAINT  Employees
GO

EXEC SP_HELP CK_TitleOfCourtesy
GO

EXEC SP_HELP CK_BirthDate
GO
```

Exercise 3
Defining PRIMARY KEY Constraints

In this exercise, you will add PRIMARY KEY constraints to all of the tables in the **ClassNorthwind** database.

► To define a PRIMARY KEY constraint

In this procedure, you first will execute a script that creates a primary key on the **Employees** table, and then you will write a statement to create a PRIMARY KEY constraint on the **Customers** table. Finally, you will execute a script that adds PRIMARY KEY constraints to the other tables in the **ClassNorthwind** database.

1. Open Labfiles\L05\Prikey1.sql, and then review and execute it to create a PRIMARY KEY constraint on the **Employees** table in the **ClassNorthwind** database.
2. Write and execute a statement that adds a PRIMARY KEY constraint called **PK_Customers** on the **CustomerID** column in the **Customers** table. L05\Answers\PriTitle.sql is a completed script for this step.

   ```
   USE ClassNorthwind
   ALTER TABLE Customers
     ADD CONSTRAINT PK_Customers PRIMARY KEY NONCLUSTERED
       (CustomerID)
   GO
   ```

   What is the impact on nonclustered indexes associated with the table when you create a PRIMARY KEY constraint?

   A PRIMARY KEY constraint automatically creates a clustered index, and, in turn, the creation of a clustered index automatically rebuilds all nonclustered indexes.

   _____

   _____

3. Open Labfiles\L05\PriKey2.sql, and then review and execute it to create PRIMARY KEY constraints on the remaining tables in the **ClassNorthwind** database.
4. Execute the **sp_helpconstraint** system stored procedure to view information on the PRIMARY KEY constraint that you created for the **orders** table. Also execute the **sp_help** system stored procedure on the constraint on **PK_Employees** in the **ClassNorthwind** database.

   ```
   EXEC SP_HELPCONSTRAINT orders
   GO

   EXEC SP_HELP PK_Employees
   GO
   ```

Exercise 4
Defining FOREIGN KEY Constraints

In this exercise, you will add FOREIGN KEY constraints to tables in the **ClassNorthwind** database.

► To define a FOREIGN KEY constraint

In this procedure, you will first execute a script that creates a foreign key on the **Orders** table, and then you will write a statement to create a FOREIGN KEY constraint on the **Orders** table. Finally, you will execute a script that adds FOREIGN KEY constraints to the other tables in the **ClassNorthwind** database.

1. Open Labfiles\L05\ForKey1.sql, and then review and execute it to create a FOREIGN KEY constraint on the **Orders** table.

   Why was it not necessary to execute any DROP INDEX statements?

   Creating a FOREIGN KEY does not automatically create an index.

   _____

   _____

   Does the FOREIGN KEY constraint prevent the referenced table from being dropped or truncated?

   Yes. You cannot drop a table referenced by a FOREIGN KEY constraint. Truncating a table referenced by a FOREIGN KEY constraint would also violate the constraint, because the primary keys that it references would no longer exist.

   _____

   _____

2. Write and execute a statement that adds a FOREIGN KEY constraint, called **FK_Products_Categories**, to the **CategoryID** column in the **Products** table referencing the **CategoryID** column in the **Categories** table. Specify an option that does not verify that the existing data conforms to the new constraint. L05\Answers\ForeignKeyProd.sql is a completed script for this step.

```
USE ClassNorthwind
ALTER TABLE dbo.Products WITH NOCHECK
   ADD CONSTRAINT FK_Products_Categories
   FOREIGN KEY(CategoryID) REFERENCES
      dbo.Categories(CategoryID)
GO
```

3. Open Labfiles\L05\ForKey2.sql, and then review and execute it to create the remaining FOREIGN KEY constraints in the **ClassNorthwind** database.
4. Execute **sp_helpconstraint** ***tablename*** to view information on some of the FOREIGN KEY constraints that you created. You can use the following tables with FOREIGN KEY constraints for this step: **Products**, **Orders**, **Order Details**, **Suppliers**, and **Employees**.

```
EXEC SP_HELPCONSTRAINT  Products
GO
EXEC SP_HELPCONSTRAINT  Employees
GO
..
```

If Time Permits
Creating Defaults and Rules

In this exercise, you will add defaults and rules to the **ClassNorthwind** database.

► To create a default

In this procedure, you will execute a script to create and bind a default, and then you will verify that the default is functioning correctly.

1. Open Labfiles\L05\CreaDefa.sql, and then review and execute it.

   This script creates and binds a default to the **Suppliers.Country** column. The default is Singapore.

2. Execute a statement that inserts a new record to verify that the default is working properly. The example below shows how to do this for the **Suppliers** table. You can change the example to include your favorite book title and author.

   ```
   USE ClassNorthwind
   INSERT Suppliers (CompanyName) VALUES ('Karl''s Bakery')
   GO
   ```

3. Write and execute a statement to query the **Suppliers** table to view the results. The example below assumes that you used the data supplied in the previous step.

   ```
   USE ClassNorthwind
   SELECT * FROM Suppliers
      WHERE Country = 'Singapore'
   GO
   ```

► To create a rule

In this procedure, you will execute a script to create and bind a rule, and then you will verify that the rule is functioning correctly.

1. Open Labfiles\L05\CreaRule.sql, and then review and execute it.

   This script creates a path rule that ensures that employee-photo paths follow the format described in the script.

2. Execute the following UPDATE statement to test the rule by attempting to update the **PhotoPath** column with an invalid path. The statement should fail because it violates the path rule.

   ```
   USE ClassNorthwind
   UPDATE Employees
      SET PhotoPath = 'http://accweb/xemmployees/new.bmp'
      WHERE LastName = 'Fuller'
   GO
   ```

Review

Topic Objective
To reinforce module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- Types of Data Integrity
- Enforcing Data Integrity
- Defining Constraints
- Types of Constraints
- Disabling Constraints
- Using Defaults and Rules
- Deciding Which Enforcement Method to Use

**********ILLEGAL FOR NON-TRAINER USE**********

1. What type of constraint would you add to the Country field in your database to ensure that your Indonesian subsidiary does business only with other Indonesian companies?

   A CHECK constraint (or a rule).

2. After implementation of the constraint, or rule, in question 1, your data entry operators are complaining that they have to enter the word Indonesia over and over again. How can you fix this?

   Create a DEFAULT constraint (or a default).

3. Your business has changed and you no longer do work in Indonesia. Your subsidiary has moved to Malaysia and is now doing business with several other East Asian countries. Currently, there are 4.5 million sales records that include Indonesia in the country field. How can you add the new countries and still preserve the rows that contain Indonesia?

   Alter the table to drop the existing constraint and then alter the table to add the new constraint. When you add the constraint, use the WITH NOCHECK option.

4. Your order entry system has two main tables: **Orders** and **Customers**. What data integrity components should you consider if you want to ensure that each order and customer can be uniquely identified? How would you manage the relationship between the two tables?

   Make sure that you define a PRIMARY KEY constraint on the Customers table. Use a FOREIGN KEY constraint in the Orders table to reference the Customers table.