

Поддержка целостности данных

Содержание темы:

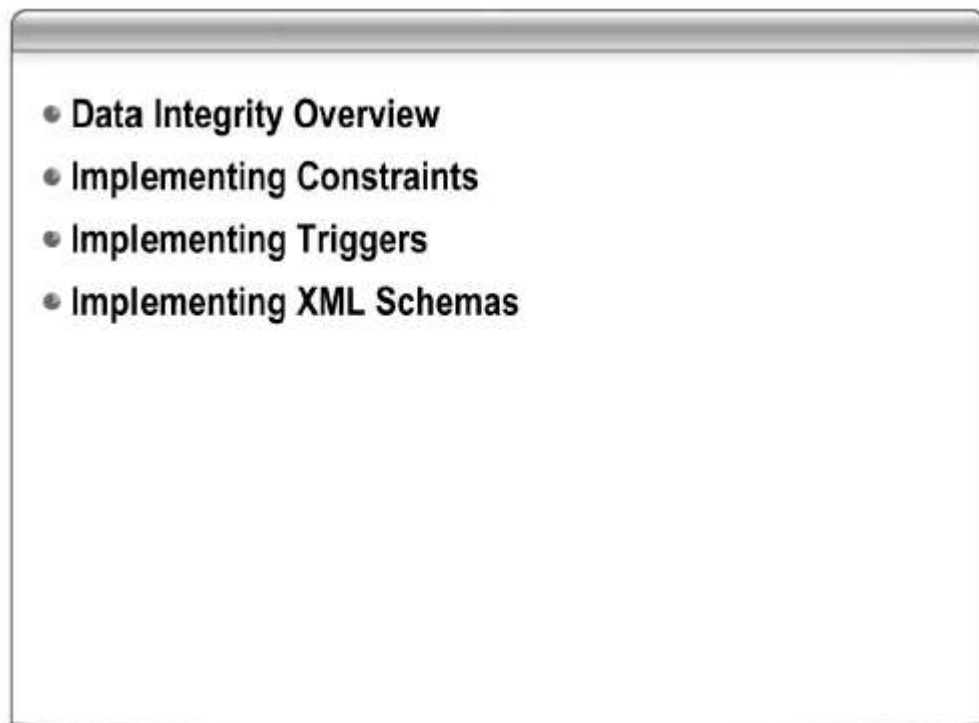
[Урок 1: Краткий обзор целостности данных](#)

[Урок 2: Реализация ограничений](#)

[Урок 3: Реализация триггеров](#)

[Урок 4: Реализация схем XML](#)

Поддержка целостности данных



Цели

После завершения этого модуля, студенты смогут:

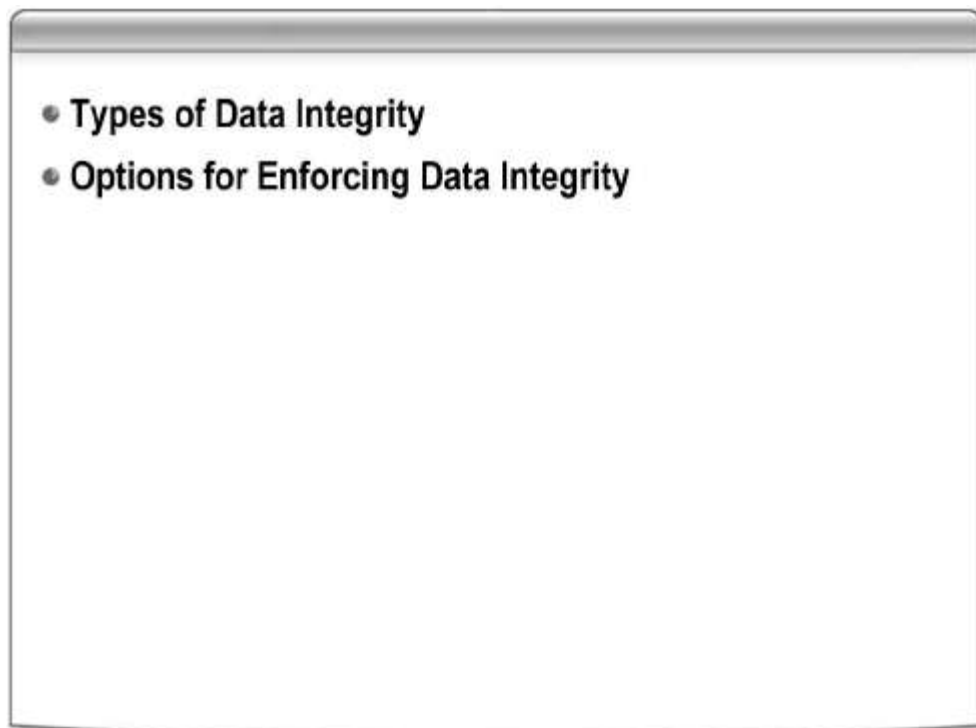
- Описывать типы целостности данных и варианты для их реализации.
- Реализовать ограничения
- Реализовать триггеры.
- Применять схемы XML.

Введение

Качество данных в базе данных в значительной степени определяет полноценность и эффективность работы приложений (и людей), которые полагаются на это качество. Кроме того, качество данных может играть немаловажную роль в успехе некоторой организации. Обеспечение целостности данных необходимо для поддержания высококачественных данных.

Этот модуль начинается с краткого обзора типов целостности данных, которая необходима Вам как разработчику базы данных и обзора возможностей, предоставленных SQL Server 2005 для решения этих проблем. Затем подробно обсуждаются три самые значительные и мощные возможности по обеспечению целостности данных SQL Server 2005: ограничения, триггеры и схемы XML.

Урок 1: Краткий обзор целостности данных



Цели урока

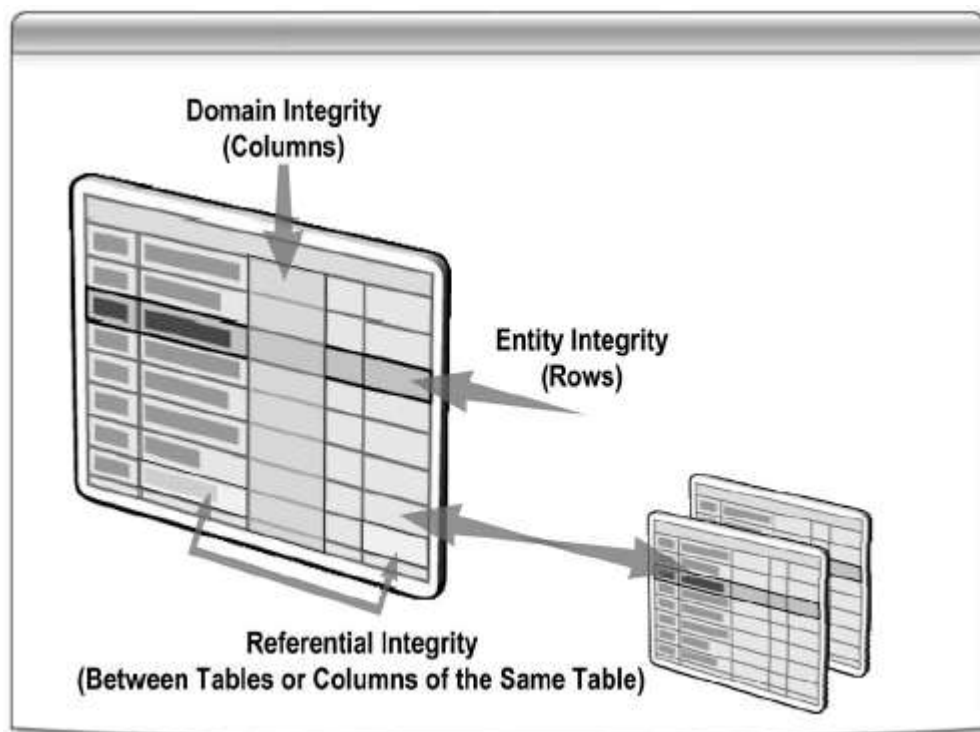
По окончании этого урока, студенты смогут:

- Описывать типы целостности данных.
- Описывать варианты реализации целостности данных.

Краткий обзор целостности данных

Важным шагом в планировании базы данных является выбор наилучшего способа реализации целостности данных. Целостность данных означает согласованность и точность данных, которые хранятся в базе данных. В этом уроке Вы узнаете о различных типах целостности данных в реляционных базах данных и вариантах ее реализации, предоставленных SQL Server 2005.

Типы целостности данных



Типы целостности данных

Обеспечение целостности данных гарантирует качество данных в базе данных. Существуют различные типы целостности данных, которую Вы должны запланировать:

- целостность домена (или столбца)
- целостность сущности
- ссылочная целостность

Целостность домена

Целостность домена (или столбца) определяет набор допустимых значений для столбца и решает, позволить ли null значения. Целостность домена часто обеспечивается с помощью проверки правильности введенных данных и может быть реализовано с помощью ограничений типа данных, формата, а также диапазона возможных значений.

Целостность сущности

Целостность сущности (или таблицы) требует, чтобы у всех строк в таблице был уникальный идентификатор, известный как PRIMARY KEY. Может ли значение первичного ключа быть изменено и может ли быть удалена целая строка, зависит от уровня ссылочной целостности.

Ссылочная целостность

Ссылочная целостность гарантирует, что всегда поддерживаются связи между первичными ключами (в таблицах, на которые ссылаются), и внешними ключами (в таблицах, которые ссылаются). Если внешний ключ ссылается на некоторую строку, и при этом не разрешено каскадное действие, то строка в таблице, на которую ссылаются, не может быть удалена, а PRIMARY KEY не может быть изменен. Вы можете определить связи внутри одной и той же таблицы или между отдельными таблицами.

Дополнительная информация. Дополнительная информация о различных типах целостности данных, содержится в разделе “Целостность данных” в SQL Server Books Online.

Варианты обеспечения целостности данных

Mechanism	Description
Data types	Define the type of data that can be stored in a column
Rules	Define the acceptable values that can be inserted into a column
Defaults	Define the value of a column if a value is not specified
Constraints	Define how the Database Engine enforces data integrity
Triggers	Define code that is executed automatically when table is modified
XML schemas	Define the acceptable content of XML documents and fragments placed in an xml data column

Варианты обеспечения целостности данных

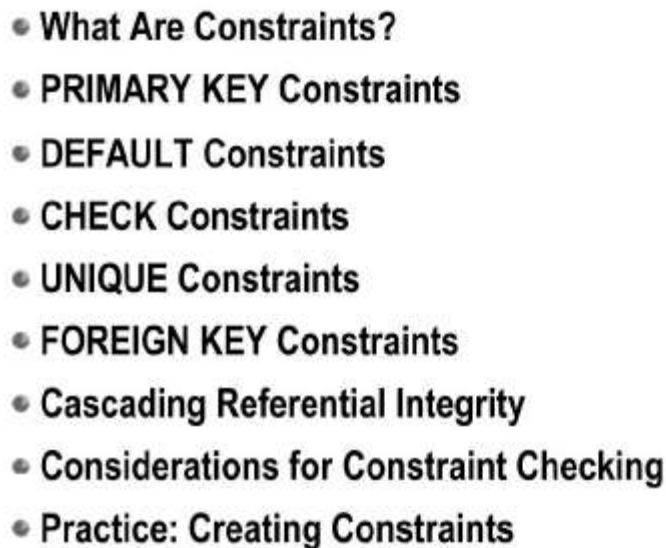
В следующей таблице содержится краткий обзор механизмов, предоставленные SQL Server 2005 для обеспечения целостности данных.

Механизм	Тип целостности	Описание
Data types	Домен	Типы данных поддерживают фундаментальные ограничения на данные, которые могут храниться в каждом столбце. Назначение типов данных для каждого столбца в таблице первый шаг в обеспечении целостности данных. Однако, одних только типов данных не достаточно, чтобы обеспечить целостность данных. Например, столбец, объявленный как тип int , гарантирует, что могут быть введены только числовые целые значения, но не может ограничить значения диапазоном от 0 до 1000.

Правила и умолчания	Домен, Сущность	Правила определяют приемлемые значения, которые может содержать столбец, а умолчание определяет значение столбца, если это значение не указано. Важно, что правила и умолчания - независимые объекты, которые могут быть связаны с одним или более столбцов или пользовательским типом данных, при этом можно определить однажды правила и умолчания, а затем, неоднократно их использовать. Неудобство использования правил и умолчаний состоит в том, что они не соответствуют American National Standards Institute (ANSI). Вы должны использовать ограничения вместо правил и умолчаний.
Ограничения	Домен, Сущность, Ссылка	Ограничения позволяют Вам определять для SQL Server 2005 Database Engine автоматический способ обеспечения целостности БД. Ограничения определяют ограничения на значения, позволенные в столбцах, и они являются стандартным механизмом обеспечения целостности. Использование ограничений предпочтительнее использования триггеров, правил и умолчаний. Оптимизатор запросов также использует определения ограничений, чтобы строить высокопроизводительные планы выполнения запросов.
Триггер	Домен, Сущность, Ссылки	Триггеры - специальный вид хранимой процедуры, который выполняется, когда на сервере базы данных происходит событие языка манипулирования данными (DML). События DML включают операторы ОБНОВЛЕНИЯ, ВСТАВКИ или УДАЛЕНИЯ, выполненные на таблице или представлении. Триггеры используются для обеспечения бизнес правил при изменении данных.
XML схемы	Домен (XML)	XML схемы определяют пространство имен, структуру и приемлемое содержание документов XML. С помощью применения XML схемы к столбцу таблицы, определенной с типом данных xml , Вы можете ограничить структуру и содержание XML данных, сохраненных в этом столбце.

Дополнительная информация. Дополнительная информация о различных способах обеспечения целостности данных в SQL Server 2005, содержится в разделе “Целостность Данных” в SQL Server Books Online.

Урок 2: Реализация ограничений

- 
- What Are Constraints?
 - PRIMARY KEY Constraints
 - DEFAULT Constraints
 - CHECK Constraints
 - UNIQUE Constraints
 - FOREIGN KEY Constraints
 - Cascading Referential Integrity
 - Considerations for Constraint Checking
 - Practice: Creating Constraints

Цели урока

По окончании этого урока, студенты смогут:

- Описывать ограничения.
- Определять ограничения первичного ключа PRIMARY KEY.
- Определять ограничения умолчаний DEFAULT.
- Определять проверочные ограничения CHECK.
- Определять ограничения уникальности UNIQUE.
- Определять ограничения внешнего ключа FOREIGN KEY.
- Определять каскадную ссылочную целостность.
- Определять причины отключения ограничений.

Реализация ограничений

Ограничения - рекомендуемый метод обеспечения целостности данных. В этом уроке, Вы изучите какие типы ограничений допустимы, какие типы ограничений использовать в зависимости от Ваших потребностей, какой тип целостности данных какое ограничение реализует и как определять ограничения.

Что такое ограничения?

Integrity type	Constraint type	Description
Domain	DEFAULT	Specifies default value for column
	CHECK	Specifies allowed value for column
	FOREIGN KEY	Specifies column in which values must exist
	NULL	Specifies whether NULL is permitted
Entity	PRIMARY KEY	Identifies each row uniquely
	UNIQUE	Prevents duplication of nonprimary keys
Referential	FOREIGN KEY	Defines columns whose value must match the primary key of this table
	CHECK	Specifies the allowed value for a column based on the contents of another column

Определение

Ограничения - ANSI-стандартный метод обеспечения целостности данных. Каждый тип целостности данных - домен, сущность и ссылка реализованы при использовании различных типов ограничений. Ограничения гарантируют, что в столбцы вводятся допустимые значения данных и что между таблицами поддерживаются связи. В следующей таблице описываются различные типы ограничений, доступные в SQL Server 2005 и определены ли они на таблице или столбце.

Тип целостности	Тип ограничения	Применение	Описание
Домен	DEFAULT (умолчание)	Столбец	Определяет значение DEFAULT для столбца, когда в операторе INSERT не указано значение. Ограничение DEFAULT – рекомендуемая альтернатива объекту default.
	CHECK (проверка)	Столбец	Определяет значение данных, которые являются допустимыми для столбца.

			Ограничение CHECK – рекомендуемая альтернатива объекту rule.
	FOREIGN KEY (внешний ключ)	Таблица	Определяет значения данных, которые являются обновляемыми и базируются на значениях столбца другой таблицы.
	NULL	Столбец	Определяет, может ли столбец иметь null-значения.
Сущность	PRIMARY KEY (первичный ключ)	Таблица	Идентифицирует каждый строку уникально – гарантирует, что пользователи не введут повторяющиеся значения и что будет создан индекс для увеличения производительности. Null-значения не допустимы.
	UNIQUE (уникальность)	Столбец	Предотвращает дублирование альтернативных (непервичных) ключей и гарантирует, что будет создан индекс, чтобы увеличить производительность. Null-значения допустимы.
Ссылка	FOREIGN KEY (внешний ключ)	Таблица	Определяет столбец или комбинацию столбцов значений, которые соответствуют первичному ключу той же самой или другой таблицы.
	CHECK (проверка)	Столбец	Определяет допустимые значения данных в столбце, основанных на значениях в других столбцах той же таблицы.

Создание ограничений

При создании таблицы с помощью оператора CREATE TABLE Вы можете создать ограничения. Ограничение уровня столбца относится к единственному столбцу и является частью его определения. Ограничение уровня таблицы может ссылаться на один или более столбцов таблицы. Все ограничения уровня таблицы специфицируются в отдельной секции после определения всех столбцов. Вы можете изменить ограничения существующей таблицы, используя оператор ALTER TABLE. Вы можете добавить ограничения к таблице с существующими данными, и Вы можете наложить ограничения на один или несколько столбцов.

Далее представлен упрощенный синтаксис оператора CREATE TABLE, который показывает, где Вы объявляете различные ограничения уровня столбца и уровня таблицы.

CREATE TABLE table_name

({ < column_definition > | < table_constraint > } [,...n])

< column_definition > ::=

{ column_name data_type }

[{ DEFAULT constant_expression | [IDENTITY [(seed , increment)]] }

[< column_constraint > [...n]]

< column_constraint > ::=

[CONSTRAINT constraint_name]

{ [NULL | NOT NULL]

| [PRIMARY KEY | UNIQUE]

| REFERENCES ref_table [(ref_column)]

[ON DELETE { CASCADE | NO ACTION }]

[ON UPDATE { CASCADE | NO ACTION }]

}

< table_constraint > ::=

[CONSTRAINT constraint_name]

{ [{ PRIMARY KEY | UNIQUE } { (column [,...n]) }]

| FOREIGN KEY (column [,...n])

REFERENCES ref_table [(ref_column [,...n])]

[ON DELETE { CASCADE | NO ACTION }]

[ON UPDATE { CASCADE | NO ACTION }]

}

Ограничение первичного ключа PRIMARY KEY

- **PRIMARY KEY** constraint identifies one or more columns in a table that constitute a primary key
- One **PRIMARY KEY** constraint is allowed per table
- Value must be unique across constituent columns
- Null values are not allowed in constituent columns

```
CREATE TABLE [HumanResources].[Department](  
    [DepartmentID] [smallint] IDENTITY(1,1) NOT NULL,  
    [Name] [dbo].[Name],  
    ...  
    CONSTRAINT [PK_Department_DepartmentID] PRIMARY KEY CLUSTERED  
        ([DepartmentID] ASC) WITH (IGNORE_DUP_KEY = OFF) ON  
        [PRIMARY]  
)
```

Определение

Ограничение первичного ключа PRIMARY KEY определяет один или несколько столбцов в таблице, которые составляют PRIMARY KEY. PRIMARY KEY уникально идентифицирует строку в таблице и реализует в таблице целостность сущности.

Перед использованием ограничения PRIMARY KEY учтите следующее:

- Таблица может иметь только одно ограничение PRIMARY KEY.
- Столбцы, включенные в ограничение PRIMARY KEY, не могут принимать null-значений.
- Значения в столбцах, определенных как PRIMARY KEY должны быть уникальными. Если ограничение PRIMARY KEY содержит больше чем один столбец, дубликаты могут появиться в одном столбце, но комбинации значений всех столбцов должны быть уникальными.
- Ограничение PRIMARY KEY создает уникальный индекс с указанными столбцами в качестве ключа индекса. Поэтому, столбцы, выбранные для PRIMARY KEY, должны следовать правилам для создания уникальных индексов. Вы можете определить кластерный или некластерный индекс (если индекс еще не существует, то DEFAULT создается кластерный индекс). Вы не можете удалить индекс, который поддерживает ограничение PRIMARY KEY. Сначала необходимо удалить ограничение PRIMARY

KEY, затем удалится индекс. Этот индекс также позволяет осуществить быстрый доступ к данным, когда в запросах используется первичный ключ.

Дополнительная информация. За дополнительной информацией об ограничении PRIMARY KEY обращайтесь к разделу “PRIMARY KEY Constraints” в SQL Server Books Online.

Когда использовать ограничение PRIMARY KEY

Использование ограничения PRIMARY KEY целесообразно в следующих случаях:

- Один или несколько столбцов в таблице должны уникально идентифицировать каждую строку (сущность) в таблице.
- Один из столбцов в таблице является столбцом идентичности.

Создание ограничения PRIMARY KEY

Вы создаете ограничения PRIMARY KEY с помощью раздела CONSTRAINT в операторах CREATE TABLE и ALTER TABLE.

В следующем коде демонстрируется оператор Transact-SQL, используемый для создания таблицы **HumanResources.Department** базы данных **AdventureWorks**. Раздел CONSTRAINT определяет кластерное ограничение первичного ключа **PK_Department_DepartmentID** на столбец **DepartmentID**.

```
CREATE TABLE [HumanResources].[Department](
    [DepartmentID] [smallint] IDENTITY(1,1) NOT NULL,
    [Name] [dbo].[Name] NOT NULL,
    [GroupName] [dbo].[Name] NOT NULL,
    [ModifiedDate] [datetime] NOT NULL,
    CONSTRAINT [PK_Department_DepartmentID] PRIMARY KEY CLUSTERED
    ([DepartmentID] ASC )WITH (IGNORE_DUP_KEY = OFF)
ON [PRIMARY] )
```

Дополнительная информация. За дополнительной информацией о создании и изменении ограничения первичного ключа обратитесь к разделу “Creating and Modifying PRIMARY KEY Constraints” в SQL Server Books Online.

Ограничение умолчания DEFAULT

- **DEFAULT constraint defines a default column value when no value is provided**
- **Only one DEFAULT constraint per column**
- **Only applicable to INSERT statements**
- **Some system supplied functions allowed**

```
CREATE TABLE [Production].[Location](  
    ...  
    [Availability] [decimal](8, 2) NOT NULL CONSTRAINT  
        [DF_Location_Availability] DEFAULT ((0.00)),  
    [ModifiedDate] [datetime] NOT NULL CONSTRAINT  
        [DF_Location_ModifiedDate] DEFAULT (getdate())  
)
```

Определение

Ограничение умолчания DEFAULT вводит значение в столбец, когда он не определен в операторе INSERT. Ограничение DEFAULT реализует целостность домена.

Перед использованием ограничения DEFAULT учтите следующие факты:

- Ограничение DEFAULT срабатывает только на оператор INSERT.
- Каждый столбец может иметь только одно ограничение DEFAULT.
- У столбца со свойством IDENTITY или типом данных **rowversion** не может быть ограничения DEFAULT.
- Ограничение DEFAULT позволяет вместо пользовательских значений использовать некоторые системные переменные и функции: USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, или CURRENT_TIMESTAMP.

Дополнительная информация. За дополнительной информацией об ограничениях DEFAULT, обратитесь к разделу “DEFAULT Definitions” SQL Server Books Online.

Когда использовать ограничение DEFAULT

Используйте ограничение DEFAULT когда:

- данные, хранимые в столбце, имеют очевидное значение по умолчанию.
- столбец не принимает null значения.
- столбец не является уникальным.

Создание ограничения DEFAULT

Вы создаете ограничение DEFAULT при использовании раздела уровня столбца CONSTRAINT в операторах CREATE TABLE и ALTER TABLE. Следующий код показывает оператор Transact-SQL, создающий таблицу **Production.Location** в базе данных **AdventureWorks**. В примере создаются ограничения DEFAULT для столбцов **CostRate.Availability**, и **ModifiedDate**.

```
CREATE TABLE [Production].[Location](  
    [LocationID] [smallint] IDENTITY(1,1) NOT NULL,  
    [Name] [dbo].[Name] NOT NULL,  
    [CostRate] [smallmoney] NOT NULL CONSTRAINT  
    [DF_Location_CostRate] DEFAULT ((0.00)),  
    [Availability] [decimal](8, 2) NOT NULL CONSTRAINT  
    [DF_Location_Availability] DEFAULT ((0.00)),  
    [ModifiedDate] [datetime] NOT NULL CONSTRAINT [DF_Location_ModifiedDate]  
    DEFAULT (getdate())  
)
```

Дополнительная информация. За дополнительной информацией о создании и изменении ограничения DEFAULT обратитесь к разделу “Creating and Modifying DEFAULT Definitions” в SQL Server Books Online.

Проверочное ограничение CHECK

- **CHECK constraints restrict the values that can be entered into a column on INSERT or UPDATE**
- **Can define multiple CHECK constraints per column**
- **Can reference columns in the same table**
- **Cannot contain subqueries**

```
ALTER TABLE [HumanResources].[EmployeeDepartmentHistory]
WITH CHECK
ADD CONSTRAINT [CK_EmployeeDepartmentHistory_EndDate] CHECK
(((EndDate)>=[StartDate] OR [EndDate] IS NULL))
```

Определение

Ограничение CHECK ограничивает значения данных, которые могут ввести пользователи в определенный столбец с помощью операторов INSERT и UPDATE. Вы можете применять проверочные ограничения на уровне столбца или таблицы. Ограничения CHECK уровня столбца ограничивают значения, которые могут быть сохранены в этом же столбце. Ограничения CHECK уровня таблицы могут ссылаться на множество столбцов в этой таблице, для перекрестной ссылки и сравнения значений столбцов.

Перед использованием ограничения CHECK учтите следующие факты:

- Ограничение CHECK проверяет данные каждый раз, когда Вы выполняете операторы INSERT или UPDATE.
- Ограничение CHECK может быть любым логическим (Булевым) выражением, которое возвращает TRUE или FALSE.
- Ограничение CHECK не может содержать подзапросы.
- Один столбец может иметь несколько ограничений CHECK.
- Ограничение CHECK не может накладываться на столбцы следующих типов: **rowversion**, **text**, **ntext**, или **image**.

- Оператор проверки согласованности БД (DBCC) CHECKCONSTRAINTS вернет все строки с данными, которые нарушают ограничение CHECK.

Дополнительная информация. За дополнительной информацией об ограничениях CHECK обратитесь к разделу “CHECK Constraints” в SQL Server Books Online.

Когда использовать ограничения CHECK

Рассмотрите использование ограничения CHECK, когда:

- Бизнес логика диктует, что данные, хранимые в столбце, должны входить в определенное множество или диапазон значений.
- Данные, хранимые в столбце, имеют естественные ограничения на значения, которые они могут содержать.
- Между столбцами таблицы существуют отношения, которые ограничивают значения столбца, которые он может содержать.

Создание ограничения CHECK

Вы можете создать ограничения CHECK при использовании раздела CONSTRAINTS уровня столбца или уровня таблицы операторов CREATE TABLE или ALTER TABLE. Следующий код демонстрирует использование оператора Transact-SQL, который добавляет ограничение CHECK к таблице **HumanResources.EmployeeDepartmentHistory** базы данных **AdventureWorks**. Это ограничение гарантирует, что значение столбца **EndDate** больше или равно значения столбца **StartDate** или является null-значением.

```
ALTER TABLE [HumanResources].[EmployeeDepartmentHistory] WITH CHECK  
    ADD CONSTRAINT [CK_EmployeeDepartmentHistory_EndDate]  
    CHECK (([EndDate]>=[StartDate] OR [EndDate] IS NULL))
```

Дополнительная информация. За дополнительной информацией о создании и изменении ограничения CHECK, обращайтесь к разделу “Creating and Modifying CHECK Constraints” в SQL Server Books Online.

Ограничения уникальности UNIQUE

- **UNIQUE constraints ensure that every value in a column is unique**
- **Only one null value allowed in a unique column**
- **Can include one or more columns**

```
CREATE TABLE [HumanResources].[Employee](  
    [EmployeeID] [int] IDENTITY(1,1) NOT NULL,  
    [NationalIDNumber] [nvarchar](15) NOT NULL UNIQUE  
    NONCLUSTERED,  
    ...  
)
```

Определение

Ограничение уникальности **UNIQUE** гарантирует, что не может быть двух строк с одинаковыми значениями в столбце. Ограничение **UNIQUE** полезно, когда у Вас уже есть первичный ключ (**PRIMARY KEY**), например, номер служащего, но Вы хотите определить другие уникальные столбцы, такие как, например, идентификационный налоговый номер (ИНН) служащего.

Прежде, чем приступить к реализации ограничения **UNIQUE**, следует рассмотреть следующие факты:

- В столбце с ограничением **UNIQUE** может появиться только одно пустое значение.
- Таблица может иметь множество ограничений **UNIQUE**, но только одно ограничение **PRIMARY KEY**.
- Ограничение **UNIQUE** реализуется через создание уникального некластерного индекса на указанный столбец или множество столбцов. Всего таких индексов должно быть не более 249.
- Двигатель БД возвратит ошибку, если Вы создадите ограничение **UNIQUE** на столбец, который содержит дублирующиеся значения.

Дополнительная информация. За дополнительной информацией об ограничениях UNIQUE обращайтесь к разделу “UNIQUE Constraints” в SQL Server Books Online.

Когда использовать ограничения UNIQUE

Рассмотрите использование ограничения UNIQUE когда:

- Таблица содержит столбцы, которые не являются первичным ключом, но индивидуально или в совокупности, должны содержать уникальные значения.
- Бизнес логика диктует, что данные, хранимые в столбце, должны быть уникальными.
- Данные, хранимые в столбце, имеют естественные ограничения уникальности на значения, которые он может содержать, например, налоговый номер (ИНН) или паспортные данные.

Создание ограничения UNIQUE

Вы можете создать ограничения UNIQUE при использовании раздела UNIQUE уровня столбца или уровня таблицы в операторах CREATE TABLE или ALTER TABLE. Следующий код демонстрирует использование оператора Transact-SQL для создания таблицы **HumanResources.Employees** базы данных **AdventureWorks**. В примере определяется ограничение UNIQUE на столбец **NationalIDNumber**, и гарантируется, что столбец не содержит null значений и то, что индекс, созданный, чтобы поддержать ограничение UNIQUE, некластерный.

```
CREATE TABLE [HumanResources].[Employee](  
    [EmployeeID] [int] IDENTITY(1,1) NOT NULL,  
    [NationalIDNumber] [nvarchar](15) NOT NULL UNIQUE NONCLUSTERED,  
    [ContactID] [int] NOT NULL,  
    ...  
)
```

Дополнительная информация. За дополнительной информацией о создании и изменении ограничений UNIQUE обратитесь к разделу “Creating and Modifying UNIQUE Constraints ” в SQL Server Books Online.

Ограничения внешнего ключа FOREIGN KEY

- **FOREIGN KEY constraints ensure referential integrity between columns in same or different tables**
- **Must reference a PRIMARY KEY or UNIQUE constraint**
- **User must have REFERENCES permission on referenced table**

```
ALTER TABLE [Sales].[SalesOrderHeader] WITH CHECK  
ADD CONSTRAINT [FK_SalesOrderHeader_Customer_CustomerID]  
FOREIGN KEY([CustomerID])  
REFERENCES [Sales].[Customer] ([CustomerID])
```

Определение

Внешний ключ - столбец или совокупность столбцов, которая используется, чтобы установить и реализовать связь между данными в двух таблицах. Ограничение внешнего ключа FOREIGN KEY реализует ссылочную целостность. Ограничение FOREIGN KEY определяет ссылку на столбец - первичный ключ (PRIMARY KEY) или столбец с ограничением уникальности UNIQUE в той же самой или другой таблице. Значения в столбце внешнего ключа должны быть в столбце первичного ключа PRIMARY KEY. Так как к значениям первичного ключа существуют ссылки, то эти значения не могут быть изменены или удалены.

Прежде, чем приступить к реализации ограничения FOREIGN KEY, следует рассмотреть следующие факты:

- Ограничение FOREIGN KEY обеспечивает ссылочную целостность по одному или нескольким столбцам. Число столбцов и их типы данных, которые определены в разделе FOREIGN KEY, должны соответствовать числу столбцов и их типам данных в разделе REFERENCES.
- В отличие от ограничений PRIMARY KEY и UNIQUE ограничения FOREIGN KEY не создают индексы автоматически.

- Чтобы другой пользователь мог создать ограничение FOREIGN KEY, ссылаясь на таблицу, которая принадлежит Вам, Вы должны предоставить этому пользователю право REFERENCES на Вашу таблицу. Это гарантирует, что ни один пользователь не может ограничить операции на Вашей таблице, создавая FOREIGN KEY со ссылкой на нее.
- Ограничение FOREIGN KEY, которое использует только раздел REFERENCES без раздела FOREIGN KEY, относится к одному столбцу таблицы.

Дополнительная информация. За дополнительной информацией об ограничениях FOREIGN KEY обратитесь к разделу “FOREIGN KEY Constraints” в SQL Server Books Online.

Когда использовать ограничения FOREIGN KEY

Рассмотрите использование ограничения FOREIGN KEY когда:

- Данные в одном или нескольких столбцах может содержать только те значения, которые содержатся в определенном столбце (или столбцах) той же самой или другой таблице.
- Строки в таблице не должны удаляться, пока существуют строки в другой таблице, которые зависят от них.

Создание ограничения FOREIGN KEY

Вы можете создать ограничения FOREIGN KEY при использовании раздела CONSTRAINT уровня столбца или уровня таблицы в операторах CREATE TABLE или ALTER TABLE. Следующий код демонстрирует, как используется оператор Transact-SQL для добавления ограничения FOREIGN KEY к таблице **Sales.SalesOrderHeader** базы данных **AdventureWorks**. В примере создается ограничение внешнего ключа **FK_SalesOrderHeader_Customer_CustomerID**, которое устанавливает ссылочную связь между столбцом **CustomerID** таблицы **Sales.SalesOrderHeader** и столбцом **CustomerID** таблицы **Sales.Customer**.

```
ALTER TABLE [Sales].[SalesOrderHeader] WITH CHECK ADD CONSTRAINT  
    [FK_SalesOrderHeader_Customer_CustomerID] FOREIGN KEY([CustomerID])  
REFERENCES [Sales].[Customer] ([CustomerID])
```

Дополнительная информация. За дополнительной информацией о создании и изменении ограничения FOREIGN KEY обратитесь к разделу “Creating and Modifying FOREIGN KEY Constraints” в SQL Server Books Online.

Каскадирование ссылочной целостности

Controlled by CASCADE clause of the FOREIGN KEY constraint		
Cascade option	UPDATE behavior	DELETE behavior
NO ACTION (Default)	Raise error; roll back operation	
CASCADE	Update foreign keys in referencing tables	Delete rows in referencing tables
SET NULL	Set foreign keys in referencing tables to NULL	
SET DEFAULT	Set foreign keys in referencing tables to DEFAULT values	

Введение

Ограничение FOREIGN KEY включает опцию CASCADE, которая позволяет любое изменение значения столбца, для которого определено ограничение UNIQUE или PRIMARY KEY, чтобы распространить это изменение на любые зависимые значения внешних ключей. Это действие известно как каскадирование ссылочной целостности.

Разделы REFERENCES операторов CREATE TABLE и ALTER TABLE поддерживают разделы ON DELETE и ON UPDATE. Эти разделы позволяют Вам определять поведение каскадирования ссылочной целостности. Возможные значения включают NO ACTION, CASCADE, SET NULL, и SET DEFAULT. NO ACTION является значением по умолчанию.

Дополнительная информация. За дополнительной информацией о каскадировании ссылочной целостности, обратитесь к разделу “Cascading Referential Integrity Constraints” в SQL Server Books Online.

Каскадное обновление

Раздел ON UPDATE ограничения FOREIGN KEY определяет поведение при попытке оператором UPDATE изменить значение первичного ключа, на которое ссылаются значения

внешних ключей других таблиц. В следующей таблице описываются возможные значения раздела ON UPDATE и их действия.

Опция	Действие
NO ACTION	Определяет, что если сделана попытка обновить ключевое значение в строке, на которую ссылаются внешние ключи в существующих строках другой таблицы, то возникает ошибка, и отменяется оператор UPDATE.
CASCADE	Определяет, что если сделана попытка обновить ключевое значение в строке, на которую ссылаются внешние ключи в существующих строках другой таблицы, то все значения столбцов, которые составляют внешний ключ, также обновляются новым значением, определенным для ключа.
SET NULL	Определяет, что если была попытка обновить строку в столбце первичного ключа, на которую ссылаются значения внешних ключей других таблиц, то все значения, которые составляют внешний ключ, устанавливаются в null. Все столбцы внешнего ключа целевой таблицы должны допускать null значения для этого ограничения.
SET DEFAULT	Определяет, что если сделана попытка обновить строку в ключевых столбцах, на которые есть ссылки из других таблиц, то все значения соответствующих внешних ключей устанавливаются в значение по умолчанию. Все столбцы внешнего ключа целевой таблицы должны иметь определение DEFAULT для этого ограничения. Если столбец допускает null значения и нет никакого явного набора значений по умолчанию, то null значение неявно становится значением по умолчанию для этого столбца. Любые определенные (не null) значения, которые установлены из-за ON UPDATE SET DEFAULT, должны иметь соответствующие значения в первичной таблице, чтобы поддержать ограничения внешнего ключа.

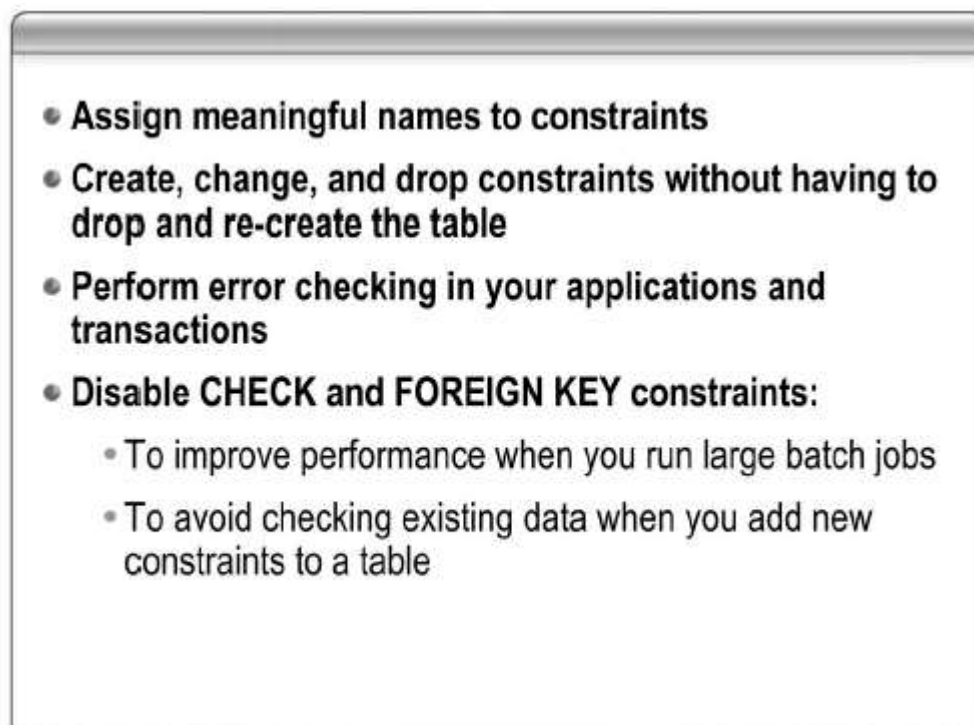
Каскадное удаление

Раздел ON DELETE ограничения FOREIGN KEY определяет действия после попытки удалить строки, на которые ссылаются значения внешних ключей других таблиц. В следующей таблице описываются возможные опции раздела ON DELETE и их действие.

Опция	Действие
NO ACTION	Определяет, что если будет сделана попытка удалить строку, на которую есть ссылки из других таблиц, то возникнет ошибка, и операция удаления будет отменена.

CASCADE	<p>Определяет, что если будет сделана попытка удалить строку, на которую есть ссылки из других таблиц, то все зависимые строки этих таблиц будут также удалены.</p>
SET NULL	<p>Определяет, что если будет сделана попытка удалить строку, на которую есть ссылки из других таблиц, то все значения внешних ключей в зависимых строках этих таблиц будут установлены в null. Чтобы выполнить это ограничение, все столбцы внешнего ключа целевой таблицы должны допускать null значения.</p>
SET DEFAULT	<p>Определяет, что если будет сделана попытка удалить строку, на которую есть ссылки из других таблиц, то все значения внешних ключей в зависимых строках этих таблиц будут установлены в их значение по умолчанию DEFAULT. Все столбцы внешнего ключа целевой таблицы должны иметь определение DEFAULT для этого ограничения. Если столбец допускает null значения, и нет никакого явного набора значений по умолчанию, то null значение неявно становится значением по умолчанию для этого столбца. Любые определенные (не null) значения, которые установлены из-за ON DELETE SET DEFAULT, должны иметь соответствующие значения в первичной таблице, чтобы поддержать ограничения внешнего ключа.</p>

Проверка ограничения



Проверка ограничения

Когда Вы создаете ограничения, Вы должны определить для них значащие имена; если Вы не определяете имена для Ваших ограничений, SQL генерирует сложные имена. Имена должны быть уникальными для данного владельца объекта БД, а также следовать правилам для идентификаторов SQL Server.

Когда Вы создаете или изменяете ограничения, учитывайте следующие факты:

- Вы можете создать, изменить, или удалить ограничения, без необходимости удаления или пересоздания таблицы.
- Вы должны встроить логику проверки ошибок в Ваши приложения и транзакции, чтобы проверить было ли нарушено ограничение.
- При добавлении новых ограничений FOREIGN KEY и CHECK к существующей непустой таблице Вы должны явно определить соответствующую опцию, если не хотите, чтобы SQL Server применял эти ограничения к существующим данным.

Совет. Для получения информации об ограничениях, выполняйте системные хранимые процедуры **sp_helpconstraint** или **sp_help**, или представления системного каталога, такие как

check_constraints, **referential_constraints**, и **table_constraints**. Кроме того, следующие системные таблицы хранят определения ограничений: **syscomments**, **sysreferences**, и **sysconstraints**.

Когда отключать ограничения

Вы можете отключать только ограничения FOREIGN KEY и CHECK. Другие ограничения необходимо удалять, а затем снова добавлять. Целесообразно отключать ограничения в следующих случаях:

- Вы должны запустить большое пакетное задание или импортировать данные, и хотите при этом оптимизировать производительность. Для гарантии того, что данные согласованы перед повторным включением ограничений, Вы должны быть уверены, что данные соответствуют отключаемым ограничениям или должны выполнить соответствующие запросы как часть пакетного задания.
- Вы можете определить ограничение на таблице, которая уже содержит данные. В результате каждая строка данных проверяется ограничением только после модификации.

Предупреждение. Отключение ограничения одной таблицы не затрагивает ограничения другой таблицы, которая ссылаются на первую. Поэтому, обновления таблицы все еще могут привести к ошибкам нарушения ограничений.

Как отключить ограничения

Когда Вы добавляете ограничение CHECK или FOREIGN KEY к таблице с существующими данными, Вы можете отключить проверку ограничения, включая в оператор ALTER TABLE опцию WITH NOCHECK. Существующие данные будут проверены только во время будущих обновлений столбца.

В следующем примере создается ограничение FOREIGN KEY на таблицу **Sales.SalesOrderHeader** и используется опция WITH NOCHECK чтобы отключить проверку существующих данных.

```
ALTER TABLE [Sales].[SalesOrderHeader] WITH NOCHECK ADD CONSTRAINT  
    [FK_SalesOrderHeader_Customer_CustomerID] FOREIGN KEY([CustomerID])  
REFERENCES [Sales].[Customer] ([CustomerID])
```

Вы можете отключить существующие ограничения CHECK и FOREIGN KEY так, чтобы не проверялись любые данные, которые Вы изменяете или добавляете в таблицу. Чтобы отключить CHECK и FOREIGN KEY, используйте опцию NOCHECK в операторе ALTER TABLE. Чтобы вернуть отключенное ограничение, выполните оператор ALTER TABLE, на сей раз с опцией CHECK.

В этом примере отключается ограничение **FK_SalesOrderHeader_Customer_CustomerID**.

```
ALTER TABLE [Sales].[SalesOrderHeader]
NOCHECK
CONSTRAINT [FK_SalesOrderHeader_Customer_CustomerID]
```

В этом примере включается проверка ограничения
FK_SalesOrderHeader_Customer_CustomerID.

```
ALTER TABLE [Sales].[SalesOrderHeader]
CHECK
CONSTRAINT [FK_SalesOrderHeader_Customer_CustomerID]
```

Совет. Чтобы определить, включено или выключено ограничение таблицы, выполните системную хранимую процедуру **sp_help**, или используйте свойство **CnstIsDisabled** функции OBJECTPROPERTY.

Урок 3: Реализация триггеров

- 
- What Are Triggers?
 - How an INSERT Trigger Works
 - How a DELETE Trigger Works
 - How an UPDATE Trigger Works
 - How an INSTEAD OF Trigger Works
 - How Nested Triggers Work
 - Considerations for Recursive Triggers
 - Practice: Creating Triggers

Цели урока

После завершения этого урока, студенты смогут:

- Определять триггеры DML.
- Описывать, как работает триггер вставки INSERT.
- Описывать, как работает триггер удаления DELETE.
- Описывать, как работает триггер обновления UPDATE.
- Описывать, как работает триггер INSTEAD OF.
- Описывать, как работают вложенные триггеры.
- Приводить основные сведения о рекурсивных триггерах.

Введение

Триггеры языка манипулирования данными (DML) - сильный инструмент, который позволяет Вам реализовать домен, сущность и ссылочную целостность данных. Из этого урока Вы узнаете, что такое триггеры DML и как они могут обеспечить целостность данных, а также

узнаете о различных типах триггеров, доступных для Вас, и как определить триггеры в Вашей базе данных.

Что такое триггер?

- **Special stored procedures that execute when INSERT, UPDATE, or DELETE statements modify a table**
- **Two categories:**
 - AFTER triggers execute after an INSERT, UPDATE, or DELETE statement
 - INSTEAD OF triggers execute instead of an INSERT, UPDATE, or DELETE statement
- **Trigger and the initiating statement are part of a single transaction**

Что такое триггер?

Триггер - специальный вид хранимой процедуры, которая выполняется когда оператор INSERT, UPDATE, или DELETE изменяет данные в указанной таблице. Триггер может запрашивать данные другой таблицы и может включать сложные операторы Transact-SQL. Триггеры часто создаются для обеспечения ссылочной целостности или согласованности среди логически связанных данных в различных таблицах. Поскольку пользователи не могут обойти триггеры, а Вам доступны эти возможности Transact-SQL, Вы можете использовать триггеры, чтобы реализовать сложную бизнес логику, которая является трудной или невозможной для реализации при использовании других механизмов поддержки целостности данных.

Рассмотрим следующие факты о триггерах:

- Триггер и оператор, на который он срабатывает, рассматривается как одна транзакция, которая может быть отменена внутри этого триггера. Если была обнаружена серьезная ошибка (например, недостаточно дисковой памяти), то вся транзакция автоматически откатывается.

- Триггеры могут каскадировать изменения через связанные таблицы базы данных; однако, эти изменения могут быть выполнены более эффективно при использовании каскадирования ссылочного ограничения целостности.
- Триггеры могут защищать против злонамеренной или неправильной операции вставки, обновления или удаления и реализовать другие ограничения, более сложные, чем при использовании ограничений CHECK.
- Триггеры могут ссылаться на столбцы других таблиц, в отличие от ограничений CHECK. Например, триггер может использовать оператор SELECT, выбирающий данные другой таблицы для сравнения с вставленными или обновленными данными, затем этот триггер может выполнить дополнительные действия, такие как изменение данных или вывод пользовательского сообщения об ошибке.
- Триггеры могут оценить состояние таблицы до и после модификации данных и выполнить действия, основанные на этом различии.
- Множество триггеров одного и того же типа (INSERT, UPDATE, или DELETE) на таблице позволяют в ответ на одну модификацию иметь место множеству различных действий.

Дополнительная информация. За дополнительной информацией о триггерах DML обращайтесь к разделу “DML Triggers” в SQL Server Books Online.

Создание триггеров

Вы можете создать триггеры при использовании оператора Transact-SQL CREATE TRIGGER. Оператор CREATE TRIGGER имеет следующий синтаксис.

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ...n ] | EXTERNAL NAME <method specifier [ ; ] > }
```

Дополнительная информация. За дополнительной информацией о создании триггеров обращайтесь к разделу "CREATE TRIGGER (Transact-SQL)" в SQL Server Books Online.

Типы триггеров

Существует две категории триггеров DML: AFTER и INSTEAD OF.

Триггеры AFTER. Триггеры AFTER выполняются *после* действия операторов INSERT, UPDATE или DELETE. Определение триггера AFTER имеет то же самое определение, что и триггер FOR, который являлся единственным доступным типом в более ранних версиях Microsoft SQL Server. Вы можете определять триггеры AFTER только для таблиц.

Триггеры INSTEAD OF. Триггеры INSTEAD OF выполняются *вместо* обычных триггерных действий (вставка, обновление, удаление). Триггеры INSTEAD OF также могут быть определены на представлениях с одной или более базисными таблицами, чтобы расширить действия операторов обновления, поддерживаемые представлениями.

Дополнительная информация. За дополнительной информацией о типах триггеров обратитесь к разделу "Types of DML Triggers" в SQL Server Books Online.

Триггеры vs ограничения

Главное преимущество триггеров заключается в том, что они могут содержать сложную логику обработки с использованием кода Transact-SQL. Поэтому, триггеры поддерживают больший набор функциональных возможностей по сравнению с ограничениями. Однако, целостность данных должна всегда реализовываться на самом низком уровне – уровне ограничений, до тех пор пока они отвечают требованиям приложения.

Триггеры являются наиболее полезными, когда возможности, поддерживаемые ограничениями, не могут отвечать функциональным потребностям приложения. Например:

- Ограничение FOREIGN KEY может проверять значение столбца только на точное соответствие значению другого столбца, если в разделе REFERENCES не определено каскадирование некоторого действия по ссылкам.
- Ограничения могут сообщать об ошибках только через стандартную систему сообщений об ошибках. Если Ваше приложение требует более сложной обработки ошибок, Вы должны использовать триггер.

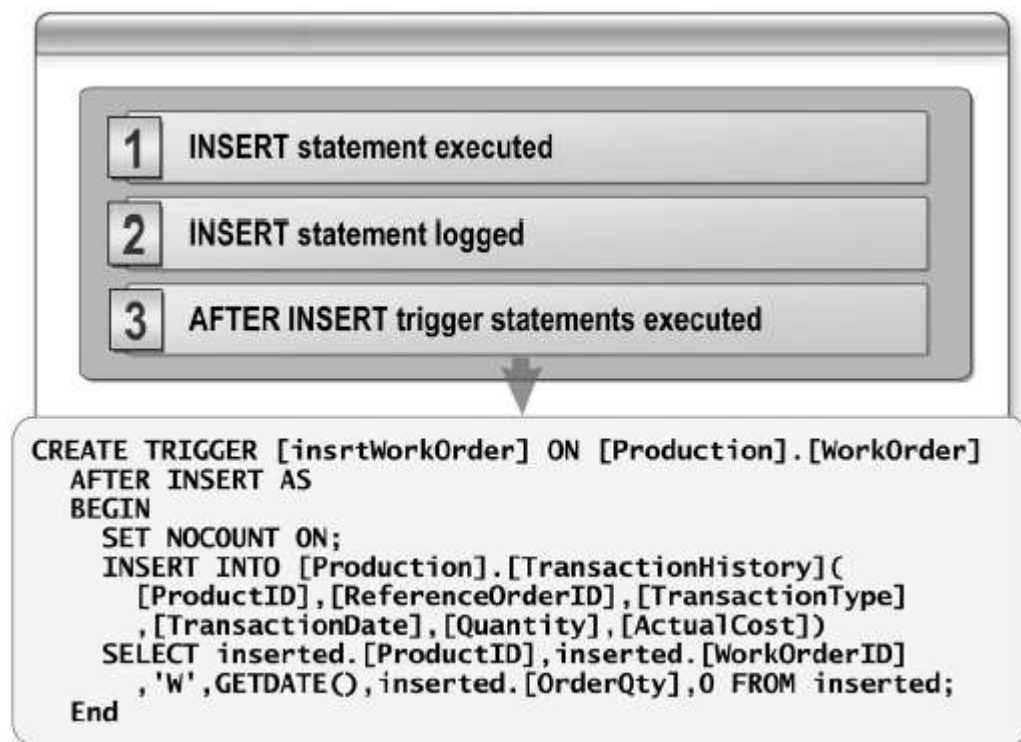
Триггеры могут каскадировать изменения через связанные таблицы в базе данных; однако, эти изменения могут быть выполнены более эффективно через ограничения:

- Триггеры могут откатить изменения, которые нарушают ссылочную целостность, таким образом, отменяя предпринятую модификацию данных. Такой триггер мог бы срабатывать, когда Вы изменяете внешний ключ, и его новое значение не соответствует первичному ключу. Однако для этой цели обычно используются ограничения FOREIGN KEY.

- Если на таблице с триггерами существуют ограничения, то они проверяются после триггера INSTEAD OF, но до триггера AFTER. Если ограничения нарушены, то действия триггера INSTEAD OF откатываются, а триггер AFTER не выполняется.

Дополнительная информация. За дополнительной информацией о сравнении триггеров и ограничений обратитесь к разделу “Triggers Compared to Constraints” в SQL Server Books Online.

Как работает триггер вставки INSERT



Определение

Триггер INSERT - триггер, который выполняется всякий раз, когда оператор INSERT вставляет данные в таблицу или представление, для которого определен этот триггер.

Как работает триггер вставки INSERT

Когда срабатывает триггер INSERT, новые строки добавляются и к основной таблице, и к таблице **inserted**. Таблица **inserted** - логическая таблица, которая содержит копии вставляемых строк. Таблица **inserted** содержит **журналированную деятельность** оператора INSERT. Таблица **inserted** позволяет Вам ссылаться на журнальные данные инициализации оператора INSERT. Триггер может проверять таблицу **inserted** для определения того, какие действия должны быть им выполнены. Строки в таблице **inserted** всегда являются дубликатами одной или нескольких строк в базовой таблице.

Любая деятельность модификации данных (операторы INSERT, UPDATE, DELETE) регистрируется в журнале транзакций, но его информацию нельзя считывать. Однако, таблица **inserted** позволяет Вам ссылаться на зарегистрированные изменения, которые вызвал оператор INSERT. Тогда Вы можете сравнить изменения (вставленные данные), чтобы

проверить их или предпринять дальнейшие действия. Кроме того, Вы можете ссылаться на вставленные данные, без необходимости сохранения информации в переменных.

Дополнительная информация. За дополнительной информацией о создании триггеров INSERT обратитесь к разделу “CREATE TRIGGER (Transact-SQL)” в SQL Server Books Online.

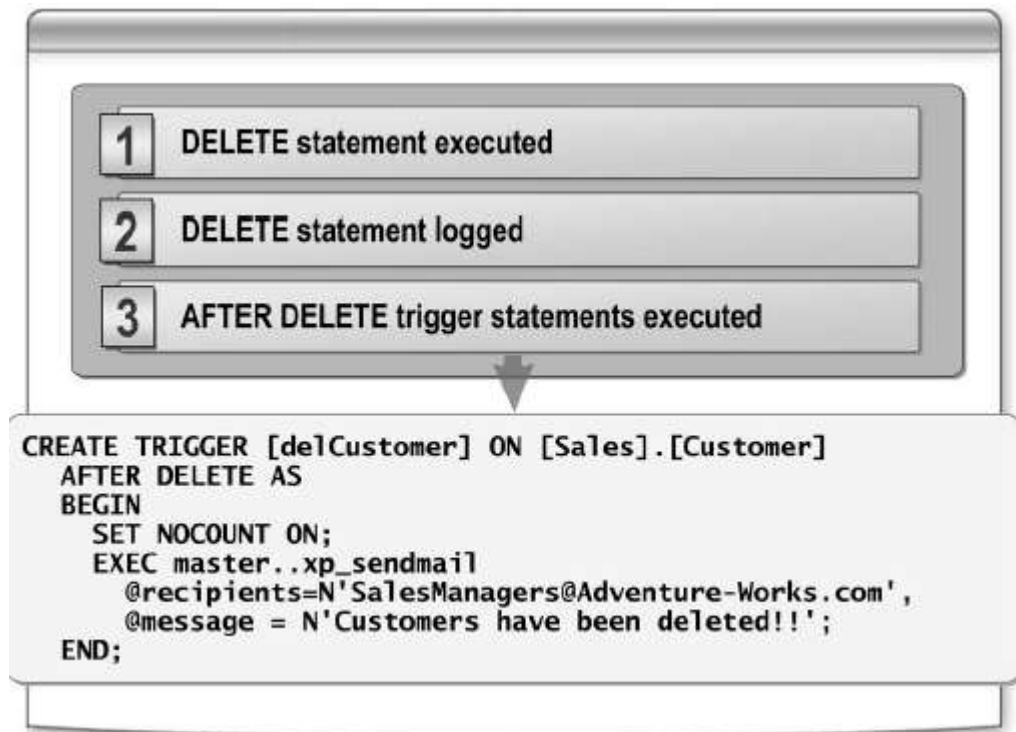
Реализация триггера INSERT

В следующем коде демонстрируется создание триггера INSERT **insrtWorkOrder** на таблице **Production.WorkOrder** базы данных **AdventureWorks**. Заметьте, что таблица **inserted** используется для работы со значениями, которые вызвали выполнение триггера.

```
CREATE TRIGGER [insrtWorkOrder] ON [Production].[WorkOrder]
AFTER INSERT AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO [Production].[TransactionHistory](
        [ProductID],[ReferenceOrderID],[TransactionType]
        ,[TransactionDate],[Quantity],[ActualCost])
    SELECT inserted. [ProductID],inserted. [WorkOrderID]
        , 'W', GETDATE(), inserted. [OrderQty], 0
    FROM inserted;
END;
```

Как работает триггер удаления DELETE



Определение

Триггер DELETE – специальный вид хранимой процедуры, которая выполняется всякий раз, когда оператор DELETE удаляет данные из таблицы или представления, на котором этот триггер был определен.

Как работает триггер удаления DELETE

Когда срабатывает триггер DELETE, удаленные строки таблицы помещаются в специальную таблицу **deleted**. Таблица **deleted** - логическая таблица, которая хранит копии удаленных строк. Таблица **deleted** позволяет сослаться на данные, записанные в журнал от инициализации оператора DELETE.

Примите к сведению следующие факты, когда Вы используете триггер DELETE:

- Когда к таблице **deleted** добавляется строка, она больше не существует в таблице базы данных; поэтому, у таблицы **deleted** и таблиц базы данных нет никаких общих строк.
- Для таблицы **deleted** выделяется память. Таблица **deleted** всегда в кеш-памяти.
- Триггер, который определен для действия DELETE, не выполняется для оператора TRUNCATE TABLE, так как TRUNCATE TABLE не журналируется.

Дополнительная информация. За дополнительной информацией о создании триггеров DELETE обращайтесь к разделу “CREATE TRIGGER (Transact-SQL)” в SQL Server Books Online.

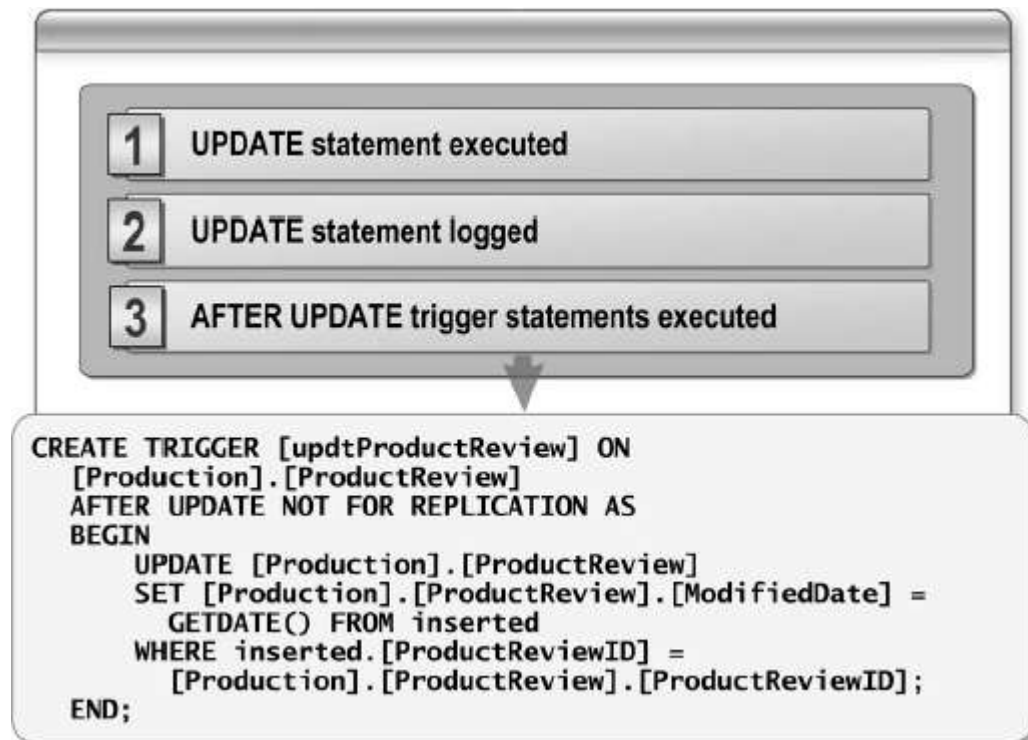
Реализация триггера DELETE

В следующем коде демонстрируется создание триггера DELETE **delCustomer** на таблице **Sales.Customer** базы данных **AdventureWorks**.

```
CREATE TRIGGER [delCustomer] ON [Sales].[Customer]
AFTER DELETE AS
BEGIN
    SET NOCOUNT ON;

    EXEC master..xp_sendmail
        @recipients=N'SalesManagers@Adventure-Works.com',
        @message = N'Customers have been deleted!!!';
END;
```


Как работает триггер обновления UPDATE



Определение

Триггер UPDATE - триггер, который выполняется всякий раз, когда оператор INSERT изменяет данные в таблице или представлении, на котором определен этот триггер.

Как работает триггер обновления UPDATE

Триггер обновления UPDATE логически выполняется в два шага:

1. Шаг УДАЛЕНИЯ, который охватывает исходный вид данных (строки до обновления).
2. Шаг ВСТАВКИ, который охватывает новый вид данных (строки после обновления).

Когда выполняется оператор UPDATE на таблице, для которой определен триггер обновления, то оригинальные строки (исходный вид данных) помещаются в таблицу **deleted**, а обновленные строки (новый вид данных) вставляются в таблицу **inserted**.

Триггер может исследовать таблицы **deleted** и **inserted**, так же как таблицу **updated**, для определения того, были ли строки обновлены, и какие действия должны быть выполнены.

Вы можете определить триггер, чтобы контролировать обновления данных на определенном столбце при использовании оператора IF UPDATE. Это позволяет триггеру легко изолировать деятельность для определенного столбца. Когда обнаруживается, что определенный столбец был обновлен, можно предпринять надлежащее действие, такое как генерация ошибочного сообщения о том, что столбец не может быть обновлен, или обработка серии операторов, основанных на недавно обновленном значении столбца.

Дополнительная информация. За дополнительной информацией о создании триггеров UPDATE обращайтесь к разделу “CREATE TRIGGER (Transact-SQL)” в SQL Server Books Online.

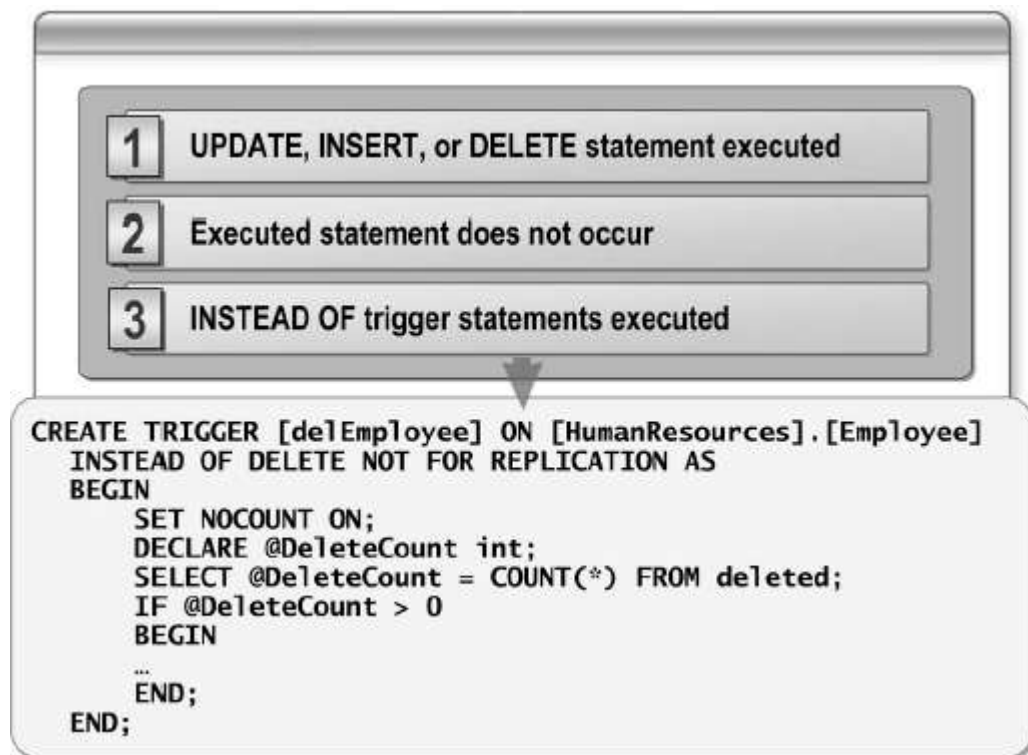
Реализация триггера UPDATE

В следующем коде демонстрируется создание триггера UPDATE **updtProductReview** на таблице **Production.ProductReview** базы данных **AdventureWorks**.

```
CREATE TRIGGER [updtProductReview] ON [Production].[ProductReview]
AFTER UPDATE NOT FOR REPLICATION AS
BEGIN
    SET NOCOUNT ON;

    UPDATE [Production].[ProductReview]
    SET [Production].[ProductReview].[ModifiedDate] = GETDATE()
    FROM inserted
    WHERE inserted.[ProductReviewID] =
[Production].[ProductReview].[ProductReviewID];
```

Как работает триггер INSTEAD OF (ВМЕСТО)



Определение

Триггер INSTEAD OF выполняется *вместо* обычного действия триггера. Триггеры INSTEAD OF могут также быть определены на представлениях с одной или более базовыми таблицами, где они могут расширить действия операторов обновления, поддерживаемые представлениями.

Как работает триггер INSTEAD OF (ВМЕСТО)

Этот триггер выполняется *вместо* исходного действия триггера. Триггеры INSTEAD OF увеличивают разнообразие типов обновлений, которые Вы можете выполнить с представлениями. Каждая таблица или представление ограничено одним триггером INSTEAD OF для каждого триггерного действия (INSERT, UPDATE, или DELETE).

Вы можете определить триггер INSTEAD OF и на таблицах и на представлениях. Вы не можете создать триггер INSTEAD OF на представлениях, которые имеют опцию WITH CHECK OPTION.

В следующем списке в общих чертах приводятся главные преимущества INSTEAD OF триггеров:

- Они позволяют представлениям, включающим множество базовых таблиц, поддерживать вставки, обновления, и удаления данных этих таблиц.
- Они позволяют Вам кодировать логику, которая может отклонить часть пакета, позволяя успешно выполниться другой части пакета.
- Они позволяют Вам определять альтернативное действие базы данных в ситуациях, отвечающих указанным условиям.

Дополнительная информация. За дополнительной информацией о создании триггеров INSTEAD OF обращайтесь к разделу "CREATE TRIGGER (Transact-SQL)" в SQL Server Books Online.

Реализация триггера INSTEAD OF

В следующем коде демонстрируется создание триггера INSTEAD OF **delEmployee** на таблице **HumanResources.Employee** база данных **AdventureWorks**.

```
CREATE TRIGGER [delEmployee] ON [HumanResources].[Employee]
INSTEAD OF DELETE NOT FOR REPLICATION AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @DeleteCount int;
    SELECT @DeleteCount = COUNT(*) FROM deleted;
    IF @DeleteCount > 0
    BEGIN
        RAISERROR
            (N'Employees cannot be deleted. They can only be marked as not current.', --
            Message
            10, -- Severity.
            1); -- State.
        -- Roll back any active or uncommittable transactions
        IF @@TRANCOUNT > 0
```

BEGIN

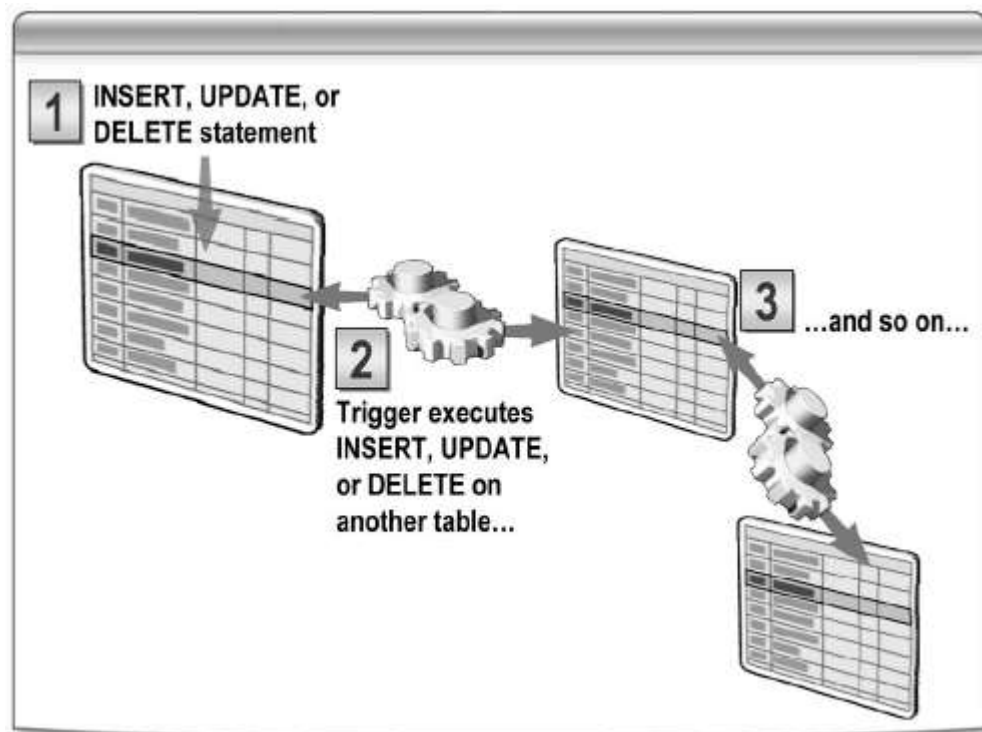
ROLLBACK TRANSACTION;

END

END;

END;

Как работают вложенные триггеры



Определение

Любой триггер может содержать оператор INSERT, UPDATE, или DELETE, который затрагивает другую таблицу. Триггеры называются вложенными, когда триггер выполняет действие, которое начинает другой триггер.

Как работают вложенные триггеры

Вы можете управлять вложенностью триггеров при использовании опции конфигурации сервера вложенных триггеров. Вложенность доступна при установке и устанавливается на уровне сервера, но Вы можете отменить и повторно сделать ее доступной при использовании системной хранимой процедуры **sp_configure**.

Триггеры могут быть вложены на глубину в 32 уровня. Когда триггер в цепочке вложений закидывается, то превышает способность вложения. Тогда триггер останавливается, и транзакция откатывается. Вы можете использовать вложенные триггеры, чтобы выполнить такие функции, как восстановление копий строк, которые были испорчены предыдущим триггером.

Примите к сведению следующие факты, когда Вы используете вложенные триггеры:

- По умолчанию опция конфигурации вложенности триггеров установлена в ON.
 - Вложенный триггер не будет срабатывать дважды на одну и ту же транзакцию триггера; триггер не вызывает сам себя в ответ на второе обновление той же самой таблицы в пределах триггера. Однако если триггер изменяет таблицу, что заставляет другой триггер срабатывать, а второй триггер изменяет эту таблицу, то первый триггер будет срабатывать рекурсивно. Чтобы предотвратить косвенную рекурсию такого вида, выключите опцию вложенных триггеров.
 - Поскольку триггер – это транзакция, отказ на любом уровне ряда вложенных триггеров отменяет всю транзакцию, и все модификации данных откатываются.
- Поэтому необходимо включать операторы PRINT, когда Вы тестируете триггеры, чтобы Вы могли определить, где произошел отказ.

Уровни вложенности

Уровень вложения увеличивается при каждом срабатывании вложенных триггеров. SQL Server поддерживает до 32 уровней вложения, но Вы могли бы ограничить уровни вложения, чтобы избежать превышения максимального уровня вложения. Вы можете использовать функцию @@NESTLEVEL, чтобы видеть текущие уровни вложения.

Отмена вложенных триггеров

Вложения является сильной особенностью, которую Вы можете использовать, чтобы поддержать целостность данных по всей базе данных. Иногда, однако, Вы хотели бы отменить вложение. Если вложение отменено, триггеры не могут выполняться каскадом (триггер не может выполнять действие, на которое срабатывает другой триггер, который инициирует следующий триггер, и так далее).

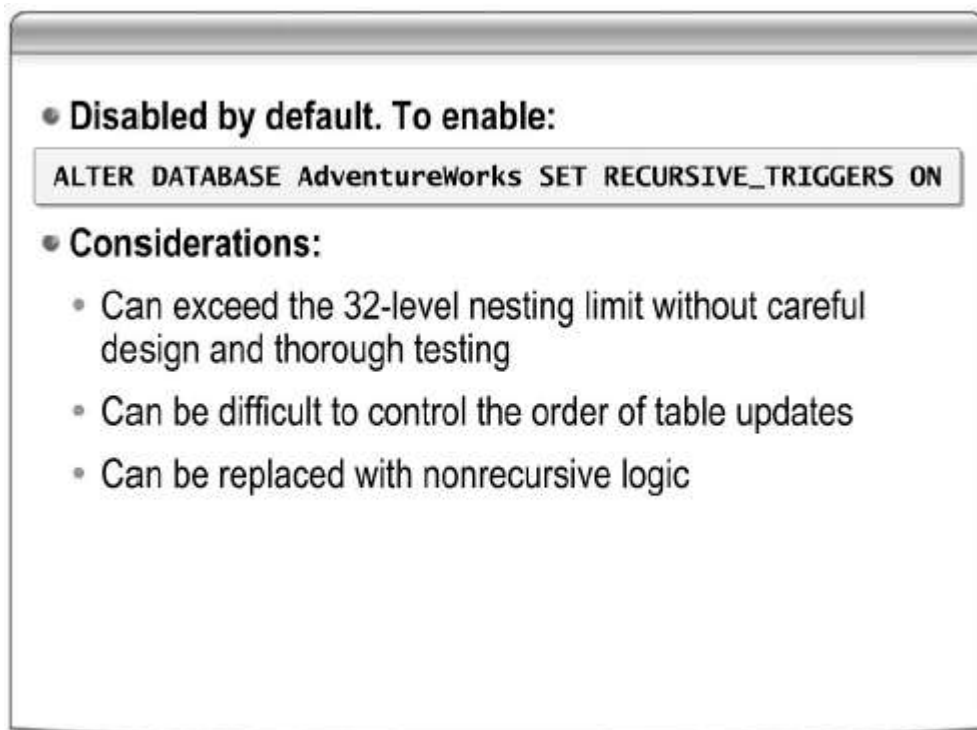
Вы могли бы решить отменить вложение потому что:

- Вложенные триггеры требуют сложного и хорошо запланированного проекта. Каскадирование изменений может привести к непреднамеренным модификациям данных.
- Модификация данных в любой точке в серии вложенных триггеров отменяет эти серии триггеров. Хотя это предлагает мощную защиту для Ваших данных, может возникнуть проблема, если Ваши таблицы должны обновляться в определенной последовательности.

Используйте следующий оператор, чтобы отменить вложение.

```
sp_configure 'nested triggers', 0
```


Основные сведения о рекурсивных триггерах



Определение

Рекурсивный триггер - триггер, который выполняет действие, которое прямо или косвенно заставляет тот же самый триггер срабатывать снова. Любой триггер может содержать оператор INSERT, UPDATE, или DELETE, который затрагивает ту же самую таблицу или другую таблицу. С установленной опцией рекурсивных триггеров триггер, который изменяет данные в таблице, может активизировать себя снова, выполняясь рекурсивно.

Существует два типа рекурсии: прямая и косвенная.

Прямая рекурсия. Прямая рекурсия происходит, когда триггер запускает и выполняет действие на той же самой таблице, которое заставляет тот же самый триггер срабатывать снова. Например, приложение обновляет таблицу **T1**, что заставляет срабатывать триггер **Trig1**. **Trig1** обновляет таблицу **T1** снова, что заставляет снова срабатывать триггер **Trig1**.

Косвенная рекурсия. Косвенная рекурсия происходит, когда триггер запускается и выполняет действие, которое заставляет срабатывать другой триггер (в той же самой или другой таблице), который обновляет на первую таблицу. Тогда первый триггер срабатывает снова. Например, приложение обновляет таблицу **T2**, что вызывает срабатывание триггера **Trig2**. **Trig2** обновляет таблицу **T3**, что заставляет срабатывать триггер **Trig3**. **Trig3** в свою очередь обновляет таблицу **T2**, что заставляет **Trig2** срабатывать снова.

Как управлять рекурсивным вызовом

При создании базы данных опция рекурсивных триггеров по умолчанию запрещена, но Вы можете разрешить использование рекурсивных триггеров с помощью оператора изменения базы данных (ALTER DATABASE). Триггер не вызывает себя рекурсивно, пока не установлена опция базы данных RECURSIVE_TRIGGERS. Используйте следующий оператор, чтобы разрешить рекурсивные триггеры.

```
ALTER DATABASE AdventureWorks SET RECURSIVE_TRIGGERS ON  
sp_dboption databasename, 'recursive triggers', True
```

Если вложенные триггеры запрещены, рекурсивные триггеры - также запрещены, независимо от установки опции рекурсивных триггеров в базе данных. Если вложенные триггеры разрешены, то косвенная рекурсия допустима, даже при выключенной опции рекурсивных триггеров. Опция рекурсивных триггеров влияет только на прямую рекурсию. (См. вышеописанные в этой теме определения прямой и косвенной рекурсии).

Таблицы **inserted** и **deleted** для данного триггера содержат строки, которые соответствуют только последнему оператору UPDATE, INSERT, или DELETE, который вызвал этот триггер.

Триггерная рекурсия может произойти на глубину до 32 уровней. Если триггерная рекурсия бесконечна, то как только будет превышено 32 уровня, автоматически произойдет завершение триггера и откат транзакции.

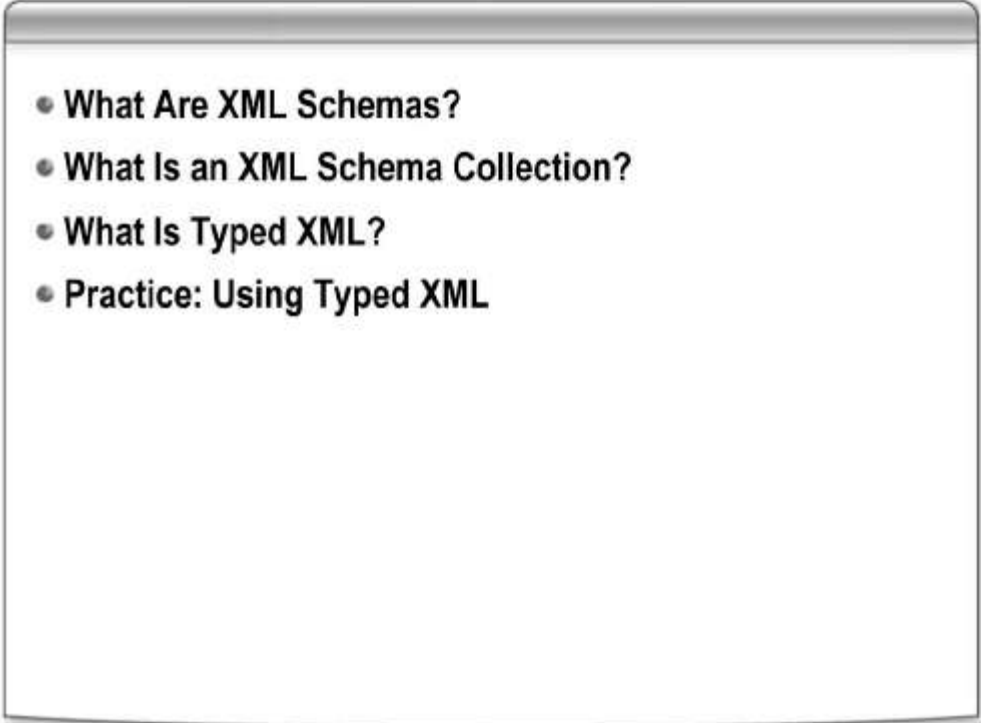
Основные сведения о рекурсивных триггерах

Рекурсивные триггеры - сложная особенность, которую Вы можете использовать, чтобы разрешить сложные связи, такие как рекурсивные ссылки (также известные как транзитивные тупики). Прежде, чем Вы будете использовать рекурсивные триггеры, примите к сведению следующие основные принципы:

- Рекурсивные триггеры сложны и должны быть хорошо разработаны и полностью проверены. Рекурсивные триггеры требуют контролируемого кода циклической логики (правил завершения). Иначе, Вы превысите 32-х уровневый предел вложения.
- Модификация данных в любой точке может выделить серию триггеров. Хотя обеспечивается возможность обработать сложные связи, может возникнуть проблема, если Ваши таблицы должны обновляться в определенном порядке.

■ Вы может создать подобные функциональные возможности без рекурсивных триггеров; однако, Ваш проект с триггерами будет существенно отличаться. При проектировании рекурсивных триггеров, каждый триггер должен содержать проверку условия, которая остановит рекурсивную обработку, когда условие станет ложным. Проект нерекурсивных триггеров должен содержать полные программные структуры организации циклов и проверки условий.

Урок 4: Реализация схемы XML

- 
- What Are XML Schemas?
 - What Is an XML Schema Collection?
 - What Is Typed XML?
 - Practice: Using Typed XML

Цели урока

После завершения этого урока, студенты смогут:

- Описать цель использования схем XML.
- Объяснить цель использования коллекции схем XML.
- Определить типизированный XML и описать, как он работает.

Введение

Тип данных **xml** поддерживает типизированный и нетипизированный XML. Чтобы использовать типизированный XML, Вы должны определить схемы XML и связать эти схемы со столбцом, параметром, или переменной типа **xml** при использовании коллекции схем XML. На этом уроке Вы узнаете, что представляют собой схемы XML и коллекции схем XML, а также, как использовать их для реализации данных типизированного XML.

Что такое схемы XML?

- Defines the elements and attributes that are valid in an XML document
- Uses the XML Schema syntax defined by the W3C

Определение

Схема XML определяет элементы и атрибуты, которые разрешены в любом документе XML, связанном с этой схемой.

Синтаксис схемы XML

Синтаксис схемы XML определен организацией World Wide Web Consortium (W3C). Схема XML содержится в элементе самого верхнего уровня **schema** и идентифицируется определением **targetNamespace**. Вот пример определения схемы XML.

```
<xsd:schema targetNamespace="http://schemas.microsoft.com/sqlserver/2004/
    07/adventure-works/ProductModelManuInstructions"
    xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventureworks/
ProductModelManuInstructions"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
```

```
<xsd:complexType name="StepType" mixed="true" >
```

```
  <xsd:choice minOccurs="0" maxOccurs="unbounded" >
```

```
    <xsd:element name="tool" type="xsd:string" />
```

```
    <xsd:element name="material" type="xsd:string" />
```

```
    <xsd:element name="blueprint" type="xsd:string" />
```

```
    <xsd:element name="specs" type="xsd:string" />
```

```
    <xsd:element name="diag" type="xsd:string" />
```

```
  </xsd:choice>
```

```
</xsd:complexType>
```

```
<xsd:element name="root">
```

```
  <xsd:complexType mixed="true">
```

```
    <xsd:sequence>
```

```
      <xsd:element name="Location" minOccurs="1"
```

```
        maxOccurs="unbounded">
```

```
        <xsd:complexType mixed="true">
```

```
          <xsd:sequence>
```

```
            <xsd:element name="step" type="StepType"
```

```
              minOccurs="1" maxOccurs="unbounded" />
```

```
          </xsd:sequence>
```

```
        <xsd:attribute name="LocationID" type="xsd:integer"
```

```
          use="required"/>
```

```
        <xsd:attribute name="SetupHours" type="xsd:decimal"
```

```
          use="optional"/>
```

```
        <xsd:attribute name="MachineHours" type="xsd:decimal"
```

```
          use="optional"/>
```

```
        <xsd:attribute name="LaborHours" type="xsd:decimal"
```

```
        use="optional"/>
        <xsd:attribute name="LotSize" type="xsd:decimal"
        use="optional"/>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Дополнительная информация. Дополнительную информацию о синтаксисе схемы XML см. в разделе “XML Schema” на вебсайте W3C.

Что такое коллекция схем XML?

- **Named collection of one or more XML schemas**
- **Associated with columns or variables of xml data type to provide typed XML capability**
- **Created by using CREATE XML SCHEMA COLLECTION statement**

Определение

Схемы XML регистрируются в объектах коллекции схем XML в базе данных. Каждая коллекция схем XML может содержать одну или несколько схем XML, каждая из которых идентифицируется при использовании пространства имен (определяется атрибутом схемы **targetNamespace**).

Как создать коллекции схем XML

Чтобы создать коллекцию схемы XML, используйте оператор CREATE XML SCHEMA COLLECTION (создание коллекции схемы XML). Синтаксис для оператора CREATE XML SCHEMA COLLECTION:

CREATE XML SCHEMA COLLECTION *sql_identifier* **AS** *Expression*

Параметры синтаксиса CREATE XML SCHEMA COLLECTION описаны в следующей таблице.

Параметр	Описание
<i>sql_identifier</i>	Идентификатор Transact-SQL для коллекции схемы XML
<i>Expression</i>	Значение XML, содержащее один или более документов схемы XML

В следующем примере показывается, как использовать оператор CREATE XML SCHEMA COLLECTION, чтобы зарегистрировать коллекцию схемы XML по имени **ResumeSchemaCollection**, которая содержит единственную схему в пространстве имен <http://schemas.adventureworks.com/EmployeeResume>.

```
CREATE XML SCHEMA COLLECTION ResumeSchemaCollection
AS
N'<?xml version="1.0" ?>
  <xsd:schema
    targetNamespace=
      "http://schemas.adventure-works.com/EmployeeResume"
    xmlns="http://schemas.adventure-works.com/EmployeeResume"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
```

```

<xsd:element name="resume">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="employmentHistory">
        <xsd:complexType>
          <xsd:sequence minOccurs="1"
            maxOccurs="unbounded">
            <xsd:element name="employer">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="xsd:string">
                    <xsd:attribute name="endDate"
                      use="optional"/>
                  </xsd:extension>
                </xsd:simpleContent>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema> '

```

Вы можете изменить коллекцию схемы XML при использовании оператора ALTER XML SCHEMA COLLECTION. Вы можете использовать оператор ALTER XML SCHEMA COLLECTION, чтобы добавить или удалить схемы из коллекции схем.

Вы можете удалить коллекцию схем XML при использовании оператора DROP XML SCHEMA COLLECTION, как показано в следующем примере.

DROP XML SCHEMA COLLECTION ResumeSchemaCollection

Вы не можете удалить коллекцию схем XML, содержащую схемы, на которые ссылаются типизированные **xml** столбцы. Вы должны сначала удалить ссылку, изменив или удалив соответствующую таблицу.

Как получить информацию о схеме XML

Вы можете получить информацию о коллекциях схем в базе данных, сделав запрос к представлению системного каталога **sys.xml_schema_collections**, как показано в следующем примере.

```
SELECT * FROM sys.xml_schema_collections
```

Также, Вы можете получить индивидуальные пространства имен XML, определенные в коллекциях схем базы данных, сделав запрос к представлению системного каталога **sys.xml_schema_namespaces**, как показано в следующем примере.

```
SELECT * FROM sys.xml_schema_namespaces
```

Вы можете просмотреть компоненты XML, определенные в базе данных, с помощью представления системного каталога **sys.xml_schema_components**, как показано в следующем примере.

```
SELECT * FROM sys.xml_schema_components
```

Что такое типизированный XML?

- Columns or variables of the xml data type that are associated with an XML schema collection
- SQL Server validates the contained XML against XML schemas in the associated XML schema collection

Определение

Типизированный XML – XML, который связан со схемой XML. Когда тип данных **xml** используется, чтобы хранить типизированный XML, SQL Server проверяет схему XML и оптимизирует внутреннее хранение данных, связывая соответствующие типы данных SQL Server с типами данных XML, определенных в схеме.

Ссылки на схему

Чтобы использовать типизированный XML, Вы должны объявить **xml** столбец или переменную и связать это с коллекцией схем XML, которая содержит схему для проверки XML.

Следующий пример показывает, как определить таблицу с использованием **xml** столбца, связанного с коллекцией схем XML **ResumeSchemaCollection**.

```
CREATE TABLE HumanResources.EmployeeResume  
(EmployeeID int,  
Resume xml (ResumeSchemaCollection))
```

Вы можете объявить типизированные **xml** переменные, как показано в следующем примере.

```
DECLARE @resumeDoc xml (ResumeSchemaCollection)
```

Как SQL Server проверяет типизированный XML

Вы присваиваете значение типизированному **xml** столбцу или переменной таким же образом, что Вы присваиваете нетипизированное значение. Однако, строка XML, которую Вы присваиваете, должна соответствовать схеме в коллекции схем XML, связанной со столбцом или переменной и должна быть объявлена в пределах того же самого пространства имен.

Следующий пример показывает, как назначить типизированное значение **xml**.

```
INSERT INTO HumanResources.EmployeeResume
```

```
VALUES
```

```
(1,
```

```
'<?xml version="1.0" ?>
```

```
<resume
```

```
    xmlns="http://schemas.adventure-works.com/EmployeeResume">
```

```
    <name>Guy Gilbert</name>
```

```
    <employmentHistory>
```

```
        <employer endDate="2000-07-07">Northwind Traders</employer>
```

```
        <employer>Adventure Works</employer>
```

```
    </employmentHistory>
```

```
</resume>')
```

CONTENT или DOCUMENT

Вы можете управлять, ограничивается ли значение типизированного **xml** столбца или переменной одним документом или может содержать фрагмент, составленный из множества документов, с помощью спецификаций **DOCUMENT** или **CONTENT** в определении. Спецификация **CONTENT** позволяет правильные фрагменты, в то время как **DOCUMENT**

гарантирует, что может быть присвоен только единственный экземпляр документа. Если ни одно из этих ключевых слов не указано, по умолчанию принимается CONTENT. Следующий пример показывает, как использовать ключевое слово DOCUMENT, чтобы ограничить допустимые значения в столбце единственным правильным документом.

```
CREATE TABLE HumanResources.EmployeeResume  
(EmployeeID int,  
Resume xml (DOCUMENT ResumeSchemaCollection))
```