Лекция 10 Обеспечение целостности данных. Триггеры.

Обеспечение целостности данных

Во время проектирования базы данных вы должны заботится о целостности данных. Правильная структура таблиц позволяет защитить данные от нарушения связей и внесения неверных значений. Вы должны определить наилучший путь обеспечения целостности данных. Целостность данных основывается на стойкости и точности данных, которые хранит база данных.

Существуют различные типы целостности данных:

- Целостность полей – указывает набор значений данных, которые являются правильными для поля, и определяет, возможно ли использование нулевого значения. Например, поле для хранения пола человека может содержать одно из двух значений – М или Ж. Во-первых, этого достаточно, во-вторых, других значений пола просто не бывает и мы должны запретить ввод других букв в данное поле. Целостность полей часто всего (и лучше) обеспечивается с помощью ограничения CHECK, формата (с помощью шаблона) или региона возможных значений для поля.
- Целостность таблицы – требуют, чтобы все строки в таблице имели уникальный идентификатор, называемый первичным ключом. Может ли первичный ключ изменяться, или может ли строка удаляться, зависит от уровня целостности. Например, в некоторых случаях можно разрешить удаление записей, но чаще всего оно должно быть запрещено. Не желательно терять данные, потому что мы в последствии не сможем узнать историю изменений в таблице.
- Целостность ссылок – подразумевает отношения между первичным ключом (таблицы, на которую ссылаются) и внешним ключом (таблицы, которая ссылается на другую) всегда защищенными. Строка основной таблицы, на которую ссылаются, не может быть удалена и первичный ключ не может быть изменен, если вторичный ключ ссылается на строку, пока не будет уничтожена связь. Иначе связь нарушается и восстановить ее потом становится проблематичным. Вы можете назначить отношения внутри таблицы или между несколькими отдельными таблицами с помощью встроенных в SQL Server средств, не надеясь на возможности языка программирования, который вы используете для доступа к данным. Конечно же, связь между таблицами можно навести и без внешних ключей, но в этом случае сервер не гарантирует целостность. Вся ответственность ложиться на программиста.

Все операторы, необходимые для реализации всех уровней целостности нам уже знакомы. Я специально вынес рассмотрение теории

обеспечения целостности после того, как мы узнали средства. Знание операторов упростит понимание теоретических данных.

Как мы можем гарантировать целостность данных? Для этого существует два способа: описанная целостность данных и предшествующая целостность данных. Пока эти понятия не понятны, но после того, как вы увидите, какими средствами достигается тот, или иной способ, все встанет на свои места.

Описанная целостность данных – вы объявляете критерии, которые данные должны содержать как часть описания объекта и после этого SQL Server автоматически гарантирует, что данные соответствуют критериям. Уже можно догадаться, что такая целостность обеспечивается с помощью ограничений CHECK, DEFAULT и внешнего ключа.

Описанная целостность является частью объявления базы данных, и объявляется с помощью ограничений, которые вы можете назначить колонкам и таблицам напрямую.

Предшествующая целостность данных – это программа, которая определяет критерии, которым должны соответствовать данные. Этот метод обеспечивается с помощью процедур и триггеров (о них мы поговорим в главе 3), которые могут выполняться на сервере или с помощью кода программ в клиентском приложении.

Вы должны минимизировать использование этого метода для упрощения бизнес логики и ошибок, но иногда без триггера не возможно гарантировать, что таблицы будут содержать нужные или разрешенные значения.

Ограничение – это основной метод обеспечения целостности данных. В этой секции описывается, как определить, какой тип ограничения использовать, какой тип данных и для какого ограничения использовать, и как определить ограничения.

CREATE TRIGGER (Transact-SQL)

Создает триггер языка обработки данных, DDL или входа. Триггер — это особая разновидность хранимой процедуры, выполняемая автоматически при возникновении события на сервере базы данных. Триггеры языка обработки данных выполняются по событиям, вызванным попыткой пользователя изменить данные с помощью языка обработки данных. Событиями DML являются процедуры INSERT, UPDATE или DELETE, применяемые к таблице или представлению. Эти триггеры срабатывают при запуске любого допустимого события независимо от того, влияет ли оно на какие-либо строки таблицы. Дополнительные сведения см. в разделе DML Triggers.

Триггеры DDL срабатывают в ответ на ряд событий языка описания данных (DDL). Эти события прежде всего соответствуют инструкциям Transact-SQL CREATE, ALTER, DROP и некоторым системным хранимым процедурам, которые выполняют схожие с DDL операции. Триггеры входа могут срабатывать в ответ на событие LOGON, возникающее при установлении пользовательских сеансов. Триггеры могут создаваться непосредственно из инструкцийTransact-SQL или методов сборок, созданных в среде CLR платформы Microsoft .NET Frameworkи переданных экземпляру SQL Server. SQL Server позволяет создавать несколько триггеров для любой инструкции.

Важно!

Вредоносный программный код внутри триггеров может быть запущен с расширенными правами доступа. Дополнительные сведения о том, как уменьшить эту угрозу, см. в статье **Управление безопасностью триггеров**.

Примечание

В этом разделе рассматривается интеграция среды CLR .NET Framework с SQL Server.Интеграция со средой CLR не применяется к базе данных SQL Azure.

Синтаксические обозначения в Transact-SQL

Синтаксис

Копировать

```
-- SQL Server Syntax
-- Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML
Trigger)

CREATE [ OR ALTER ] 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 [ ; ] >
}

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]

<method_specifier> ::=
    assembly_name.class_name.method_name
```

Копировать

```
-- SQL Server Syntax
-- Trigger on an INSERT, UPDATE, or DELETE statement to a
-- table (DML Trigger on memory-optimized tables)

CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
```

```
ON { table }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_statement  [ ; ] [ ,...n ] }

<dml_trigger_option> ::=
    [ NATIVE_COMPILATION ]
    [ SCHEMABINDING ]
    [ EXECUTE AS Clause ]
```

Копировать

```
-- Trigger on a CREATE, ALTER, DROP, GRANT, DENY,
-- REVOKE or UPDATE statement (DDL Trigger)

CREATE [ OR ALTER ] TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier >  [ ;
] }

<ddl_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

Копировать

```
-- Trigger on a LOGON event (Logon Trigger)

CREATE [ OR ALTER ] TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR| AFTER } LOGON
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier >  [ ;
] }

<logon_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

Синтаксис

Копировать

```
-- Azure SQL Database Syntax
-- Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML
Trigger)

CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
 [ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
  AS { sql_statement  [ ; ] [ ,...n ] [ ; ] > }
```

```
<dml_trigger_option> ::=
        [ EXECUTE AS Clause ]
```
Копировать
```
-- Azure SQL Database Syntax
-- Trigger on a CREATE, ALTER, DROP, GRANT, DENY,
-- REVOKE, or UPDATE STATISTICS statement (DDL Trigger)

CREATE [ OR ALTER ] TRIGGER trigger_name
ON { DATABASE }
 [ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement  [ ; ] [ ,...n ]  [ ; ] }

<ddl_trigger_option> ::=
    [ EXECUTE AS Clause ]
```

Аргументы

OR ALTER
Применимо к: Azure База данных SQL, SQL Server (начиная с SQL Server 2016 (13.x) с пакетом обновления 1 (SP1)).

Условно изменяет триггер только в том случае, если он уже существует.

schema_name
Имя схемы, которой принадлежит триггер DML. Действие триггеров DML ограничивается областью схемы таблицы или представления, для которых они созданы. Аргумент *schema_name* не может указываться для триггеров DDL или триггеров входа.

trigger_name
Имя триггера. Аргумент *trigger_name* должен соответствовать правилам для идентификаторов— за исключением того, что *trigger_name* не может начинаться с символов # или ##.

table | *view*
Таблица или представление, в которых выполняется триггер DML, иногда указывается как таблица триггера или представление триггера. Указание уточненного имени таблицы или представления не является обязательным. На представление может ссылаться только триггер INSTEAD OF. Триггеры DML не могут быть описаны в локальной или глобальной временных таблицах.

DATABASE
Применяет область действия триггера DDL к текущей базе данных. Если этот аргумент определен, триггер срабатывает всякий раз при возникновении в базе данных события типа *event_type* или *event_group*.

ALL SERVER
Применимо к: с SQL Server 2008 до SQL Server 2017.

Применяет область действия триггера DDL или триггера входа к текущему серверу. Если этот аргумент определен, триггер срабатывает всякий раз при возникновении на текущем сервере события типа *event_type* или *event_group*.

WITH ENCRYPTION
Применимо к: с SQL Server 2008 до SQL Server 2017.

Затемняет текст инструкции CREATE TRIGGER. Использование параметра WITH ENCRYPTION не позволяет публиковать триггер как часть репликации SQL Server. Параметр WITH ENCRYPTION не может быть указан для триггеров CLR.

EXECUTE AS
Указывает контекст безопасности, в котором выполняется триггер. Позволяет управлять учетной записью пользователя, используемой экземпляром SQL Server для проверки разрешений на любые объекты базы данных, ссылаемые триггером.

Этот параметр является обязательным для триггеров в таблицах, оптимизированных для памяти.

Дополнительные сведения см. в разделе Предложение EXECUTE AS (Transact-SQL).

NATIVE_COMPILATION
Указывает, что триггер компилируется в собственном коде.

Этот параметр является обязательным для триггеров в таблицах, оптимизированных для памяти.

SCHEMABINDING
Гарантирует, что таблицы, на которые ссылается триггер, нельзя удалить или изменить.

Этот параметр является обязательным для триггеров в таблицах, оптимизированных для памяти, и не поддерживается для триггеров в обычных таблицах.

FOR | AFTER
Тип AFTER указывает, что триггер DML срабатывает только после успешного выполнения всех операций в инструкции SQL, запускаемой триггером. Все каскадные действия и проверки ограничений, на которые имеется ссылка, должны быть успешно завершены, прежде чем триггер сработает.

Если единственным заданным ключевым словом является FOR, аргумент AFTER используется по умолчанию.

Триггеры AFTER не могут быть определены на представлениях.

INSTEAD OF
Указывает, что триггер DML срабатывает *вместо* инструкции SQL, используемой триггером, переопределяя таким образом действия инструкций триггера. Аргумент INSTEAD OF не может быть указан для триггеров DDL или триггеров входа.

На каждую инструкцию INSERT, UPDATE или DELETE в таблице или представлении может быть определено не более одного триггера INSTEAD OF. Однако можно определить представления на представлениях, где у каждого представления есть собственный триггер INSTEAD OF.

Использование триггеров INSTEAD OF не допускается в поддерживающих обновление представлениях, которые используют параметр WITH CHECK OPTION. SQL Server вызывает ошибку, если триггер INSTEAD OF добавляется к поддерживающему обновление представлению с параметром WITH CHECK OPTION. Пользователь должен удалить этот параметр при помощи инструкции ALTER VIEW перед определением триггера INSTEAD OF.

{ [DELETE] [,] [INSERT] [,] [UPDATE] }
Определяет инструкции изменения данных, по которым срабатывает триггер DML, если он применяется к таблице или представлению. Необходимо указать как минимум одну инструкцию. В определении триггера разрешены любые их сочетания в любом порядке.

Для триггеров INSTEAD OF параметр DELETE не разрешен в таблицах, имеющих ссылочную связь с указанием каскадного действия ON DELETE. Аналогично, параметр UPDATE не разрешен в таблицах, у которых есть ссылочная связь с указанием каскадного действия ON UPDATE.

WITH APPEND
Применимо к: с SQL Server 2008 до SQL Server 2008 R2.

Указывает, что требуется добавить триггер существующего типа. Аргумент WITH APPEND не может быть использован для триггеров INSTEAD OF или при явном указании триггера AFTER.Аргумент WITH APPEND может использоваться только при указании параметра FOR без INSTEAD OF или AFTER из соображений поддержки обратной совместимости. Аргумент WITH APPEND не может быть указан, если указан параметр EXTERNAL NAME (в случае триггера CLR).

event_type
Имя языкового события Transact-SQL, которое после выполнения вызывает срабатывание триггера DDL. Список событий, которые могут быть использованы в триггерах DDL, приведен в разделе DDL-события.

event_group
Имя стандартной группы событий языка Transact-SQL. Триггер DDL срабатывает после возникновения любого события языка Transact-SQL, принадлежащего к группе *event_group*.Список групп событий, которые могут быть использованы в триггерах DDL, приведен в разделе Группы DDL-событий.

После завершения инструкции CREATE TRIGGER параметр *event_group* работает в режиме макроса, добавляя охватываемые им типы события в представление каталога sys.trigger_events.

NOT FOR REPLICATION
Применимо к: с SQL Server 2008 до SQL Server 2017.

Указывает, что триггер не может быть выполнен, если агент репликации изменяет таблицу, используемую триггером.

sql_statement
Условия и действия триггера. Условия триггера указывают дополнительные критерии, определяющие, какие события — DML, DDL или событие входа — вызывают срабатывание триггера.

Действия триггера, указанные в инструкциях языка Transact-SQL, вступают в силу после попытки использования операции.

Триггеры могут содержать любое количество инструкций языка Transact-SQL любого типа, за некоторыми исключениями. Дополнительные сведения см. в подразделе "Примечания".Триггеры разработаны для контроля или изменения данных на основании инструкций модификации или определения данных; они не возвращают пользователю никаких данных.Инструкции языка Transact-SQL в составе триггера часто содержат выражения языка управления потоком.

Триггеры DML используют логические (концептуальные) таблицы deleted и inserted. По своей структуре они подобны таблице, на которой определен триггер, то есть таблице, к которой применяется действие пользователя. В таблицах deleted и inserted содержатся старые или новые значения строк, которые могут быть изменены действиями пользователя. Например, для запроса всех значений таблицы `deleted` можно использовать инструкцию:
SQLКопировать
```
SELECT * FROM deleted;
```

Дополнительные сведения см. в разделе Использование таблиц inserted и deleted.

Триггеры DDL и триггеры входа собирают сведения о запускающих событиях с помощью функции EVENTDATA (Transact-SQL). Дополнительные сведения см. в разделе Использование функции EVENTDATA.

SQL Server позволяет обновлять столбцы типа **text**, **ntext** или **image** с помощью триггера INSTEAD OF в таблицах или представлениях.
Важно!

Типы данных **ntext**, **text** и **image** будут исключены в следующей версии Microsoft SQL Server. Следует избегать использования этих типов данных при новой разработке и запланировать изменение приложений, использующих их в настоящий момент. Вместо них следует использовать типы данных **nvarchar(max)**, **varchar(max)**и **varbinary(max)** . Как триггеры AFTER, так и триггеры INSTEAD OF поддерживают данные типов **varchar(MAX)**, **nvarchar(MAX)** и **varbinary(MAX)** в таблицах inserted и deleted.

Для триггеров в таблицах, оптимизированных для памяти, единственной инструкцией *sql_statement*, разрешенной на верхнем уровне, является блок ATOMIC. В блоке ATOMIC допускается только T-SQL, разрешенный в процедурах, компилируемых в собственном коде.

< method_specifier > **Применимо к**: с SQL Server 2008 по SQL Server 2017.

Указывает метод сборки для связывания с CLR-триггером. Этот метод не должен принимать аргументы и возвращать значения void. Аргумент *class_name* должен быть допустимым идентификатором SQL Server и существовать как класс в сборке с видимостью сборки. Если класс имеет имя, содержащее точки (.) для разделения частей пространства имен, имя класса должно быть заключено в квадратные скобки ([ ]) или двойные кавычки (" "). Класс не может быть вложенным.
Примечание

По умолчанию возможность SQL Server запускать код CLR отключена. Можно создавать, изменять и удалять объекты базы данных, которые ссылаются на модули управляемого кода, но эти ссылаемые модули не будут выполнены на экземплярах SQL Server, пока параметр **clr enabled** не будет включен с помощью процедуры **sp_configure**.

Примечания о триггерах DML

Триггеры DML часто используются для применения бизнес-правил и обеспечения целостности данных. В SQL Server декларативное ограничение ссылочной целостности обеспечивается инструкциями ALTER TABLE и CREATE TABLE. Однако декларативное ограничение ссылочной целостности не обеспечивает ссылочную целостность между базами данных. Ограничение ссылочной целостности подразумевает выполнение правил связи между первичными и внешними ключами таблиц. Для обеспечения ограничений ссылочной целостности используйте в инструкциях ALTER TABLE и CREATE TABLE ограничения PRIMARY KEY и FOREIGN KEY. Если ограничения распространяются на таблицу триггера, они проверяются после срабатывания триггера INSTEAD OF и до выполнения триггера

AFTER. В случае нарушения ограничения выполняется откат действий триггера INSTEAD OF, и триггер AFTER не срабатывает.

Первые и последние триггеры AFTER, которые будут выполнены в таблице, могут быть определены с использованием процедуры sp_settriggerorder. Для таблицы можно определить только один первый и один последний триггер для каждой из операций INSERT, UPDATE и DELETE. Если в таблице есть другие триггеры AFTER, они будут выполняться случайным образом.

Если инструкция ALTER TRIGGER меняет первый или последний триггер, первый или последний набор атрибутов измененного триггера удаляется, а порядок сортировки должен быть установлен заново с помощью процедуры sp_settriggerorder.

Триггер AFTER выполняется только после того, как вызывающая срабатывание триггера инструкция SQL была успешно выполнена. Успешное выполнение также подразумевает завершение всех ссылочных каскадных действий и проверки ограничений, связанных с измененными или удаленными объектами. Триггер AFTER не вызывает рекурсивное срабатывание триггера INSTEAD OF в одной и той же таблице.

Если триггер INSTEAD OF, определенный для таблицы, выполняет по отношению к таблице какую-либо инструкцию, которая бы снова вызвала срабатывание триггера INSTEAD OF, триггер рекурсивно не вызывается. Вместо этого инструкция обрабатывается так, как если бы у таблицы отсутствовал триггер INSTEAD OF, и начинается применение последовательности ограничений и выполнение триггера AFTER. Например, если триггер определен в виде триггера INSTEAD OF INSERT для таблицы и выполняет инструкцию INSERT для этой же таблицы, инструкция INSERT не вызывает нового срабатывания триггера. Команда INSERT, выполняемая триггером, начинает процесс применения ограничений и взвода всех триггеров AFTER INSERT, определенных для данной таблицы.

Если триггер INSTEAD OF, определенный для представления, выполняет по отношению к представлению какую-либо инструкцию, которая бы снова вызвала срабатывание триггера INSTEAD OF, триггер рекурсивно не вызывается. Вместо этого инструкция выполняет изменение базовых таблиц, на которых основано представление. В данном случае определение представления должно удовлетворять всем ограничениям, установленным для обновляемых представлений. Определение обновляемых представлений см. в разделе Изменение данных через представление.

Например, если триггер определен как INSTEAD OF UPDATE для представления и выполняет инструкцию UPDATE для этого же представления, инструкция UPDATE, выполняемая триггером, не вызывает нового срабатывания триггера. Инструкция UPDATE, выполняемая в триггере, обрабатывает представление так, как если бы у представления не имелось триггера INSTEAD OF. Столбцы, измененные с помощью

инструкции UPDATE, должны принадлежать одной базовой таблице. Каждая модификация базовой таблицы вызывает применение последовательности ограничений и взвод триггеров AFTER, определенных для данной таблицы.

Проверка действий инструкций UPDATE или INSERT на указанные столбцы

Триггер языка Transact-SQL можно сконструировать для выполнения конкретных действий, основанных на изменении определенных столбцов с помощью инструкций UPDATE или INSERT. Используйте для этих целей в теле триггера конструкции UPDATE() или COLUMNS_UPDATED. Конструкция UPDATE() проверяет действие инструкций UPDATE или INSERT на одном столбце. С помощью конструкции COLUMNS_UPDATED проверяются действия инструкций UPDATE или INSERT, проводимых на нескольких столбцах, и возвращается битовый шаблон, показывающий, какие столбцы были вставлены или обновлены.

Ограничения триггеров

Инструкция CREATE TRIGGER должна быть первой инструкцией в пакете и может применяться только к одной таблице.

Триггер создается только в текущей базе данных, но может, тем не менее, содержать ссылки на объекты за пределами текущей базы данных.

Если для уточнения триггера указано имя схемы, имя таблицы необходимо уточнить таким же образом.

Одно и то же действие триггера может быть определено более чем для одного действия пользователя (например, INSERT и UPDATE) в одной и той же инструкции CREATE TRIGGER.

Триггеры INSTEAD OF DELETE/UPDATE нельзя определить для таблицы, у которой есть внешний ключ, определенный для каскадного выполнения операции DELETE/UPDATE.

Внутри триггера может быть использована любая инструкция SET. Выбранный параметр SET остается в силе во время выполнения триггера, после чего настройки возвращаются в предыдущее состояние.

Во время срабатывания триггера результаты возвращаются вызывающему приложению так же, как и в случае с хранимыми процедурами. Чтобы предотвратить вызванное срабатыванием триггера возвращение результатов приложению, не следует включать инструкции SELECT, возвращающие результат, или инструкции, которые выполняют в триггере присвоение переменных. Триггер, содержащий либо инструкции SELECT, которые возвращают результаты пользователю, либо инструкции, выполняющие присвоение переменных, требует особого обращения; эти возвращаемые результаты должны быть перезаписаны во все приложения, в которых разрешены изменения таблицы триггера. Если в

триггере происходит присвоение переменной, следует использовать инструкцию SET NOCOUNT в начале триггера, чтобы предотвратить возвращение каких-либо результирующих наборов.

Хотя инструкция TRUNCATE TABLE по своей сути является инструкцией DELETE, она не активирует триггер, поскольку операция не записывает удаление отдельных строк. Однако беспокоиться о случайном обходе триггера DELETE таким образом нужно только пользователям с разрешениями на выполнение инструкции TRUNCATE TABLE.

Инструкция WRITETEXT (с ведением журнала и без него) не запускает триггеры.

Следующие инструкции языка Transact-SQL не разрешены в триггерах DML:

ALTER DATABASE	CREATE DATABASE	DROP DATABASE
RESTORE DATABASE	RESTORE LOG	RECONFIGURE

Кроме того, использование следующих инструкций Transact-SQL в тексте триггера DML не допускается, если он применяется к таблице или представлению, которые являются целью действий триггера.

CREATE INDEX (в т.ч CREATE SPATIAL INDEX и CREATE XML INDEX)	ALTER INDEX	DROP INDEX
DBCC DBREINDEX	ALTER PARTITION FUNCTION	DROP TABLE
ALTER TABLE, если используется в следующих целях:		
Добавление, изменение или удаление столбцов.		
Переключение секций.		
Добавление или удаление ограничений PRIMARY KEY и UNIQUE.		

Примечание

Поскольку SQL Server не поддерживает пользовательских триггеров в системных таблицах, рекомендуется не создавать пользовательские триггеры для системных таблиц.

Оптимизация триггеров DML

Триггеры работают в транзакциях (подразумеваемым или иным способом) и, несмотря на открытость, блокируют ресурсы. Блокировка действует до тех пор, пока транзакция не будет зафиксирована (COMMIT) или отклонена (ROLLBACK). Чем дольше выполняется триггер, тем выше вероятность блокирования другого процесса. Таким образом, при написании триггеров необходимо стремиться свести к минимуму продолжительность их выполнения. Одним из способов добиться этого является освобождение триггера в том случае, если инструкция DML изменяет 0 строк.

Чтобы освободить триггер для команды, которая не изменяет ни одной строки, используйте системную переменную ROWCOUNT_BIG.

Это реализуется в следующем фрагменте кода T-SQL, который должен присутствовать в начале каждого триггера DML:
SQLКопировать

```
IF (ROWCOUNT_BIG() = 0)
RETURN;
```

Примечания о триггерах DDL

Триггеры DDL, как и стандартные триггеры, выполняют хранимые процедуры в ответ на какое-либо событие. В отличие от стандартных триггеров, они не срабатывают в ответ на выполнение инструкций UPDATE, INSERT или DELETE по отношению к таблице или представлению. Вместо этого триггеры срабатывают в первую очередь в ответ на инструкции языка определения данных (DDL). Это инструкции CREATE, ALTER, DROP, GRANT, DENY, REVOKE и UPDATE STATISTICS. Системные хранимые процедуры, выполняющие операции, подобные операциям DDL, также могут запускать триггеры DDL.

Важно!

Протестируйте триггеры DDL, чтобы получить ответ на выполнение системных хранимых процедур. Например, инструкция CREATE TYPE и хранимые процедуры sp_addtype и sp_rename вызовут срабатывание триггера DDL, созданного для события CREATE_TYPE.

Дополнительные сведения о триггерах DDL см. в разделе Триггеры DDL.

Триггеры DDL не срабатывают в ответ на события, влияющие на локальные или глобальные временные таблицы и хранимые процедуры.

В отличие от триггеров DML, триггеры DDL не ограничены областью схемы. Поэтому для запроса метаданных о триггерах DDL нельзя воспользоваться такими функциями как OBJECT_ID, OBJECT_NAME, OBJECTPROPERTY и OBJECTPROPERTYEX. Используйте вместо них представления каталога. Дополнительные сведения см. в статье Получение сведений о триггерах DDL.

Примечание

Триггеры DDL сервера находятся в папке **Триггеры** обозревателя объектов среды Среда SQL Server Management Studio. Эта папка находится под папкой **Объекты сервера** .Триггеры DDL, доступные в области базы данных, находятся в папке **Триггеры базы данных**. Эта папка находится в папке **Программирование** соответствующей базы данных.

Триггеры входа

Триггеры входа выполняют хранимые процедуры в ответ на событие LOGON. Это событие вызывается при установке пользовательского сеанса с экземпляром SQL Server. Триггеры входа срабатывают после завершения этапа проверки подлинности при входе, но перед тем, как пользовательский сеанс реально устанавливается. Следовательно, все сообщения, которые возникают внутри триггера и обычно достигают пользователя, такие как сообщения об ошибках и сообщения от инструкции PRINT, перенаправляются в журнал ошибок SQL Server .Дополнительные сведения см. в разделе Триггеры входа.

Если проверка подлинности завершается сбоем, триггеры входа не срабатывают.

В триггерах входа не поддерживаются распределенные транзакции. Если срабатывает триггер входа, содержащий распределенную транзакцию, возвращается ошибка 3969.

Отключение триггера входа

Триггер входа может эффективно запрещать подключения к службам Компонент Database Engine для всех пользователей, в том числе членов предопределенной роли сервера **sysadmin**. Если триггер входа запрещает соединения, члены предопределенной роли сервера **sysadmin**могут подключаться с помощью выделенного административного соединения или путем вызова Компонент Database Engine в режиме минимальной конфигурации (-f).Дополнительные сведения см. в разделе Параметры запуска службы Database Engine.

Общие соглашения о триггерах

Возвращаемые результаты

Возможность возвращать результаты из триггеров будет исключена из следующей версии SQL Server. Триггеры, возвращающие результирующие наборы, могут привести к непредвиденному поведению приложений, не предназначенных для работы с ними. Не используйте в разрабатываемых приложениях триггеры, возвращающие результирующие наборы, и запланируйте изменение приложений, которые используют их в настоящее время.Чтобы триггеры не возвращали результирующие наборы, для параметра disallow results from triggers необходимо установить значение 1.

Триггеры входа всегда запрещают возврат результирующих наборов, и данное поведение нельзя настроить. Если триггер входа формирует результирующий

набор, то триггер не выполняется и попытка входа, вызванная триггером, запрещается.

Несколько триггеров

SQL Server позволяет создавать несколько триггеров для каждого события DML, DDL и LOGON.Например, если инструкция CREATE TRIGGER FOR UPDATE выполняется в таблице, уже имеющей триггер UPDATE, дополнительно создается триггер обновления. В более ранних версиях SQL Server был разрешен только один триггер в каждой таблице для каждого события изменения данных INSERT, UPDATE или DELETE.

Рекурсивные триггеры

SQL Server разрешает рекурсивный вызов триггеров, если с помощью инструкции ALTER DATABASE включена настройка RECURSIVE_TRIGGERS.

В рекурсивных триггерах могут возникать следующие типы рекурсии:

- Косвенная рекурсия

  При косвенной рекурсии приложение обновляет таблицу T1. Это событие вызывает срабатывание триггера TR1, обновляющего таблицу T2. Это вызывает срабатывание триггера T2 и обновление таблицы T1.

- Прямая рекурсия

  При прямой рекурсии приложение обновляет таблицу T1. Это событие вызывает срабатывание триггера TR1, обновляющего таблицу T1. Поскольку таблица T1 уже была обновлена, триггер TR1 срабатывает снова и т. д.

  В следующем примере используются оба типа рекурсий: прямая и косвенная. Допустим, для таблицы T1 определены два триггера: TR1 и TR2. Триггер TR1 рекурсивно обновляет таблицу T1. Инструкция UPDATE выполняет каждый из триггеров TR1 и TR2 один раз. В дополнение к этому срабатывание триггера TR1 вызывает выполнение триггеров TR1 (рекурсивно) и TR2. В таблицах inserted и deleted триггера содержатся строки, которые относятся только к инструкции UPDATE, вызвавшей срабатывание триггера.

Примечание

Описанная ситуация имеет место только в том случае, если настройка RECURSIVE_TRIGGERS включена с помощью инструкции ALTER DATABASE. Определенного порядка выполнения нескольких триггеров, заданных для какого-либо конкретного события, не существует. Каждый триггер должен быть самодостаточным.

Отключение настройки RECURSIVE_TRIGGERS предотвращает выполнение только прямых рекурсий. Чтобы отключить косвенную рекурсию, с помощью хранимой процедуры sp_configure присвойте параметру сервера nested triggers значение 0.

Если один из триггеров выполняет инструкцию ROLLBACK TRANSACTION, никакие другие триггеры, вне зависимости от уровня вложенности, не срабатывают.

Вложенные триггеры

Вложенность триггеров может достигать максимум 32 уровня. Если триггер изменяет таблицу, для которой определен другой триггер, то запускается второй триггер, вызывающий срабатывание третьего и т.д. Если любой из триггеров в цепочке отключает бесконечный цикл, то уровень вложенности превышает допустимый предел, и срабатывание триггера отменяется. Если триггер на языке Transact-SQL выполняет управляемый код с помощью ссылки на метод, тип или статистическую функцию среды CLR, эта ссылка считается одним из допустимых 32 уровней вложенности. Методы, вызываемые из управляемого кода, под это ограничение не подпадают.

Чтобы отменить вложенные триггеры, присвойте значение 0 параметру nested triggers хранимой процедуры sp_configure. В конфигурации по умолчанию вложенные триггеры разрешены. Если вложенные триггеры отключены, рекурсивные триггеры тоже будут отключены, вне зависимости от настройки RECURSIVE_TRIGGERS, установленной с помощью инструкции ALTER DATABASE.

Первый триггер AFTER, вложенный в триггер INSTEAD OF, срабатывает, даже если параметру конфигурации сервера **nested triggers** присвоено значение 0. Однако при таком значении параметра последующие триггеры AFTER не срабатывают. Рекомендуется проверить приложения на наличие вложенных триггеров, чтобы определить, соответствуют ли приложения бизнес-правилам в случае, если параметру конфигурации сервера **nested triggers**присвоено значение 0, и выполнить соответствующие изменения.

Отложенная интерпретация имен

В SQL Server разрешены хранимые процедуры, триггеры и пакеты на языке Transact-SQL, которые содержат ссылки на таблицы, не существующие в момент компиляции. Такая возможность называется отложенной интерпретацией имен.

Разрешения

Для создания триггера DML требуется разрешение ALTER на таблицу или представление, в которых создается триггер.

Для создания триггера DDL с областью действия в пределах сервера (ON ALL SERVER) или триггера входа требуется разрешение CONTROL SERVER на сервер. Для создания триггера DDL с областью видимости в пределах базы данных (ON

DATABASE) требуется разрешение ALTER ANY DATABASE DDL TRIGGER на текущую базу данных.

Примеры

А. Использование триггера DML с предупреждающим сообщением

Следующий триггер DML отправляет клиенту сообщение, когда кто-то пытается добавить или изменить данные в таблице `Customer` в базе данных AdventureWorks2012.

SQLКопировать

```
CREATE TRIGGER reminder1
ON Sales.Customer
AFTER INSERT, UPDATE
AS RAISERROR ('Notify Customer Relations', 16, 10);
GO
```

Б. Использование триггера DML с предупреждающим сообщением, отправляемым по электронной почте

В следующем примере указанному пользователю (`MaryM`) по электронной почте отправляется сообщение при изменении таблицы `Customer`.

SQLКопировать

```
CREATE TRIGGER reminder2
ON Sales.Customer
AFTER INSERT, UPDATE, DELETE
AS
   EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'AdventureWorks2012 Administrator',
        @recipients = 'danw@Adventure-Works.com',
        @body = 'Don''t forget to print a report for the sales force.',
        @subject = 'Reminder';
GO
```

В. Использование триггера DML AFTER для принудительного применения бизнес-правил между таблицами PurchaseOrderHeader и Vendor

Поскольку ограничение CHECK может содержать ссылки только на столбцы, для которых определены ограничения на уровне столбцов или таблицы, любые межтабличные ограничения (в данном случае бизнес-правила) должны быть заданы в виде триггеров.

В следующем примере создается триггер DML в базе данных AdventureWorks 2012. Этот триггер проверяет оценку кредитоспособности поставщика (не 5) при попытке добавить новый заказ на покупку в таблицу `PurchaseOrderHeader`. Для получения сведений о кредитоспособности поставщика требуется ссылка на таблицу `Vendor`. В случае слишком низкой кредитоспособности выводится соответствующее сообщение и вставка не выполняется.

SQLКопировать

```
-- This trigger prevents a row from being inserted in the
Purchasing.PurchaseOrderHeader
-- table when the credit rating of the specified vendor is set to 5 (below
average).

CREATE TRIGGER Purchasing.LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT
AS
IF (@@ROWCOUNT_BIG  = 0)
RETURN;
IF EXISTS (SELECT *
           FROM Purchasing.PurchaseOrderHeader AS p
           JOIN inserted AS i
           ON p.PurchaseOrderID = i.PurchaseOrderID
           JOIN Purchasing.Vendor AS v
           ON v.BusinessEntityID = p.VendorID
           WHERE v.CreditRating = 5
          )
BEGIN
RAISERROR ('A vendor''s credit rating is too low to accept new
purchase orders.', 16, 1);
ROLLBACK TRANSACTION;
RETURN
END;
GO

-- This statement attempts to insert a row into the PurchaseOrderHeader table
-- for a vendor that has a below average credit rating.
-- The AFTER INSERT trigger is fired and the INSERT transaction is rolled
back.

INSERT INTO Purchasing.PurchaseOrderHeader (RevisionNumber, Status,
EmployeeID,
VendorID, ShipMethodID, OrderDate, ShipDate, SubTotal, TaxAmt, Freight)
VALUES (
2
,3
,261
,1652
,4
,GETDATE()
,GETDATE()
,44594.55
,3567.564
,1114.8638 );
GO
```

Г. Использование триггера DDL уровня базы данных

В следующем примере триггер DDL используется для предотвращения удаления синонимов в базе данных.

SQLКопировать

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_SYNONYM
AS
IF (@@ROWCOUNT = 0)
RETURN;
   RAISERROR ('You must disable Trigger "safety" to drop synonyms!',10, 1)
   ROLLBACK
GO
DROP TRIGGER safety
ON DATABASE;
GO
```

Д. Использование триггера DDL уровня сервера

В следующем примере триггер DDL используется для вывода сообщения при возникновении на данном экземпляре сервера любого из событий CREATE DATABASE, а функция `EVENTDATA`используется для получения текста соответствующей инструкции на языке Transact-SQL.Примеры использования функции EVENTDATA в триггерах DDL см. в разделе Использование функции EVENTDATA.

Применимо к: с SQL Server 2008 до SQL Server 2017.

SQLКопировать

```
CREATE TRIGGER ddl_trig_database
ON ALL SERVER
FOR CREATE_DATABASE
AS
    PRINT 'Database Created.'
    SELECT
EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]','nvarchar(ma
x)')
GO
DROP TRIGGER ddl_trig_database
ON ALL SERVER;
GO
```

Е. Использование триггера входа

В следующем примере триггера входа выполняется запрет попытки подключения к SQL Serverв качестве члена имени входа *login_test*, если для этого имени входа уже запущено три сеанса.

Применимо к: с SQL Server 2008 до SQL Server 2017.

SQLКопировать

```
USE master;
GO
CREATE LOGIN login_test WITH PASSWORD = '3KHJ6dhx(0xVYsdf' MUST_CHANGE,
    CHECK_EXPIRATION = ON;
```

```
GO
GRANT VIEW SERVER STATE TO login_test;
GO
CREATE TRIGGER connection_limit_trigger
ON ALL SERVER WITH EXECUTE AS 'login_test'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN()= 'login_test' AND
    (SELECT COUNT(*) FROM sys.dm_exec_sessions
            WHERE is_user_process = 1 AND
                original_login_name = 'login_test') > 3
    ROLLBACK;
END;
```

Ж. Просмотр событий, вызвавших срабатывание триггера

В следующем примере выполняются запросы к представлениям каталога `sys.triggers` и `sys.trigger_events` с целью определения, какие события языка Transact-SQL вызывали срабатывание триггера `safety`. Триггер `safety`, созданный в примере Г, приведен выше.

SQLКопировать

```
SELECT TE.*
FROM sys.trigger_events AS TE
JOIN sys.triggers AS T ON T.object_id = TE.object_id
WHERE T.parent_class = 0 AND T.name = 'safety';
GO
```