Оновлення даних. Інструкції INSERT, UPDATE, DELETE. Цілісність даних.

# Инструкция INSERT (Transact-SQL)

SQL Server 2008 R2

Добавляет одну или несколько новых строк в таблицу или представление в SQL Server 2008 R2. Примеры см. в разделе Примеры использования инструкции INSERT (Transact-SQL).

Синтаксические обозначения в Transact-SQL

Синтаксис

Transact-SQL

-- Standard INSERT syntax

[ WITH <common\_table\_expression> [ ,...n ] ]

INSERT

{

[ TOP (expression) [ PERCENT ] ]

[ INTO ]

{ <object> | rowset\_function\_limited

[ WITH ( <Table\_Hint\_Limited> [ ...n ] ) ] }

{

[ (column\_list) ]

[ <OUTPUT Clause> ]

{ VALUES ( { DEFAULT | NULL | expression } [ ,...n ] ) [ ,...n ]

| derived\_table

| execute\_statement

| <dml\_table\_source>

| DEFAULT VALUES

}

}

}

[; ]

Transact-SQL

-- External tool only syntax

INSERT

{

[BULK]

[ database\_name . [ schema\_name ] . | schema\_name . ]

[ table\_name | view\_name ]

( <column\_definition> )

[ WITH (

[ [ , ] CHECK\_CONSTRAINTS ]

[ [ , ] FIRE\_TRIGGERS ]

[ [ , ] KEEP\_NULLS ]

[ [ , ] KILOBYTES\_PER\_BATCH = kilobytes\_per\_batch ]

[ [ , ] ROWS\_PER\_BATCH = rows\_per\_batch ]

[ [ , ] ORDER ( { column [ ASC | DESC ] } [ ,...n ] ) ]

[ [ , ] TABLOCK ]

) ]

}

[; ]

Transact-SQL

<object> ::=

{

[ server\_name . database\_name . schema\_name .

| database\_name .[ schema\_name ] .

| schema\_name .

]

table\_or\_view\_name

}

<dml\_table\_source> ::=

SELECT <select\_list>

FROM ( <dml\_statement\_with\_output\_clause> )

[AS] table\_alias [ ( column\_alias [ ,...n ] ) ]

[ WHERE <search\_condition> ]

[ OPTION ( <query\_hint> [ ,...n ] ) ]

<column\_definition> ::=

column\_name <data\_type>

[ COLLATE collation\_name ]

[ NULL | NOT NULL ]

<data type> ::=

[ type\_schema\_name . ] type\_name

[ ( precision [ , scale ] | max ]

Аргументы

WITH <common\_table\_expression>

Определяет временный именованный результирующий набор, также называемый обобщенным табличным выражением, определенным в области инструкции INSERT. Результирующий набор получается из инструкции SELECT.

Обобщенные табличные выражения также используются инструкциями SELECT, DELETE, UPDATE, MERGE и CREATE VIEW. Дополнительные сведения см. в разделе WITH обобщенное\_табличное\_выражение (Transact-SQL).

TOP (expression) [ PERCENT ]

Задает количество или процент случайных строк для вставки. Выражение expression может быть либо количеством, либо процентом строк. Строки, на которые ссылается выражение TOP, используемое с INSERT, UPDATE и DELETE, не упорядочены.

Инструкции INSERT, UPDATE и DELETE требуют заключения аргумента expression в круглые скобки в выражении TOP. Дополнительные сведения см. в разделах TOP (Transact-SQL).

INTO

Необязательное ключевое слово, которое можно использовать между ключевым словом INSERT и целевой таблицей.

server\_name

Имя связанного сервера, на котором расположена таблица или индексированное представление. Аргумент server\_name можно задавать в виде имени связанного сервера или с помощью функции OPENDATASOURCE.

Если аргумент server\_name задается в виде связанного сервера, то необходимы аргументы database\_name и schema\_name. Если аргумент server\_name задается с помощью функции OPENDATASOURCE, то аргументы database\_name и schema\_name могут применяться не ко всем источникам данных, в зависимости от возможностей поставщика OLE DB, который обращается к удаленному объекту. Дополнительные сведения см. в разделах Распределенные запросы.

database\_name

Имя базы данных.

schema\_name

Имя схемы, которой принадлежит таблица или представление.

table\_or view\_name

Имя таблицы или представления, которые принимают данные.

В качестве источника таблицы в инструкции INSERT можно использовать табличную переменную внутри своей области.

Представление, на которое ссылается аргумент table\_or\_view\_name, должно быть обновляемым и ссылаться только на одну базовую таблицу в предложении FROM данного представления. Например, инструкция INSERT в многотабличном представлении должна использовать аргумент column\_list, который ссылается только на столбцы из одной базовой таблицы. Дополнительные сведения об обновляемых представлениях см. в разделе CREATE VIEW (Transact-SQL).

rowset\_function\_limited

Либо функция OPENQUERY, либо функция OPENROWSET. Использование этих функций зависит от возможностей поставщика OLE DB, который обращается к удаленному объекту. Дополнительные сведения см. в разделах Распределенные запросы.

WITH ( <table\_hint\_limited> [... n ] )

Задает одну или несколько табличных подсказок, разрешенных для целевой таблицы. Необходимо использовать ключевое слово WITH и круглые скобки.

Нельзя использовать подсказки READPAST, NOLOCK, и READUNCOMMITTED. Дополнительные сведения о табличных подсказках см. в разделе Табличные подсказки (Transact-SQL). Важно!

Возможность указать подсказки HOLDLOCK, SERIALIZABLE, READCOMMITTED, REPEATABLEREAD или UPDLOCK в целевых таблицах инструкций INSERT будет удалена в будущих версиях SQL Server. Эти подсказки не влияют на производительность инструкций INSERT. Избегайте применять их в новых разработках и запланируйте внесение изменений в приложения, использующие их в настоящее время.

Указание подсказки TABLOCK для целевой таблицы инструкции INSERT приведет к тем же последствиям, что и указание подсказки TABLOCKX. К таблице будет применена монопольная блокировка.

(column\_list)

Список из одного или нескольких столбцов, в которые нужно вставить данные. Аргумент column\_list должен быть заключен в круглые скобки и разделен запятыми.

Если столбец не внесен в column\_list, то компонент Database Engine должен обеспечить значение, основанное на определении столбца; в противном случае строку нельзя будет загрузить. Компонент Database Engine автоматически задает значение для столбца, если столбец имеет следующие характеристики.

Имеется свойство IDENTITY. Используется следующее значение приращения для идентификатора.

Имеется стандартное значение. Используется стандартное значение для столбца.

Имеется тип данных timestamp. В этом случае используется текущее значение отметки времени.

Допускаются значения NULL. Используется значение NULL.

Вычисляемый столбец. Используется вычисленное значение.

Аргумент column\_list и список значений необходимо использовать, когда в столбец идентификаторов вставляются явно заданные значения, а параметру SET IDENTITY\_INSERT необходимо присвоить значение ON для таблицы.

Предложение OUTPUT

Возвращает вставленные строки во время операции вставки. Результаты могут возвращаться в обрабатывающее приложение или вставляться в таблицу или табличную переменную для дальнейшей обработки.

Предложение OUTPUT не поддерживается инструкциями DML, которые ссылаются на локальные секционированные представления, распределенные секционированные представления, расположенные удаленно таблицы или инструкции INSERT, содержащие аргумент execute\_statement. Предложение OUTPUT INTO не поддерживается в инструкциях INSERT, содержащих предложение <dml\_table\_source>.

VALUES

Позволяет использовать один или несколько списков вставляемых значений данных. Для каждого столбца в column\_list, если этот параметр указан или присутствует в таблице, должно быть одно значение. Список значений должен быть заключен в скобки.

Если значения в списке идут в порядке, отличном от порядка следования столбцов в таблице, или не для каждого столбца таблицы определено значение, то необходимо использовать аргумент column\_list для явного указания столбца, в котором хранится каждое входное значение.

Можно использовать конструктор строк Transact-SQL (также называемый конструктором табличных значений), позволяющий указать несколько строк в одной инструкции INSERT. Этот конструктор строк состоит из одного предложения VALUES со списками из нескольких значений, заключенными в круглые скобки и разделенными запятыми. Дополнительные сведения см. в разделе Конструктор табличных значений (Transact-SQL).

DEFAULT

Указывает компоненту Database Engine необходимость принудительно загружать значения по умолчанию, определенные для столбца. Если для столбца не задано значение по умолчанию и он может содержать значение NULL, вставляется значение NULL. В столбцы с типом данных timestamp вставляется следующее значение отметки времени. Значение DEFAULT недопустимо для столбца идентификаторов.

expression

Константа, переменная или выражение. Выражение не может содержать инструкцию EXECUTE.

При ссылке на символьные типы данных nchar, nvarchar и ntext в Юникоде «выражение» должно начинаться с заглавной буквы «N». Если префикс «N» не указан, SQL Server выполнит преобразование строки в кодовую страницу, соответствующую параметрам сортировки базы данных или столбца, действующим по умолчанию. Любые символы, не входящие в эту кодовую страницу, будут утрачены. Дополнительные сведения см. в разделе Программирование на стороне сервера с использованием Юникода.

derived\_table

Любая допустимая инструкция SELECT, возвращающая строки данных, которые загружаются в таблицу. Инструкция SELECT не может содержать обобщенное табличное выражение (CTE).

execute\_statement

Любая допустимая инструкция EXECUTE, возвращающая данные с помощью инструкций SELECT или READTEXT.

Если аргумент execute\_statement используется с инструкцией INSERT, каждый результирующий набор должен быть совместим со столбцами в таблице или списке column\_list.

Аргумент execute\_statement может применяться для выполнения хранимых процедур на том же сервере или на сервере, расположенном удаленно. На удаленном сервере выполняется процедура, результирующий набор возвращается на локальный сервер и загружается в таблицу на локальном сервере. В распределенной транзакции нельзя выполнить инструкцию execute\_statement для связанного сервера с замыканием на себя, если при соединении включен режим MARS (множественный активные результирующий набор).

Если аргумент execute\_statement возвращает данные с инструкцией READTEXT, необходимо учитывать, что каждая инструкция READTEXT может возвращать не более 1 МБ (1024 КБ) данных. Аргумент execute\_statement также может использоваться с расширенными процедурами. В этом случае он вставляет данные, возвращенные основным потоком расширенной процедуры, но выходные данные, возвращенные потоками, отличными от основного, не будут вставлены.

Возвращающий табличное значение параметр нельзя указывать в качестве объекта инструкции INSERT EXEC, но его можно указать в виде источника в строке INSERT EXEC или в хранимой процедуре. Дополнительные сведения см. в разделе Возвращающие табличное значение параметры (компонент Database Engine).

<dml\_table\_source>

Указывает, что вставленные в целевую таблицу строки были возвращены предложением OUTPUT инструкции INSERT, UPDATE, DELETE или MERGE с возможной фильтрацией предложением WHERE. Если используется аргумент <dml\_table\_source>, целевая таблица внешней инструкции INSERT должна удовлетворять следующим ограничениям:

Быть базовой таблицей, а не представлением.

Не быть удаленной таблицей.

Не иметь определенных для нее триггеров.

Не участвовать в связях «первичный-внешний ключ».

Объект не должен участвовать в репликации слиянием или обновляемых подписках для репликации транзакций.

Уровень совместимости базы данных должен быть не ниже 100. Дополнительные сведения см. в разделе Предложение OUTPUT (Transact-SQL).

<select\_list>

Список с разделителями-запятыми, указывающий, какие столбцы возвращены предложением OUTPUT для вставки. Столбцы в <select\_list> должны быть совместимы со столбцами, в которые вставляются значения. <select\_list> не может ссылаться на агрегатные функции или TEXTPTR. Примечание

Любые перечисленные в списке SELECT переменные ссылаются на свои исходные значения, независимо от любых изменений, произошедших с ними в <dml\_statement\_with\_output\_clause>.

<dml\_statement\_with\_output\_clause>

Допустимая инструкция INSERT, UPDATE, DELETE или MERGE, возвращающая изменяемые строки в предложении OUTPUT. Инструкция не может содержать предложение WITH и использовать удаленные таблицы или секционированные представления в качестве целевых. Если указаны UPDATE или DELETE, это не могут быть использующие курсор инструкции UPDATE или DELETE. На исходные строки нельзя ссылаться как на вложенные инструкции DML.

WHERE <search\_condition>

Любое предложение WHERE, содержащее допустимый критерий поиска <search\_condition>, фильтрующее строки, которые возвращены аргументом <dml\_statement\_with\_output\_clause>. Дополнительные сведения см. в разделах Условие поиска (Transact-SQL). При использовании в этом контексте критерий <search\_condition> не должен содержать вложенных запросов, определяемых пользователем скалярных функций, выполняющих доступ к данным, агрегатных функций, TEXTPTR или полнотекстовых предикатов поиска.

DEFAULT VALUES

Заполняет новую строку значениями по умолчанию, определенными для каждого столбца.

BULK

Используется внешними средствами для передачи потока двоичных данных. Этот параметр не предназначен для использования с такими средствами, как среда Среда SQL Server Management Studio, SQLCMD, OSQL или прикладными программными интерфейсами для доступа к данным, такими как собственный клиент SQL Server.

FIRE\_TRIGGERS

Указывает, что при передаче потока двоичных данных будут выполняться триггеры INSERT, определенные для целевой таблицы. Дополнительные сведения см. в разделах BULK INSERT (Transact-SQL).

CHECK\_CONSTRAINTS

Указывает, что при передаче потока двоичных данных будет выполняться проверка всех ограничений целевой таблицы или представления. Дополнительные сведения см. в разделах BULK INSERT (Transact-SQL).

KEEPNULLS

Указывает, что пустые столбцы во время передачи потока двоичных данных должны сохранить значение NULL. Дополнительные сведения см. в разделах Сохранение значений NULL или использование значений по умолчанию при массовом импорте данных.

KILOBYTES\_PER\_BATCH = kilobytes\_per\_batch

Определяет приблизительное число килобайт данных в пакете как kilobytes\_per\_batch. Дополнительные сведения см. в разделах BULK INSERT (Transact-SQL).

ROWS\_PER\_BATCH =rows\_per\_batch

Указывает приблизительное число строк в потоке двоичных данных. Дополнительные сведения см. в разделах BULK INSERT (Transact-SQL).

Примечание. Если список столбцов отсутствует, то возникает синтаксическая ошибка.

Рекомендации

Для возврата количества строк, которое было вставлено из клиента, используйте функцию @@ROWCOUNT. Дополнительные сведения см. в разделе @@ROWCOUNT (Transact-SQL).

Рекомендации по массовому импорту данных

Использование инструкции INSERT INTO…SELECT для массового импорта данных с минимальным протоколированием

Инструкция INSERT INTO <целевая\_таблица> SELECT <столбцы> FROM <исходная\_таблица> может эффективно перенести большое количество строк из одной таблицы, например промежуточной, в другую таблицу с минимальным протоколированием. Минимальное протоколирование может повысить производительность выполнения инструкции и снизить вероятность того, что во время операции будет заполнен весь журнал транзакций.

Для минимального протоколирования этой инструкции необходимо выполнение следующих требований.

Модель восстановления базы данных настроена на простое или неполное протоколирование.

Целевой таблицей является пустая или непустая куча.

Целевая таблица не используется в репликации.

Для целевой таблицы указана подсказка TABLOCK.

Для строк, которые вставляются в кучу в результате действия вставки в инструкции MERGE, также может применяться минимальное протоколирование.

В отличие от инструкции BULK INSERT, которая удерживает менее строгую блокировку массового обновления, инструкция INSERT INTO…SELECT с подсказкой TABLOCK удерживает монопольную блокировку (X) таблицы. Это означает, что не поддерживается возможность вставки строк с помощью параллельных операций вставки. Дополнительные сведения о блокировках см. в разделе Режимы блокировки.

Использование предложений OPENROWSET и BULK для массового импорта данных

Функция OPENROWSET может принимать следующие табличные подсказки, обеспечивающие оптимизацию массовой загрузки с инструкцией INSERT.

Использование подсказки TABLOCK может свести к минимуму число записей в журнале для операции вставки. Для базы данных должна быть установлена простая модель восстановления или модель восстановления с неполным протоколированием. Кроме того, целевая таблица не может использоваться в репликации. Дополнительные сведения см. в разделах Предварительные условия для минимального ведения журнала массового импорта данных.

Проверку ограничений FOREIGN KEY и CHECK можно временно отключить с помощью подсказки IGNORE\_CONSTRAINTS.

Выполнение триггеров можно временно отключить с помощью подсказки IGNORE\_TRIGGERS.

Подсказка KEEPDEFAULTS обеспечивает возможность вставки установленного по умолчанию значения столбца таблицы, если таковое имеется, вместо значения NULL, применяемого в том случае, если запись данных не содержит значения для этого столбца.

Подсказка KEEPIDENTITY позволяет использовать значения идентификаторов в файле импортированных данных для столбца идентификаторов в целевой таблице.

Эти оптимизации похожи на оптимизации, доступные для команды BULK INSERT. Дополнительные сведения см. в разделах Табличные подсказки (Transact-SQL).

Типы данных

При вставке строк необходимо учитывать поведение следующих типов данных:

Если значение загружается в столбцы с типом данных char, varchar или varbinary, то дополнение или усечение конечных пробелов (пробелы для char и varchar, нули для varbinary) определяет параметр SET ANSI\_PADDING, определенный для столбца при создании таблицы. Дополнительные сведения см. в разделах SET ANSI\_PADDING (Transact-SQL).

В следующей таблице показаны операции по умолчанию для параметра SET ANSI\_PADDING, установленного в значение OFF.

Тип данных

Стандартная операция

char

Заполнение значения пробелами до заданной ширины столбца.

varchar

Удаление конечных пробелов до последнего непробельного символа или до одного пробела, если строка состоит только из пробелов.

varbinary

Удаление конечных нулей.

Если пустая строка ('') загружена в столбец с типом данных varchar или text, то операцией по умолчанию будет загрузка строки нулевой длины.

Вставка значения NULL в столбец text или image не приводит ни к созданию допустимого текстового указателя, ни к предварительному распределению 8-килобайтной текстовой страницы. Дополнительные сведения о вставке данных text и image см. в разделе Применение функций для работы с типами данных text, ntext и image.

Столбцы, созданные с типом данных uniqueidentifier, содержат двоичные 16-байтные величины специального формата. В отличие от столбцов идентификаторов компонента Database Engine не создает автоматически значения для столбцов с типом данных uniqueidentifier. Во время операции вставки переменные с типом данных uniqueidentifier и строковые константы вида xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (36 символов, включая дефисы, где x — шестнадцатеричная цифра в диапазоне от 0-9 или a-f) можно использовать для столбцов uniqueidentifier. Например, 6F9619FF-8B86-D011-B42D-00C04FC964FF является допустимым значением переменной uniqueidentifier или столбца. Используйте функцию NEWID() для получения идентификатора GUID.

Вставка значений в столбцы определяемого пользователем типа

Вставлять значения в столбцы определяемого пользователем типа можно следующими способами.

Предоставление значения определяемого пользователем типа.

Предоставление значения типом системных данных SQL Server, если определяемый пользователем тип поддерживает явное или неявное преобразование из этого типа. В следующем примере показано, как вставляются значения из столбца определяемого пользователем типа Point путем явного преобразования из строки.

other

INSERT INTO Cities (Location)

VALUES ( CONVERT(Point, '12.3:46.2') );

Двоичное значение также может предоставляться без выполнения явного преобразования, так как все определяемые пользователем типы могут быть неявно преобразованы из двоичного. Дополнительные сведения о преобразовании и определяемых пользователем типах данных см. в разделе Выполнение операций над определяемыми пользователем типами данных.

Вызов определяемой пользователем функции, которая возвращает значение определяемого пользователем типа. В следующих примерах используется определяемая пользователем функция CreateNewPoint() для создания новых значений определяемого пользователем типа Point и вставки значения в таблицу Cities.

other

INSERT INTO Cities (Location)

VALUES ( dbo.CreateNewPoint(x, y) );

Обработка ошибок

Для инструкции INSERT можно реализовать обработку ошибок, указав инструкцию в конструкции TRY…CATCH. Дополнительные сведения см. в разделе Использование конструкции TRY...CATCH в языке Transact-SQL.

Если инструкция INSERT нарушает ограничение или правило, либо в ней присутствует значение, несовместимое с типом данных столбца, то при выполнении инструкции происходит сбой и отображается сообщение об ошибке.

Если инструкция INSERT загружает несколько строк с помощью инструкции SELECT или EXECUTE, то любые нарушения правил или ограничений, возникающие из-за загружаемых значений, приводят к остановке выполнения инструкции, и ни одна из строк не будет загружена.Если при выполнении инструкции INSERT возникает арифметическая ошибка (переполнение, деление на ноль или ошибка домена), компонент Database Engine обрабатывает эти ошибки так же, как если бы параметру SET ARITHABORT было присвоено значение ON. Выполнение пакета прекращается и выводится сообщение об ошибке. Во время оценки выражения, когда параметры SET ARITHABORT и SET ANSI\_WARNINGS установлены в значение OFF, если в инструкции INSERT, DELETE или UPDATE происходит арифметическая ошибка переполнения, деления на ноль или ошибка области определения, SQL Server вставляет или обновляет значение NULL. Если целевой столбец не пустой, вставка или обновление не осуществляются, и пользователь получает ошибку. Дополнительные сведения см. в разделах Режим работы при значении ON параметров ARITHABORT и ARITHIGNORE.

Совместимость

Если триггер INSTEAD OF определен в операциях INSERT для таблицы или представления, то триггер выполняется вместо инструкции INSERT. Дополнительные сведения о триггерах INSTEAD OF см. в разделе CREATE TRIGGER (Transact-SQL).

Ограничения

Если во время вставки значений в удаленные таблицы указаны не все значения для всех столбцов, то необходимо указать столбцы, в которые вставляются заданные значения.

Инструкции INSERT не учитывает настройки параметра SET ROWCOUNT в местных и расположенных удаленно секционированных представлениях. Кроме того, этот параметр не поддерживается для инструкций INSERT, применяемых к удаленным таблицам. Важно!

Использование инструкции SET ROWCOUNT не будет оказывать влияния на инструкции DELETE, INSERT и UPDATE в следующей версии SQL Server. В новых разработках следует избегать совместного использования инструкции SET ROWCOUNT с инструкциями DELETE, INSERT и UPDATE. Также необходимо запланировать внесение изменений в используемые приложения. Вместо нее рекомендуется использовать предложение TOP.

Режим блокировки

Инструкция INSERT всегда получает монопольную блокировку (X) на таблицу, которую она изменяет, и держит блокировку до тех пор, пока транзакция не завершится. Если ресурс удерживается монопольной (X) блокировкой, то другие транзакции не могут изменять данные. Операции считывания будут допускаться только при наличии подсказки NOLOCK или уровня изоляции незафиксированной операции чтения. Дополнительные сведения см. в разделе Блокировка в компоненте Database Engine.

Режим ведения журнала

Инструкция INSERT всегда полностью регистрируется в журнале, кроме случаев использования функции OPENROWSET с ключевым словом BULK или при выполнении инструкции INSERT INTO <целевая\_таблица> SELECT <столбцы> FROM <исходная\_таблица>. Для этих операций возможно минимальное протоколирование. Дополнительные сведения см. в подразделе «Рекомендации по массовой загрузке данных» этого раздела.

Безопасность

При соединении со связанным сервером отправляющий сервер указывает имя входа и пароль для подключения к принимающему серверу от его имени. Для работы этого соединения необходимо создать сопоставление имен входа между связанными серверами с помощью хранимой процедуры sp\_addlinkedsrvlogin. Дополнительные сведения см. в разделе Безопасность для связанных серверов.

При использовании функции OPENROWSET(BULK…) важно понимать, каким образом SQL Server обрабатывает олицетворение. Дополнительные сведения см. в подразделе «Вопросы безопасности» в разделе Массовый импорт данных при помощи инструкции BULK INSERT или OPENROWSET(BULK...).

Разрешения

Требуется разрешение INSERT на целевую таблицу.

Разрешения INSERT предоставлены по умолчанию членам предопределенной роли сервера sysadmin, членам предопределенных ролей баз данных db\_owner и db\_datawriter, а также владельцу таблицы. Члены ролей sysadmin, db\_owner и db\_securityadmin, а также владелец таблицы могут передавать разрешения другим пользователям.

Чтобы выполнить инструкцию INSERT с параметром BULK функции OPENROWSET, необходимо быть членом предопределенной роли сервера sysadmin или bulkadmin.

# UPDATE (Transact-SQL)

SQL Server 2008 R2

Изменяет существующие данные в одном или нескольких столбцах таблицы или представления SQL Server 2008 R2. Примеры см. в разделе Примеры.

Синтаксические обозначения в Transact-SQL

**Синтаксис**

Transact-SQL

[ WITH <common\_table\_expression> [...n] ]

UPDATE

[ TOP (expression) [ PERCENT ] ]

{ { table\_alias | <object> | rowset\_function\_limited

[ WITH ( <Table\_Hint\_Limited> [ ...n ] ) ]

}

| @table\_variable

}

SET

{ column\_name= { expression | DEFAULT | NULL }

| { udt\_column\_name.{ { property\_name= expression

| field\_name=expression }

| method\_name(argument [ ,...n ] )

}

}

| column\_name { .WRITE (expression,@Offset,@Length) }

| @variable=expression

| @variable= column=expression

| column\_name { += | -= | \*= | /= | %= | &= | ^= | |= } expression

| @variable { += | -= | \*= | /= | %= | &= | ^= | |= } expression

| @variable=column { += | -= | \*= | /= | %= | &= | ^= | |= } expression

} [ ,...n ]

[ <OUTPUT Clause> ]

[ FROM { <table\_source> } [ ,...n ] ]

[ WHERE { <search\_condition>

| { [ CURRENT OF

{ { [ GLOBAL ] cursor\_name }

| cursor\_variable\_name

}

]

}

}

]

[ OPTION ( <query\_hint> [ ,...n ] ) ]

[ ; ]

<object> ::=

{

[ server\_name . database\_name . schema\_name .

| database\_name .[ schema\_name ] .

| schema\_name .

]

table\_or\_view\_name}

**Аргументы**

WITH <common\_table\_expression>

Задает временный именованный результирующий набор или представление, которые называются обобщенным табличным выражением (CTE), определяемым в пределах области действия инструкции UPDATE. Результирующий набор обобщенного табличного выражения, на который ссылается инструкция UPDATE, является производным простого запроса. Дополнительные сведения см. в разделе WITH обобщенное\_табличное\_выражение (Transact-SQL).

TOP ( expression) [ PERCENT ]

Указывает количество или процент строк, которые будут обновлены. expression может быть либо числом, либо процентной долей строк.

Строки, на которые ссылается выражение TOP, используемое с инструкцией INSERT, UPDATE, MERGE или DELETE, не расположены в определенном порядке.

В инструкциях INSERT, UPDATE, MERGE и DELETE выражение expression должно быть заключено в скобки. Дополнительные сведения см. в разделе TOP (Transact-SQL).

table\_alias

Псевдоним, заданный в предложении FROM и представляющий таблицу или представление, строки которых будут обновлены.

server\_name

Имя связанного сервера, на котором расположена таблица или индексированное представление. Аргумент server\_name можно задавать в виде имени связанного сервера или с помощью функции OPENDATASOURCE.

Если аргумент server\_name задается в виде связанного сервера, то необходимы аргументы database\_name и schema\_name. Если аргумент server\_name задается с помощью функции OPENDATASOURCE, то аргументы database\_name и schema\_name могут применяться не ко всем источникам данных, в зависимости от возможностей поставщика OLE DB, который обращается к удаленному объекту. Дополнительные сведения см. в разделах Распределенные запросы.

database\_name

Имя базы данных.

schema\_name

Имя схемы, к которой принадлежит таблица или представление.

table\_or view\_name

Имя таблицы или представления, из которых должны обновляться строки. Представление, на которое ссылается аргумент table\_or\_view\_name, должно быть обновляемым и ссылаться ровно на одну базовую таблицу в предложении FROM данного представления. Дополнительные сведения об обновляемых представлениях см. в разделе CREATE VIEW (Transact-SQL).

rowset\_function\_limited

Либо функция OPENQUERY, либо функция OPENROWSET. Использование этих функций зависит от возможностей поставщика OLE DB, который обращается к удаленному объекту. Дополнительные сведения см. в разделе Распределенные запросы.

WITH (<Table\_Hint\_Limited> )

Указывает одну или несколько табличных подсказок, разрешенных для целевой таблицы. Необходимо использовать ключевое слово WITH и круглые скобки. Использование аргументов NOLOCK и READUNCOMMITTED запрещено. Сведения о табличных подсказках см. в разделе Табличные подсказки (Transact-SQL).

@table\_variable

Задает переменную table в качестве табличного источника.

SET

Задает список обновляемых имен столбцов или переменных.

column\_name

Столбец, содержащий обновляемые данные. Столбец с именем column\_name должен существовать в table\_or view\_name. Столбцы идентификаторов не могут быть обновлены.

expression

Переменная, литеральное значение, выражение или инструкция подзапроса выборки (заключенная в скобки), которые возвращают единственное значение. Значение, возвращаемое expression, заменяет существующее значение в column\_name или @variable. Примечание

При ссылке на типы данных символов Юникода nchar, nvarchar и ntext «выражение» должно начинаться с заглавной буквы N. Если префикс N не указан, SQL Server выполнит преобразование строки в кодовую страницу, соответствующую параметрам сортировки базы данных или столбца, действующим по умолчанию. Любые символы, не входящие в эту кодовую страницу, будут утрачены.

DEFAULT

Указывает, что существующее в столбце значение будет заменено значением по умолчанию, определенным для данного столбца. Также может использоваться для присвоения значения NULL, если столбец не имеет значений по умолчанию и может принимать значения NULL.

{ += | -= | \*= | /= | %= | &= | ^= | |= }

Составной оператор присваивания:

+= сложить и присвоить

-= вычесть и присвоить

\*= умножить и присвоить

/= разделить и присвоить

%= получить остаток от деления и присвоить

&= выполнить побитовое И и присвоить

= выполнить побитовое исключающее ИЛИ и присвоить

= выполнить побитовое ИЛИ и присвоить

Дополнительные сведения см. в разделе Составные операторы (Transact-SQL).

udt\_column\_name

Столбец определяемого пользователем типа.

property\_name | field\_name

Общее свойство или общий элемент данных определяемого пользователем типа.

method\_name(argument [ ,... n] )

Не статичный метод общего мутатора udt\_column\_name, принимающий один или несколько аргументов.

.WRITE (expression,@Offset,@Length)

Указывает, что должен быть изменен раздел значения column\_name . expression заменяет в column\_name@Length элементы, начиная с позиции @Offset. С этим предложением можно указывать только столбцы типа varchar(max), nvarchar(max) или varbinary(max). column\_name не может иметь значения NULL и не может дополняться именем таблицы или псевдонимом таблицы.

expression — значение, которое копируется в столбец column\_name. Аргумент expression должен иметь результат типа column\_name или неявно приводиться к этому типу. Если для expression установлено значение NULL, аргумент @Length не учитывается, а значение в column\_name усекается с позиции, на которую указывает аргумент @Offset.

Аргумент @Offset — начальная точка в значении column\_name, в которую записывается значение expression. Аргумент @Offset — отсчитываемая от нуля порядковая позиция с типом данных bigint, может быть отрицательным числом. Если аргумент @Offset имеет значение NULL, операция обновления добавляет значение expression в конец существующего значения аргумента column\_name, а аргумент @Length пропускается. Если значение аргумента @Offset больше, чем длина значения аргумента column\_name, компонент Database Engine возвращает ошибку. Если сумма значений @Offset и @Length превышает длину базового значения столбца, удаление выполняется до последнего символа этого значения. Если сумма значений @Offset и LEN(expression) больше, чем базовый объявленный размер, возникает ошибка.

@Length — это длина раздела в столбце, начиная с @Offset, который заменяется expression. @Length имеет тип данных bigint и не может принимать отрицательные значения. Если аргумент @Length имеет значение NULL, операция обновления удаляет все данные, начиная со значения @Offset до конца значения column\_name.

Дополнительные сведения см. ниже, в разделе «Примечания».

@variable

Объявленная переменная, которой присваивается значение, возвращенное expression.

SET @variable = column = expression присваивает переменной то же значение, что и столбцу. Это отличается от предложения SET @variable = column, column = expression, присваивающего переменной значение столбца до обновления.

<OUTPUT\_Clause>

Возвращает обновленные данные или основанные на них выражения в рамках выполнения операции UPDATE. Предложение OUTPUT не поддерживается ни в одной инструкции DML, целью которой являются удаленные таблицы или представления. Дополнительные сведения см. в разделе Предложение OUTPUT (Transact-SQL).

FROM <источник\_таблицы>

Определяет, что для определения критериев операции обновления используется таблица, представление или производная таблица. Дополнительные сведения см. в разделе Предложение FROM (Transact-SQL).

Если обновляемый объект совпадает с объектом в предложении FROM, а в предложении FROM имеется только одна ссылка на этот объект, псевдоним объекта указывать не обязательно. Если обновляемый объект встречается в предложении FROM несколько раз, одна, и только одна, ссылка на этот объект не должна указывать псевдоним таблицы. Все остальные ссылки на объект в предложении FROM должны включать псевдоним объекта.

Представление с триггером INSTEAD OF UPDATE не может быть целью инструкции UPDATE с предложением FROM.

WHERE

Задает условия, ограничивающие обновляемые строки. Существует два вида обновлений в зависимости от используемой формы предложения WHERE.

В поисковых обновлениях задается условие поиска строк, предназначенных к удалению.

В позиционных обновлениях используется предложение CURRENT OF для указания курсора. Операция обновления выполняется в текущем положении курсора.

<search\_condition>

Задает условие, которому должны удовлетворять обновляемые строки. Условие поиска может также представлять собой условие, на котором основано соединение. Количество предикатов, которое может содержать условие поиска, неограниченно. Дополнительные сведения о предикатах и условиях поиска см. в разделе Условие поиска (Transact-SQL).

CURRENT OF

Определяет, что обновление выполняется в текущей позиции указанного курсора.

GLOBAL

Указывает, что аргумент cursor\_name ссылается на глобальный курсор.

cursor\_name

Имя открытого курсора, из которого должна быть произведена выборка. Если существует как глобальный, так и локальный курсор с именем cursor\_name, этот аргумент ссылается на глобальный курсор, если указан аргумент GLOBAL, в противном случае он ссылается на локальный курсор. Курсор должен позволять производить обновления.

cursor\_variable\_name

Имя переменной курсора. Аргумент cursor\_variable\_name должен ссылаться на курсор, разрешающий обновления.

OPTION ( <подсказка\_запроса> [ ,... n ] )

Определяет, что для настройки способа, которым компонент Database Engine обрабатывает инструкцию, используются подсказки оптимизатора. Дополнительные сведения см. в разделе Подсказки в запросах (Transact-SQL).

**Рекомендации**

В инструкции UPDATE можно использовать имена переменных для показа старых и новых значений, но только в том случае, если инструкция UPDATE обрабатывает одну запись. Если инструкция UPDATE затрагивает несколько записей, для возврата старых и новых значений каждой записи используйте предложение OUTPUT.

Проявляйте осторожность, указывая предложение FROM при задании критериев для операции обновления. Результаты инструкции UPDATE не определены, если инструкция включает предложение FROM, в котором для каждого вхождения обновляемого столбца не задано единственное значение, то есть если инструкция UPDATE не является детерминированной. Это может привести к непредвиденным результатам. Например, в инструкции UPDATE в следующем скрипте обе строки таблицы Table1 удовлетворяют условиям предложения FROM. Поэтому неясно, какая строка таблицы Table1 должна использоваться для обновления строки таблицы Table2.

Transact-SQL

USE AdventureWorks2008R2;

GO

IF OBJECT\_ID ('dbo.Table1', 'U') IS NOT NULL

DROP TABLE dbo.Table1;

GO

IF OBJECT\_ID ('dbo.Table2', 'U') IS NOT NULL

DROP TABLE dbo.Table2;

GO

CREATE TABLE dbo.Table1

(ColA int NOT NULL, ColB decimal(10,3) NOT NULL);

GO

CREATE TABLE dbo.Table2

(ColA int PRIMARY KEY NOT NULL, ColB decimal(10,3) NOT NULL);

GO

INSERT INTO dbo.Table1 VALUES(1, 10.0), (1, 20.0);

INSERT INTO dbo.Table2 VALUES(1, 0.0);

GO

UPDATE dbo.Table2

SET dbo.Table2.ColB = dbo.Table2.ColB + dbo.Table1.ColB

FROM dbo.Table2

INNER JOIN dbo.Table1

ON (dbo.Table2.ColA = dbo.Table1.ColA);

GO

SELECT ColA, ColB

FROM dbo.Table2;

То же самое может произойти при сочетании предложений FROM и WHERE CURRENT OF. В следующем примере обе строки таблицы Table2 удовлетворяют условиям предложения FROM в инструкции UPDATE. Не определено, какая строка из Table2 должна использоваться для обновления строки в Table1.

Transact-SQL

USE AdventureWorks2008R2;

GO

IF OBJECT\_ID ('dbo.Table1', 'U') IS NOT NULL

DROP TABLE dbo.Table1;

GO

IF OBJECT\_ID ('dbo.Table2', 'U') IS NOT NULL

DROP TABLE dbo.Table2;

GO

CREATE TABLE dbo.Table1

(c1 int PRIMARY KEY NOT NULL, c2 int NOT NULL);

GO

CREATE TABLE dbo.Table2

(d1 int PRIMARY KEY NOT NULL, d2 int NOT NULL);

GO

INSERT INTO dbo.Table1 VALUES (1, 10);

INSERT INTO dbo.Table2 VALUES (1, 20), (2, 30);

GO

DECLARE abc CURSOR LOCAL FOR

SELECT c1, c2

FROM dbo.Table1;

OPEN abc;

FETCH abc;

UPDATE dbo.Table1

SET c2 = c2 + d2

FROM dbo.Table2

WHERE CURRENT OF abc;

GO

SELECT c1, c2 FROM dbo.Table1;

GO

**Поддержка совместимости**

Использование инструкции SET ROWCOUNT не будет оказывать влияния на инструкции DELETE, INSERT, MERGE и UPDATE в следующей версии SQL Server. В новых разработках следует избегать совместного использования инструкции SET ROWCOUNT с инструкциями DELETE, INSERT и UPDATE. Также необходимо запланировать внесение изменений в используемые приложения. Кроме того, рекомендуется переписать инструкции DELETE, INSERT и UPDATE, которые используют инструкцию SET ROWCOUNT в настоящее время, используя ключевое слово TOP.

Поддержка использования подсказок READUNCOMMITTED и NOLOCK в предложении FROM, применяемом к целевой таблице инструкции UPDATE или DELETE, будет удалена в следующей версии SQL Server. Следует избегать использования этих подсказок в таком контексте в новой разработке и запланировать изменение приложений, использующих их в настоящий момент.

**Типы данных**

Все столбцы типов char и nchar дополняются справа до заданной длины.

Если параметр ANSI\_PADDING имеет значение OFF, все конечные пробелы удаляются из данных, вставленных в столбцы varchar и nvarchar, за исключением строк, содержащих только пробелы. Эти строки усекаются до пустых строк. Если ANSI\_PADDING имеет значение ON, вставляются конечные пробелы. Драйвер ODBC для Microsoft SQL Server и поставщик OLE DB для SQL Server автоматически устанавливают ANSI\_PADDING ON для каждого соединения. Этот параметр можно настроить в источниках данных ODBC или устанавливая атрибуты или свойства соединений. Дополнительные сведения см. в разделе SET ANSI\_PADDING (Transact-SQL).

**Обновление типов данных большого объема**

Для частичного или полного обновления типов данных varchar(max), nvarchar(max) и varbinary(max) используется предложение .WRITE (expression, @Offset,@Length). Например, частичное обновление столбца с типом данных varchar(max) может удалить или изменить только первые 200 символов, тогда как полное обновление удалит или изменит все данные в столбце.

В целях увеличения производительности рекомендуется вставлять или обновлять данные фрагментами, кратными 8040 байтам.

Компонент Database Engine преобразует частичное обновление в полное, если инструкция UPDATE приводит к одному из следующих действий.

Изменения ключевого столбца секционированного представления или таблицы.

Изменение более одной строки, а также обновление ключа неуникального кластеризованного индекса на непостоянное значение.

Нельзя использовать предложение . WRITE для обновления столбца NULL или для присваивания аргументу column\_name значения NULL.

Параметры @Offset и @Length указываются в байтах для типов данных varbinary и varchar и в символах для типа данных nvarchar. Соответствующие смещения вычисляются для параметров сортировки в двухбайтовых кодировках (DBCS).

Если на столбец, изменяемый предложением .WRITE, ссылается предложение OUTPUT, то указанному столбцу в табличной переменной возвращается полное значение данного столбца в первоначальном образе deleted.column\_name или в последующем образе inserted.column\_name.

Чтобы добиться функциональности предложения .WRITE при обработке других символьных или двоичных типов данных, используется STUFF (Transact-SQL).

Обновление данных FILESTREAM

Инструкция UPDATE позволяет обновить поля FILESTREAM значением NULL, пустым значением или встроенными данными относительно небольшого размера. Однако при работе с большими объемами данных более эффективно передавать поток в файл с использованием интерфейсов Win32. При обновлении поля FILESTREAM происходит изменение базовых данных BLOB в файловой системе. Если в поле FILESTREAM содержится значение NULL, данные BLOB, связанные с этим полем, удаляются. Для частичного обновления данных потока FILESTREAM недопустимо использовать метод .WRITE(). Дополнительные сведения см. в разделе Общие сведения о FILESTREAM.

Обновление столбцов типа text, ntext и image

Изменение столбцов типа text, ntext или image с помощью инструкции UPDATE инициализирует столбец, присваивает ему допустимый текстовый указатель и выделяет по крайней мере одну страницу данных, если столбец не обновляется значением NULL. Если инструкция UPDATE могла обновить несколько строк при обновлении как кластеризованного ключа, так и одного или нескольких столбцов типа text, ntext или image, частичное обновление этих столбцов выполняется как полная замена значений.

Чтобы заменить или изменить большие блоки данных типов text, ntext или image, вместо UPDATE используется инструкция WRITETEXT или UPDATETEXT. Важно!

Типы данных ntext, text и image будут удалены в следующей версии MicrosoftSQL Server. Следует избегать использования этих типов данных при новой разработке и запланировать изменение приложений, использующих их в настоящий момент. Вместо этого следует использовать типы данных nvarchar(max), varchar(max) и varbinary(max). Дополнительные сведения см. в разделе Использование типов данных больших значений.

**Обработка ошибок**

Для инструкции UPDATE можно реализовать обработку ошибок, заключив ее в конструкцию TRY…CATCH. Дополнительные сведения см. в разделе Использование конструкции TRY...CATCH в языке Transact-SQL.

Если обновление строки нарушает ограничение, правило или установку NULL для столбца либо новое значение имеет несовместимый тип данных, то инструкция отменяется, возвращается ошибка и никакие записи не обновляются.

Если инструкция UPDATE при оценке выражения встречает арифметическую ошибку (переполнение, деление на ноль или ошибку домена), обновление не выполняется. Остальная часть пакета не выполняется и возвращается сообщение об ошибке.

Если обновление столбца или столбцов, участвующих в кластеризованном индексе, приводит к тому, что размер кластеризованного индекса и строки превышает 8 060 байт, обновление заканчивается неудачей и возвращается сообщение об ошибке.

**Совместимость**

Инструкции UPDATE разрешается использовать в теле определяемых пользователем функций только в том случае, если изменяемая таблица является табличной переменной.

Если для операций UPDATE по отношению к таблице определен триггер INSTEAD OF, вместо инструкции UPDATE запускается этот триггер. Предыдущие версии SQL Server поддерживали для инструкции UPDATE и других инструкций изменения данных только триггеры AFTER.

**Ограничения**

В инструкции UPDATE, которая прямо или косвенно ссылается на представление с определенным для него триггером INSTEAD OF, не может быть указано предложение FROM. Дополнительные сведения о триггерах INSTEAD OF см. в разделе CREATE TRIGGER (Transact-SQL).

Значение параметра SET ROWCOUNT не учитывается инструкциями UPDATE для удаленных таблиц и для локальных и удаленных секционированных представлений.

Если обобщенное табличное выражение указывается в качестве цели инструкции UPDATE, должны совпадать все ссылки на это выражение в инструкции. Например, если для обобщенного табличного выражения в предложении FROM назначается псевдоним, то этот псевдоним должен использоваться для всех остальных ссылок на обобщенное табличное выражение. Однозначные ссылки на обобщенное табличное выражение необходимы в силу того, что обобщенное табличное выражение не имеет идентификатора объекта, который используется SQL Server для обнаружения неявной связи между объектом и псевдонимом. В отсутствие такой связи план запроса может непредвиденным образом построить работу с соединениями, что приведет к нежелательным результатам запроса. В следующих примерах показаны правильные и неверные методы задания обобщенного табличного выражения, когда оно является целевым объектом операции обновления.

Transact-SQL

USE tempdb;

GO

-- UPDATE statement with CTE references that are correctly matched.

DECLARE @x TABLE (ID int, Value int);

DECLARE @y TABLE (ID int, Value int);

INSERT @x VALUES (1, 10), (2, 20);

INSERT @y VALUES (1, 100),(2, 200);

WITH cte AS (SELECT \* FROM @x)

UPDATE x -- cte is referenced by the alias.

SET Value = y.Value

FROM cte AS x -- cte is assigned an alias.

INNER JOIN @y AS y ON y.ID = x.ID;

SELECT \* FROM @x;

GO

Ниже приводится результирующий набор.

ID Value

------ -----

1 100

2 200

(2 row(s) affected)

Transact-SQL

-- UPDATE statement with CTE references that are incorrectly matched.

USE tempdb;

GO

DECLARE @x TABLE (ID int, Value int);

DECLARE @y TABLE (ID int, Value int);

INSERT @x VALUES (1, 10), (2, 20);

INSERT @y VALUES (1, 100),(2, 200);

WITH cte AS (SELECT \* FROM @x)

UPDATE cte -- cte is not referenced by the alias.

SET Value = y.Value

FROM cte AS x -- cte is assigned an alias.

INNER JOIN @y AS y ON y.ID = x.ID;

SELECT \* FROM @x;

GO

Ниже приводится результирующий набор.

ID Value

------ -----

1 100

2 100

(2 row(s) affected)

Режим ведения журнала

Инструкция UPDATE протоколируется полностью, однако обновления с помощью метода .WRITE, вставляющие или присоединяющие новые данные, протоколируются на минимальном уровне, если установлена простая модель восстановления базы данных или модель восстановления с неполным протоколированием. Протоколирование не сокращается до минимума, если при помощи метода .WRITE обновляются существующие значения. Дополнительные сведения см. в разделе Операции, для которых возможно минимальное протоколирование.

Безопасность

Разрешения

Требуются разрешения на выполнение UPDATE в целевой таблице. Также требуются разрешения SELECT на обновляемую таблицу, если инструкция UPDATE содержит предложение WHERE или если аргумент expression в предложении SET использует столбец в этой таблице.

Разрешения UPDATE по умолчанию предоставляются членам предопределенной роли сервера sysadmin, членам предопределенных ролей баз данных db\_owner и db\_datawriter, а также владельцу таблицы. Члены ролей sysadmin, db\_owner и db\_securityadmin, а также владелец таблицы могут передавать разрешения другим пользователям.

**Примеры**

Категория

Используемые элементы синтаксиса

Базовый синтаксис

UPDATE

Ограничение обновляемых строк

WHERE • TOP • WITH обобщенное табличное выражение • WHERE CURRENT OF

Настройка значений столбца

вычисляемые значения • составные операторы • значения по умолчанию • вложенные запросы

Указание целевых объектов, отличных от стандартных таблиц

представления • табличные переменные • псевдонимы таблицы

Обновление данных с использованием данных из других таблиц

FROM

Обновление строк в удаленной таблице

связанный сервер • OPENQUERY • OPENDATASOURCE

Обновление типов данных больших объектов

.WRITE • OPENROWSET

Обновление определяемых пользователем типов данных

определяемые пользователем типы

Переопределение поведения по умолчанию оптимизатора запросов с помощью подсказок

табличные подсказки • подсказки в запросах

Сбор результатов инструкции UPDATE

Предложение OUTPUT

Использование инструкции UPDATE в других инструкциях

Хранимые процедуры • TRY…CATCH

Базовый синтаксис

В примерах из этого раздела показаны основные функции инструкции UPDATE и используется минимально необходимый синтаксис.

A. Использование простой инструкции UPDATE

В следующем примере обновляется один столбец для всех строк в таблице Person.Address.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Person.Address

SET ModifiedDate = GETDATE();

Б. Обновление нескольких столбцов

В следующем примере выполняется обновление значений в столбцах Bonus, CommissionPct и SalesQuota для всех строк в таблице SalesPerson.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Sales.SalesPerson

SET Bonus = 6000, CommissionPct = .10, SalesQuota = NULL;

GO

Ограничение обновляемых строк

В примерах в этом разделе описываются способы ограничения количества строк, на которые влияет инструкция UPDATE.

A. Использование предложения WHERE

В следующем примере предложение WHERE используется для указания строк, которые необходимо обновить. Инструкция обновляет значение в столбце Color таблицы Production.Product для всех строк, в которых имеется существующее значение Red в столбце Color и имеется значение в столбце Name, который начинается с Road-250.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Production.Product

SET Color = N'Metallic Red'

WHERE Name LIKE N'Road-250%' AND Color = N'Red';

GO

Б. Использование предложения TOP

В следующем примере предложение TOP используется для ограничения числа строк, изменяемых в процессе выполнения инструкции UPDATE. Если в инструкции UPDATE указано предложение TOP (n), операция обновления выполняется для произвольного подмножества в n строк. В следующем примере столбец VacationHours в таблице Employee обновляется на 25 % для 10 произвольных строк.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE TOP (10) HumanResources.Employee

SET VacationHours = VacationHours \* 1.25 ;

GO

Если нужно применить изменения с предложением TOP в определенной последовательности, укажите в подзапросе выборки SELECT инструкцию ORDER BY. В следующем примере изменяется длительность отпуска для 10 сотрудников, имеющих наибольший стаж работы.

Transact-SQL

UPDATE HumanResources.Employee

SET VacationHours = VacationHours + 8

FROM (SELECT TOP 10 BusinessEntityID FROM HumanResources.Employee

ORDER BY HireDate ASC) AS th

WHERE HumanResources.Employee.BusinessEntityID = th.BusinessEntityID;

GO

В. Использование предложения WITH обобщенное\_табличное\_выражение

В следующем примере обновляется значение PerAssemnblyQty для всех частей и компонентов, прямо или косвенно используемых для создания ProductAssemblyID 800. Обобщенное табличное выражение возвращает иерархический список частей, которые непосредственно используются для сборки ProductAssemblyID 800, и частей, которые используются для сборки этих компонентов, и т. д. Изменяются только строки, возвращаемые обобщенным табличным выражением. Дополнительные сведения о рекурсивных обобщенных табличных выражениях см. в разделе Рекурсивные запросы, использующие обобщенные табличные выражения.

Transact-SQL

USE AdventureWorks2008R2;

GO

WITH Parts(AssemblyID, ComponentID, PerAssemblyQty, EndDate, ComponentLevel) AS

(

SELECT b.ProductAssemblyID, b.ComponentID, b.PerAssemblyQty,

b.EndDate, 0 AS ComponentLevel

FROM Production.BillOfMaterials AS b

WHERE b.ProductAssemblyID = 800

AND b.EndDate IS NULL

UNION ALL

SELECT bom.ProductAssemblyID, bom.ComponentID, p.PerAssemblyQty,

bom.EndDate, ComponentLevel + 1

FROM Production.BillOfMaterials AS bom

INNER JOIN Parts AS p

ON bom.ProductAssemblyID = p.ComponentID

AND bom.EndDate IS NULL

)

UPDATE Production.BillOfMaterials

SET PerAssemblyQty = c.PerAssemblyQty \* 2

FROM Production.BillOfMaterials AS c

JOIN Parts AS d ON c.ProductAssemblyID = d.AssemblyID

WHERE d.ComponentLevel = 0;

Г. Использование предложения WHERE CURRENT OF

В следующем примере предложение WHERE CURRENT OF используется только для обновления строк, на которых установлен курсор. Если курсор основан на соединении, изменяется только таблица table\_name, указанная в инструкции UPDATE. Другие таблицы, участвующие в курсоре, не затрагиваются.

Transact-SQL

USE AdventureWorks2008R2;

GO

DECLARE complex\_cursor CURSOR FOR

SELECT a.BusinessEntityID

FROM HumanResources.EmployeePayHistory AS a

WHERE RateChangeDate <>

(SELECT MAX(RateChangeDate)

FROM HumanResources.EmployeePayHistory AS b

WHERE a.BusinessEntityID = b.BusinessEntityID) ;

OPEN complex\_cursor;

FETCH FROM complex\_cursor;

UPDATE HumanResources.EmployeePayHistory

SET PayFrequency = 2

WHERE CURRENT OF complex\_cursor;

CLOSE complex\_cursor;

DEALLOCATE complex\_cursor;

GO

Настройка значений столбца

В примерах этого раздела описывается обновление столбцов с помощью вычисляемых значений, вложенных запросов и значений DEFAULT.

A. Указание вычисляемых значений

В следующих примерах используются вычисляемые значения в инструкции UPDATE. В примере удваивается значение столбца ListPrice для всех строк в таблице Product.

Transact-SQL

USE AdventureWorks2008R2 ;

GO

UPDATE Production.Product

SET ListPrice = ListPrice \* 2;

GO

Б. Задание составного оператора

В следующем примере демонстрируется использование переменной @NewPrice для увеличения цены красных велосипедов прибавлением 10 к текущей цене.

Transact-SQL

USE AdventureWorks2008R2;

GO

DECLARE @NewPrice int = 10;

UPDATE Production.Product

SET ListPrice += @NewPrice

WHERE Color = N'Red';

GO

В следующем примере используется составной оператор + = для добавления данных ' - tool malfunction' к существующему значению в столбце Name для строк, имеющих значение ScrapReasonID от 10 до 12.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Production.ScrapReason

SET Name += ' - tool malfunction'

WHERE ScrapReasonID BETWEEN 10 and 12;

В. Задание вложенного запроса в предложении SET

В следующем примере используется вложенный запрос в предложении SET для определения значения, которое используется для обновления столбца. Вложенный запрос должен возвращать только скалярное значение (то есть одно значение для каждой строки). В примере изменяется столбец SalesYTD в таблице SalesPerson для отображения самой последней информации о продажах, зафиксированной в таблице SalesOrderHeader. Вложенный запрос проводит статистическую обработку сведений о продажах по всем продавцам в инструкции UPDATE.Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Sales.SalesPerson

SET SalesYTD = SalesYTD +

(SELECT SUM(so.SubTotal)

FROM Sales.SalesOrderHeader AS so

WHERE so.OrderDate = (SELECT MAX(OrderDate)

FROM Sales.SalesOrderHeader AS so2

WHERE so2.SalesPersonID = so.SalesPersonID)

AND Sales.SalesPerson.BusinessEntityID = so.SalesPersonID

GROUP BY so.SalesPersonID);

GO

Г. Обновление строк с использованием значений DEFAULT

В следующем примере для столбца CostRate задается значение по умолчанию (0.00) для всех строк, значение CostRate которых больше 20.00.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Production.Location

SET CostRate = DEFAULT

WHERE CostRate > 20.00;

Указание целевых объектов, отличных от стандартных таблиц

В примерах этого раздела описаны методы обновления строк с указанием представления, псевдонима таблицы или табличной переменной.

A. Указание представления в качестве целевого объекта

В следующем примере выполняется обновление строк таблицы путем указания представления в качестве целевого объекта. Определение представления ссылается на несколько таблиц, однако инструкция UPDATE была успешно выполнена, поскольку она ссылается на столбцы только одной из базовых таблиц. При выполнении инструкции UPDATE произойдет сбой, если были указаны столбцы из обеих таблиц. Дополнительные сведения см. в разделе Изменение данных через представление.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Person.vStateProvinceCountryRegion

SET CountryRegionName = 'United States of America'

WHERE CountryRegionName = 'United States';

Б. Задание псевдонима таблицы в качестве целевого объекта

В следующем примере производится обновление строк в таблице Production.ScrapReason. Псевдоним таблицы, заданный для ScrapReason в предложении FROM, указывается как целевой объект в предложении UPDATE.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE sr

SET sr.Name += ' - tool malfunction'

FROM Production.ScrapReason AS sr

JOIN Production.WorkOrder AS wo

ON sr.ScrapReasonID = wo.ScrapReasonID

AND wo.ScrappedQty > 300;

В. Задание табличной переменной в качестве целевого объекта

В следующем примере производится обновление строк в табличной переменной.

Transact-SQL

USE AdventureWorks2008R2;

GO

-- Create the table variable.

DECLARE @MyTableVar table(

EmpID int NOT NULL,

NewVacationHours int,

ModifiedDate datetime);

-- Populate the table variable with employee ID values from HumanResources.Employee.

INSERT INTO @MyTableVar (EmpID)

SELECT BusinessEntityID FROM HumanResources.Employee;

-- Update columns in the table variable.

UPDATE @MyTableVar

SET NewVacationHours = e.VacationHours + 20,

ModifiedDate = GETDATE()

FROM HumanResources.Employee AS e

WHERE e.BusinessEntityID = EmpID;

-- Display the results of the UPDATE statement.

SELECT EmpID, NewVacationHours, ModifiedDate FROM @MyTableVar

ORDER BY EmpID;

GO

Обновление данных с использованием данных из других таблиц

В примерах этого раздела описаны методы обновления строк одной таблицы на основе данных в другой таблице.

A. Использование инструкции UPDATE с данными из другой таблицы

В следующем примере изменяется столбец SalesYTD в таблице SalesPerson для отображения самой последней информации о продажах, зафиксированной в таблице SalesOrderHeader.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Sales.SalesPerson

SET SalesYTD = SalesYTD + SubTotal

FROM Sales.SalesPerson AS sp

JOIN Sales.SalesOrderHeader AS so

ON sp.BusinessEntityID = so.SalesPersonID

AND so.OrderDate = (SELECT MAX(OrderDate)

FROM Sales.SalesOrderHeader

WHERE SalesPersonID = sp.BusinessEntityID);

GO

В предыдущем примере подразумевается, что на конкретного менеджера по продажам на каждую конкретную дату записана только одна продажа и выполнены все текущие обновления. Если в один и тот же день для указанного менеджера может иметься более одной продажи, приведенный пример будет работать неправильно. Пример выполняется без ошибок, но каждое из значений SalesYTD обновляется только для одной из продаж, вне зависимости от действительного количества продаж в этот день. Это происходит потому, что одиночная инструкция UPDATE никогда не обновляет одну и ту же строку дважды.

В ситуации, когда у данного менеджера по продажам имеется несколько продаж в день, все продажи для каждого из менеджеров по продажам должны быть собраны вместе инструкцией UPDATE, как показано в следующем примере.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Sales.SalesPerson

SET SalesYTD = SalesYTD +

(SELECT SUM(so.SubTotal)

FROM Sales.SalesOrderHeader AS so

WHERE so.OrderDate = (SELECT MAX(OrderDate)

FROM Sales.SalesOrderHeader AS so2

WHERE so2.SalesPersonID = so.SalesPersonID)

AND Sales.SalesPerson.BusinessEntityID = so.SalesPersonID

GROUP BY so.SalesPersonID);

GO

Обновление строк в удаленной таблице

В примерах в этом разделе описаны способы обновления строк в удаленной целевой таблице с использованием в качестве ссылки на удаленную таблицу связанного сервера или функции, возвращающей набор строк.

А. Обновление данных в удаленной таблице с использованием связанного сервера

В следующем примере обновляется таблица на удаленном сервере. Этот пример начинается с создания ссылки на удаленный источник данных с помощью хранимой процедуры sp\_addlinkedserver. Затем имя связанного сервера MyLinkServer указывается в качестве одного из четырех компонентов имени объекта в формате server.catalog.schema.object. Обратите внимание, что необходимо указать действительное имя сервера для @datasrc.

Transact-SQL

USE master;

GO

-- Create a link to the remote data source.

-- Specify a valid server name for @datasrc as 'server\_name' or 'server\_name\instance\_name'.

EXEC sp\_addlinkedserver @server = N'MyLinkServer',

@srvproduct = N' ',

@provider = N'SQLNCLI10',

@datasrc = N'<server name>',

@catalog = N'AdventureWorks2008R2';

GO

USE AdventureWorks2008R2;

GO

-- Specify the remote data source using a four-part name

-- in the form linked\_server.catalog.schema.object.

UPDATE MyLinkServer.AdventureWorks.HumanResources.Department

SET GroupName = N'Public Relations'

WHERE DepartmentID = 4;

Б. Обновление данных в удаленной таблице с помощью функции OPENQUERY

В следующем примере выполняется обновление строки в удаленной таблице с помощью вызова функции OPENQUERY, возвращающей набор строк. В этом примере используется имя связанного сервера, созданного в предыдущем примере.

Transact-SQL

UPDATE OPENQUERY (MyLinkServer, 'SELECT GroupName FROM HumanResources.Department WHERE DepartmentID = 4')

SET GroupName = 'Sales and Marketing';

В. Обновление данных в удаленной таблице с помощью функции OPENDATASOURCE

В следующем примере выполняется вставка строки в удаленную таблицу с помощью вызова функции OPENDATASOURCE, возвращающей набор строк. Укажите для источника данных допустимое имя сервера в формате server\_name или server\_name\instance\_name. Возможно, потребуется настроить у экземпляра SQL Server параметр Ad Hoc Distributed Queries. Дополнительные сведения см. в разделе Параметр ad hoc distributed queries.

Transact-SQL

UPDATE OPENQUERY (MyLinkServer, 'SELECT GroupName FROM HumanResources.Department WHERE DepartmentID = 4')

SET GroupName = 'Sales and Marketing';

Обновление типов данных больших объектов

В примерах в этом разделе описываются методы обновления значений в столбцах, определенных с типами данных больших объектов (LOB).

A. Использование инструкции UPDATE с предложением .WRITE для изменения данных в столбце nvarchar(max)

В следующем примере предложение .WRITE используется для обновления частичного значения столбца DocumentSummary типа nvarchar(max) в таблице Production.Document . Слово components заменяется словом features путем указания слова для замены, начального положения (смещения) заменяемого слова в существующих данных и количества заменяемых символов (длины). В этом примере также используется предложение OUTPUT для возвращения первоначального и последующего образов столбца DocumentSummary в табличную переменную @MyTableVar.

Transact-SQL

USE AdventureWorks2008R2;

GO

DECLARE @MyTableVar table (

SummaryBefore nvarchar(max),

SummaryAfter nvarchar(max));

UPDATE Production.Document

SET DocumentSummary .WRITE (N'features',28,10)

OUTPUT deleted.DocumentSummary,

inserted.DocumentSummary

INTO @MyTableVar

WHERE Title = N'Front Reflector Bracket Installation';

SELECT SummaryBefore, SummaryAfter

FROM @MyTableVar;

GO

Б. Использование UPDATE с предложением .WRITE для добавления и удаления данных в столбце типа nvarchar(max)

В следующем примере данные добавляются в столбец типа nvarchar(max), имеющий текущее значение NULL, и удаляются из него. Поскольку предложение .WRITE не может использоваться для изменения столбца со значением NULL, этот столбец сначала заполняется временными данными. Затем они заменяются правильными данными с помощью предложения .WRITE. В дополнительных примерах данные добавляются в конец значения столбца, удаляются из столбца (усекаются) и, наконец, удаляются частичные данные из столбца. Инструкции SELECT выводят на экран изменения данных, создаваемые каждой из инструкций UPDATE.

Transact-SQL

USE AdventureWorks2008R2;

GO

-- Replacing NULL value with temporary data.

UPDATE Production.Document

SET DocumentSummary = N'Replacing NULL value'

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

SELECT DocumentSummary

FROM Production.Document

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

-- Replacing temporary data with the correct data. Setting @Length to NULL

-- truncates all existing data from the @Offset position.

UPDATE Production.Document

SET DocumentSummary .WRITE(N'Carefully inspect and maintain the tires and crank arms.',0,NULL)

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

SELECT DocumentSummary

FROM Production.Document

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

-- Appending additional data to the end of the column by setting

-- @Offset to NULL.

UPDATE Production.Document

SET DocumentSummary .WRITE (N' Appending data to the end of the column.', NULL, 0)

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

SELECT DocumentSummary

FROM Production.Document

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

-- Removing all data from @Offset to the end of the existing value by

-- setting expression to NULL.

UPDATE Production.Document

SET DocumentSummary .WRITE (NULL, 56, 0)

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

SELECT DocumentSummary

FROM Production.Document

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

-- Removing partial data beginning at position 9 and ending at

-- position 21.

UPDATE Production.Document

SET DocumentSummary .WRITE ('',9, 12)

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

SELECT DocumentSummary

FROM Production.Document

WHERE Title = N'Crank Arm and Tire Maintenance';

GO

В. Использование инструкции UPDATE с функцией OPENROWSET для изменения столбца типа varbinary(max)

В следующем примере существующее изображение в столбце varbinary(max) заменяется новым изображением. Для загрузки изображения в столбец используется функция OPENROWSET с параметром BULK. В этом примере предполагается, что по заданному пути существует файл с именем Tires.jpg.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Production.ProductPhoto

SET ThumbNailPhoto = (

SELECT \*

FROM OPENROWSET(BULK 'c:\Tires.jpg', SINGLE\_BLOB) AS x )

WHERE ProductPhotoID = 1;

GO

Г. Использование UPDATE для изменения данных FILESTREAM

В следующем примере инструкция UPDATE используется для изменения данных в файлах файловой системы. Не рекомендуется использовать этот метод для потоковой передачи больших объемов данных в файл. Используйте соответствующие интерфейсы Win32. В следующем примере любой текст в записи файла заменяется текстом Xray 1. Дополнительные сведения см. в разделе Общие сведения о FILESTREAM.

SQL

UPDATE Archive.dbo.Records

SET [Chart] = CAST('Xray 1' as varbinary(max))

WHERE [SerialNumber] = 2;

Обновление определяемых пользователем типов данных

В следующих примерах изменяются значения в столбцах определяемых пользователем типов данных CLR. Описываются три метода. Дополнительные сведения об определяемых пользователем столбцах см. в разделе Определяемые пользователем типы данных CLR.

A. Использование системных типов данных

Определяемый пользователем тип можно обновить путем предоставления значения в типе системных данных SQL Server при поддержке определяемым пользователем типа явного или неявного преобразования из этого типа. Следующий пример демонстрирует, как обновить значение в столбце определяемого пользователем типа Point путем явного преобразования строки.

other

UPDATE dbo.Cities

SET Location = CONVERT(Point, '12.3:46.2')

WHERE Name = 'Anchorage';

Б. Вызов метода

Определяемый пользователем тип можно обновить путем вызова метода, отмеченного в качестве мутатора определяемого пользователем типа, для выполнения обновления. Следующий пример вызывает метод мутатора типа Point с именем SetXY. Это обновляет состояние экземпляра типа.

other

UPDATE dbo.Cities

SET Location.SetXY(23.5, 23.5)

WHERE Name = 'Anchorage';

В. Изменение значения свойства или элемента данных

Определяемый пользователем тип можно обновить путем изменения значения зарегистрированного свойства или общедоступного элемента данных пользовательского типа. Выражение, предоставляющее значение, должно допускать неявное преобразование в тип свойства. В следующем примере изменяется значение свойства X определяемого пользователем типа Point.

other

UPDATE dbo.Cities

SET Location.X = 23.5

WHERE Name = 'Anchorage';

Переопределение поведения по умолчанию оптимизатора запросов с помощью подсказок

Примеры в этом разделе описывают использование табличных подсказок и подсказок в запросах для временного переопределения поведения оптимизатора запросов при обработке инструкции UPDATE. Внимание!

Поскольку оптимизатор запросов SQL Server обычно выбирает наилучший план выполнения запроса, подсказки рекомендуется использовать только опытным разработчикам и администраторам баз данных в качестве последнего средства.

A. Задание табличной подсказки

В следующем примере задается табличная подсказка TABLOCK. Эта подсказка указывает, что на таблицу Production.Product накладывается совмещаемая блокировка, удерживаемая до завершения инструкции UPDATE.

Transact-SQL

USE AdventureWorks2008R2;

GO

UPDATE Production.Product

WITH (TABLOCK)

SET ListPrice = ListPrice \* 1.10

WHERE ProductNumber LIKE 'BK-%';

GO

Б. Задание подсказки в запросе

В следующем примере задается подсказка в запросеOPTIMIZE FOR (@variable) в инструкции UPDATE. Эта подсказка указывает на необходимость использования оптимизатором запросов при компиляции и оптимизации запросов конкретного значения локальной переменной. Значение используется только в процессе оптимизации запроса, но не в процессе выполнения.

Transact-SQL

USE AdventureWorks2008R2;

GO

CREATE PROCEDURE Production.uspProductUpdate

@Product nvarchar(25)

AS

SET NOCOUNT ON;

UPDATE Production.Product

SET ListPrice = ListPrice \* 1.10

WHERE ProductNumber LIKE @Product

OPTION (OPTIMIZE FOR (@Product = 'BK-%') );

GO

-- Execute the stored procedure

EXEC Production.uspProductUpdate 'BK-%';

Сбор результатов инструкции UPDATE

Примеры в этом разделе описывают использование предложения OUTPUT для возврата данных для всех строк, изменившихся в результате выполнения инструкции UPDATE, или выражений на основе этих данных. Эти результаты могут быть возвращены приложению, например для вывода подтверждающих сообщений, архивирования и т. п.

A. Использование инструкции UPDATE с предложением OUTPUT

В следующем примере значения в столбце VacationHours в первых 10 строках таблицы Employee уменьшаются до 25 % от исходных значений, а в столбец ModifiedDate устанавливается текущая дата. Предложение OUTPUT возвращает значение VacationHours, существующее до применения инструкции UPDATE в столбце deleted.VacationHours, и обновленное значение в столбце inserted.VacationHours к табличной переменной @MyTableVar.

Две следующие инструкции SELECT возвращают значения в табличную переменную @MyTableVar, а результаты операции обновления — в таблицу Employee. Дополнительные примеры использования предложения OUTPUT см. в разделе Предложение OUTPUT (Transact-SQL).

Transact-SQL

USE AdventureWorks2008R2;

GO

DECLARE @MyTableVar table(

EmpID int NOT NULL,

OldVacationHours int,

NewVacationHours int,

ModifiedDate datetime);

UPDATE TOP (10) HumanResources.Employee

SET VacationHours = VacationHours \* 1.25,

ModifiedDate = GETDATE()

OUTPUT inserted.BusinessEntityID,

deleted.VacationHours,

inserted.VacationHours,

inserted.ModifiedDate

INTO @MyTableVar;

--Display the result set of the table variable.

SELECT EmpID, OldVacationHours, NewVacationHours, ModifiedDate

FROM @MyTableVar;

GO

--Display the result set of the table.

SELECT TOP (10) BusinessEntityID, VacationHours, ModifiedDate

FROM HumanResources.Employee;

GO

Использование инструкции UPDATE в других инструкциях

В примерах в этом разделе описывается использование UPDATE в других инструкциях.

A. Использование UPDATE в хранимой процедуре

В следующем примере инструкция UPDATE используется в хранимой процедуре. Процедура принимает один входной параметр @NewHours и один выходной параметр @RowCount. Значение параметра @NewHours используется в инструкции UPDATE для обновления столбца VacationHours в таблице HumanResources.Employee. Выходной параметр @RowCount используется для возврата значения числа задействованных строк в локальную переменную. Выражение CASE используется в предложении SET для условного определения значения, которое задано для столбца VacationHours. Если для сотрудника применяется почасовая ставка оплаты (SalariedFlag = 0), то в столбце VacationHours устанавливается текущее количество часов плюс значение, заданное в @NewHours. В противном случае в столбце VacationHours указывается значение, заданное в @NewHours.

Transact-SQL

USE AdventureWorks2008R2;

GO

CREATE PROCEDURE HumanResources.Update\_VacationHours

@NewHours smallint

AS

SET NOCOUNT ON;

UPDATE HumanResources.Employee

SET VacationHours =

( CASE

WHEN SalariedFlag = 0 THEN VacationHours + @NewHours

ELSE @NewHours

END

)

WHERE CurrentFlag = 1;

GO

EXEC HumanResources.Update\_VacationHours 40;

Б. Использование UPDATE в блоке TRY…CATCH

В следующем примере инструкция UPDATE используется в блоке TRY…CATCH для обработки ошибок выполнения, которые могут возникнуть во время операции обновления. Дополнительные сведения см. в разделе Использование конструкции TRY...CATCH в языке Transact-SQL.

Transact-SQL

USE AdventureWorks2008R2;

GO

BEGIN TRANSACTION;

BEGIN TRY

-- Intentionally generate a constraint violation error.

UPDATE HumanResources.Department

SET Name = N'MyNewName'

WHERE DepartmentID BETWEEN 1 AND 2;

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

IF @@TRANCOUNT > 0

COMMIT TRANSACTION;

GO

# DELETE (Transact-SQL)

SQL Server 2008 R2

Удаляет строки из таблиц и представлений.

Синтаксические обозначения в Transact-SQL

Синтаксис

other

[ WITH <common\_table\_expression> [ ,...n ] ]

DELETE

[ TOP ( expression ) [ PERCENT ] ]

[ FROM ]

{ <object> | rowset\_function\_limited

[ WITH ( <table\_hint\_limited> [ ...n ] ) ]

}

[ <OUTPUT Clause> ]

[ FROM <table\_source> [ ,...n ] ]

[ WHERE { <search\_condition>

| { [ CURRENT OF

{ { [ GLOBAL ] cursor\_name }

| cursor\_variable\_name

}

]

}

}

]

[ OPTION ( <Query Hint> [ ,...n ] ) ]

[; ]

<object> ::=

{

[ server\_name.database\_name.schema\_name.

| database\_name. [ schema\_name ] .

| schema\_name.

]

table\_or\_view\_name

}

Аргументы

WITH <common\_table\_expression>

Задает временный именованный результирующий набор, также называемый обобщенным табличным выражением, который определяется в области действия инструкции DELETE. Результирующий набор получается из инструкции SELECT.

Обобщенные табличные выражения также можно использовать в инструкциях SELECT, INSERT, UPDATE и CREATE VIEW. Дополнительные сведения см. в разделе WITH обобщенное\_табличное\_выражение (Transact-SQL).

TOP (expression) [ PERCENT ]

Задает число или процент случайных строк для удаления. Выражение expression может быть либо числом, либо процентом строк. Строки, на которые ссылается выражение TOP, используемое с инструкциями INSERT, UPDATE и DELETE, не упорядочиваются.

Разделение круглыми скобками выражения expression в выражении TOP требуется в инструкциях INSERT, UPDATE и DELETE. Дополнительные сведения см. в разделе TOP (Transact-SQL).

FROM

Необязательное ключевое слово, которое можно использовать между ключевым словом DELETE и целевым аргументом table\_or\_view\_name или rowset\_function\_limited.

server\_name

Имя сервера (с использованием имени связанного сервера или функции OPENDATASOURCE в качестве имени сервера), на котором расположена таблица или представление. Если аргумент server\_name указывается, аргументы database\_name и schema\_name обязательны.

database\_name

Имя базы данных.

schema\_name

Имя схемы, которой принадлежит таблица или представление.

table\_or view\_name

Имя таблицы или представления, откуда удаляются строки.

Переменную table в пределах ее области действия также можно использовать в качестве источника таблицы в инструкции DELETE.

Представление, на которое ссылается аргумент table\_or\_view\_name , должно быть обновляемым и ссылаться ровно на одну базовую таблицу в предложении FROM данного представления. Дополнительные сведения об обновляемых представлениях см. в разделе CREATE VIEW (Transact-SQL).

rowset\_function\_limited

Функция OPENQUERY или OPENROWSET в зависимости от возможностей поставщика. Дополнительные сведения о возможностях, требуемых для поставщика см. в разделе Требования UPDATE и DELETE для поставщиков OLE DB.

WITH ( <table\_hint\_limited> [... n] )

Задает одну или несколько табличных подсказок, разрешенных для целевой таблицы. Ключевое слово WITH и круглые скобки обязательны. Использование ключевых слов NOLOCK и READUNCOMMITTED запрещено. Дополнительные сведения о табличных подсказках см. в разделе Табличные подсказки (Transact-SQL).

<OUTPUT\_Clause>

Возвращает удаленные строки или выражения, основанные на них, как часть операции DELETE. Предложение OUTPUT не поддерживается ни в каких инструкциях DML, направленных на представления и удаленные таблицы. Дополнительные сведения см. в разделе Предложение OUTPUT (Transact-SQL).

FROM <table\_source>

Задает дополнительное предложение FROM. Это расширение языка Transact-SQL для инструкции DELETE позволяет задавать данные из <table\_source> и удалять соответствующие строки из таблицы в первом предложении FROM.

Это расширение, в котором задается соединение, может быть использовано вместо вложенного запроса в предложении WHERE для указания удаляемых строк.

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

WHERE

Указывает условия, используемые для ограничения числа удаляемых строк. Если предложение WHERE не указывается, инструкция DELETE удаляет все строки из таблицы.

Предусмотрено два вида операций удаления в соответствии с тем, что указывается в предложении WHERE.

Операции удаления с поиском указывают условие поиска для уточнения строк, которые будут удалены. Например, WHERE column\_name = value.

Операции удаления по позиции используют предложение CURRENT OF для указания курсора. Удаление осуществляется в текущей позиции курсора. Эта операция может быть более точной, чем инструкция DELETE по найденному, которая использует предложение WHERE search\_condition для указания удаляемых строк. Инструкция DELETE по найденному удаляет несколько строк, если условие поиска не определяет уникально одну строку.

<search\_condition>

Указывает ограничивающие условия для удаляемых строк. Количество предикатов, которое может содержать условие поиска, не ограничено. Дополнительные сведения см. в разделе Условие поиска (Transact-SQL).

CURRENT OF

Указывает выполнение инструкции DELETE в текущей позиции указанного курсора.

GLOBAL

Указывает, что аргумент cursor\_name ссылается на глобальный курсор.

cursor\_name

Имя открытого курсора, из которого производится выборка. Если существует как глобальный, так и локальный курсор с именем cursor\_name, то, когда указывается GLOBAL, этот аргумент указывает на глобальный курсор; в противном случае — на локальный курсор. Курсор должен позволять производить обновления.

cursor\_variable\_name

Имя переменной курсора. Переменная курсора должна содержать ссылку на курсор, обновления которого разрешены.

OPTION ( <query\_hint> [ ,... n] )

Ключевые слова, показывающие, что подсказки оптимизатора применяются при настройке способа обработки инструкции компонентом Database Engine. Дополнительные сведения см. в разделе Подсказки в запросах (Transact-SQL).

Замечания

Инструкцию DELETE можно использовать в теле пользовательской функции, если изменяемым объектом является переменная table.

При выполнении инструкции DELETE может произойти ошибка, если она нарушает триггер или пытается удалить строку, на которую ссылаются данные в другой таблице с помощью ограничения FOREIGN KEY. Если инструкция DELETE удаляет несколько строк и одна из удаленных строк нарушает триггер или ограничение, то эта инструкция отменяется, т.е. возвращается ошибка и строки не удаляются.

В случае арифметической ошибки (переполнение, деление на ноль или выход за пределы допустимых значений), возникающей в ходе вычисления выражения при выполнении инструкции DELETE, компонент Database Engine будет обрабатывать эти ошибки, как если бы параметр SET ARITHABORT имел значение ON. Оставшаяся часть пакетной операции отменяется и возвращается сообщение об ошибке.

Значение параметра SET ROWCOUNT не учитывается в инструкциях DELETE для удаленных таблиц и локальных и удаленных секционированных представлений.

При необходимости удаления всех строк в таблице следует использовать инструкцию DELETE без предложения WHERE либо использовать инструкцию TRUNCATE TABLE. Инструкция TRUNCATE TABLE выполняется быстрее, чем инструкция DELETE, и использует меньше системных ресурсов и ресурсов журнала транзакций.

Удаление строк из кучи

Когда строки удаляются из кучи, компонент Database Engine может использовать строку или страницу блокировки для операции. В результате пустые страницы, в которых выполняются операции удаления, остаются размещенными для кучи. Если их не освободить, занимаемое ими место не может быть использовано под другие объекты базы данных.

Чтобы удалить из кучи строки и освободить страницы, воспользуйтесь одним из следующих методов.

Задайте подсказку TABLOCK в инструкции DELETE. Это приведет к тому, что при выполнении операции удаления в таблице будет установлена совмещаемая блокировка, а не блокировка строки или страницы, что позволит освободить страницы. Дополнительные сведения о подсказке TABLOCK см. в разделе Табличные подсказки (Transact-SQL).

Если из таблицы удаляются все строки, пользуйтесь инструкцией TRUNCATE TABLE.

Перед удалением строк создайте в куче кластеризованный индекс. Потом его можно будет удалить. Этот метод потребует больше времени и потребляет больше временных ресурсов.

Дополнительные сведения о блокировках см. в разделе Блокировка в компоненте Database Engine.

Использование триггера INSTEAD OF в операциях DELETE

Если триггер INSTEAD OF определен в операциях DELETE над таблицей или представлением, то он выполняется вместо инструкции DELETE. В предыдущих версиях SQL Server имеется поддержка только триггеров AFTER для DELETE и других инструкций изменения данных. Предложение FROM нельзя указывать в инструкции DELETE, ссылающейся (прямо или косвенно) на представление, в котором указан триггер INSTEAD OF. Дополнительные сведения о триггерах INSTEAD OF см. в разделе CREATE TRIGGER (Transact-SQL).

Разрешения

Разрешения DELETE необходимы для целевой таблицы. Разрешения SELECT также необходимы, если инструкция содержит предложение WHERE.

Разрешения DELETE назначаются по умолчанию членам предопределенной роли сервера sysadmin, предопределенных ролей базы данных db\_owner и db\_ddladmin, а также владельцу таблицы. Члены ролей sysadmin, db\_owner и db\_securityadmin, а также владелец таблицы могут передавать разрешения другим пользователям.

Примеры

А. Использование инструкции DELETE с предложением WHERE

Следующий пример удаляет все строки из таблицы SalesPersonQuotaHistory, поскольку не указано предложение WHERE, ограничивающее количество удаляемых строк.

Transact-SQL

USE AdventureWorks2008R2;

GO

DELETE FROM Sales.SalesPersonQuotaHistory;

GO

Б. Использование инструкции DELETE для набора строк

Следующий пример удаляет все строки таблицы ProductCostHistory, у которых значение в столбце StandardCost больше чем 1000.00.

Transact-SQL

USE AdventureWorks2008R2;

GO

DELETE FROM Production.ProductCostHistory

WHERE StandardCost > 1000.00;

GO

В. Использование инструкции DELETE для текущей строки курсора

Следующий пример удаляет одну строку из таблицы EmployeePayHistory с помощью курсора complex\_cursor. Операция удаления затрагивает только одну строку, выбранную в данный момент курсором.

Transact-SQL

USE AdventureWorks2008R2;

GO

DECLARE complex\_cursor CURSOR FOR

SELECT a.BusinessEntityID

FROM HumanResources.EmployeePayHistory AS a

WHERE RateChangeDate <>

(SELECT MAX(RateChangeDate)

FROM HumanResources.EmployeePayHistory AS b

WHERE a.BusinessEntityID = b.BusinessEntityID) ;

OPEN complex\_cursor;

FETCH FROM complex\_cursor;

DELETE FROM HumanResources.EmployeePayHistory

WHERE CURRENT OF complex\_cursor;

CLOSE complex\_cursor;

DEALLOCATE complex\_cursor;

GO

Г. Использование инструкции DELETE на основе вложенного запроса и с помощью расширения языка Transact-SQL

Следующий пример показывает использование расширения языка Transact-SQL для удаления записей из базовой таблицы, основанной на соединяющих или взаимосвязанных вложенных запросах. В первой инструкции DELETE показано основанное на вложенных запросах и совместимое с ISO решение, а во второй инструкции DELETE используется расширение языка Transact-SQL. Оба запроса удаляют строки из таблицы SalesPersonQuotaHistory, основанной на сведениях о продажах за текущий год, хранящихся в таблице SalesPerson.

Transact-SQL

-- SQL-2003 Standard subquery

USE AdventureWorks2008R2;

GO

DELETE FROM Sales.SalesPersonQuotaHistory

WHERE BusinessEntityID IN

(SELECT BusinessEntityID

FROM Sales.SalesPerson

WHERE SalesYTD > 2500000.00);

GO

Transact-SQL

-- Transact-SQL extension

USE AdventureWorks2008R2;

GO

DELETE FROM Sales.SalesPersonQuotaHistory

FROM Sales.SalesPersonQuotaHistory AS spqh

INNER JOIN Sales.SalesPerson AS sp

ON spqh.BusinessEntityID = sp.BusinessEntityID

WHERE sp.SalesYTD > 2500000.00;

GO

Д. Использование инструкции DELETE с предложением TOP

Следующий пример удаляет 2.5 процента строк (27 строк) из таблицы ProductInventory.

Transact-SQL

USE AdventureWorks2008R2;

GO

DELETE TOP (2.5) PERCENT

FROM Production.ProductInventory;

GO

Е. Использование инструкции DELETE с предложением OUTPUT

Следующий пример демонстрирует способы сохранения результатов инструкции DELETE в табличную переменную.

Transact-SQL

USE AdventureWorks2008R2;

GO

DELETE Sales.ShoppingCartItem

OUTPUT DELETED.\*

WHERE ShoppingCartID = 20621;

--Verify the rows in the table matching the WHERE clause have been deleted.

SELECT COUNT(\*) AS [Rows in Table] FROM Sales.ShoppingCartItem WHERE ShoppingCartID = 20621;

GO

Ж. Использование предложения OUTPUT с аргументом from\_table\_name в инструкции DELETE

Следующий пример удаляет строки в таблице ProductProductPhoto на основе критерия поиска, указанного в предложении FROM инструкции DELETE. Предложение OUTPUT возвращает столбцы из таблицы, в которой проводится удаление, DELETED.ProductID, DELETED.ProductPhotoID и столбцы из таблицы Product. Оно используется в предложении FROM для указания удаляемых строк.

Transact-SQL

USE AdventureWorks2008R2;

GO

DECLARE @MyTableVar table (

ProductID int NOT NULL,

ProductName nvarchar(50)NOT NULL,

ProductModelID int NOT NULL,

PhotoID int NOT NULL);

DELETE Production.ProductProductPhoto

OUTPUT DELETED.ProductID,

p.Name,

p.ProductModelID,

DELETED.ProductPhotoID

INTO @MyTableVar

FROM Production.ProductProductPhoto AS ph

JOIN Production.Product as p

ON ph.ProductID = p.ProductID

WHERE p.ProductModelID BETWEEN 120 and 130;

--Display the results of the table variable.

SELECT ProductID, ProductName, ProductModelID, PhotoID

FROM @MyTableVar

ORDER BY ProductModelID;

GO

# Правила целостности данных

Главная особенность SQL-технологий наличие у сервера СУБД специальных средств контроля целостности данных, не зависящих от клиентских программ и привязанных непосредственно к таблицам. Т.е. принципиально не важно, каким образом осуществляется доступ к базе данных: через SQL-консоль, через ODBC-драйвера из приложения Windows, через WWW-connector из Internet-браузера или через DBI-интерфейс Perl. В любом из этих случаев, за контролем целостности данных следит сервер, и при нарушении правил целостности данных сервер известит клиента об ошибке.

К структурам контроля целостности данных относятся ограничители (constraint), которые привязаны к столбцам и триггеры (trigger), которые могут быть привязаны как к столбцам, так и к строкам в таблице.

Ограничители это элементарные проверки или условия, которые выполняются для операций вставки и модификации значения столбца. Если данная проверка не проходит или условие не выполняется, то вставка или модификация отменяется, а в программу клиента передается ошибка.

SQL-серверы, как правило, поддерживают следующие ограничители.

NOT NULL - проверка на непустое значение. NULL - специальное понятие в СУБД, которое означает "пусто". "Пусто" и "0(ноль)" не равны друг другу!

UNIQUE - проверка на уникальность. Вставляемое значение должно быть уникально для данного столбца по всей таблице. Может содержать пустые значения.

PRIMARY KEY - первичный ключ. Значение в столбце считается первичным ключом, если оно непустое и уникально в пределах столбца данной таблицы. Первичный ключ может быть составным и представлять собой комбинацию столбцов. Тогда чтобы считаться первичным ключом, каждое из группы значений не должно быть пустыми и формируемые строки значений первичного ключа должны быть уникальны в пределах таблицы. Первичный ключ - основа для построения индексов по таблице.

SQL-технология позволяет на уровне столбца задавать домены значений, т.е. строго определенные наборы или диапазоны значений, для помещаемых в столбец данных. В частности можно реализовывать ограничения ссылочной целостности (referential integrity constraint) и проверки фиксированного условия. Ограничение ссылочной целостности не позволяет значениям из столбца одной таблицы принимать значения кроме как из присутствующих в столбце другой таблицы. Это делается при помощи ограничителей FOREIGN KEY (внешний ключ) и REFERENCES (указатель ссылки). Таблица, содержащая FOREIGN KEY, считается родительской таблицей. Таблица, содержащая REFERENCES, считается дочерней таблицей. Внешний ключ и указатель ссылки могут находиться в одной таблице, т.е. родительская таблица одновременно является дочерней.

FOREIGN KEY - внешний ключ. Назначает столбец или комбинацию столбцов в текущей (родительской) таблице в качестве внешнего ключа для ссылки из других таблиц.

REFERENCES - указатель ссылки (или родительский ключ). Указывает на столбец (комбинацию столбцов) в родительской таблице, ограничивающую значения в текущей (дочерней) таблице.

Для использования ограничений ссылочной целостности должны выполняться некоторые условия. В частности, родительская и дочерняя таблицы должны находиться в пределах одного аппаратного сервера базы данных, они не могут находиться на различных узлах распределенной базы данных. Столбцы, участвующие в отношении ограничения ссылочной целостности обязаны иметь один и тот же тип данных.

Ограничения ссылочной целостности используются при каскадном удалении, т.е. при удалении записи в родительской таблице удаляются все записи с указанным ключом из дочерних таблиц, и наоборот при запрете удаления/модификации, т.е. при наличии зависимых записей в дочерних таблицах, значение ключа записи в родительской таблице нельзя удалить или модифицировать.

CHECK - проверка фиксированного условия. В данном ограничителе явно указывается условие, которое должно выполняться для вставляемого или модифицируемого значения в столбце. Например: check (user in 'ALEX','JUSTAS') - в столбце user могут содержаться только значения 'ALEX' и 'JUSTAS', попытка вставки значения 'SHTIRLITZ' будет интерпретирована как ошибочная , check (user\_salary between 1000 and 5000) - столбец user\_salary может принимать целочисленные значения в диапазоне от 1000 до 5000 и т.д. При формировании условий с некоторыми ограничениями могут использоваться функции, например check (user = upper(user)), в данном случае имя пользователя должно вводиться только в верхнем регистре. Есть и ограничения, например, CHECK не может содержать подзапросы (SELECT).

Обычно ограничители задаются при создании таблиц. Но в дальнейшем их можно изменять, удалять или временно запрещать при помощи соответствующих команд СУБД.

Триггеры - это сохраненная откомпилированная процедура, которая связана с определенной таблицей. Триггеры, в отличие от ограничителей, могут выполнять сколь угодно сложные манипуляции над данными. Помимо операций модификации и вставки, триггеры могут срабатывать и при удалении данных из таблицы. Можно также задавать порядок срабатывания триггера относительно операции, т.е. выполниться ли триггер перед операцией вставки/модификации/удаления значения из столбца (или всей строки) или непосредственно после такой операции.

Некоторые типовые применения триггеров:

Прозрачный аудит (не зависящий от клиентских программ и невидимый для них) и регистрация событий, связанных с доступом к определенным таблицам или столбцам в таблицах.

Генерация значений в столбцах на основе значений в других столбцах при вставке/модификации строки данных.

Манипуляции над зависимыми таблицами в особенности, если они находятся на других узлах распределенной базы данных, чего нельзя сделать при помощи ограничителей.

В случае необходимости триггеры можно запрещать, а затем разрешать. Запрещение триггеров применяется обычно при массовых загрузках данных в таблицы извне, с целью уменьшения времени загрузки. Понятие триггера как выполнение кода по событию в том же Oracle используется весьма широко. В частности, оно является основным при разработке клиентских программ при помощи SQL\*Forms. Триггеры пишутся на процедурных расширениях SQL.

Обработка данных в многопользовательской СУБД.

Основное требование к многопользовательским СУБД - обеспечение непротиворечивости данных в системе, при сохранении максимальной производительности и конкуренции в доступе к данным для пользователей.

Конкуренция в доступе к данным означает, что каждый из пользователей независим от остальных пользователей в потребности обработки данных. Система, во избежание порчи данных, самостоятельно устанавливает очередность работы с данными для пользователей. В случае необходимости пользователи могут ожидать своей очереди для работы с данными. Одной из главной целей многопользовательской СУБД является максимальное уменьшение этого времени ожидания до такой степени, чтобы оно (в идеале) стало незаметным для пользователя.

Кроме того, сервер СУБД должен предотвращать взаимно разрушающие манипуляции с данными нескольких пользователей при их одновременной работе. Например, если система не предусматривает такую возможность, то менеджеры принимающие заказы от клиентов на поставку товара, и выполняющие их резервирование на складе, могут зарезервировать товара больше чем фактически имеется в наличии. В этом случае обеспечен неприятный разговор с клиентом, заказ которого будет впоследствии отменен.

Более неприятная ситуация возможна в банке: если одновременно исполняется несколько клиентских платежных поручений с одного счета, то при неконтролируемом списании с клиентского счета возможен отрицательный остаток, что недопустимо.

Контроль нужен также в системах резервирования билетов на транспорте, чтобы билет на одно и то же место не был продан разными кассирами разным пассажирам.

Несмотря на различия в реализации, серверы СУБД используют общие способы управления данными и доступом к ним.

Атомарность SQL-выражений при работе с данными.

Под атомарностью выражения понимается неизменность (фиксация во времени) набора данных, с которыми это выражение работает на всем протяжении своего исполнения. Т.е. если мы выполняем оператор UPDATE над определенной таблицей, то состояние таблицы на момент начала выполнения операции фиксируется во времени и не изменяется до конца выполнения оператора. Этот набор данных для текущего выполняемого выражения не может быть изменен другим пользователем или даже другой сессией этого же пользователя, которая пытается выполнить операцию модификации этих же данных в этой же таблице.

Распараллеливание операций.

Типовые операции с таблицей в базе данных состоят из многих однотипных операций, например оператор UPDATE, который модифицирует 5000 строк в таблице, по своей сути состоит из 5000 операций, каждая из которых может быть выполнена независимо. В связи с этим такие операторы очень хорошо распараллеливаются при использовании многопроцессорных систем. Это позволяет выровнять нагрузку в системе между разными процессорами, при том условии что СУБД умеет работать в многопроцессорной конфигурации, и уменьшить время ответа системы.

Обеспечение максимальной производительности.

С целью сокращения времени различных пользователей на манипуляции с данными используется ряд следующих методов. Их работа находится на уровне, скрытом даже от программиста СУБД, но о них стоит упомянуть т.к. они иллюстрируют серьезные различия с xBase-технологией.

Строго говоря, эта информация справедлива лишь в отношении Oracle, но другие СУБД используют подобные принципы.

Процессы, выполняющие чтение блоков данных, никогда не ожидают процессов, выполняющих запись тех же блоков данных.

Процессы, выполняющие запись блоков данных, при отсутствии явных блокировок со стороны пользователя, не ожидают процессов, выполняющих чтение тех же блоков данных.

Процессы, выполняющие запись блоков данных, ожидают другие процессы, выполняющие запись, только в случае если они пытаются выполнить запись данных в одни и те же блоки данных.

Данные приемы позволяют существенно уменьшить время ожидания ответа системы и увеличить ее производительность.

**Транзакции**

Многопользовательские системы широко используют понятие транзакций. Транзакция - это логическая единица работы, которая состоит из одного или нескольких SQL-выражений. Группа выражений отмеченных как транзакция рассматривается как единое и неделимое целое. В случае если в одном из выражений обработки данных происходит ошибка, то транзакция отменяется целиком. Таким образом, система возвращается в состояние предшествующее началу транзакции, обеспечивая при этом физическую и логическую непротиворечивость данных.

Например, мы пытаемся модифицировать таблицу при помощи оператора UPDATE. В одном из столбцов этим оператором устанавливается недопустимое значение с точки зрения правил целостности для этой таблицы. Срабатывание ограничителя приведет к тому, что сервер СУБД не позволит выполнить такую модификацию и известит нас ошибкой, а механизм контроля транзакций вызывает отмену всего выполняемого выражения и производит откат к предыдущему состоянию таблицы, сохраняя, таким образом, целостность и непротиворечивость данных. В данном примере транзакция работает с одним SQL-выражением. В случае если выражений несколько, то откатывается результат работы всех выражений составляющих единую транзакцию.

Чтобы транзакциями можно было пользоваться, в системе должен быть включен режим регистрации транзакций. После этого система сохраняет информацию о предыдущих состояниях объектов в системе в так называемых журналах транзакций. Журналы транзакций - это специальные файлы, управляемые сервером СУБД, в которых записываются все изменения произошедшие с момента начала транзакций для всех транзакций в системе.

Если происходит явное сохранение изменений в системе (по команде COMMIT) или неявное сохранение изменений (по завершению группы SQL-выражений, формирующих транзакцию или по завершению сеанса пользователя), то все изменения произошедшие с момента начала транзакции вносятся в систему, и информация о данной транзакции удаляется из журнала.

Для облегчения управления системой в режиме регистрации транзакций существует возможность задания так называемых промежуточных точек сохранения. Промежуточная точка сохранения по команде SAVEPOINT явно помечает состояние системы и предоставляет возможность восстановления состояния БД на момент ее сохранения по команде ROLLBACK. В данном случае ROLLBACK откатывает систему к указанной точке. Обычно промежуточных точек сохранения для одного пользователя может быть несколько.

В случае если транзакция по каким-то причинам не может быть завершена, то происходит неявный откат. Его причиной, могут быть, например, ошибка при выполнении одного из SQL-выражений, составляющих транзакцию, или обрыв связи с инициатором транзакции. При этом по информации из журнала восстанавливается предыдущее состояние объектов, которые пыталась модифицировать текущая транзакция, после чего информация о транзакции удаляется из журнала. Откат может быть и явным - по команде ROLLBACK.

Данная схема справедлива для Oracle, где транзакция начинается с выполнением первого оператора, прочие сервера могут работать по-другому. Например в Informix DS, транзакция начинается явно, при помощи команды BEGIN WORK.

В SQL-бочке меда есть своя ложка дегтя. Для всех SQL-серверов использующих журнальный режим регистрации транзакций существует проблема, так называемых "длинных" транзакций. Это транзакции, которые затрагивают очень большой объем данных (сопоставимый с количеством свободного места на дисках) и в этом случае журналы регистрации транзакций могут переполниться. Если их рост ничем неограничен, то они могут израсходовать у ОС всю доступную дисковую память, что не есть хорошо, т.к. операционная система и сервер СУБД в этом случае остаются в непредсказуемом состоянии. Если их рост ограничен, то при переполнении журналов СУБД выдает соответствующую ошибку и операция откатывается. Чтобы избежать таких ситуаций программист должен разделить длинную транзакцию на короткие транзакции.

Блокировки.

Для того чтобы пользователи не искажали взаимно используемые данные, сервер СУБД, при многопользовательской работе, использует механизм блокировок. Блокировки по аналогии с базами данных на основе файлов могут быть как разделяемые, так и исключительные. Блокировки могут устанавливаться как на таблицу целиком, так и на строку в таблице. Аналогично в xBase-технологиях: блокировки могут устанавливаться как на xBase-файл, так и на запись в xBase-файле.

Блокировки связаны с транзакциями. Если выполняется отмена транзакции, то снимаются все связанные с этой транзакцией блокировки.

Многие блокировки выполняются неявно для пользователя, они выставляются, например, операторами UPDATE, INSERT. Существуют явные операторы задания блокировок, например, LOCK TABLE или операторы, имеющие клаузы блокировки, например SELECT : FOR UPDATE. Соответственно есть операторы и для снятия блокировок.

Многие SQL-серверы имеют специальные способы обнаружения и предотвращения взаимных блокировок (deadlocks), которые могут занимать ресурсы СУБД на неопределенное время.

Способы, которыми обеспечиваются блокировки, зависят от реализации сервера, и описываются в его документации. Виды блокировок также зависят от используемого сервера. В Informix существуют, т.н. promotional locks, это означает, что если клиентский процесс не может блокировать в исключительном режиме ресурс, то такая блокировка ставится в очередь и ей предоставляется ресурс после снятия текущей исключительной блокировки другого процесса. Oracle7 так не делает, если он не может установить исключительную блокировку в течение указанного сервером времени, то клиент извещается об ошибке.