Transact-SQL справочник для начинающих

1. Описание справочника

Данный справочник будет выглядеть следующее образом, сначала я приведу небольшое оглавление с навигацией, затем начнется сам справочник, по каждому пункту будут комментарии, пояснения и примеры, также, если мы уже подробно рассматривали или использовали где-либо в материалах на нашем сайте тот или иной объект или действие, я, конечно же, буду ставить ссылки, для того чтобы Вы могли посмотреть подробные примеры или как использовать то или иное действие на практике.

Так как охватить абсолютно все просто невозможно, поэтому не удивляйтесь, если Вы что-то здесь не обнаружили. Еще раз повторю, что данный справочник создан для начинающих программистов на Transact-SQL, а также для простых админов, которым периодически требуется выгружать какие-то данные с SQL сервера.

2. База данных

Даже начинающий программист Transact-SQL должен уметь создать базу данных или изменить ее свойства, поэтому прежде чем рассматривать таблицы, представления, функции и все остальное, давайте разберем процесс создания, изменения и удаления базы данных на Transact-SQL.

Создание

Для того чтобы создать базу данных необходимо выполнить следующий запрос:

```
create database test
```

где, test это название базы данных.

Подробней о создание базы данных на SQL сервере мы разговаривали в материале Как создать базу данных в MS Sql 2008

Удаление

Если вам необходимо удалить базу данных, то можете использовать запрос:

```
drop database test
```

Изменение

Для изменений параметров базы данных можно использовать графический интерфейс Management Studio, в котором все параметры подробно описаны, а можно посылать запросы ALTER DATABASE, например, для включения автоматического сжатия базы данных test, используем следующий запрос

```
ALTER DATABASE test SET AUTO_SHRINK ON;
  --А для выключения
ALTER DATABASE test SET AUTO_SHRINK OFF;
```

Надеюсь понятно, ALTER DATABASE команда на изменение, test название изменяемой базы данных, SET команда, указывающая на то, что мы будем изменять параметры базы данных, AUTO_SHRINK непосредственно сам параметр, ON/OFF значение параметра.

3. Типы данных

Самые распространенные и часто используемые

Точные числа

- tinyint - 1 байт
- smallint - 2 байта
- int - 4 байта
- bigint - 8 байт
- numeric и decimal (тип с фиксированной точностью и масштабом)
- money - 8 байт
- smallmoney - 4 байт

Приблизительные числа

- float [(n)] – размер зависит от n (n может быть от 1 до 53, по умолчанию 53)
- real - 4 байта

Дата и время

- date – дата
- time - время
- datetime - дата, включающая время дня с долями секунды в 24-часовом формате.

Символьные строки

- char [(n)] – строка с фиксированной длиной, где n длина строки (от 1 до 8000). Размер при хранении составляет n байт.
- varchar [(n | max)] - строка с фиксированной длиной, где n длина строки (от 1 до 8000). Если указать max, то, максимальный размер при хранении составит 2^31-1 байт (2 ГБ), а при указании n то фактическая длина введенных данных плюс 2 байта.
- text – строковые данные переменной длины, максимальный размер 2 147 483 647 байт (2 ГБ).
- nchar [(n)] - строка с фиксированной длиной в Юникоде, где n длина строки (от 1 до 4000). Размер при хранении составляет удвоенное значение n в байтах
- nvarchar [(n | max)] - строка с фиксированной длиной в Юникоде, где n длина строки (от 1 до 4000). При указании max, максимальный размер при хранении составит 2^31-1 байт (2 ГБ), а если n, то удвоенная фактическая длина введенных данных плюс 2 байта.
- ntext - строковые данные переменной длины, с максимальной длиной строки 1 073 741 823 байт.

Двоичные данные

- binary [(n)] - двоичные данные с фиксированной длиной, размером n байт, где n значение от 1 до 8000. Размер при хранении составляет n байт.
- varbinary [(n | max)] - двоичные данные с переменной длиной, где n может иметь значение от 1 до 8000. Если указать max то максимальный размер при хранении составит 2^31-1 байт(2 ГБ). При указании n то размер хранения это фактическая длина введенных данных плюс 2 байта.
- image - двоичные данные переменной длины, размером от 0 до 2^31 – 1 (2 147 483 647) байт.

Другие

- xml –хранение xml данных. Подробно рассматривали в материале Transact-sql – работа с xml, а если Вы вообще не знаете что такое XML, то об это мы разговаривали в статье Основы XML для начинающих.
- table – хранение результирующего набора строк.

4. Таблицы

Примеров создания таблиц на этом сайте достаточно, так как практически в каждой статье связанной с SQL я привожу пример создания тестовой таблицы, но для закрепления знаний, давайте создадим, модифицируем и удалим тестовую таблицу. Как раз посмотрим на то, как задаются типы данных полей в таблицах на Transact-SQL.

Создание

```
CREATE TABLE test_table(
  [id] [int] IDENTITY(1,1) NOT NULL,--идентификатор, целое число int, не разрешены значения NULL
  [fio] [varchar](50) NULL, --ФИО, строка длиной 50 символов, значения NULL разрешены
  [summa] [float] NULL, --сумма, приблизительное числовое значение,  значения NULL разрешены
  [date_create] [datetime] NULL, --дата и время, значения NULL разрешены
  [comment] [varchar](100) NULL --строка длиной 100 символов, значения NULL разрешены
) ON [PRIMARY]
GO
```

Добавление колонки

```
ALTER TABLE test_table add  prosto_pole  numeric(18, 0) NULL
```

Где,

- test_table - это название таблицы;
- add - команда на добавление;
- prosto_pole – название колонки;
- pole numeric(18, 0) – тип данных новой колонки;
- NULL – параметр означающий что в данном поле можно хранить значение NULL.

Изменение типа данных

Давайте изменим, тип данных нового поля, которое мы только что создали (prosto_pole) с numeric(18, 0) на bigint и увеличим длину поля comment до 300 символов.

```
ALTER TABLE test_table ALTER COLUMN prosto_pole bigint;
ALTER TABLE test_table ALTER COLUMN comment varchar(300);
```

Примечание! *SQL сервер не сможет выполнить изменение типа данных если преобразование значений в этих полях невозможно, в этом случае придется удалять колонку, со всем данными, и добавлять заново или очищать все данные в этом поле.*

Удаление колонки

Для удаления определенной колонки используем команду drop, например, для удаления поля prosto_pole используем следующий запрос

```
ALTER TABLE test_table drop COLUMN prosto_pole
```

Удаление таблицы

Для того чтобы удалить таблицу напишем вот такой простой запрос, где test_table
и есть таблица для удаления

```
drop TABLE test_table
```

5. Представления

Очень полезным объектом в базе данных является представление (VIEW) или по-нашему просто вьюха. Если кто не знает, то представление, это своего рода хранимый запрос, к которому можно обращаться также как и к таблице. Давайте создадим представление на основе тестовой таблицы test_table, и допустим, что очень часто нам требуется писать запрос, например по условию сумма больше 1000, поэтому для того чтобы каждый раз не писать этот запрос мы один раз напишем представление, и впоследствии будем обращаться уже к нему.

Создание

```
CREATE VIEW test_view
AS
        SELECT id, fio,  comment from test_table
                WHERE  summa > 1000
GO
```

Пример обращения к представлению:

```
Select * from test_view
```

Изменение

```
ALTER VIEW test_view
AS
        SELECT id, fio,  comment from test_table
                WHERE  summa > 1500
GO
```

Удаление

```
DROP VIEW test_view
```

6. Системные представления

В СУБД MS SQL Server есть таки системные объекты, которые могут предоставить иногда достаточно полезную информацию, например системные представления. Сейчас мы разберем парочку таких представлений. Обращаться к ним можно также как и к обычным представлениям (например, select * from *название представление*)

- sys.all_objects – содержит все объекты базы данных, включая такие параметры как: название, тип, дата создания и другие.
- sys.all_columns – возвращает все колонки таблиц с подробными их характеристиками.
- sys.all_views – возвращает все представления базы данных.
- sys.tables – все таблицы базы данных.
- sys.triggers – все триггеры базы данных.
- sys.databases – все базы данных на сервере.
- sys.sysprocesses – активные процессы, сессии в базе данных.

Их на самом деле очень много, поэтому все разобрать, не получиться. Если Вы хотите посмотреть, как их можно использовать на практике, то это мы уже делали, например, в материалах Как узнать активные сеансы пользователей в MS Sql 2008

7. Функции

MS SQL сервер позволяет создавать функции которые будут возвращать определенные данные, другими словами пользователь сам может написать функцию и в дальнейшем ее использовать, например, когда необходимо получить значение требующие сложных вычислений или сложную выборку данных, иногда просто для уменьшения кода, когда вызов функции заменят часто требующиеся значения в разных запросах и приложениях.

Создание

```
CREATE FUNCTION test_function
(@par1 bigint, @par2 float)
RETURNS varchar(300)
AS
BEGIN
  DECLARE @rezult varchar(300)
     select @rezult=comment from test_table
                          where id =  @par1 and summa > @par2
  RETURN @rezult
END
```

Где,

- CREATE FUNCTION – команда на создание объекта функция;
- test_function – название новой функции;
- @par1 и @par2 – входящие параметры;
- RETURNS varchar(300) – тип возвращаемого результата;
- DECLARE @rezult varchar(300) – объявление переменной с типом varchar(300);
- Инструкция select в нашем случае и есть действия функции;
- RETURN @rezult – возвращаем результат;
- BEGIN и END – соответственно начала и конец кода функции.

Пример использования ее в запросе:

```
  select test_function(1, 20)
```

Изменение

```
ALTER FUNCTION test_function
(@par1 bigint, @par2 float)
RETURNS varchar(300)
AS
BEGIN
  DECLARE @rezult varchar(300)
     select @rezult=comment from test_table_new
                          where id =  @par1 and summa >= @par2
  RETURN @rezult
END
```

Удаление

```
  DROP FUNCTION test_function
```

8. Встроенные функции

Помимо того, что SQL сервер позволяет создавать пользовательские функции, он также предоставляет возможность использовать встроенные функции, которые за Вас уже написали разработчики СУБД. Их очень много, поэтому самые распространенные я разбил на группы и попытался их кратко описать.

Системные функции

Здесь я приведу несколько примеров функций, которые возвращают какие-то системные данные

- @@VERSION – возвращает версию SQL сервера;
- @@SERVERNAME – возвращает имя сервера;
- SUSER_NAME() – имя входа пользователя на сервер, другими словами, под каким логином работает тот или иной пользователь;
- user_name() – имя пользователя базы данных;
- @@SERVICENAME – название сервиса СУБД;
- @@IDENTITY - последний вставленный в таблицу идентификатор;
- db_name() - имя текущей базы данных;
- db_id() – идентификатор базы данных.

Агрегатные функции

Функции, которые вычисляют какое-то значение на основе набора (группы) значений. Если при вызове этих функций нужно указать колонку для вывода результата, то необходимо выполнить группировку данных (group by) по данному полю. Подробно данную конструкцию мы рассматривали в статье Transact-SQL группировка данных group by

- avg – возвращает среднее значение;
- count – количество значений;
- max – максимальное значение;
- min – минимальное значение;
- sum – сумма значений.

Пример использования:

```
select count(*) as count,
                SUM(summa) as sum,
                MAX(id) as max,
                MIN(id) as min,
                AVG(summa) as avg
from test_table
```

Строковые функции

Данный вид функций соответственно работает со строками.

Left (*строковое выражение*, *количество символов*) – возвращает указанное число символов строки начиная слева.

```
select LEFT('Пример по работе функции left', 10)
--Результат 'Пример по'
```

Right (*строковое выражение*, *количество символов*) – возвращает указанное число символов строки начиная справа

```
select Right('Пример по работе функции Right', 10)
-- Результат 'кции Right'
```

Len (*строка*) – возвращает длину строки.

```
select len('Пример по работе функции len')
--Результат 28
```

Lower (*строка*) – возвращает строку в которой все символы приведены к нижнему регистру.

```
select lower('Пример по работе функции lower')
--Результат 'пример по работе функции lower'
```

Upper (*строка*) - возвращает строку в которой все символы приведены к верхнему регистру.

```
select Upper('Пример по работе функции Upper')
--Результат 'ПРИМЕР ПО РАБОТЕ ФУНКЦИИ UPPER'
```

Ltrim (*строка*) – возвращает строку в которой все начальные пробелы удалены.

```
select ltrim('     Пример по работе функции ltrim') --Результат 'Пример по работе функции ltrim'
```

Rtrim (*строка*) – возвращает строку в которой все пробелы справа удалены

```
select Rtrim ('     Пример по работе функции Rtrim      ')
-- Результат'     Пример по работе функции Rtrim'
```

Replace (*строка, что ищем, на что заменяем*) – заменяет в строковом выражении все вхождения указанные во втором параметре, символами указанным в третьем параметре.

```
select Replace  ('Пример по работе функции Replace', 'по работе', 'ЗАМЕНА' )
-- Результат 'Пример ЗАМЕНА функции Replace'
```

Replicate (*строка, количество повторений*) – повторяет строку (первый параметр) столько раз, сколько указанно во втором параметре.

```
select Replicate  ('Пример Replicate ', 3 )
-- Результат 'Пример Replicate Пример Replicate Пример Replicate '
```

Reverse (*строка*) – возвращает все в обратном порядке.

```
select Reverse ('Пример по работе функции Reverse')
-- Результат 'esreveR иицкнуф етобар оп ремирП'
```

Space (*число пробелов*) – возвращает строку в виде указанного количества пробелов.

```
select Space(10)
-- Результат '          '
```

Substring (*строка, начальная позиция, сколько символов*) – возвращает строку, длиной в число указанное в третьем параметре, начиная с символа указанного во втором параметре.

```
select Substring('Пример по работе функции Substring', 11, 14)
-- Результат 'работе функции'
```

Про строковые функции мы также разговаривали в материале Сочетание строковых функций на Transact-SQL

Математические функции

Round (*число, точность округления*) – округляет числовое выражение до числа знаков указанного во втором параметре

```
select Round(10.4569, 2)
-- Результат '10.4600'
```

Floor (*число*) – возвращает целое число, округленное в меньшую сторону.

```
select Floor(10.4569)
-- Результат '10'
```

Ceiling (*число*) – возвращает целое число, округленное в большую сторону.

```
select Ceiling (10.4569)
-- Результат '11'
```

Power (*число, степень*) - возвращает число возведенное в степень указанную во втором параметре.

```
select Power(5,2)
-- Результат '25'
```

Square (*число*) – возвращает числовое значение, возведенное в квадрат

```
select Square(5)
-- Результат '25'
```

Abs (*число*) – возвращает абсолютное положительное значение

```
select Abs(-5)
-- Результат '5'
```

Log(*число*) – натуральный логарифм с плавающей запятой.

```
select Log(5)
-- Результат '1,6094379124341'
```

Pi – число пи.

```
select Pi()
-- Результат '3,14159265358979'
```

Rand – возвращает случайное число с плавающей запятой от 0 до 1

```
select rand()
-- Результат '0,713273187517105'
```

Функции даты и времени

Getdate() – возвращает текущую дату и время

```
select Getdate()
-- Результат '2014-10-24 16:36:23.683'
```

Day(*data*) – возвращает день из даты.

```
select Day(Getdate())
-- Результат '24'
```

Month(дата) – возвращает номер месяца из даты.

```
select Month(Getdate())
-- Результат '10'
```

Year (*data*) –возвращает год из даты

```
select year(Getdate())
-- Результат '2014'
```

DATEPART(*раздел даты*, *data*) – возвращает из даты указанный раздел (DD,MM,YYYY и др.)

```
select DATEPART(MM,GETDATE())
-- Результат '10'
```

Isdate (*data*) – проверят, введенное выражение, является ли датой

```
select Isdate(GETDATE())
-- Результат '1'
```

Функции преобразование

Cast (*выражение as тип данных*) – функция для преобразования одного типа в другой. В примере мы преобразуем тип float в int

```
select cast(10.54 as int)
--результат 10
```

Convert – (*тип данных, выражение, формат даты*) – функция для преобразования одного типа данных в другой. Очень часто ее используют для преобразования даты, используя при этом третий необязательный параметр формат даты.

```
select GETDATE(), CONVERT(DATE, GETDATE(), 104)
--Результат
--2014-10-24 15:20:45.270 – без преобразования;
--2014-10-24 после преобразования.
```

9. Табличные функции

Создаются, для того чтобы получать из них данные как из таблиц, но после различного рода вычислений. Подробно о табличных функциях мы разговаривали в материале Transact-sql – Табличные функции и временные таблицы

Создание

```
--название нашей функции
CREATE FUNCTION fun_test_tabl
(
--входящие параметры и их тип
@id int
)
--возвращающее значение, т.е. таблица
RETURNS TABLE
AS
--сразу возвращаем результат
RETURN
(
		--сам запрос или какие то вычисления
		SELECT * from test_table where id = @id
)
GO
```

Изменение

```
--название нашей функции
ALTER FUNCTION fun_test_tabl
(
--входящие параметры и их тип
@id int
)
--возвращающее значение, т.е. таблица
RETURNS TABLE
AS
--сразу возвращаем результат
RETURN
(
		--сам запрос или какие то вычисления
		SELECT * from test_table where id = @id and summa > 100
)
GO
```

Удаление

```
DROP FUNCTION fun_test_tabl
```

Как видите, для того чтобы создать, изменить или удалить такие функции используются такие же операторы, как и для обычных функций, отличия лишь в том какой тип возвращает функция.

Пример обращения к этой функции

```
select * from fun_test_tabl(1)
```

10. Процедуры

Процедуры – это набор SQL инструкций, которые компилируется один раз, и могут принимать, как и функции, различные параметры. Используются для упрощения расчетов, выполнения групповых действий.

Создание

```
CREATE PROCEDURE sp_test_procedure
(@id int)
AS
  --объявляем переменные
  declare @sum float
 --SQL инструкции
  set @sum = 100
  update test_table set summa = summa + @sum where id = @id
GO
```

Изменение

```
ALTER PROCEDURE sp_test_procedure
(@id int)
AS
  --объявляем переменные
  declare @sum float
 --SQL инструкции
  set @sum = 500
  update test_table set summa = summa + @sum where id = @id
GO
```

Удаление

```
  drop PROCEDURE sp_test_procedure
```

Вызов процедуры

Можно вызывать по разному, например:

```
EXECUTE sp_test_procedure 1
  --или
EXEC sp_test_procedure 1
```

Где, EXECUTE и EXEC вызов процедуры, sp_test_procedure соответственно название нашей процедуры, 1 значение параметра

11. Системные процедуры

Системные процедуры – это процедуры для выполнения различных административных действий как над объектами на сервере, так и над конфигурацией самого сервера. Вызываются они также как и обычные процедуры, но в контексте любой базы данных.

Их огромное множество, поэтому приведу всего несколько примеров.

sp_configure – процедура для отображения и внесения изменений в конфигурацию ядра СУБД. Первый параметр название параметра конфигурации, второй параметр значение.

Пример

```
--изменяем значение параметра
exec sp_configure 'Ad Hoc Distributed Queries',1
reconfigure --применяем
exec sp_configure --просто просматриваем значения всех параметров
```

где, 'Ad Hoc Distributed Queries' это название параметра, 1 соответственно значение, на которое мы хотим изменить, reconfigure применят введенное значение.

На практике мы применяли эту процедуру в материале Межбазовый запрос на Transact-SQL

sp_executesql – выполняет инструкцию или набор инструкций Transact-SQL, которые могут формироваться динамически. Данную процедуры мы использовали в материале журналирование изменений данных в таблице на Transact-SQL

Пример

```
EXECUTE sp_executesql   N'SELECT * FROM test_table  WHERE id = @id',  N'@id int',  @id = 1
```

Где, первый параметр sql инструкция (строка в Юникоде), второй определение всех параметров встроенных в sql инструкцию, третий значение параметров.

sp_help – возвращает подробные сведения о любом объекте базы данных.

Пример

```
EXECUTE sp_help 'test_table'
```

sp_rename – переименовывает объект в базе данных. Можно использовать для переименования таблиц, индексов, название колонок в таблицах, Не рекомендуется использовать эту процедуру для переименования пользовательских процедур, триггеров, функций.

Пример переименования таблицы

```
EXEC sp_rename 'test_table', 'test_table_new'
```

где, первым параметром идет объект со старым названием, а второй параметр это новое название объекта.

Пример переименования столбца в таблице

```
EXEC sp_rename 'test_table.summa', 'summa_new', 'COLUMN'
```

где, третьим параметром указывается, что переименовывается колонка.

12. Триггеры

Триггер – это обычная процедура, но вызывается она событием, а не пользователем. Событие, например, может быть вставка новой строки в таблицу (insert), обновление данных в таблице (update) или удаление данных из таблицы (delete).

Создание

```
CREATE TRIGGER trg_test_table_update ON test_table
        for UPDATE --можно также delete, insert
AS
        BEGIN
        --sql инструкции в случае UPDATE
        END
GO
```

Изменение

```
ALTER TRIGGER trg_test_table_update ON test_table
        for insert --можно также delete, update
AS
        BEGIN
        --sql инструкции в случае insert
        END
GO
```

Удаление

```
DROP TRIGGER trg_test_table_update
```

Включение/Отключение

```
--отключение
DISABLE TRIGGER trg_test_table_update ON test_table;
--включение
ENABLE TRIGGER trg_test_table_update ON test_table;
```

Также о триггерах мы разговаривали в статье - Как создать триггер на Transact-SQL.

13. Индексы

Это объект базы данных, который повышает производительность поиска данных, за счет сортировки данных по определенному полю. Если провести аналогию то, например, искать определенную информацию в книге намного легче и быстрей по его оглавлению, чем, если бы этого оглавления не было. В СУБД MS SQL Server существует следующие типы индексов:

Кластеризованный индекс - при таком индексе строки в таблице сортируются с заданным ключом, т.е. указанным полем. Данный тип индексов у таблицы в MS SQL сервере может быть только один и, начиная с MS SQL 2000, он автоматически создается при указании в таблице первичного ключа (PRIMARY KEY).

Некластеризованный индекс – при использовании такого типа индексов в индексе содержатся отсортированные по указанному полю указатели строк, а не сами строки, за счет чего происходит быстрый поиск необходимой строки. Таких индексов у таблицы может быть несколько.

Колоночный индекс (columnstore index) – данный тип индексов основан на технологии хранения данных таблиц не виде строк, а виде столбцов (отсюда и название), у таблицы может быть один columnstore индекс.

При использовании такого типа индексов таблица сразу становится только для чтения, другими словами добавить изменить данные в таблице уже будет нельзя, для этого придется отключать индекс, добавлять/изменять данные, затем включать индекс обратно.

Такие индексы подходят для очень большого набора данных, используемых в хранилищах.

Операции, в которых используются агрегатные функции с использованием группировки, выполняются намного быстрей (в несколько раз!) при наличии такого индекса.

Columnstore index доступен начиная с 2012 версии SQL сервера в редакциях Enterprise, Developer и Evaluation.

Создание

Кластеризованного индекса

```
CREATE CLUSTERED INDEX  idx_clus_one
  ON test_table(id)
Go
```

Где, CREATE CLUSTERED INDEX это инструкция к созданию кластеризованного индекса, idx_clus_one название индекса, test_table(id) соответственно таблица и ключевое поле для сортировки.

Некластеризованного индекса

```
CREATE  INDEX  idx_no_clus
 ON test_table(summa)
Go
```

Columnstore index

```
CREATE  columnstore INDEX  idx_columnstore
 ON test_table(date_create)
 go
```

Отключение

```
--отключение
ALTER INDEX idx_no_clus ON test_table DISABLE
--включение, перестроение
ALTER INDEX idx_no_clus ON test_table REBUILD
```

Удаление

```
DROP INDEX idx_no_clus ON test_table
GO
```

14. Курсоры

Курсор это своего рода тип данных, который используется в основном в процедурах и триггерах. Он представляет собой обычный набор данных, т.е. результат выполнения запроса.

Пример (все это в коде процедуры)

```
--объявляем переменные
DECLARE @id bigint
DECLARE @fio varchar(100)
DECLARE @summa float
--объявляем курсор
DECLARE test_cur CURSOR FOR
		SELECT id, fio, summa FROM  test_table
--открываем курсор
OPEN test_cur
--считываем данные первой строки в курсоре
--и записываем их в переменные
FETCH NEXT FROM test_cur INTO @id, @fio, @summa
--запускаем цикл до тех пор, пока не закончатся строки в курсоре
WHILE @@FETCH_STATUS = 0
BEGIN
		--на каждую итерацию цикла можем выполнять sql инструкции
		--.....................SQL инструкции.....................
		--считываем следующую строку курсора
		FETCH NEXT FROM test_cur INTO @id, @fio, @summa
END
--закрываем курсор
CLOSE test_cur
DEALLOCATE test_cur
```

Подробно о курсорах мы разговаривали в материале Использование курсоров и циклов в Transact-SQL

15. Запросы DML

DML (*Data Manipulation Language*) – это операторы SQL с помощью которых осуществляется манипуляция данными. К ним относятся select, update, insert, delete.

Select

Оператор SQL с помощью которого осуществляется выборка данных. Подробно о нем мы разговаривали в материале Язык запросов SQL – Оператор SELECT

Пример

```
select * from test_table
```

Update

Используется для обновления данных

Пример

```
--обновятся все строки в таблице
update test_table set summa=500
--обновятся только строки, у которых id больше 10
update test_table set summa=100
		where id > 10
```

Insert

Оператор на добавление данных

```
--добавление одной строки
insert into test_table (fio, summa, date_create, comment)
		values ('ФИО',100, '26.10.2014', 'тестовая запись')
--массовое добавление на основе запроса
insert into test_table
		select fio, summa, date_create, comment
				from test_table
```

Detele

С помощью этого оператора можно удалить данные.

Пример

```
--очищение всей таблицы
delete test_table
--удаление только строк попавших под условие
delete test_table
		where summa > 100
```

16. Табличные подсказки

Табличные подсказки переопределяют поведение оптимизатора запросов по умолчанию на время выполнения инструкции языка обработки данных (DML) указанием способа блокировки, одного или более индексов, операции обработки запроса, например просмотра таблицы или поиска в индексе, или других параметров. Табличные подсказки указываются в предложении FROM инструкции DML и относятся к таблицам и представлениям, указанным в этом предложении.

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.

Синтаксис

WITH (<table_hint> [[,]...n])

Аргументы

WITH **(** <табличная_подсказка> **)** [[**,**]...*n*]

За некоторыми исключениями, табличные подсказки поддерживаются только при указании с ключевым словом WITH. Указание круглых скобок обязательно.

Важно!
Отсутствие ключевого слова WITH является устаревшей возможностью и будет исключено в следующих версиях Microsoft SQL Server. Всегда указывайте ключевое слово WITH при разработке новых программ, внесите изменения в приложения, где оно отсутствует в настоящий момент.
Разделение подсказок пробелами (а не запятыми) является устаревшей возможностью и будет исключено в следующих версиях Microsoft SQL Server. Всегда указывайте запятые при разработке новых программ и внесите изменения в приложения, где они отсутствуют в настоящий момент.

Следующие табличные подсказки можно указывать как с ключевым словом WITH, так и без него: NOLOCK, READUNCOMMITTED, UPDLOCK, REPEATABLEREAD, SERIALIZABLE, READCOMMITTED, FASTFIRSTROW, TABLOCK, TABLOCKX, PAGLOCK, ROWLOCK, NOWAIT, READPAST, XLOCK и NOEXPAND. Если такие табличные подсказки указываются без ключевого слова WITH, подсказки следует задавать отдельно. Например, FROM t WITH (TABLOCK). Если эта подсказка указана с другим параметром, то необходимо указать подсказку с ключевым словом WITH. Например, FROM t WITH (TABLOCK, INDEX(myindex)).
Ограничения применяются в случаях, когда подсказки используются в запросах к базам данных с уровнем совместимости 90 и более.

NOEXPAND

Указывает, что при обработке запроса оптимизатором запросов никакие индексированные представления не расширяются для доступа к базовым таблицам. Оптимизатор запросов обрабатывает представление так же, как и таблицу с кластеризованным индексом. Аргумент NOEXPAND применяется только для индексированных представлений. Дополнительные сведения см. в разделе «Примечания».

INDEX (*index_value* [,... *n*]) | INDEX = (*index_value*)

Синтаксис INDEX(index_value) указывает имя или идентификатор одного или нескольких индексов, используемых при обработке инструкции оптимизатором запросов. Альтернативный синтаксис INDEX = (index_value) допустим только для единичного значения индекса.
Если имеется кластеризованный индекс, аргумент INDEX(0) приводит к просмотру кластеризованного индекса, а INDEX(1) — к просмотру или поиску по кластеризованному индексу. Если кластеризованного индекса нет, аргумент INDEX(0) приводит к просмотру таблицы, а INDEX(1) интерпретируется как ошибка.
Если в отдельном списке подсказок используется несколько индексов, повторяющиеся индексы пропускаются, а остальные используются для получения строк из таблицы. Порядок индексов в подсказке индекса имеет значение. Несколько подсказок индекса также принудительно выполняют операции И с индексами, и оптимизатор запросов применяет столько условий, сколько возможно для каждого из индексов, к которым он получает доступ. Если

коллекция индексов с подсказками не включает все указанные в запросе столбцы, выборка для получения остальных столбцов выполняется после того, как компонентом SQL Server Database Engine будут получены все индексированные столбцы.

Примечание
Если подсказка индекса, ссылающаяся на несколько индексов, используется в таблице фактов в соединении типа «звезда», оптимизатор не учитывает индекс и возвращает предупреждение. Кроме того, выполнение операции ИЛИ с индексами также не разрешено для таблицы с указанной подсказкой индекса.

Максимальное число индексов, которое может быть указано в табличной подсказке, составляет 250 некластеризованных индексов.

NOLOCK

Равнозначен аргументу READUNCOMMITTED. Дополнительные сведения об аргументе READUNCOMMITTED см. далее в этом разделе.

Примечание
Для инструкций UPDATE и DELETE. В будущей версии Microsoft SQL Server эта возможность будет удалена. Избегайте использования этой возможности в новых разработках и запланируйте изменение существующих приложений, в которых она применяется.

NOWAIT

Указывает компоненту Database Engine вернуть сообщение сразу после наложения блокировки на таблицу. Аргумент NOWAIT равнозначен указанию SET LOCK_TIMEOUT 0 для конкретной таблицы.

READUNCOMMITTED

Указывает, что чтение недействительных результатов разрешено. Для предотвращения ситуаций, когда другие транзакции изменяют данные, считанные текущей транзакцией, не накладываются совмещаемые блокировки, а монопольные блокировки других транзакций не мешают текущей транзакции считывать заблокированные данные. Разрешение чтения измененных результатов может привести к повышению параллелизма за счет считывания изменений данных, откат которых произведен другими транзакциями. Это в свою очередь может сопровождаться ошибками транзакции, представлением пользователю незафиксированных данных, повторным появлением некоторых записей или их отсутствием. Дополнительные сведения о «грязных» чтениях, операциях чтения без возможности повторения и фиктивных операциях чтения см. в разделе Эффекты параллелизма.

Подсказки READUNCOMMITTED и NOLOCK применяются только к блокировкам данных. Все запросы, включая запросы с подсказками READUNCOMMITTED и NOLOCK, получают блокировку Sch-S (стабильность схемы) в процессе компиляции и выполнения. Поэтому запросы блокируются, если параллельная транзакция удерживает в таблице блокировку Sch-M (изменение схемы). Например, операция языка DDL получает блокировку Sch-M, прежде чем изменяет данные схемы. Все параллельные запросы, включая выполняемые с подсказками READUNCOMMITTED или NOLOCK, блокируются при попытке получить блокировку Sch-S. И наоборот, запрос, удерживающий блокировку Sch-S, блокирует параллельную транзакцию, которая пытается получить блокировку Sch-M. Дополнительные сведения о работе блокировок см. в разделе Совместимость блокировок (компонент Database Engine).

Подсказки READUNCOMMITTED и NOLOCK для таблиц, измененных операциями вставки, обновления или удаления, указать нельзя. Оптимизатор запросов SQL Server не учитывает подсказки READUNCOMMITTED и NOLOCK в предложении FROM, применяемые к целевой таблице инструкции UPDATE или DELETE.

Примечание
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.

Минимизировать конфликты блокировок во время защиты транзакций от «грязных» чтений незафиксированных изменений данных можно следующими способами.

- Уровень изоляции READ COMMITTED с параметром базы данных READ_COMMITTED_SNAPSHOT, установленным в значение ON.
- Уровень изоляции моментального снимка (SNAPSHOT).

Примечание
Если выдается сообщение об ошибке 601 при заданном параметре READUNCOMMITTED, ее следует разрешить так же, как и ошибку взаимоблокировки (1205), и затем повторить инструкцию.