Transact SQL

Структурированный язык запросов(SQL) - это язык, разработанный корпорацией IBM в 1970. Он фактически стал стандартом, как язык реляционных баз данных. Диалект SQL, который используется в MS SQL Server называется Transact SQL(PL/SQL - Oracle, SPL - Informix).

Transact SQL добавляет к базовому SQL ключевые слова, позволяющие производить формирование инструкций выборки, сохранения данных и манипуляции над ними. При разработке диалекта SQL, применяемого в MS SQL Server, корпорация Microsoft добавила к нему дополнительные возможности и расширения, что делают многие разработчики СУБД.

Создание таблиц базы данных и использование типов данных

Таблицы - это хранящиеся внутри базы данных каталоги взаимосвязанной информации. В более ранних системах таблицы являлись эквивалентом физических файлов. Каждая таблица представляла собой отдельный файл. Например, в базе данных dBase файл базы данных на диске соответствует одной таблице.

Столбцы состоят из блоков информации, имеющей индивидуальные признаки, и традиционно называются полями.

Строки - это набор полей, еще их называют записями.

Таблица создается с помощью команды

CREATE TABLE [[имя_базы_данных.]владелец.]<имя таблицы>

(<имя столбца> < тип данных> [NOT NULL| NULL] IDENTITY [(начальное значение, инкремент)] [условие], <имя колонки>, <тип данных>...);

После ключевых слов **CREATE TABLE** вводится имя таблицы. Внутри круглых скобок записывается имя и тип столбца. Можно(но необязательно) указать базу данных, в которой создается таблица, а также владельца таблицы. Если при создании таблицы не указан ее владелец, то владельцем таблицы будет текущий пользователь. Часто таблицы создаются с применением бюджета системного администратора. Это делается для того, чтобы ограничить последующий доступ к таблицам. Владельцу базы данных, в которых определена таблицы, автоматически предоставляется право **CREATE TABLE**, позволяющее создание таблиц.

Главным в определении таблицы является указание типов данных для ее столбцов. Язык Transact SQL позволяет определить различные типы данных, включая и те, которые предназначены для хранения символов, чисел и битовых образов.

Числовые целые типы данных

int(integer) - 4 байта, -2^31 ... 2^31(1 бит для знака числа)

smallint - 2 байта

tinyint - 1 байт

Пример

CREATE TABLE number_example(

int1 int,

int2 smallint,

int3 tinyint);

INSERT INTO number_example VALUES (4000000, 32767, 255)

SELECT * FROM number_example

int 1	int2	int3
4000000	32767	255

!!! Контролируется диапазон значений для каждого типа данных

Числовые типы данных с плавающей точкой

real - 4 байта

float([n]) - 8 байт

decimal(p,s)

numeric(p,s)

Символьные типы данных

char(n) - до 255 символов

varchar(n) - для хранения строк переменной длины, не превышающих 255 символов

Пример

CREATE TABLE string_example(

char1 CHAR(5),

char2 VARCHAR(5));

INSERT INTO string_example VALUES ('AB', 'CD')

SELECT * FROM string_example

char1	char2
AB _ _ _	CD

Типы данных datetime и smalldatetime

Предназначаются для хранения даты и времени, так как хранить дату и время в виде строки символов неудобно.

datetime - позволяет определить для хранения в столбце таблицы дату и время от 1/1/1753 до 12/31/9999. Выделено 8 байт. SQL сервер использует первые 4 байта для хранения числа дней после или перед базовой датой - 1 января 1990 года. Отрицательное значение представляет собой количествоа дней до базовой даты, а положительное - после. Время хранится во вторых четырех битах, как число миллисекунд после полуночи. Для отображения значений, хранящихся в виде данных типа datetime, по умолчанию используется формат, например Oct 22 1999 11:14 PM. При употреблении значений типа datetime в инструкции **INSERT** или любой другой их надо заключить в одинарные кавычки. Допускается ввести сначала дату, а потом время, или наоборот. SQL отличит одно значение от другого.

Формат ввода даты определяется с помощью команды **set dateformat**:

Sep 23 1953

6/27/79

6.24.79

Время должно вводиться в следующем формате: часы, минуты, секунды и миллисекунды. Разделитель :

smalldatetime - 4 байта

Можно хранить дату и время начиная от 1/1/1900 до 6/6/2079. Позволяет получить двойную экономию памяти

Пример

```
CREATE TABLE date_table(

date1 DATETIME,

date2 SMALLDATETIME

);

INSERT INTO date_table VALUES ( 'Jan 1 1953', 'Jan 1 1990')

SELECT * FROM date_table
```

date1	date2
Jan 1 1953 12:00 AM	Jan 1 1990 12:00 AM

Специальные типы данных

bit - данные хранятся в одном бите.

timestamp - часто используется для уникальной идентификации строки. Когда обновляются поля в строке, существует общая практика, упоминать столбец, определенный как timestamp, в предложении where инструкции **UPDATE**. В этом случае гарантируется, что инструкция **UPDATE** обновит только одну строку таблицы. Значение типа timestamp уникально, поскольку сервер поддерживает и обновляет его всякий раз, когда вставляется новая или обновляется существующая строка.

binary(n) - двоичный тип данных. Используется для хранения битовых образов размером не более 255 байтов.

varbinary(n) - ограничен в отличие от binary только реальной длиной значения.

Типы данных text и image

Эти типы данных используются при необходимости хранить большое количество символьной или двоичной информации.

text - от 1 до 2 миллиардов байтов текстовой информации

image - используется для хранения больших битовых образов размером от 1 до 2 миллиардов.

money - 8 байт. предназначен доя хранения денежных единиц в виде целых и дробных частей.

smallmoney - 4 байта

Пример

```
CREATE TABLE imagetext_table(

image1 IMAGE,

text1 TEXT

);

INSERT INTO imagetext_table VALUES('123456789abc=', '789abc+/')
```

Null и Not Null

Кроме указанных типов данных для столбцов таблицы, для каждого из них можно указать характеристику Null или Not Null.

Характеристика столбца таблицы Null позволяет игнорировать ввод значений в этот столбец. Если определена характеристика not Null, то SQL сервер не позволит при добавлении строки игнорировать этот столбец. Null это не то же самое, что и 0, или пробел, или символ Null кода ASCII.

Пример

```
CREATE TABLE NULLtable(

x int NULL,

y CHAR(10) NULL

);

INSERT INTO NULLtable VALUES (NULL, NULL)

SELECT * FROM NULLtable
```

x	y
NULL	NULL

Свойство Identity

Можно определить столбец со свойством **IDENTITY**. При этом формируется начальное значение, которое автоматически добавляется в столбец для первой введенной строки, так и инкремент, который добавляется в столбец для каждой введенной строки(происходит автоматически).

Пример

CREATE TABLE identity_table(

name CHAR(15),

row_number integer identity(1,1)

);

INSERT INTO identity_table (name) VALUES ('Bob Smith')

INSERT INTO identity_table (name) VALUES ('Mary Jones')

SELECT * FROM identity_table

name	row_number
Bob Smith	**1**
Mary Jones	**2**

Условие на значение столбца Primary Key

Пример

CREATE TABLE employees(

name CHAR(20),

department VARCHAR(20),

badge INTEGER,

constraint badge employees primary key clustered(badge)

);

Проверочное условие на значение столбца

Пример

CREATE TABLE employees(

name CHAR(20),

department VARCHAR(20),

badge INTEGER CHECK valid_department(department in ('Seles', 'Field Service', 'Software')

)

Добавление данных в таблицу с помощью инструкции INSERT

После того, как таблица создана, с помощью инструкции INSERT в нее можно добавить данные.

Формат оператора:

INSERT INTO имя_таблицы (имя_столбца_1, ... , имя_столбца_n) VALUES (значение_1, ... , значение_n)

Не обязательно указывать имена столбцов и их значения в том же порядке, в котором они определены в таблице. Эти значения вводятся в порядке расположения имен столбцов предыдущей строки.

Пример

INSERT INTO empoloyees (name, department, badge) VALUES ('Bob Smith', 'SALESS', 1834)

Выборка данных из базы

Для задания информации, которая должна быть возвращена из базы данных, используются различные составляющие оператора **SELECT**

Пример

SELECT * FROM employee

С помощью предложения from можно задать несколько таблиц, как это сделано в примере

Пример

SELECT * FROM employee, pay

SELECT * FROM company.dbo.employee

SELECT name, badge FROM employee

Для задания нужных строк таблицы используется ключевое слово **WHERE**, которое позволяет сформировать дополнительно предложение инструкции **SELECT**

Пример

SELECT * FROM employee WHERE department = "sales"

Использование операторов сравнения в предложении **WHERE: =, ! = , <>, <, >, > =, < =, LIKE**

Пример

SELECT * FROM employee WHERE department <> "sales"

SELECT * FROM employee WHERE badge <5000

Использование оператора сравнения LIKE

При формировании запроса с помощью ключевого слова **LIKE** вместо полного значения столбца указывают фрагмент значения. Маска % - означает любые символы.

Пример

SELECT * FROM employee WHERE department LIKE "sa%"

Логические операторы: OR, AND, NOT

Примеры

SELECT * FROM employee WHERE department = "field service" or department = "logistics"

SELECT * FROM employee WHERE name = "Bob Smith" and badge = 1834

SELECT * FROM employee WHERE not department = "Field Service"

Оператор BETWEEN

Пример

SELECT * FROM employee WHERE badge BETWEEN 2000 and 7000

Оператор IN

Пример

SELECT * FROM employee WHERE badge IN (3211, 6732, 4411)

Предложение ORDER BY

Необходимый порядок выборки и отображения строк можно задать с помощью предложения **ORDER BY** как часть инструкции **SELECT**

Пример

SELECT * FROM employee ORDER BY department

SELECT * FROM employee ORDER BY department, badge desc

Если после имени столбца указать имя еще одного столбца, то по значениям второго столбца будут упорядочены строки, содержащие одинаковые значения в первом столбце. desc после имени второго столбца добавлено для изменения порядка сортировки строк второго столбца по убыванию.

Использование ключевого слова DISTINCT для выборки уникального значения столбцов

В таблице БД можно разрешить или запретить повторяющиеся строки. Повторяющиеся строки хранятся в таблице до тех пор, пока для нее не определено какое-либо ограничивающее условие на значение столбца, например уникальный ключ. Хотя повторяющиеся строки можно запретить, но в некоторых столбцах останутся одинаковые значения.

Пример

SELECT DISTINCT department FROM employee

Предложение GROUP BY

Данное предложение делит таблицу на группы строк. Строки, имеющие одно и то же значение заданного столбца, помещены в одну группу, и эта операция выполнена над всеми строками указанного столбца.

Пример

SELECT department, "headcount" = count(*) FROM employee GROUP BY department

Строки с одинаковыми названиями отделов сначала объединяются в группы, но при этом для каждой группы отображается только одна строка. Во второй строке выводится количество строк в каждой группе после выполнения группировки.

Использование подзапросов

Законченную инструкцию **SELECT** можно встроить внутрь другой инструкции **SELECT(подзапрос)**

Пример

SELECT * FROM employee WHERE department = (SELECT department FROM employee5 WHERE name = "Bob Smith")