Основные понятия языка Transact-SQL. Соглашения о синтаксисе Transact-SQL. Многокомпонентные имена. Идентификатор. Константы и переменные. Типы данных. Команды присваивания, ветвления, циклов.

Transact-SQL структурированный язык запросов, являющийся инструментальным средством для описания, управления и манипулирования реляционными данными в СУБД Microsoft SQL Server. Первый структурированный язык запросов SEQUEL (Structured English Query Language) для доступа к реляционным данным был разработан фирмой IBM в 1974 году. В 1992 году был разработан первый международный стандарт SQL-92.

В начале 1970-х гг. компания IBM разработала язык программирования SEQUEL (Structured English QUEry Language — структурированный английский язык запросов), который использовался в ее фирменной СУРБД под названием System R. Позже из-за споров, связанных с торговой маркой, имя языка было из- менено на SQL. В 1986 г. SQL стал стандартом Американского национального института стандартов (ANSI), а через год прошел стандартизацию Международной организации по стандартизации (ISO). С тех пор раз в несколько лет ANSI и ISO выпускают новые версии языка. За это время свет увидели следующие стандарты: SQL-86 (1986), SQL-89 (1989), SQL-92 (1992), SQL:1999 (1999), SQL:2003 (2003), SQL:2006 (2006), SQL:2008 (2008) и SQL:2011 (2011).

SQL напоминает английский язык и имеет логическую структуру. В отличие от других языков программирования, которые используют императивный под- ход, в SQL применяется декларативный стиль. Другими словами, с помощью SQL описывается, что и как вы хотите получить, а все действия по обработке запроса выполняет СУРБД.

SQL сочетает элементы языков DDL (Data Definition Language — язык описания данных), DML (Data Manipulation Language — язык управления данными) и DCL (Data Control Language — язык управления БД). DDL описывает объекты и содержит инструкции CREATE, ALTER, DROP и т. д. Язык DML, который Язык DML, который позволяет запрашивать и редактировать данные, включает операторы SELECT, INSERT, UPDATE, DELETE, TRUNCATE и MERGE.

T-SQL — диалект языка SQL, который предоставляет некоторые фирменные расширения. При знакомстве с каждым элементом синтаксиса я буду отдельно от- мечать, стандартный он или нет.

В архитектуре «клиент – сервер» язык SQL занимает очень важное место. Именно он используется как язык общения клиентского программного обеспечения с серверной СУБД, расположенной на удаленном компьютере. Так, клиент посылает серверу запрос на языке SQL, а сервер разбирает его, интерпретирует, выбирает план выполнения, выполняет запрос и отсылает клиенту результат.

История возникновения и стандарты языка SQL

История возникновения языка SQL восходит к 1970 году, когда доктор **Е.Ф. Кодд предложил реляционную модель в качестве новой модели базы данных**. Для доказательства жизнеспособности новой модели данных внутри компании IBM был создан мощный исследовательский **проект, получивший название System/R**. Проект включал разработку собственно реляционной СУБД и специального языка запросов к базе данных. Так в начале 70-х годов появился первый исследовательский прототип реляционной СУБД. Для этого прототипа разрабатывались и опробовались разные языки запросов, один из которых получил название **SEQUEL** (Structured English Query Language). С момента создания и до наших дней этот язык претерпел массу изменений, но идеология и произношение названия остались неизменными (***аббревиатура SQL иногда читается «Эс-Кю-Эль», а иногда «Сиквел»).***

Период с 1979 года (окончание проекта System/R) до настоящего времени характеризуется развитием и совершенствованием языка SQL и его постоянно увеличивающейся ролью в индустрии, связанной с созданием и эксплуатацией баз данных.

Когда ведут речь о стандартах в области, связанной с разработкой программного обеспечения, обычно подразумевают две организации: ANSI (American National Standards Institute) – Американский национальный институт стандартов;
ISO (International Standards Organization) – Международную организацию по стандартизации.

Работа над официальным стандартом языка SQL началась в 1982 году в рамках комитета ANSI. В 1986 году был утвержден первый вариант стандарта ANSI, а в 1987 году этот стандарт был утвержден и ISO. В 1989 году стандарт претерпел незначительные изменения, но именно этот вариант получил название SQL-1 или SQL-89.

В чем особенность SQL-89? За время разработки стандарта (1982– 1989 гг.) были созданы, представлены на рынке и активно использовались несколько различных СУБД, в которых в том или ином виде был реализован некоторый диалект языка SQL. С учетом того, что разработкой стандартов занимались те же люди, кто внедрял SQL в СУБД, стандарт SQL-89 представлял собой плод множества компромиссов, приведших к наличию в нем большого количества «белых пятен», т.е. мест, которые не были описаны, а отданы на усмотрение разработчиков диалекта. В результате чуть ли не все имеющиеся диалекты стали совместимыми со стандартом, но особой пользы это не принесло.

Следующая реализация стандарта была призвана решить эту проблему. В результате длительных обсуждений и согласований **в 1992 году был принят новый стандарт ANSI SQL-2 или SQL-92. SQL-92 заполнил многие «белые пятна», впервые добавив в стандарт возможности, еще не реализованные в существующих коммерческих СУБД**.

Работа над стандартизацией продолжается и сейчас. Конечно, SQL-92 не решил всех проблем, связанных с наличием нескольких диалектов языка. Одно только описание стандарта разрослось от ста страниц (SQL-89) до 600 (SQL-92). Кроме того, все разработчики как игнорировали, так и игнорируют некоторые положения стандарта, с одной стороны, отказываясь реализовывать некоторые его части и, с другой стороны, реализуя то, что отсутствует в стандарте. Однако не все так плохо, как может показаться. Несмотря на имеющиеся отличия, все коммерческие СУБД поддерживают некоторое ядро языка, описанное в стандарте SQL-92, одинаково. Отличий не очень много, они не носят слишком принципиального характера

Достоинства языка SQL

Для ознакомления с достоинствами языка обратимся к соответствующей литературе. Вот некоторые из них:

- межплатформенная переносимость;
- наличие стандартов;
- одобрение и поддержка компанией IBM (СУБД DB2);
- поддержка со стороны компании Microsoft (СУБД SQL Server, протокол ODBC и технология ADO);
- реляционная основа;
- высокоуровневая структура;
- возможность выполнения специальных интерактивных запросов;
- обеспечение программного доступа к базам данных;
- возможность различного представления данных;
- полноценность как языка, предназначенного для работы с базами данных;
- возможность динамического определения данных;
- поддержка архитектуры клиент/сервер;
- поддержка корпоративных приложений;
- расширяемость и поддержка объектно-ориентированных технологий;
- возможность доступа к данным в Интернете;
- интеграция с языком Java (протокол JDBC);
- промышленная инфраструктура.

Язык SQL (Structured Query Language – структурированный язык запросов) применяется для общения пользователя с реляционной базой данных язык SQL? может иметь специфичную реализацию (свой диалект) и состоит из трех частей:

DDL (Data Definition Language) – язык определения данных. Предназначен для создания базы данных (таблиц, индексов и т.д.) и редактирования ее схемы.

Основные представители данного класса:

CREATE - создание объектов,

ALTER - изменение объектов,

DROP - удаление объектов.

DML (Data Manipulation Language) – язык обработки данных. Содержит операторы для внесения изменений в содержимое таблиц базы данных.

Язык DML позволяет осуществлять манипуляции с данными таблиц, т.е. с ее строками. Он позволяет делать выборку данных из таблиц, добавлять новые данные в таблицы, а так же обновлять и удалять существующие данные, который содержит следующие конструкции:

- SELECT – выборка данных
- INSERT – вставка новых данных
- UPDATE – обновление данных
- DELETE – удаление данных
- MERGE – слияние данных

DCL (Data Control Language) – язык управления данными. Содержит операторы для разграничения доступа пользователей к объектам базы данных. • DCL (Data Control Language)- предназначен для назначения прав на объекты базы данных. Основные представители данного класса: GRANT - разрешение на объект, DENY - запрет на объект, REVOKE - отмена разрешений и запретов на объект.

Как видно из написанного выше, SQL решает все рассмотренные ранее вопросы, предоставляя пользователю достаточно простой и понятный механизм доступа к данным, не связанный с конструированием алгоритма и его описанием на языке программирования высокого уровня. Так, вместо указания того, **как** необходимо действовать, пользователь при помощи операторов SQL объясняет СУБД, **что** ему нужно сделать. Далее СУБД сама анализирует текст запроса и определяет, как именно его выполнять.

- **Язык DDL** (язык описания данных) служит для создания и модификации структуры БД, т.е. для создания/изменения/удаления таблиц и связей.

  DDL (Data Definition Language)- используются для создания объектов в базе данных.

Основные понятия языка Transact-SQL

Идентификатор. Любой объект в базе данных должен быть идентифицирован. Для этого ему присваивают имя. **Идентификаторы** - это специальные символы, которые используются с переменными для идентифицирования их типа или для группировки слов в переменную.

Существуют правила задания идентификатора:

- Длина идентификатора от 1 до 128 символов.
- Идентификатор может включать любые символы, определенные стандартом Unicode Standard 2.0 кроме запрещенных символов.
- В идентификатор не должны входить слова, являющиеся зарезервированными.
- Первым символом в идентификаторе может быть: буква алфавита или символ подчеркивания «_».
- Для обозначения **временных объектов (переменных и параметров)** разрешается использовать в качестве первого символа «@».

- Transact-SQL не различает регистр. Например, идентификатор *Фамилия* соответствует идентификатору *фамилия*.

Типы идентификаторов:

- @ - идентификатор локальной переменной (пользовательской).
- @@ - идентификатор глобальной переменной (встроенной).
- # - идентификатор локальной таблицы или процедур
- ## - идентификатор глобальной таблицы или процедуры.
- [ ] - идентификатор группировки слов в переменную.

Идентификатор, заданный в соответствии с этими правилами называется **обычным идентификатором**. Кроме обычных идентификаторов можно задавать идентификаторы с разделителем.

Примеры обычного идентификатора: Студент, Группа, Курс, Оценка.

Идентификаторы с разделителем заключаются в квадратные скобки ([ ]) или в двойные кавычки (" ") и может при необходимости содержать недопустимый символ или зарезервированные слова.

Примеры идентификатора с разделителем: [Общая ведомость], [Номер группы], [Код специальности].

Именование.

SQL Server достаточно лояльно относиться к именам таблиц.

Можно *использовать в качестве имен русские слова, и даже имена из нескольких слов. Только если имя состоит из нескольких слов, оно должно заключаться в квадратные скобки. Посмотрим на следующий пример:*

```
CREATE TABLE Товары
(
  [Дата] datetime,
  [Название товара] varchar(50),
  [Цена] money,
  [Количество] numeric(10, 2)
)
```

В данном случае все имена объектов имеют русские названия.

С помощью квадратных скобок можно создавать поля и из зарезервированных слов. Например, если попытаться создать таблице ***с именем поля min****, то сервер вернет ошибку, потому что это зарезервированное имя (есть такой оператор). Но если имя поля из одного не зарегистрированного слова заключить в квадратные скобки, то ошибки не будет.*

Квадратными скобками нужно выделять и имена полей из нескольких слов. Если в примере выше убрать квадратные скобки с имени поля "Название товара", то сценарий закончит выполнение ошибкой.

Синтаксические обозначения в Transact-SQL (Transact-SQL)

Transact-SQL.

Обозначение	**Используется для**
Прописные буквы	Ключевые слова Transact-SQL.
Курсив	Пользовательские параметры синтаксиса Transact-SQL.

Обозначение	Используется для
Полужирный	Имена баз данных типов, таблиц, столбцов, индексов, хранимых процедур, программ, типов данных и текст должны вводиться в точном соответствии с примером.
| (вертикальная черта)	Разделяет элементы синтаксиса внутри квадратных или фигурных скобок. Может быть использован только один из этих элементов.
[ ] (квадратные скобки)	Необязательный элемент синтаксиса.
{ } (фигурные скобки)	Обязательные элементы синтаксиса. Фигурные скобки не вводятся.
[**,** *...n*]	Указывает на то, что предшествующий элемент можно повторить *n* раз. Отдельные вхождения элемента разделяются запятыми.
[*...n*]	Указывает на то, что предшествующий элемент можно повторить *n* раз. Отдельные вхождения элемента разделяются пробелами.
;	Признак конца инструкции Transact-SQL. Хотя точка с запятой не требуется для большинства инструкций в этой версии SQL Server, этот символ станет обязательным в будущей версии.
<label> ::=	Имя синтаксического блока. Используйте это соглашение для группирования и маркировки сегментов с длинным синтаксисом или элемента синтаксиса, который может использоваться в нескольких расположениях в пределах одной инструкции. Каждое расположение, в котором может быть использован синтаксический блок, обозначается меткой, заключенной в шевроны: <label>. Набор представляет собой коллекцию выражений, например <grouping set>; а список — коллекцию наборов, например <composite element list>.

Многочастные имена

Если не указано иное, все ссылки Transact-SQL на имена объектов базы данных могут быть четырехсоставными именами, записываемыми в следующей форме.

server_name.[*database_name*].[*schema_name*].*object_name*
| *database_name*.[*schema_name*].*object_name*
| *schema_name.object_name*
| *object_name*

server_name

Указывает имя связанного или удаленного сервера.

database_name

Указывает имя базы данных SQL Server, если объект хранится на локальном экземпляре SQL Server. Когда объект находится на связанном сервере, аргумент *database_name* указывает каталог OLE DB.

schema_name

Если объект находится в базе данных SQL Server, указывает имя схемы, которая содержит объект. Когда объект находится на связанном сервере, аргумент *schema_name* указывает имя схемы OLE DB.

object_name

Ссылается на имя объекта.

При ссылке на конкретный объект нет необходимости всякий раз указывать сервер, базу данных и схему — компонент Компонент SQL Server Database Engine попытается определить этот объект. Однако, если объект не удается найти, возвращается ошибка.

Примечание

Чтобы избежать ошибок разрешения имен, при указании объекта области схемы рекомендуется указать имя схемы.

Чтобы пропустить промежуточные узлы, для обозначения их позиций используйте точки. В следующей таблице показаны допустимые форматы имен объектов.

Формат ссылки на объект	**Описание**
server.database.schema.object	Четырехчастное имя.
server.database..object	Имя схемы пропущено.
server..schema.object	Имя базы данных пропущено.
server...object	Имя базы данных и имя схемы пропущены.
database.schema.object	Имя сервера пропущено.
database.object	Имя сервера и имя схемы пропущены.
schema.object	Имя сервера и имя базы данных пропущены.
Object	Имена сервера, базы данных и схемы пропущены.

Константы и переменные

Константами называются постоянные величины, значения которых не могут быть изменены. Можно использовать константы разных типов:

- числовые – 10, 105.25
- символьные (строковые) – ‘Иванов’, ‘Базы данных’
- десятично-шестнадцатеричные – 0x23657, 0xFF0A12.

Переменная – именованная область памяти определенного объема, которая может изменяться. Переменная используется для хранения и передачи данных и физически является последовательностью нескольких байт. То, как сервер будет воспринимать последовательность байт, принадлежащих переменной зависит от типа данных переменной.

Переменные используются в сценариях и для хранения временных данных. Чтобы работать с переменной, ее нужно объявить, притом объявление должно быть осуществлено в той транзакции, в которой выполняется команда, использующая эту переменную. Иначе говоря, после завершения транзакции, то есть после команды GO, переменная уничтожается.

Тип переменной определяется при ее объявлении. **Объявление переменной выполняется с помощью команды DECLARE, имеющей следующий синтаксис:**

```
DECLARE @имя_переменной тип_данных [,…]
```

С помощью одной команды может быть объявлено несколько переменных. После объявления значение переменной не определено.

Для присвоения значение переменной используются команды **SET и SELECT,** имеющие следующий синтаксис:

```
SET @имя_переменной=значение
SELECT @имя_переменной=значение
```

Командой SET можно присвоить значение только одной переменной:

```
SET @количество=28
SET @группа='1011'
```

Командой SELECT можно присвоить значения нескольким переменным:

```
SELECT @количество=28,@группа='1011'
```

С помощью команды SELECT можно присвоить значение переменной в теле запроса. Например:

```
DECLARE @количество_студентов int
SELECT @количество_студентов=COUNT(фамилия)
FROM студент WHERE номер_группы='1701'
```

Переменные

Объявление переменной выполняется командой DECLARE, задание значения переменной осуществляется либо командой SET, либо SELECT:

```
USE TestDatabase
-- Объявление переменных
DECLARE @EmpID int, @EmpName varchar(40)
-- Задание значения переменной @EmpID
SET @EmpID = 1
-- Задание значения переменной @EmpName
SELECT @EmpName = UserName FROM Users WHERE UserID = @EmpID
-- Вывод переменной @EmpName в результат запроса
SELECT @EmpName AS [Employee Name]
GO
```

Выражение - это совокупность операндов и операторов.

Операндами в выражении могут быть:

- **константы и переменные**;
- **функции** - заранее созданные именованные программы, которые выполняют обработку данных и возвращают определенные значения;
- **имена полей таблиц (столбцов).** Пользователь указывает в выражении имя интересующего столбца, а сервер автоматически подставляет соответствующее значение;
- **подзапрос** – запрос, данные которого используются другим запросом. В результате работы подзапроса сервер создает временную таблицу с необходимой структурой, копирует в нее данные и использует полученный набор в выражении.

Выполняемые над операндами действия задаются с помощью операторов.

Операторы делятся на следующие типы:

- ***Арифметические операции*** (в порядке убывания приоритета выполнения): унарные (+) и (-), умножение (*) и деление (/), остаток от целочисленного деления (%), сложение (+) и вычитание (-).
- ***Строковые операторы*** - +.
- ***Операторы сравнения***: равно (=), больше (>), меньше (<), меньше или равно (<= или !>), больше или равно (>= или !<), не равно (!= или <>).
- ***Логические операторы*** NOT, AND и OR.

Операторы

Операторы - это специальные команды, предназначенные для выполнения простых операций над переменными:

- *Арифметические операторы: "*" - умножить, "/" - делить, "%" - модуль от деления, "+" - сложить , "-" - вычесть, "()" - скобки.*
- *Операторы сравнения: "=" - равно, ">" - больше, "<" - меньше, ">=" - больше или равно, "<=" меньше или равно, "<>" - не равно.*
- *Операторы соединения: "+" - соединение строк.*
- *Логические операторы: "AND" - и, "OR" - или , "NOT" - не.*

Логическими являются также операторы:

- **BETWEEN** - проверяет, лежит ли значение в указанном диапазоне;
- **LINE** – проверяет, соответствует ли значение указанному шаблону;

- **IN** – проверяет, является есть ли значение в указанном списке;
- **ALL** – выполняет сравнение для набора данных. Если условие выполнено для всего набора данных, возвращает значение TRUE;
- **ANY** - – выполняет сравнение для набора данных. Если условие выполнено хотя бы для одного элемента из набора данных, возвращает значение TRUE;
- **EXIST** – проверяет существование данных;
- **SOME** - выполняет сравнение для набора данных. Если условие, выполнено хотя бы для одного элемента из набора данных, возвращает значение TRUE (аналог ANY).

Системные функции

Спецификация Transact-SQl значительно расширяет стандартные возможности SQL благодаря встроенным функциям:

• ***Агрегативные*** *функции- функции, которые работают с коллекциями значений и выдают одно значение.*

Типичные представители: ***AVG*** *- среднее значение колонки, SUM - сумма колонки,* ***MAX*** *- максимальное значение колонки,* ***COUNT*** *- количество элементов колонки.*

• ***Скалярные функции****- это функции, которые возвращают одно значение, работая со скалярными данными или вообще без входных данных. Типичные представители:* ***DATEDIFF*** *- разница между датами,* ***ABS*** *- модуль числа,* ***DB_NAME*** *- имя базы данных,* ***USER_NAME*** *- имя текущего пользователя,* ***LEFT*** *- часть строки слева.*

• ***Функции-указатели****- функции, которые используются как ссылки на другие данные. Типичные представители: OPENXML - указатель на источник данных в виде XML-структуры, OPENQUERY - указатель на источник данных в виде другого запроса.*

Типы данных

Тип данных определяет диапазон значений, которые можно сохранить в переменной или в поле таблицы.

SQL Server поддерживает **встроенные (системные) типы** данных и позволяет на их основе создавать множество **пользовательских типов**. Ниже рассмотрены некоторые из наиболее часто используемых встроенных типов.

Числовые типы:

Числовые типы данных

1) ***Целочисленные типы данных*** (общее название integer):

- **SMALLINT**: хранит числа от –32 768 до 32 767. Занимает 2 байта
- **INT**: хранит числа от –2 147 483 648 до 2 147 483 647. Занимает 4 байта. Наиболее используемый тип для хранения чисел.
- **BIT**: хранит значение от 0 до 16. Может выступать аналогом булевого типа в языках программирования (в этом случае значению true соответствует 1, а значению false - 0). При значениях до 8 (включительно) занимает 1 байт, при значениях от 9 до 16 - 2 байта.
- **TINYINT**: хранит числа от 0 до 255. Занимает 1 байт. Хорошо подходит для хранения небольших чисел.
- **BIGINT**: хранит очень большие числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, которые занимают в памяти 8 байт.

2) ***Нецелочисленные типы данных***

Подразделяются на два типа: ***десятичные*** (decimal) и ***приблизительные*** (approximate).

Десятичные данные хранятся в виде последовательности цифр.

- **DECIMAL**: хранит числа с фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой.

Данный тип может принимать два параметра precision и scale: DECIMAL(precision, scale).

Параметр precision представляет максимальное количество цифр, которые может хранить число. Это значение должно находиться в диапазоне от 1 до 38. По умолчанию оно равно 18.

Параметр scale представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра precision. По умолчанию оно равно 0.

- **NUMERIC**: данный тип аналогичен типу DECIMAL.

Приблизительные типы используются для работы с данными, имеющими значения от очень малых величин до предельно больших. К ним относятся:

- **FLOAT**: хранит числа от –1.79E+308 до 1.79E+308. Занимает от 4 до 8 байт в зависимости от дробной части.

Может иметь форму опредеения в виде FLOAT(n), где n представляет число бит, которые используются для хранения десятичной части числа (мантиссы). По умолчанию n = 53.

- **REAL**: хранит числа от –340E+38 to 3.40E+38. Занимает 4 байта. Эквивалентен типу FLOAT(24).

Денежные типы данных

Используются для хранения данных о денежных суммах. Позволяет хранить после запятой четыре знака. ***К денежным типам относятся***:

- **SMALLMONEY**: хранит дробные значения от -214 748.3648 до 214 748.3647. Предназначено для хранения денежных величин. Занимает 4 байта. Эквивалентен типу DECIMAL(10,4).
- **MONEY**: хранит дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807. Представляет денежные величины и занимает 8 байт. Эквивалентен типу DECIMAL(19,4).

Тип даты и времени

Существует встроенные типы, позволяющие хранить сведения о дате и о времени одновременно, или отдельно. С данными этого типа можно выполнять арифметические операции сложения и вычитания, а также сравнивать их. К временным типам относятся:

- **DATE**: хранит даты от 0001-01-01 (1 января 0001 года) до 9999-12-31 (31 декабря 9999 года). Занимает 3 байта.
- **TIME**: хранит время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт.

Может иметь форму TIME(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.

- **DATETIME**: хранит даты и время от 01/01/1753 до 31/12/9999. Занимает 8 байт.
- **DATETIME2**: хранит даты и время в диапазоне от 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999. Занимает от 6 до 8 байт в зависимости от точности времени.

Может иметь форму DATETIME2(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.

- **SMALLDATETIME**: хранит даты и время в диапазоне от 01/01/1900 до 06/06/2079, то есть ближайшие даты. Занимает от 4 байта.
- **DATETIMEOFFSET**: хранит даты и время в диапазоне от 0001-01-01 до 9999-12-31. Сохраняет детальную информацию о времени с точностью до 100 наносекунд. Занимает 10 байт.

Распространенные форматы дат:

- yyyy-mm-dd - 2017-07-12
- dd/mm/yyyy - 12/07/2017
- mm-dd-yy - 07-12-17

В таком формате двузначные числа от 00 до 49 воспринимаются как даты в диапазоне 2000-2049. А числа от 50 до 99 как диапазон чисел 1950 - 1999.

- Month dd, yyyy - July 12, 2017

Распространенные форматы времени:

- hh:mi - 13:21
- hh:mi am/pm - 1:21 pm
- hh:mi:ss - 1:21:34
- hh:mi:ss:mmm - 1:21:34:12
- hh:mi:ss:nnnnnnn - 1:21:34:1234567

Символьные и текстовые типы данных

Для хранения текстовой информации используются символьные и текстовые типы данных.

К символьным типам данных относятся:

Строковые типы данных

- **CHAR**: хранит строку длиной от 1 до 8 000 символов. На каждый символ выделяет по 1 байту. Не подходит для многих языков, так как хранит символы не в кодировке Unicode. Используется для хранения набора символов длинной n (n_{max} = 8000). Строка данного типа является строкой фиксированной длины. Если строка содержит символов меньше, чем n, то в конец строки добавляются пробелы. Если символов больше n, то часть конечных символов будет потеряна. Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом CHAR(10) будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.
- **VARCHAR**: хранит строку. На каждый символ выделяется 1 байт. Можно указать конкретную длину для столбца - от 1 до 8 000 символов, например, VARCHAR(10). Если строка должна иметь больше 8000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб: VARCHAR(MAX).

Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

В отличие от типа CHAR если в столбец с типом VARCHAR(10) будет сохранена строка в 5 символов, то в столце будет сохранено именно пять символов.

- **NCHAR**: хранит строку в кодировке Unicode длиной от 1 до 4 000 символов. На каждый символ выделяется 2 байта. Например, NCHAR(15)
- **NVARCHAR**: хранит строку в кодировке Unicode. На каждый символ выделяется 2 байта. Можно задать конкретный размер от 1 до 4 000 символов: . Если строка должна иметь больше 4000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб.
- Еще два типа **TEXT** и **NTEXT** являются устаревшими и поэтому их не рекомендуется использовать. Вместо них применяются VARCHAR и NVARCHAR соответственно.

Примеры определения строковых столбцов:

```
1  Email VARCHAR(30),
2  Comment NVARCHAR(MAX)
```

- *Text* – предназначен для хранения очень большого количества символов (до 2^{31}-1). Для данных типа text память выделяется страницами (8 Кбайт).
- *Ntext* – предназначен для хранения текста большого объема в формате Unicode.

Бинарные типы данных

Бинарные типы данных используются для хранения последовательности двоичных значений большой длины***. В бинарных типах данных пользователь может хранить***

любые значения, начиная от текста и заканчивая исполняемым кодом программы. К бинарным типам относятся:

- **BINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт. Под хранимое значение будет выделяться n+4 байта (четыре байта используются для хранения значения n- описание длины).
- **VARBINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт, либо до 2^31–1 байт при использовании значения MAX (VARBINARY(MAX)). Этот тип аналогичен binary, отличается тем, что для хранения значения выделяется ровно столько байт, сколько ввел пользователь плюс 4 байта на описание длины.

Еще один бинарный тип - тип IMAGE является устаревшим, и вместо него рекомендуется применять тип VARBINARY.

- тип *Image.* Позволяет хранить битовые значения длиной до 2^{31}-1 (2 147 483 647). Память выделяется целыми страницами.

Кроме рассмотренных типов данных используемых для представления каких-то определенных значений (дата, текст, числа и т.д.), в SQL-Server имеются специальные типы данных, используемые в основном для внутренних нужд.

Остальные типы данных

- **UNIQUEIDENTIFIER**: уникальный идентификатор GUID (по сути строка с уникальным значением), который занимает 16 байт.
- **TIMESTAMP**: некоторое число, которое хранит номер версии строки в таблице. Занимает 8 байт.
- **CURSOR**: представляет набор строк.
- **HIERARCHYID**: представляет позицию в иерархии.
- **SQL_VARIANT**: может хранить данные любого другого типа данных T-SQL.
- **XML**: хранит документы XML или фрагменты документов XML. Занимает в памяти до 2 Гб.
- **TABLE**: представляет определение таблицы.
- **GEOGRAPHY**: хранит географические данные, такие как широта и долгота.
- **GEOMETRY**: хранит координаты местонахождения на плоскости.

Примечание. Для использования русских символов (не ASCII кодировки) используются типы данных с приставкой "n" (nchar, nvarchar, ntext), которые кодируют символы двумя байтами. Иначе говоря, для работы с Unicode используются типы данных с "n".
Примечание. Для данных переменной длины используются типы данных с приставкой "var". Типы данных без приставки "var" имеют фиксированную длину области памяти, неиспользованная часть которой заполняется пробелами или нулями.

Преобразование типов данных. В SQL-Server имеется две функции, позволяющие преобразовывать значения одного типа в значения любого другого типа (если такое преобразование возможно).

Формат функций:

CAST (значение AS тип_данных)

CONVERT(тип_данных[(длина)], значение)

Аргумент *значение* задает величину, которую необходимо преобразовать.

Аргумент *тип_данных* определяет новый тип данных.

Управляющие конструкции Transact-SQL.

Управляющие конструкции языка предназначены для группировки команд, организации ветвлений и циклов.

BEGIN *...* ***END***

Выполняет объединение команд в единый блок. Объединенные в блок команды воспринимаются интерпретатором как одна команда.

Оператор IF

Условный оператор позволяет выполнять указанную команду или блок команд только при соблюдении указанного логического условия.

Синтаксис оператора:

```
IF логическое_условие
блок_команд_1 [ ELSE
  блок_команд_2 ]
```

Если логическое условие возвращает значение TRUE, выполняется первый блок команд, в противном случае (значение FALSE) –второй блок команд. Если при несоблюдении условия не требуется выполнения действий, то ключевое слово ELSE можно не указывать.

Оператор CASE

Оператор выбора для реализации разветвления по многим направлениям. Синтаксис оператора:

```
CASE
WHEN логическое_условие
THEN блок_команд
[… n]
[ ELSE блок_команд ]
END
```

В операторе может быть указано множество блоков WHEN…THEN, о чем свидетельствует параметр […n]. Если логическое условие блока WHEN…THEN возвращает значение TRUE, то выполняется указанный блок команд и оператор CASE завершает свою работу. Если логическое условие возвращает значение FALSE, то проверяется логическое условие следующего блока. Если ни одно из указанных условий не выполняется то после ключевого слова ELSE можно указать выполняемый в этом случае блок команд.

Операторы WHILE, BREAK, CONTINUE

Оператора WHILE *позволяет организовать цикл.*

Формат оператора: ***WHILE логическое_условие блок_команд [BREAK] блок_команд [CONTINUE***]

блок_команд

Оператор WHILE осуществляет циклическое выполнение блока команд, если указанное логическое условие возвращает значение TRUE.

С помощью оператора BREAK можно принудительно остановить работу и выйти из цикла.

Оператор CONTINUE позволяет начать цикл заново, не дожидаясь выполнения всех команд цикла

Задержка - команда, задерживающая выполнение сценария (WAITFOR)

Вызов ошибки - команда, генерирующая ошибку выполнения сценария (RAISERROR)

В языке SQL можно использовать 2 вида комментариев (однострочный и многострочный):

-- однострочный комментарий и

/* многострочный комментарий */