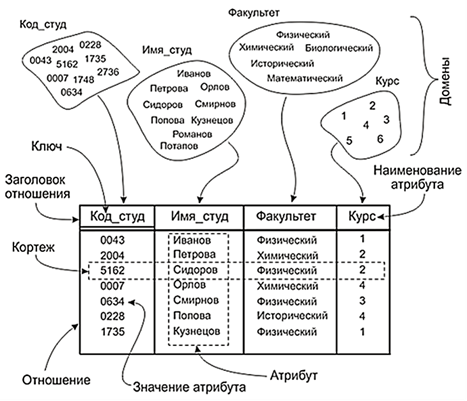
**Модуль 2. Основы взаимодействия с MS SQL Server (2 пары)**

**Цель занятия:** ознакомиться с таблицами и их ограничениями, индексами, рассмотреть системные базы данных и таблицы, изучить классификацию языка SQL.

**План занятия:**

1. Проверка домашнего задания. Актуализация знаний.
2. Таблицы:
   1. первичный ключ;
   2. значение по умолчанию;
   3. уникальность.
3. Типы данных:
   1. целочисленные типы;
   2. типы данных для хранения текста;
   3. вещественные типы данных;
   4. типы для хранения даты и времени;
   5. типы данных с фиксированной точкой;
   6. другие типы данных.
4. Индекс:
   1. что такое индекс?;
   2. цели и задачи индекса;
   3. внутреннее устройство индекса.
5. Системные базы данных и таблицы.
6. Запросы:
   1. введение в язык структурированных запросов SQL;
   2. язык SQL:
      * стандарты языка SQL;
      * диалекты языка SQL;
      * диалект Transact-SQL;
   3. понятия DDL, DML, DCL.
7. Практическое задание.
8. Домашнее задание.
9. **Задание на закрепление темы «Введение в теорию баз данных»**

****

**Пример. Создание базы данных с удалением существующей "старой" базы данных**

USE master;

GO

IF DB\_ID('MyDataBase') IS NOT NULL

DROP DATABASE MyDataBase;

GO

CREATE DATABASE MyDataBase;

GO

**Создание базы данных BestDatabase**

USE master;

GO

IF DB\_ID('MyDataBase') IS NOT NULL

DROP DATABASE MyDataBase;

GO

CREATE DATABASE MyDataBase

ON PRIMARY (NAME = MyDataBase\_dat,

FILENAME = 'C:\Databases\Winner.mdf',

SIZE = 5 MB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 1 MB)

LOG ON (NAME = MyDataBase\_log,

FILENAME = 'C:\Databases\Winner.ldf',

SIZE = 2 MB,

MAXSIZE = 30 MB,

FILEGROWTH = 1 MB);

GO

**Внесение изменений в файловые группы базы данных (для базы данных MultyGroup переименовывается существующая файловая группа MultyGroup2 в MultyGroup0 и добавляется еще одна с именем только что измененной группы MultyGroup2)**

USE master;

GO

ALTER DATABASE MultyGroup

MODIFY FILEGROUP MultyGroup2

NAME = MultyGroup0;

GO

ALTER DATABASE MultyGroup

ADD FILEGROUP MultyGroup2;

GO

1. **Таблицы.**

MS SQL Server 2012 является *реляционной системой управления базами данных*, сокращенно РСУБД. Вкратце можно сказать о реляционных системах то, что все данные в таких базах данных хранятся в *таблицах*. В основе реляционных систем лежит строгий безупречный математический аппарат.

Важной особенностью всех без исключения баз данных является то, что в базе данных помимо самих данных хранятся и описания этих данных — *метаданные*. Это позволяет сильно уменьшить зависимость программ от данных на логическом уровне.

Главным объектом реляционных баз данных являются таблицы.

***Таблица*** - это объект базы данных, который хранит данные в виде совокупности строк и колонок.

Таблица (table) содержит произвольное количество **строк (row)** или, что то же самое, **записей (record)**. На самом деле максимальное количество строк в таблице ограничивается объемом внешней памяти, доступной для хранения данных базы данных. Разумеется, таблица может быть и пустой, т. е. не содержать ни одной строки.

Все строки одной таблицы имеют одинаковую структуру. Они состоят из **столбцов** (column). Столбцы иногда называют **полями** (field). Таблица должна содержать как минимум один столбец. У каждого столбца есть уникальное имя (в пределах таблицы), и все значения в одном столбце имеют один тип (число, текст, дата...).

Чтобы задать таблицу, вы должны решить, сколько колонок она будет иметь и данные каких типов будут в ней храниться. При задании типа данных у колонки задаются следующие атрибуты:

* *Категория (тип) данных*, которые могут содержаться в колонке (например, символьные данные, целые числа или изображения);
* *Размер (длина) данных*, хранимых в колонке;
* *Точность чисел* (этот атрибут применяется только для числовых типов данных), т.е. количество цифр, содержащихся в числах;
* *Масштаб чисел* (этот атрибут применяется только для числовых типов данных), т.е. количество цифр, способных помещаться справа от десятичной точки.

При выборе типа данных для столбца следует отдавать предпочтение типу, который позволит хранить любые возможные для этого столбца значения и занимать при этом минимальное место на диске. Типы данных в MS SQL Server можно разделить на восемь категорий:

1. *Целочисленные данные*

* **bit (1 байт).** Может хранить только значения 0, 1 или null (пустое значение, сообщающее об отсутствии данных). Его удобно использовать в качестве индикатора состояния – включено/выключено, да/нет, истина/ложь.
* **tinyint (1 байт)**.Целые значения от 0 до 255.
* **smallint (2 байта)**. Диапазон значений от -215 (-32768) до 215 (3767).
* **int (4 байта)**. Может содержать целочисленные данные от -231 (-2147483648) до 231 (21474833647).
* **bigint (8 байт)**. Включает в себя данные от -263 (9223372036854775808) до 263 (9223372036854775807). Удобен для хранения очень больших чисел, не помещающихся в типе данных int.

1. *Типы данных для хранения текста – символьные строки*

* **char [(n)].** Содержит символьные не Unicode-данные фиксированной длины от 1 до 8000 знаков. Например, char(10), определяет, что это строка фиксированной длины, количество символов в которой не может превышать 10. Если вы в этом поле сохраните 1 символ, то на жестком диске все равно будет храниться все поле, размером 10 символов. При попытке сохранить строку большего размера, чем 10 символов, будет ошибка.
* **varchar [(n | max)]**. Содержит символьные не Unicode-данные переменной длины от 1 до 8000 знаков. Max указывает, что максимальный размер хранилища равен 2 ^ 31-1 байт (2 ГБ). Размер хранения — это фактическая длина введенных данных плюс 2 байта. varchar(10) обозначает, что это строка переменной длины, максимальное кол-во символов в которой не может превышать 10, но если в этом поле сохранить 1 символ, то на жестком диске будет храниться 1 символ. Если же ввести больше символов, чем 10, то это приведет к ошибке при сохранении данных. Т.о., этот тип данных является более экономным, но это отражается на производительности базы данных.
* **nchar ([n])**. Содержит данные Unicode фиксированной длины от 1 до 4000 символов. Подобно всем типам данных Unicode его удобно использовать для хранения небольших фрагментов текста, которые будут считываться разноязычными клиентами. Размер хранения — два раза n байт. Если кодовая страница параметров сортировки использует двухбайтовые символы, размер хранилища по-прежнему n байт. В зависимости от строки, размер хранилища n байтов может быть меньше, чем значение, указанное для n.
* **nvarchar [(n | Max )].** Содержит данные Unicode переменной длины от 1 до 4000 символов. Max указывает, что максимальный размер хранилища равен 2 ^ 31-1 байт (2 ГБ). Размер хранилища в байтах вдвое больше числа введенных символов + 2 байта.
* **text.** Данные переменной длины не в Юникоде в кодовой странице сервера и с максимальной длиной строки 2^31-1 (2 147 483 647). Если в кодовой странице сервера используются двухбайтовые символы, объем занимаемого типом пространства все равно не превышает 2 147 483 647 байт. Он может быть менее 2 147 483 647 байт — в зависимости от строки символов.
* **ntext**. Данные переменной длины в кодировке Юникод с максимальной длиной строки 2^30 - 1 (1 073 741 823) байт. Размер памяти в байтах вдвое превышает длину введенной строки. Синонимом по стандарту ISO для типа ntext является national text.

1. *Десятичные данные*

* **decimal**. Содержит числа с фиксированной точностью от -1038-1 до 1038-1. Он использует два параметра: точность и степень. Точностью называется общее количество знаков, хранящееся в поле, а степень – это количество знаков справа от десятичной запятой.
* **numeric**. Это синоним типа данных decimal – они идентичны.

1. *Денежные типы данных*

* **money** (8 байт). Содержит денежные значения от -263 до 263 с десятичной точностью от денежной единицы. Удобен для хранения денежных сумм, превышающих 214768,3647.
* **smallmoney** (4 байта). Содержит значения от -214748,3648 до 214748,3647 с десятичной точностью.

1. *Данные с плавающей точкой*

* **float**. Содержит числа с плавающей запятой от -1,79Е+38 до 1,79Е+38.
* **real**. Содержит числа с плавающей запятой от -3,40Е+38 до 3,40Е+38.

1. *Типы данных даты и времени*

* **datetime** (8 байт). Содержит дату и время в диапазоне от 1 января 1753 года до 31 декабря 9999 года с точностью 3,33 мс.
* **smalldatetime** (4 байта). Содержит дату и время, начиная от 1 января 1900 года и заканчивая 6 июнем 2079, с точностью до 1 минуты.

1. *Двоичные типы данных*

* **binary.** Содержит двоичные данные фиксированной длины до 8000 байт.
* **varbinary.** Содержит двоичные данные переменной длины до 8000 байт.
* **image.** Этот тип представляет двоичные данные переменной длины, включающие от 0 до 2^31 – 1 (2 147 483 647) байт.

1. *Специализированные типы данных*

* **sql\_variant**. Используется для хранения значения с различными типами данных.
* **timestamp**. Используется для установки временных меток записей при вставке, которые соответствующим образом обновляются. Удобен для отслеживания изменений в данных.
* **uniqueidentifier**. Глобальный уникальный идентификатор.
* **xml**. Используется для хранения целых документов или фрагментов XML.

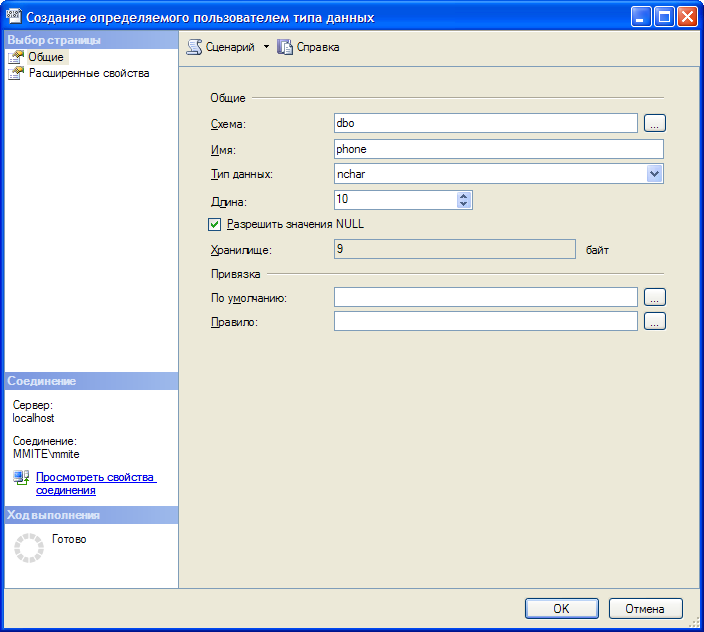
##### Создание пользовательских типов данных

***Пользовательские типы данных*** (типы данных задаваемые пользователями) являются индивидуально настроенными системными типами данных. Такая настройка полезна, когда вы имеете несколько таблиц, в колонках которых должны храниться данные одинаковых типов, а вы хотите генерировать точное соответствие колонок каждой из таблиц по типу, длине и возможности применения null-значений. Создав тип данных с осмысленным именем, вы упростите себе программирование и согласованность данных в ваших таблицах. Например, в вашей базе данных есть таблица, в которой хранятся сведения личного характера о сотрудниках организации и есть другая таблица, хранящая сведения о доходах и расходах сотрудников вашей организации. Чтобы гарантировать, что в данные в колонках обеих таблиц, содержащих сведения о табельных номерах сотрудников будут иметь одинаковый тип данных, можно создать пользовательский тип данных и присвоить его обеим колонкам. При создании типа данных вы должны задать следующую информацию: имя типа данных; системный тип данных, на основании которого создается новый тип данных; возможность хранения null-значения.

В качестве примера создадим тип данных phone, который будет использоваться в таблице Customer для хранения телефонного номера клиента.

Создание пользовательских типов данных с помощью графического интерфейса утилиты Management Studio

1. В дереве обозревателя объектов раскройте папки «Базы данных» (Databases), а затем вашу базу данных. Далее выберите «Программирование - Типы» (Programming – Types). В контекстном меню узла «Определяемые пользователем типы данных» (User Defined Data Types) выберите команду «Создать определяемый пользователем тип данных» (New User Defined Data Types).
2. В появившемся окне в текстовом поле «Имя» (Name) введите phone. В раскрывающемся списке «Тип данных» (Data Type) выберите nchar. В качестве длины введите 10. Отметьте параметр «Разрешить значения null» (Allow Nulls), чтобы иметь возможность не указывать телефонный номер при добавлении нового клиента.
3. В секции «Привязки» оставьте пустые значения и щелкните на кнопке Ok. Созданный пользовательский тип данных должен появиться в дереве обозревателя объектов.



Создание пользовательских типов данных с помощью T-SQL

Cистемная хранимая процедура sp\_addtype является командой T-SQL, применяемой для создания пользовательских типов данных. При помощи запуска этой команды, когда вы работаете с модельной базой данных (model), можно организовать применение нового типа данных во всех вновь создаваемых вами БД. Запуск той команды при работе с пользовательской БД новый тип можно будет использовать только в данной БД.

Use Model

GO

Sp\_addtypekod\_type, "smallint", "Not Null"

GO

***Создание таблиц***

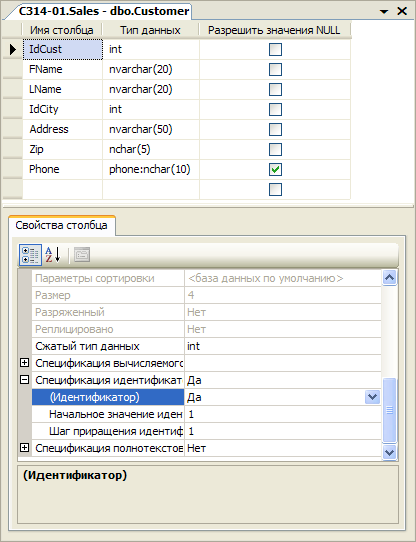
Создание таблиц с помощью графического интерфейса утилиты Management Studio

Создадим в нашей базе данных пять таблиц. Первая таблица, Customer, будет хранить информацию о клиентах, вторая таблица City – справочник городов, третья, Product, - информацию о товарах, четвертая, Order, будет содержать подробную информацию о заказах и пятая, OrdItem, - о составе заказа (перечне товаров входящих в заказ). Ниже представлены все поля этих таблиц и их основные свойства.

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя столбца** | **Тип данных** | **Разрешить null** | **Описание** |
| **Customer** | | | |
| IdCust | int, identity | нет | Уникальный идентификационный номер клиента, на который можно ссылаться в других таблицах |
| FName | nvarchar(20) | нет | Имя клиента |
| LName | nvarchar(20) | нет | Фамилия клиента |
| IdCity | int | нет | Ссылка на номер города |
| Address | nvarchar(50) | нет | Адрес клиента |
| Zip | nchar(5) | нет | Почтовый индекс клиента |
| Phone | phone | да | Телефонный номер клиента |
| **City** | | | |
| IdCity | int, identity | нет | Уникальный идентификационный номер города |
| CityName | nvarchar(20) | нет | Название города |
| **Product** | | | |
| IdProd | int, identity | нет | Уникальный идентификационный номер для каждого товара |
| Description | nvarchar(100) | нет | Короткое текстовое описание товара |
| InStock | int | нет | Количество единиц продукта на складе |
| **Order** | | | |
| IdOrd | int, identity | нет | Уникальный идентификационный номер заказа |
| IdCust | int | нет | Ссылка на номер клиента |
| OrdDate | smalldatetime | нет | Дата и время размещения заказа |
| **OrdItem** | | | |
| IdOrd | int | нет | Ссылка на номер заказа |
| IdProd | int | нет | Ссылка на номер товара |
| Qty | int | нет | Количество единиц товара в заказе |
| Price | money | нет | Цена товара |

Сначала создадим таблицу Customer:

1. В дереве обозревателя объектов в нашей базе данных в контекстном меню узла «Таблицы» (Table) выберите команду «Создать таблицу…» (New Table). В рабочей области должна появиться вкладка с конструктором таблиц. Каждая строка таблицы в открывшейся вкладке обозначает одну колонку таблицы базы данных. Каждая колонка таблицы - какой-либо атрибут колонки таблицы – тип данных, длину или способность хранить null-значения.
2. В первую строку в столбце «Имя столбца» (Column Name) введите IdCust, в столбце «Тип данных» (Data Type) выберите int. Убедитесь что параметр «Разрешить значения null» (Allow Nulls) отключен.
3. В нижней половине экрана в разделе «Свойства столбцов» введите описание поля и измените значение параметра «Спецификация идентификатора / (Идентификатор)» на «Да» для того чтобы значения номера клиента формировались автоматически. Свойство «Идентифицирующий столбец» (Identity), обычно используемое совместно с типом данных int, предназначено для автоматического приращения значения на единицу при добавлении каждой новой записи. К примеру, клиент, добавленный в таблицу первым, будет иметь значение идентификатора 1, вторым – 2, третьим – 3, и т.д.
4. Аналогичным образом введите описания всех остальных полей и закройте окно конструктора таблиц. Введите в качестве имени таблицы Customer. Вновь созданная таблица должна появиться в дереве обозревателя объектов в папке «Таблицы».



**Задание для самостоятельной работы:** В соответствие с вышеприведенным описанием создайте оставшиеся четыре таблицы: City, Product, Order и OrdItem.

Применение операторов T-SQL для создания таблиц

Чтобы создать таблицу БД нужно использовать следующий скрипт:

CREATE TABLE Product\_Info

( ProductJD smallint IDENTITY (1,1) NOT NULL,

Product\_Name char(20) NOT NULL,

Description char(30) NULL,

Price smallmoney NOT NULL,

Brand\_ID kod\_type)

***Создание и использование умолчаний, ограничений и правил***

Умолчания, ограничения и правила *-* это необязательные атрибуты, которые мож­но определять по колонкам и таблицам базы данных. У*молчания* (зна­чения по умолчанию) - это значения, которые заносятся в определенную колонку. *Ограничения* (constraints) используются как способ идентифицирования допустимых значений для колонки (чтобы откло­нять недопустимые значения), а также как средство обеспечения целостности дан­ных в таблицах базы данных и между связанными таблицами. Ограничение только по одной колонке называется ограничением на значение (колонки):оно ограничивает значения только этой колонки. Ограничение, которое влияет на несколько колонок, называется ссылочным ограничением:в этом случае комбинация значений для колонок, указанных в данном ограничении, должна отвечать требованиям этого ограни­чения. Имеется пять типов ограничений: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY и CHECK. Альтернативой использованию ограничения CHECK является объект типа Rule. Rule-объект создается от­дельно от таблицы, но не удаляется при удалении таблицы. Использование ограничения CHECK является предпочтительным методом ограничения значений колонок, а Rule-объекты по­лезно использовать, когда одно правило нужно использовать для нескольких коло­нок или определенных пользователем типов.

Создание умолчаний, ограничений и правил с помощью T-SQL

***Null-значение***(null value) - это неизвестное значение, для которого применяется обозначение NULL*.* Null-значение в колонке обычно означает, что для данной строки этой колонки нет данных, потому что значение неизвестно, либо не имеет смысла, либо не задано или будет задано в будущем. Null-значения - это не пустые значения и не значения числа 0, их настоящие значения неизвестны (unknown), поэтому никакие два null-значения не являются равными.

***Свойство IDENTITY***. Когда вы создаете таблицу, вы можете задать одну из колонок как идентифицирующую колонку (identity column), добавив к определению колонки свойство IDENTITY. Если колонка создается со свойством IDENTITY, то SQL Server автоматически генерирует для этой колонки значение строки, рассчитываемое по начальному значе­нию и значению приращения. Начальное значениеявляется значением идентификации для первой строки, вставленной в таблицу. Приращение - это величина, на которую SQL Server увеличивает значение идентификации для последовательно вводимых строк. Каждый раз при вводе стро­ки SQL Server присваивает текущее значение идентификации элементу данных в колонке идентификации, вводимому в новую строку. Следующая введенная строка получит значение идентификации, большее, чем текущее максимальное значение идентификации на величину приращения. Идентифицирующие колонки обычно применяются в ограничениях первичного ключа в таблицах, благодаря которым возможна уникальная идентификация строк. Например, если вы зададите IDENTITY(1, 10), то значение идентифицирующей колонки для первой введенной строки будет равно 1, для второй строки будет равно 10, для третьей строки - 20, и т.д. Если начальное значение или приращение не задать, то для них будут применяться значения по умолчанию, равные 1 и 1. Идентифицирующие колонки не могут содержать значения по умолчанию и для них не разрешено применение null-значений. В каждой из таблиц может иметься только одна идентифици­рующая колонка.

По умолчанию, непосредственный ввод данных в идентифицирующие колонки невозможен и они не могут быть изменены. Если вы хотите повторить ввод удаленной строки и хотите сохранить старое идентифицирующее значение этой строки, то вы можете преодолеть стандартные настройки, применив такой оператор:

SET IDENTITY INSERT имя.таблицы ON

При помощи этого оператора можно вставить строку и назначить нужное вам зна­чение идентифицирующей колонки. Закончив ввод строки, нужно отменить воз­можность вставки в идентифицирующую колонку при помощи такого оператора:

SET IDENTITYINSERT имя\_таблицы OFF

После этого, SQL Server, в качестве начального значения, применяемого при добавлении следующих строк, возьмет самое большое значение из данной колонки.

***Создание умолчания*** для колонки с помощью оператора CREATE TABLE является предпочтительным, стандартным методом. Следующий оператор создает в базе данных MyDB таблицу, содержащую умолчания для обеих колонок, - columnA (типа char) и columnB (типа int):

CREATE TABLE MyTable

(columnA char(15) NULL DEFAULT "шт",

columnB int NULL DEFAULT 0)

Значение по умолчанию «шт» для колонки columnA совместимо с типом данных char этой колонки, и значение по умолчанию 0 для колонки columnB совместимо с типом данных int. Если при встав­ке новой строки в таблицу не указывается конкретное значение для одной или обеих колонок, то используется соответствующее значение по умолчанию. Поэтому един­ственным способом присваивания этим колонкам значения NULL является явная вставка NULL. Null-значения допустимы, поскольку для обеих колонок указан атрибут NULL. Если бы колонки были определены как NOT NULL, то вы не могли бы выполнять явную вставку значения NULL.

***Ограничение PRIMARY KEY*** используется, чтобы задать первичный ключтаблицы, представляемый колонкой или набором колонок, уникальным образом идентифицирующих строку таблицы. Поскольку первичный ключ идентифицирует строку, соответствующая колонка никогда не содержит значения NULL. Если вы определяете ограничение PRIMARY KEY по набору коло­нок, это ограничение указывает, что комбинация значений этих колонок должна быть уникальной для каждой строки. Ограничение PRIMARY KEY не допускает дублированных значений. Если ограничение PRIMARY KEY присваивается колонке или набору колонок, то по этой колонке или колонкам первичного ключа автоматически создается уникальный индекс. Таблица может иметь только одно ограничение PRIMARY KEY. Колонка с атрибутом IDENTITY хорошо подходит для первичного ключа. Следующий оператор T-SQL представляет один из способов задания колонки SSN как первичного ключа, когда вы определяете таблицу.

CREATE TABLE customer

( first\_name char(20) NOT NULL,

midinit char(1) NULL,

last\_name char(20) NOT NULL,

SSN char(11) PRIMARY KEY,

Cust\_phone char(10) NULL)

Используя альтернативный способ, вы можете присвоить имя этому ограничению, добавив ключевое слово CONSTRAINT. Чтобы присвоить имя PK\_SSN вашему ограничению PRIMARY KEY, используйте следующий оператор:

CREATE TABLE customer

(first\_name char(20) NOT NULL,

midinit char(1) NULL,

last\_name char(20) NOT NULL,

SSN char(11) CONSTRAINT PK\_SSN PRIMARY KEY,

cust\_phone char(10) NULL)

Вы можете также задать ограничение PRIMARY KEY после того, как определены все колонки таблицы. При использовании этого синтаксиса имя колонки должно быть заключено в круглые скобки и указано после предложения CONSTRAINT, как это показано в следующем операторе:

CREATE TABLE customer

(first\_name char(20) NOT NULL,

midinit char(1) NULL,

last\_name char(20) NOT NULL,

cust\_phone char(10) NULL,

CONSTRAINT PK\_SSN PRIMARY KEY (SSN))

***Ограничение FOREIGN KEY*** определяет внешний ключ*,* который задает связь между двумя таблицами. Колонка или колонки внешнего ключа одной таблицы ссылают­ся на потенциальный ключ (одна или несколько колонок) в другой таблице. При вставке строки в таблицу с ограничением FOREIGN KEY значения, которые должны быть внесены в колонку или колонки, определенные как внешний ключ, сравниваются со значениями в потенциальном ключе ссылочной таблицы. Если ни одна из строк ссылочной таблицы не соответствует значениям во внешнем ключе, то встав­ка новой строки не выполняется. Но если значения внешнего ключа, которые нуж­но внести в таблицу, все же имеются в потенциальном ключе другой таблицы, то вставка новой строки будет выполнена. Если значение, которое должно быть занесено в таблицу с ограничением FOREIGN KEY, равно NULL, то это тоже допустимо.

Проверка ограничений FOREIGN KEY происходит также в тех случаях, когда вы хотите обновить какую-либо строку в ссылочной таблице или в таблице с внешним ключом. Вы не сможете обновить какое-либо значение потенциального клю­ча или внешнего ключа, если это приведет к нарушению ограничения. Существует одно исключение из этого правила, когда вы обновляете ссылочную таблицу с по­мощью опции ON UPDATE CASCADE оператора T-SQL CREATE TABLE. Кроме того, ограничения FOREIGN KEY проверяются, если вы хотите уда­лить какую-либо строку из ссылочной таблицы. Вы не сможете удалить строку из ссылочной таблицы, если строка какой-либо таблицы с внешним ключом (таблицы, содержащей ограничение FOREIGN KEY) содержит ссылку на значение в колонке внешнего ключа.

Иными словами, для каждой строки в таблице с внешним ключом должна существовать соответствующая строка в ссылочной таблице, и эту строку нельзя удалить, пока на нее имеется ссылка. Существует также исключение из этого правила: вы можете удалить строку из ссылочной таблицы с помощью опции ON DELETE CASCADE оператора T-SQL CREATE TABLE.

Внешний ключ может ссылаться только на те колонки, которые содержат в ссылочной таблице ограничение PRIMARY KEY или UNIQUE. Если вы попытаетесь создать внешний ключ, который ссылается на колонку, не являющуюся частью одного из этих ограничений, то SQL Server возвратит сообщение об ошибке. Кроме того, тип данных и размер колонки или колонок внешнего ключа должны совпадать со ссылочной колонкой или колонками.

Чтобы получить более ясное представление о внешних ключах, рассмотрим некоторые примеры. Сначала мы создадим таблицу с именем items (товары), которая содержит ограничение PRIMARY KEY по колонке itemid (идентификатор товара), как в следующем операторе:

CREATE TABLE items

(itemname char(15) NOT NULL,

itemid smallint NOT NULL IDENTITY(1,1),

price smallmoney NULL,

item\_desc varchar(30) NOT NULL DEFAULT "none",

CONSTRAINT PK\_itemid PRIMARY KEY (item id))

Затем мы создадим таблицу с именем inventory, содержащую ограничение FOREIGN KEY с именем FK\_itemid, которое ссылается на колонку itemid в таб­лице items, как в следующем операторе:

CREATE TABLE inventory

(store\_id tinyint NOT NULL,

itemid smallint NOT NULL,

item\_quantity tinyint NOT NULL, CONSTRAINT

FK\_itemid FOREIGN KEY (itemid) REFERENCES items(itemid))

***Ограничение UNIQUE*** обеспечивает, что в колонке или наборе колонок не будут допускаться дублированные значения; иными словами, обеспечивается уникальность значений в этой колонке или наборе колонок. Для поддержки этой уникальности SQL Server создает по умолчанию уникальный индекс по колонке или колонкам, указанным в ограничении UNIQUE. Ограничение UNIQUE можно использовать для любой колонки, которая не яв­ляется частью ограничения PRIMARY KEY. Ограничение UNIQUE можно использовать для колонок, в которых разрешены null-значения, в то время как ограничения PRIMARY KEY нельзя использовать для таких колонок. На колонку с ограничением UNIQUE мо­жет ссылаться ограничение FOREIGN KEY. По одной таблице можно задавать несколько ограничений UNIQUE, пока общее число индексов для этой таблицы не превышает 250 индексов.

Чтобы создать ограничение UNIQUE по таблице с помощью T-SQL, используйте оператор CREATE TABLE. Например, следующий оператор создает таблицу customer (покупатель) с ограничением UNIQUE по колонке SSN в виде индекса:

CREATE TABLE customer

(first\_name char(20) NOT NULL,

midinit char( 1) NULL,

last\_name char(20) NOT NULL,

SSN char(11) NOT NULL UNIQUE CLUSTERED,

cust\_phone char(10) NULL)

***Ограничение CHECK*** используется, чтобы ограничить множество допустимых для колонки значений определенными значениями. Значения, которые используются при вставке в колонку или обновлении колонки, проверяются на истинность (значение TRUE) указанного в ограничении булева условия поиска. Например, если бы нам нужно было ограничить диапазон возможных значений, допустимых для колонки price (цена) таблицы items, величинами от $0,01 до $500,00, то мы использовали бы следующий оператор:

CREATE TABLE items

(itemname char(15) NOT NULL,

itemid smallint NOT NULL IDENTITY (1,1),

price smallmoney NULL,

item\_desc varchar(30) NOT NULL DEFAULT ‘none’,

CONSTRAINT PK\_itemid PRIMARY KEY (itemid),

CONSTRAINT CK\_price CHECK (price >=0.01 AND price <= 500.00) )

***Rule-объект***. Привязка Rule-объекта к колонке или к определенному пользователем типу выполняется с помощью системной хранимой процедуры sp\_bindrule. В качестве примера мы создадим Rule-объект, который выполняет ту же функцию, что и ограничение CHECK, которое мы создали выше. Наше правило использует имя переменной ©price для ссылки на колонку price таблицы items. Имя перемен­ной должно начинаться с символа «@», а имя можете выбрать произвольно. Сначала мы создадим данное правило и затем выполним его привязку к колонке, как это показано ниже:

USE MyDB

GO

CREATE RULE price rule AS

(@price >=0.01 AND @price <= 500.00)

GO

sp\_bindrule "price rule", "items.price", "futureonly"

GO

Для отмены привязки правила и его удаления используйте следующий оператор:

sp\_unbindrule "items.price"

GO

DROP RULE price\_rule

GO

1. **Структуры памяти SQL Server**

Прежде чем приступать к рассмотрению понятие индекса, значение и применение данного инструмента, обсудить основные структурные элементы иерархии памяти SQL Server.

Нужно обратить внимание студентов на то, что В SQL Server данные представлены набором структур. Данные структуры должны быть известны и понятны, поскольку с ними приходится сталкиваться при рассмотрении различных понятий СУБД SQL Server в том числе и индексов.

Привести краткое представление иерархии структур данных, которая используется в SQL Server.

1. **База данных – самый высокий порядок иерархии.**

Состоит из физического файла базы данных (.MDF), а также файла журнала (.LDF).

1. **Экстент**

Блок памяти, который используется Server-ом для размещения таблиц и индексов.

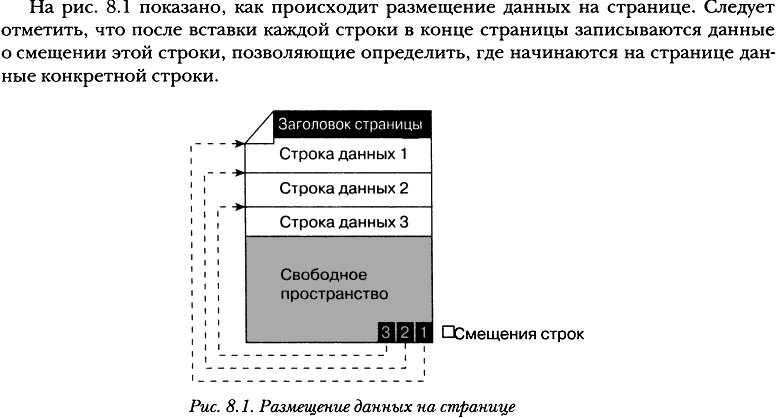
В том случае, если при заполнении экстента потребовалось записать дополнительные строки, сервер выделяет цельный экстент. Преимущество такого подхода заключается в заблаговременном распределении памяти.

1. **Страница**

Блок памяти, который используется Server-ом для размещения объектов в каждом конкретном экстенте. Каждая страница выполнена из 8 страниц.

Является представителем последнего уровня иерархии представляющей общую структуру памяти СУБД SQL Server. На странице размещаются строки таблиц и индексов.

Рисунок представляющий структуру страницы:



Страницы можно разделит на два типа:

Страницы данных – хранятся данные, представленные в таблицах.

Страницы индексов – хранятся данные различных уровней индекса.

1. **Строки**

Отдельный объект в базе данных, содержащий непосредственно информацию. Строки, как элементы структуры иерархии также иногда используются при разработке, например при блокировках.

1. **«Зачем нужны индексы»?**

После рассмотрения иерархии структуры памяти СУБД SQL Server следует объяснить и мотивировать студентов в понимании особенности и важности механизма индексов при построении эффективной и производительной базы данных.

Для этого можно использовать изученные ранее понятия в языках программирования структур данных и различных способов поиска данных.

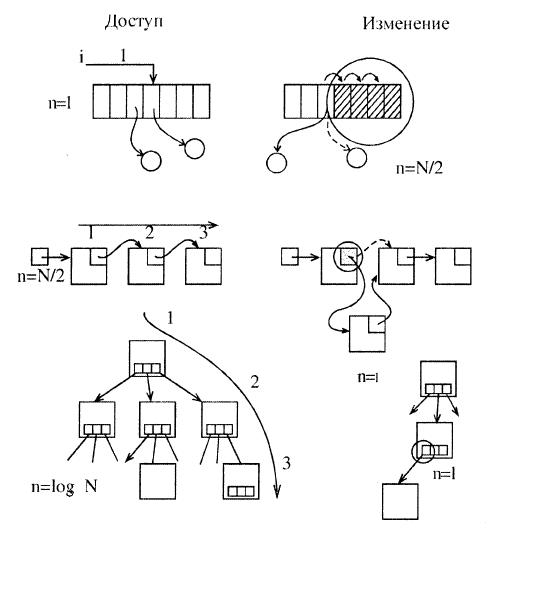
Особенно акцентировать внимание на не эффективных способах поиска (линейный поиск), а также эффективных (бинарный поиск). Если нужно привести примеры (например, на языке С/С++, в зависимости от уровня группы).

Также, вкратце напомнить причины появления таких структур памяти как списки и деревья и их преимущества и недостатки (рис. 1).

Можно провести некоторую аналогию с поиском данных в таблицах СУБД SQL Server, а также таблицах языков С/С++ (например, массивах структур), привести схематические примеры возможных способов упорядочивания элементов в данных структурах и хранения информации о их порядке в дополнительных структурах памяти.

Дать первое определение индексов как одного из способов эффективного поиск информации в базах данных, который основан на физической сортировке табличных данных или создании специальных структур для хранения информации о сортировке по отдельных полям таблицы.

Провести аналогии с уже существующими в быту организациями индексов, например словарях и книгах. Указать их различия в подходах организации поиска (словарь: поиск по букве, поиск по указателю).

рис. 1.

1. **Понятие и архитектура индекса.**

После предварительной подготовки, аргументации и обсуждений важности, структуры можно перейти к рассмотрению понятия индекса и его строения в СУБД.

**Индекс** – список указателей на физическое размещение строк, упорядоченный по значению определенного поля (или группы полей). Используется для ускорения поиска. Нужные строки отыскиваются по индексу, при этом просмотр остальных строк не выполняется.

Рассказать студентам, что данные в таблице сохраняются не упорядочено, это происходит на физическом уровне. Когда данные добавляются, они попадают в конец таблицы, но не всегда можно гарантировать упорядоченность данных, поскольку данные можно также удалять и при этом данная область будет отмечена как удаленная, что может привести к неупорядоченности.

Указать проблемы, которые возникают при выборке данных из таблицы (прямой перебор).

Далее перейти к рассмотрению механизма построение индексов, основанного на B-деревьях. Для этого объяснить (вспомнить из курса С, С++) понятие сбалансированного дерева. Подчеркнуть, что такие деревья имеют большое практическое значение, так как их использование сокращает машинное время, требуемое на выполнение различных алгоритмов.

Говорят, что бинарное дерево идеально сбалансировано, если для каждого его узла количество узлов в левом и правом поддеревьях различается не более чем на 1, т.е. примерно половина элементов расположена справа от узла, а другая половина - слева.

Рассмотреть примеры (иллюстрации) дерева индекса и их видов.

Дерево индекса имеет корневой узел и только один. В этом узле размещаются указатели, ссылающиеся на страницы данных. На страницах в строках располагаются значения, которые размещаются упорядочено. Вместе со значениями также хранятся адреса других страниц для перехода. Страницы, которые не имеют адресов на следующий уровень, называются листовыми. Если же корень сразу ссылается на листья, можно говорить об одноуровневом дереве (рис. 2).

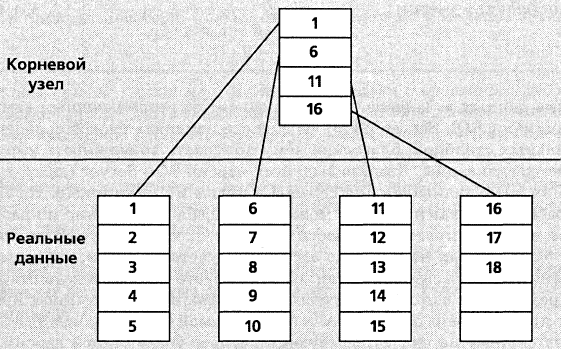


Рис.2

Если же объем информации не может быть представлен одноуровневым деревом, тогда появляются промежуточные уровни, на которых находятся промежуточные узлы (рис. 3).

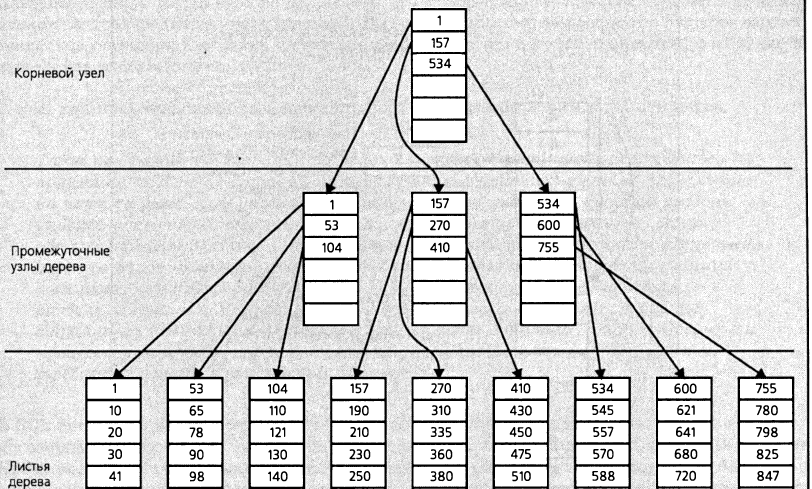


Рис. 3

Проиллюстрировать студентам процесс разбиения страницы при ее переполнении и последствиях (изменения структуры индекса).

При вставке данных, если страница переполняется, она разделяется на 2, в каждую из них переносится половина записей из исходной страницы. Разбиение на нижнем уровне требует записи нового значения на верхнем уровне, что также может вызвать разбиение.

Вывод: все операции изменения данных в поле, по которому выполнено индексировании, требуют затрат времени. Если на таблице задано несколько индексов, все они будут обновляться при изменении данных (рис. 4).

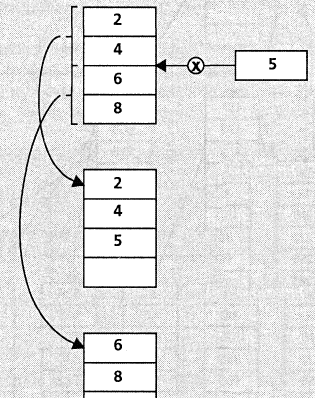


рис. 4

После рассмотрения архитектуры индекса рассмотреть схематические примеры нахождения указанного значения или диапазона значений используя индекс. При рассмотрении примеров подчеркнуть возрастающую эффективность указателей при работе при возрастании объема информации в таблице.

Пример:

Ищем 12: в корневом узле: 1 <= 12 < 157, в промежуточном узле: 1 <= 12 < 53, в листовом узле: 10 <= 12 < 20, переходим на страницу, указатель на которую хранится в паре с числом 10.

1. **Типы индексов.**

После рассмотрения общих характеристик и структуры индексов в СУБД перейти к рассмотрению существующих типов индексов. Рассказать, что в SQL Server существует два основных типа индексов (кластеризованный и некластеризованный), один из которых разделяется еще на два подтипа (некластеризованный индекс на неупорядоченной таблице, некластеризованный индекс на кластерном индексе).

После указания классификации рассмотреть подробней каждый тип и подтипы индексов.

**Кластерный индекс**

Если на таблице задан кластерный индекс, информация в ней хранится в упорядоченном виде.

Отдельные строки идентифицируются при помощи кластерного ключа (cluster key) – столбца, определяющего кластерный ключ.

Если в данном столбце обнаруживаются не уникальные данные, то к ним будет добавлен соответствующий суффикс. Такой подход гарантирует для гарантии уникальность.

Далее нужно указать студентам основные особенности кластерного индекса.

На таблице может быть задан только 1 кластерный индекс. На листовом уровне кластерного индекса находятся данные, т.е. информация на физическом уровне хранится упорядоченно. При добавлении новой записи имеющиеся данные пересортировываются, чтобы сохранить физическую упорядоченность.

При необходимости добавить запись в заполненную страницу происходит разбиение страниц.

Представить студентам графическое представление кластеризированного индекса (рис. 5)

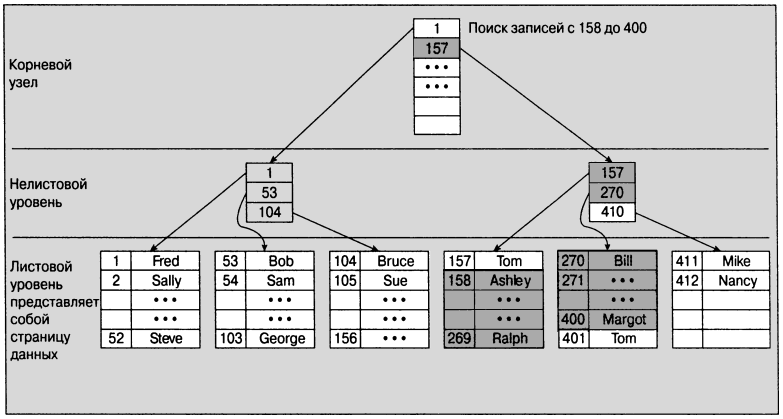


Рис. 5

**Некластеризованный индекс на неупорядоченной таблице**

Рассмотреть со студентами особенности и отличия строения и работы некластеризованного индекса на неупорядоченной таблице, подчеркнув основные отличия данного типа индексов от предыдущего.

Указать, что каждая строка таблицы имеет уникальный идентификатор (RID), включающий: номер экстента, номер страницы, смещение строки.

Отличия заключаются в том, что в листовых узлах индекса хранятся значения RID, по которому можно перейти непосредственно к данным.

Если, например, в выше рассматриваемом кластерном индексе произвести поиск диапазона элементов, то мы можем получить несколько последовательно идущих страниц.

Если же такую операцию произвести с некластеризованным индексом на неупорядоченной таблице, результат может оказаться не на последовательно расположенных страницах, что в свою очередь приведет к затратам производительности.

Продемонстрировать графическое представление заданного индекса (рис. 6)

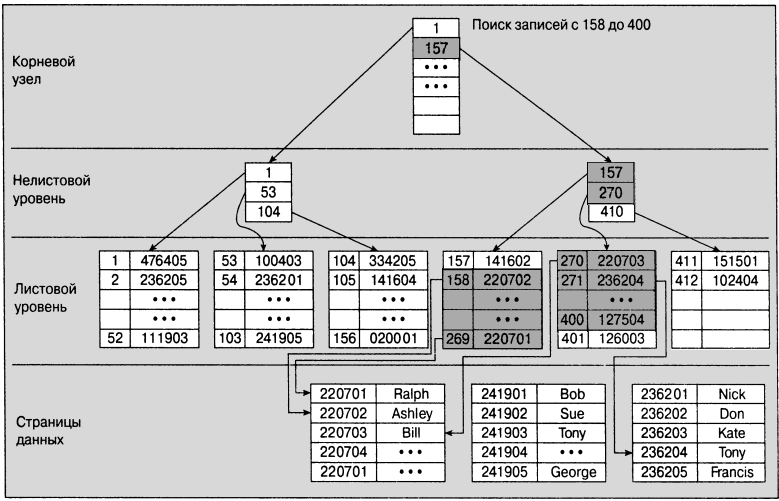


Рис. 6

Также учесть особенности некластеризованного индекса на неупорядоченной таблице, среди них указать, что на таблице может быть несколько таких индексов.

**Некластеризованный индекс на кластерном индексе**

Рассмотреть со студентами особенности и отличия строения и работы некластеризованного индекса на кластерном индексе, подчеркнув основные отличия данного типа индексов, а также аналогии с уже рассмотренными.

Указать основное сходство данных индексов и некластеризованных на неупорядоченной таблице, которое заключается в организации уровней дерева индекса, где промежуточные и листовые уровни отличаются.

Указать основные отличия, которые заключаются в организации листового уровня и перехода от листового уровня к данным через кластерный индекс.

Продемонстрировать графическое представление заданного индекса (рис. 7)

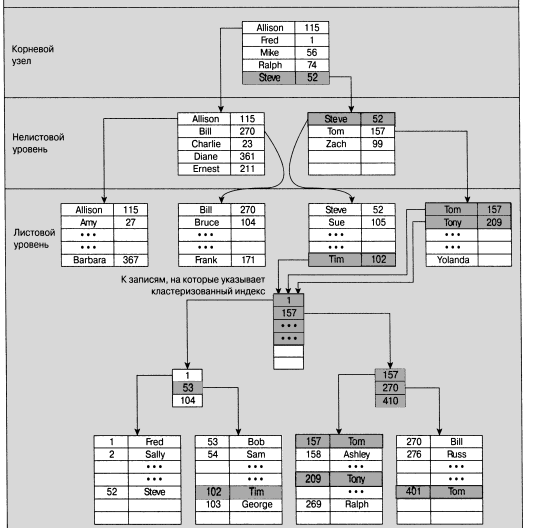


рис. 7

**Индексы** — это первое, что необходимо хорошо понимать в работе *SQL Server*, но странным образом базовые вопросы не слишком часто задаются на форумах и получают не так уж много ответов.

*Роб Шелдон* отвечает на эти, вызывающие смущение в профессиональных кругах, вопросы об индексах в *SQL Server*: одни из них мы просто стесняемся задать, а прежде чем задать другие сначала подумаем дважды.

# Основы индексов в SQL Server

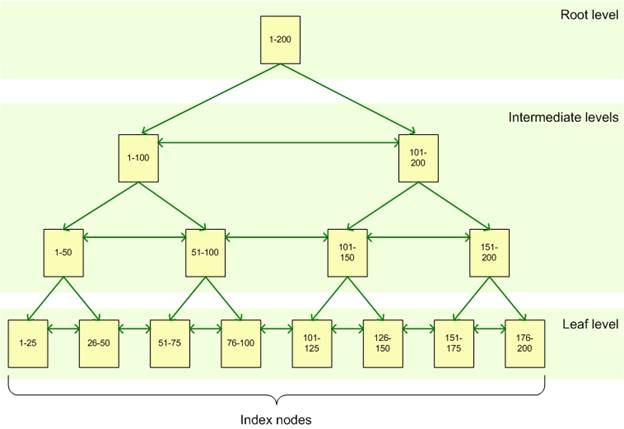
Одним из важнейших путей достижения высокой производительности *SQL Server* является использование индексов. Индекс ускоряет процесс запроса, предоставляя быстрый доступ к строкам данных в таблице, аналогично тому, как указатель в книге помогает вам быстро найти необходимую информацию. В этой статье я приведу краткий обзор индексов в *SQL Server* и объясню как они организованы в базе данных и как они помогают ускорению выполнения запросов к базе данных.

### Структура индекса

Индексы создаются для столбцов таблиц и представлений. Индексы предоставляют путь для быстрого поиска данных на основе значений в этих столбцах. Например, если вы создадите индекс по первичному ключу, а затем будете искать строку с данными, используя значения первичного ключа, то *SQL Server* сначала найдет значение индекса, а затем использует индекс для быстрого нахождения всей строки с данными. Без индекса будет выполнен полный просмотр (сканирование) всех строк таблицы, что может оказать значительное влияние на производительность.

Вы можете создать индекс на большинстве столбцов таблицы или представления. Исключением, преимущественно, являются столбцы с типами данных для хранения больших объектов (*LOB*), таких как *image*, *text*или *varchar(max)*. Вы также можете создать индексы на столбцах, предназначенных для хранения данных в формате *XML*, но эти индексы устроены немного иначе, чем стандартные и их рассмотрение выходит за рамки данной статьи. Также в статье не рассматриваются *columnstore*индексы. Вместо этого я фокусируюсь на тех индексах, которые наиболее часто применяются в базах данных *SQL Server*.

Индекс состоит из набора страниц, узлов индекса, которые организованы в виде древовидной структуры — *сбалансированного дерева*. Эта структура является иерархической по своей природе и начинается с корневого узла на вершине иерархии и конечных узлов, листьев, в нижней части, как показано на рисунке:



Когда вы формируете запрос на индексированный столбец, подсистема запросов начинает идти сверху от корневого узла и постепенно двигается вниз через промежуточные узлы, при этом каждый слой промежуточного уровня содержит более детальную информацию о данных. Подсистема запросов продолжает двигаться по узлам индекса до тех пор, пока не достигнет нижнего уровня с листьями индекса. К примеру, если вы ищете значение 123 в индексированном столбе, то подсистема запросов сначала на корневом уровне определит страницу на первом промежуточном (intermediate) уровне. В данном случае первой страница указывает на значение от 1 до 100, а вторая от 101 до 200, таким образом подсистема запросов обратится ко второй странице этого промежуточного уровня. Далее будет выяснено, что следует обратиться к третьей странице следующего промежуточного уровня. Отсюда подсистема запросов прочитает на нижнем уровне значение самого индекса. Листья индекса могут содержать как сами данные таблицы, так и просто указатель на строки с данными в таблице, в зависимости от типа индекса: кластеризованный индекс или некластеризованный.

**Кластеризованный индекс**

Кластеризованный индекс хранит реальные строки данных в листьях индекса. Возвращаясь к предыдущему примеру, это означает что строка данных, связанная со значение ключа, равного 123 будет храниться в самом индексе. Важной характеристикой кластеризованного индекса является то, что все значения отсортированы в определенном порядке либо возрастания, либо убывания. Таким образом, таблица или представление может иметь только один кластеризованный индекс. В дополнение следует отметить, что данные в таблице хранятся в отсортированном виде только в случае если создан кластеризованный индекс у этой таблицы.

Таблица не имеющая кластеризованного индекса называется кучей.

##### **Некластеризованный индекс**

В отличие от кластеризованного индекса, листья некластеризованного индекса содержат только те столбцы (*ключевые*), по которым определен данный индекс, а также содержит указатель на строки с реальными данными в таблице. Это означает, что системе подзапросов необходима дополнительная операция для обнаружения и получения требуемых данных. Содержание указателя на данные зависит от способа хранения данных: кластеризованная таблица или куча. Если указатель ссылается на кластеризованную таблицу, то он ведет к кластеризованному индексу, используя который можно найти реальные данные. Если указатель ссылается на кучу, то он ведет к конкретному идентификатору строки с данными. Некластеризованные индексы не могут быть отсортированы в отличие от кластеризованных, однако вы можете создать более одного некластеризованного индекса на таблице или представлении, вплоть до 999. Это не означает, что вы должны создавать как можно больше индексов. Индексы могут как улучшить, так и ухудшить производительность системы. В дополнение к возможности создать несколько некластеризованных индексов, вы можете также включить дополнительные столбцы (*included column*) в свой индекс: на листьях индекса будет храниться не только значение самих индексированных столбцов, но и значения этих не индексированных дополнительных столбцов. Этот подход позволит вам обойти некоторые ограничения, наложенные на индекс. К примеру, вы можете включить неидексируемый столбец или обойти ограничение на длину индекса (900 байт в большинстве случаев).

### Типы индексов

В дополнение к тому, что индекс может быть либо кластеризованным, либо некластеризованным, возможно его дополнительно сконфигурировать как составной индекс, уникальный индекс или покрывающий индекс.

##### **Составной индекс**

Такой индекс может содержать более одного столбца. Вы можете включить до 16 столбцов в индекс, но их общая длина ограничена 900 байтами. Как кластеризованный, так и некластеризованный индексы могут быть составными.

##### **Уникальный индекс**

Такой индекс обеспечивает уникальность каждого значения в индексируемом столбце. Если индекс составной, то уникальность распространяется на все столбцы индекса, но не на каждый отдельный столбец. К примеру, если вы создадите уникальных индекс на столбцах *ИМЯ*и *ФАМИЛИЯ*, то полное имя должно быть уникально, но отдельно возможны дубли в имени или фамилии.

Уникальный индекс автоматически создается когда вы определяете ограничения столбца: первичный ключ или ограничение на уникальность значений:

* *Первичный ключ*

Когда вы определяете ограничение первичного ключа на один или несколько столбцов, тогда *SQL Server* автоматически создаёт уникальный кластеризованный индекс, если кластеризованный индекс не был создан ранее (в этом случае создается уникальный некластеризованный индекс по первичному ключу)

* *Уникальность значений*

Когда вы определяете ограничение на уникальность значений, тогда *SQL Server* автоматически создает уникальный некластеризованный индекс. Вы можете указать, чтобы был создан уникальный кластеризованный индекс, если кластеризованного индекса до сих пор не было создано на таблице

##### **Покрывающий индекс**

Такой индекс позволяет конкретному запросу сразу получить все необходимые данные с листьев индекса без дополнительных обращений к записям самой таблицы.

### Проектирование индексов

Насколько полезны индексы могут быть, настолько аккуратно они должны быть спроектированы. Поскольку индексы могут занимать значительное дисковое пространство, вы не захотите создавать индексов больше, чем необходимо. В дополнение, индексы автоматически обновляются когда сама строка с данными обновляется, что может привести к дополнительным накладным расходам ресурсов и падению производительности. При проектирование индексов должно приниматься во внимание несколько соображений относительно базы данных и запросов к ней.

##### **База данных**

Как было отмечено ранее индексы могут улучить производительность системы, т.к. они обеспечивают подсистему запросов быстрым путем для нахождения данных. Однако, вы должны также принять во внимание то, как часто вы собираетесь вставлять, обновлять или удалять данные. Когда вы изменяете данные, то индексы должны также быть изменены, чтобы отразить соответствующие действия над данными, что может значительно снизить производительность системы. Рассмотрим следующие рекомендации при планировании стратегии индексирования:

* Для таблиц которые часто обновляются используйте как можно меньше индексов.
* Если таблица содержит большое количество данных, но их изменения незначительны, тогда используйте столько индексов, сколько необходимо для улучшение производительности ваших запросов. Однако хорошо подумайте перед использованием индексов на небольших таблицах, т.к. возможно использование поиска по индексу может занять больше времени, нежели простое сканирование всех строк.
* Для кластеризованных индексов старайтесь использовать настолько короткие поля насколько это возможно. Наилучшим образом будет применение кластеризованного индекса на столбцах с уникальными значениями и не позволяющими использовать NULL. Вот почему первичный ключ часто используется как кластеризованный индекс.
* Уникальность значений в столбце влияет на производительность индекса. В общем случае, чем больше у вас дубликатов в столбце, тем хуже работает индекс. С другой стороны, чем больше уникальных значения, тем выше работоспособность индекса. Когда возможно используйте уникальный индекс.
* Для составного индекса возьмите во внимание порядок столбцов в индексе. Столбцы, которые используются в выражениях *WHERE*(к примеру, *WHERE FirstName = 'Charlie'*) должны быть в индексе первыми. Последующие столбцы должны быть перечислены с учетом уникальности их значений (столбцы с самым высоким количеством уникальных значений идут первыми).
* Также можно указать индекс на вычисляемых столбцах, если они соответствуют некоторым требованиям. К примеру, выражение которые используются для получения значения столбца, должны быть детерминистическими (всегда возвращать один и тот же результат для заданного набора входных параметров).

##### **Запросы к базе данных**

Другое соображение которое следует учитывать при проектировании индексов это какие запросы выполняются к базе данных. Как было указано ранее, вы должны учитывать как часто изменяются данные. Дополнительно следует использовать следующие принципы:

* Старайтесь вставлять или модифицировать в одном запросе как можно больше строк, а не делать это в несколько одиночных запросов.
* Создайте некластеризованный индекс на столбцах которые часто используются в ваших запросах в качестве условий поиска в *WHERE*и соединения в *JOIN*.
* Рассмотрите возможность индексирования столбцов, использующихся в запросах поиска строк на точное соответствие значений.

**5. Системные базы данных и таблицы**

В SQL Server существует два вида баз данных — системные базы данных и пользовательские базы данных.

**Системные базы данных** создаются автоматически при инсталляции SQL Server и изменяются, как правило, системой. Хотя некоторые изменения в отдельных системных базах данных может выполнить и пользователь, но это следует делать только в том случае, если пользователь точно понимает, что он делает и к каким результатам это может привести. Профессиональные программисты SQL Server часто после инсталляции системы вносят нужные им изменения в базу данных model, которая используется при создании пользовательских баз данных.

**Пользовательские базы данных** создаются, изменяются и удаляются пользователем (профессиональным пользователем, т. е. человеком, знающим очень многое о базах данных). В такие пользовательские базы данных вы помещаете все те данные, которые необходимы вам (вашему заказчику) для решения задач конкретной предметной области. Сюда же можно помещать и другие нужные для работы объекты базы данных — хранимые процедуры, пользовательские типы данных, триггеры, функции и др.

**Системные базы данных** содержат сведения, необходимые для работы SQL Server.

Системными базами данных являются:

* **master;**
* **model;**
* **msdb;**
* **tempdb.**

Существует еще одна скрытая системная база данных resource, которая хранит системные объекты, входящие в состав SQL Server. Эта база данных не отображается в списке системных баз данных, никакие сведения о ней естественными средствами получить невозможно. Напрямую к ней обратиться нельзя, однако существуют средства (системные функции и системные представления), позволяющие получить из нее некоторые данные. Об этой базе данных чуть позже.

**База данных master**

База данных master является, пожалуй, наиболее важной системной базой данных в SQL Server. Она содержит все данные, необходимые для работы с СУБД. Она также содержит данные о конфигурации сервера базы данных, сведения обо всех пользовательских базах данных, созданных в экземпляре сервера: характеристики баз данных, характеристики и размещение файлов каждой базы данных. Настоятельно рекомендуется выполнять ее резервное копирование при создании, изменении или удалении любой базы данных пользователя. SQL Server не сможет выполняться, если база данных master недоступна.

Главная служебная база данных всего сервера. В ней хранится общая служебная информация сервера: настройки его работы, список баз данных на сервере с информацией о настройках каждой базы данных и ее файлах, информация об учетных записях пользователей, серверных ролях и т.п.

База данных состоит из двух файлов: из файла данных (логическое имя master, имя файла базы данных master.mdf) и файла журнала транзакций (логическое имя mastlog, имя файла mastlog.ldf).

**База данных model**

Основным назначением базы данных model является хранение шаблонов для всех вновь создаваемых пользователем баз данных. При создании новой пользовательской базы данных в нее из базы данных model копируются типы данных. Создаваемым базам данных присваиваются значения по умолчанию многочисленных характеристик, которые также выбираются из базы данных model. Если вы добавите новые объекты в базу данных model, то эти объекты будут копироваться во все вновь создаваемые пользовательские базы данных текущего экземпляра сервера.

База данных состоит из двух файлов: файла данных (логическое имя modeldev, имя файла model.mdf) и журнала транзакций (логическое имя modellog, имя файла modellog.ldf).

**База данных msdb**

В SQL Server существуют средства создания расписаний (schedule) для автоматического выполнения заданий, ведения истории их выполнения и для выдачи предупреждающих сообщений (alert). Все это хранится в базе данных msdb. Используется в основном компонентом SQL Server Agent. В этой базе данных также хранится история создания резервных копий, SSIS-пакеты, сведения о репликациях. Используется компонентами Service Broker и database mail.

Эта база данных в основном используется для хранения информации службы SQL Server Agent (пакетных заданий, предупреждений и т.п.), но в нее записывается и другая служебная информация (например, история резервного копирования).

Эта база данных состоит из двух файлов: файла данных (логическое имя MSDBData, имя файла MSDBData.mdf) и журнала транзакций (логическое имя MSDBLog, имя файла MSDBLog.ldf).

**База данных tempdb**

В системной базе данных tempdb хранятся временные объекты, создаваемые пользователями (в первую очередь это временные таблицы, которые существуют только на время выполнения соответствующей программы пользователя, где они были созданы), это внутренние объекты, создаваемые сервером базы данных при выполнении запросов, а также ряд других объектов. Во многих случаях использование базы данных tempdb позволяет повысить производительность системы при выполнении различных операций с базами данных. Эта база данных создается заново при каждом запуске SQL Server.

База данных так же, как и все остальные системные базы данных, использует два файла: файл данных (логическое имя tempdev, имя файла tempdb.mdf) и журнал транзакций (логическое имя templog, имя файла templog.ldf).

**База данных resource**

В SQL Server существует и такой невидимый обычными средствами объект, как скрытая база данных resource. Эта база данных в схеме sys содержит системные объекты SQL Server (системные хранимые процедуры, представления, функции), которые доступны из любой пользовательской и системной базы данных. Использование ресурсной БД облегчает внесение изменений при установке пакетов исправлений и иных обновлений за счет простой замены на новую версию базы данных.

Эта база данных хранится отдельно от всех других системных баз данных. Несмотря на то, что она сильно засекречена, сообщу вам имена ее файлов: файл данных имеет физическое имя mssqlsystemresource.mdf, файл журнала транзакций — физическое имя mssqlsystemresource.ldf.

# Системные таблицы.

При создании базы данных вы сразу же получаете набор системных таблиц. Вы их, конечно же, видели. Их имена начинаются с sys. Вот и пришло время узнать, какую информацию они хранят и для чего они служат.

Повторюсь. Любая создаваемая база данных представляет из себя копию базы данных model. Т.о. все системные таблицы, в исходном виде, можно найти в ней.

### syscolumns

В ней содержится общее описание по каждому из полей таблиц, представлений и по каждому из параметров хранимых процедур, которые есть в базе данных. Например, обязательное ли поле для заполнения, либо нет, установлено ли значение по умолчанию и пр.

### syscomments

Эта таблица хранит синтаксис определения все представлений, ограничений, хранимых процедур и триггеров. При удалении значения из этой таблицы вручную, может привести к некорректной работе того объекта, определение которого было удалено.

### sysdepends

В этой таблице сохраняется информация о зависимостях между представлениями, хранимыми процедурами, триггерами и таблицами, которые указываются при их определении.

### sysfilegroups

Содержит информацию о Группах файлов, которые есть в базе данных.

### sysfiles

В ней хранится информация о логическом и физическом названиях файлов, о их максимальном размере и пр. Эта таблица является виртуальной и не может быть модифицирована напрямую.

### sysаforeignkeys

Хранит информацию обо всех ограничениях типа Внешний Ключ. Т.е. указывается внешний ключ какой таблицы находится в связи с первичным ключом другой таблицы.

### sysindexes

Хранит информацию обо всех индексах, которые есть в базе данных.

### sysindexkeys

Содержит информацию о индексах и полях, на которые они указывают.

### sysmembers

В этой таблице храниться информация о пользователях и ролях, к которым они относятся. Роль - это набор прав и ограничений, который определяет возможность запрета доступа к одним данным и разрешение на доступ, и уровень доступа к другим данным.

### sysobjects

Если таблица syscomments хранит синтаксис ограничений и хранимых процедур, то в этой таблице находится их описание. Например, уникальный номер, идентификатор владельца и т.д.

### syspermissions

Здесь указываются права и ограничения, установленные на определенного пользователя, группу или роль.

### sysreferences

Содержит карту связей таблиц.

### systypes

Хранит список все типов данных (пользовательских и системных), которые могут использоваться при создании таблиц.

### sysusers

В этой таблице указываются все пользователи и роли SQL Server и Windows.

Запросы:

**Введение в язык структурированных запросов SQL**

**SQL (Structured Query Language) — язык структурированных запросов.**

**SQL** (Structured Query Language, язык структурированных запросов) — это специальный язык, используемый для определения данных, доступа к данным и их обработки. Язык SQL относится к *непроцедурным* (nonprocedural) языкам — он лишь описывает нужные компоненты (например, таблицы) и желаемые результаты, не указывая, как именно эти результаты должны быть получены. Каждая реализация SQL является надстройкой над процессором базы данных (database engine), который интерпретирует операторы SQL и определяет порядок обращения к структурам БД для корректного и эффективного формирования желаемого результата.

Стандарт SQL определяется *ANSI* — American National Standarts Institute (Американским Национальным Институтом Стандартов) и в настоящее время принят *ISO* — International Standarts Organization (Международной Организацией по Стандартизации).

SQL — *непроцедурный язык:* серверу базы данных сообщается, что нужно сделать и каким образом. Для обработки запроса сервер базы данных транслирует команды SQL во внутренние процедуры. Благодаря тому, что SQL скрывает детали обработки данных, его легко использовать.

В начале 1970-х годов в одной из исследовательских лабораторий компании IBM была разработана *экспериментальная реляционная СУБД System R* (англ.), для которой затем был создан специальный язык **SEQUEL**, позволявший относительно просто управлять данными в этой СУБД. Аббревиатура SEQUEL расшифровывалась как Structured English QUEry Language - структурированный английский язык запросов. Позже по юридическим соображениям язык SEQUEL был переименован в **SQL** и официальным произношением стало [,es kju:' el] - эс-кью-эл.

Первыми СУБД, поддерживающими новый язык, стали в **1979 году** **OracleV2** для машин VAX от компании Relational Software Inc. и **System/38** от IBM, основанная на System/R.

Первый официальный стандарт языка SQL был принят ANSI в 1986 году и ISO в 1987 году (так называемый SQL-86) и несколько уточнён в 1989 году. Дальнейшее развитие языка поставщиками СУБД потребовало принятия в 1992 году нового расширенного стандарта (ANSI SQL-92 или просто SQL2). Следующим стандартом стал SQL:1999 (SQL3). В настоящее время действует стандарт, принятый в 2003 году (SQL:2003) с небольшими модификациями, внесёнными позже.

Начиная с 1986 года, комитеты ISO (International Organization for Standardization) и ANSI (American National Standards Institute) приступили к созданию ряда стандартов языка SQL, которые впоследствии были приняты и получили следующие названия: SQL86, SQL89, SQL92 и SQL99.

**Стандарт SQL86** зафиксировал минимальный стандартный синтаксис языка SQL.

**Стандарт SQL89** был принят в **1989** году. Он вводил набор операторов языка SQL, которые должны были реализовывать все СУБД, заявляющие поддержку стандарта SQL89. На практике каждая реальная коммерческая СУБД предоставляет значительно более широкий набор возможностей, чем предусмотрено стандартом. Так, несмотря на то, что большинство СУБД на момент принятия стандарта уже поддерживали встроенный и динамический SQL, в стандарте SQL89 правила встраивания языка SQL в процедурный язык программирования (такой как язык С) и правила использования динамического SQL прописаны не были.

В **стандарте SQL92** было определено три уровня соответствия:

* основной (Entry);
* средний (Intermediate);
* полный (Full).

При этом, для того чтобы объявить СУБД поддерживающей стандарт SQL92, большинство производителей реализовывали только основной уровень соответствия.

Стандарт **SQL99**, при разработке именовавшийся как SQL3, стандартизировал объектные расширения языка SQL и некоторые процедурные расширения языка SQL. К моменту принятия этого стандарта большинство коммерческих СУБД, таких как Oracle, уже де-факто ввели использование объектных типов и наследования.

В **1999** году были приняты пять частей стандарта **SQL:1999**.

Первая часть (**SQL/Framework**) посвящена *описанию концептуальной структуры стандарта*. В этой части приводится развернутая аннотация следующих четырех частей и формулируются требования к реализациям, претендующим на соответствие стандарту.

Вторая часть SQL:1999 (**SQL/Foundation**) образует *базис стандарта*. Вводится система типов языка, формулируются правила определения функциональных зависимостей и возможных ключей, определяются синтаксис и семантика основных операторов SQL:

* операторов определения и манипулирования схемой базы данных;
* операторов манипулирования данными;
* операторов управления транзакциями;
* операторов управления подключениями к базе данных и т. д.

Третью часть занимает уточненная по сравнению с SQL/92 спецификация **SQL/CLI (**Call-Level Interface (интерфейс уровня вызовов), - расширение стандарта SQL, закреплённое в **ISO/IEC 9075-3:2003**. Данное расширение определяет общее взаимодействие компонентов (структур и процедур), которое может использоваться при выполнении SQL-запросов от приложения, написанного на других языках программирования. Расширение SQL/CLI определяет порядок, в котором по отдельности вызываются SQL-запросы и SQL/CLI-процедуры из исходного кода вызывающего приложения). В четвертой части специфицируется **SQL/PSM** – синтаксис и семантика языка определения хранимых процедур. Наконец, в пятой части – **SQL/Bindings** – определяются правила связывания SQL для стандартных версий языков программирования.

В конце 2003 года был принят и опубликован новый вариант международного стандарта **SQL:2003**. Многие специалисты считали, что в варианте стандарта, следующем за SQL:1999, будут всего лишь исправлены неточности SQL:1999. Но на самом деле, в SQL:2003 специфицирован ряд новых и важных свойств, с небольшими модификациями, внесёнными позже в 2008 году.

Наиболее серьезные изменения языка SQL, специфицированные в части 2 стандарта SQL:2003, касаются следующих аспектов:

• типы данных;

• подпрограммы, вызываемые из SQL;

• расширенные возможности оператора CREATE TABLE;

• новый объект схемы – генератор последовательностей;

• новые виды столбцов – идентифицирующие столбцы (identity column) и генерируемые столбцы (generated column);

• новый оператор MERGE;

Претерпела некоторые изменения общая организация стандарта. Стандарт SQL:2003 состоит из следующих частей:

• 9075-1, SQL/Framework;

• 9075-2, SQL/Foundation;

• 9075-3, SQL/CLI;

• 9075-4, SQL/PSM;

• 9075-9, SQL/MED;

• 9075-10, SQL/OLB;

• 9075-11, SQL/Schemata;

• 9075-13, SQL/JRT;

• 9075-14, SQL/XML.

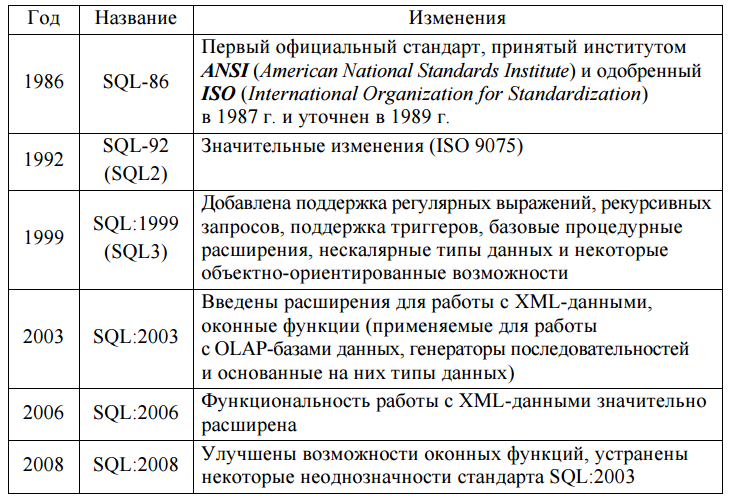
Части 1-4 и 9-10 с необходимыми изменениями остались такими же, как и в SQL:1999. Часть 5 (SQL/Bindings) перестала существовать; соответствующие спецификации включены в часть 2.

Раздел части 2 SQL:1999, посвященный информационной схеме, выделен в отдельную часть 11. Появились две новые части – 13 и 14.

Часть 13 полностью называется «SQL Routines and Types Using the Java Programming Language» («Использование подпрограмм и типов SQL в языке программирования Java»). Появление такой части стандарта оправдано повышенным вниманием к языку Java со стороны ведущих производителей SQL-ориентированных СУБД.

Наконец, последняя часть SQL:2003 посвящена спецификациям языковых средств, позволяющих работать с XML-документами в среде SQL.

Несмотря на старания разработчиков, процесс стандартизации явно не поспевает за происходящими изменениями.



Тем не менее, можно сказать, что базовый набор операторов SQL, включающий операторы определения схемы БД, выборки и манипулирования данными, авторизации доступа к данным, поддержки встраивания SQL в языки программирования и операторы динамического SQL, в коммерческих реализациях устоялся и более или менее соответствует стандарту.

**Диалекты языка SQL в СУБД**

Несмотря на наличие международного стандарта ANSI SQL, многие компании, занимающиеся разработкой СУБД, вносят изменения в язык SQL, применяемый в разрабатываемой СУБД, тем самым отступая от стандарта. Каждая из реализаций языка SQL в конкретной СУБД называется диалектом. Функции, которые добавляются к стандарту языка разработчиками коммерческих реализаций, принято называть расширениями. Например, в стандарте языка SQL определены конкретные типы данных, которые могут храниться в базах данных. Во многих реализациях этот список расширяется за счет разнообразных дополнений.

Выделяют три уровня соответствия стандарту ANSI/ISO — *начальный, промежуточный и полный*. В настоящее время не существует ни одного диалекта, полностью соответствующего стандарту. Производители СУБД (например, Oracle, Microsoft, Borland, Informix, Sybase) применяют собственные реализации SQL, отвечающие как минимум начальному уровню соответствия стандарту и содержащие некоторые расширения, специфические для данной СУБД. Не существует двух совершенно идентичных диалектов. Более того, поскольку разработчики баз данных вводят в системы все новые функциональные средства, они постоянно расширяют свои диалекты языка SQL, в результате чего отдельные диалекты все больше и больше отличаются друг от друга. Это имеет свои достоинства и недостатки.

Конкретная реализация языка, может включать в себя более широкие возможности по сравнению со стандартом SQL, например, больше типов данных, большее количество команд, больше дополнительных возможностей у имеющихся команд. Такие возможности делают работу с конкретной СУБД более эффективной. Кроме того, такие нестандартные возможности языка проходят практическую апробацию и со временем могут быть включены в стандарт. Недостаток в том, что различия в синтаксисе реализаций SQL затрудняют перенос приложений из одной системы в другую. Например, если приложение было написано для базы данных MS SQL Server с использованием своего диалекта SQL – языка Transact-SQL, то при переносе системы в базу данных ORACLE, не все конструкции языка будут понятны соответствующему диалекту SQL – языку PL/SQL.

В широко распространенных в настоящее время СУБД используются следующие диалекты языка SQL:

* **PL/SQL – в СУБД Oracle;**
* **Transact-SQL – в СУБД Microsoft SQL;**
* **Informix-SQL – в СУБД Informix;**
* **Jet SQL – Microsoft Access.**

Язык Jet SQL почти соответствует стандарту ANSI SQL.

**Диалект Transact-SQL**

Несмотря на наличие стандартов, практически в каждой СУБД применяется свой диалект языка.

T**ransact-SQL (T-SQL)** — процедурное расширение языка SQL, созданное компанией Microsoft (для Microsoft SQL Server) и Sybase (для Sybase ASE).

SQL был расширен такими дополнительными возможностями как:

* управляющие операторы,
* локальные и глобальные переменные,
* различные дополнительные функции для обработки строк, дат, математики и т.п.,
* поддержка аутентификации Microsoft Windows.

Язык Transact-SQL является ключом к использованию MS SQL Server. Все приложения, взаимодействующие с экземпляром MS SQL Server, независимо от их реализации и пользовательского интерфейса, отправляют серверу инструкции Transact-SQL.

**Элементы синтаксиса**

### Директивы сценария

Директивы сценария — это специфические команды, которые используются только в MS SQL. Эти команды помогают серверу определять правила работы со скриптом и транзакциями. Типичные представители: GO — информирует программы SQL Server об окончании пакета инструкций Transact-SQL, EXEC (или EXECUTE) — выполняет процедуру или скалярную функцию.

### Комментарии

Комментарии используются для создания пояснений для блоков сценариев, а также для временного отключения команд при отладке скрипта. Комментарии бывают как строковыми, так и блоковыми:

-- — строковый комментарий исключает из выполнения только одну строку, перед которой стоят два минуса.

/\* \*/ — блоковый комментарий исключает из выполнения целый блок команд, заключённый в указанную конструкцию. Допускает вложенные комментарии.

### Типы данных

Как и в языках программирования, в SQL существуют различные типы данных для хранения переменных:

* Числа — для хранения числовых переменных (bit, int, tinyint, smallint, bigint, numeric, decimal, money, smallmoney, float, real).
* Даты — для хранения даты и времени (datetime, smalldatetime, date, time, datetime2, datetimeoffset).
* Символы — для хранения символьных данных (char, nchar, varchar, nvarchar).
* Двоичные — для хранения бинарных данных (binary, varbinary).
* Большеобъемные — типы данных для хранения больших бинарных данных (text, ntext, image).
* Специальные — указатели (cursor), 16-байтовое шестнадцатеричное число, которое используется для GUID (uniqueidentifier), штамп изменения строки (timestamp), версия строки (rowversion), таблицы (table).

*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» (от слова national). Строковые константы с* Unicode *также записываются с «n» в начале.*

*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.*

### Идентификаторы

Идентификаторы — это специальные символы, которые используются с переменными для идентифицирования их типа или для группировки слов в переменную. Типы идентификаторов:

* @ — идентификатор локальной переменной (пользовательской).
* @@ — идентификатор глобальной переменной (встроенной).
* # — идентификатор локальной таблицы или процедуры.
* ## — идентификатор глобальной таблицы или процедуры.
* [ ] — идентификатор группировки слов в переменную (работают как стандартные " ").

### Переменные

Переменные используются в сценариях и для хранения временных данных. Чтобы работать с переменной, её нужно объявить, при том объявление должно быть осуществлено в той транзакции и пакете инструкций, в которой выполняется команда, использующая эту переменную. Иначе говоря, после завершения транзакции, а также после команды GO, переменная уничтожается.

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**

*Примечание. В этом примере используется группировка слов в лексему — конструкция [Employee Name] воспринимается как одна лексема, так как слова заключены в квадратные скобки.*

### Операторы

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

* Арифметические операторы: «\*» — умножить, «/» — делить, «%» — остаток от деления, «+» — сложить, «-» — вычесть, «()» — скобки.
* Операторы сравнения: «=» — равно, «>» — больше, «<» — меньше, «>=» — больше или равно, «<=» меньше или равно, «<>» («!=») — не равно, between (вместо «>=», «<=»).
* Операторы соединения: «+» — соединение (конкатенация) строк.
* Логические операторы: «AND» — и, «OR» — или, «NOT» — не.
* Операторы со множествами: «IN».

### 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.

*Примечание. Полный список функций можно найти в справке к SQL серверу.*

*Примечание. К скалярным функциям можно также отнести и глобальные переменные, которые в тексте сценария вызываются двойной собакой «@@».*

Пример:

USE TestDatabase

*-- Использование агрегативной функции для подсчета средней зарплаты*

**SELECT** **AVG**(BaseSalary) **AS** [Average salary] **FROM** Positions

**GO**

*-- Использование скалярной функции для получения имени базы данных*

**SELECT** DB\_NAME() **AS** [**Database** name]

**GO**

*-- Использование скалярной функции для получения имени текущего пользователя*

**DECLARE** @MyUser char(30)

**SET** @MyUser = USER\_NAME()

**SELECT** 'The current user''s database username is: '+ @MyUser

**GO**

*-- Использование функции-указателя для получения данных с другого сервера*

**SELECT** \* **FROM** OPENQUERY(OracleSvr, 'SELECT name, id FROM owner.titles')

**GO**

### Выражения

Выражение — это комбинация символов и операторов, которая получает на вход скалярную величину, а на выходе дает другую величину или исполняет какое-то действие. В Transact-SQL выражения делятся на 3 типа: DDL, DCL и DML.

* DDL (Data Definition Language) — используются для создания объектов в базе данных. Основные представители данного класса: CREATE — создание объектов, ALTER — изменение объектов, DROP — удаление объектов.
* DCL (Data Control Language) — предназначены 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.
* DML (Data Manipulation Language) — используются для запросов и изменения данных. Основные представители данного класса: SELECT — выборка данных, INSERT — вставка данных, UPDATE — изменение данных, DELETE — удаление данных.

Пример:

USE TestDatabase

*-- Использование DDL*

**CREATE** **TABLE** TempUsers (UserID int, UserName nvarchar(40), DepartmentID int)

**GO**

*-- Использование DCL*

**GRANT** **SELECT** **ON** Users **TO** **public**

**GO**

*-- Использование DML*

**SELECT** UserID, UserName + ' ' + UserSurname **AS** [**User** **Full** Name] **FROM** Users

**GO**

*-- Использование DDL*

**DROP** **TABLE** TempUsers

**GO**

### Управление выполнением сценария

В Transact-SQL существуют специальные команды, которые позволяют управлять потоком выполнения сценария, прерывая его или направляя в нужную ветвь.

* Блок группировки — структура, объединяющая список выражений в один логический [блок](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (BEGIN … END).
* Блок условия — структура, проверяющая выполнения определённого условия (IF … ELSE).
* Блок цикла — структура, организующая повторение выполнения логического блока (WHILE … BREAK … CONTINUE).
* Переход — команда, выполняющая переход потока выполнения сценария на указанную метку (GOTO).
* Задержка — команда, задерживающая выполнение сценария (WAITFOR).
* Вызов ошибки — команда, генерирующая ошибку выполнения сценария (RAISERROR).

#### Data Definition Language (DDL)

DDL - Data Definition Language. Язык определения данных. Используется для создания и изменения параметров баз данных, таблиц, представлений и пр. Эти операторы перечислены в табл. 1.

**Таблица 1**

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| CREATE TABLE | Применяется для добавления новой таблицы к базе данных |
| DROP TABLE | Применяется для удаления таблицы из базы данных |
| ALTER TABLE | Применяется для изменения структуры имеющейся таблицы |
| CREATE VIEW | Применяется для добавления нового представления к базе данных |
| DROP VIEW | Применяется для удаления представления из базы данных |
| CREATE INDEX | Применяется для создания индекса для данного поля |
| DROP INDEX | Применяется для удаления существующего индекса |
| CREATE SCHEMA | Применяется для создания новой схемы в базе данных |
| DROP SCHEMA | Применяется для удаления схемы из базы данных |
| CREATE DOMAIN | Применяется для создания нового домена |
| ALTER DOMAIN | Применяется для переопределения домена |
| DROP DOMAIN | Применяется для удаления домена из базы данных |

1. Create Database - создание базы данных.

2. Drop Database - удаление базы данных.

3. Alter Database - изменение параметров базы данных.

#### 

#### Data Manipulation Language (DML)

Data Manipulation Language содержит операторы, позволяющие выбирать, добавлять, удалять и модифицировать данные. Обратите внимание на то, что эти операторы не обязаны завершать транзакцию, внутри которой они вызваны. Операторы DML представлены в табл. 2.

**Таблица 2**

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| SELECT | Применяется для выбора данных |
| INSERT | Применяется для добавления строк к таблице |
| DELETE | Применяется для удаления строк из таблицы |
| UPDATE | Применяется для изменения данных |

Иногда оператор SELECT относят к отдельной категории, называемой Data Query Language (DQL).

#### 

#### Data Control Language (DCL)

Операторы Data Control Language, иногда называемые операторами Access Control Language, применяются для осуществления административных функций, присваивающих или отменяющих право (привилегию) использовать базу данных, таблицы в базе данных, а также выполнять те или иные операторы SQL. Операторы DCL представлены в табл. 4.

**Таблица 4**

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| GRANT | Применяется для присвоения привилегии (назначение прав доступа) |
| REVOKE | Применяется для отмены привилегии (аннулирование прав доступа) |

**Достоинства и недостатки языка SQL**

|  |
| --- |
|  |
| Достоинства | Недостатки |
| Универсальность: несмотря на существование разных диалектов языка, в большой степени SQL не зависит от СУБД, на которой он используется | Изначально SQL разрабатывался как язык, максимально приближенный по виду к английскому, однако со временем развился в достаточно сложный инструмент, малопонятный для среднего пользователя |
| Стандарты: будучи высоко развитым языком с более чем двадцатилетней историей, SQL имеет стандарты, которые сохраняют устойчивость языка в контексте новых разработок | Некоторые особенности языка SQL создают возможность для ошибок при работе с реляционными БД: SQL разрешает повторяющиеся строки в таблицах, неопределенные значения записей (NULL), колонки таблиц пронумерованы, разрешены пустые и повторяющиеся имена полей |
| Декларативность: при отправлении SQL-запроса, пользователь не должен определять технически, каким образом будут получены данные из БД - система управления базой данных самостоятельно переведет операторы SQL на технический язык более низкого уровня | В некоторых случаях отступления от стандарта все же становятся достаточно сильными, что мешает при разработке решения для разных систем |