**Лабораторная работа №1: Создание баз данных**

**Утилита SQL Server Management Studio**

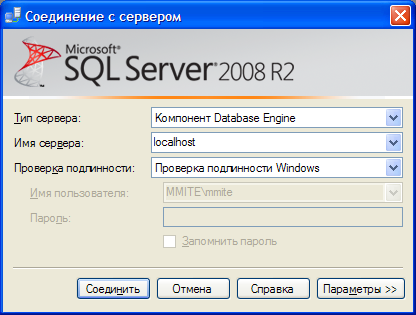
Подавляющую массу задач администрирования SQL Server можно выполнить в графической утилите SQL Server Management Studio. В ней можно создавать базы данных и все ассоциированные с ними объекты (таблицы, представления, хранимые процедуры и др.). Здесь вы можете выполнить последовательности инструкций Transact-SQL (запросы). В этой утилите можно выполнить типовые задачи обслуживания баз данных, такие как резервирование и восстановление. Здесь можно настраивать систему безопасности базы данных и сервера, просматривать журнал ошибок и многое другое.

Для запуска Management Studio в меню «Пуск» операционной системы выберите пункт «Microsoft SQL Server 2008 R2\Среда SQL Server Management Studio». Когда откроется окно программы, вас попросят подключиться к какому либо серверу баз данных SQL Server.

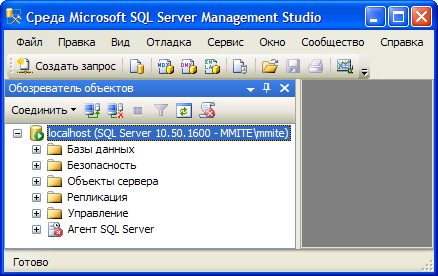
**Подключение к серверу**

В окне «Соединение с сервером» необходимо указать следующую информацию:

* *Тип сервера*. Здесь следует выбрать, к какой именно службе необходимо подключится. Оставьте вариант «Компонент Database Engine».
* *Имя сервера*. Позволяет указать, к какому серверу будет осуществляться подключение. По умолчанию имя SQL Server совпадает с именем компьютера. Выберите ваш локальный компьютер.
* *Проверка подлинности*. Способ аутентификации, можно выбрать «Проверка подлинности Windows» или «Проверка подлинности SQL Server». Первый способ использует учетную запись, под которой текущий пользователь осуществил вход в Windows. Вариант SQL Server использует свою собственную систему безопасности. Оставьте вариант проверки подлинности Windows.



После подключения экземпляр сервера будет отображаться на панели «Обозреватель объектов».



Окно Management Studio имеет следующую структуру:

* Оконное меню – содержит полный набор команд для управления сервером и выполнения различных операций.
* Панель инструментов – содержит кнопки для выполнения наиболее часто производимых операций. Внешний вид данной панели зависит от выполняемой операции.
* Панель «Обозреватель объектов». Это панель с древовидной структурой, отображающая все объекты сервера, а также позволяющая производить различные операции, как с самим сервером, так и с его базами данных и их объектами. Обозреватель объектов является основным инструментом для разработки.
* Рабочая область. В рабочей области производятся все действия с базой данных, а также отображается её содержимое.

Прежде чем перейти к созданию своих собственных рабочих баз данных рассмотрим служебные базы данных SQL Server, которые создаются автоматически в процессе его установки. Если мы раскроем узел «Базы данных – Системные базы данных» в обозревателе объектов, то увидим следующий набор служебных баз данных:

* *master*. 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.
* *msdb*. 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).
* *model*. Эта база данных является шаблоном для создания новых баз данных в SQL Server. Если внести в нее изменения, например, создать набор таблиц, то эти таблицы будут присутствовать во всех создаваемых базах данных.
* *tempdb*. Эта база данных предназначена для временных таблиц и хранимых процедур, создаваемых пользователями и самим SQL Server. Эта база данных создается заново при каждом запуске SQL Server.

**Создание пользовательских баз данных**

База данных представляет собой группу файлов, хранящихся на жестком диске. Эти файлы могут относиться к трем типам: файлы с первичными данными, файлы с вторичными данными и файлы журнала транзакций. Любая база данных SQL Server содержит, по крайней мере, два файла: первичный файл данных (с расширением .mdf) и файл журнала транзакций (с расширением .ldf). Существует два способа их создания:

* графически с помощью SQL Server Management Studio
* посредством кода Transact-SQL

**Создание баз данных в SQL Server Management Studio**

Использование данной утилиты является самым простым способом создания базы данных. Создадим базу данных Sales (Продажи), которую позже заполним таблицами, представлениями и другими объектами, предназначенными для отдела продаж.

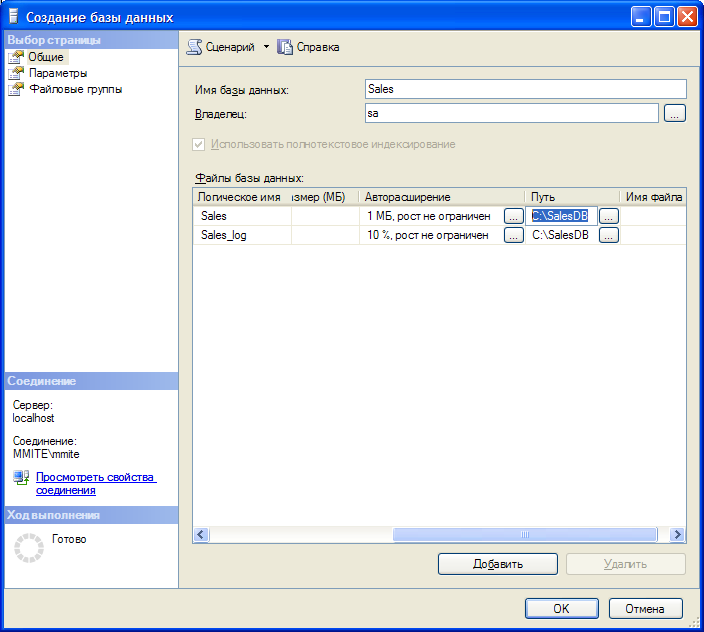
1. В окне «Обозреватель объектов» найдите и раскройте папку «Базы данных». Щелкните на ней правой кнопкой мыши и выберите команду «Создать базу данных…».
2. В открывшемся диалоговом окне «Создание базы данных» на странице «Общие» введите следующую информацию:

Имя базы данных: Sales

Владелец: sa

В таблице «Файлы базы данных» измените путь к файлам данных и журнала на ваш каталог.

Для всех остальных параметров оставьте значения по умолчанию.



1. Для создания базы данных щелкните «OK». Вы должны увидеть свою новую базу данных в окне «Обозреватель объектов».

**Создание баз данных с помощью Transact-SQL**

Для программного создания базы данных (например, в программе установки приложения) используется инструкция CREATE DATABASE языка T-SQL (сокращенная форма от Transact-SQL). Данная инструкция может включать в себя множество опций, определяющих различные параметры новой базы данных.

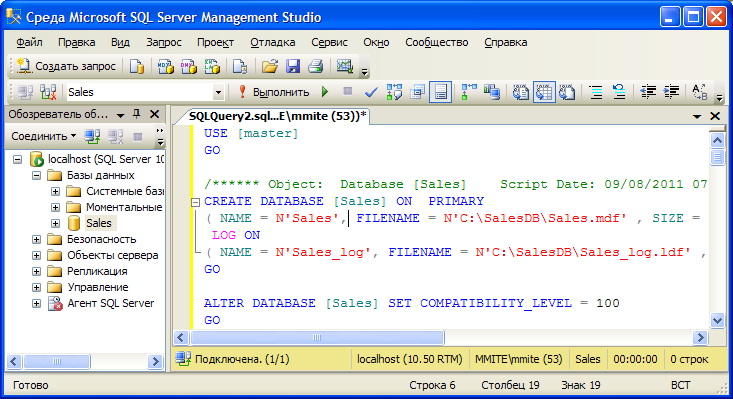
Сценарий создания новой базы данных может быть сгенерирован на основе уже существующей базы данных. Для этого в SQL Server Management Studio в контекстном меню узла «Sales» выберите команду «Создать сценарий для базы данных – Используя CREATE – Буфер обмена». В результате в буфер обмена будет сохранен текст запроса на создание новой базы данных с параметрами, указанными при создании базы данных Sales в Management Studio.

Для проверки работоспособности сгенерированного запроса на создание базы данных удалим базу данных Sales. В контекстном меню базы данных выберите команду «Удалить» и в появившемся диалоговом окне нажмите кнопку «OK». База данных со всеми файлами должна исчезнуть.

Чтобы воспользоваться сгенерированным заранее запросом на создание базы данных выполните следующие шаги:

1. В контекстном меню базы Sales выберите команду «Создать запрос» или щелкните соответствующую кнопку на панели инструментов .
2. В открывшемся окне редактора SQL вставьте из буфера обмена сгенерированный запрос.
3. Для запуска запроса на выполнение щелкните кнопку  на панели инструментов или нажмите клавишу F5.
4. Обновите содержимое дерева обозревателя объектов командой «Обновить» из контекстного меню узла «Базы данных». База данных Sales должна вновь появиться в списке доступных.

При необходимости перед выполнением текст запроса может быть изменен в этом же окне.



Рассмотрим основные опции сгенерированной инструкции CREATE DATABASE:

* Имя новой базы данных. Указывается непосредственно после ключевого слова CREATE DATABASE. В данном случае это Sales.
* ON. Это опция указывает на файловую группу, которая представляет собой логическую группу вторичных файлов данных и используется для управления размещением пользовательских объектов (таких как таблицы и индексы). Опция PRIMARY после аргумента ON используется для указания группы файлов PRIMARY, в которую по умолчанию входят все созданные файлы, и которая является единственной группой файлов, содержащей первичный файл данных.
* NAME. Логическое имя базы данных, которое будет применяться для ссылки на нее из кода T-SQL.
* FILENAME. Это имя и путь файла базы данных, хранящегося на жестком диске.
* SIZE. Исходный размер файлов данных.
* MAXSIZE. Максимальный размер, до которого может расти база данных.
* FILEGROWTH. Это приращение расширения файла

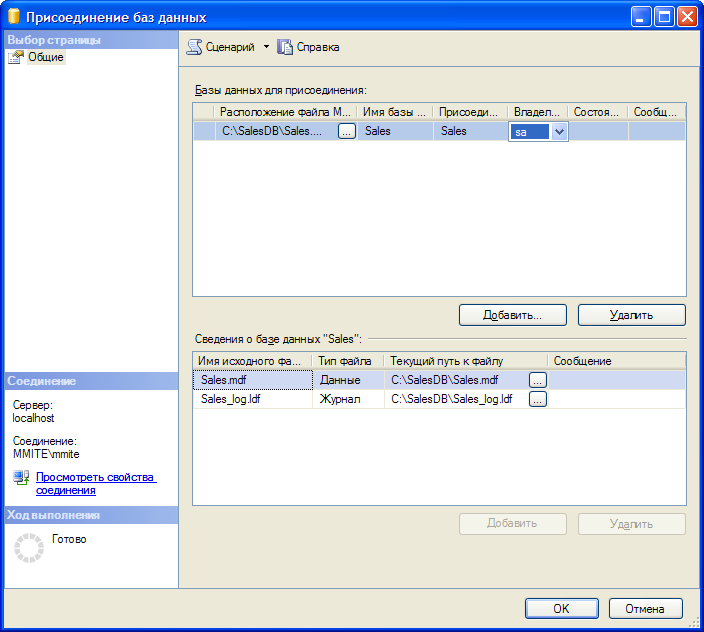
Параметры в разделе LOG ON аналогичны параметрам в разделе CREATE DATABASE. Однако они определяют параметры файла журнала транзакций.

Общий синтаксис инструкции CREATE DATABASE со всеми возможными опциями можно посмотреть в справочной системе. Для этого в редакторе запросов выделите слова CREATE DATABASE и нажмите клавишу F1.

**Отсоединение и присоединение базы данных**

Для переноса базы данных на другой сервер необходимо отсоединить ее от текущего сервера. Для этого в контекстном меню базы данных Sales выберите команду «Задачи - Отсоединить…». В диалоговом окне «Отсоединение базы данных» нажмите кнопку «OK» и убедитесь, что Sales исчезла из списка баз данных в дереве обозревателя объектов. Теперь файлы базы данных могут быть перенесены на другой сервер.

Для присоединения базы данных к серверу выберите в контекстном меню узла «Базы данных» команду «Присоединить…». В диалоговом окне «Присоединение базы данных» с помощью кнопки «Добавить…» выберите созданный на предыдущих этапах файл Sales.mdf (ldf файл будет определен системой автоматически), измените владельца на sa и нажмите кнопку «OK». База данных Sales должна появиться в списке дерева обозревателя объектов.



**Лабораторная работа №2: Создание таблиц и ограничений**

*Таблицы* представляют собой объекты базы данных, используемые непосредственно для хранения всех данных. Одним из самых главных правил организации баз данных является то, что в одной таблице должны храниться данные лишь об одном конкретном типе сущности (например, клиенты, товары, заказы и т. п.).

Данные в таблицах организованы по полям и записям. Поля (или столбцы таблицы) содержат определенный тип информации, например, фамилию, адрес, телефонный номер. Запись (или строка таблицы) - группа связанных полей, содержащих информацию об отдельном экземпляре сущности.

Любое поле таблицы характеризуется как минимум тремя обязательными свойствами:

* *Имя столбца*. Реализует способ обращения к конкретному полю в таблице. Рекомендуется всегда присваивать полям смысловые имена.
* *Тип данных*. Определяет, информация какого типа может храниться в данном поле.
* *Разрешить значения null*. Определяет, допустимо ли для данного поля отсутствие фактических данных, для обозначения которого используется так называемый маркер пустого значения null.

**Типы данных**

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. Текстовые данные

* char. Содержит символьные не Unicode-данные фиксированной длины до 8000 знаков.
* varchar. Содержит символьные не Unicode-данные переменной длины до 8000 знаков.
* nchar. Содержит данные Unicode фиксированной длины до 4000 символов. Подобно всем типам данных Unicode его удобно использовать для хранения небольших фрагментов текста, которые будут считываться разноязычными клиентами.
* nvarchar. Содержит данные Unicode переменной длины до 4000 символов.

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.

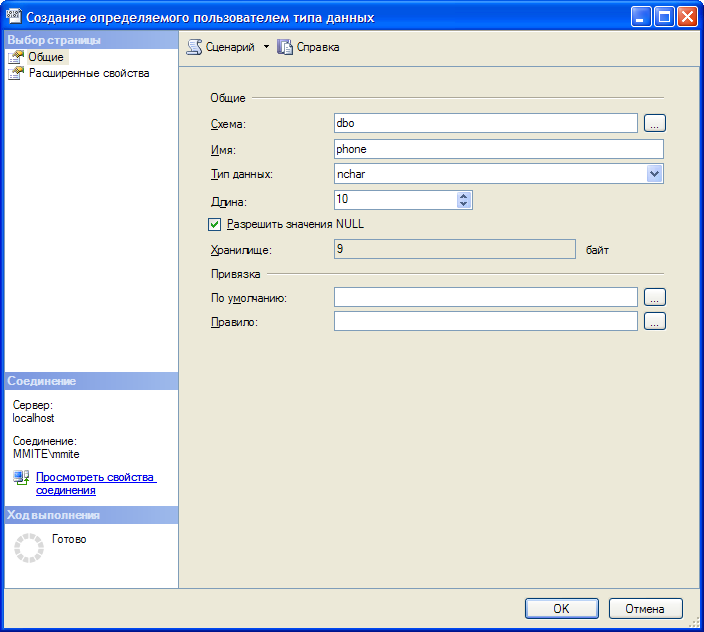
1. Специализированные типы данных

* 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**

SQL Server позволяет на основе системных типов данных создавать пользовательские типы со всеми предварительно заданными параметрами, включая все ограничения и умолчания. В качестве примера создадим тип данных phone, который будет использоваться в таблице Customer для хранения телефонного номера клиента. Для его создания воспользуемся графическим интерфейсом утилиты Management Studio.

1. В дереве обозревателя объектов раскройте папки «Базы данных – Sales – Программирование - Типы». В контекстном меню узла «Определяемые пользователем типы данных» выберите команду «Создать определяемый пользователем тип данных».
2. В появившемся окне в текстовом поле «Имя» введите phone. В раскрывающемся списке «Тип данных» выберите nchar. В качестве длины введите 10. Отметьте параметр «Разрешить значения null», 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.



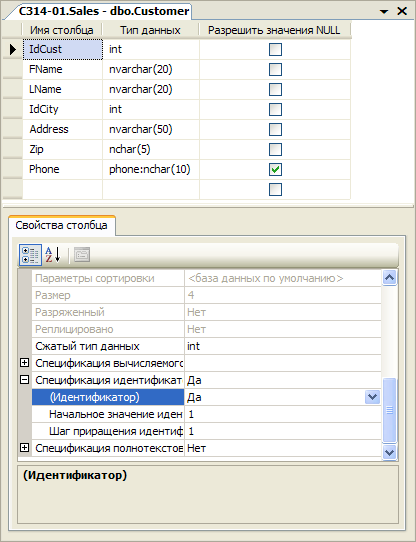
**Создание таблиц**

Создадим в базе данных Sales пять таблицы. 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 |

Таблицы можно создавать как в графическом интерфейсе (в утилите Management Studio), так и с помощью кода T-SQL. Воспользуемся самым простым, графическим способом. Сначала создадим таблицу Customer:

1. В дереве обозревателя объектов в базе данных Sales в контекстном меню узла «Таблицы» выберите команду «Создать таблицу…». В рабочей области должна появиться вкладка с конструктором таблиц.
2. В первую строку в столбце «Имя столбца» введите IdCust, в столбце «Тип данных» выберите int. Убедитесь что параметр «Разрешить значения null» 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.

**Создание ограничений**

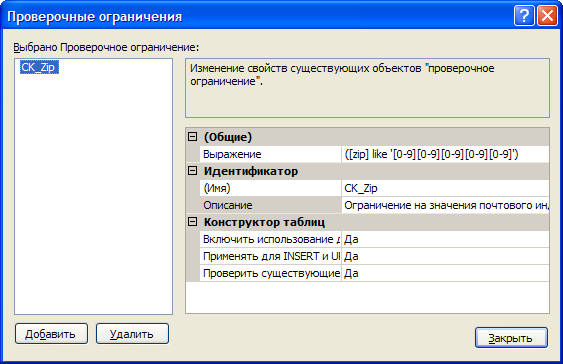
Перед тем как начать работать с таблицами следует ограничить вводимые в них данные в целях обеспечения так называемой целостности данных, т. е. ограничить возникновение в базе данных некорректных или противоречивых данных вследствие добавления, изменения или удаления какой-либо записи, например, ввод отрицательной цены или количества товара. Существует четыре типа целостности данных: доменная, сущностная, ссылочная и пользовательская (или бизнес-правила). Рассмотрим основные инструменты, предоставляемые в SQL Server для их реализации.

**Обеспечение доменной целостности.** Ограничение диапазона данных, вводимых пользователем в поле. Основными инструментами обеспечения доменной целостности являются ограничения проверки и значения по умолчанию.

**Использование проверочных ограничений**

Ограничения на проверку используются для ограничения данных, принимаемых полем, даже если они имеют корректный тип. Например, поле Zip (почтовый индекс) имеет тип nchar(5), т.е. чисто теоретически оно может принимать буквы. Это может стать проблемой, поскольку не существует почтовых индексов с буквами. Рассмотрим, как создать ограничение на проверку, запрещающее вводить в это поле буквы.

1. В контекстном меню папки «Ограничения» таблицы Customer выберите команду «Создать ограничение».
2. В открывшемся окне «Проверочные ограничения» заполните следующие поля:
   * Имя: CK\_Zip
   * Выражение: ([zip] like '[0-9][0-9][0-9][0-9][0-9]'). Данное выражение описывает ограничение, принимающее пять символов, которыми могут быть только цифры от 0 до 9.
   * Описание: Ограничение на значения почтового индекса
3. Щелкните на кнопке «Закрыть» и закройте конструктор таблиц (он был открыт, когда вы начали создавать ограничение) с сохранением изменений.

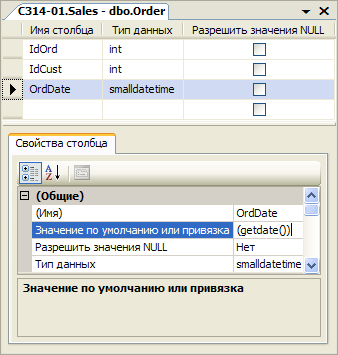


**Задание для самостоятельной работы:** Создайте ограничения для полей InStock таблицы Product и Qty, Price таблицы OrdItem, запрещающие ввод в них отрицательных значений. В данном случае выражение проверки будет иметь вид (Имя поля > 0) для полей Qty, Price и (Instock>=0) для столбца InStock.

**Использование значений по умолчанию**

Установка для полей значений по умолчанию это отличный способ избавить пользователя от излишней работы, если значения этих полей во всех записях, как правило, принимают одни и те же значения. Так в таблице заказов Order вполне логично определить по умолчанию значение поля OrdDate (дата заказа) в виде текущей даты. В этом случае при добавлении записи о новом заказе в случае пропуска этого поля оно будет автоматически заполняться значением системной даты. Для создания такого свойства выполните следующие шаги:

1. Раскройте папку «Столбцы» таблицы Order и в контекстном меню поля «OrdDate» выберите команду «Изменить».
2. В свойстве столбца «Значение или привязка по умолчанию» введите getdate(). Эта функция T-SQL возвращает текущую системную дату.
3. Щелкните на кнопке Сохранить и выйдите из конструктора таблиц.



**Задание для самостоятельной работы:** Установите для поля InStock (количество единиц продукта на складе) таблицы Product в качестве значения по умолчанию ноль.

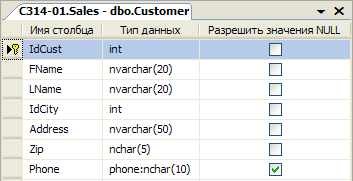
**Обеспечение сущностной целостности**. Обеспечение гарантии уникальности записей в таблицах и предотвращение их дублирования. Основными инструментами обеспечения целостности сущностей являются первичные ключи и ограничения уникальности.

**Создание первичных ключей**

Первичный ключ используется для обеспечения гарантии уникальности каждой записи в таблице. Он состоит из одного (простой ключ) или нескольких (составной ключ) столбцов с гарантированно уникальными значениями для каждой записи таблицы. Если пользователь попытается ввести в поля первичного ключа дублирующее значение будет сгенерирована ошибка и модификация данных будет отменена.

В качестве примера создадим первичный ключ для таблицы Customer. В данном случае идеальным кандидатом на роль первичного ключа выступает столбец IdCust, поскольку значения, содержащиеся в нем, являются уникальными по определению (для него установлено свойство identity). Следует отметить, что в качестве первичного ключа могут быть взяты и реальные атрибуты клиента, например, ИНН, номер страхового свидетельства, серия и номер паспорта вместе взятые (пример составного ключа), но использование различных разновидностей, так называемых, суррогатных ключей (identity, uniqueidentifier) обеспечивает большую степень сущностной целостности (поскольку реальные атрибуты могут все же со временем измениться) и является распространенной практикой. Для создания первичного ключа в таблице Customer выполните следующие шаги:

1. В контекстном меню таблицы Customer выберите команду «Проект».
2. В окне конструктора таблиц щелкните правой кнопкой мыши на поле IdCust и выберите команду «Задать первичный ключ» или нажмите кнопку  на панели инструментов. Обратите внимание на то, что слева от поля IdCust теперь отображается значок ключа, указывающий, что поле является первичным ключом.
3. Закройте конструктор таблиц с сохранением изменений



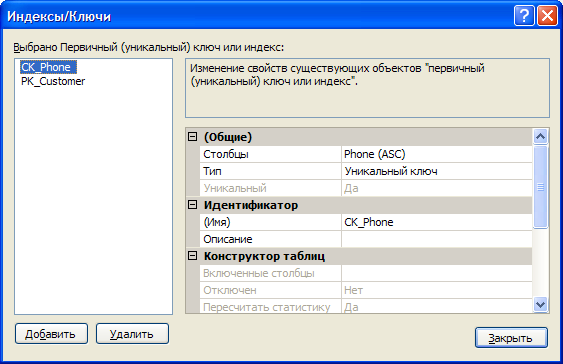
**Задание для самостоятельной работы:** Аналогичным образом создайте первичные ключи для остальных таблиц в соответствие с ниже приведенной таблицей.

|  |  |
| --- | --- |
| Таблица | Первичный ключ |
| City | IdCity |
| Product | IdProd |
| Order | IdOrd |
| OrdItem | IdOrd, IdProd (для выбора нескольких столбцов при установке составного ключа воспользуйтесь клавишами Shift или Ctrl) |

**Использование ограничений на уникальность**

Между ограничениями первичного ключа и ограничениями на уникальность существует два отличия. Первое состоит в том, что первичные ключи используются вместе с внешними ключами для обеспечения целостности ссылок (рассматривается в следующем разделе). Второе отличие заключается в том, что ограничения на уникальность позволяют вставлять в его поля пустые значения (null), чего нельзя делать с первичными ключами. Во всем остальном они служат одной цели – обеспечить уникальность данных, вставляемых в поле. Ограничение на уникальность следует использовать в тех случаях, когда нужно гарантировать, что дублирующие значения не будут добавляться в поле, не являющееся частью первичного ключа, в частности, все потенциальные ключи должны быть организованы в виде ограничений уникальности. Хорошим примером такого поля, требующего ограничение на уникальность, является поле ИНН или серия и номер паспорта, поскольку эти поля должны быть уникальными у каждого человека. Такого идеального кандидата на роль уникального ограничения в нашей таблице Customer нет. Поэтому создадим его по полю Phone, которое также повторяться у разных клиентов не должно.

1. Для открытия конструктора таблиц в контекстном меню таблицы Customer выберите команду «Проект». На панели инструментов нажмите на кнопку «Управление индексами и ключами» .
2. В открывшемся окне «Индексы и ключи» щелкните кнопку «Добавить» и введите следующие параметры для нового уникального ключа:
   * Столбцы: Phone
   * Тип: Уникальный ключ
   * (Имя): CK\_Phone



1. Закройте конструктор таблиц с сохранением изменений.

**Задание для самостоятельной работы:** Аналогичным образом создайте ограничение уникальности по полю CityName таблицы City, чтобы обеспечить отсутствие в справочнике городов с одинаковыми названиями, а также по полю Description таблицы Product, чтобы иметь возможность отличить один товар от другого.

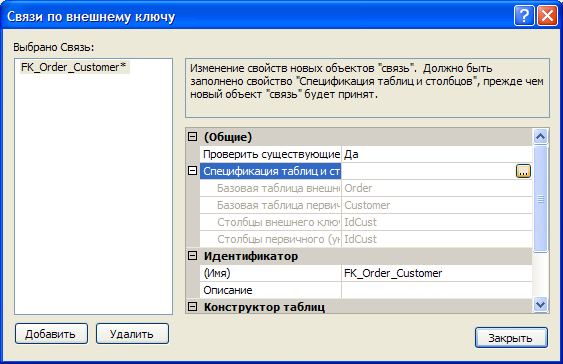
**Обеспечение целостности ссылок**

Сейчас в базе данных Sales имеются пять таблиц, которые тесно взаимосвязаны между собой и соответственно данные содержащиеся в них должны быть согласованы и непротиворечивы. Например, в таблице Order не должно быть записей о заказах для клиента, данные о котором отсутствуют в таблице Customer. Чтобы гарантировать отсутствия в базе данных таких записей необходимо обеспечить целостность ссылок.

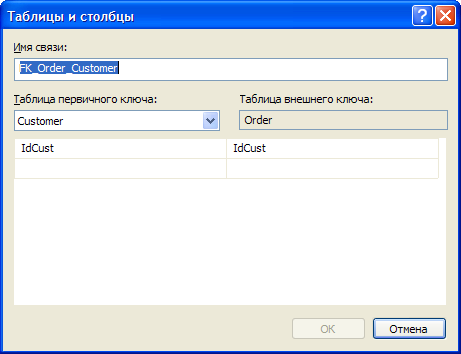
Суть обеспечения целостности ссылок очевидна из названия: данные в одной таблице, ссылающиеся на данные из другой таблицы, защищены от некорректного обновления. В терминологии SQL Server это называется *декларативной ссылочной целостностью* и достигается путем связывания первичного ключа одной из таблиц с внешним ключом другой таблицы (создается так называемое ограничение внешнего ключа).

*Внешний ключ* используется в комбинации с первичным для связывания двух таблиц по общему столбцу (столбцам). К примеру, можно связать таблицы Customer и Order по столбцу IdCust, который присутствует в обеих таблицах. Поскольку поле IdCust таблицы Customer является его первичным ключом можно использовать поле IdCust таблицы Order в качестве внешнего ключа, который свяжет эти две таблицы. После организации такого ограничения будет невозможно добавить запись в таблицу Order, если в таблице Customer нет записи с соответствующим значением IdCust. Кроме того, при отсутствии каскадирования (рассматривается в следующем разделе) невозможно удалить запись из таблицы Customer при наличии связанных с ней записей в таблице Order, поскольку нельзя оставлять заказ без информации о клиенте. Для создания описанного ограничения внешнего ключа в Management Studio выполните следующие шаги:

1. В контекстном меню папки «Ключи» таблицы Order выберите команду «Создать внешний ключ…».



1. В открывшемся окне «Отношения внешнего ключа» заполните следующие поля:
   * (Имя): FK\_Order\_Customer
   * Спецификация таблиц и столбцов: Для заполнения данного блока щелкните на кнопке с многоточием и в появившемся окне «Таблицы и столбцы» в качестве таблицы первичного ключа выберите Customer, а полей связи - IdCust.



1. Закройте все открывшиеся окна с сохранением изменений.

**Использование каскадной ссылочной целостности**

При наличии ограничения внешнего ключа с параметрами по умолчанию вы не можете удалить запись или изменить значение первичного ключа главной таблицы в случае наличия связанных записей в подчиненной таблице (в которой организовано ограничение внешнего ключа). Однако это поведение можно изменить, используя *каскадную ссылочную целостность*.

Настроить правила каскадирования можно при создании ограничения внешнего ключа в окне «Связи по внешнему ключу» изменяя значения параметров «Правило обновления» и «Правило удаления» блока «Спецификация INSERT и UPDATE». Оба этих параметра могут содержать четыре значения, описанные в следующей таблице.

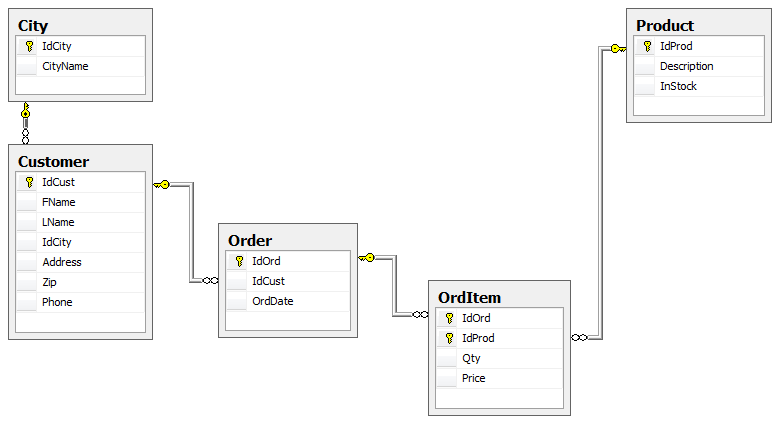
|  |  |  |
| --- | --- | --- |
| Настройка | Правило удаления | Правило обновления |
| Нет действия | Невозможно удалить в главной таблице строку, на которую есть ссылки в подчиненной | Невозможно обновить значения полей первичного ключа главной таблицы при наличии связанных записей в подчиненной |
| Каскадо | При удалении строки в главной таблице все связанные строки в подчиненной также будут удалены | При обновлении значений полей первичного ключа главной таблицы соответствующим образом будут изменены и их значения во всех связанных строках подчиненной таблицы |
| Присвоить Null | При удалении строки в главной таблице во всех связанных строках подчиненной полям вторичного ключа будет присвоено значение Null | При обновлении значений полей первичного ключа главной таблицы во всех связанных строках подчиненной таблицы полям вторичного ключа будет присвоено значение Null |
| Присвоить значение по умолчанию | При удалении строки в главной таблице во всех связанных строках подчиненной полям вторичного ключа будут присвоены значения по умолчанию | При обновлении значений полей первичного ключа главной таблицы во всех связанных строках подчиненной таблицы полям вторичного ключа будут присвоены значения по умолчанию |

**Задание для самостоятельной работы:** Создайте ограничение внешнего ключа FK\_OrdItem\_Order в таблице OrderItem для связи таблиц Order и OrderItem по полю IdOrd. При этом настройте правило каскадного удаления, установив в качестве параметра «Спецификация INSERT и UPDATE\Правило удаления» значение «Каскадно», что приведет к автоматическому удалению всех товаров из заказа при удалении самого заказа.

**Использование диаграмм баз данных**

*Диаграммы базы данных* представляют собой графическое отображение схемы (целиком или частично) базы данных с таблицами и столбцами, а также связей между ними. Создадим диаграмму базы данных:

1. В контекстном меню папки «Диаграммы базы данных» выберите команду «Создать диаграмму базы данных».
2. В диалоговом окне «Добавление таблиц» выберите все таблицы и нажмите на кнопку «Добавить».
3. Добавив таблицы, щелкните на кнопке «Закрыть» и вы увидите созданную диаграмму базы данных (на рисунке представлен окончательный вид диаграммы: некоторые связи у вас могут отсутствовать).



Используя диаграмму базы данных ограничения внешнего ключа можно создавать значительно быстрее: лишь перетаскивая поля из одной таблицы в другую. В качестве примера создадим внешний ключ в таблице Customer по полю IdCity для связи с таблицей City:

1. Выделите в таблице City поле IdCity и, не отпуская кнопку мыши, перетащите его на поле IdCity таблицы Customer.
2. В диалоговых окнах «Таблицы и столбцы» и «Связь по внешнему ключу» примите настройки по умолчанию.
3. Сохраните диаграмму базы данных под именем ILM.
4. Расположите таблицы в канонической форме (главные таблицы выше подчиненных) в соответствии с вышеприведенным рисунком.

**Задание для самостоятельной работы:** Аналогичным образом создайте связь между таблицами Product и OrdItem по полю IdProduct. Окончательный список связей между таблицами со всеми их характеристиками представлен в следующей таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| Главная таблица | Подчиненная таблица | Поле связи (внешний ключ) | Правила каскадирования |
| City | Customer | IdCity | Без действия |
| Customer | Order | IdCust | Без действия |
| Order | OrdItem | IdOrd | Каскадное удаление |
| Product | OrdItem | IdProd | Без действия |

**Задание для самостоятельной работы:** После настройки всех ограничений можно наполнить таблицы данными. Для этого в контекстном меню таблицы выберите команду «Изменить первые 200 строк» и появившейся в рабочей области вкладке введите новые записи, заполняя все необходимые столбцы. В процессе внесения данных проверьте работоспособность всех созданных ранее ограничений:

* Ограничений проверки: попробуйте ввести в поле Zip (почтовый индекс) таблицы Customer нечисловые значения, а в поля InStock таблицы Product и Qty, Price таблицы OrdItem - отрицательные.
* Значений по умолчанию: убедитесь, что при пропуске полей OrdDate и InStock таблиц Order и Product для них устанавливаются значения по умолчанию в виде текущей системной даты и нуля соответственно.
* Ограничений первичного и уникального ключа: попробуйте ввести в таблицы записи с дублирующими значениями первичного или уникального ключа.
* Ограничений внешнего ключа: попробуйте ввести несогласованные данные в связанные таблицы, например, заказ для несуществующего клиента или удалить запись из любой главной таблицы при наличии связанных записей в подчиненной при отсутствии правил каскадирования.
* Правил каскадирования: убедитесь, что при удалении записи из таблицы Order все связанные записи из таблицы OrdItem удаляются автоматически.

**Лабораторная работа №3: Основы Transact SQL: Простые (однотабличные) выборки данных**

SQL — это аббревиатура выражения Structured Query Language (язык структурированных запросов). SQL основывается на реляционной алгебре и специально разработан для взаимодействия с реляционными базами данных.

SQL является, прежде всего, информационно-логическим языком, предназначенным для описания хранимых данных, их извлечения и модификации. SQL не является языком программирования. Вместе с тем конкретные реализации языка, как правило, включают различные процедурные расширения.

Язык SQL представляет собой совокупность операторов, которые можно разделить на четыре группы:

* DDL (Data Definition Language) - операторы определения данных
* DML (Data Manipulation Language) - операторы манипуляции данными
* DCL (Data Control Language) - операторы определения доступа к данным
* TCL (Transaction Control Language) - операторы управления транзакциями

SQL является стандартизированным языком. Стандартный SQL поддерживается комитетом стандартов ANSI (Американский национальный институт стандартов), и соответственно называется ANSI SQL.

Многие разработчики СУБД расширили возможности SQL, введя в язык дополнительные операторы или инструкции. Эти расширения необходимы для выполнения дополнительных функций или для упрощения выполнения определенных операций. И хотя часто они очень полезны, эти расширения привязаны к определенной СУБД и редко поддерживаются более чем одним разработчиком. Все крупные СУБД и даже те, у которых есть собственные расширения, поддерживают ANSI SQL (в большей или меньшей степени). Отдельные же реализации носят собственные имена (PL-SQL, Transact-SQL и т.д.). Transact-SQL (T-SQL) – реализация языка SQL корпорации Microsoft, используемая, в частности, и в SQL Server.

**Запросы на выборку данных (оператор SELECT)**

SELECT – наиболее часто используемый SQL оператор. Он предназначен для выборки информации из таблиц. Чтобы при помощи оператора SELECT извлечь данные из таблицы, нужно указать как минимум две вещи — что вы хотите выбрать и откуда.

**Выборка отдельных столбцов**

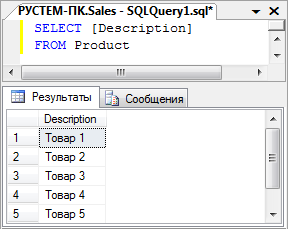
SELECT [Description]

FROM Product

В приведенном выше операторе используется оператор SELECT для выборки одного столбца под названием Description из таблицы Product. Искомое имя столбца указывается сразу после ключевого слова SELECT, а ключевое слово FROM указывает на имя таблицы, из которой выбираются данные.

Для создания и тестирования данного запроса в Management Studio выполните следующие шаги:

1. В контекстном меню базы Sales выберите команду «Создать запрос» или щелкните соответствующую кнопку на панели инструментов .
2. В открывшемся окне создания нового запроса введите представленные выше инструкции SQL.
3. Для запуска запроса на выполнение щелкните кнопку  на панели инструментов или нажмите клавишу F5. В нижней части экрана должны появиться результаты.



1. Management Studio позволяет сохранять пакеты SQL. Это полезно для сохранения сложных запросов, которые будут повторно запускаться в будущем. Для этого щелкните кнопку  на панели инструментов. По умолчанию файлы запросов сохраняются с расширением .sql. В дальнейшем сохраненный запрос может быть открыт командой «Открыть файл».

**Выборка нескольких столбцов**

Для выборки из таблицы нескольких столбцов используется тот же оператор SELECT. Отличие состоит в том, что после ключевого слова SELECT необходимо через запятую указать несколько имен столбцов.

SELECT [Description], InStock

FROM Product

**Выборка всех столбцов**

Помимо возможности осуществлять выборку определенных столбцов (одного или нескольких), при помощи оператора SELECT можно запросить все столбцы, не перечисляя каждый из них. Для этого вместо имен столбцов вставляется групповой символ “звездочка” (\*). Это делается следующим образом.

SELECT \*

FROM Product

**Сортировка данных**

В результате выполнения запроса на выборку данные выводятся в том порядке, в котором они находятся в таблице. Для точной сортировки выбранных при помощи оператора SELECT данных используется предложение ORDER BY. В этом предложении указывается имя одного или нескольких столбцов, по которым необходимо отсортировать результаты. Взгляните на следующий пример.

SELECT IdProd, [Description], InStock

FROM Product

ORDER BY InStock

Это выражение идентично предыдущему, за исключением предложения ORDER BY, которое указывает СУБД отсортировать данные по возрастанию значений столбца InStock.

**Сортировка по нескольким столбцам**

Чтобы осуществить сортировку по нескольким столбцам, просто укажите их имена через запятую. В следующем коде выбираются три столбца, а результат сортируется по двум из них — сначала по количеству, а потом по названию.

SELECT IdProd, [Description], InStock

FROM Product

ORDER BY InStock, [Description]

Важно понимать, что при сортировке по нескольким столбцам порядок сортировки будет таким, который указан в запросе. Другими словами, в примере, приведенном выше, продукция сортируется по столбцу Description, только если существует несколько строк с одинаковыми значениями InStock. Если никакие значения столбца InStock не совпадают, данные по столбцу Description сортироваться не будут.

**Указание направления сортировки**

В предложении ORDER BY можно также использовать порядок сортировки по убыванию. Для этого необходимо указать ключевое слово DESC. В следующем примере продукция сортируется по количеству в убывающем порядке плюс по названию продукта.

SELECT IdProd, [Description], InStock

FROM Product

ORDER BY InStock DESC, [Description]

Ключевое слово DESC применяется только к тому столбцу, после которого оно указано. В предыдущем примере ключевое слово DESC было указано для столбца InStock, но не для Description. Таким образом, столбец InStock отсортирован в порядке убывания, а столбец Description в возрастающем порядке (принятым по умолчанию).

**Фильтрация данных**

В таблицах баз данных обычно содержится много информации и довольно редко возникает необходимость выбирать все строки таблицы. Гораздо чаще бывает нужно извлечь какую-то часть данных таблицы для каких-либо действий или отчетов. Выборка только необходимых данных включает в себя критерий поиска, также известный под названием предложение фильтрации. В операторе SELECT данные фильтруются путем указания критерия поиска в предложении WHERE. Предложение WHERE указывается сразу после названия таблицы (предложения FROM) следующим образом:

SELECT IdProd, [Description], InStock

FROM Product

WHERE InStock = 0

Этот оператор извлекает значения всех столбцов из таблицы товаров, но показывает не все строки, а только те, значение в столбце InStock (Количество товаров на складе) которых равно 0, т.е. только список отсутствующих на складе товаров.

При совместном использовании предложений ORDER BY и WHERE, предложение ORDER BY должно следовать после WHERE.

В предыдущем примере проводилась проверка на равенство, т.е. определялось, содержится ли в столбце указанное значение. SQL поддерживает весь спектр условных (логических) операций, которые приведены в следующей таблице.

|  |  |
| --- | --- |
| Операция | Описание |
| = | Равенство |
| <> | Неравенство |
| != | Неравенство |
| < | Меньше |
| <= | Меньше или равно |
| !< | Не меньше |
| > | Больше |
| >= | Больше или равно |
| !> | Не больше |
| BETWEEN | Между двумя указанными значениями |
| IS NULL | Значение NULL |

В следующем примере осуществляется выборка всех клиентов, для которых не указан контактный телефон.

SELECT FName, LName, Phone

FROM Customer

WHERE PHONE IS NULL

Для поиска диапазона значений можно использовать операцию BETWEEN. Ее синтаксис немного отличается от других операций предложения WHERE, так как для нее требуются два значения: начальное и конечное. Например, операцию BETWEEN можно использовать для поиска товаров, количество которых находится в промежутке между 5 и 10.

SELECT IdProd, [Description], InStock

FROM Product

WHERE InStock BETWEEN 5 AND 10

Для объединения в предложении WHERE нескольких условий необходимо использовать логические операторы AND и (или) OR. Оператор AND требует одновременного выполнения обоих условий. Запишем предыдущий запрос посредством объединения двух операции сравнения оператором AND.

SELECT IdProd, [Description], InStock

FROM Product

WHERE (InStock >= 5) AND (InStock <= 10)

Ключевое слово AND указывает СУБД возвращать только те строки, которые удовлетворяют всем перечисленным критериям отбора. В данном случае будут выбраны только те товары, количество которых находится в промежутке от 5 до 10.

Оператор OR указывает СУБД выбирать только те строки, которые удовлетворяют хотя бы одному из условий.

SELECT IdCity, CityName

FROM City

WHERE (CityName = 'Москва') OR (CityName = 'Казань')

Посредством этого SQL запроса из справочника городов выбираются только Москва и Казань. Ключевое слово OR указывает СУБД использовать какое-то одно условие, а не сразу два. Если бы здесь использовалось ключевое слово AND, мы бы не получили никаких данных.

Если вы внимательно рассмотрите выражение в предыдущем предложении WHERE, то заметите, что значения, с которыми сравниваются названия городов, заключены в одинарные кавычки. Одинарные кавычки используются для определения границ строки (строковой константы). При работе со строковыми константами их всегда необходимо отделять одинарными кавычками.

Предложения WHERE могут содержать любое количество логических операторов AND и OR. Комбинируя их можно создавать сложные фильтры. Однако при комбинировании ключевых слов AND и OR необходимо учитывать, что оператор AND выполняется раньше оператора OR, т.е. имеет более высокий приоритет. Изменить приоритет можно с помощью круглых скобок.

В следующем примере осуществляется выборка из таблицы клиентов всех Ивановых и Петровых, для которых не указан контактный телефон.

SELECT FName, LName, Phone

FROM Customer

WHERE (LName = 'Иванов' OR LName = 'Петров') AND PHONE IS NULL

В случае отсутствия скобок результат был бы не верным, а именно включал бы в себя всех Петровых без контактного телефона и всех Ивановых без каких либо ограничений.

Для определения входит ли сравниваемое значение в определенное заданное множество можно воспользоваться оператором IN. При этом все допустимые значения, заключенные в скобки, перечисляются через запятую. В частности предыдущий пример с использованием оператора IN может быть записан в более компактной форме.

SELECT FName, LName, Phone

FROM Customer

WHERE LName IN ('Иванов','Петров') AND PHONE IS NULL

Для отрицания какого-то условия используется логический оператор NOT. Поскольку NOT никогда не используется сам по себе (а только вместе с другими логическими операторами), его синтаксис немного отличается от синтаксиса остальных операторов. В отличие от них, NOT вставляется перед названием столбца, значения которого нужно отфильтровать, а не после. В следующем примере отбираются все клиенты, для которых имеются сведения об их контактом телефоне.

SELECT FName, LName, Phone

FROM Customer

WHERE NOT PHONE IS NULL

Для фильтрации данных по критерию соответствия определенной символьной строки заданному шаблону используется оператор LIKE. Шаблон может включать обычные символы и символы-шаблоны. Во время сравнения с шаблоном необходимо, чтобы его обычные символы в точности совпадали с символами, указанными в строке. Символы-шаблоны могут совпадать с произвольными элементами символьной строки. Использование символов-шаблонов с оператором LIKE предоставляет больше возможностей, чем использование обычных операторов сравнения. Шаблон может включать в себя следующие символы-шаблоны.

|  |  |  |
| --- | --- | --- |
| Символ-шаблон | Описание | Пример |
| % | Любое количество символов | Инструкция WHERE FName LIKE 'А%' выполняет поиск и выдает всех клиентов, имена которых начинаются на букву ‘А’. |
| \_ | Любой одиночный символ | Инструкция WHERE LName LIKE '\_етров' выполняет поиск и выдает всех клиентов, фамилии которых состоят из шести букв и заканчиваются сочетанием ‘етров’ (Петров, Ветров и т.п.). |
| [] | Любой символ, указанный в квадратных скобках | Инструкция WHERE LName LIKE '[Л-С]омов' выполняет поиск и выдает всех клиентов, фамилии которых заканчиваются на ‘омов’ и начинаются на любую букву в промежутке от ‘Л’ до ‘С’, например Ломов, Ромов, Сомов и т.п. |
| [^] | Любой символ, кроме перечисленных в квадратных скобках | Инструкция WHERE LName LIKE 'ив[^а]%' выполняет поиск и выдает всех клиентов, фамилии которых начинаются на ‘ив’ и третья буква отличается от ‘а’. |

В следующем примере осуществляется выборка всех товаров, названия которых начинаются на букву Т.

SELECT \*

FROM Product

WHERE [Description] LIKE 'Т%'

**Создание вычисляемых полей**

Конструкция SELECT кроме имен столбцов таблиц может также включать так называемые вычисляемые поля. В отличие от всех выбранных нами ранее столбцов, вычисляемых полей на самом деле в таблицах базы данных нет. Они создаются "на лету" SQL-оператором SELECT. Рассмотрим следующий пример.

SELECT IdCust AS 'Номер клиента', FName + ' ' +LName AS 'Фамилия и имя клиента'

FROM Customer

Здесь создается вычисляемое поле, которому с помощью ключевого слова AS дан псевдоним ‘Фамилия и имя клиента’. Оно позволяет объединить (произвести конкатенацию) с помощью оператора + фамилию, пробел и имя клиента в одно поле (столбец). Псевдоним может быть задан и для обычного столбца таблицы. В частности здесь столбцу IdCust задан псевдоним ‘Номер клиента’.

Еще одним способом использования вычисляемых полей является выполнение математических операций над выбранными данными. Рассмотрим пример.

SELECT IdProd, Qty, Price, Qty \* Price AS 'Стоимость'

FROM OrdItem

WHERE IdOrd = 1

Здесь с помощью оператора умножения \* вычисляется общая стоимость каждого товара в заказе с кодом 1 как произведение количества на цену.

**Исключение дублирующих записей**

Для исключения из результата выборки повторяющихся строк используется ключевое слово DISTINCT, которое указывается сразу после SELECT. В следующем примере осуществляется вывод всех фамилий клиентов. Даже если среди них есть однофамильцы, каждая фамилия будет выведена только один раз.

SELECT DISTINCT LName

FROM Customer

**Задание для самостоятельной работы:** Сформулируйте на языке SQL запросы на выборку следующих данных:

* Список всех заказов за определенный период времени (например, сентябрь 2010 года) отсортированный по дате заказа;
* Список всех товаров, названия которых включают слово ‘монитор’ с указанием их остатка на складе.

**Использование агрегатных функций**

В SQL определено множество встроенных функций различных категорий, среди которых особое место занимают агрегатные функции, оперирующие значениями столбцов множества строк и возвращающие одно значение. Аргументами агрегатных функций могут быть как столбцы таблиц, так и результаты выражений над ними. Агрегатные функции и сами могут включаться в другие арифметические выражения. В следующей таблице приведены наиболее часто используемые стандартные унарные агрегатные функции.

|  |  |
| --- | --- |
| **Функция** | **Возвращаемое значение** |
| COUNT | Количество значений в столбце или строк в таблице |
| SUM | Сумма |
| AVG | Среднее |
| MIN | Минимум |
| MAX | Максимум |

Общий формат унарной агрегатной функции следующий:

имя\_функции([АLL | DISTINCT] выражение)

где DISTINCT указывает, что функция должна рассматривать только различные значения аргумента, а ALL — все значения, включая повторяющиеся (этот вариант используется по умолчанию). Например, функция AVG с ключевым словом DISTINCT для строк столбца со значениями 1, 1, 1 и 3 вернет 2, а при наличии ключевого слова ALL вернет 1,5.

Агрегатные функции применяются во фразах SELECT и HAVING. Здесь мы рассмотрим их использование во фразе SELECT. В этом случае выражение в аргументе функции применяется ко всем строкам входной таблицы фразы SELECT. Кроме того, во фразе SELECT нельзя использовать и агрегатные функции, и столбцы таблицы (или выражения с ними) при отсутствии фразы GROUP BY, которую мы рассмотрим в следующем разделе.

Функция COUNT имеет два формата. В первом случае возвращается количество строк входной таблицы, во втором случае — количество значений аргумента во входной таблице:

* COUNT(\*)
* COUNT([DISTINCT | ALL] выражение)

Простейший способ использования этой функции — подсчет количества строк в таблице (всех или удовлетворяющих указанному условию). Для этого используется первый вариант синтаксиса.

**Запрос:** Количество видов продукции, информация о которых имеется в базе данных.

SELECT COUNT(\*) AS 'Количество видов продукции'

FROM Product

Во втором варианте синтаксиса функции COUNT в качестве аргумента может быть использовано имя отдельного столбца. В этом случае подсчитывается количество либо всех значений в этом столбце входной таблицы, либо только неповторяющихся (при использовании ключевого слова DISTINCT).

**Запрос:** Количество различных имен, содержащихся в таблице Customer.

SELECT COUNT(DISTINCT FNAME)

FROM Customer

Использование остальных унарных агрегатных функции аналогично COUNT за тем исключением, что для функций MIN и MAX использование ключевых слов DISTINCT и ALL не имеет смысла. С функциями COUNT, MAX и MIN кроме числовых могут использоваться и символьные поля. Если аргумент агрегатной функции не содержит значений, функция COUNT возвращает 0, а все остальные - значение NULL.

**Запрос:** Дата последнего заказа до 1 сентября 2010 года.

SELECT MAX(OrdDate)

FROM [Order]

WHERE OrdDate<'1.09.2010'

**Задание для самостоятельной работы:** Сформулируйте на языке SQL запросы на выборку следующих данных:

* Суммарная стоимость всех заказов;
* Количество различных городов, содержащихся в таблице Customer.

**Запросы с группировкой строк**

Описанные выше агрегатные функции применялись ко всей таблице. Однако часто при создании отчетов появляется необходимость в формировании промежуточных итоговых значений, то есть относящихся к данным не всей таблицы, а ее частей. Для этого предназначена фраза GROUP BY. Она позволяет все множество строк таблицы разделить на группы по признаку равенства значений одного или нескольких столбцов (и выражений над ними). Фраза GROUP BY должна располагаться вслед за фразой WHERE (если она отсутствует, то за фразой FROM).

При наличии фразы GROUP BY фраза SELECT применяется к каждой группе, сформированной фразой группировки. В этом случае и действие агрегатных функций, указанных во фразе SELECT, будет распространяться не на всю результирующую таблицу, а только на строки в пределах каждой группы. Каждое выражение в списке фразы SELECT должно принимать единственное значение для группы, то есть оно может быть:

* константой;
* агрегатной функцией, которая оперирует всеми значениями аргумента в пределах группы и агрегирует их в одно значение (например, в сумму);
* выражением, идентичным стоящему во фразе GROUP BY;
* выражением, объединяющим приведенные выше варианты.

Самым простым вариантом использования фразы GROUP BY является группировка по значениям одного столбца.

**Запрос:** Количество клиентов по городам.

SELECT IdCity, COUNT(\*) AS 'Кол-во клиентов'

FROM Customer

GROUP BY IdCity

Если в запросе используются фразы и WHERE, и GROUP BY, строки, не удовлетворяющие условию фразы WHERE, исключаются до выполнения группировки. Вследствие этого группировка производится только по тем строкам, которые удовлетворяют условию.

**Запрос:** Количество клиентов по городам с фамилией ‘Иванов’.

SELECT IdCity, COUNT(\*) AS 'Кол-во клиентов'

FROM Customer

WHERE LName = 'Иванов'

GROUP BY IdCity

SQL позволяет группировать строки таблицы и по нескольким столбцам. В этом случае имена столбцов перечисляются во фразе GROUP BY через запятую.

**Запрос:** Количество клиентов по каждой фамилии и имени.

SELECT LName, FName, COUNT(\*)

FROM Customer

GROUP BY LName, FName

Для отбора строк среди полученных групп применяется фраза HAVING. Она играет такую же роль для групп, что и фраза WHERE для исходных таблиц, и может использоваться лишь при наличии фразы GROUP BY. В предложении SELECT фразы WHERE, GROUP BY и HAVING обрабатываются в следующем порядке.

1. Фразой WHERE отбираются строки, удовлетворяющие указанному в ней условию;
2. Фраза GROUP BY группирует отобранные строки;
3. Фразой HAVING отбираются группы, удовлетворяющие указанному в ней условию.

Значение условия, указываемого во фразе HAVING, должно быть уникальным для всех строк каждой группы. Поэтому правила использования имен столбцов и агрегатных функций во фразе HAVING такие же, как и для фразы SELECT при наличии фразы GROUP BY. Это значит, что во фразе HAVING в качестве операндов сравнения можно использовать только группируемые столбцы или агрегатные функции.

**Запрос:** Список городов, количество клиентов из которых больше 10.

SELECT IdCity

FROM Customer

GROUP BY IdCity

HAVING COUNT(\*)>10

**Задание для самостоятельной работы:** Сформулируйте на языке SQL запросы на выборку следующих данных:

* Список всех заказов с указанием их суммарной стоимости;
* Список клиентов, которые за заданный период (например, сентябрь 2010 года) совершили более 3 заказов.