SQL Server - это система управления базами данных (СУБД) от Microsoft. Обычно база данных используется для хранения почти всех данных сайта. Альтернативной базы данных может быть хранение данных в XML файлах.

Что надо знать о базах данных:

- таблицы, столбцы, внешние ключи, первичные ключи, типы данных, индексы, основные типы запросов
- как сделать правильно структуру базы данных
- как создавать таблицы и отношения между ними
- как делать индексы для базы
- как создавать хранимые процедуры

Основные операции с базой производят в Management Studio. Management Studio - это оболочка для работы с SQL Server. Вы можете ее ставить локально или на сервере, где находится сама база.

Задание 1. Установите SQL Server 2008 Express Edition к себе на компьютер с Management Studio Express (можете их скачать с официального сайта Microsoft абсолютно бесплатно).

Чтобы подключиться к SQL Server надо указать имя сервера (для локального сервера - .\sqlexpress), логин и пароль. При установке Management Studio устанавливается локальный SQL Server. Т.е. говоря простым языком, устанавливается 2 программы - SQL Server (это сама СУБД) и Management Studio (средство для взаимодействия с базами данных).
Есть 2 режима аутентификации - SQL Server и Windows (пока выбирайте Windows, чтобы локально зайти на свой сервер). При работе с сайтом на удаленном сервере (VPS) вы будете подключаться в основном по SQL Auth.

В Management Studio Вы должны уметь делать следующие вещи:
- создать таблицу
- создать первичный ключ таблицы
- выполнить запрос
- создать хранимую процедуру
- создать индекс
- создать внешний ключ
- наполнение данными таблиц

Выполнять эти операции можно как через интерфейс Management Studio, так и через SQL.

Задание 2. Найдите оба варианта выполнения основный операций через интерфейс Management Studio и через SQL. Например, для поиска “Как создать таблицу программно” можно написать в Google “SQL Server create table sql”, а через интерфейс - “SQL Server create table via Management Studio”

По сути каждая таблица - это набор из столбцов, первичного и внешних ключей, ограничений, индексов.

Связи между таблицами

Есть таблица Товаров (id, name, price, categoryID) и Категорий (id, name)
Товар может лежать в категории. categoryID - это внешний ключ на таблицу Категории.

В данном примере мы имеем отношение **"один ко многим"**, т.е. одна категория может содержать несколько товаров.

Существует тип отношений **"один к одному"** - редко встречается. Это по сути данные относящиеся к одной сущности, но по каким-то причинам хранящиеся в разных таблицах.
Например, Статьи - вся информация о статье хранится в одной таблице, а сам текст статьи(по соображениям производительности) - в другой.

Также существует тип отношений **"многие ко многим"**. Например, в социальной сети - это группы и люди. Один человек может быть нескольких группах, и группа может содержать нескольких человек.
Чтобы разрешить этот момент в базе создается дополнительная таблица связи с 2 внешними ключам:

```
Users (id, name)
Groups (id, name)
UsersGroups (id, userID, groupID)
```

Внешний ключ в общем случае нужен нам для связи таблиц и для ускорения доступа к данным. Дополнительная функция ключа – обеспечение целостности данных в базе. Т.е. если у вас в таблице есть внешний ключ categoryID – то он может принимать только конкретные значения из другой таблицы, либо быть пустым (null). Желательно делать внешние ключи в одинаковом стиле. Мы делаем это поле всегда заканчивающимся на ID, например typeID. Так сразу понятно, что это внешний ключ на таблицу типов. В нашем случае практически всегда внешний ключ ссылается на первичный ключ с именем id в другой таблице. Так происходит, потому что в наших таблицах всегда стоит такой первичный ключ без исключений. Обязательно проставляйте внешние ключи, если они есть.

Примечание: Чтобы изменить ограничение FOREIGN KEY, его необходимо удалить и повторно создать с новым определением.

Пример создания внешнего ключа при создании таблицы при помощи SQL:

```
CREATE TABLE [dbo].[ad_userSubs](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[userGuid] [uniqueidentifier] NULL,
	[created] [datetime] NOT NULL,
	[posted] [datetime] NULL,
	[filterID] [int] NOT NULL,
 CONSTRAINT [PK_ad_userSubs] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[ad_userSubs]
WITH CHECK ADD  CONSTRAINT [FK_ad_userSubs_ad_userFilters] FOREIGN KEY([filterID])
REFERENCES [dbo].[ad_userFilters] ([id])ON DELETE CASCADE
GO
ALTER TABLE [dbo].[ad_userSubs] CHECK CONSTRAINT [FK_ad_userSubs_ad_userFilters]
GO
```

Здесь создается таблица ad_userSubs, которая содержит поле filterID, являющееся внешним ключом на поле id из таблицы ad_userFilters.

Примечание. Обычно с ключами проще работать через интерфейс Management Studio. Наберите на Youtube “Как создать внешний ключ SQL Server” - так проще и нагляднее.

Во время выбора типа удаления внешнего ключа следует помнить о следующем:

1) каскадное удаление следует ставить тогда, когда дочерняя сущность не имеет смысла без родительской. Например, лог изменения цены товара не имеет смысла без товара.
2) каскадное удаление нельзя ставить тогда, когда дочерняя категория является самостоятельно важной таблицей с данными. Например, товар и товары в корзине, так как при каскадном удалении “товар-товар корзины” будет потеряна вся история о покупках. В этом случае необходимо выбирать Установка в NULL или Не удалять.

Примечание. Мы в дальнейшем будем использовать EntityFramework, он по умолчанию для загрузки связанных записей он использует lazy loading(загрузка связанной записи в отдельном запросе). Запрос в базу - это одна из самых тяжелых операций, их количество надо сводить к минимуму. Для этого нужно за меньшее количество запросов выбрать максимальное количество данных. В конкретном случае можно использовать оператор Include. В его параметрах надо указать, какие связанные записи мы должны загрузить вместе с запросом.

Задание 3. Придумайте по 5 примеров для каждого типа отношений.

Задание 4. Напишите свою инструкцию со скринами как создавать отношение на примере Товара и Категории.

Индексы

Индексы нужны для ускорения извлечения данных. Индекс – это отдельная упорядоченная таблица по определенному полю (или по нескольким полям). Она создается и управляется самим (или самой - кому как нравится) SQL Server - т.е. вам надо правильно его настроить и все - все остальное делает SQL Server самостоятельно.

Общее правило: создавайте индексы на те поля, по которым идет фильтрация (участвует в разделе WHERE оператора SELECT), либо сортировка, либо объединение таблиц. Очень часто это может быть внешним ключом. Внешние ключи – это первый кандидат на создание индекса. Еще момент – индекс замедляет добавление записей (потому что при этом по сути добавляется не одна запись, а две – в таблицу и в индекс). Поэтому если ваши данные очень часто меняются – то возможно вам не нужен индекс по выбранным полям.

Советы:

1. в таблицу лучше всегда помещать первым поле id int (primary key) с автоинкрементом (identity (1,1)). Называйте его всегда однообразно для всех таблиц.
2. внешние ключи всегда именуйте с постфиксом ID, например CategoryID. Так вы сразу будете отличать внешний ключ от простого поля (самое плохое - назвать category - непонятно, это текстовое поле или число?)
3. для внешних ключей правильно используйте правило удаление дочерних записей - обычно имеет смысл ставить null (если не так важна привязка к родительской сущности, например, к категории), либо каскадное удаление (когда дочерние данные удаляются вместе родительским элементов). Правило на обновление не особо актуально для нашего способа разработки, т.к. мы ставим внешний ключ ссылкой на первичный ключ id.

4. не создавайте индекс на строковые поля без сильной необходимости. Это очень увеличивает размер базы.
5. если присутствует сложная бизнес логика - возможно, имеет смысл использовать для нее хранимую процедуру. Это лучше, чем выполнять несколько запросов в C#. Кстати, это способ оптимизации медленных участков кода C# (уменьшить обращение к базе до одного). Если у вас очень большие таблицы и сложная обработка - переносите тяжелые участки кода в SQL. Это не очень красиво с точки зрения архитектуры, но зато работает быстрее. Все-таки больше всего негатива у нас вызывают именно медленно работающие программы (например, Skype на телефоне). Скорость работы приложения - один из ключевых факторов.
6. название таблиц делайте с префиксом подсистемы, например cms_userPosts.
7. очень важно соблюдать всегда одну и ту же нотацию. Придерживайтесь всегда одного и того же стиля наименования таблиц и столбцов. Необдуманно криво названное поле будет тянуться по всему проекту. Поэтому не допускайте халатности в этом отношении
8. Как мы именуем переменные:
    a. таблицы префикс_названиеНазваниеНазвание
    b. для столбцов - просто с маленькой без префикса. Для внешних ключей приставка ID (categoryID)
    c. для хранимых процедур - префикс_действиеОбъект. Т.е старайтесь, чтобы из названия хранимой процедуры было понятно, что она делает (именно в форме действия - важен глагол).

Типы данных

Основные типы:
- Для строк лучше всего использовать nvarchar (ни в коем случае не используйте nchar, при этом длина строки строго зафиксирована и не зависит от содержимого - т.е. сложно потом будет сравнивать строки, т.к. они будут дополняться ненужными нам пробелами). Если поле очень большое, то лучше использовать nvarchar(MAX) или text, но при этом размер должен быть адекватным, чтобы не замедлять работу в дальнейшем.
- для чисел используйте int, float. Важный момент - не нужно всех поражать своим знанием типов и использовать long там, где можно использовать int. Это в дальнейшем немного усложнит обработку таких значений в C# (т.е. по возможности не удивляйте своих коллег такими моментами, лучше удивите их быстрыми запросами SQL).
- Булевский тип - bit.
- Деньги храните либо в типе money, либо в decimal (18,2), либо в банке.

- Дата и время – тип datetime. Важный момент – изучите различные функции работы с датами (getdate, datediff, dateadd и др) – это будет часто встречаться.
- Очень важный тип – это uniqueidentifier. Это GUID – уникальный 32-битный код. Причем никакие два компьютера во всем мире не создадут два одинаковых значения GUID (вернее вероятность такого критически мала). Идентификаторы GUID в первую очередь используются для назначения идентификаторов, которые должны быть уникальными в рамках сети, содержащей много компьютеров в различных расположениях. Значение идентификатора GUID для столбца uniqueidentifier формируется с помощью функции newid().

Задание 5. Создайте на SQL такой скрипт, который запишет в таблицу 10 млн записей с Guid. Через запрос проверьте, что в таблице нет дубликатов Guid (подсказка используйте Group by).

Задание 6. Изучите методы работы со строками в SQL

Иногда возникает необходимость поменять тип в существующей таблице. При этом поменять его при помощи визуальных средств редактора проблематично. В таких случаях это можно сделать при помощи SQL:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype
```

Задание 7. Разобраться самостоятельно, что такое курсоры и кратко описать + сделать пример использования.

Примечание. По возможности избегайте работы с курсорами, старайтесь обходиться простыми запросами.

Хранимые процедуры

Для ускорения работы с данными рационально создавать и использовать хранимые процедуры. Для это существует несколько причин.

- Во-первых, SQL Server парсит, оптимизирует и компилирует хранимые процедуры, поэтому они выполняются быстро без необходимости повторять эти шаги каждый раз.
- Во-вторых, они выполняются на стороне базы данных, а не в коде приложения. Т.е. минимум тратится ресурсов на дополнительные обращения к базе данных.
- В-третьих, использование хранимых процедур позволяет сэкономить траффик, так как клиент посылает серверу только запрос, сервер его обрабатывает и возвращает только результат, который обычно значительно меньше, чем полный набор данных.

Рассмотрим пример создания процедуры:

```
CREATE PROCEDURE [dbo].[GetFilterDemoTable]
@orderNum nvarchar(64) = '',
@clientID int = 0,
@createdMin datetime,
@createdMax datetime,
@statusIDs nvarchar(256) = '',
@sort1 nvarchar(20) ='',
@direction1 nvarchar(5) ='',
@page int = 1,
@pageSize int = 10,
@total int output
AS
DECLARE @statuses TABLE(itemID int);
insert  into @statuses
 SELECT cast(Value as int) FROM [dbo].[split] (@statusIDs,',')

 declare @tbl table (id int, orderNum nvarchar(50), statusID int, statusName nvarchar(64),
       addedBy nvarchar(64), clientID int, clientName nvarchar(128), [creat] varchar(10))

-- фильтрация
insert into @tbl
 SELECT    SelOrd.id,  SelOrd.orderNum,  SelOrd.statusID,  (select  name  from  crm_orderStatuses  where
id=SelOrd.statusID) As statusName,
 SelOrd.addedBy,  SelOrd.clientID,    (select  fio  from  crm_clients  where  id=clientID)  clientName,
convert(varchar(10),SelOrd.[created],102) As [creat]
  from crm_orders as SelOrd
  where
  (@clientID=0 or SelOrd.clientID = @clientID) and
  (@createdMin <= SelOrd.created and SelOrd.created<=@createdMax) and
  ( @statusIDs='' or SelOrd.statusID in (select itemID from @statuses) ) and
  (@orderNum='' or SelOrd.orderNum like '%'+@orderNum+'%')

set @total = @@ROWCOUNT
 -- форматирование результата + пагинация + сортировка
select * from(
 select  tb.id,  tb.orderNum,  tb.statusID,  tb.statusName,  tb.addedBy,  tb.clientID,    tb.clientName,
convert(varchar(10),convert(date,tb.[creat]),104) As [created], ROW_NUMBER()
 OVER (
       order by
         CASE WHEN @direction1 = 'up' THEN
        CASE
           WHEN @sort1 = 'orderNum' THEN orderNum
           WHEN @sort1 = 'statusName' THEN statusName
                  WHEN @sort1 = 'clientName' THEN clientName
                  else tb.[creat]
        END
```

```
      END ASC
    , CASE WHEN @direction1 = 'down' or @direction1 = '' THEN
        CASE
           WHEN @sort1 = 'orderNum' THEN orderNum
           WHEN @sort1 = 'statusName' THEN statusName
                 WHEN @sort1 = 'clientName' THEN clientName
                 else tb.[creat]
        END
    END DESC
 )  as number
 from @tbl as tb
 ) as ptbl
 WHERE number BETWEEN ((@page - 1) * @pageSize + 1) AND (@page * @pageSize)
```

В приведенном примере передается список параметров:

```
@orderNum nvarchar(64) = '',
@clientID int = 0,
@createdMin datetime,
@createdMax datetime,
@statusIDs nvarchar(256) = '',
@sort1 nvarchar(20) ='',
@direction1 nvarchar(5) ='',
@page int = 1,
@pageSize int = 10,
@total int output
```

Параметры следуют после названия процедуры через запятую, именуются, начиная с символа @, для каждого параметра нужно указать тип и можно указать значение по умолчанию. Если параметр возвращает значение, следует добавить ключевое слово OUTPUT(@total int output).

Если процедура возвращает набор структурированных данных, имеет смысл создать внутри временную таблицу, поместить результат в нее и вернуть полученные данные на выходе. В нашем примере создается временная таблица следующим образом:

```
declare @tbl table (id int, orderNum nvarchar(50), statusID int, statusName nvarchar(64),
       addedBy nvarchar(64), clientID int, clientName nvarchar(128), [creat] varchar(10))
```

Далее в нее помещаются отфильтрованные данные при помощи инструкции

```
insert into @tbl SELECT
```

Внутри хранимой процедуры можно создать параметр нужного типа и присвоить ему значение:

```
DECLARE @statuses TABLE(itemID int);
set @total = @@ROWCOUNT
```

Если процедура содержит выходной параметр, то при вызове процедуры в запросе его значение можно получить при помощи оператора печати PRINT:

```
CREATE PROCEDURE spGetEmployeeCountByGender
@Gender nvarchar(20),
@EmployeeCount int Output
AS
BEGIN
SELECT @EmployeeCount = COUNT(Id)
FROM tblEmployee
WHERE Gender = @Gender
END

DECLARE @EmployeeTotal int
EXECUTE spGetEmployeeCountByGender 'Female', @EmployeeTotal output
PRINT @EmployeeTotal
```

Если процедура возвращает набор строк, то нужно выполнить следующий select запрос:

```
SELECT sp.Column1,sp.Column2,sp.Column3
FROM sp_name (any Argumnets) sp
```

То есть имеется процедура sp_name с параметрами any_Arguments, которая в качестве результата возвращает набор строк как минимум с такими полями, как Colunm1, Colunm2, Colunm3.

Полезно знать о некоторых важных системных процедурах. А именно:

- sp_help SP_Name : используется для получения информации о названиях параметров процедуры, их типах и т.д. Эта процедура может быть применена к любому объекту БД(таблица, триггер и т.п.)
- sp_helptext SP_Name : используется для получения текста хранимой процедуры

Задание 8. Создать структуру базы для своего проекта, то есть проекта, который Вы будете разрабатывать в течение курса для себя. Структура базы должны содержать набор таблиц и связей между ними (у таблицы обозначаем первичный ключ (PK), внешние ключи (FK) и у каждого поля прописываем тип).

Задание 9 (для продвинутых). Для автоматизации гостиничного бизнеса (бронирование и обслуживание номеров) составьте базу для сети отелей. Напишите краткое описание созданной базы.

Примечание. В курсе Extend мы рассматриваем некоторые моменты по оптимизации базы данных. Их несложно самостоятельно найти в Google. К примеру есть такие специальные запросы, которые позволят отследить самые ресурсоемкие операции на базе (по нагрузке процессора и по размеру требуемой памяти на запрос).

Также изучите, что такое план выполнения запроса. SQL Server позволяет вам узнать как именно запрос выполнялся и где именно возникают основные задержки по запросу. Особенно это критично при работе со сложными запросами, где сложно сразу сказать о причине “медлительности” запроса.

Ссылки:

1. Учебник по MS Management Studio - http://technet.microsoft.com/ru-ru/library/bb934498.aspx
2. SQL примеры - http://technet.microsoft.com/en-us/library/ms187731.aspx
3. Создание хранимых процедур и функций - http://youtu.be/ZbK3jI-q7Ts
4. Skype-чат для проработки вопросов - https://join.skype.com/lDicwkDpdeDu