Как обычно начнем без раскачки.

SQL - язык запросов для управления данными в БД. Критически важно владеть этим гибким инструментом извлечения данных из базы. Можно делать очень сложные выборки данных. Проводить анализ в разных разрезах. Именно в этом сила автоматизации по сравнению с Excel или ручной обработкой.

Основное что Вы должны знать в SQL:
1. извлечение данных из базы - работа с SELECT запросами (использование констукций ORDER BY, GROUP BY, агрегирующих функций, подзапросов, INNER JOIN, Union, Having)
2. манипуляция данными:
   - вставка - INSERT
   - удаление - DELETE (для быстрой очистки таблицы можно использовать TRANCATE TABLE).
   - обновление - UPDATE

Оператор SELECT

Этот оператор используется для извлечения записей из одной или нескольких таблиц в базе данных SQL Server.
Простейшая форма оператора SELECT:

```
SELECT {выражение}
FROM {список таблиц}
WHERE {условия};
```

```
Полная форма оператора:
SELECT [ ALL | DISTINCT ]
[ TOP (top_value) [ PERCENT ] [ WITH TIES ] ]
{выражение}
FROM {условия}
WHERE conditions
[ GROUP BY expressions ]
[ HAVING condition ]
[ ORDER BY expression [ ASC | DESC ];
```

Используемые параметры и аргументы:
ALL
Необязательный аргумент. Возвращать все подходящие записи
DISTINCT
Необязательный аргумент. Удалять дубликаты из результирующего набора.
TOP (top_value)
Необязательный аргумент. Если аргумент указан, запрос вернет верхнее количество строк результирующего набора на основе *top_value*. TOP(10) вернет первые верхние 10 строк из результирующего набора.

PERCENT
Необязательный аргумент. Если указан, запрос вернет верхнее количество строк на основе значения переданного процента от общего числа строк, равного 100%.
WITH TIES
Необязательный аргумент. Если указан, будут возвращены дополнительные строки из результирующего набора, которые соответствуют последнему значению из TOP на основе ORDER BY. Аргумент может быть указан только в SELECT выражениях и только если указано значение сортировки ORDER BY. Например, пусть результирующим набором без учета TOP у нас является: 1, 1, 1, 1, 2,2,2,3. Запрос имеет такую форму:

```
SELECT TOP(5) WITH TIES column1 from {table}
ORDER BY column1
```

Итоговый результат будет дополнен всеми значениями “2”: 1, 1, 1, 1, 2,2,2

{выражение}
Колонки или вычисления, которые вы хотите вернуть в результате.
{таблицы}
Таблицы, из которых вы хотите вернуть записи. Как минимум одна таблица должна быть указана в пункте FROM.
{условия}
Список условий, которым должны удовлетворять выбранные в результате записи.
GROUP BY
Необязательный аргумент. Собирает результаты в группы по одной или нескольким колонкам.
HAVING
Необязательный аргумент. Используется в комбинации с GROUP BY, чтобы ограничить группы возвращаемых строк только теми, для которых условие Истина.
ORDER BY
Необязательный аргумент. Используется для сортировки записей в результирующем наборе.

Манипуляция с данными

Язык обработки данных DML - инструкции предназначены для добавления данных, изменения данных, запроса данных и удаления данных из базы данных SQL Server.

UPDATE изменяет существующие данные в таблице. Общий синтаксис:

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

DELETE удаляет одну или несколько строк из таблицы или представления. Общий синтаксис:

```
DELETE FROM table_name
WHERE some_column=some_value;
```

TRUNCATE TABLE удаляет все строки в таблице, не записывая в журнал удаление отдельных строк. Инструкция TRUNCATE TABLE похожа на инструкцию DELETE без предложения WHERE, однако TRUNCATE TABLE выполняется быстрее и требует меньших ресурсов системы и журналов транзакций. Синтаксис:
TRUNCATE TABLE table_name

INSERT добавляет одну или несколько строк в таблицу или представление. Существует две формы использования оператора INSERT. Первая не содержит определение имен колонок, куда будут вставлены данные, а только их значения, подразумевается, что порядок передачи данных будет соответствовать определению колонок в БД, иначе возникнет ошибка:

```
INSERT INTO table_name
VALUES (value1,value2,value3,...);
Вторая форма определяет название колонок, куда будут вставлены данные:
INSERT INTO table_name (column1,column2,column3,...)
VALUES (value1,value2,value3,...);
```

Задание 1. Изучить оператор INSERT INTO SELECT и его использование, привести пример собственного созданного запроса для данных учебной БД.

Примечание. Инструкция INSERT INTO <целевая_таблица> SELECT <столбцы> FROM <исходная_таблица> может эффективно перенести большое количество строк из одной таблицы (например, промежуточной) в другую таблицу с минимальным протоколированием.

Задание 2(для продвинутых). Попробовать при помощи простого SELECT вставить миллион записей в таблицу базы в цикле. Проанализировать, насколько это быстро работает. Самостоятельно найти решение, как это сделать гораздо быстрее.

В крайних случаях, когда сложно обойтись одними запросами, можно использовать курсоры (это, по сути, проход по таблице в цикле). Однако по возможности старайтесь избегать работы с курсорами.

Подзапросы

Очень эффективное средство - это подзапросы. С помощью подзапросов можно очень гибко формировать отчеты.

Вложенным запросом(подзапросом) называется запрос, помещаемый в инструкцию SELECT, INSERT, UPDATE или DELETE или в другой вложенный запрос. Подзапрос может быть использован везде, где разрешены выражения.

Простой подзапрос - извлекаем всех сотрудников из IT и HR отделов:

```
select * from users where depID in (select id from departments where code='hr' or code='it')
```

Вложенный во внешнюю инструкцию SELECT запрос, имеет следующие компоненты:

- обычный запрос SELECT, включающий обычные компоненты списка выборки;
- обычное предложение FROM, включающее одно или более имен таблиц или представлений;
- необязательное предложение WHERE;
- необязательное предложение GROUP BY;
- необязательное предложение HAVING.

Запрос SELECT вложенного запроса всегда заключен в скобки. Он может включать предложение ORDER BY только вместе с предложением TOP.

Вложенный запрос может быть вложен в предложение WHERE или HAVING внешней инструкции SELECT, INSERT, UPDATE или DELETE или в другой вложенный запрос. Подзапрос может появляться везде, где может использоваться выражение, если он возвращает одно значение.
В следующем примере удваивается значение столбца ListPrice таблицы Production.Product. Вложенный запрос в предложении WHERE ссылается на таблицуPurchasing.ProductVendor для ограничения количества обновляемых строк таблицы Product только теми, у которых BusinessEntity равно 1540.

```
UPDATE Production.Product
SET ListPrice = ListPrice * 2
WHERE ProductID IN
    (SELECT ProductID
     FROM Purchasing.ProductVendor
     WHERE BusinessEntityID = 1540);
```

Вложенные запросы, начинающиеся с операторов сравнения, часто включают агрегатные функции, потому что они возвращают одиночное значение. Например, следующая инструкция находит названия всех продуктов, у которых цена по прейскуранту больше, чем средняя цена по прейскуранту.

```
SELECT                                                    Name
FROM                                                      Production.Product
WHERE                                          ListPrice                      >
    (SELECT                                     AVG                           (ListPrice)
     FROM Production.Product)
```

Поскольку вложенные запросы, начинающиеся с немодифицированных операторов сравнения, должны возвращать одиночное значение, они не могут включать предложения GROUP BY или HAVING (за исключением случаев, когда достоверно известно, что предложение GROUP BY или HAVING возвратит одиночное значение). Например, следующий запрос находит продукты, оцененные выше, чем самый дешевый продукт, который находится в подкатегории 14.

```
SELECT                                                    Name
FROM                                                      Production.Product
WHERE                                          ListPrice                      >
    (SELECT                                     MIN                           (ListPrice)
     FROM                                                 Production.Product
     GROUP                                  BY                               ProductSubcategoryID
     HAVING ProductSubcategoryID = 14)
```

Примечание. В некоторых случаях для оптимизации лучше использовать Join, а в некоторых подзапросы. Пробуйте по-разному и смотрите время выполнения запроса.

Еще момент: вы можете эффективно использовать подзапросы для получения различных данных по каким то сущностям, например по пользователям

```
SELECT Name,
(select ...) sum,
(select ...) manager,
(select ...) orders
FROM                                                      Users
```

Соединения

С помощью соединений можно получать данные из двух или нескольких таблиц на основе логических связей между ними. Соединения можно разделить на следующие категории.
1.Внутренние соединения используют оператор сравнения для установки соответствия строк из двух таблиц на основе значений общих столбцов в каждой таблице. Примером может

быть получение всех строк, в которых идентификационный номер студента одинаковый как в таблице students, так и в таблице courses.

Примером внутреннего соединения:

```
SELECT                                                                  *
FROM                    HumanResources.Employee                 AS      e
    INNER               JOIN                Person.Person           AS      p
    ON                  e.BusinessEntityID              =           p.BusinessEntityID
ORDER                                       BY                              p.LastName
```

Такое внутреннее соединение называется эквисоединением. При таком соединении сохраняются все столбцы обеих таблиц и только те строки, для которых в соединяющем столбце имеется равное значение.

2.Внешние соединения. Внешние соединения бывают левыми, правыми и полными.

2.1.LEFT JOIN или LEFT OUTER JOIN

Результирующий набор левого внешнего соединения включает все строки из левой таблицы, заданной в предложении LEFT OUTER, а не только те, в которых соединяемые столбцы соответствуют друг другу. Если строка в левой таблице не имеет совпадающей строки в правой таблице, результирующий набор строк содержит значения NULL для всех столбцов списка выбора из правой таблицы.

Рассмотрим соединение таблиц Product и ProductReview по столбцам ProductID. В результате будут выведены только те продукты, для которых были написаны обзоры. Чтобы включить в результаты все продукты, независимо от того, были ли написаны обзоры, используйте левое внешнее соединение ISO. Пример запроса:

```
SELECT                              p.Name,                         pr.ProductReviewID
FROM                                  Production.Product                            p
LEFT            OUTER               JOIN            Production.ProductReview      pr
ON                      p.ProductID                     =                   pr.ProductID
```

Ключевые слова LEFT OUTER JOIN включают в вывод все строки таблицы Product независимо от того, есть ли для них соответствующие значения в столбцеProductID таблицы ProductReview. Обратите внимание на то, что в результатах, где для продукта нет соответствующего обзора, строки содержат значение NULL в столбце ProductReviewID.

2.2. RIGHT JOIN или RIGHT OUTER JOIN

Правое внешнее соединение является обратным для левого внешнего соединения. Возвращаются все строки правой таблицы. Для левой таблицы возвращаются значения NULL каждый раз, когда строка правой таблицы не имеет совпадающей строки в левой таблице.

Рассмотрим соединение таблиц SalesTerritory и SalesPerson по столбцам TerritoryID. В результате будут выведены все территории, которым был назначен менеджер по продажам. Оператор правого внешнего соединения ISO, RIGHT OUTER JOIN, включает в результаты все строки второй таблицы независимо от того, есть ли для них совпадающие данные в первой таблице.

Чтобы включить в результаты всех менеджеров по продажам независимо от того, есть ли связанные с ними территории, используйте правое внешнее соединение ISO. Пример запроса Transact-SQL и результаты правого внешнего соединения:

```
SELECT st.Name AS Territory, sp.BusinessEntityID
FROM Sales.SalesTerritory st
RIGHT OUTER JOIN Sales.SalesPerson sp
ON st.TerritoryID = sp.TerritoryID ;
```

2.3. FULL JOIN или FULL OUTER JOIN

Полное внешнее соединение возвращает все строки из правой и левой таблицы. Каждый раз, когда строка не имеет соответствия в другой таблице, столбцы списка выбора другой таблицы содержат значения NULL. Если между таблицами имеется соответствие, вся строка результирующего набора содержит значения данных из базовых таблиц.

Рассмотрим соединение таблиц Product и SalesOrderDetail по столбцам ProductID. В результате будут показаны только те продукты, на которые есть заказы. Оператор полного внешнего соединения ISO, FULL OUTER JOIN, включает в результаты все строки из обеих таблиц независимо от того, есть ли для них совпадающие данные в другой таблице.

В запрос с полным внешним соединением можно включить предложение WHERE и получить только те строки, с которыми не совпадают никакие строки в другой таблице. Следующий запрос вернет только те продукты, на которые нет заказов, а также те заказы, которым не соответствуют продукты (хотя в этом случае всем заказам соответствует какой-то продукт).

```
SELECT p.Name, sod.SalesOrderID
FROM Production.Product p
```

```
FULL OUTER JOIN Sales.SalesOrderDetail sod
ON p.ProductID = sod.ProductID
WHERE p.ProductID IS NULL
OR sod.ProductID IS NULL
ORDER BY p.Name ;
```

3.Перекрестные с соединения

Перекрестное соединение возвращает все строки из левой таблицы. Каждая строка из левой таблицы соединяется со всеми строками из правой таблицы. Перекрестные соединения называются также декартовым произведением.

Примечание. Если вы только начинаете работать с SQL - используйте Inner Join - в большинстве случаев это лучше всего подходит. Обычно объединение идет по внешнему ключу, поэтому значение null не должно появляться в таблице - т.е. имеет смысл использовать именно Inner join.
В плане оптимизации Inner join наиболее предпочтительный элемент.

Объединения

Объединяет результаты двух или более запросов в один результирующий набор, в который входят все строки, принадлежащие всем запросам в объединении. Операция UNION отличается от соединений столбцов из двух таблиц.
Ниже приведены основные правила объединения результирующих наборов двух запросов с помощью операции UNION:

- Количество и порядок столбцов должны быть одинаковыми во всех запросах.
- Типы данных должны быть совместимыми.

UNION
Указывает на то, что несколько результирующих наборов следует объединить и возвратить в виде единого результирующего набора.
UNION ALL
Объединяет в результирующий набор все строки. Это относится и к дублирующимся строкам. Если опущено, дубликаты строк удаляются.

При выполнении следующего примера в результирующий набор включается содержимое столбцовProductModelID и Name таблиц ProductModel и Gloves.

```
SELECT ProductModelID, Name
```

```
FROM                                                    Production.ProductModel
WHERE          ProductModelID          NOT          IN          (3,          4)
UNION
SELECT                                 ProductModelID,                        Name
FROM                                                                   dbo.Gloves
ORDER                                      BY                             Name;
```

Подробнее можно ознакомиться по ссылкам:
http://msdn.microsoft.com/ru-ru/library/bb510741.aspx
http://www.sql.ru/docs/mssql/tsql_ref/
http://ru.wikipedia.org/wiki/Transact-SQL

Пример. Есть задача вывести только первых 3 пользователей из базы, имя которых начинается с буквы А.

Знаете как сделать?
2 мин на размышление.

Сделали?
Честно говоря, когда я писал пример, я тоже не знал ответ)
Что делать, если Вы не знаете как реализовать? Просто напишите правильный запрос в Google:

```
SQL Server select first 3 record from table
```

и второй запрос:

```
SQL Server where column start with letter
```

Выполните эти 2 поиска, в первую очередь, обращая внимание на stackoverflow.com и будет вам счастье.

Функции и операторы SQL Server

В SQL есть дополнительные функции работы со строками и датами. Осторожно используйте их с большими массивами данных - это сильно замедляет работу операторов select.
Опять же нет смысла знать все функции - Вы их всегда можете найти.
Самые популярные функции имеет смысл превращать в свои функции. Например, функция split, которая превращает строку "aaa,bbb,ccc" в таблицу с 3 строками.

Рассмотрим некоторые функции более подробно.
1)Функции работы с датой

GETDATE ( ) - Возвращает значение типаdatetime, которое содержит дату и время компьютера, на котором запущен экземпляр SQL Server.
DAY | MONTH | YEAR (*date*) - Возвращает целое число, представляющее день|месяц|год указанной даты *date*.
DATEDIFF (*datepart*,*startdate* ,*enddate*) - Возвращает количество границ*datepart* даты или времени, пересекающихся между двумя указанными датами.
Вывод форматированной даты:
Для вывода даты в формате DD.MM.YYYY

```
SomeDate = CONVERT(VarChar(50), getdate(), 104)
```

Выведется текущая дата в указанном формате. Формат указывается цифрами, в данном случае 104.
Существуют и другие форматы.

2)Функции работы со строками
CONCAT (string_value1, string_value2 [, string_valueN]) - Возвращает строку, являющуюся результатом объединения двух или более строковых значений.
LEN (string_expression) - Возвращает количество символов указанного строкового выражения, исключая конечные пробелы.

3)Конвертация типов
В Sql Server возможна конвертация строки в число при помощи двух стандартных функций:
CAST (expression AS data_type [(length)])
CONVERT (data_type [(length)] , expression [, style])
Верное использование:

```
select CONVERT(int, '12356')
select CAST('123' as int)
Вызов, приводящий к генерированию ошибок:
select CONVERT(int, '12356.23')
select CAST('123fgfg' as int)
```

4)Оператор выбора case when
Служит для оценки списка условий и возвращения одного из нескольких возможных выражений результатов.
Выражение CASE имеет два формата:

- простое выражение CASE для определения результата сравнивает выражение с набором простых выражений;
- поисковое выражение CASE для определения результата вычисляет набор логических выражений.

Оба формата поддерживают дополнительный аргумент ELSE.
Выражение CASE может использоваться в любой инструкции или предложении, которые допускают допустимые выражения. Например, выражение CASE можно использовать в таких инструкциях, как SELECT, UPDATE, DELETE и SET, а также в таких предложениях, как select_list, IN, WHERE, ORDER BY и HAVING.

Simple CASE expression:

```
CASE input_expression
     WHEN when_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
Searched CASE expression:
CASE
     WHEN Boolean_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```

Использование инструкции SELECT с простым выражением CASE:

```
SELECT   ProductNumber, Category =
       CASE ProductLine
       WHEN 'R' THEN 'Road'
       WHEN 'M' THEN 'Mountain'
       WHEN 'T' THEN 'Touring'
       WHEN 'S' THEN 'Other sale items'
       ELSE 'Not for sale'
       END,
   Name
FROM Production.Product
ORDER BY ProductNumber;
```

Использование инструкции SELECT с поисковым выражением CASE:

```
SELECT   ProductNumber, Name, "Price Range" =
      CASE
       WHEN ListPrice =  0 THEN 'Mfg item - not for resale'
       WHEN ListPrice < 50 THEN 'Under $50'
       WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Under $250'
       WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Under $1000'
       ELSE 'Over $1000'
      END
FROM Production.Product
ORDER BY ProductNumber ;
```

Использование выражения CASE в предложении ORDER BY:

```
SELECT BusinessEntityID, SalariedFlag
```

```
FROM HumanResources.Employee
ORDER BY CASE SalariedFlag WHEN 1 THEN BusinessEntityID END DESC
       ,CASE WHEN SalariedFlag = 0 THEN BusinessEntityID END;
```

Использование выражения CASE в предложении HAVING:

```
SELECT JobTitle, MAX(ph1.Rate)AS MaximumRate
FROM HumanResources.Employee AS e
JOIN HumanResources.EmployeePayHistory AS ph1 ON e.BusinessEntityID = ph1.BusinessEntityID
GROUP BY JobTitle
HAVING (MAX(CASE WHEN Gender = 'M'
       THEN ph1.Rate
       ELSE NULL END) > 40.00
     OR MAX(CASE WHEN Gender  = 'F'
       THEN ph1.Rate
       ELSE NULL END) > 42.00)
ORDER BY MaximumRate DESC;
```

5)Оператор LIKE

Синтаксис:

match_expression [NOT] LIKE pattern [ESCAPE escape_character]

Применение оператора LIKE с использованием шаблона %:

```
SELECT p.FirstName, p.LastName, ph.PhoneNumber
FROM Person.PersonPhone AS ph
INNER JOIN Person.Person AS p
ON ph.BusinessEntityID = p.BusinessEntityID
WHERE ph.PhoneNumber LIKE '415%'
ORDER by p.LastName;
```

6)ISNULL (check_expression, replacement_value) - Заменяет значение NULL указанным замещающим значением.

Следующий пример демонстрирует расчет среднего значения веса всех продуктов. Все записи со значением NULL в столбце Weight таблицы Product заменяются значением 50.

```
SELECT                                  AVG(ISNULL(Weight,                                  50))
FROM                                                                        Production.Product;
```

Следует упомянуть оператор IN. Он определяет, совпадает ли указанное значение с любым значением в подзапросе или в списке.

Задание 3. Найдите и попробуйте использовать следующие наиболее популярные функции:

- конвертация строки в число
- вывод форматированной даты

- оператор Like
- получение подстроки из текста
- вычислить разницу дат

Иногда также возникает необходимость добавить колонку в созданную раннее таблицу. Это можно сделать таким образом:

```
ALTER TABLE table_name  ADD column_name type_name [ NULL | NOT NULL ];
```

Агрегатные функции

Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение.Агрегатные функции, за исключением COUNT, не учитывают значения NULL. Агрегатные функции часто используются в выражении GROUP BY инструкции SELECT.
Часто используемые агрегатные функции:
SUM ([ALL | DISTINCT] expression) - Возвращает сумму всех, либо только уникальных, значений в выражении. Функция SUM может быть использована только для числовых столбцов. Значения NULL пропускаются.
COUNT ({ [[ALL | DISTINCT] expression] | * }) - Возвращает количество элементов в группе.
Обычно в предложении GROUP BY используется предложение HAVING. Когда GROUP BY не используется, предложение HAVING работает так же, как и предложение WHERE.
В следующем примере, который использует простое предложение HAVING, из таблицы SalesOrderDetail извлекается сумма всех полей SalesOrderID, значение которых превышает $100000.00.

```
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
HAVING SUM(LineTotal) > 100000.00
ORDER BY SalesOrderID ;
```

Конструкцию **order by** можно использовать для сортировки результатов группировки.В следующем примере находится средняя цена книг каждого типа, а затем результаты располагаются в соответствии с этими средними ценами:

```
select type, avg(price)
from titles
group by type
order by avg(price)
```

Давайте перейдем к практике, но перед этим хотелось бы сделать небольшое отступление:

В предыдущей главе мы поверхностно рассмотрели SQL Server. Думаю, что при установке SQL Server у большинства из Вас возникли проблемы. И это нормально.
Ненормально может быть только то, что кто-то не справившись с установкой проклинает все, особенно меня, и бросает курс. Таким людям хочу сказать спасибо, т.к. нам не придется на них тратить свое время.

Сразу приготовьтесь к тому, что решать проблемы - это будет Вашим постоянным хобби. И если Вы дрогнули перед настройкой SQL Server, то что же будет при отладке сложного JS механизма на удаленном сервере с ошибкой, проявляющейся только в Internet Explorer.
Чем раньше Вы овладеете навыком решения технических проблем - тем проще Вам будет потом. Раньше программистам было гораздо сложнее - если не было более опытного программиста, то приходилось просто биться головой о задачу. Сейчас есть Google - практически любая проблема решается меньше, чем за полчаса при правильной постановке задачи.

Перейдем к практике:

пусть есть база с такими таблицами

- Сотрудник (id, name, depID, salary)
- Подразделение (id, name, parentDepID) - 2 уровней (если parentDepID null - то управление, если нет - то отдел в соотв управлении)
- Выплаты зарплаты (id, userID, createdDate, sum)

Задание 4. Напишите следующие запросы:
- средняя ЗП по отделам
- сколько человек в управлении
- сколько было выплат за последний месяц
- самые 3 дорогостоящих сотрудника за последние 2 года
- затраты по отделам за 2014 год

Задание 5(для продвинутых).
- затраты по последним 12 месяцам (здесь необходимо использовать курсор, либо группировку по году и месяцу от даты)

Разбиение на страницы в SQL (paging)

Разбиение на странице в SQL Server можно осуществить, используя встроенную ранжирующую функцию ROW_NUMBER(), она возвращает последовательный номер строки в секции результирующего набора, где 1 соответствует первой строке в каждой из секций. А также используя встроенную системную функцию @@ROWCOUNT - она возвращает число строк, затронутых при выполнении последней инструкции.

Рассмотрим хранимую процедуру из предыдущей главы:

```
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 параметрах процедуры передается размер страницы @pageSize и номер страницы @page, данные которой нужно вернуть в результирующем наборе. Далее весь результирующий набор делится на секции на основании этих параметров и выводятся значения подходящей секции.
Вообще пример получился довольно сложный - в нем есть много лишних элементов (например, создание подтаблицы). Однако изучите все эти дополнительные элементы также. Это реальный пример из одного проекта.

Более простой пример (взят, конечно же, со stackoverflow - http://stackoverflow.com/questions/109232/what-is-the-best-way-to-paginate-results-in-sql-server):

```
SELECT                                                                  *
FROM          (   SELECT          ROW_NUMBER()   OVER   (   ORDER   BY   OrderDate   )   AS   RowNum,   *
         FROM                                                                  Orders
         WHERE                                              OrderDate          >=          '1980-01-01'
       )                                      AS                                  RowConstrainedResult
WHERE                                                 RowNum                  >=                  1
    AND                               RowNum                          <                           20
ORDER BY RowNum
```

Бекап и восстановление БД

```
Как             делать          бекап           базы            на              сервере:
DECLARE                             @MyFileName                             varchar(50)
```

```
SELECT @MyFileName = (SELECT 'C:\backup\dbname_' + convert(varchar(50),GetDate(),112) + '.bak')
BACKUP DATABASE [DBNAME] TO DISK=@MyFileName
```

Таким образом, этот скрипт будет создавать бекапы вашей базы с именем файла, содержащим текущую дату, что удобно.

Примечание. Обязательно делайте резервные копии своих проектов. Если это уже запущенный в эксплуатацию проект - то как минимум 1 раз в день. Технически это делается на сервере с помощью назначенного задания (Sheduled taks).

Восстановление БД из back-файла:

```
RESTORE DATABASE BDName
FROM DISK = 'C:\temp1\filename.bak'
WITH REPLACE,
MOVE 'dbname' TO 'C:\DBs\dbName.mdf',
MOVE 'dbname_log' TO 'C:\DBs\dbName.ldf'
```

Иногда возникает проблема с тем, что вы не знаете какие именно имена написать в первый параметр Move (т.е. это 'dbname' и 'dbname_log').
Для этого используется специальный запрос, который позволяет получить информацию о файлах базы данных исходя из файла бекапа (.bak).

https://msdn.microsoft.com/en-us/library/ms190747.aspx

```
USE AdventureWorks2012;
RESTORE HEADERONLY
FROM DISK = N'C:\AdventureWorks2012-FullBackup.bak' ;
GO
```

И последний момент про бекапы - помните, что вы можете восстанавливать бекапы только на той же версии или выше. Т.е. если база была SQL Server 2014, то вы не сможете восстановить ее на SQL Server 2008. Только на 2014 или выше (причем при повышении произойдет преобразование базы - повышение версии базы).

Пожалуй, по SQL я рассказал все, что хотел. Конечно это только основы. Есть еще куча вопросов, которые мы не затронули.

Важно понимать, что в 98% Вам все это не понадобится. Используйте простые и надежные средства без ошибок и оперативно - и это уже будет большой шаг.

Не гонитесь узнать все сразу. Для большинства проектов того, что было перечислено, вполне достаточно. Изучите это как можно тщательнее на практике и тогда будет результат.

В следующих главах мы переходим к коду C#. Мы рассмотрим как создавать классы для доступа к данным, которые предоставляет SQL Server.

Эта глава будет полезна как веб-разработчикам, так и настольным их коллегам, т.к., по сути, веб-разработка - это в первую очередь HTML, CSS, JS. Но сначала нам надо получить данные из базы, обработать их в C#, и только затем выводить пользователю в браузер.