

SQL-запросы

Активная база	Use Library; Execute sp_help; Execute sp_help test_table -- информация о таблице
По заданным столбцам	Select N, Name, Price, Izd, Themes From books
Все столбцы	Select * from books
Псевдоним	Именам полей, при вычитке, можно дать псевдоним. Например, чтоб вместо слова Name в заголовке было указано Название. Для этого используется ключевое слово As. Select N as '№', Name as 'Название', Pages as 'Кол-во страниц' From Books
Арифметические операции	В запрос на выборку можно включать арифметические операции. Например, в базе хранится закупочная цена книг, а вам нужно показать цену, по которой вы их продаете (на 20% дороже, чем закупочная цена). Для этого достаточно выполнить такой запрос. Select N as '№', Name as 'Название', Pages as 'Кол-во страниц', Price*1.2 as 'Цена продажи' From Books При вычитке, каждое значение в поле Price будет увеличено на 20%. (Содержимое базы данных не изменится!!!).
Условия проверки	<ul style="list-style-type: none"> • < - меньше • > - больше • <= - меньше либо равно • >= - больше либо равно • <> - проверка на неравенство • = - проверка на равенство • !> - не больше чем • !< - не меньше чем • is NULL - проверка на отсутствие записи Для объединения условий используются операторы объединения: <ul style="list-style-type: none"> • and - логическое И (общее условие будет верным, только если все объединенные условия верны) • or - логическое Или (общее верно, если хотя бы одно из объединяемых условий верно)
Цена больше 50грн	Select * From Books Where Price>50
Сортировка значений по полям	Select Name,Price From books order by Price Desc Полный аналог этого запроса. Select Name,Price From books order by 2 Desc

<p>In</p> <p>Ключевое слово In используется для проверки на наличие чего либо в множестве. Что имеется в виду.</p>	<pre>select name, izd From books Where izd='Питер' or izd='Бином' or izd='Блиц' or izd='Афон'</pre> <p>Новый вариант этого запроса выглядит следующим образом:</p> <pre>select name, izd From books Where izd in ('Питер', 'Бином', 'Блиц', 'Афон')</pre>
<p>Between</p> <p>Between используется для проверки на принадлежность определенного значения указанному диапазону.</p>	<p>Нужно показать все книги, у которых кол-во страниц больше 500, но меньше 650</p> <pre>Select name,pages From books Where pages>=500 and pages<=650 order by 2 asc</pre> <p>Новый вариант этого запроса:</p> <pre>Select name,pages From books Where pages between 500 and 650 order by 2 asc</pre> <p>Также between можно использовать для работы с первой буквой строки. Но это будет проверка букв по ASCII кодам. Вот только при работе с цифрами верхний диапазон включается в диапазон проверки, а при работе с буквами - нет.</p> <pre>Select name,pages From books Where name between 'r' and 'e' order by 1 asc</pre>
<p>distinct</p>	<p>Исключение повторов</p> <pre>SELECT distinct Themes as 'Тематика', Category as 'Категория' FROM books WHERE Category is NULL</pre>
<p>Like</p>	<p>В шаблоне можно указывать такие служебные символы и комбинации символов:</p> <ol style="list-style-type: none"> 1. % - заменяет любое кол-во любых символов в строке. 2. _ - заменяет любой один символ. 3. [набор символов] - определяет, что на этом месте может стоять один из перечисленных символов. 4. [первый символ-второй символ] - определяет, что на этом месте может стоять один из символов в указанном диапазоне. 5. [^набор символов] - определяет, что на этом месте может стоять любой, кроме перечисленных символов. 6. [^первый символ-второй символ] - определяет, что на этом месте может стоять любой, кроме символов из указанного диапазона. <p>Необходимо вычитать все книги, в названии которых есть слово Словарь.</p> <pre>Select name,price From books Where name like '%словарь%'</pre> <p>Необходимо вычитать все названия книг, в названии которых первая буква либо А, либо З.</p> <pre>Select name,price From books Where name like '[АЗ]%'</pre>
<p>Not</p>	<p>Перед каждым из этих перечисленных ключевых слов можно указать ключевое слово Not, отрицающее то, что указано после него.</p> <p>Показать все книги, цена у которых не находится в диапазоне от 25 до 150.</p> <pre>Select name,price From books Where price not between 25 and 150 Order by 2 desc</pre>
<p>SELECT INTO</p>	<p>С помощью этого оператора можно создать таблицу, значения в которой будут являться значениями результата запроса на выборку. Созданная таким образом таблица может быть либо временной (которая автоматически уничтожится после завершения сеанса связи с сервером), либо постоянной. И в том и в другом случае для столбцов желательно указывать псевдонимы (при помощи ключевого слова as), чтобы в дальнейшем не было проблем с идентификацией полей созданной таблицы.</p> <pre>Select name as book,izd as Press Into Table1 From books Where new=1</pre> <p>Создастся новая таблица Table1, состоящая из двух полей book - название книги и press - издательство. При этом хранятся там только книги новинки (new=1) из таблицы Books.</p>

	<p>Все что вам нужно сделать для создания временной таблицы, это перед именем создаваемой таблицы оператора Select Into указать #. При этом будет создана ЛОКАЛЬНАЯ временная таблица, которая уничтожится после завершения текущего сеанса работы с SQL Server. Также перед именем можно указать ##. Тогда создастся ГЛОБАЛЬНАЯ временная таблица, которая будет доступна всем подключенным к серверу пользователям и уничтожена она будет после отключения от сервера последнего пользователя.</p> <p>Select Name,Themes,Pages,Date,Price Into ##temp_table1 From books where price>250 Order by Name</p> <p>Учитывая, что псевдонимы для столбцов не указаны, будут взяты имена из таблицы books.</p>
UPDATE	<p>Служит для обновления (изменения) информации в файле.</p> <p>Необходимо увеличить цену книг новинок на 20%.</p> <p>Update books Set price=price*1.2 where new=1</p> <p>Если не указать условие, то изменение будет произведено со всеми строками в указанных полях. Необходимо установить категорию "неизвестная", там, где категория не указана.</p> <p>Update books Set category='Неизвестная' where category is NULL</p> <p>При помощи Update можно очистить содержимое столбца, указав имя_поля=NULL и не указав условие.</p>
DELETE	<p>Оператор Delete используется для удаления записей (строк) из таблицы.</p> <p>Учтите, что после удаления восстановить информацию невозможно.</p> <p>ВНИМАНИЕ!!! Если не указать условие в запросе на удаление, то удалится вся информация из указанной таблицы.</p> <p>Необходимо удалить все книги, дата издания у которых неизвестна.</p> <p>Delete from books Where date is null</p> <p>Необходимо удалить все справочники (в названии категории есть слово справочник).</p> <p>Delete from books Where category like '%справочник%'</p>
INSERT INTO	<p>INSERT INTO table_name (field1, field2,...fieldN) VALUES (value1, value2,...valueN);</p>

Функции агрегирования

SUM(Имя_поля)	возвращает сумму значений в перечисленном столбце
COUNT([ALL DISTINCT] Имя_поля] *)	<p>возвращает кол-во либо всех значений в запросе (ALL), либо всех, не учитывая повторяющиеся (DISTINCT), либо кол-во строк в результате запроса (при указании *)</p> <p><code>select count(id) as "Кол-во издательств" from press</code></p> <p>Показать кол-во книг, кол-во страниц в которых больше 1000.</p> <p><code>select count(ALL books.name) as "кол-во" from books where pages>1000</code></p>
MAX(Имя_поля)	возвращает максимум из указанного поля
MIN(Имя_поля)	возвращает минимум из указанного поля
AVG(Имя_поля)	<p>возвращает среднее арифметической из всех значений, которые есть в указанном поле</p> <p>Показать среднее арифметическое цен всех книг.</p> <p><code>select avg(price) as "Среднее" from books</code></p>

GROUP BY и HAVING

Group by позволяет разбить общий результат, который возвращает функция агрегирования, по определенному критерию (сгруппировать данные по критерию). Общий синтаксис:

Select агрегатная_функция, поле1, поле2,...полеN

From имя_таблицы

Where условие

Group by поле1, поле2,...полеN

Критерием будет являться то поле, которое указывается рядом с агрегатной функцией. Не забывайте, что после Group by должны указываться все поля, которые стоят рядом с функцией агрегирования. В противном случае это приведет к ошибке.

Вывести общее количество книг от каждого издательства

Select count(books.id) as "Кол-во", press.name

From books,press

Where books.id_press=press.id

Group by press.name

Order by 1 desc

Показать кол-во книг, взятых каждой из групп студентов. (база данных library).

Select count(s_cards.id_book) as "Кол-во взятых книг", groups.name as "Название группы"

From groups,students,s_cards

where groups.id=students.id_group and students.id=s_cards.id_student

group by groups.name

Иногда нужно указывать рядом с функцией агрегирования несколько полей. Тогда после Group by нужно указывать все перечисленные в запросе поля.

Показать кол-во книг, взятых каждым студентом, группу, имя студента и фамилию.

Select count(s_cards.id_book) as "Кол-во", groups.name as "Группа", students.firstname, students.lastname

From groups,students,s_cards

Where groups.id=students.id_group and students.id=s_cards.id_student

Group by groups.name,students.firstname,students.lastname

	Кол-во	Группа	firstname	lastname
1	1	18П2	Вячеслав	Зезик
2	1	18П2	Галина	Инащенко
3	1	18П2	Игорь	Удовик
4	1	18П2	Ольга	Мантуляк
5	1	18П2	Петр	Кацевич
6	1	18П2	Юрий	Минаев
7	2	19Д	Александр	Любенко
8	1	9А	Елена	Таран

Важно помнить, что после Group by нельзя указывать поле, которого нет в перечислении для вывода на экран.

GROUP BY

HAVING

Директива Having служит для наложения условия на поле, которое указывается в группировке.

Select агрегатная_функция, поле1, поле2,...полеN

From имя_таблицы

Where условие

Group by поле1, поле2,...полеN

Having {[условие_по_агрегатной_функции]либо[условие_по_сгруппированному_полю]}

Условие по сгруппированному полю либо по агрегатной функции указать можно **только** в директиве Having.

Показать на экран среднее кол-во страниц по каждой из тематик, при этом показать только тематики, у которых среднее кол-во больше 400.

Select avg(books.pages) as "Средняя сумма страниц", themes.name

from books,themes

where books.id_themes=themes.id

group by themes.name

having avg(books.pages)>400

Показать на экран сумму страниц по каждой из тематик, при этом учитывать только книги с кол-вом страниц больше 300.

Select sum(books.pages) as "Сумма", themes.name

From books,themes

Where books.id_themes=themes.id and books.pages>300

Group by themes.name

books.pages не участвует в группировке, и поэтому оно указывается в Where. Указать имя поля, которое не участвует в группировке, после директивы Having нельзя!!!

Показать на экран сумму страниц по каждой из тематик, при этом учитывать только книги с кол-вом страниц больше 300, но учитывать, при этом, только тематики 'Программирование', 'Сети' и 'Web-дизайн'.

Select sum(books.pages) as "Сумма", themes.name

From books,themes

Where books.id_themes=themes.id and books.pages>300

Group by themes.name

Having themes.name in ('Программирование','Сети','Web-дизайн')

	Сумма	name
1	1552	Web-дизайн
2	1912	Программирование
3	480	Сети

SOME, ANY

Використовуючи підзапити знайти постачальників, товарів яких немає в продажі. Використайте для пошуку оператор ANY або SOME	<pre>SELECT Supplier.Name AS 'Поставщик', Product.Name AS 'Поставленный товар, которого нет в продаже' FROM Supplier, Delivery, Product WHERE Delivery.IdSupplier = Supplier.Id AND Delivery.IdProduct = Product.Id AND NOT Product.Id = ANY (SELECT Sale.IdProduct FROM Sale)</pre>
---	---

EXISTS

--10. Виведіть список виробників, які розміщуються не в Україні. Відсортуйте вибірку в зростаючому порядку назв виробників. Для даного підзапиту не використовуйте об'єднання таблиць, а лише корельовані підзапити та оператор EXISTS	<pre>SELECT sup.Name AS 'Производитель' FROM Supplier sup WHERE EXISTS (SELECT * FROM Address ad WHERE sup.IdAddress = ad.Id AND EXISTS (SELECT * FROM Country co WHERE co.Name NOT LIKE 'Украина' AND ad.IdCountry = co.Id)) ORDER BY 1</pre>

LEFT OUTER JOIN

Вивести інформацію про те, товарів яких виробників в базі даних не існує.

Для виведення повноцінної інформації скористайтесь зовнішнім об'єднанням

```
SELECT Producer.Name AS 'Производитель',
       Product.Name AS 'Товар'
FROM   Producer LEFT OUTER JOIN Product ON Product.IdProducer = Producer.Id
```

8	Сони ЛТД	Холодильник
9	Лонгсдейл ЛТД	NULL
10	Найк ЛТД	NULL
11	Минайлов ПП	Телевизор
12	Квартет ЛТД	Чай

UNION

Для применения команды **UNION** существует правила:

- число и порядок следования колонок должны быть одинаковы во всех запросах
- типы данных должны быть совместимы
- имена колонок определяются по первой части объединения
- **UNION** автоматически исключает дубликаты строк из вывода. Если вы хотите, чтобы все строки из запросов попали в результат используйте **UNION ALL**
- Вместе с **UNION** может использоваться **ORDER BY** для упорядочивания вывода. При этом **ORDER BY** указывается только после последнего запроса, входящего в **UNION**.

Подзапросы

Сначала выполняется вложенный запрос, а только после этого основной. Глубина вложенности ничем не ограничивается, поэтому лучше сказать, что с начала выполнится самый последний вложенный запрос, потом предпоследний и последним будет выполнен основной запрос. Любой вложенный запрос заключается в круглые скобки.

Существует ряд ограничений, которые накладываются на вложенные запросы:

1. Вложенный запрос необходимо всегда заключать в круглые скобки.
2. Вложенный запрос не может содержать Select into, Order by.
3. Вложенный запрос не может возвращать более одного столбца, если он включен в директиву In.
4. Вложенный запрос не может указываться в Order by.

необходимо показать название книги, у которой максимальное кол-во страниц из всей базы данных books	<pre>select name,pages from books where pages= (select max(pages)from books)</pre> <table><tr><th></th><th>name</th><th>pages</th></tr><tr><td>1</td><td>Компьютерная математика . Теория и практ</td><td>1296</td></tr></table>		name	pages	1	Компьютерная математика . Теория и практ	1296
	name	pages					
1	Компьютерная математика . Теория и практ	1296					
Необходимо показать студентов, которые брали книги тематик 'Базы данных', либо 'Web-дизайн'.	<p>Дело все в том, что вложенный запрос, который вычитывает id тематик, возвращает более одной записи и поэтому проверка его результата на равенство приводит к ошибке, как и запрос на уровень выше о id книг, этих тематик. Чтобы можно было сделать такой запрос необходимо знак " = " заменить на IN (проверка на членство в множестве).</p> <pre>Select students.firstname from students,s_cards where s_cards.id_book in (select id from books where id_themes in (select id from themes where name in ('Базы данных','Web-дизайн'))) and students.id=s_cards.id_student</pre>						

Операторы

1. **Оператори**, які дозволяють виконувати набір дій над одним або декількома компонентами виразу.
 - **Арифметичні**: додавання (+), віднімання (-), множення (*), ділення (/), цілочисельного ділення одного операнда на другий (\), оператор ділення по модулю (Mod), піднесення до степеня (^). В ролі операндів можуть бути як числа, так і значення полів.
 - **Порівняння**: >, <, >=, <=, =, <>. Якщо один з операндів має значення NULL, то результатом порівняння також буде NULL.
 - **Логічні (булівські)**, результатом роботи яких є логічне значення True (-1), False (0) або NULL. Їх використовують для комбінування результатів виконання двох і більше операцій. Це: логічне І (And), включаюче АБО (Or), логічне заперечення (Not) та виключаюче АБО (Xor).
 - **Конкатенації** - для об'єднання кількох рядків (&). Про нього ми вже говорили вище.
 - **Оператори SQL**: BETWEEN..AND.., IN, LIKE, IS NULL (перевірка на рівність нулю).
2. **Літерали** – це значення в явному їх представленні.
 - **Числові**, які можуть містити знак розділення (в десяткових числах) та знак мінус (-) для від'ємних значень, символи е або Е. Наприклад: 3,4567E-01, 12000, -25.
 - **Текстові (рядкові)** – будь-які друковані символи (А-Я, 0-9, знаки пунктуації тощо). Їх слід писати в одинарних або подвійних лапках. Наприклад: "Мелодрама", "Київ", "Готель "Україна" '.
 - **Дати та часу**. В більшості СУБД дата та час пишуться в одинарних лапках, але це може бути і інший довільний символ. В MS Access цим символом є символ хеш (#). Але не забувайте, що ця вимога лише для виразів SQL і при введенні значення дати або часу через користувацький інтерфейс даний символ не вказується. Наприклад: #01.03.99#, #15-лнв-2001#.

Функції дозволяють спростити процес встановлення умови для вибірки даних. В MS Access можна використовувати як вбудовані функції, так і визначені користувачем (написані на VBA). Наведемо короткий перелік часто використовуваних вбудованих функцій:

- **Дати та часу**:
 - Date() – повертає поточну дату;
 - DateAdd("d", -15, [ДатаПоставки]) - повертає дату, що на 15 днів передує даті, заданої значенням поля «ДатаПоставки»;
 - DateDiff("d", [ДатаПоставки],[ДатаПродажу]) – повертає значення, що є різницею значень полів «ДатаПоставки» та «ДатаПродажу»;
 - Year(#12.06.01#) - повертає рік з вказаної дати, тобто 2001
- **Рядкові**:
 - Format(Date, #dd-mm-yyyy#) – повертає відформатовану дату;
 - InStr("Місто", "С") – повертає число, що вказує позицію першого входження одного рядка в інший, тобто 3;
 - LCase("МІСТО") – переводить рядок в нижній регістр;
 - Left([Місто],2) – відображає два перших сивола значення поля «Місто»;
 - Right([Місто],3) - відображає три останніх сивола значення поля «Місто»;
 - Trim([Назва]) – повертає значення поля «Назва» без пропусків (space).
- **Приведення типу даних**:
 - Val("12.35") – конвертує текст в число, тобто повертає 12,35;
 - Str(12,35) – конвертує число в текст, – "12,35".

Для пошуку значень по шаблону використовується оператор **LIKE**. Шаблон виразу може включати в себе:

* або % - в даній позиції може бути присутній 0 або більше символів;

? або _ - в даній позиції обов'язково присутній один довільний символ;

- в даній позиції присутня одна цифра;

[a-z] - в даній позиції обов'язково присутній один символ з вказаного діапазону;

[abc] - в даній позиції обов'язково присутній один символ з вказаного діапазону значень;

[!a-z] - в даній позиції обов'язково присутній один символ, що не входить в вказаний діапазон;

[!abc] - в даній позиції обов'язково присутній один символ, що не входить в вказаний діапазон значень.

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR_LENGTH()	Return number of characters in argument
CHAR()	Return the character for each integer passed
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CONCAT_WS()	Return concatenate with separator
CONCAT()	Return concatenated string
ELT()	Return string at index number
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Return the index (position) of the first argument in the subsequent arguments

Name	Description
<u>FIND IN SET()</u>	Return the index position of the first argument within the second argument
<u>FORMAT()</u>	Return a number formatted to specified number of decimal places
<u>FROM_BASE64()</u>	Decode to a base-64 string and return result
<u>HEX()</u>	Return a hexadecimal representation of a decimal or string value
<u>INSERT()</u>	Insert a substring at the specified position up to the specified number of characters
<u>INSTR()</u>	Return the index of the first occurrence of substring
<u>LCASE()</u>	Synonym for LOWER()
<u>LEFT()</u>	Return the leftmost number of characters as specified
<u>LENGTH()</u>	Return the length of a string in bytes
<u>LIKE</u>	Simple pattern matching
<u>LOAD_FILE()</u>	Load the named file
<u>LOCATE()</u>	Return the position of the first occurrence of substring
<u>LOWER()</u>	Return the argument in lowercase
<u>LPAD()</u>	Return the string argument, left-padded with the specified string
<u>LTRIM()</u>	Remove leading spaces
<u>MAKE_SET()</u>	Return a set of comma-separated strings that have the corresponding bit in bits set
<u>MATCH</u>	Perform full-text search
<u>MID()</u>	Return a substring starting from the specified position
<u>NOT LIKE</u>	Negation of simple pattern matching
<u>NOT REGEXP</u>	Negation of REGEXP
<u>OCT()</u>	Return a string containing octal representation of a number
<u>OCTET_LENGTH()</u>	Synonym for LENGTH()
<u>ORD()</u>	Return character code for leftmost character of the argument
<u>POSITION()</u>	Synonym for LOCATE()
<u>QUOTE()</u>	Escape the argument for use in an SQL statement
<u>REGEXP</u>	Pattern matching using regular expressions
<u>REPEAT()</u>	Repeat a string the specified number of times
<u>REPLACE()</u>	Replace occurrences of a specified string
<u>REVERSE()</u>	Reverse the characters in a string
<u>RIGHT()</u>	Return the specified rightmost number of characters
<u>RLIKE</u>	Synonym for REGEXP
<u>RPAD()</u>	Append string the specified number of times
<u>RTRIM()</u>	Remove trailing spaces
<u>SOUNDEX()</u>	Return a soundex string
<u>SOUNDS LIKE</u>	Compare sounds
<u>SPACE()</u>	Return a string of the specified number of spaces
<u>STRCMP()</u>	Compare two strings
<u>SUBSTR()</u>	Return the substring as specified
<u>SUBSTRING_INDEX()</u>	Return a substring from a string before the specified number of occurrences of the delimiter
<u>SUBSTRING()</u>	Return the substring as specified
<u>TO_BASE64()</u>	Return the argument converted to a base-64 string
<u>TRIM()</u>	Remove leading and trailing spaces
<u>UCASE()</u>	Synonym for UPPER()
<u>UNHEX()</u>	Return a string containing hex representation of a number
<u>UPPER()</u>	Convert to uppercase
<u>WEIGHT_STRING()</u>	Return the weight string for a string

Условия для текстовых полей

Чтобы добавить записи, которые...	Используйте это условие	Результат запроса
Точно соответствуют определенному значению, например "Китай"	"Китай"	Возвращает записи, в которых поле "СтранаРегион" содержит значение "Китай".
Не соответствуют определенному значению, например "Мексика"	Not "Мексика"	Возвращает записи, в которых значением поля "СтранаРегион" не является "Мексика".
Начинаются с заданной строки символов, например "С"	Like C*	Возвращает записи всех стран или регионов, названия которых начинаются с буквы "С", таких как Словакия и США. ПРИМЕЧАНИЕ Символ "звездочка" (*) в выражении обозначает любую строку символов. Он также называется подстановочным знаком. Список таких знаков см. в статье Справочные сведения о подстановочных знаках в приложении Access .
Не начинаются с заданной строки символов, например "С"	Not Like C*	Возвращает записи всех стран или регионов, названия которых не начинаются с буквы "С".
Содержат заданную строку, например "Корея"	Like "*Корея*"	Возвращает записи всех стран или регионов, названия которых содержат строку "Корея".
Не содержат заданную строку, например "Корея"	Not Like "*Корея*"	Возвращает записи всех стран или регионов, названия которых не содержат строку "Корея".
Заканчиваются заданной строкой, например "ина"	Like "*ина"	Возвращает записи всех стран или регионов, названия которых заканчиваются на "ина", таких как "Украина" и "Аргентина".
Не заканчиваются заданной строкой, например "ина"	Not Like "*ина"	Возвращает записи всех стран или регионов, названия которых не заканчиваются на "ина", как в названиях "Украина" и "Аргентина".
Содержат пустые значения (или значения отсутствуют)	Is Null	Возвращает записи, в которых это поле не содержит значения.
Не содержат пустых значений	Is Not Null	Возвращает записи, в которых это поле содержит значение.
Содержат пустую строку	"" (прямые кавычки)	Возвращает записи, в которых поле имеет пустое значение (но не значение NULL). Например, записи о продажах другому отделу могут содержать пустое значение в поле "СтранаРегион".
Не содержат пустых строк	Not ""	Возвращает записи, в которых поле "СтранаРегион" имеет непустое значение.
Содержит нулевые	"" Or Is Null	Возвращает записи, в которых значение в поле отсутствует

Чтобы добавить записи, которые...	Используйте это условие	Результат запроса
значения или пустые строки		или является пустым.
Ненулевые и непустые	Is Not Null And Not ""	Возвращает записи, в которых поле "СтранаРегион" имеет непустое значение, не равное NULL.
При сортировке в алфавитном порядке следуют за определенным значением, например "Мексика"	>= "Мексика"	Возвращает записи с названиями стран и регионов, начиная с Мексики и до конца алфавита.
Входят в определенный диапазон, например от А до Г	Like "[А-Г]*"	Возвращает страны и регионы, названия которых начинается с букв от "А" до "Г".
Совпадают с одним из двух значений, например "Словакия" или "США"	"Словакия" Or "США"	Возвращает записи для США и Словакии.
Содержат одно из значений, указанных в списке	In("Франция", "Китай", "Германия", "Япония")	Возвращает записи всех стран или регионов, указанных в списке.
Содержат определенные знаки в заданном месте значения поля	Right([СтранаРегион], 1) = "а"	Возвращает записи всех стран или регионов, названия которых заканчиваются на букву "а".
Соответствуют заданной длине	Len([СтранаРегион]) > 10	Возвращает записи стран или регионов, длина названия которых превышает 10 символов.
Соответствуют заданному шаблону	Like "Лив???"	Возвращает записи стран или регионов, названия которых состоят из пяти символов и начинаются с "Лив", например Ливия и Ливан. ПРИМЕЧАНИЕ Символы ? и _ в выражении обозначают один символ. Они также называются подстановочными знаками. Знак _ нельзя использовать в одном выражении с символом ?, а также с подстановочным знаком *. Вы можете использовать подстановочный знак _ в выражении, где есть подстановочный знак %.

Форматирование текста (MS Access)

<i>Result:</i> '210.60'	Format (210.6, "#,##0.00")
<i>Result:</i> '210.60'	Format (210.6, "Standard")
<i>Result:</i> '98.10%'	Format (0.981, "Percent")
<i>Result:</i> '\$1,267.50'	Format (1267.5, "Currency")

Условия для числовых полей, полей с денежными значениями и полей счетчиков

Следующие примеры относятся к полю "ЦенаЗаЕдиницу", основанном на таблице, в которой хранится информация о товарах. Условие задается в строке **Условие отбора** поля на бланке запроса.

Товары

★

🔑

КодТовара

НаименованиеТовара

КодПоставщика

КодКатегории

КоличествоНаЕдиницу

ЦенаЗаЕдиницу

НаСкладе

Заказано

МинЗапас

ПоставкиПрекращены

Поле:

Таблица:

Сортировка:

Вывод на экран:

Условие отбора:

или:

НаименованиеТовара	ЦенаЗаЕдиницу
Товары	Товары
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Чтобы добавить записи, которые...	Используйте это условие	Результат запроса
Точно соответствуют определенному значению, например 1000	1000	Возвращает записи, в которых цена за единицу товара составляет 1000 ₺.
Не соответствуют значению, например 10 000	Not 10 000	Возвращает записи, в которых цена за единицу товара не равна 10 000 ₺.
Содержат значение, которое меньше заданного, например 1000	< 1000 <= 1000	Возвращает записи, в которых цена товара меньше 1000 ₺ (<1000). Второе выражение (<=1000) отображает записи, в которых цена не больше 1000 ₺.
Содержат значение, которое больше заданного, например 999,99	>999,99 >=999,99	Возвращает записи, в которых цена товара больше 999,99 ₺ (>999,99). Второе выражение отображает записи, цена в которых не меньше 999,99 ₺.
Содержат одно из двух значений, например 200 или 250	200 или 250	Возвращает записи, в которых цена товара равна 200 или 250 ₺.
Содержат значение, которое входит в определенный диапазон	>499,99 and <999,99 или Between 500 and 1000	Возвращает записи товаров с ценами в диапазоне от 499,99 до 999,99 ₺ (не включая эти значения).

Чтобы добавить записи, которые...	Используйте это условие	Результат запроса
Содержат значение, которое не входит в определенный диапазон	<500 or >1000	Возвращает записи, в которых цена товара не находится в диапазоне от 500 до 1000 ₺.
Содержит одно из заданных значений	In(200, 250, 300)	Возвращает записи, в которых цена товара равна 200, 250 или 300 ₺.
Содержат значение, которое заканчивается на заданные цифры	Like "*4,99"	Возвращает записи товаров, цена которых заканчивается на 4,99, например 4,99 ₺, 14,99 ₺, 24,99 ₺ и т. д. <div>ПРИМЕЧАНИЕ</div> Знаки * и % в выражении обозначают любое количество символов. Они также называются подстановочными знаками. Знак % нельзя использовать в одном выражении с символом *, а также с подстановочным знаком ?. Вы можете использовать подстановочный знак % в выражении, где есть подстановочный знак _.
Содержат пустые значения (или значения отсутствуют)	Is Null	Возвращает записи, для которых не введено значение в поле "ЦенаЗаЕдиницу".
Содержат непустые значения	Is Not Null	Возвращает записи, в поле "ЦенаЗаЕдиницу" которых указано значение.

Условия для полей "Дата/время"

Проверка текущей даты

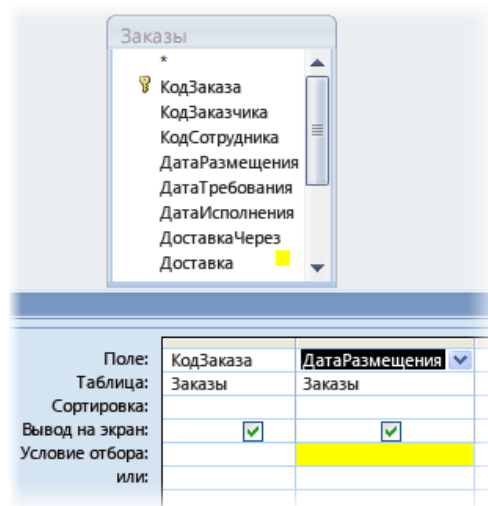
```
select GETDATE() --2012-05-01 10:14:13.403
```

you can get day, month and year separately by doing:

```
select DAY(getdate()) --1  
select month(getdate()) --5  
select year(getdate()) --2012
```

```
select cast (GETDATE() as DATE) --2012-05-01
```

Следующие примеры относятся к полю "ДатаЗаказа", основанном на таблице, в которой хранится информация о заказах. Условие задается в строке **Условие отбора** поля на бланке запроса.



Чтобы добавить записи, которые...	Используйте это условие	Результат запроса
Точно соответствуют значению, например 02.02.2006	#02.02.2006#	Возвращает записи транзакций, выполненных 2 февраля 2006 г. Обязательно ставьте знаки # до и после значений даты, чтобы Access мог отличить значения даты от текстовых строк.
Не соответствуют значению, такому как 02.02.2006	Not #02.02.2006#	Возвращает записи транзакций, выполненных в любой день, кроме 2 февраля 2006 г.
Содержат значения, которые предшествуют определенной дате, например 02.02.2006	< #02.02.2006#	Возвращает записи транзакций, выполненных до 2 февраля 2006 г. Чтобы просмотреть транзакции, выполненные в определенную дату или до нее, воспользуйтесь

Чтобы добавить записи, которые...	Используйте это условие	Результат запроса
		оператором <= вместо оператора <.
Содержат значения, которые следуют за определенной датой, например 02.02.2006	> #02.02.2006#	Возвращает записи транзакций, выполненных после 2 февраля 2006 г. Чтобы просмотреть транзакции, выполненные в определенную дату или после нее, воспользуйтесь оператором >= вместо оператора >.
Содержат значения, которые входят в определенный диапазон дат	>#02.02.2006# and <#04.02.2006#	Возвращает записи транзакций, выполненных в период между 2 и 4 февраля 2006 г. Кроме того, для фильтрации по диапазону значений, включая конечные значения, вы можете использовать оператор Between . Например, выражение Between #02.02.2006# and #04.02.2006# идентично выражению >=#02.02.2006# and <=#04.02.2006#.
Содержат значения, которые не входят в определенный диапазон	<#02.02.2006# or >#04.02.2006#	Возвращает записи транзакций, выполненных до 2 февраля 2006 г. или после 4 февраля 2006 г.
Содержат одно из двух заданных значений, например 02.02.2006 или 03.02.2006	#02.02.2006# or #03.02.2006#	Возвращает записи транзакций, выполненных 2 или 3 февраля 2006 г.
Содержит одно из нескольких значений	In (#01.02.2006#, #01.03.2006#, #01.04.2006#)	Возвращает записи транзакций, выполненных 1 февраля 2006 г., 1 марта 2006 г. или 1 апреля 2006 г.
Содержат дату, которая выпадает на определенный месяц (вне зависимости от года), например декабрь	DatePart("m"; [ДатаПродажи]) = 12	Возвращает записи транзакций, выполненных в декабре любого года.
Содержат дату, которая выпадает на определенный квартал (вне зависимости от года), например первый	DatePart("q"; [ДатаПродажи]) = 1	Возвращает записи транзакций, выполненных в первом квартале любого года.
Содержат текущую дату	Date()	Возвращает записи транзакций, выполненных сегодня. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи, в поле "ДатаЗаказа" которых указано 2

Чтобы добавить записи, которые...	Используйте это условие	Результат запроса
		февраля 2006 г.
Содержат вчерашнюю дату	Date()-1	Возвращает записи транзакций, выполненных вчера. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за 1 февраля 2006 г.
Содержат завтрашнюю дату	Date() + 1	Возвращает записи транзакций, которые будут выполнены завтра. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за 3 февраля 2006 г.
Содержат даты, которые выпадают на текущую неделю	DatePart("ww"; [ДатаПродажи]) = DatePart("ww"; Date()) and Year([ДатаПродажи]) = Year(Date())	Возвращает записи транзакций, выполненных за текущую неделю. Неделя начинается в воскресенье и заканчивается в субботу.
Содержат даты, которые выпадают на прошлую неделю	Year([ДатаПродажи])* 53 + DatePart("ww"; [ДатаПродажи]) = Year(Date())* 53 + DatePart("ww"; Date()) - 1	Возвращает записи транзакций, выполненных за прошлую неделю. Неделя начинается в воскресенье и заканчивается в субботу.
Содержат даты, которые выпадают на следующую неделю	Year([ДатаПродажи])* 53+DatePart("ww"; [ДатаПродажи]) = Year(Date())* 53+DatePart("ww"; Date()) + 1	Возвращает записи транзакций, которые будут выполнены на следующей неделе. Неделя начинается в воскресенье и заканчивается в субботу.
Содержат дату, которая выпадает на последние 7 дней	Between Date() and Date()-6	Возвращает записи транзакций, выполненных за последние 7 дней. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за период с 24 января 2006 г. по 2 февраля 2006 г.
Содержат дату, которая выпадает на текущий месяц	Year([ДатаПродажи]) = Year(Now()) And Month([ДатаПродажи]) = Month(Now())	Возвращает записи за текущий месяц. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за февраль 2006 г.
Содержат дату, которая выпадает на прошлый месяц	Year([ДатаПродажи])* 12 + DatePart("m"; [ДатаПродажи]) = Year(Date())* 12 + DatePart("m"; Date()) - 1	Возвращает записи за прошлый месяц. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за январь 2006 г.
Содержат дату, которая выпадает на следующий месяц	Year([ДатаПродажи])* 12 + DatePart("m"; [ДатаПродажи]) = Year(Date())* 12 + DatePart("m"; Date()) + 1	Возвращает записи за следующий месяц. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за март 2006 г.
Содержат дату, которая выпадает на последние 30	Between Date() And DateAdd("M", -1, Date())	Записи о продажах за месяц. Если сегодняшняя дата — 02.02.2006 г., вы

Чтобы добавить записи, которые...	Используйте это условие	Результат запроса
дней или 31 день		увидите записи за период с 2 января 2006 г. по 2 февраля 2006 г.
Содержат дату, которая выпадает на текущий квартал	Year([ДатаПродажи]) = Year(Now()) And DatePart("q"; Date()) = DatePart("q"; Now())	Возвращает записи за текущий квартал. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за первый квартал 2006 г.
Содержат дату, которая выпадает на прошлый квартал	Year([ДатаПродажи])*4+DatePart("q";[ДатаПродажи]) = Year(Date())*4+DatePart("q";Date())- 1	Возвращает записи за прошлый квартал. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за последний квартал 2005 г.
Содержат дату, которая выпадает на следующий квартал	Year([ДатаПродажи])*4+DatePart("q";[ДатаПродажи]) = Year(Date())*4+DatePart("q";Date())+1	Возвращает записи за следующий квартал. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за второй квартал 2006 г.
Содержат дату, которая выпадает на текущий год	Year([ДатаПродажи]) = Year(Date())	Возвращает записи за текущий год. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за 2006 г.
Содержат дату, которая выпадает на прошлый год	Year([ДатаПродажи]) = Year(Date()) - 1	Возвращает записи транзакций, выполненных в прошлом году. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за 2005 г.
Содержат дату, которая выпадает на следующий год	Year([ДатаПродажи]) = Year(Date()) + 1	Возвращает записи транзакций, которые будут выполнены в следующем году. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за 2007 г.
Содержат дату, которая приходится на период с 1 января до текущей даты (записи с начала года до настоящего момента)	Year([ДатаПродажи]) = Year(Date()) and Month([ДатаПродажи]) <= Month(Date()) and Day([ДатаПродажи]) <= Day(Date())	Возвращает записи транзакций, которые приходятся на период с 1 января текущего года до сегодняшней даты. Если сегодняшняя дата — 02.02.2006 г., вы увидите записи за период с 1 января 2006 г. по 2 февраля 2006 г.
Содержат прошедшую дату	< Date()	Возвращает записи транзакций, выполненных до сегодняшнего дня.
Содержат будущую дату	> Date()	Возвращает записи транзакций, которые будут выполнены после сегодняшнего дня.
Фильтр пустых (или отсутствующих) значений	Is Null	Возвращает записи, в которых не указана дата транзакции.
Фильтр непустых значений	Is Not Null	Возвращает записи, в которых указана дата транзакции.

Name	Description
ADDDATE()	Adds dates
ADDTIME()	Adds time
CONVERT_TZ()	Converts from one timezone to another
CURDATE()	Returns the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Returns the current time
DATE_ADD()	Adds two dates
DATE_FORMAT()	Formats date as specified
DATE_SUB()	Subtracts two dates
DATE()	Extracts the date part of a date or datetime expression
DATEDIFF()	Subtracts two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Returns the name of the weekday
DAYOFMONTH()	Returns the day of the month (1-31)
DAYOFWEEK()	Returns the weekday index of the argument
DAYOFYEAR()	Returns the day of the year (1-366)
EXTRACT	Extracts part of a date
FROM_DAYS()	Converts a day number to a date
FROM_UNIXTIME()	Formats date as a UNIX timestamp
HOUR()	Extracts the hour
LAST_DAY	Returns the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Creates a date from the year and day of year
MAKETIME	MAKETIME()
MICROSECOND()	Returns the microseconds from argument
MINUTE()	Returns the minute from the argument
MONTH()	Returns the month from the date passed
MONTHNAME()	Returns the name of the month
NOW()	Returns the current date and time
PERIOD_ADD()	Adds a period to a year-month
PERIOD_DIFF()	Returns the number of months between periods
QUARTER()	Returns the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Returns the second (0-59)
STR_TO_DATE()	Converts a string to a date
SUBDATE()	When invoked with three arguments a synonym for DATE_SUB()
SUBTIME()	Subtracts times
SYSDATE()	Returns the time at which the function executes
TIME_FORMAT()	Formats as time
TIME_TO_SEC()	Returns the argument converted to seconds
TIME()	Extracts the time portion of the expression passed
TIMEDIFF()	Subtracts time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression. With two arguments, the sum of the arguments
TIMESTAMPADD()	Adds an interval to a datetime expression
TIMESTAMPDIFF()	Subtracts an interval from a datetime expression
TO_DAYS()	Returns the date argument converted to days
UNIX_TIMESTAMP()	Returns a UNIX timestamp
UTC_DATE()	Returns the current UTC date
UTC_TIME()	Returns the current UTC time
UTC_TIMESTAMP()	Returns the current UTC date and time
WEEK()	Returns the week number
WEEKDAY()	Returns the weekday index
WEEKOFYEAR()	Returns the calendar week of the date (1-53)
YEAR()	Returns the year
YEARWEEK()	Returns the year and week

Транзакции

Транзакция - это механизм, позволяющий объединить набор операций в один логический блок, определяющий либо подтверждение полного выполнения каждой из операций, либо отмена всех из них.

Любая транзакция обладает четырьмя важными свойствами, определенными в английском языке, как **свойства ACID (Atomicity, Consistency, Isolation, Durability)**.

1. **Atomicity - атомарность.** Это свойство определяет неделимость транзакции, т.е. не может выполняться только какая-либо ее часть, она выполняется либо полностью, либо не выполняется вообще.
2. **Consistency** - связность. Определяет, что, как в случае успешного, так и аварийного завершения транзакции, это не может повлиять на целостность данных.
3. **Isolation** - изолированность. Определяет, что одна транзакция не может взаимодействовать с другими транзакциями.
4. **Durability** - надежность. Определяет, что выполнение транзакции не зависит от внешних факторов. Т.е. даже если вдруг выключили свет и сервер отключился в момент выполнения транзакции, то если она не закончила свое выполнение, после перезагрузки сервера, вся информация останется в первоначальном виде.

Транзакции подразделяются на 3 вида.

1. Явные
2. Неявные
3. Автоматические

Явные транзакции

Явные - это транзакции, заданные явно. Такие транзакции

Begin Tran, либо Begin Transaction	начинаются с ключевых слов
Commit Tran, либо Commit Word	Заканчиваются словами для успешного завершения транзакции
Rollback Tran, Rollback Transaction, либо Rollback Work	для полного отката транзакции

Например.

Begin Transaction Insert into books values(771,8899,0,'Проверка работы транзакции',520,'Test',120,'20x20/15', 2004-05-03,1,'Использование транзакций','транзакции') Select * from books Where N=771 Commit Transaction
--

Иногда требуется проверить, успешно ли завершился какой-либо запрос и относительно этого определить, начинать ли выполнение следующего запроса. Для отслеживания успешности операций служит специальная системная переменная **@@ERROR**. Если Запрос прошел успешно, то в эту переменную записывается ноль, если же произошла ошибка, то в нее записывается код ошибки. Это значение хранится в переменной до начала выполнения следующей транзакции

```

Begin Transaction
Insert into books
values(771,8899,0,'Проверка работы транзакции',520,'Test',120,'20x20/15',
2004-05-03,1,'Использование транзакций','транзакции')
IF @@ERROR<>0
    BEGIN
        PRINT 'Error occurred'
        rollback transaction
    END
ELSE
Commit Transaction

```

Если вы поле N таблицы books сделаете первичным ключем (при помощи изменения настроек таблицы) и выполните указанную транзакцию, то на экране появится стандартное сообщение о том, что в поле, которое является первичным, нельзя записать два одинаковых значения, и, учитывая, что в переменной @@ERROR будет не ноль, то высветится еще настроенное нами сообщение Error occurred и транзакция будет откатана.

Команда **Rollback transaction** откатывает всю выполняемую транзакцию. Но иногда требуется сохранить какую-либо уже успешно законченную операцию и ее не откатывать, даже если произошла ошибка в операции, указанной ниже. Для этого служит установка **точки сохранения**.

Точка сохранения - устанавливается при помощи команды

Save Transaction имя_точки_сохранения

Тогда, для отката того, что находится после этой точки сохранения необходимо указывать команду:

Rollback transaction имя_точки_сохранения

commit transaction

Если указать **Rollback transaction**, то будет произведен откат всей транзакции.

Если не указать commit transaction после Rollback transaction имя_точки_сохранения, то транзакция зависнет и доступа к базе данных, к которой производился запрос, не будет, вплоть до вызова этого же запроса с командой commit transaction.

Пример.

```

Begin Transaction
insert into books (N,name,new,price) values (770,'Test1',1,12.53)
Save transaction savepoint1
insert into books (N,name,new,price) values (771,'Test1',1,12.53)
insert into books (N,name,new,price) values (770,'Test1',1,12.53)
if @@ERROR<>0
    begin
        print 'my error!!!'
        Rollback Transaction savepoint1
        Commit Transaction
    end
else
    Commit Transaction

```

Давайте рассмотрим данный пример подробнее.

- Определяем начало транзакции.
- Добавляем в таблицу books значения 770 - первичный ключ, Test1 - название книги, 1 - это книга новинка (это поле обязательное для заполнения), 12.53 - цена этой воображаемой

книги.

- Устанавливаем точку сохранения.
- Добавляем в базу данных еще одну книгу, но номер ей указываем 771. Такого номера нет, потому этот запрос тоже выполнится.
- **ВНИМАНИЕ** Добавляем в таблицу снова книгу с номером 770. Если N - первичный ключ, то происходит ошибка добавления записи, т.к. номера повторяются.
- Учитывая, что произошла ошибка, в переменной @@ERROR будет не ноль и команда PRINT 'строка' выведет на экран сообщение об ошибке.
- Производим откат всех операций, произведенных после точки сохранения. **Книга с номером 770 в базе должна остаться, а с номером 771, хотя запрос на добавление был выполнен успешно, сохранена не будет, за счет отката транзакции.**
- Завершаем транзакцию.

На этом примере очень четко можно рассмотреть работу транзакции. Мы определяем, что должен выполняться весь блок из трех запросов на добавление информации. Но мы также ограничиваем откат транзакции, используя точку сохранения. Т.е. если вызвать Rollback Transaction, то произойдет откат двух удачно выполненных запросов, но, учитывая точку сохранения, откатывается все, кроме тех операций, которые указаны до точки сохранения.

Неявные транзакции

Работая с транзакциями, не всегда удобно для каждой из них указывать **Begin Transaction**. Если включить механизм неявных транзакций, то Begin Transaction дописывается автоматически. Но, учитывая, что транзакцию можно как закончить, так и отменить, ни Commit transaction, ни Rollback Transaction автоматически дописываться не будут. Их все равно необходимо в конце транзакции указывать явно.

Для включения механизма неявных транзакций необходимо указать:

```
Set Implicit_transactions ON
```

Чтобы выключить:

```
Set Implicit_transactions OFF
```

Автоматические транзакции

Все дело в том, что даже если не указывать начало и конец транзакции, они все равно присутствуют. Например, запрос на вставку данных можно рассматривать как отдельную автоматическую транзакцию.

Пример.

```
insert into books  
values(772,8882,1,'Test2',125.23,'test_izd',550,'test_fromat',  
2005-08-30,12501,'test_themes','test_category')
```

В результате этой транзакции будет либо добавлена информация во все поля, либо не будет сохранена ни в одном из них. Так любой запрос сам по себе является транзакцией.

Временные таблицы

```
/*3) Показать тематику, самую популярную среди преподавателей */
/*(a) Суммируем какие книги по ID брались сколько раз */
SELECT T_Cards.Id_Book AS 'IdBook',
       COUNT(T_Cards.Id_Book) AS 'Кол-во взятых книг'
INTO #tmp_id
FROM T_Cards
GROUP BY T_Cards.Id_Book

/*(b) Суммируем кол-во книг по каждой теме */
SELECT Themes.Name AS 'Тема',
       COUNT(#tmp_id.[Кол-во взятых книг]) AS 'Кол-во взятых книг'
INTO #tmp_theme
FROM Themes, Books, #tmp_id
WHERE Books.Id_Themes = Themes.Id
      AND #tmp_id.IdBook = Books.Id
GROUP BY Themes.Name

/*(c) Отбираем самую популярную среди преподавателей тему книг */
SELECT #tmp_theme.Тема AS 'Тема',
       #tmp_theme.[Кол-во взятых книг] AS 'Кол-во взятых книг'
FROM #tmp_theme
WHERE #tmp_theme.[Кол-во взятых книг] =
(SELECT MAX(#tmp_theme.[Кол-во взятых книг]) FROM #tmp_theme)
```

Локальная переменная

```
DECLARE @totalSum int
SET @totalSum = (SELECT SUM(Sale.Quantity) FROM Sale)
```

Создание таблиц, баз и ограничений скриптами

Создание базы данных	CREATE DATABASE dbname
Создание таблицы	<pre>Create table Test_table (id int identity(1,1) not null primary key, name varchar(25), surname varchar(25), rights varchar(20) not null default 'user')</pre>
Заполнение значений	<pre>INSERT INTO Customers (CustomerName, Country) VALUES ('Cardinal', 'Norway'); INSERT INTO Table (Column1, Column2) VALUES (Value1, Value2), (Value1, Value2)</pre>
Ограничения	<p>Ограничения можно поделить на такие группы:</p> <ol style="list-style-type: none"> 1. Ограничение Первичный Ключ. 2. Ограничение по Умолчанию. 3. Ограничение на Проверку. 4. Ограничение Уникальности. 5. Ограничение Внешний Ключ. <p>Проверка наличия ограничений :</p> <pre>Execute sp_help; --Получаем список таблиц и представлений Execute sp_help test_table -- информация о таблице</pre> <p>При работе с базой данных порой возникает ситуация, когда нужно удалить какой либо столбец. Нельзя удалить столбец, если у него указано какое-либо ограничение!!!</p>
Создание ограничений	<pre>Create table имя_таблицы (имя_поля1 тип_данных [identity NULL NOT NULL] Constraint имя_ограничения Тип_ограничения Имя_поля, имя_поля2 тип_данных [NULL NOT NULL] Constraint имя_ограничения Тип_ограничения Имя_поля,) Либо Create table имя_таблицы (имя_поля1 тип_данных [identity NULL NOT NULL] , имя_поля2 тип_данных [NULL NOT NULL], Constraint имя_ограничения1 Тип_ограничения Имя_поля1, Constraint имя_ограничения2 Тип_ограничения Имя_поля2,</pre>

)						
Ограничение Первичный Ключ	<div>Create table Test_table2</div> <div>(</div> <div>id int identity(1,1) not null Constraint PK_1 primary key(id),</div> <div>name varchar(25)</div> <div>)</div> <div>Смотрим, что получилось: Execute sp_help test_table2</div> <table><tr><th></th><th>constraint_type</th><th>constraint_name</th></tr><tr><td>1</td><td>PRIMARY KEY (clustered)</td><td>PK_1</td></tr></table> <div>Как раз то, что нам требовалось.</div> <div>А создать таблицу при помощи запроса, в которой будет составной первичный ключ, можно только используя ограничение.</div> <div>Create table Test_table3</div> <div>(</div> <div>name varchar(25) not null,</div> <div>project varchar(25) not null,</div> <div>date smalldatetime,</div> <div>exp_date int,</div> <div>Constraint PK_Mult1 primary key(name,project)</div> <div>)</div> <div>Т.о. мы получаем таблицу, у которой составной первичный ключ состоит из полей name и project.</div>		constraint_type	constraint_name	1	PRIMARY KEY (clustered)	PK_1
	constraint_type	constraint_name					
1	PRIMARY KEY (clustered)	PK_1					
Ограничение по Умолчанию	<div>Ограничение по умолчанию можно тоже установить, дав ему при этом указанное вами, а не SQL Server имя.</div> <div>Create table Test_table4</div> <div>(</div> <div>id int identity(1,1) not null constraint MyPK1 primary key(id),</div> <div>name varchar(25),</div> <div>rights varchar(25) not null constraint Rights_def1 default('user')</div> <div>)</div>						
Ограничение на Проверку	<div>Ограничение на проверку определяет, что прежде чем записать данные в ячейку, будет произведена проверка того условия, которое указано в ограничении.</div> <div>Create table Test_table5</div> <div>(</div> <div>id int identity(1,1) not null constraint MyPK2 primary key(id),</div> <div>name varchar(25),</div> <div>age int not null constraint age_chk1 check(age between 1 and 110),</div> <div>sex varchar(10) not null constraint sex_chk1 check(sex in('male','female'))</div> <div>)</div> <div>При добавлении значений в эту таблицу, при несовпадении условий, указанных в проверках, будет ошибка!!!</div> <div>Также можно указать ограничение, ссылаясь при этом на значения нескольких полей.</div> <div>Create table Test_table6</div> <div>(</div> <div>id int identity(1,1) not null constraint MyPK3 primary key(id),</div> <div>col1 int not null,</div> <div>col2 int not null,</div> <div>col3 int,</div> <div>Constraint My_Chk1 CHECK (col2>col1 and col2<col3)</div> <div>)</div>						
Ограничение Уникальности	<div>Существует возможность указать, что значения в поле не должны повторяться.</div> <div>Create table Test_table7</div> <div>(</div> <div>id int identity(1,1) not null constraint MyPK4 primary key(id),</div> <div>genre varchar(25) constraint MyUnique1 unique(genre)</div> <div>)</div>						

Ограничение Внешний Ключ	<p>Например. Создадим две таблицы и их свяжем.</p> <pre>Create table authors (id int identity(1,1) not null constraint MyPK5 primary key(id), name varchar(25) not null constraint MyUnique2 unique(name)) Create table Books (id int identity(1,1) not null constraint MyPK6 primary key(id), name varchar(25) not null, id_author int, Constraint Books_FK1 foreign key (id_author) References authors (id) On Update Cascade On Delete Cascade)</pre>
--------------------------	---

Модификация таблиц. Оператор ALTER TABLE

Общий синтаксис	<pre>Alter table Имя_таблицы { [Add имя_поля спецификаторы --Добавление столбца] [Alter Column имя_поля новые_спецификаторы --модифицирование столбца] [Drop имя_поля --удаление столбца] [Drop Constraint имя_ограничения --удаление ограничения] [Add Constraint имя_ограничения имя_ограничения тип_ограничения] }</pre>
-----------------	---

Добавление поля	<p><code>Alter table Table1</code> <code>Add Hello varchar(25)</code></p> <p>Поле со спецификатором NOT NULL может быть добавлено только если также будет указано ограничение по Умолчанию.</p> <p><code>Alter table Table1</code> <code>Add Hello2 varchar(25) not null constraint MyDf1 default('Hello')</code></p>
Модифицирование поля	<p>Единственное, что не подлежит модификации у поля при помощи запроса <code>Alter table имя_таблицы Alter column</code>, это изменение названия столбца. Это можно сделать только при помощи процедуры <code>sp_rename</code>.</p> <p>Необходимо поменять тип столбца Hello в таблице Table1 на int.</p> <p><code>Alter table Table1</code> <code>Alter column Hello int</code></p> <p>Вы не можете модифицировать либо удалить поле, у которого указаны ограничения, поэтому с начала необходимо удалить ограничения, и только потом модифицировать либо удалять поле.</p>
	<p><code>UPDATE table_name</code> <code>SET column1=value1,column2=value2,...</code> <code>WHERE some_column=some_value</code></p>
Удаление ограничений. Удаление полей	<p>Сначала необходимо удалить ограничение и только после этого само поле</p> <p><code>Alter table Table</code> <code>Drop Constraint MyDF1;</code></p> <p><code>Alter table Table</code> <code>Drop Column Hello2</code></p>
Добавление ограничений	<p>Если забыли указать ограничение при создании таблицы, то вы можете это сделать при помощи Alter table.</p> <p>Например, вам необходимо указать значение по умолчанию для поля Col1 таблицы Table2.</p> <p><code>Alter table Table2</code> <code>add constraint MyDF2 default (50) for Col1</code></p> <p>Необходимо использовать ключевое слово <code>for</code>, чтобы указать столбец, которому указывается ограничение. Для добавления ограничений на первичный ключ, уникальные значения и внешний ключ этого делать не нужно.</p>

Удаление таблиц. Оператор DROP TABLE

Общий синтаксис: `Drop Table Имя_таблицы`

Все очень просто, но есть несколько нюансов, о которых вы должны помнить.

1. Нельзя удалять таблицу, если ее первичный ключ связан с другой таблицей. С начала нужно удалить ограничения Внешний Ключ у таблиц, связанных с удаляемой.
2. Нельзя удалить системные таблицы.
3. Нельзя удалить таблицы, если у вас на это нет прав

Представления

Представления представляют из себя очень мощный механизм, позволяющий управлять введенными данными, а также позволяющий работать не со всей таблицей, а с ее частью, при этом требуемую часть можно указывать самостоятельно, плюс к этому существует возможность настраивать права доступа на какое то представление отдельно, тем самым упрощается система раздачи прав.

Положительная сторона работы с представлениями состоит в том, что по отношению к ним можно делать запросы на выборку, а также запросы на обновление информации, тем самым, изменяя структуру той таблицы, данные из которой находятся в представлении.

Общий синтаксис	<div>Create View Имя_представление AS Select Имя_поля1, Имя_поля2, Имя_поляN From Имя_таблицы Where Условие [With Check Option]</div> <div>Создается Представление, состоящее из значений, полученных из запроса Select. Имена полей представления будут совпадать с именами полей, указанными в запросе Select, но при желании их можно переименовать, указав псевдонимы для имен. Это удобно при создании представлений, использующем данные из нескольких таблиц.</div> <div>При написании многотабличного запроса Select, вы обязаны указывать псевдонимы для полей с одинаковыми именами, т.к. в противном случае будет ошибка при создании представления!!!</div>						
Пример	<div>Вычитать самую дешевую книгу из тематик Программирование, Базы данных клиент-сервер, Мультимедиа. База данных Books (многотабличная).</div> <div>Вытягиваем книги требуемых издательств.</div> <div>Create View MyView3 as select books.price,books.name as book,themes.name as theme From books,themes Where books.id_theme=themes.id and themes.name in('Программирование', 'Базы данных клиент-сервер', 'Мультимедиа')</div> <div>Из них находим книгу, у которой цена минимальная.</div> <div>Select book,theme From MyView3 Where price = (Select min(price) from MyView3)</div> <table><tr><td></td><td>book</td><td>theme</td></tr><tr><td>1</td><td>Oracle 8: рекомендации разработчикам</td><td>Базы данных клиент-сервер</td></tr></table>		book	theme	1	Oracle 8: рекомендации разработчикам	Базы данных клиент-сервер
	book	theme					
1	Oracle 8: рекомендации разработчикам	Базы данных клиент-сервер					
Модификация Представлений	<div>Alter View Имя_представления AS Select Имя_поля1, Имя_поля2, Имя_поляN From Имя_таблицы Where Условие</div> <div>Если вы заметили, то вся разница лишь в том, что слово Create заменено на Alter. Но Alter View использовать лучше в том случае, если вы заботитесь о раздаче прав. Если удалить представление, то все права на него аннулируются и после повторного создания нужно будет настраивать их заново, а Alter View позволяет изменить представление, и оставить права на него не тронутыми.</div>						
Удаление Представлений	<div>Drop View Имя_представления</div>						
Изменение данных через Представления	<div>Существует возможность изменить данные, которые находятся в представлении, тем самым будет произведено изменение значений в исходной таблице. Имеется в виду, что при удалении либо изменении, либо добавлении информации в представление, она изначально изменяется либо добавляется в ту таблицу, значения из которой находятся в представлении.</div>						

	<p>Существует набор ограничений на изменение данных через представления.</p> <ol style="list-style-type: none"> 1. При добавлении значений в представление, поля основной таблицы, имеющие спецификатор Not NULL, обязаны иметь значение по умолчанию. 2. Производить изменение данных в представлении, содержащем информацию из нескольких таблиц можно, только если изменения данных затрагивают только одну из них. 3. Если в представлении есть поля, являющиеся результатом функции агрегирования, то значения этих полей менять нельзя. 4. Если в представлении есть оператор UNION(он будет рассмотрен в следующих уроках), то к этому представлению можно делать только запросы на выборку. <p>Опция With Check Option позволяет указать, что ограничение, указанное в запросе Select обязано действовать при добавлении данных через представление.</p> <p>Пример.</p> <pre>Create View MyView4 as Select firstname, lastname From students Where firstname like 'A%' With Check Option</pre> <p>Будет создано представление, в котором будет список студентов, имена у которых начинается с буквы А. Если после этого будет произведена попытка добавления данных, то, учитывая что указано With Check Option, добавлять можно будет только людей, у которых имя начинается с буквы А.</p>
--	--

Системные таблицы

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

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

syscolumns	В ней содержится общее описание по каждому из полей таблиц, представлений и по каждому из параметров хранимых процедур, которые есть в базе данных. Например, обязательное ли поле для заполнения, либо нет, установлено ли значение по умолчанию и пр.
syscomments	Эта таблица хранит синтаксис определения все представлений, ограничений, хранимых процедур и триггеров. При удалении значения из этой таблицы вручную, может привести к некорректной работе того объекта, определение которого было удалено.
sysdepends	В этой таблице сохраняется информация о зависимостях между представлениями, хранимыми процедурами, триггерами и таблицами, которые указываются при их определении.
sysfilegroups	Содержит информацию о Группах файлов, которые есть в базе данных.
sysfiles	В ней хранится информация о логическом и физическом названиях файлов, о их максимальном размере и пр. Эта таблица является виртуальной и не может быть модифицирована напрямую.
sysforeignkeys	Хранит информацию обо всех ограничениях типа Внешний Ключ. Т.е. указывается внешний ключ какой таблицы находится в связи с первичным ключом другой таблицы.
sysindexes	Хранит информацию обо всех индексах, которые есть в базе данных.
sysindexkeys	Содержит информацию о индексах и полях, на которые они указывают.
sysmembers	В этой таблице храниться информация о пользователях и ролях, к которым они относятся. Роль - это набор прав и ограничений, который определяет возможность запрета доступа к одним данным и разрешение на доступ, и уровень доступа к другим данным.
sysobjects	Если таблица syscomments хранит синтаксис ограничений и хранимых процедур, то в этой таблице находится их описание. Например, уникальный номер, идентификатор владельца и т.д.
syspermissions	Здесь указываются права и ограничения, установленные на определенного пользователя, группу или роль.
sysreferences	Содержит карту связей таблиц.
systypes	Хранит список все типов данных (пользовательских и системных), которые могут использоваться при создании таблиц.
sysusers	В этой таблице указываются все пользователи и роли SQL Server и Windows.

Процедуры

Есть несколько основных положительных моментов у хранимых процедур:

1. После первого выполнения, хранимая процедура компилируется, и код ее выполнения хранится в кэше в оперативной памяти, что очень повышает быстродействие выполнения хранимой процедуры относительно обычных запросов и представлений.
2. Любой пользователь может получить доступ к хранимой процедуре, при этом он может не иметь права на работу с теми объектами, которые она в себя включает.

При работе с хранимыми процедурами, существует набор ограничений:

1. В хранимой процедуре не может быть двух переменных с одним и тем же именем.
2. В хранимой процедуре не может быть больше 1024 параметров.
3. В хранимой процедуре нельзя использовать директивы:
 - Create View
 - Create Procedure
 - Create Trigger
 - Create Role
 - Create Rule
 - Create Default

Общий синтаксис:

Create Procedure имя_хранимой_процедуры

As

Запрос_SQL

Чтобы потом ее вызвать, необходимо указать:

Execute имя_хранимой_процедуры

Пример. Необходимо создать хранимую процедуру, выводящую на экран список студентов, не вернувших книги

Create Procedure Taken_books

As

Select students.firstname,students.lastname,books.name

From students,s_cards,books

Where students.id=s_cards.id_student and s_cards.id_book=books.id
and s_cards.datein is null

Чтобы получить результат достаточно указать:

Execute Taken_books

Просмотр синтаксиса запроса процедуры	Execute sp_helptext Books_By_Theme
Просмотр структуры хранимой процедуры	execute sp_help Best_student
Шифрование	Create Procedure имя_хранимой_процедуры

процедуры	<p>@аргумент1 тип_данных[=значение_по_умолчанию][Output], With Encryption</p> <p>As Запрос_SQL</p>
Передача параметров в процедуру	<p>Create Procedure имя_хранимой_процедуры</p> <p>@аргумент1 тип_данных[=значение_по_умолчанию][Output], @аргумент2 тип_данных[=значение_по_умолчанию][Output],</p> <p>As Запрос_SQL</p> <p>Output определяет, что этот аргумент является возвращаемым значение этой ф-ции.</p> <p>Пример. Нужно показать книги определенной тематики, при этом шаблон названия тематики необходимо передать при вызове (многотабличная books).</p> <p>Create Procedure Books_By_Theme @Name varchar(25) As</p> <pre> Select books.name, themes.name from books,themes where books.id_theme=themes.id and themes.name like @Name </pre> <p>Вызываем: Execute Books_By_theme '[Г-М]%'</p>
	<p>Существует возможность вернуть какие-либо значения из функции. Для этого существует несколько вариантов.</p> <p>Первый. Указать при определении параметров спецификатор OUTPUT у тех параметров, которые получают результат. Для начала необходимо научиться объявлять собственные переменные. Для этого используется следующий синтаксис:</p> <p>Declare @имя_переменной тип_данных, @имя_переменной2 тип_данных, ... @имя_переменнойN тип_данных</p> <p>Чтобы записать значения в переменные используется следующий синтаксис:</p> <p>ВНИМАНИЕ!!!</p> <p>Select имя_переменной1 = значение1, имя_переменной2 = значение2, ... имя_переменнойN = значениеN</p> <p>Либо</p> <p>Set имя_переменной1 = значение1 Set имя_переменной2 = значение2 ... Set имя_переменнойN = значениеN</p> <p>Чтобы вызвать хранимую процедуру и получить возвращаемые значения используется следующий синтаксис: Предположим, что у хранимой процедуры 4 параметра и последних два нужно получить.</p> <p>ВНИМАНИЕ!!!</p> <p>Execute имя_хранимой_процедуры значение1, значение2, имя_переменной1 Output, имя_переменной2 Output</p> <p>Если не указать Output, то значения не сохраняться!!!</p>

	<p>Второй вариант вернуть значение - использовать ключевое слово return. Тогда можно не указывать параметры, в которые будет записан результат. Но при помощи return можно вернуть только одно целочисленное значение.</p> <p>Если вы хотите получить результат хранимой процедуры, возвращающей значение через return, необходимо использовать следующий синтаксис:</p> <pre> Declare @имя_переменной Execute @имя_переменной = имя_хранимой процедуры значение1, значение2 ... значениеN Select 'Значение переменной = ', @имя_переменной </pre> <p>Пример хранимой процедуры, складывающей 2 числа, переданных в качестве параметров.</p> <pre> Create Procedure MySum @a int,@b int As Declare @s int Set @s=@a+@b return @s </pre> <p>Вызов:</p> <pre> Declare @Summ int Execute @Summ=MySum 1,25 Select 'Summa = ',@Summ </pre>
<p>Операторы ветвления</p>	<p>При написании хранимых процедур можно использовать операторы ветвления. Это известные вам if, else if, else. Блоки, которые должны быть выполнены, заключаются между ключевыми словами Begin и End.</p> <p>Есть специальная функция, позволяющая, изначально, выдавать сообщение об ошибке. Но ее очень удобно использовать для компоновки выводимой на экран строки. Она называется Raiserror.</p> <pre> Raiserror ('строка со спецификаторами',степень_тяжести_ошибки,состояние_ошибки_на_момент_вызова, подставляемые_переменные); </pre> <p>В качестве спецификаторов может указываться:</p> <ol style="list-style-type: none"> 1. %d - целое число 2. %s - строка 3. %u - беззнаковое целое <p>Дробные, к сожалению, не поддерживаются.</p> <p>Степень тяжести указывается в пределах от 0 до 25. От 0 до 18 - могут указываться пользователями, а</p> <p>от 19 до 25 - критические ошибки, которые могут указывать только члены группы sysadmin. В случае таких ошибок подключение клиента с сервером разрывается</p> <p>Состояние_ошибки_на_момент_вызова Должно быть указано значение от 0 до 127.</p> <p>Подставляемые_переменные - те переменные, которые должны быть подставлены на место спецификаторов.</p>

	<p>Пример. Создаем хранимую процедуру, возвращающую максимальное из двух чисел.</p> <pre>Create Procedure MyMax @a int, @b int as if(@a>@b) return @a else return @b Declare @max int Execute @max=MyMax 8, 9 raiserror('Max from %d and %d is %d',0,1,8,9,@max) Max from 8 and 9 is 9</pre>
Цикл	<pre>while (логическое_выражение) begin тело end Create Procedure DistSum @a int, @b int as Declare @s int Set @s=@a while @a<@b begin Set @s=@s+@a+1 Set @a=@a+1 end return @s --Вызов: Declare @s int Execute @s=DistSum 2,4 raiserror('Summa chisel v diapazone = %d',0,1,@s) Summa chisel v diapazone = 9</pre>

Функции

Функции можно поделить на 3 вида:

1. **Скалярные.** Функции, возвращающие одно значение определенного типа.
2. **Однозапросные.** Функции, имеющие внутри только один Select запрос.
3. **Многозапросные.** Функции, внутрь которых помещен большой кусок кода, включающий операторы ветвления, циклы, запросы и пр.

Общий синтаксис	<pre>Create Function имя_функции (@имя_параметра1[=значение_по_умолчанию], @имя_параметра2[=значение_по_умолчанию], ... @имя_параметраN[=значение_по_умолчанию]) Returns возвращаемый_тип_данных [WITH ENCRYPTION] AS Begin ТЕЛО Return значение End Вызов: Либо Select имя_базы_данных.имя_владельца.имя_функции (передача_параметров)--параметры в круглых скобках Либо Declare @имя_переменной тип_данных Execute @имя_переменной=имя_функции значение1,значение2...значениеN/*достаточно указать только имя функции. Значения передаются без круглых скобок!!!*/ raiserror('значение = спецификатор_типа',0,1,@имя_переменной)</pre>
-----------------	---

Скалярные функции

Изначально хранимые процедуры возвращают значения, чтобы уведомить пользователя о какой-либо ошибке, к примеру. В основном же они используются для администрирования и быстрого выполнения набора запросов. Мы же их использовали для получения значений, которые нужны нам. Это лучше делать при помощи функций.

Например, необходимо создать функцию, возвращающую слово "Четное", если переданное число четное, "Ноль" - если передали ноль и "Нечетное" - если переданное число нечетное.	<pre>Create function Chet (@a int) returns varchar(25) As begin Declare @res varchar(25) if (@a=0) begin Set @res='Ноль' end else if (@a%2=0) begin Set @res='Четное' end else</pre>
---	--

	<pre> begin Set @res='Нечетное' end return @res end --Вызов: Select dbo.Chet(123) Либо: Declare @c varchar(25) Execute @c=Chet 28 raiserror('Число %s',0,1,@c) нечетное </pre>
Необходимо написать функцию, которая вернет кол-во подключений к серверу в текущий момент.	<p>Для этого нам понадобится таблица SYSPROCESSES, которая находится в базе данных Master. Поле LOGINAME сохраняет имена пользователей, подключенных к серверу. Все что требуется - их посчитать.</p> <pre> Create function UsersNum ()/*Если принимаемых параметров нет, то круглые скобки все равно должны быть указаны.*/ returns int as begin Declare @c int Select @c=count(Distinct(loginame)) from sysprocesses return @c end --Вызов: Select dbo.UsersNum () </pre>

Однозапросные функции

Здесь необходимо вам рассказать о еще одном типе данных. Это не скалярный тип, называемый TABLE. Он используется, когда из функции, к примеру, необходимо вернуть результат, представляющий набор строк и столбцов, что и представляет из себя таблицу. При этом вы обязательно должны помнить, что для каждого поля возвращаемой таблицы, должно указываться свое имя.

У возвращаемой таблицы должен быть следующий принцип создания: во **From указываются псевдонимы для используемых таблиц и во всем запросе используются эти псевдонимы. Результирующим полям также должны быть даны псевдонимы.**

Создание списка книг	<pre> Create function SpisokKnig2 () returns table as return (Select b.name as book, a.firstname+' '+a.lastname as author, c.name as Category, t.name as theme /*Для каждого поля указан псевдоним!!!*/ from books b, authors a, categories c, themes t </pre>
----------------------	--

	<pre> /*Для каждой таблицы указан псевдоним!!!*/ where b.id_author=a.id and b.id_category=c.id and b.id_themes=t.id) --Вызов: Select * from dbo.SpisokKnig2() </pre>
--	---

Многозапросные функции

Весь смысл этих функций состоит в том, что результирующая таблица не всегда создается при помощи одного запроса. Точнее, более или менее нужные функции всегда создаются при помощи набора запросов.

Общий синтаксис создания переменной	<pre> Declare @имя_переменной (имя_поля1 тип_данных спецификаторы, имя_поля2 тип_данных спецификаторы, ... имя_поляN тип_данных спецификаторы) </pre>
Общий синтаксис создания многозапросной функции	<pre> Create Function имя_функции (@имя_параметра1[=значение_по_умолчанию], @имя_параметра2[=значение_по_умолчанию], ... @имя_параметраN[=значение_по_умолчанию]) Returns @имя_переменной TABLE {определение полей возвращаемой таблицы} [WITH ENCRYPTION] AS Begin ТЕЛО Сохранение в возвращаемую переменную требуемых значений RETURN End </pre>
Пример – выбрать список книг по авторам, взятым студентами и преподавателями	<pre> Create function Books_By_Authors () returns @Books_Authors table (author varchar(25), amm int) /*Определяем переменную и тип возвращаемой таблицы!!!*/ as begin Declare @temp_books table (author1 varchar(25), amm1 int) /*Объявляем временную таблицу!!!*/ /*Записываем в нее список авторов и кол-во взятых студентами книг этих авторов!!!*/ insert @temp_Books Select authors.firstname+' '+authors.lastname,count(s_cards.id_book) from authors,books,s_cards where authors.id=books.id_author and books.id=s_cards.id_book group by authors.firstname,authors.lastname /*Дописываем в нее список авторов и кол-во взятых преподавателями книг этих авторов!!!*/ insert @temp_Books Select authors.firstname+' '+authors.lastname,count(t_cards.id_book) from authors,books,t_cards where authors.id=books.id_author and books.id=t_cards.id_book group by authors.firstname,authors.lastname /*Объявляем еще одну временную таблицу!!!*/ </pre>

	<pre> Declare @temp_books2 table (author2 varchar(25), amm2 int) /*Записываем в нее содержимое первой временной таблицы, сумируя при этом кол-во книг одного автора!!!*/ insert @temp_books2 Select t.author1,sum(t.amm1) from @temp_books t group by t.author1 /*Записываем содержимое в возвращаемую переменную!!!*/ insert @Books_authors Select t.author2, t.amm2 from @temp_books2 t return end --Вызов: Select * from Books_By_Authors()</pre>
<p>Функция, которая возвращает список книг, отвечающих набору критериев (например, имя автора, фамилия автора, тематика, категория), и отсортированный по номеру поля, указанному в 5-м параметре, в направлении, указанном в 6-м параметре</p>	<pre> CREATE FUNCTION SelectBooksByDefiniteParams(@authorFirstName varchar(50), @authorLastName varchar(50), @themesName varchar(50), @categoryName varchar(50), @sortColumn int, @sortDirection int) RETURNS @BooksTable table (НазваниеКниги varchar(500), ИмяАвтора varchar(50), ФамилияАвтора varchar(50), Тема varchar(50), Категория varchar(50)) AS BEGIN -- отбор вставляемых в таблицу полей INSERT @BooksTable SELECT Books.Name, Authors.FirstName, Authors.LastName, Themes.Name, Categories.Name FROM Books, Authors, Themes, Categories WHERE Authors.FirstName = @authorFirstName AND Authors.LastName = @authorLastName AND Themes.Name = @themesName AND Categories.Name = @categoryName AND Books.Id_Author = Authors.Id AND Books.Id_Themes = Themes.Id AND Books.Id_Category = Categories.Id ORDER BY CASE WHEN @sortColumn = 1 AND @sortDirection = 0 THEN Books.Name END ASC, CASE WHEN @sortColumn = 1 AND @sortDirection = 1 THEN Books.Name END DESC, CASE WHEN @sortColumn = 2 AND @sortDirection = 0 THEN Authors.FirstName END ASC, CASE WHEN @sortColumn = 2 AND @sortDirection = 1 THEN Authors.FirstName END DESC, CASE WHEN @sortColumn = 3 AND @sortDirection = 0 THEN Authors.LastName END ASC, CASE WHEN @sortColumn = 3 AND @sortDirection = 1 THEN Authors.LastName END DESC, CASE WHEN @sortColumn = 4 AND @sortDirection = 0 THEN Themes.Name END ASC, CASE WHEN @sortColumn = 4 AND @sortDirection = 1 THEN Themes.Name END DESC, CASE WHEN @sortColumn = 5 AND @sortDirection = 0 THEN Categories.Name END ASC,</pre>


```

CASE WHEN @sortColumn = 5 AND @sortDirection = 1 THEN Categories.Name END DESC

RETURN

END

--Вызов функции
SELECT * FROM SelectBooksByDefiniteParams ('Алексей',
                                             'Архангельский',
                                             'Программирование',
                                             'C++ Builder',
                                             1,
                                             0)

```

Триггеры

Триггер не имеет параметров и вызывается неявно. Он сродни капкану, когда вы его настраиваете на определенное действие, и когда это действие наступает, он срабатывает автоматически.

Триггеры можно поделить на две группы:

1. **INSTEAD OF** - это те, которые должны работать вместо указанного действия.
2. **FOR | AFTER** - те, которые должны работать во время, либо после указанного действия.

Возможные варианты действий:

1. **INSERT**
2. **UPDATE**
3. **DELETE**

Общий синтаксис:

```

Create Trigger имя_триггера
ON имя_таблицы
FOR {AFTER | INSTEAD OF} {INSERT | UPDATE | DELETE}
[WITH ENCRYPTION]
AS Тело

```

В SQL Server существует две таблицы - **INSERTED** и **DELETED**. Их можно и нужно использовать при написании триггеров. В эти таблицы данные попадают следующим образом: Если вы добавляете данные, то с начала они попадают в таблицу INSERTED, а только потом в основную таблицу, Если вы удаляете данные, то в начале они удаляются из основной таблицы, а потом попадают в таблицу DELETED. Структура этих таблиц абсолютно такая же как и у таблицы, над которой производится определенное действие. При **обновлении** информации производится удаление и последующая вставка данных. Т.о. с помощью данных в этих таблицах можно контролировать производимые изменения.

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

При создании триггеров существует набор правил:

1. Нельзя создавать триггеры во временных таблицах.
2. Триггеры не могут принимать параметры.
3. Триггер нельзя вызвать явно.
4. Триггеры не могут возвращать значения.
5. Для одного действия в одной таблице не может быть более одного триггера INSTEAD OF.
6. В триггерах нельзя использовать такие операторы:
 - DROP
 - CREATE
 - ALTER DATABASE
 - ALTER TABLE
 - GRANT
 - REVOKE
 - SELECT INTO

При каждом запросе на обновление данных в таблице Books вы хотите видеть сообщение о том, сколько записей было изменено. Для этого понадобится одна системная переменная, в которую автоматически записывается кол-во строк предыдущего запроса. Она называется **@@rowcount**.

```
Create Trigger Show_Upd_Amm
On Books
For Update
As
```

```
        raiserror('Было изменено %d записей',0,1,@@rowcount)
```

--Срабатывать этот триггер будет автоматически, при обновлении данных в таблице Books. Для проверки, увеличим кол-во всех книг издательства BHV на 3 экземпляра.

```
Update books
Set books.quantity=books.quantity+3
From press
Where books.id_press=press.id
and press.name like '%BHV%'
--Результат: Было изменено 9 записей
```

Необходимо при занесении данных о товаре выдать ошибку, если с дня производства прошло

```
Create Trigger Check_Date_trigger
On shop
for insert
as
Declare @InsDate smalldatetime
```

<p>больше месяца.</p>	<pre>Select @InsDate=date_in from inserted/*получаем дату товара, который добавляется (date_in - поле таблицы shop, в которую производится вставка данных)*/ if (@InsDate<=getdate()-30) /*Проверяем, не прошло ли 30 дней*/ Begin raiserror('Это слишком старый товар',0,1) raiserror('Данные о товаре сохранены не будут',0,1) Rollback transaction end else Begin Print('Insert Ok!!!') end</pre>
<p>Без триггера не обойтись при проверке данных нескольких таблиц. Например. Нельзя удалить диск (база данных CD), если он на верхушке рейтинга продаж.</p>	<pre>create trigger Check_cd_delete on cd for delete as Declare @SellAmm int, @cd_name varchar(25), @best_cd varchar(25) Select @cd_name=deleted.name from deleted/*Получаем название удаляемого диска.*/ Declare @Svodka table (imya varchar(25), kolv int) insert @Svodka/*Вычитываем информацию о названиях дисков и о их популярности*/ select cd.name,count(id_cd) from selling,cd where selling.id_cd=cd.id group by cd.name Select @best_cd=s.imya from @Svodka s/*Находим название самого популярного диска по продажам*/ where s.kolv=(Select max(kolv) from @Svodka) if(@best_cd=@cd_name)/*Проверяем совпадение названий.*/ begin raiserror('You can not delete this cd!!!',0,1) rollback transaction end else begin print ('deleting query was successfull!!!') end</pre>
<p>Также существует указать триггер вместо выполнения определенного запроса. Для этого необходимо указать INSTEAD OF.</p> <p>При удалении книги издательства 'BHV Киев' выдать ошибку.</p>	<pre>Create Trigger Not_BHV On books Instead of Delete As Declare @BHV_id int Select @BHV_id=id from press where press.name ='BHV Киев'/*Получаем идентификатор издательства BHV Киев*/ if (exists (select * from deleted where id_press=@BHV_id))/*Проверяем, есть ли такой идентификатор в удаляемых книгах.*/ raiserror ('You can not delete BHV Киев!!!',0,1)</pre>
<p>Также, триггеры необходимы,</p>	<pre>Create Trigger People_copy On People</pre>

если нужно, например, при вставке сдублировать данные в другую таблицу. Например, при добавлении какого-либо человека, скопировать данные о нем в резервную таблицу.	<pre>After Insert As Insert into copypeople select * from Inserted</pre>
Если книга добавляется в базу, она должна быть удалена из таблицы Удаленные	<pre>CREATE TRIGGER DeleteFromDeleted ON Books AFTER INSERT AS BEGIN DECLARE @nameBook varchar(100) SELECT @nameBook = Удаленные.Name FROM Удаленные, inserted WHERE Удаленные.Name = inserted.Name IF (@nameBook IS NOT NULL) BEGIN DELETE FROM Удаленные WHERE Удаленные.Name = @nameBook PRINT 'Книга добавлена в базу и удалена из таблицы Удаленные: ' + @nameBook END INSERT INTO Books VALUES ('Самоучитель Visual FoxPro 6.0', 512, 1999, 1, 11, 11, 2, 'Самоучитель', 1)</pre>
Чтобы при взятии определенной книги, ее кол-во уменьшалось на 1	<pre>CREATE TRIGGER DecreaseAmount ON S_Cards AFTER INSERT AS BEGIN UPDATE Books SET Books.Quantity = Books.Quantity - 1 FROM inserted WHERE Books.Id = inserted.Id_Book END -- выдача книги студенту текущей датой для проверки триггера INSERT INTO S_Cards (Id_Student, Id_Book, DateOut, DateIn, Id_Lib) VALUES (8, 1, GETDATE(), NULL, 2)</pre>
Чтобы при возврате определенной книги, ее кол-во увеличивалось на 1	<pre>CREATE TRIGGER IncreaseAmount ON S_Cards AFTER UPDATE AS BEGIN UPDATE Books SET Books.Quantity = Books.Quantity + 1 FROM inserted WHERE Books.Id = inserted.Id_Book END -- проверка триггера на возврат книги</pre>

	<pre>UPDATE S_Cards SET DateIn = GETDATE() WHERE Id_Book = 18</pre>
Чтобы нельзя было выдать книгу, которой уже нет в библиотеке (по кол- ву)	<pre>CREATE TRIGGER NotAllowToGiveWithNullAmount ON S_Cards FOR INSERT AS BEGIN DECLARE @amountBooks int SELECT @amountBooks = Books.Quantity FROM Books, inserted WHERE Books.Id = inserted.Id_Book IF (@amountBooks = 0) BEGIN print 'Этой книги нет в наличии' ROLLBACK TRANSACTION END END -- выдача книги студенту текущей датой для проверки триггера INSERT INTO S_Cards (Id_Student, Id_Book, DateOut, DateIn, Id_Lib) VALUES (2, 3, GETDATE(), NULL, 2)</pre>
Чтобы нельзя было выдать более трех книг одному студенту	<pre>CREATE TRIGGER NotAllowGiveMoreThreeBooks ON S_Cards FOR INSERT AS BEGIN DECLARE @amountTakenBooks int SELECT @amountTakenBooks = COUNT(S_Cards.Id_Student) FROM S_Cards, inserted WHERE S_Cards.DateIn IS NULL AND S_Cards.Id_Student = inserted.Id_Student IF (@amountTakenBooks > 3) BEGIN print 'Нельзя выдать студенту более трёх книг' ROLLBACK TRANSACTION END END -- выдача книги студенту текущей датой для проверки триггера -- осуществлена 2а раза удачно, 1 - нет INSERT INTO S_Cards (Id_Student, Id_Book, DateOut, DateIn, Id_Lib) VALUES (17, 3, GETDATE(), NULL, 2)</pre>
Чтобы при удалении книги, данные о ней копировались в таблицу Удаленные	<pre>CREATE TABLE Удаленные (Id int, Name varchar(100), Pages int, YearPress int, IdThemes int, IdCategory int, IdAuthor int, IdPress int, Comment varchar(50), Quantity int) CREATE TRIGGER InfoAboutDeletedBooks ON Books AFTER DELETE AS INSERT INTO Удаленные SELECT *</pre>

	FROM DELETED
Запрет на удаление таблицы	<pre>--CREATE TRIGGER create trigger notAllowDelete on Kievstar instead of delete as print 'Not allow to delete such table' DELETE FROM Kievstar</pre>
Проверка данных при вставке в таблицу	<pre>CREATE trigger with check summ CREATE TRIGGER notAbilitySumm ON Kievstar FOR INSERT AS DECLARE @sum money SELECT @sum = ValueMoney FROM Kievstar IF (@sum > 100000) ROLLBACK TRANSACTION INSERT INTO Kievstar VALUES ('0953335566', 'Name', 'MiddleName', 'Surname', 200)</pre>
Запрет на удаление более чем одной строки	<pre>CREATE TRIGGER notDeleteThanOneDate ON Kievstar FOR DELETE AS DECLARE @count int SELECT @count = COUNT(id) FROM deleted IF (@count > 1) ROLLBACK TRANSACTION DELETE FROM Kievstar WHERE id = 2</pre>
Увеличена сумма на 5%	<pre>CREATE TRIGGER updateDataForPrint ON Kievstar FOR UPDATE AS print 'Увеличена сумма на 5% ' + CAST(@@rowcount AS varchar(5)) --проверка триггера UPDATE Kievstar SET ValueMoney = ValueMoney*1.05</pre>
После обновления данных в таблице – печать всей таблицы	<pre>CREATE TRIGGER updateShow ON Kievstar AFTER UPDATE AS SELECT* FROM Kievstar UPDATE Kievstar SET ValueMoney = ValueMoney*1.05</pre>

Индексы

Для чего же они нужны. Изначально, данные в таблице сохранены не упорядоченно (на физическом уровне). Как это, вы спросите. Ну на самом деле, при добавлении данных в таблицу - она добавляется в конец ее, но две последовательно добавленных записи не будут находится "рядом". При удалении, удаленная область помечается как удаленная. Получается своего рода неупорядоченность.

А теперь рассмотрим возникающую проблему - Выборка данных. При выборке производится так называемый прямой перебор, просматривающий записи поочередно. Это аналогично тому, что мы бы искали книгу в библиотеке, причем они на полках выставлены были бы в порядке добавления. Понятное дело, что это очень длительный процесс и по мере увеличения объема, он будет занимать все больше времени.

Так вот ИНДЕКС первоочередно служит как раз для ускорения поиска по индексированному полю. Происходит такой процесс: Индекс создается, тем самым создается объект, который связывается с указанной таблицей, либо другим объектом. Индекс будет представлять из себя объект, в котором будет храниться пара типа "расположение данных в таблице" - "значение в поле индекса". Учитывая, что данные в индексе перегруппировываются при каждом добавлении либо удалении данных из таблицы, то поиск производится так - сначала поиск производится по значениям в поле Индекса, после этого берется адрес страницы данных, в которой находится требуемая запись, и поиск производится по записям найденной страницы. Аналогом является поиск по каталогу библиотеки. С начала по списки находим книги, и определяем полку, на которой она лежит. После этого ищем книгу на указанной полке.

Так что индексы создаются для тех полей, которые указываются в условии поиска, чтобы увеличить производительность. Также индексы используются для обеспечения уникальной идентификации хранимых данных.

После этого, обычно, возникает вопрос, а может проиндексировать все поля. Никто не спорит - можно, но ведь одна запись в индексе равна объему хранимого значения плюс объем данных, использующийся для хранения информации о месторасположении записи в таблице. Если проиндексировать все, то объем индексов, в результате, будет превышать размер самой хранимой информации.

Также вы должны помнить, что при вычитке больше 20% информации обычно оптимальность индексов теряется.

Если одновременно осуществляется обращение к нескольким полям, то существует возможность создать составной индекс.

```
Create Index имя_индекса On  
[имя_таблицы |  
имя_представления] (имя_поля1  
[ASC | DESC], имя_поля2, ...  
имя_поляN)
```

набора полей. ASC и DESC определяют направление сортировки для значений указанного поля. ASC - используется по умолчанию и при этом создается индекс со значениями, отсортированными по возрастанию, т.е. от А до Я. DESC - сортировка по убыванию. Если нужно одновременно иметь восходящий и нисходящий индексы, то их нужно создавать несколько, и с соответствующей сортировкой.

Пример.

```
create index MyIndex1 on books (name ASC)
```

Будет создан восходящий индекс на поле name таблицы Books. Запросы, у которых будет определено условие поиска по названию книги, будут производиться быстрее.

Еще один вариант проверить уникальность введенной комбинации значений полей:

	<p>Create Unique index MyIndex2 on authors (firstname,lastname)</p> <p>При попытке сделать запрос типа:</p> <pre>insert into authors(firstname,lastname) values ('Александр','Матросов')</pre> <p>возникнет ошибка, т.к. такая комбинация имени и фамилии уже есть.</p>
<p>Удаление Индексов</p> <p>При удалении помните, что нельзя удалить индекс с системной таблицы.</p> <p>Drop Index Имя_таблицы.Имя_индекса</p>	<p>Чтобы удалить индекс на таблицу books, необходимо написать следующий запрос:</p> <p>Drop Index Books.MyIndex1</p> <p>После удаления индекса, память, которая под него была выделена, освобождается.</p>
<p>Хранимая процедура SP_HELPINDEX</p>	<p>Служит, чтобы просмотреть информацию о том, какие индексы существуют у определенной таблицы.</p> <p>Execute sp_helpindex имя_таблицы имя_представления</p>

Администрирование БД

Модель защиты данных

Для начала, нужно определить принципы работы службы защиты данных.

С начала разберем разницу между авторизацией и аутентификацией. Аутентификация - это проверка возможности подключения к ресурсу. По сути это банальная проверка логина и пароля. Авторизация - это дополнительная проверка на то, имеет ли пользователь разрешение на использование этого ресурса. Как видите - это не одно и то же.

До этих пор мы использовали с вами смешанный режим аутентификации. При данном способе вы используете учетную запись Windows для того чтобы зарегистрироваться как системный администратор. Нам нужно углубиться в понимание процесса, который при этом происходит. Есть 3 уровня проверки:

1. Сетевое подключения к SQL Server. - Для этого необходима аутентификация в Windows.
2. Регистрация в SQL Server. - Не обойтись без аутентификации в SQL Server.
3. Регистрация в определенной базе данных. - Требуется аутентификация в базе данных.(Авторизация).

Идем по порядку:

Аутентификация в Windows.

Она необходима для адекватного входа в Windows. После входа в Windows производится проверка сетевого подсоединения и устанавливается соединение с сервером SQL Server. Далее в силу вступает следующий этап, это:

Аутентификация в SQL Server.

При регистрации в SQL Server вы должны ввести имя и пароль зарегистрированного пользователя. В случае правильного введения этих данных, пользователь подключается к SQL Server. Если возникает ошибка, то пользователь не сможет работать с SQL Server. В случае же успеха, необходимо пройти еще один этап:

Аутентификация в базе данных.

Для защиты от несанкционированного доступа, каждая база данных SQL Server имеет собственную защиту. Единственный вариант получить доступ - указать регистрационное имя, позволяющее обращаться к данным, хранящимся в базе данных.

Режимы защиты данных.

Переходим к ознакомлению режимов защиты, которые используются в SQL Server. Их 2:

1. Смешанный режим защиты.
2. Режим интегрированной защиты Windows.

Смешанный режим защиты.

При этом способе защиты пользователь может подключиться к SQL Server двумя способами, либо при помощи учетной записи Windows, либо при помощи учетной записи SQL Server. Т.о. этот режим защиты можно разделить на два: Режим Аутентификации Windows и Режим Аутентификации SQL Server.

Режим Аутентификации Windows.

Это самый удобный способ, если вы не хотите обременять пользователей вводить логины и пароли по нескольку раз. Этот режим аутентификации позволяет подключаться к SQL Server, введя логин и пароль при входе в Windows. Но при этом ответственность по управлению учетными записями полностью предоставляется операционной системе.

Я не буду рассказывать, как создавать пользователя, либо группу, т.к. вы это и так знаете (используйте MMC). Но после этого нужно указать SQL Server, что эти учетные записи могут быть использованы для подключения у нему. Для этого служат две хранимых процедуры:

1. **SP_GRANTLOGIN** - предоставление права регистрации.
2. **SP_REVOKELOGIN** - аннулирование прав регистрации.

Общий синтаксис:

```
Execute sp_grantlogin 'имя'
```

```
Execute sp_revokelogin 'имя'
```

Имя должно быть представлено в виде **Имя_компьютера\имя_пользователя**, либо **Имя_домена\имя_группы_пользователей**. Имя пользователя и Имя группы пользователей, как видите, может указываться как на локальной машине, так и на домене (при запуске SQL Server на домене) и определяет, что указанному пользователю, либо группе пользователей разрешается осуществлять подключение к серверу SQL .

группы Test_group имеют право на подключение к SQL Server	Результатом этого запроса должна быть строка: Granted login access to 'HOME\Test_group' . <pre>Execute sp_grantlogin 'HOME\Test_group'</pre>
Аннулирование прав.	<pre>Execute sp_revokelogin 'HOME\Test_group'</pre>
аннулирует право на доступ определенного пользователя из определенной группы	Если вы аннулировали право на доступ определенному пользователю, а группа, в которую он входит, имеет возможность регистрироваться в SQL Server, то и пользователь с аннулированными правами все еще будет иметь возможность подсоединяться. Чтобы этого избежать, необходимо воспользоваться хранимой процедурой SP_DENYLOGIN . <pre>Execute sp_denylogin [@loginame=]'логин'</pre> Эта хранимая процедура аннулирует право на доступ определенного пользователя из определенной группы.

Режим Аутентификации SQL Server.

При этом способе аутентификации пользователь должен ввести логин и пароль для подключения к серверу. Windows, при этом, участия не принимает. Список логинов и паролей хранится в базе данных

master в таблице sysxlogins. Этот способ подключения не настолько безопасен, как при Windows аутентификации и поэтому не является рекомендуемым для использования. Однако же на системах 9x всегда используется именно он.

При установке SQL Server в таблице sysxlogins автоматически будет создана учетная запись sa, даже если был выбран режим интегрированной защиты данных. Если установка производилась на Windows 2000, то вместо учетной записи sa будет создана группа системных администраторов, которые автоматически будут отнесены к роли sysadmin.

За пароли, хранящиеся в таблице sysxlogins, можете не беспокоиться, т.к. они представлены в зашифрованном виде и доступ к ним имеют только администраторы базы данных.

создания учетной записи в SQL Server	<pre>Execute sp_addlogin [@loginname='логин', [@passwd='пароль', 'имя_базы_данных']]</pre>
Как видите, добавляется учетная запись Test_user, с паролем test, и базой по умолчанию books.	<p>Логин - имя учетной записи, которая должна быть добавлена, Пароль - пароль, Имя_базы_данных - база данных по умолчанию. Это та база, которая будет загружаться автоматически при регистрации указанной учетной записи. Если ничего не указать, то по умолчанию будет открыта база данных master. Но этого не достаточно, чтобы иметь доступ к данным, хранящимся внутри.</p> <pre>Execute sp_addlogin @loginname='userForLook',@passwd='userForLook',@defdb='vacancies'</pre>

Просмотр прав учетки	<p>Чтобы просмотреть список всех учетных записей и баз данных, с которыми они в праве работать, необходимо воспользоваться специальной хранимой процедурой SP_HELPLOGINS.</p> <pre>Execute sp_helplogins</pre>
Удаление учетки	<p>Чтобы удалить учетную запись, необходимо использовать следующий синтаксис:</p> <pre>Execute sp_droplogin 'Test_user'</pre>
Смена пароля учетки	<p>Но это еще не все. Бывают случаи, когда нужно изменить пароль для существующего пользователя. Для этого служит хранимая процедура SP_PASSWORD. Общий синтаксис:</p> <pre>Execute sp_password [@old='старый_пароль', [@new='новый_пароль', [@loginname='логин']</pre> <pre>Execute sp_password @old='test', @new='new_test', @loginname='Test_user'</pre>
Изменение БД по умолчанию	<p>Также существует возможность изменить базу данных, заданную по умолчанию. Для этого нужна хранимая процедура SP_DEFAULTDB.</p> <pre>Execute sp_defaultdb [@loginname='логин', [@defdb='имя_базы_по_умолчанию']</pre>

Пользователи базы данных

Как мы уже говорили, для использования данных, хранящихся в базе данных не достаточно иметь возможность подключиться к серверу. Необходимо еще иметь доступ к самой базе данных. Если пользователь подключился к серверу, но не прошел авторизацию, то при попытке обращения к базе данных будет выведена ошибка. Т.о. далее речь пойдет о предоставлении доступе конкретным пользователям к определенным базам.

В каждой базе данных есть специальная таблица sysusers, в которую записывается информация о пользователях текущей базы данных.

ВНИМАНИЕ!!! Пользователь будет добавлен в текущую базу данных.

Имя_учетной_записи - это имя, которое указывается для подключения в SQL Server (т.е. кому предоставляется доступ). Имя_пользователя_в_БД - это имя, которое будет иметь пользователь в текущей базе данных. Этот параметр не обязателен. Если его не указать, то автоматически будет взято имя_учетной_записи.

Добавление пользователя	<pre>Execute sp_grantdbaccess [@loginame=]'имя_учетной_записи' [, [@name_in_db=]'имя_пользователя_в_БД'] use library --открытие базы данных library Execute sp_grantdbaccess 'test_user' /*предоставляем пользователю Test_user доступ к базе данных library.*/</pre>
Получение пользователей базы	<p>Чтобы получить список всех пользователей текущей базы данных необходимо воспользоваться хранимой процедурой SP_HELPUSER. Общий синтаксис:</p> <pre>Execute sp_helpuser [[@name_in_db=]'Имя_пользователя_в_БД']</pre> <p>Если не указать Имя_пользователя_в_БД, то отображены будут все пользователи текущей базы данных.</p>
Смена владельца БД	<p>При создании базы данных в нее всегда сразу же добавляется один пользователь: это ее владелец. У владельца есть полностью все права на работу с базой данных. Вы помните, что при установке SQL Server необходимо было указать имя пользователя. Там мы указывали sa. Так вот sa автоматически становится владельцем всех баз данных, которые есть на сервере (либо пользователь, указанный при установке). Изначально пользователь dbo соответствует учетной записи sa.</p> <p>Для поддержания безопасности системы иногда приходится менять владельца базы данных. Для этого служит хранимая процедура SP_CHANGEDBOWNER. Общий синтаксис:</p> <pre>Execute sp_changedbowner [@loginame=]'имя_нового_владельца'</pre> <p>Имя_нового_владельца это логин пользователя, который становится владельцем базы данных.</p>
Удаление	Для удаления пользователя используется хранимая процедура

пользователя	SP_REVOKEDBACCESS. Общий синтаксис: <code>Execute sp_revokedbaccess [@name_in_db=]'имя_пользователя_в_БД'</code> Имя_пользователя_в_БД - имя пользователя, которому закрывается доступ к текущей базе данных.
--------------	---

Существует еще одна учетная запись, позволяющая подключиться к базе данных - это пользователь **guest (гость)**. Эту учетную запись можно добавить в базу данных, при этом любой пользователь, не имея прав подключения к базе данных, может зайти в нее как гость. Эта учетная запись позволяет только сделать базу данных активной. Все остальные действия запрещены.

ВНИМАНИЕ!!! Учетная запись **guest** должна быть, чтобы пользователь, который имеет доступ к подключению к серверу, но не имеет права доступа ни к одной базе, все же мог подключиться без сообщения об ошибке.

Использование ролей. Роли уровня приложения.

Пришло время разобраться с понятием Ролей в SQL Server. Роль представляет из себя совокупность прав, которые доступны включенным в роль пользователям. В Windows вы сталкивались с таким понятием как группа (например, группа Администраторы). Роль - это аналог группы в Windows, но чтобы не было недопониманий, в SQL Server используется термин Роль. Роль очень удобно использовать для обращения к целой группе пользователей, которые в нее входят.

Первая роль, с которой вы познакомитесь - это роль **PUBLIC**. Любая база данных по умолчанию включает в себя эту роль и все пользователи, роли и группы входят в ее состав и не могут быть из нее удалены. Поэтому администраторы должны быть очень внимательны, при определении прав этой роли, т.к. дав этой роли права, эти права распространяются на всех абсолютно пользователей базы данных, даже тех, которые будут присоединены в дальнейшем.

Роли можно разделить на несколько видов:

1. Роли уровня сервера.
2. Роли уровня базы данных.
3. Роли уровня приложений.

Роли уровня сервера.

В SQL Server задано 8 ролей уровня сервера.

1. **sysadmin** - пользователи, включенные в эту роль, являются владельцами всех баз данных SQL Server. У этих пользователей есть права на выполнение **любых** операций с SQL Server, даже таких как распределение прав доступа среди других пользователей и настройка системы безопасности SQL Server.

Пользователь sa принадлежит этой роли, и удалить его из нее нельзя!!! Также закрыть доступ определенному пользователю роли sysadmin к определенной базе невозможно!!!

Эта учетная запись собирает воедино возможности всех остальных ролей.

2. **serveradmin** - это пользователи, которые являются администраторами сервера, но не имеют отношения к базам данных. Члены этой роли имеют возможность изменять параметры системы (sp_configure), завершать работу SQL Server, выполняя команду SHUTDOWN.

3. **setupadmin** - отнесенные к этой группе пользователи в праве определять хранимые процедуры, запускаемые при старте сервера. Также у них есть права добавлять, удалять и настраивать удаленные связанные серверы.
4. **securityadmin** - пользователи этой группы создают и управляют учетными записями SQL Server, а также определяют права доступа к базам данных. Они в праве изменять пароли всех пользователей, кроме пользователей, включенных в sysadmin. Также они имеют возможность считывать журнал ошибок.
5. **processadmin** - эта роль определяет возможность следить за процессами, которые происходят в SQL Server и в базах данных. Они в праве удалять зависшие запросы, используя для этого команду KILL.
6. **dbcreator** - обычно в эту роль заносятся пользователи, являющиеся владельцами баз данных, т.к. эта роль включает права на резервное копирование, восстановление баз данных и журналов транзакций, а также на создание, изменение, переименование и удаление баз данных.
7. **bulkadmin** - пользователи этой группы в праве вызывать оператор BULK INSERT, служащий обычно для массовой вставки данных.
8. **diskadmin** - эта группа определяет права на управление файлами, подключением устройства резервного копирования. Эта роль была создана для совместимости с предыдущими версиями SQL Server.

Добавление пользователя	<p>Для добавления определенного пользователя в определенную роль уровня сервера необходимо использовать системную хранимую процедуру SP_ADDSRVROLEMEMBER. Общий синтаксис:</p> <pre>Execute sp_addsrvrolemember [@loginname=]'логин_пользователя', [@rolename=]'имя_роли_уровня_сервера'</pre> <p>Добавим test_user к роли уровня сервера processadmin</p> <pre>Execute sp_addsrvrolemember @loginname='test_user', @rolename='processadmin'</pre> <p>Учтите, что для выполнения этой хранимой процедуры у вас должно быть достаточно прав.</p>
Удаление пользователя	<p>Для удаления пользователя из определенной роли уровня сервера используется хранимая процедура SP_DROPSRVROLEMEMBER. Общий синтаксис:</p> <pre>Execute sp_dropsrvrolemember [@loginname=]'логин_пользователя', [@rolename=]'имя_роли_уровня_сервера'</pre> <p>произведем удаление пользователя test_user из роли уровня сервера processadmin.</p> <pre>Execute sp_dropsrvrolemember @loginname='test_user', @rolename='processadmin'</pre>

Эти же операции можно произвести и с помощью Microsoft SQL Server Management Studio. Для этого нужно в требуемом сервере развернуть папку Security и зайти в Roles, после чего выбрать в контекстном меню требуемой роли уровня сервера пункт меню Properties.

Роли уровня базы данных.

Каждая база данных имеет в своем распоряжении набор ролей. Для каждой базы данных эти роли обособлены, и влияние на другие базы данных они не оказывают. Т.е. если пользователь получил права в одной базе, это не открывает ему доступ к другой базе данных.

Всего заранее определенных ролей уровня базы данных 9.

1. **db_owner** - эта роль назначается владельцам базы данных. Эти пользователи в праве манипулировать любыми данными и настройками в базе данных. В пределах текущей базы их права не ограничены.
2. **db_accessadmin** - это пользователи, которые имеют право на управление учетными записями в базе данных. В частности раздают и анулируют права на ее использование.
3. **db_securityadmin** - эта роль определяет права на администрирование системы защиты базы данных. Она включает возможность создавать роли, назначать пользователей в определенную роль и настраивать доступ к содержимому базы данных.
4. **db_ddladmin** - пользователи этой роли в праве манипулировать объектами базы данных (создавать, удалять, модифицировать), но у них нет прав на определение прав доступа к этим объектам.
5. **db_backupoperator** - эти пользователи имеют право производить резервное копирование базы данных.
6. **db_datareader** - эта роль определяет возможность всех входящих в ее состав пользователей считывать данные из любой таблицы, представления и функции, без ограничений.
7. **db_datawriter** - эти пользователи в праве добавлять, обновлять и удалять данные в текущей базе данных.
8. **db_denydatareader** - эта роль запрещает всем ее пользователям производить вычитку данных.
9. **db_denydatawriter** - пользователям этой роли запрещается производить добавление, обновление и удаление данных из таблиц и представлений текущей базы данных.

Создание собственной роли	<p>Для этого служит хранимая процедура SP_ADDROLE. Общий синтаксис:</p> <pre>Execute sp_addrole [@rolename='имя_роли' [,[@ownername='владелец_роли']</pre> <p>Имя_роли - это название создаваемой роли. Оно должно быть уникально в пределах базы данных. Владелец_роли - пользователь, который будет владельцем текущей роли. Если не указать этот параметр, то по умолчанию будет взята учетная запись dbo.</p> <p>Для того чтобы иметь право на добавление роли, пользователь должен быть либо членом роли уровня сервера sysadmin, либо членом роли уровня базы данных db_securityadmin.</p>
Удаление роли	<p>Чтобы удалить созданную роль необходимо использовать хранимую процедуру SP_DROPROLE. Общий синтаксис:</p> <pre>Execute sp_droprole [@rolename='имя_роли'</pre> <p>ВНИМАНИЕ!!! Удалить можно только роль, в которой нет ни одного пользователя</p>
Добавить пользователя	<p>Чтобы добавить какого либо пользователя в роль уровня базы данных используется</p>

в роль	<p>хранимая процедура SP_ADDROLEMEMBER.</p> <pre>Execute sp_addrolemember [@membername=]'имя', [@rolename=]'имя_роли_уровня_базы_данных'</pre>
Удаление пользователя из роли	<p>Для удаления пользователя из роли уровня базы данных используется хранимая процедура SP_DROPROLEMEMBER.</p> <pre>Execute sp_droprolemember [@membername=]'имя', [@rolename=]'имя_роли_уровня_базы_данных'</pre> <p>И в одной и другой хранимых процедурах имя - это либо имя пользователя, либо название роли.</p>

ВНИМАНИЕ!!! Добавить пользователя в определенную роль можно только в том случае, если этот пользователь добавлен в базу данных при помощи либо Microsoft SQL Server Management Studio, либо хранимой процедуры sp_grantdbaccess. В противном случае произойдет ошибка, т.к. имя добавляемого пользователя найдено не будет.

Хранимая процедура SP_HELPROLE позволят просмотреть список всех ролей, которые есть в базе данных.

Добавим пользователя test_user в сервер, потом в базу данных library и определим его в созданную нами же роль test_role.	<pre>execute sp_addlogin @loginame='test_user', @passwd='test'</pre> <p>/*Добавляем пользователя test_user в SQL Server.*/</p> <pre>use library /*Открываем базу данных.*/</pre> <pre>execute sp_grantdbaccess @loginame='test_user'</pre> <p>/*Добавляем пользователя test_user в активную базу данных.*/</p> <pre>execute sp_addrole @rolename='test_role'</pre> <p>/*Создаем в активной базе данных роль test_role.*/</p> <pre>execute sp_addrolemember @rolename='test_role', @membername='test_user'</pre> <p>/*Добавляем в роль test_role пользователя test_user.*/</p> <pre>execute sp_helprole</pre> <p>/*Просматриваем список существующих ролей.*/</p>
список пользователей текущей базы данных	<pre>execute sp_helpuser</pre>
Чтобы просмотреть конкретную роль, существует возможность передать ее имя в качестве параметра хранимой процедуры	<pre>execute sp_helprole</pre>

- Db_datareader – grants SELECT to all tables & views in a database
- Db_datawriter – grants INSERT, UPDATE and DELETE to all tables & views in a database
- Db_denydatareader – denys SELECT to all tables & views in a database

- Db_denydatawriter – denies INSERT, UPDATE and DELETE to all tables & views in a database
 - EXEC sp_addrolemember 'db_datawriter','username';
 - -- Add a user to a role
 - GRANT SELECT TO username;
 - -- Grant a permission to a user

Роли уровня приложений.

Казалось бы, чего еще не хватает. Но есть еще один уровень ролей, это роли уровня приложений.

Принцип функционирования лучше всего разобрать на примере. Представьте, у вас есть база данных и набор пользователей, для которых уже настроены права доступа. Предположим, в определенный момент времени, вам необходимо провести манипуляции с данными, при этом ваше приложение должно само указать права для всех остальных пользователей, в процессе своей работы. Для этого и служит роль уровня приложения.

Роль уровня приложения не может содержать в себе пользователей. Эта роль активируется, указав правильный пароль, и тогда все права ролей остальных пользователей игнорируются, и в силу вступают права роли на уровне приложения. Т.е. само приложение будет отвечать за права пользователей, и их не нужно будет переопределять вручную. Эта роль деактивируется только при отключении пользователя (приложения), который активировал роль уровня приложения, от сервера.

А теперь, рассмотрим принципы создания, удаления и активирования этой роли.

Создание роли	<p>Создается такая роль при помощи хранимой процедуры SP_ADDAPPROLE. Общий синтаксис:</p> <pre>Execute sp_addapprole [@rolename]='название_роли', [@password]='пароль'</pre> <p>Пример. Создаем роль уровня приложения:</p> <pre>Execute sp_addapprole @rolename='AppRole1', @password='App1'</pre> <p>Для ее активации необходима хранимая процедура SP_SETAPPROLE. Общий синтаксис:</p> <pre>Execute sp_setapprole [@rolename]='название_роли', [@password=] {Encrypt N 'пароль'}, [@encrypt=]'тип_шифрования'</pre> <p>Если не указать тип шифрования, то пароль будет передан серверу, как обычный текст. Также можно указать 'odbc'. Это значение будет определять, что нужно использовать специальную функцию для шифрования данных, прежде чем пароль будет послан серверу. После слова Encrypt указывается N, чтобы преобразовать строку в формат Unicode, в котором должна быть шифруемая строка</p>
Пример активирования заранее созданной роли уровня приложения.	<pre>Execute sp_setapprole 'AppRole1', {Encrypt N 'App1'}, 'odbc'</pre>
удаления роли уровня	<p>Для этого служит хранимая процедура SP_DROPAPPROLE.</p>

приложения	<p>Execute sp_dropapprole [@rolename=]'название_роли'</p> <p>Т.е. чтобы удалить созданную нами роль, необходимо сделать следующий запрос:</p> <p>Execute sp_dropapprole @rolename='AppRole1'</p>
изменить пароль роли уровня приложения	<p>Также можно изменять пароль роли уровня приложения, используя хранимую процедуру SP_APPROLEPASSWORD.</p> <p>Execute sp_approlepassword [@rolename=]'название_роли', [@newpwd=]'новый_пароль'</p>

Управление правами доступа

До сих пор вы учились, как создавать роли, их преимущества, мощь и пр. Все это время говорилось о том, какие права предоставляют роли. Теперь мы с вами разберем, как настраивать эти права для отдельно взятых пользователей и для группы в целом.

Все права доступа можно разделить на такие категории:

1. Специальные права доступа.
2. Объектные права доступа.
3. Командные права доступа.

Специальные права доступа.

Это права, которые заранее определены ролями на уровне сервера.

Например, пользователи, входящие в состав группы dbcreator имеют специальные права доступа, т.к. являются создателями базы данных.

Все остальные права, которые есть - это права пользователей.

Объектные права доступа.

Это права, которые представляют совокупность прав по управлению объектами базы данных. Объектом базы данных являются таблицы, представления, хранимые процедуры и пр. Указывая право доступа, необходимо четко указывать объект, для которого это право выдается.

К объектным правам доступа относятся:

- Select
- Insert
- Update
- Delete
- References
- Execute

Общий синтаксис раздачи таких прав доступа:

```
Grant {ALL [PRIVILEGES] | разрешение1, разрешение2... разрешениеN}
{
[имя_поля1, имя_поля2... имя_поляN]
ON {имя_таблицы | имя_представления}
| ON {имя_таблицы | имя_представления} [(имя_поля1, имя_поля2... имя_поляN)]
| ON {имя_хранимой_процедуры}
| ON {имя_пользовательской_функции}
}
TO имя_учетной_записи1, имя_учетной_записи2... имя_учетной_записиN
[With grant option]
```

- 1. All - определяет, что будут предоставляться все права, которые существуют у указанного объекта.
- 2. PRIVILEGES - ключевое слово, которое определяет, что после него указано перечисление предоставляемых прав доступа. Осталось для совместимости с ранними версиями.
- 3. *перечисление_разрешений*. Здесь перечисляются операторы, при помощи которых будет существовать доступ, к указанным ниже объектам. Например Select, Delete, Execute.
- 4. *перечисление_полей*. Это поля, доступ к которым предоставляется.
- 5. *Имя_таблицы* - это имя таблицы, к которой предоставляется доступ. (аналогично для представления, хранимой процедуры и функции).
- 6. TO - определяет список имен пользователей, либо ролей, на которые распространяются указанные выше права доступа.
- 7. With grant option - определяет, что перечисленные выше пользователи смогут предоставленные им права раздавать другим пользователям. **Но только если это будет разрешено командными правами доступа для пользователя.**

чтобы раздать право на вычитку и обновление информации пользователю test_user для таблицы books базы данных library	<pre>Grant Select, Update ON books TO test_user</pre> <p>Этот запрос определяет, что пользователь test_user получает право производить запросы Select и Update по отношению к базе данных books.</p>
права на добавление информации	<pre>Grant Insert ON books TO test_role</pre>

ВНИМАНИЕ!!! После этого запроса, все пользователи, включенные в роль test_role, получают право на добавление информации в таблицу books базы данных library. В том числе пользователь test_user.

Чтобы аннулировать право доступа используется ключевое слово **REVOKE**.

```
REVOKE [GRANT OPTION FOR]
{ALL [PRIVILEGES] | право1, право2... правоN }
{
[имя_поля1, имя_поля2... имя_поляN]
ON {имя_таблицы | имя_представления}
| ON {имя_таблицы | имя_представления} [(имя_поля1, имя_поля2... имя_поляN)]
| ON {имя_хранимой_процедуры}
| ON {имя_пользовательской_функции}
}
FROM имя_учетной_записи1, имя_учетной_записи2... имя_учетной_записиN
[CASCADE]
```

1. Grant option for - определяет, что с пользователя, либо роли снимается право на передачу определенных прав другим пользователям.
2. All - определяет, что аннулируются все права, которые существуют у указанного объекта.
3. PRIVILEGES - ключевое слово, которое определяет, что после него указано перечисление прав доступа, которые снимаются. Осталось для совместимости с ранними версиями.
4. *перечисление_прав*. Здесь перечисляется список объектных прав, которые аннулируются.
5. *перечисление_полей*. Это поля, доступ к которым закрывается.
6. *Имя_таблицы* - это имя таблицы, с которой снимаются права доступа. (аналогично для представления, хранимой процедуры и функции).
7. FROM - определяет список имен пользователей, либо имен ролей, для которых аннулируются указанные выше права доступа.
8. CASCADE - определяет, что доступ аннулируется не только для указанного пользователя, но и для тех, кому эти пользователи раздавали права.

Теперь снимем с пользователя test_user право на обновление	<p>Revoke Update ON books From test_user</p> <p>После этого test_user будет не в праве обновлять информацию, хранимую в таблице books.</p>
--	--

Но бывает еще один случай. Вы даете права определенной роли. Мы уже определили, что все входящие в нее пользователи, автоматически получают их (для этого роли и служат). А что делать, если для определенного пользователя должно действовать все, кроме определенного права. Именно в этом случае и поможет **Отказ в предоставлении объектных прав доступа**. Для этого используется ключевое слово Deny.

<pre> DENY {ALL [PRIVILEGES] разрешение1, разрешение2... разрешениеN} { [имя_поля1, имя_поля2... имя_поляN] ON {имя_таблицы имя_представления} ON {имя_таблицы имя_представления} [(имя_поля1, имя_поля2... имя_поляN)] ON {имя_хранимой_процедуры} ON {имя_пользовательской_функции} } TO имя_учетной_записи1, имя_учетной_записи2... имя_учетной_записиN [CASCADE] --Например, запретим право на добавление пользователю test_user. Deny Insert On books To test_user </pre>

После этого запроса, пользователь test_user не сможет добавлять информацию в books, но на остальных членов роли test_role это не повлияет.

ВНИМАНИЕ!!! Не важно, как пользователь получил право, через роль или оно было предоставлено ему напрямую. Запрос Deny, в любом случае, деактивирует указанные в списке права.

Командные права доступа.

К командным правам доступа относятся права типа:

1. CREATE DATABASE
2. CREATE DEFAULT
3. CREATE FUNCTION
4. CREATE PROCEDURE
5. CREATE RULE
6. CREATE TABLE
7. CREATE VIEW
8. BACKUP DATABASE
9. BACKUP LOG

ВНИМАНИЕ!!! Вы должны быть очень внимательны при раздаче таких прав, т.к. если у пользователя есть право на создание объекта, то при его создании он автоматически становится его владельцем и получает все права на использование!!! При предоставлении командных прав на создание, пользователь также получает права на изменение и удаление объектов!!!

Для раздачи таких прав используется также ключевое слово Grant.

```
Grant {ALL | разрешение1, разрешение2... разрешениеN}
TO имя_учетной_записи1, имя_учетной_записи2... имя_учетной_записиN
--Аннулирование командных прав доступа:
Revoke {ALL [PRIVILEGES] | право1, право2... правоN }
FROM имя_учетной_записи1, имя_учетной_записи2... имя_учетной_записиN
--И в командных правах доступа также есть отказ:
DENY {ALL [PRIVILEGES] | разрешение1, разрешение2... разрешениеN}
TO имя_учетной_записи1, имя_учетной_записи2... имя_учетной_записиN
--Например, даем право пользователю test_user на создание представлений.
Grant Create View To test_user
--Даем роли права на создание таблиц.
Grant Create Table To test_role
```

ВНИМАНИЕ!!! Право пользователю создавать таблицы не дает ему право на удаление других таблиц. Он может управлять только теми, которые сам создал.

ДОПОЛНЕНИЯ ПО БД

Иерархическая база данных – каждый объект при таком хранении информации представляется в виде определенной сущности, то есть, у этой сущности могут быть дочерние элементы, родительские элементы, а у тех дочерних могут быть еще дочерние элементы, но есть один объект, с которого все начинается. Получается своеобразное дерево. Примером иерархической базы данных может быть, документ в формате XML или файловая система компьютера,

Сетевые базы данных, являются своеобразной модификацией иерархических баз данных. Отлична от иерархических тем, что у дочернего элемента может быть несколько предков, то есть, элементов стоящих выше него.

Реляционная система управления базами данных:

- 1) Все данные в таких базах данных хранятся в таблицах
- 2) В основе реляционных систем лежит строгий безупречный математический аппарат.
- 3) Важной особенностью всех без исключения баз данных является то, что в базе данных помимо самих данных хранятся и описания этих данных — *метаданные*. Это позволяет сильно уменьшить зависимость программ от данных на логическом уровне.
- 4) Главным объектом реляционных баз данных являются таблицы.

Таблица

Особенности

- Таблица (table) содержит произвольное количество строк (row, записей (record)).
- Максимальное количество строк в таблице ограничивается объемом внешней памяти, доступной для хранения данных базы данных.
- Таблица может быть и пустой, т. е. не содержать ни одной строки.
- Все строки одной таблицы имеют **одинаковую** структуру. Они состоят из *столбцов* (column, полей (field))
- Таблица должна содержать как минимум один столбец.

Данные

- Основной характеристикой столбца является его тип данных (datatype).
- Каждый тип данных в SQL имеет имя.
- Типы данных могут быть предварительно определенными в системе (predefined), их иногда называют *системными, встроенными или базовыми типами данных*.
- Это также могут быть данные, *определенные пользователем* (user-defined). В некоторых системах есть еще один термин для пользовательских типов данных — *домен* (domain).
- *Типы данных* — числовые (целочисленные, дробные с фиксированной точкой и числа с плавающей точкой), строковые, логические, типы данных даты и времени.
- Существует тип данных, обычно называемый *двоичным большим объектом* (Binary Large Object, BLOB), который позволяет хранить любые большие по объему данные — форматированные тексты, изображения, звук, видео. По мере развития программной отрасли в мире программного обеспечения появляются новые типы данных, например XML, или пространственные (spatial) типы данных. Все эти типы данных поддерживаются в системе MS SQL Server.

Неизвестное значение NULL

- 1) Среди значений, которые может принимать столбец, в реляционных базах данных также используется и пустое или неизвестное значение NULL.
- 2) Не стоит смешивать его с нулевым значением у числового столбца или строкой с нулевой длиной для строкового типа данных. Такое значение присваивается тем столбцам, реальные значения которых нам не известны или которые в принципе неприменимы для конкретного объекта. Примером может служить дата рождения, которая часто требуется при описании какого-либо человека. Иногда бывает так, что эта дата нам просто не известна. При этом большинство задач обработки данных может решаться и при отсутствии таких данных. В этом случае полю присваивается значение NULL.

Индексы

- Объект базы данных *индекс* (index) используется для отдельных таблиц.
- Для каждой таблицы можно создавать один кластерный и до 999 обычных индексов.
- В таблице выбирается столбец или несколько столбцов, по которым формируется индекс. В результате в базе данных на внешнем носителе создается упорядоченная структура, которая будет содержать значения индексированных столбцов для каждой строки таблицы.
- Индексы позволяют ускорить процесс выборки данных из таблицы и процесс упорядочивания выбранных данных.
- Индексы также могут быть использованы для обеспечения уникальности значений столбцов, входящих в состав индекса.
- Можно создавать так называемые *кластерные индексы*. Такие индексы в самых нижних узлах своей структуры содержат и строки таблицы.
- В таблице может быть только один кластерный индекс. Кластерные индексы позволяют увеличить скорость выборки отдельных строк таблицы из базы данных.
- Не имеющая кластерного индекса таблица называется *кучей* (heap).
- Индексы создаются разработчиками базы данных для отдельных таблиц. В некоторых случаях система автоматически создает индексы для ключей таблицы.
- Хорошо созданные индексы могут сильно повысить производительность системы.
- В то же время безобразно спроектированные индексы могут резко снизить производительность.

Ключи в таблицах

<i>первичный ключ</i> (primary key) - это столбец или группа столбцов, значение которых однозначно определяет конкретную строку таблицы	<ul style="list-style-type: none">▪ Таблица может иметь один, и только один первичный ключ▪ Первичный ключ позволяет на основании значения столбцов, входящих в состав этого ключа, отыскать в базе данных ровно одну строку в указанной таблице или установить тот факт, что соответствующей строки в таблице не существует.▪ Основным требованием к первичному ключу является его уникальность. То есть в таблице не должно быть двух различных строк, имеющих одинаковое значение первичного ключа.▪ Ни один столбец, входящий в состав первичного ключа, не может иметь значения NULL (в описании таких столбцов должно, как правило, явно присутствовать предложение NOT NULL).▪ Первичные ключи часто присутствуют в реализации отношений между таблицами базы данных в связке "внешний ключ/первичный ключ".▪ Система управления базами данных автоматически создает индекс для первичного ключа таблицы. По умолчанию этот индекс является кластерным.
<i>уникальный ключ</i> (unique) - одним из назначений уникальных ключей является устранение дублирования значений	<ul style="list-style-type: none">▪ Каждая таблица может содержать произвольное количество <i>уникальных ключей</i>▪ В состав уникального ключа, как и в случае первичного ключа, может входить один или более столбцов таблицы.▪ В отличие от первичного ключа столбцы уникального ключа могут иметь значение NULL.▪ В таблице не может быть двух разных строк, имеющих одинаковое значение уникального ключа.▪ Одним из назначений уникальных ключей является устранение дублирования значений, как и в случае уникальных индексов.▪ Система управления базами данных автоматически создает индекс для каждого уникального ключа таблицы.▪ Индекс, создаваемый для уникального ключа, может быть кластерным, если для таблицы не существует другого кластерного индекса.▪ Уникальный ключ может присутствовать в связке таблиц вида "внешний

	ключ/ уникальный ключ".
<i>внешний ключ (foreign key) - это столбец или группа столбцов таблицы, которые ссылаются на первичный или уникальный ключ другой или этой же самой таблицы.</i>	<ul style="list-style-type: none"> Требование к значению столбцов, входящих в состав внешнего ключа, следующее: <ul style="list-style-type: none"> - либо все столбцы внешнего ключа должны иметь значение NULL, - либо таблица (<i>главная</i> или, иными словами, <i>родительская</i>), на первичный или уникальный ключ которой ссылается внешний ключ <i>подчиненной</i> (или <i>дочерней</i>) таблицы, должна иметь строку со значением первичного или уникального ключа, которое в точности равно значению внешнего ключа дочерней таблицы.

Задание первичных ключей таблиц

- Для каждой таблицы желательно использовать первичный ключ.
- Таблица может иметь только один первичный ключ.
- Важно правильно выбрать столбец или группу столбцов таблицы, которые войдут в состав первичного ключа.
- Основное требование к первичному ключу — его уникальность. В таблице не может быть двух разных строк, имеющих одинаковые значения первичного ключа.
- Второе реальное требование к первичному ключу — его относительно малый размер.
- Часто первичные ключи принимают участие в связке "внешний ключ/первичный ключ". Для реализации этого отношения подчиненные, дочерние, таблицы должны включать в

свой состав в качестве внешнего ключа столбцы, входящие в состав первичного ключа главной, родительской, таблицы. Кроме того, для первичного ключа система строит индексы. Все это в случае большого по размерам ключа увеличивает объем требуемой внешней памяти и может сильно ухудшить временные характеристики системы.

В общем случае, когда в базе данных нужно хранить различные сведения по людям, не привязываясь ни к каким организациям, учебным структурам, то лучшим решением будет использование *искусственного первичного ключа*. Иногда в литературе можно встретить термин "суррогатный" (surrogate) первичный ключ.

В состав столбцов таблицы в этом случае добавляется целочисленный столбец, который и будет искусственным первичным ключом. В SQL Server такой столбец должен быть описан с атрибутом **IDENTITY**. Столбцы, которым системой автоматически присваивается уникальное значение, в литературе называются *автоинкрементными* (auto increment).

В SQL Server есть еще один способ создания и использования искусственного первичного ключа. Это применение последовательностей (sequence) для получения уникального значения.

Отношения между таблицами в базе данных

Декларативная целостность данных (declarative data integrity) – связи между таблицами "внешний ключ/первичный ключ" и "внешний ключ/уникальный ключ".

Декларативная целостность обеспечивает непротиворечивость данных в базе данных — в случае правильного проектирования базы данных.

Декларативная целостность базы данных обеспечивается системой управления базами данных. Система отменяет все попытки добавления и изменения данных, которые нарушают заданную средствами операторов DDL-целостность (т. е. непротиворечивость) данных — в базу данных не может быть помещена строка таблицы, чей внешний ключ не соответствует ни одному значению первичного или уникального ключа родительской таблицы, на который ссылается этот внешний ключ. Нельзя также внести изменение в существующую строку таблицы, если изменяемое значение нарушает целостность данных.

Ограничения таблицы

<i>Вид ограничения</i>	<i>Описание</i>
Первичные, уникальные и внешние ключи таблиц	
<i>на значения, помещаемые в столбцы таблицы</i>	Это ограничение CHECK , благодаря которому в таблицу не может быть помещена новая строка или выполнено изменение данных уже существующей в таблице строки, если будет нарушено указанное ограничение. При задании ограничения можно указать довольно сложные условия, которым должно удовлетворять значение одного столбца или значения группы столбцов таблицы
<i>значение по умолчанию</i>	Это ограничение DEFAULT . Если при добавлении в таблицу новой строки не было задано значение какого-то столбца, то ему будет присвоено значение по умолчанию. Если при описании столбца не было явно указано значение по умолчанию (ограничение DEFAULT), то этим значением является NULL. Значение по умолчанию используется только при добавлении новой строки в таблицу, но не при изменении значений данных существующей строки;
<i>допустимость для столбца значения NULL</i>	Предложение NOT NULL в описании столбца запрещает помещать в этот столбец значение NULL.

Представления

Представление (view) — это объект базы данных, при обращении к которому происходит выборка данных из таблицы или из нескольких таблиц базы данных при помощи оператора SELECT или при обращении к хранимой процедуре. Представление позволяет скрыть от пользователя сложный процесс выборки данных. Кроме того, представление позволяет повысить безопасность данных, предоставляя пользователю только те данные, к которым у него существуют полномочия, за счет выдачи разрешения на представление, а не на базовую таблицу (таблицы).

Результатом обращения к представлению, как и в случае обычной выборки данных из таблицы при использовании оператора SELECT, является набор данных.

Представления бывают *изменяемые* и *неизменяемые*. Изменяемое представление позволяет вносить изменения в данные, полученные из представления, откуда они автоматически будут распространены в базовые таблицы представления, т. е. в таблицы, к которым обращается это представление. Неизменяемые представления такой возможности не предоставляют.

Хранимые процедуры и триггеры

- Язык SQL содержит подмножество языковых средств, называемое *языком хранимых процедур и триггеров* PSQL.
- В этом подмножестве можно описывать, каким именно образом выбирается очередная запись из базы данных, что нужно сделать с отдельными столбцами этой записи
- Существует возможность описания внутренних переменных, оператор присваивания, операторы ветвления, операторы циклов и другие императивные средства.
- возможно также применение рекурсии

Хранимые процедуры (stored procedure) – программы, хранящиеся в базе данных и выполняющие различные действия, обычно с данными из базы данных, хотя процедуры могут и не осуществлять никаких обращений к базе. К хранимым процедурам могут обращаться любые программы, работающие с базой данных, к ним также могут обращаться и другие хранимые процедуры и триггеры. Допустима и рекурсия, когда хранимая процедура обращается к самой себе. Хранимые процедуры выполняются на стороне сервера, а не на стороне клиента. Во многих случаях это может резко снизить сетевой трафик при решении различных задач работы с базой данных и повысить производительность системы.

Функции, определенные пользователем (user defined functions, UDF) - Это программные компоненты, к которым можно обращаться из триггеров, хранимых процедур, из других программных компонентов. Функции выполняют конкретные действия и возвращают ровно одно значение.

Триггеры (trigger), так же как и хранимые процедуры, являются программами, выполняющимися на стороне сервера. Однако напрямую обращение к триггерам件 невозможно. Они автоматически вызываются при наступлении некоторого события базы данных — например, при добавлении, изменении или удалении строк кон кретной таблицы. Триггеры могут вызываться при соединении с базой данных, а также в некоторых других случаях.

События базы данных (event). Хранимые процедуры и триггеры могут выдавать события — сообщения о появлении некой ситуации базы данных; такие сообщения могут перехватываться и обрабатываться клиентскими программами. Событиями могут быть ошибки в базе данных, которые выявляются не декларативным, а императивным способом — т. е. не при описании ограничений, таких как связка "внешний ключ/первичный (уникальный) ключ", а при выполнении более сложных проверок на соответствие вводимых данных требованиям предметной области. Часто события создаются при простых действиях с базой данных: при добавлении, изменении или удалении данных из конкретной таблицы. Они дают возможность проинформировать других клиентов о выполненных действиях. Такие события бывают полезными при синхронизации работы нескольких клиентов с одними и теми же данными в базе данных.

Пользователи, привилегии и роли базы данных

- Сведения о пользователях, имеющих доступ к базам данных экземпляра сервера, хранятся в самой системе.
- Местом хранения является внутренний каталог сервера.
- У пользователя, описанного в системе, есть имя и пароль.
- В SQL Server для авторизации пользователей рекомендуется использование средств авторизации операционной системы Windows.

Привилегии (полномочия) к объектам баз данных назначаются пользователям администратором базы данных. Привилегиями могут быть права на выполнение выборки, удаления, добавления и изменения данных конкретной таблицы базы данных, права на выполнение отдельных хранимых процедур, представлений.

Полномочия отдельному пользователю или группе пользователей могут назначаться прямым путем, а могут предоставляться при помощи механизма ролей (role).

Роль — это объект базы данных, которому назначаются некоторые полномочия к отдельным объектам базы данных. Затем роль может назначаться различным пользователям. В момент соединения с базой данных при указании роли пользователь получает все полномочия, предоставленные данной роли.

Транзакции

Транзакция является "механизмом" базы данных. Это некоторая законченная, иногда довольно сложная единица работы с данными и/или метаданными базы данных. Все операторы работы с базой данных (как с данными, так и с метаданными) выполняются в рамках — или еще говорится, *в контексте* — какой-либо транзакции. Исключением является оператор SELECT, который может выполняться и вне контекста транзакции. В контексте транзакции выполняется, как правило, группа операторов,

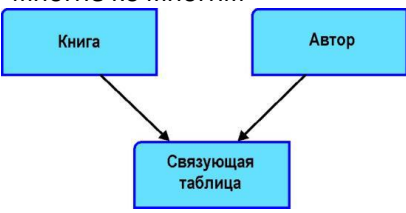
переводящих базу данных из одного непротиворечивого состояния в другое непротиворечивое состояние.

Все действия операторов одной транзакции могут быть либо подтверждены (оператор **COMMIT**), и тогда выполненные ими изменения будут зафиксированы в базе данных, либо отменены (оператор **ROLLBACK**). После подтверждения транзакции все изменения, выполненные операторами в ее контексте, станут видны другим параллельным процессам (иногда и неподтвержденные изменения бывают видны другим процессам, но об этом потом). Отмененные действия не сохраняются в базе данных.

Транзакциям могут задаваться некоторые характеристики, которые определяют поведение транзакции по отношению к другим параллельным процессам, а также допустимые одновременные действия других процессов.

Транзакции являются важным средством обеспечения одновременной работы с базой данных большого количества клиентских процессов, осуществляющих оперативную обработку данных. Понятно, что если два пользователя будут одновременно менять одну и ту же запись, ничего хорошего не получится. Второй клиент попросту отменит изменения, внесенные первым, причем первый ничего про это не узнает. Он будет пребывать в полной уверенности, что сделанные им изменения остались в базе данных. Не узнает о выполненных изменениях также и второй, хотя если бы он знал, что запись уже поменялась, возможно, он захотел бы внести туда совсем другие изменения. SQL Server использует два механизма для разграничения многопользовательской работы. Исторически более ранним является **механизм блокировок**. Когда первый пользователь меняет запись, она блокируется для изменений всеми остальными пользователями. Тогда второй пользователь не сможет внести в нее изменения одновременно с первым. Когда SQL Server "отпустит" запись, второй пользователь сможет прочитать уже ее новое значение и, если захочет ее изменить, сделает это, по крайней мере, осознанно. Существуют различные уровни строгости блокировок.

Реализация отношений в реляционной модели

"один к одному"	<p>Если между двумя таблицами базы данных появляется отношение "один к одному", то лучше объединить эти таблицы в одну. Основной причиной использования этого отношения является экономия памяти и увеличение скорости выполнения запросов.</p> <p>Такое отношение используется в том случае, если связь между двумя таблицами не является обязательной.</p> <p>Не так часто в реальной жизни встречаются случаи, когда требуется использовать отношение "один к одному". Например, адрес-человек</p>
"один ко многим"	<p>Такое отношение можно назвать <i>универсальным</i> — с его помощью можно представить практически любые отношения в базе данных.</p> <p>Пусть есть таблица, содержащая список стран. Вторая таблица содержит список регионов каждой страны (республики, области, штаты в США, графства в Великобритании).</p> <p>Первичным ключом таблицы стран будет некоторый код страны (существует международный стандарт для кодов всех стран). Во вторую таблицу, таблицу регионов, помимо остальных столбцов нужно добавить поле внешнего ключа (код страны), которое будет ссылаться на первичный ключ первой таблицы, таблицы стран. Первичным ключом второй таблицы нужно сделать составной ключ — код страны и код региона.</p>
"многие ко многим"	<div><pre>graph TD; Книга[Книга] --> Связующая[Связующая таблица]; Автор[Автор] --> Связующая;</pre></div> <p>между таблицей авторов и таблицей книг. Одна книга может быть написана несколькими авторами, один автор может написать несколько книг</p> <p>Реализация отношения "многие ко многим" осуществляется добавлением в базу данных третьей, связующей, таблицы и установлением двух необходимых связей "один ко многим".</p> <p>В данном примере добавляется связующая таблица между таблицей книг и таблицей авторов. Устанавливаются отношения "один ко многим" между книгой и связующей таблицей и "один ко многим" между автором и связующей таблицей</p>

	<p>Связующая таблица содержит только два столбца — "код книги" и "код автора", связывая книги и авторов.</p> <p>Столбец "код книги" в связующей таблице является внешним ключом, который ссылается на первичный ключ (код книги) в таблице книг. Столбец "код автора" — внешний ключ, ссылающийся на первичный ключ таблицы авторов.</p> <p>Первичный ключ связующей таблицы состоит из двух столбцов: "код книги" и "код автора". В данном случае при описании книг и их авторов комбинация значений этих столбцов является уникальной и может быть использована в качестве первичного ключа, однако часто существуют и иные ситуации, где для связующей таблицы приходится вводить свой искусственный первичный ключ.</p>
--	---

Нормализация таблиц

Первая задача при проектировании базы данных — составление списка таблиц и разработка структуры каждой таблицы.

Нормальные формы определяют правила, которым должны соответствовать структуры таблиц. Общеизвестными являются шесть нормальных форм, хотя в литературе по базам данных можно найти и гораздо большее количество форм нормализации. На практике обычно используется третья нормальная форма.

- Нормальные формы используются в таком порядке: первая, вторая, третья, форма Бойса — Кодда, четвертая и пятая.
- Выполнение правил нормализации для таблицы обычно приводит к разделению таблицы на две или более таблиц с меньшим количеством столбцов. Эти таблицы для наглядности отображения данных за счет связки "внешний ключ/первичный ключ" или "внешний ключ/уникальный ключ" снова могут быть соединены в процессе выборки данных в операторе SELECT при помощи операции соединения (JOIN).
- Одним из основных результатов разделения таблиц в соответствии с правилами нормализации является уменьшение избыточности данных.
- Подробные сведения о каждом объекте (или "сущности", entity) предметной области содержатся ровно один раз в конкретной таблице. В таблицах, где используются эти сущности, осуществляется лишь ссылка на нужную строку в соответствующей таблице.

Первая нормальная форма (1НФ или 1NF — first normal form)	значение любого столбца было единственным, атомарным. Иными словами, в таблице не должно быть повторяющихся групп																								
Вторая нормальная форма (2NF)	<ul style="list-style-type: none">▪ Необходимо соблюсти условия первой нормальной формы▪ любой неключевой столбец зависел от всего первичного ключа таблицы, а не от его части. Это правило относится только к тому случаю, когда первичный ключ образован из нескольких столбцов. <p>Таблица 2.3. Неверно спроектированная таблица регионов</p> <table><tr><th>Код страны</th><th>Код региона</th><th>Центр региона</th><th>Страна</th></tr><tr><td>RUS</td><td>32</td><td>Брянск</td><td>Россия</td></tr><tr><td>RUS</td><td>25</td><td>Владивосток</td><td>Россия</td></tr><tr><td>RUS</td><td>15</td><td>Владикавказ</td><td>Россия</td></tr><tr><td>RUS</td><td>33</td><td>Владимир</td><td>Россия</td></tr><tr><td>RUS</td><td>34</td><td>Волгоград</td><td>Россия</td></tr></table> <p>Первичный ключ для этой таблицы состоит из двух полей — "Код страны" и "Код региона". Столбец "Страна" зависит только от части первичного ключа: "Код страны". Этот столбец следует из таблицы просто убрать. Название (да и любые другие характеристики) страны всегда можно будет найти на основании значения внешнего ключа "Код страны".</p>	Код страны	Код региона	Центр региона	Страна	RUS	32	Брянск	Россия	RUS	25	Владивосток	Россия	RUS	15	Владикавказ	Россия	RUS	33	Владимир	Россия	RUS	34	Волгоград	Россия
Код страны	Код региона	Центр региона	Страна																						
RUS	32	Брянск	Россия																						
RUS	25	Владивосток	Россия																						
RUS	15	Владикавказ	Россия																						
RUS	33	Владимир	Россия																						
RUS	34	Волгоград	Россия																						

Третья нормальная форма (3NF)	<ul style="list-style-type: none">■ требует соблюдения условий второй нормальной формы■ ни один неключевой столбец не зависел от другого неключевого столбца <p>Таблица 2.4. Неверно спроектированная таблица отделов</p> <table><tr><th>Код отдела</th><th>Название отдела</th><th>Код руководителя</th><th>Фамилия руководителя</th></tr><tr><td>01</td><td>Продажи</td><td>384</td><td>Теплов</td></tr><tr><td>02</td><td>Маркетинг</td><td>291</td><td>Ожеред</td></tr><tr><td>03</td><td>Бухгалтерия</td><td>124</td><td>Майоров</td></tr></table> <p>Столбец "Код отдела" в этой таблице является первичным ключом. Столбец "Фамилия руководителя" зависит не от первичного ключа, а от неключевого столбца "Код руководителя". Столбец "Фамилия руководителя" следует убрать из таблицы.</p> <p>В базе данных должна уже существовать или быть вновь создана таблица, описывающая всех сотрудников организации. Первичным ключом такой таблицы должен быть код сотрудника. В таблице отделов через значение столбца "Код руководителя", который является внешним ключом, ссылающимся на первичный ключ таблицы сотрудников, всегда можно найти все необходимые характеристики руководителя отдела.</p>	Код отдела	Название отдела	Код руководителя	Фамилия руководителя	01	Продажи	384	Теплов	02	Маркетинг	291	Ожеред	03	Бухгалтерия	124	Майоров
Код отдела	Название отдела	Код руководителя	Фамилия руководителя														
01	Продажи	384	Теплов														
02	Маркетинг	291	Ожеред														
03	Бухгалтерия	124	Майоров														
Нормальная форма Бойса — Кодда (BCNF)	<p>является как бы развитием третьей нормальной формы. Она запрещает в качестве столбца, входящего в состав первичного ключа, использовать столбец, который функционально зависит от неключевого столбца, т. е. значение такого столбца можно выбрать из другой таблицы базы данных. Трудно себе представить разработчиков, которые могут создавать таблицы такой изощренной (или просто неразумной) структуры.</p>																
Четвертая нормальная форма (4NF)	<p>запрещает независимые отношения типа "один ко многим" между ключевыми и неключевыми столбцами. Это требование на представленном обычном языке звучит довольно странно, однако оно очень четко описывается математически в реляционной алгебре.</p>																
Пятая нормальная форма (5NF)	<p>доводит процесс нормализации до логического финала, разбивая таблицы на минимально возможные части для устранения в них всей избыточности данных. Нормализованная таким образом таблица обычно содержит минимальное количество данных (чаще всего только один столбец), помимо первичного ключа. При этом общий объем данных в базе данных за счет большого количества таблиц сильно увеличивается, что, как правило, ухудшает производительность системы.</p>																

Проектирование баз данных

концептуальный (содержательный) уровень	осуществляется анализ предметной области, для решения задач которой проектируется система обработки данных и база данных. Выявляются и описываются объекты (object) или сущности (entity) предметной области, их свойства, атрибуты (attribute), определяются связи, отношения (relationship) между сущностями, определяется список задач обработки данных, фиксируются требования к временным и иным характеристикам системы.
логический уровень	создается <i>логическая модель</i> базы данных. В первую очередь создаются все таблицы с использованием операторов Transact-SQL. Создаются необходимые триггеры, хранимые процедуры, представления, пользовательские функции. Часто этого

	бывает достаточно для получения хорошо спроектированной базы данных.
Физический уровень	<i>уровень</i> проектирования используется для настройки физических характеристик базы данных. Возможность влиять на физические аспекты хранения данных может привести к повышению производительности и даже надежности всей системы. MS SQL Server предоставляет средства тонкой настройки физических характеристик базы данных, позволяя для одной базы данных создавать несколько файлов данных, файловые группы, указывая, какие данные и в каком порядке должны размещаться в отдельных файловых группах, в конкретных файлах. Можно создавать секционированные таблицы, дающие возможность повысить производительность системы, ее отказоустойчивость.

Основные сведения о составе языка Transact-SQL

Язык SQL и его диалект, используемый в SQL Server, Transact-SQL, можно представить в виде группы подязыков, частей. По традиции каждый такой подязык называют языком.

В Transact-SQL выделяются следующие части:

<i>язык определения данных</i> (DDL, Data Definition Language);	DDL применяется для работы с объектами базы данных, с метаданными. Для действий с метаданными используются следующие группы операторов: <ul style="list-style-type: none"> CREATE. Это операторы, при помощи которых создаются новые объекты базы данных — в первую очередь таблицы, затем пользовательские типы данных, индексы, хранимые процедуры, триггеры, роли и др. При использовании оператора CREATE DATABASE создается и сама база данных. DROP. Операторы этого вида удаляют ранее созданные ненужные, как потом выяснилось, объекты базы данных. Те же таблицы, пользовательские типы данных и иные объекты. Оператор позволяет удалить и базу данных. ALTER. Это операторы, которые позволяют изменить уже существующие в базе данных ранее созданные объекты и характеристики базы данных.
<i>язык манипулирования данными</i> (DML, Data Manipulation Language);	Для работы с собственно данными в базах данных используются операторы DML, позволяющие создавать, изменять и удалять данные. В состав DML входит и оператор, выполняющий одну из наиболее важных функций в базе данных. Это оператор поиска, выборки данных. Для данных в базе данных используются четыре основных оператора: <ul style="list-style-type: none"> добавления данных INSERT; изменения существующих данных UPDATE; удаления данных DELETE; выборки (поиска) данных SELECT.
<i>язык управления доступом к данным</i> (DCL, Data Control Language);	Язык управления доступом к данным DCL содержит операторы, назначающие, отменяющие и удаляющие полномочия к объектам базы данных для пользователей и ролей, а именно: <ul style="list-style-type: none"> предоставления полномочий к защищаемому объекту GRANT; отмены полномочия DENY; удаления полномочия REVOKE.
<i>язык управления транзакциями</i> (TCL, Transaction Control Language);	Язык управления транзакциями TCL включает в себя операторы, осуществляющие запуск, подтверждение, откат или создание точки сохранения транзакции. Это следующие операторы: <ul style="list-style-type: none"> операторы старта обычной или распределенной транзакции: BEGIN

	<p>TRANSACTION и BEGIN DISTRIBUTED TRANSACTION;</p> <ul style="list-style-type: none"> ▪ операторы подтверждения транзакции: COMMIT TRANSACTION, COMMIT WORK; ▪ операторы отката транзакции: ROLLBACK TRANSACTION, ROLLBACK WORK; ▪ оператор создания точки сохранения SAVE TRANSACTION.
<p><i>язык хранимых процедур и триггеров</i> или процедурное расширение SQL (Stored Procedures and Triggers Language)</p>	<p>Язык хранимых процедур и триггеров содержит операторы, обеспечивающие процедурные императивные средства обработки данных. Язык используется в соответствии с его названием при создании хранимых процедур, функций, определенных пользователем, и триггеров.</p>