## Лекция 3

1. Повторение: Top и distinct, ORDER BY, TOP WITH TIES;

2. Полезные функции: IIF, CASE;

3. Работа с датой;

4. WHERE. Операторы IN, BETWEEN, LIKE.

**Домашнее задание:**

Можно выполнять БДЗ 1 (матрицы и схема данных) и МДЗ 1.

Обсудим, как хранить полиномы от одной переменной в БД. Предлагается следующая структура хранения полиномов в БД:

|  |
| --- |
| **Polynom** |
| **P\_id** |
| **Pow** |
| Coeff |

Здесь P\_ID – идентификатор полинома в БД (мы же планируем хранить сразу много полиномов в одной таблице), Pow – степень переменной, Coeff – коэффициент при этой степени. Понятно ли Вам, что в данной схеме с указанными ключами, мы можем хранить только приведенные полиномы?

Давайте положим полином в базу: x^2 + 1

|  |  |  |
| --- | --- | --- |
| **P\_ID** | **Pow** | **Coeff** |
| 1 | 0 | 1 |
| 1 | 2 | 1 |

Обратите внимание, что мы не положили в БД нулевой коэффициент при x в степени 1. В принципе в БДЗ полиномы могут храниться в любом виде, как с нулевыми коэффициентами, так и без. Но следует учитывать, что степень полинома – максимальная степень при ненулевом элементе, а колво нулевых коэффициентов нужно считать с учетом того, что не все нулевые коэфф хранятся в БД.

При выполнении БДЗ Вы можете называть столбцы в БД, как Вам нравится. Если у Вас иная схема хранения, то пришлите мне её для обсуждения (или можно обсудить на занятии).

Как хранить полиномы от нескольких переменных, мы подумаем чуть позже.

Заодно, обсудим хранение матриц в БД.

|  |
| --- |
| **Matrix** |
| **M\_id** |
| **col\_id** |
| **row\_id** |
| val |

**M\_id** – идентификатор матрицы в базе данных (как отличать одну матрицу от другой)

Col\_id – номер колонки

Row\_id – номер столбца

Val – значение элемента по указанному «адресу» (колонка + столбец).

Пример:

Давайте положим матрицу в базу:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| M\_id | col\_id | row\_id | val |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 4 |
| 1 | 2 | 1 | 2 |
| 1 | 2 | 2 | 5 |
| 1 | 3 | 1 | 3 |
| 1 | 3 | 2 | 6 |

Подумайте, всё ли Вам понятно?

**Тонкости работы** *Top и Order by:*

Если есть возможность, то сравните поведение Top (Limit) в MS Access, MS SQL, MY SQL, PostGre в случае дублирующихся записей.

Пример для сравнения:

|  |  |  |  |
| --- | --- | --- | --- |
| **ндок** | **Дата** | **Покупатель\_ID** | **Сумма** |
| 1 | 10.09.21 | 3 | 200 |
| 2 | 12.09.21 | 2 | 90 |
| 3 | 15.09.21 | 2 | 20 |
| 4 | 16.09.21 | 3 | 50 |

SELECT top 3 Покупатель\_ID

FROM Документы

В T-SQL Вы получите 3 записи, и в MS Access – 3.

Если же переписать запрос следующим образом:

SELECT top 3 Покупатель\_ID

FROM Документы

ORDER BY Покупатель\_ID asc

В T-SQL Вы получите 3 записи, и в MS Access – 4!

В T-SQL есть дополнительное служебное слово «With ties», используемое только с order by.

SELECT top 3 with ties Покупатель\_ID

FROM Документы

ORDER BY Покупатель\_ID asc

В случае отсутствия with ties – выведутся только 3 произвольные покупателя, в случае использования – выведутся все, которые совпадают с 3-мя наилучшими по критерию отбора, в данном случае – 4 строки. Проверьте, что произойдет, если в ORDER BY указать сортировку по дате (выведется опять 3 строки, как и в случае в MS Access).

SELECT top 3 with ties Покупатель\_ID

FROM Документы

ORDER BY Дата desc

PS: включение дополнительного поля в SELECT не повлияет на колво записей

**Упражнение на пройденный материал:**

Подумайте, понятны ли Вам результаты следующих упражнений:

|  |  |
| --- | --- |
| **id1** | **id2** |
| 1 | 10 |
| 1 | 20 |
| 2 | 30 |

|  |  |  |  |
| --- | --- | --- | --- |
| SELECT \*  FROM T  ORDER BY id1, id2 desc | SELECT TOP 1 id2  FROM T  ORDER BY id1 | SELECT TOP 1 WITH TIES id2  FROM T  ORDER BY id1 | SELECT TOP 1 WITH TIES id1  FROM T  ORDER BY id2 |

Почему не работает следующий запрос?

SELECT distinct id2

FROM @T

ORDER BY id1

**Полезная функция**

iif (в MS Access)/ CASE (в T-SQL). C SQL Express 2013 IIF есть и в T-SQL.

**Синтаксис:**

**Access и MS SQL (> 2008):**

iif(A=B; Выводится если Истина; Выводится если Ложь)

A и B – некие выражения (Например, Поле1 = 5 или Поле1 = Поле2)

**MS SQL:**

CASE

WHEN A = B THEN ЧтоВыводить

WHEN A = C THEN ….

ELSE что выводить иначе (является опциональным, если не выполняется ни одно из условий в WHEN, то выводится NULL – неопределенное значение)

END

Для CASE приведен только один пример синтаксиса. Подробнее можете прочитать в HELP.

Пример:

|  |  |
| --- | --- |
| **id1** | **id2** |
| 1 | 10 |
| 1 | 20 |
| 2 | 30 |

SELECT \*,

CASE ID1 WHEN 1 THEN 'Первый' WHEN 2 THEN 'Второй' ELSE 'Нет' END AS Коммент1,

CASE WHEN ID1 + ID2 < 20 THEN 'Сумма меньше 20 ' WHEN ID1 + ID2 > 10 THEN 'Сумма больше 10' ELSE 'Неизвестно' END Коммент2

FROM @T

|  |  |  |  |
| --- | --- | --- | --- |
| id1 | id2 | Коммент1 | Коммент2 |
| 1 | 10 | Первый | Сумма меньше 20 |
| 1 | 20 | Первый | Сумма больше 10 |
| 2 | 30 | Второй | Сумма больше 10 |

**Работа с датой:**

Дата – это дробное число, где целая часть – это количество дней, прошедших с «0», дробная часть – прошедшая часть от суток.

Например, 04.10.21 15:45:35 в числовом виде: целая часть (целое колво дней с «0» даты), дробная часть = (15\*3600+45\*60+35)/86400.

Отсюда очевидно, что 05.10.21 < 05.10.21 15:45:35.

Обратите внимание, что в Access/Excel и MS SQL разные нули. Поэтому если предпримете попытку перенести данные с датой в числовом виде из одной системы в другую, а потом переконвертируете обратно в дату, то получите результат с разницей в 2 дня.

Сегодня в Access (04.10.21): 43377.

Сегодня в MS SQL (04.10.21): 43375.

Сегодня в Excel (04.10.21): 43377.

Проверить это можно следующим образом:

**MS SQL:** SELECT CAST(GETDATE() as float)

Здесь: GETDATE() – функции, которые выдают текущую дату с временем.

В Postgre: current\_date, current\_timestamp

Если нужно **отфильтровать по месяцу/году/дню неделю/номеру недели**:

**Функции работы с датами:**

DatePart*("yyyy", Дата*) – выделяет нужную часть даты (в текущем синтаксисе - год)

DateSerial (*Year, Month, Day*) – собирает дату (только Access)

TimeSerial (*Hours, Minutes, Seconds*) – собирает время (только Access)

Year, Month, Day (Дата) – год, месяц, день от даты.

Extract (Postgre) или date\_part – аналог DatePart, date\_trunc (Postgre) – усекает дату до нужного размера.

MySQL: date\_part

**Если нужно что-то вычислить, связанное с датой:**

Можно из друг-друга вычесть даты, приведенные к float, а можно воспользоваться специальными функциями:

DATEDIFF(*datepart, startdate, enddate*) – считает разницу между начальной и конечной датой в единицах, определенных пользователем. Возвращает целое число.

DATEADD(*datepart***,** *number***,** *date*) – формирует новую дату, добавляя к заданной дате указанной колво указанных частей даты (секунд, минут, дней, лет и тд)

В PostGre: +, -

В MySQL: datediff, date\_add

**Функции приведения типов CAST и CONVERT**:

Как работать с функцией **CAST**:

**Тонкости с преобразованием типов (из MSDN)**

|  |  |  |
| --- | --- | --- |
| **Из** | **В** | **Поведение** |
| Float | Int | Truncate |
| Float | datetime | Round |
| datetime | Int | Round |

Подумайте, понятен ли Вам результат следующего запроса:

SELECT CAST(4.5 as int), CAST(4.2 as int), CAST(4.7 as int), 5/3,

CAST(GETDATE() as int), CAST(GETDATE() as float), CAST(CAST(GETDATE() as float) as int)

В MS SQL для приведения к числовому типу данных или к дате используются функции CAST (в PostGre) или CONVERT. Например, CAST(что-то as date) говорит, что нужно привести аргумент к типу данных дата (без времени). Так же есть следующие типы дат: datetime, date, time, smalldatetime и тд.

Например, CAST(Дата as date) – откинет время от даты, CAST (Дата as time) – оставит только время.

Даты могут быть представлены и отрицательными целыми числами:

MS SQL:

SELECT CAST(CAST('01.01.1800' as DATETIME) as float)

Подумайте, как из даты с временем сделать только дату без времени без использования CAST(Дата as date) (взять целую часть от числа)?

**Полезные функции в MS SQL для форматирования**: CAST, CONVERT, FLOOR, ROUND

Полезные функции в Postgre для **форматирования**: to\_char, to\_number\_to\_timestamp, to\_date.

**Как в целом работать с датой:**

При просмотре таблицы или результата запроса дата отображается независимо от Ваших настроек:

YYYY-MM-DD hh:mm:ss

|  |  |  |
| --- | --- | --- |
| **id** | **DT** | **D** |
| 1 | 2021-10-05 15:12:39 | 2021-10-05 |
| 2 | 2021-05-05 15:12:39 | 2021-05-05 |

Если нужно **отобрать нужные даты:**

В целом, в каждой сессии по умолчанию используются настройки, связанные, например, с вашим дефолтным языком в сессии. Вы можете его переписать прямо в сессии, но признаком хорошего тона считается использование языконезависимых форматов, хотя бы по причине того, что у другого пользователя Вашего запроса в базе могут быть важны именно его языковые настройки.

Языконезависимым форматом для даты считается формат, например:

‘yyyymmdd hh:mm:ss’, ‘yyyymmdd’ или ‘yyyy-mm-ddThh:mm:ss’ (в случае использования datetime)

‘yyyymmdd’ или ‘yyyy-mm-dd (в случае использования date)

Не забывайте про SET dateformat dmy (SET datestyle TO postgres, dmy). А лучше, используйте языконезависимые форматы даты.

В примерах в приложении Вы можете посмотреть на особенности работы с датой при различных дефолтных языках.

**Запросы с параметрами:**

Под запросом с параметром понимаются запросы, у которых в условии прописывается параметр, задаваемый пользователем каждый раз заново при запуске запроса. Смысл в том, что параметр можно использовать внутри одного запроса несколько раз, при этом только однократно изменив его значение. Это удобно, если Ваш запрос длинный и нужно запускать его несколько раз с разными значениями параметром. Так же это удобно, если в будущем Ваш скрипт станет функцией или процедурой с входным параметром.

**В Access:**

PARAMETERS [Имя покупателя] Text;

SELECT \*

FROM Документы

WHERE Покупатель = [Имя покупателя];

Параметров может быть несколько, указываются через запятую.

В случае использования QBE параметр прописывается точно также как обычное условие в строках «условие отбора». Предложение PARAMETERS прописывать необязательно. При запуске запроса Access предложить Вам ввести информацию вместо «Имя покупателя».

**В MS SQL:**

Комментарий по поводу MS SQL: большая просьба писать запросы в спец окне для запросов (Query Analyzer), которое вызывается по нажатию кнопки «New Query» или «Написать новый запрос».

Объявление параметра в T-SQL (на самом деле это просто задание значения переменной):

DECLARE @param as int -- объявление параметра типа int

SET @param = 1 -- установка значения параметра

SELECT \*

FROM Продажи

WHERE Товар\_iD = @param

Далее в этом же окне Query Analyzer можно обращаться к этой переменной по имени при написании запросов. В дальнейшем мы будем писать процедуру/функции, у которых есть входные параметры, которые задаются снаружи, поэтому можно заранее привыкать к тому, что если что-то можно задать в начале запроса в виде параметра, то лучше так и делать.

**В MySQL:**

set @q :=’название товара’;

SELECT \*

FROM Продажи

WHERE Товар = @q

**В PostGre:**

\set param 1

SELECT \*

FROM Продажи

WHERE Товар  = :param

**WHERE:** *Как использовать операторы like, In, Between.*

WHERE NOT (П1>Значение1 or П1 < Значение2) AND П2 <> значение3

Порядок приоритетов: сначала арифметические операции, потом NOT, AND и OR. Можно ставить скобки, но если Вы будете читать чужой код, то следует помнить про приоритеты.

Если у Вас текстовая строка типа nchar или nvarchar, то сравнение с другой строкой лучше проводить при помощи N’строка’, чтобы не снижать производительность запроса (не выключать индексирование, например).

**Оператор** **IN**:

Оператор **IN** определяет набор значений, в которое данное значение может или не может быть включено. Например, условие

SELECT \* FROM Документы

WHERE Покупатель\_ID in (1, 2)

выберет все документы, по которым закупались покупатели с идентификаторами 1 или 2. Аналог «ИЛИ»

**Оператор** **BETWEEN:**

Оператор **BETWEEN** в отличие от определения по номерам из набора, как это делает IN, BETWEEN определяет диапазон. Синтаксис такой: ключевое слово BETWEEN с начальным значением, ключевое **AND** и конечное значение. В отличие от IN, BETWEEN чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку.

SELECT \* FROM Товары

WHERE Масса BETWEEN 0.10 AND 0.12

*Причем, крайние значения также входят в набор. Аналог <= и >=*

**Упражнение 1:**

В наших документах присутствуют покупатели (предположим, что у нас в таблице Документы не Покупатель\_ID, а Покупатель и только фамилия): Иванов, Андреев, Петров, Сидоров. Что выдаст запрос:

SELECT \*

FROM Покупатели

WHERE Покупатель BETWEEN 'А' AND 'С';

**Оператор LIKE:**

**LIKE** следует применять только к текстовым полям. Оператор используется для поиска строк, подходящих под шаблон, например в тех случаях, когда точное написание искомого слова неизвестно, или требуется определить записи отвечающие группе условий.

Синтаксис (MS Access) таков:

SELECT \*

FROM Номенклатура

WHERE Наименование like “\*карта\*”

С помощью такого запроса можно получить товары, наименование которых содержит подстроку «карта»: видеокарта, звуковая карта и тд.

Для формирования шаблонов используются следующие символы:

|  |  |  |
| --- | --- | --- |
|  | Access | MS SQL (SQL Express) |
| **Любой одиночный символ.**  В приведенных примерах:  Под шаблон подходит как слово «апельсин», так и слово с орфографической ошибкой «опельсин» | “?” WHERE Наименование like “?пельсин” | “\_” WHERE Наименование like ‘\_пельсин’ |
| **Последовательность любого числа символов (включая отсутствие)**  В примере под шаблон подходят все слова, заканчивающиеся на «нов», в том числе и слово «нов», но не фамилия «Иванова». | “\*”  WHERE Наименование like “\*нов” | “%”  WHERE Наименование like ‘%нов’ |
| **Диапазон возможных символов:**  Вариант 1: указываем явно, что на первом месте в слове любая буква от А до О. В текущем примере под шаблон подходят любые слова заканчивающиеся на «пельсин» и начинающиеся на любую букву от А до О. То есть и «апельсин», и «бпельсин», и «опельсин» | WHERE Наименование like “[А-О]пельсин” | WHERE Наименование like ‘[А-О]пельсин’ |
| **Диапазон возможных символов:**  Вариант 2: Необязательно указывать диапазон, можно просто перечислить возможные буквы. В данном примере явно указали, что возможные написания либо «Апельсин», либо «Опельсин», либо «Юпельсин» | WHERE Наименование like “[АОЮ]пельсин” | WHERE Наименование like ‘[АОЮ]пельсин’ |
| Также можно указать, что на конкретном месте может быть любая буква, **кроме перечисленных в диапазоне.**  В указанном примере под шаблон попадают все слова, заканчивающиеся на «пельсин» и начинающиеся с любой буквы, кроме букв от Б до О. | WHERE Наименование like “[!Б-О]пельсин” | WHERE Наименование like ‘[^Б-О]пельсин’ |
| Если нужно найти слово, в котором есть служебный символ. Под шаблон в примере попадают любые слова, которые заканчиваются на «?пельсин» (в Access) или на «\_пельсин» (в MS SQL), и начинающиеся на произвольный одиночный символ. Например, «А?пельсин» (Access) или «А\_пельсин» (MS) | WHERE Наименование like “?[?]пельсин” | WHERE Наименование like “\_[\_]пельсин”  Либо читайте про Escape символы |

Для работы с датой просьба использовать операторы сравнения и функции работы с датой (см выше)! Ни в коем случае не пользоваться LIKE для фильтрафии по дате.

**Более подробную информацию по шаблонам можно почитать, например, здесь:**

ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de\_6tsql/html/581fb289-29f9-412b-869c-18d33a9e93d5.htm

Здесь можно найти информацию о том, что like на самом деле зависит от Collation Вашего сервера, а так же от конкретно выбранного типа данных.

**Пара слов про замыкающие пробелы:**

Посмотрите на результат запроса:

DECLARE @T TABLE (id int, txt1 char(10), txt2 nchar(10), txt3 varchar(10), txt4 nvarchar(10))

INSERT INTO @T

VALUES (1, 'a', 'a', 'a', 'a'), (2, 'a ', 'a ', 'a ', 'a ')

SELECT id, LEN(txt1) Len1, LEN(txt2) Len2, LEN(txt3) Len3, LEN(txt4) Len4

FROM @T

SELECT id, DATALENGTH(txt1) DLen1, DATALENGTH(txt2) DLen2, DATALENGTH(txt3) DLen3, DATALENGTH(txt4) DLen4

FROM @T

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Len1 | Len2 | Len3 | Len4 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | DLen1 | DLen2 | DLen3 | DLen4 |
| 1 | 10 | 20 | 1 | 2 |
| 2 | 10 | 20 | 2 | 4 |

Что мы видим, что при вставке строки с замыкающим пробелом в тип varchar и nvarchar, этот пробел никуда не исчезает. На него тоже выделяется место.

Коммент: В MySQL и PostGre функция длины (LENGTH) учитывает замыкающий пробел.

Поэтому, если Вам действительно не нужны замыкающие или начинающие пробелы, то пользуйтесь функциями LTRIM, RTRIM.

Почитайте про настройку: SET ANSI\_PADDING (влияет на то, как вставляются данные с замыкающими пробелами, никак не влияет на сравнение строк). По стандарту ANSI предполагается поведение с вставкой пробелов (как в примере выше).

Подумайте: как работает distinct, если в столбце указаны значения ‘a’, ‘a ’ для типа данных varchar?

MS SQL: выведется однократно

MySQL: выведется однократно

Post Gre: выведутся обе.

Как **сравниваются строки с замыкающими пробелами**:

Принцип (в соответствии с ANSI/ISO SQL-92) говорит: добавим в сравниваемые строки столько пробелов, чтобы их длина начала совпадать, после этого сравниваем. Поэтому строки ‘a’ и ‘a ’ равны (в смысле сравнения) в соответствии со стандартом.

SELECT \*

FROM @T

WHERE txt1 != txt2 OR txt1 != txt3 OR txt1 != txt4 OR

txt2 != txt3 OR txt2 != txt4 OR txt3 != txt4

и

DECLARE @p1 CHAR(1), @p2 NCHAR(1), @p3 CHAR(2), @p4 NCHAR(2)

SET @p1 = 'a'

SET @p2 = 'a'

SET @p3 = 'a '

SET @p4 = 'a '

SELECT id,

CASE WHEN txt1 = @p1 THEN 1 ELSE 0 END, CASE WHEN txt1 = @p2 THEN 1 ELSE 0 END, CASE WHEN txt1 = @p3 THEN 1 ELSE 0 END, CASE WHEN txt1 = @p4 THEN 1 ELSE 0 END,

CASE WHEN txt2 = @p1 THEN 1 ELSE 0 END, CASE WHEN txt2 = @p2 THEN 1 ELSE 0 END, CASE WHEN txt2 = @p3 THEN 1 ELSE 0 END, CASE WHEN txt2 = @p4 THEN 1 ELSE 0 END,

CASE WHEN txt3 = @p1 THEN 1 ELSE 0 END, CASE WHEN txt3 = @p2 THEN 1 ELSE 0 END, CASE WHEN txt3 = @p3 THEN 1 ELSE 0 END, CASE WHEN txt3 = @p4 THEN 1 ELSE 0 END,

CASE WHEN txt4 = @p1 THEN 1 ELSE 0 END, CASE WHEN txt4 = @p2 THEN 1 ELSE 0 END, CASE WHEN txt4 = @p3 THEN 1 ELSE 0 END, CASE WHEN txt4 = @p4 THEN 1 ELSE 0 END

FROM @T

Как работает **Like в случае замыкающих пробелов (необязательно для изучения начинающим)**:

Если все аргументы в Ansi формате, то сравнение по правилам ANSI (замыкающие пробелы исключаются), если же хотя бы один аргумент в Unicode, то сравнение по правилам Unicode и все замыкающие пробелы сохраняются.

В шаблоне важны ВСЕ символы. Если правая часть Like содержит замыкающий пробел, то SQL Server НЕ добавляет в поле пробелы до совпадения длины (в отличие от сравнение строк).

SELECT id,

CASE WHEN txt1 like @p1 THEN 1 ELSE 0 END r1,

CASE WHEN txt1 like @p2 THEN 1 ELSE 0 END r2,

CASE WHEN txt1 like @p3 THEN 1 ELSE 0 END r3,

CASE WHEN txt1 like @p4 THEN 1 ELSE 0 END r4

FROM @T

id r1 r2 r3 r4

1 1 0 1 0

2 1 0 1 0

txt1 char(10): 'a' и 'a '

SELECT id,

CASE WHEN txt2 like @p1 THEN 1 ELSE 0 END r1,

CASE WHEN txt2 like @p2 THEN 1 ELSE 0 END r2,

CASE WHEN txt2 like @p3 THEN 1 ELSE 0 END r3,

CASE WHEN txt2 like @p4 THEN 1 ELSE 0 END r4

FROM @T

id r1 r2 r3 r4

1 0 0 0 0

2 0 0 0 0

Txt2 nchar(10): 'a' и 'a '

SELECT id,

CASE WHEN txt3 like @p1 THEN 1 ELSE 0 END r1,

CASE WHEN txt3 like @p2 THEN 1 ELSE 0 END r2,

CASE WHEN txt3 like @p3 THEN 1 ELSE 0 END r3,

CASE WHEN txt3 like @p4 THEN 1 ELSE 0 END r4

FROM @T

id r1 r2 r3 r4

1 1 1 0 0

2 1 0 1 1

Txt3 varchar(10): 'a' и 'a '

SELECT id,

CASE WHEN txt4 like @p1 THEN 1 ELSE 0 END r1,

CASE WHEN txt4 like @p2 THEN 1 ELSE 0 END r2,

CASE WHEN txt4 like @p3 THEN 1 ELSE 0 END r3,

CASE WHEN txt4 like @p4 THEN 1 ELSE 0 END r4

FROM @T

id r1 r2 r3 r4

1 1 1 0 0

2 0 0 1 1

Txt4 nvarchar(10): 'a' и 'a '

Пример для повторения:

DECLARE @T TABLE (id int, txt nvarchar(10))

INSERT INTO @T

VALUES (1, 'a '), (2, ' a'), (3, 'a')

SELECT \*, LEN(txt), DATALENGTH(txt)

FROM @T

id txt

1 a 1 4

2 a 2 4

3 a 1 2

SELECT distinct txt

FROM @T

a

a

Синтаксис запросов с группировкой мы выучим позже: главное здесь – это смысл, что группируем по одинаковым значениям поля txt.

SELECT txt, COUNT(\*)

FROM @T

GROUP BY txt

a 1

a 2

SELECT \*

FROM @T

WHERE txt like 'a'

3 a

SELECT \*

FROM @T

WHERE txt = 'a'

1 a

3 a

**MS Access:**

SELECT IIF("a"="a ",1,0), IIF("a" Like "a ",1,0), IIF("a " Like "a ",1,0)

FROM T;

T – произвольная таблица, можно с одной строкой.