## Лекция 3

**Темы:**

- Top и distinct;

- Полезные функции: IIF, CASE;

- Работа с датой;

- WHERE. Операторы IN, BETWEEN, LIKE.

- Запросы с группировкой. GROUP BY, HAVING, Агрегатные функции.

**Домашнее задание:**

Можно выполнять БДЗ 1 (матрицы) и МДЗ1 и 2.

**Лекция:**

Обсудим, как хранить матрицы В БД:

Предлагается следующая структура хранения 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? При выполнении БДЗ 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).

Пример для TOP и Distinct:

|  |  |  |  |
| --- | --- | --- | --- |
| **ндок** | **Дата** | **Покупатель\_ID** | **Сумма** |
| 1 | 10.10.16 | 3 | 200 |
| 2 | 12.10.16 | 2 | 90 |
| 3 | 15.10.16 | 2 | 20 |
| 4 | 16.10.16 | 3 | 50 |

**Совместное использование TOP и distinct.**

Имейте в виду, что сначала «работает» distinct, а после TOP.

Сравните результаты запросов (distinct top можно использовать без ORDER BY):

SELECT distinct top 2 Покупатель\_ID

FROM Документы

ORDER BY Покупатель\_ID asc

Выведутся значения 2 и 3 по одному разу

И

SELECT top 2 Покупатель\_ID

FROM Документы

ORDER BY Покупатель\_ID asc

(выведутся 2 значения «2»)

**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, 16.10.15 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 16.10.15 < 16.10.15 15:45:35.

Обратите внимание, что в Access 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 (13.10.16): 42656.

Сегодня в MS SQL (13.10.16): 42654.

Сегодня в Excel (13.10.16): 42656.

Проверить это можно следующим образом:

**MS Access:**

SELECT Cdbl(CDATE(Now()))

FROM T; (T – произвольная таблица хотя бы с 1-й таблицей)

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

Здесь: Getdate() и Now() – функции, которые выдают текущую дату с временем.

В Postgre: current\_date, current\_timestamp

В Access: функция CDATE() используется для приведения к типу данных DATETIME, функция CDBL() используется для приведения к числовому типу данных.

В MS SQL для приведения к числовому типу данных или к дате используются функции CAST (в Post Gre) 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:

Access:

SELECT Cdbl(CDATE("01.01.1800"))

FROM T; (в T должна быть хотя бы одна произвольная строка)

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 Access для этого**: CLng, Int, Format и тд.

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

**Немного по функциям Access**:

**Format**(*expression*[, *format*[, *firstdayofweek*[, *firstweekofyear*]]])

Возвращает строковую переменную в нужном формате. Соответственно, если нужно перевести в числовое (дату и тд), то следует дополнительно использовать: Cint, Cdbl, Cdate, Cstr.

Для даты format:

m- месяц, d, dd- день, ww – номер недели в году, yy (yyyy) – номер года, w – день недели

Также формат может быть задан пользователем:

Format(5459.4, "##,##0.00") ' Returns "5,459.40".

Format(334.9, "###0.00") ' Returns "334.90".

Format(5, "0.00%") ' Returns "500.00%".

Все примеры использования можно посмотреть в help к Access или MSDN.

С помощью FORMAT необязательно работать только с датами, можно работать и с числами и строками, делать все буквы строчными или заглавными и тд.

**Пример форматирования даты в Access:**

Например, CDate(Format("10.01.13";"mm.dd.yy")) будет восприниматься как дата 01.10.13.

**Как в целом работать с датой:**

|  |  |
| --- | --- |
| id | data |
| 1 | 06.10.16 |
| 2 | 10.06.16 |

Если нужно **отобрать нужные даты:**

SELECT \*, MONTH(data), YEAR(data)

FROM T

WHERE data >= '20161001'

SET dateformat DMY

SELECT \*

FROM T

WHERE data >= '01.10.16'

SET dateformat MDY

SELECT \*

FROM T

WHERE data >= '01.10.16'

Не забывайте про SET dateformat dmy (SET datestyle TO postgres, dmy).

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

**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)

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)

**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.*

**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** применим только к текстовым полям. Оператор используется для поиска подстрок, 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**:

Если правая часть Like содержит замыкающий пробел, то SQL Server НЕ добавляет в шаблон или поле пробелы до совпадения длины. (в отличие от сравнение строк).

Кроме того, в шаблоне важны ВСЕ символы. В выражении пробелы не важны, если это ANSI тип данных, и важны, если это Unicode (хотя бы с одной стороны, шаблона или выражения).

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.

**Запросы с Группировкой полей:**

Что такое запросы с группировкой? В случае, если нам необходимо посчитать некоторую агрегатную величину по каким-либо полям, используются запросы с группировкой.

Возможные агрегатные функции: SUM, AVG, MIN, MAX, *FIRST, LAST*, STDEV, COUNT и тд. FIRST и LAST есть только в MS Access (уже и в MS SQL).

Пример запроса:

SELECT Покупатель\_ID, SUM (Сумма)

FROM Документы

GROUP BY Покупатель\_ID

Такой запрос сгруппирует поля из документов по покупателю и просуммирует значения покупок из документов. То есть в результате такой запрос посчитает по каждому покупателю сумму его закупок из таблицы Документы. (Группировка – берем поля, указанные в выражении GROUP BY, для одинаковых комбинаций вычисляем одну величину).

Пример:

Были данные

|  |  |  |  |
| --- | --- | --- | --- |
| **Ндок** | **…** | **Покупатель** | **Сумма** |
| 1 |  | Петров | 1 |
| 2 |  | Иванов | 10 |
| 3 |  | Сидоров | 14 |
| 4 |  | Петров | 20 |
| 5 |  | Сидоров | 21 |

Результат запроса:

Петров 21

Иванов 10

Сидоров 35

Запрос можно переписать как

SELECT SUM (Сумма)

FROM Документы

GROUP BY Покупатель\_ID

В результате выполнения не выведется фамилия покупателя, но сумма будет посчитана по каждому персонально.

Что выведется, если написать

SELECT SUM (Сумма)

FROM Документы

?

Подумайте, что будет в результате (поле Дата есть в таблице Продажи):

SELECT Покупатель, Дата, SUM(Сумма)

FROM Документы

GROUP BY Покупатель, Дата

В MS SQL есть следующая возможность:

SELECT COUNT(Покупатель), COUNT(distinct Покупатель), COUNT(\*)

FROM Документы

Что выведет такой запрос?

Так же в MS SQL distinct можно использовать и внутри других агрегатных функций, например SUM(distinct Сумма).

Подумайте, есть ли различия в результатах следующих запросов?

|  |  |
| --- | --- |
| SELECT Покупатель  FROM Документы  GROUP BY Покупатель | SELECT distinct Покупатель  FROM Документы |

**Комментарий:**

Использующим MySQL:

Такой синтаксис в MySQL:

SELECT Покупатель, MAX(Сумма)

FROM Документы

не выдаст ошибку синтаксиса, но покупатель, которого выведет этот запрос – будет произвольным, совсем не обязательно тем, которому соответствует максимальная сумма.

Упражнение: Как без использования подзапросов вывести покупателей, которые купили суммарно на максимальное количество денег?

SELECT TOP 1 WITH TIES Покупатель\_ID, SUM(Сумма) Оборот

FROM Документы

GROUP BY Покупатель\_ID

ORDER BY 2 desc

Обратите внимание на то, что сортировать можно по номеру поля в SELECT.

**WHERE и HAVING**

Если в результирующем запросе требуется провести отбор по групповому значению, то необходимо использовать команду HAVING. Раздел HAVING может содержать только те аргументы, которые имеют единственное значение для группы.

Можно ли переносить условия из WHERE в HAVING?

Варианты:

SELECT Покупатель, SUM (Сумма\_RUR)

FROM Документы

GROUP BY Покупатель

HAVING Покупатель = ‘Иванов’

и

SELECT Покупатель, SUM (Сумма\_RUR)

FROM Документы

WHERE Покупатель = ‘Иванов’

GROUP BY Покупатель

равносильны.

А вот запросы:

SELECT Покупатель, SUM (Сумма\_RUR)

FROM Документы

WHERE Сумма\_RUR > 100

GROUP BY Покупатель

и

SELECT Покупатель, SUM (Сумма\_RUR)

FROM Документы

GROUP BY Покупатель

HAVING SUM (Сумма\_RUR) > 100

выдадут абсолютно разные результаты.

Упражнение 1: Почему???

Упражнение 2: Что выдаст запрос:

SELECT Покупатель, SUM (Сумма\_RUR)

FROM Документы

GROUP BY Покупатель

HAVING MIN (Сумма\_RUR) > 100

Что будет результатом следующего запроса:

DECLARE @T TABLE (id int, val float)

INSERT INTO @T VALUES (1, 10), (2, 15)

SELECT MIN(id)

FROM @T

HAVING SUM(val) > 10

Есть небольшая тонкость в использовании HAVING:

Следующий запрос в MS Access отработает, а на SQL Server нет.

SELECT SUM(Сумма)

FROM Документы

HAVING Покупатель\_ID = 2

А вот следующий запрос не будет работать ни там, ни там.

SELECT Покупатель\_ID

FROM Документы

HAVING Покупатель\_ID = 2

Такой запрос будет работать в Access.

SELECT SUM(Сумма)

FROM Документы

WHERE Сумма > 1

HAVING Покупатель\_ID = 2

Но все же настоятельно предлагается писать запросы в соответствии со спецификацией.