**Модуль №5 «Функции агрегирования»**

**Слайд №2.** **Цели занятия:**

* закрепить навыки построения многотабличных запросов, применения операторов языка DML;
* научиться применять агрегатные функции: COUNT, AVG, SUM, MIN, MAX;
* ознакомиться с понятием группировки данных;
* выяснить разницу между использованием ключевых слов HAVING и WHERE.

**Слайд №3.** **План занятия:**

1. Актуализация знаний.

2. Выполнение лабораторной работы на тему: «Многотабличные базы данных».

3. Функции агрегирования: функция COUNT;

* функция AVG;
* функция SUM;
* функция MIN;
* функция MAX.

4. Понятие группировки. Ключевое слово GROUP BY.

5. Ключевое слово HAVING:

* принципы использования HAVING;
* сравнительный анализ HAVING и WHERE.

6. Домашнее задание.

**Слайд №4.** Функции агрегирования.

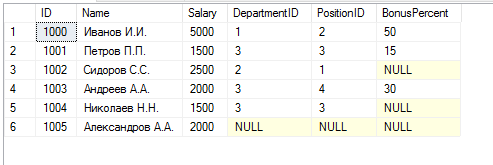
* **Агрегатные функции** – это функции, которые выполняют вычисление на наборе значений и возвращают одиночное значение.
* Все агрегатные функции являются *детерминированными*. Это означает, что агрегатные функции возвращают одну и ту же величину при каждом их вызове на одном и том же наборе входных значений.

**Слайд №5.** Основные и наиболее часто используемые агрегатные функции:

|  |  |
| --- | --- |
| **Название** | **Описание** |
| COUNT(\*) | Возвращает количество строк полученных оператором «SELECT … WHERE …». В случае отсутствии WHERE, количество всех записей таблицы. |
| COUNT(столбец/выражение) | Возвращает количество значений (не равных NULL), в указанном столбце/выражении |
| COUNT(DISTINCT столбец/выражение) | Возвращает количество уникальных значений, не равных NULL в указанном столбце/выражении |
| SUM(столбец/выражение) | Возвращает сумму по значениям столбца/выражения |
| AVG(столбец/выражение) | Возвращает среднее значение по значениям столбца/выражения. NULL значения для подсчета не учитываются. |
| MIN(столбец/выражение) | Возвращает минимальное значение по значениям столбца/выражения |
| MAX(столбец/выражение) | Возвращает максимальное значение по значениям столбца/выражения |

Агрегатные функции позволяют нам сделать расчет итогового значения для набора строк, полученных при помощи оператора SELECT.

**Слайд №6.** Пусть мы имеем некоторую таблицу Employees:



**Слайд №7.** Рассмотрим каждую агрегатную функцию на примере:  
**SELECT**

**COUNT**(\*) [Общее кол-во сотрудников],

**COUNT**(**DISTINCT** DepartmentID) [Число уникальных отделов],

**COUNT**(**DISTINCT** PositionID) [Число уникальных должностей],

**COUNT**(BonusPercent) [Кол-во сотрудников у которых указан % бонуса],

**MAX**(BonusPercent) [Максимальный процент бонуса],

**MIN**(BonusPercent) [Минимальный процент бонуса],

**SUM**(Salary/100\*BonusPercent) [Сумма всех бонусов],

**AVG**(Salary/100\*BonusPercent) [Средний размер бонуса],

**AVG**(Salary) [Средний размер ЗП]

**FROM** Employees

**Слайд №8. Результаты, вычисленные при помощи агрегатных функций**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Общее кол-во сотрудников | Число уникальных отделов | Число уникальных должностей | Кол-во сотрудников у которых указан % бонуса | Максимальный процент бонуса | Минимальный процент бонуса | Сумма всех бонусов | Средний размер бонуса | Средний размер ЗП |
| 6 | 3 | 4 | 3 | 50 | 15 | 3325 | 1108.33333333333 | 2416.66666666667 |

Для задания псевдонимов колонок мы воспользовались синтаксисом […].

**Разберем, каким образом получилось каждое возвращенное значение, а за одно вспомним конструкции базового синтаксиса оператора SELECT.**

**Возврат к слайду №7.** Во-первых, т.к. мы в запросе не указали WHERE-условия, то итоги будут считаться для детальных данных, которые получаются запросом:

**SELECT** \* **FROM** Employees

т.е. для всех строк таблицы Employees.

**Слайд №9.** Для наглядности выберем только поля и выражения, которые используются в агрегатных функциях:

**SELECT**

DepartmentID,

PositionID,

BonusPercent,

Salary/100\*BonusPercent [Salary/100\*BonusPercent],

Salary

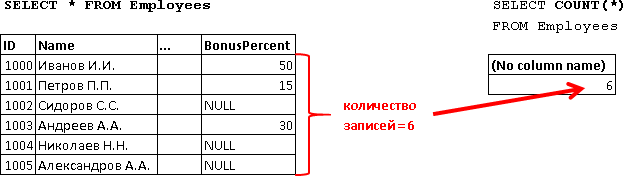
**FROM** Employees

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DepartmentID** | **PositionID** | **BonusPercent** | **Salary/100\*BonusPercent** | **Salary** |
| 1 | 2 | 50 | 2500 | 5000 |
| 3 | 3 | 15 | 225 | 1500 |
| 2 | 1 | NULL | NULL | 2500 |
| 3 | 4 | 30 | 600 | 2000 |
| 3 | 3 | NULL | NULL | 1500 |
| NULL | NULL | NULL | NULL | 2000 |

Это исходные данные (детальные строки), по которым и будут считаться итоги агрегированного запроса.

**Теперь разберем каждое агрегированное значение:**

**Слайд №10. COUNT(\*)** – т.к. мы не задали в запросе условия фильтрации в блоке WHERE, то COUNT(\*) дало нам общее количество записей в таблице, т.е. это количество строк, которое возвращает запрос: **SELECT** \* **FROM** Employees



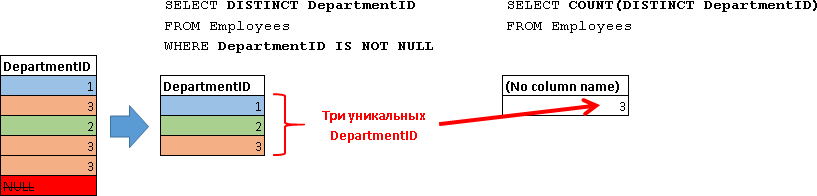
**Слайд №11. COUNT(DISTINCT DepartmentID)** – вернуло нам значение 3, т.е. это число соответствует числу уникальных значений департаментов указанных в столбце DepartmentID без учета NULL значений. Пройдемся по значениям колонки DepartmentID и раскрасим одинаковые значения в один цвет (не стесняйтесь, для обучения все методы хороши):

Отбрасываем NULL, после чего, мы получили 3 уникальных значения (1, 2 и 3). Т.е. значение, получаемое COUNT(DISTINCT DepartmentID), в развернутом виде можно представить следующей выборкой:

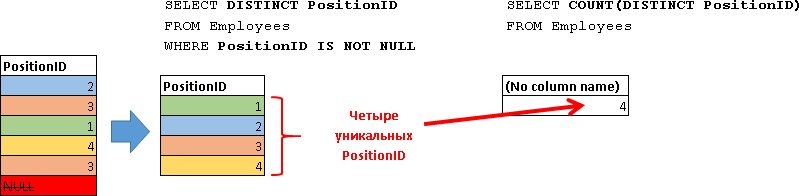
**SELECT** **DISTINCT** DepartmentID *-- 2. берем только уникальные значения*

**FROM** Employees

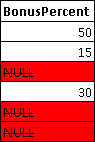
**WHERE** DepartmentID **IS** **NOT** NULL *-- 1. отбрасываем NULL значения*



**Слайд №12. COUNT(DISTINCT PositionID)** – то же самое, что было сказано про COUNT(DISTINCT DepartmentID), только полю PositionID.



**Слайд №13. COUNT(BonusPercent)** – возвращает количество строк, у которых указано значение BonusPercent, т.е. подсчитывается количество записей, у которых BonusPercent IS NOT NULL. Здесь нам будет проще, т.к. не нужно считать уникальные значения, достаточно просто отбросить записи с NULL значениями. Берем значения колонки BonusPercent и вычеркиваем все NULL значения:

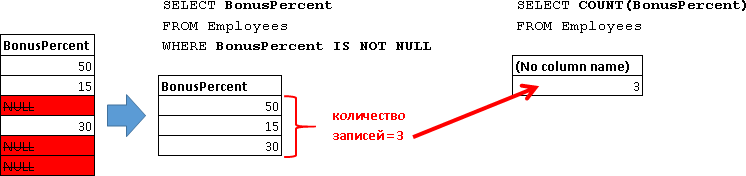


Остается 3 значения. Т.е. в развернутом виде выборку можно представить так:

**SELECT** BonusPercent *-- 2. берем все значения*

**FROM** Employees

**WHERE** BonusPercent **IS** **NOT** NULL *-- 1. отбрасываем NULL значения*



Т.к. мы не использовали слова DISTINCT, то посчитаются и повторяющиеся BonusPercent в случае их наличия, без учета BonusPercent равных NULL.

**Слайд №14.** Для примера давайте сделаем сравнение результата с использованием DISTINCT и без него. Для большей наглядности воспользуемся значениями поля DepartmentID:

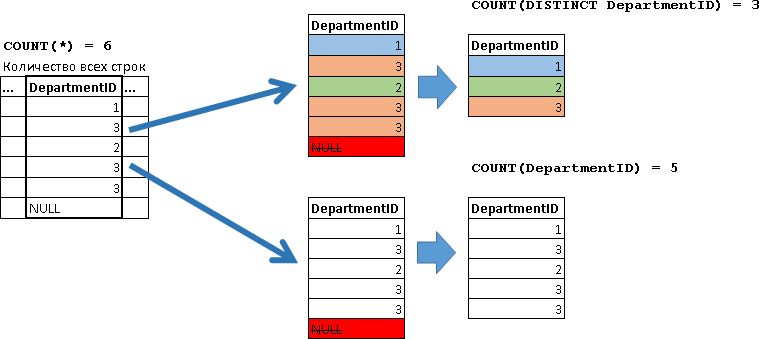
**SELECT**

**COUNT**(\*), *-- 6*

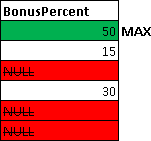
**COUNT**(**DISTINCT** DepartmentID), *-- 3*

**COUNT**(DepartmentID) *-- 5*

**FROM** Employees



**Слайд №15. MAX(BonusPercent)** – возвращает максимальное значение BonusPercent, опять же без учета NULL значений. Берем значения колонки BonusPercent и ищем среди них максимальное значение, на NULL значения не обращаем внимания:



Т.е. мы получаем следующее значение:

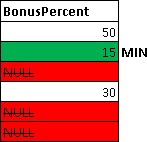
**SELECT** TOP 1 BonusPercent

**FROM** Employees

**WHERE** BonusPercent **IS** **NOT** NULL

**ORDER** **BY** BonusPercent **DESC** *-- сортируем по убыванию*

**MIN(BonusPercent)** – возвращает минимальное значение BonusPercent, опять же без учета NULL значений. Как в случае с MAX, только ищем минимальное значение, игнорируя NULL:



Т.е. мы получаем следующее значение:

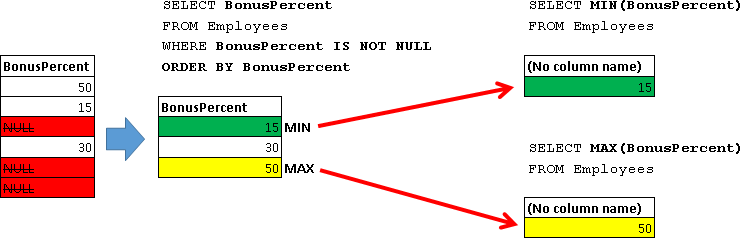
**SELECT** TOP 1 BonusPercent

**FROM** Employees

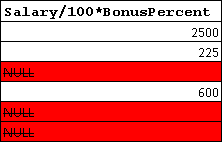
**WHERE** BonusPercent **IS** **NOT** NULL

**ORDER** **BY** BonusPercent *-- сортируем по возрастанию*

**Слайд №16.** Наглядное представление MIN(BonusPercent) и MAX(BonusPercent):



**Слайд №17. SUM(Salary/100\*BonusPercent)** – возвращает сумму всех не NULL значений. Разбираем значения выражения (Salary/100\*BonusPercent):

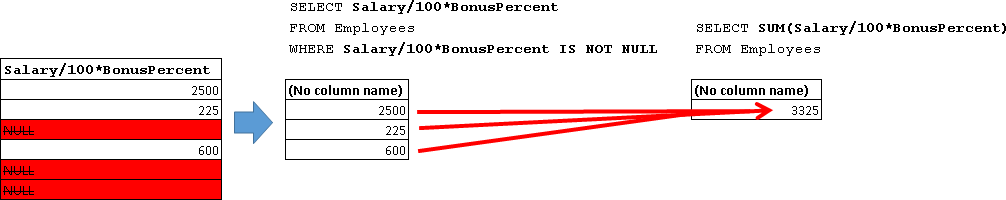


Т.е. происходит суммирование следующих значений:

**SELECT** Salary/100\*BonusPercent

**FROM** Employees

**WHERE** Salary/100\*BonusPercent **IS** **NOT** NULL



**Слайд №18. AVG(Salary/100\*BonusPercent)** – возвращает среднее значение. NULL-выражения не учитываются, т.е. это соответствует второму выражению:

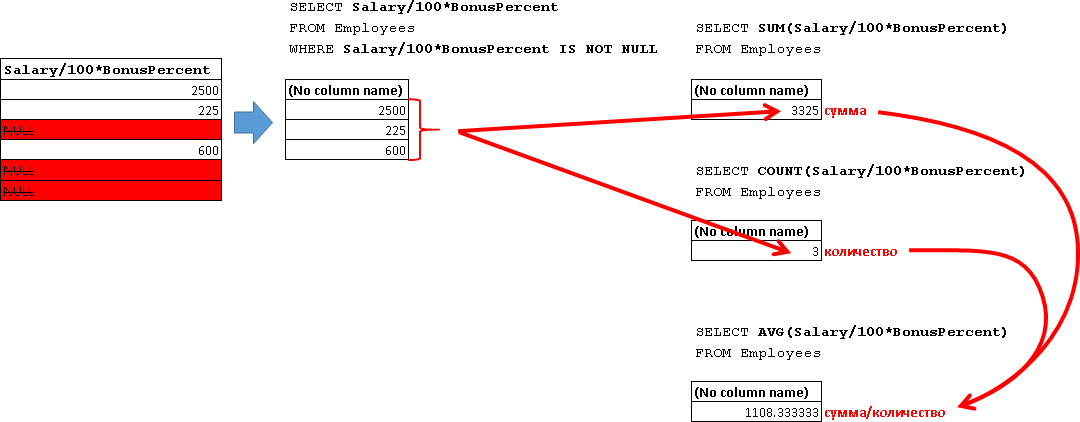
**SELECT**

**AVG**(Salary/100\*BonusPercent), *-- 1108.33333333333*

**SUM**(Salary/100\*BonusPercent)/**COUNT**(Salary/100\*BonusPercent), *-- 1108.33333333333*

**SUM**(Salary/100\*BonusPercent)/**COUNT**(\*) *-- 554.166666666667*

**FROM** Employees



Т.е. опять же NULL-значения не учитываются при подсчете количества.

Если же вам необходимо вычислить среднее по всем сотрудникам, как в третьем выражении, которое дает 554.166666666667, то используйте предварительное преобразование NULL значений в ноль:

**SELECT**

**AVG**(**ISNULL**(Salary/100\*BonusPercent,0)), *-- 554.166666666667*

**SUM**(Salary/100\*BonusPercent)/**COUNT**(\*) *-- 554.166666666667*

**FROM** Employees

**AVG(Salary)** – собственно, здесь все то же самое что и в предыдущем случае, т.е. если у сотрудника Salary равен NULL, то он не учтется. Чтобы учесть всех сотрудников, соответственно делаете предварительное преобразование NULL значений AVG(ISNULL(Salary,0))

**Подведем некоторые итоги:**

COUNT(\*) – служит для подсчета общего количества строк, которые получены оператором «SELECT … WHERE …»

* во всех остальных вышеперечисленных агрегатных функциях при расчете итога, NULL-значения не учитываются;
* если нам нужно учесть все строки, это больше актуально для функции AVG, то предварительно необходимо осуществить обработку NULL значений, например, как было показано выше «AVG(ISNULL(Salary,0))»

Соответственно при задании с агрегатными функциями дополнительного условия в блоке WHERE, будут подсчитаны только итоги, по строкам, удовлетворяющим условию. Т.е. расчет агрегатных значений происходит для итогового набора, который получен при помощи конструкции SELECT. Например, сделаем все тоже самое, но только в разрезе ИТ-отдела:

**Слайд №19.**

**SELECT**

**COUNT**(\*) [Общее кол-во сотрудников],

**COUNT**(**DISTINCT** DepartmentID) [Число уникальных отделов],

**COUNT**(**DISTINCT** PositionID) [Число уникальных должностей],

**COUNT**(BonusPercent) [Кол-во сотрудников у которых указан % бонуса],

**MAX**(BonusPercent) [Максимальный процент бонуса],

**MIN**(BonusPercent) [Минимальный процент бонуса],

**SUM**(Salary/100\*BonusPercent) [Сумма всех бонусов],

**AVG**(Salary/100\*BonusPercent) [Средний размер бонуса],

**AVG**(Salary) [Средний размер ЗП]

**FROM** Employees

**WHERE** DepartmentID=3 *-- учесть только ИТ-отдел*

**Слайд №20.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Общее кол-во сотрудников** | **Число уникальных отделов** | **Число уникальных должностей** | **Кол-во сотрудников у которых указан % бонуса** | **Максимальный процент бонуса** | **Минимальный процент бонуса** | **Сумма всех бонусов** | **Средний размер бонуса** | **Средний размер ЗП** |
| 3 | 1 | 2 | 2 | 30 | 15 | 825 | 412.5 | 1666.66666666667 |

Предлагаю вам, для большего понимания работы агрегатных функций, самостоятельно проанализировать каждое полученное значение. Расчеты здесь ведем, соответственно, по детальным данным полученным запросом:

**Слайд №21.**

**SELECT**

DepartmentID,

PositionID,

BonusPercent,

Salary/100\*BonusPercent [Salary/100\*BonusPercent],

Salary

**FROM** Employees

**WHERE** DepartmentID=3 *-- учесть только ИТ-отдел*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DepartmentID** | **PositionID** | **BonusPercent** | **Salary/100\*BonusPercent** | **Salary** |
| 3 | 3 | 15 | 225 | 1500 |
| 3 | 4 | 30 | 600 | 2000 |
| 3 | 3 | NULL | NULL | 1500 |

Идем, дальше. В случае, если агрегатная функция возвращает NULL (например, у всех сотрудников не указано значение Salary), или в выборку не попало ни одной записи, а в отчете, для такого случая нам нужно показать 0, то функцией ISNULL можно обернуть агрегатное выражение:

**SELECT**

**SUM**(Salary),

**AVG**(Salary),

*-- обрабатываем итог при помощи ISNULL*

**ISNULL**(**SUM**(Salary),0),

**ISNULL**(**AVG**(Salary),0)

**FROM** Employees

**WHERE** DepartmentID=10 *-- здесь специально указан несуществующий отдел, чтобы запрос не вернул записей*

|  |  |  |  |
| --- | --- | --- | --- |
| **(No column name)** | **(No column name)** | **(No column name)** | **(No column name)** |
| NULL | NULL | 0 | 0 |

Важно понимать назначение каждой агрегатной функции и то каким образом они производят расчет, т.к. в SQL это главный инструмент, который служит для расчета итоговых значений. В данном случае мы рассмотрели, как каждая агрегатная функция ведет себя самостоятельно, т.е. она применялась к значениям всего набора записей полученным командой SELECT. Дальше мы рассмотрим, как эти же функции применяются для вычисления итогов по группам, при помощи конструкции GROUP BY.

## GROUP BY – группировка данных

До этого мы уже вычисляли итоги для конкретного отдела, примерно следующим образом:

**SELECT**

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=3 *-- данные только по ИТ отделу*

А теперь представьте, что нас попросили получить такие же цифры в разрезе каждого отдела. Конечно мы можем засучить рукава и выполнить этот же запрос для каждого отдела. Итак, сказано-сделано, пишем 4 запроса: (слайды №23,24)

**SELECT**

'Администрация' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=1 *-- данные по Администрации*

**SELECT**

'Бухгалтерия' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=2 *-- данные по Бухгалтерии*

**SELECT**

'ИТ' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=3 *-- данные по ИТ отделу*

**SELECT**

'Прочие' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

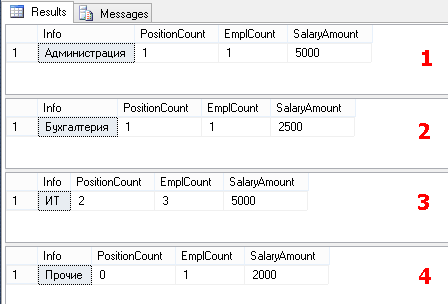
**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID **IS** NULL *-- и еще не забываем данные по внештатникам*

**Слайд № 25.** В результате мы получим 4 набора данных:



Обратите внимание, что мы можем использовать поля, заданные в виде констант – 'Администрация', 'Бухгалтерия', …

В общем, все цифры, о которых нас просили, мы добыли, объединяем все в Excel и отдаем директору. Отчет директору понравился, и он говорит: «а добавьте еще колонку с информацией по среднему окладу». И как всегда это нужно сделать очень срочно. Мда, что делать?! Вдобавок представим еще, что отделов у нас не 3, а 15.

**Слайд № 26.** Вот как раз то примерно для таких случаев служит конструкция **GROUP BY**:

**SELECT**

DepartmentID,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount,

**AVG**(Salary) SalaryAvg *-- плюс выполняем пожелание директора*

**FROM** Employees

**GROUP** **BY** DepartmentID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DepartmentID** | **PositionCount** | **EmplCount** | **SalaryAmount** | **SalaryAvg** |
| NULL | 0 | 1 | 2000 | 2000 |
| 1 | 1 | 1 | 5000 | 5000 |
| 2 | 1 | 1 | 2500 | 2500 |
| 3 | 2 | 3 | 5000 | 1666.66666666667 |

Мы получили все те же самые данные, но теперь используя только один запрос!

Пока не обращайте внимание, на то, что департаменты у нас вывелись в виде цифр, дальше мы научимся выводить все красиво.

В предложении **GROUP BY** можно указывать несколько полей «GROUP BY поле1, поле2, …, полеN», в этом случае группировка произойдет по группам, которые образовывают значения данных полей «поле1, поле2, …, полеN».

**Слайд № 27.** Для примера, сделаем группировку данных в разрезе Отделов и Должностей:

**SELECT**

DepartmentID,PositionID,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**GROUP** **BY** DepartmentID,PositionID

|  |  |  |  |
| --- | --- | --- | --- |
| **DepartmentID** | **PositionID** | **EmplCount** | **SalaryAmount** |
| NULL | NULL | 1 | 2000 |
| 2 | 1 | 1 | 2500 |
| 1 | 2 | 1 | 5000 |
| 3 | 3 | 2 | 3000 |
| 3 | 4 | 1 | 2000 |

**Давайте, теперь на этом примере, попробуем разобраться как работает GROUP BY**

**Слайд № 28.** Для полей, перечисленных после GROUP BY из таблицы Employees определяются все уникальные комбинации по значениям DepartmentID и PositionID, т.е. происходит примерно следующее:

**SELECT** **DISTINCT** DepartmentID,PositionID

**FROM** Employees

|  |  |
| --- | --- |
| **DepartmentID** | **PositionID** |
| NULL | NULL |
| 1 | 2 |
| 2 | 1 |
| 3 | 3 |
| 3 | 4 |

**Слайд № 29.** После чего делается пробежка по каждой комбинации и делаются вычисления агрегатных функций:

**SELECT**

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID **IS** NULL **AND** PositionID **IS** NULL

**SELECT**

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=1 **AND** PositionID=2

*-- ...*

**SELECT**

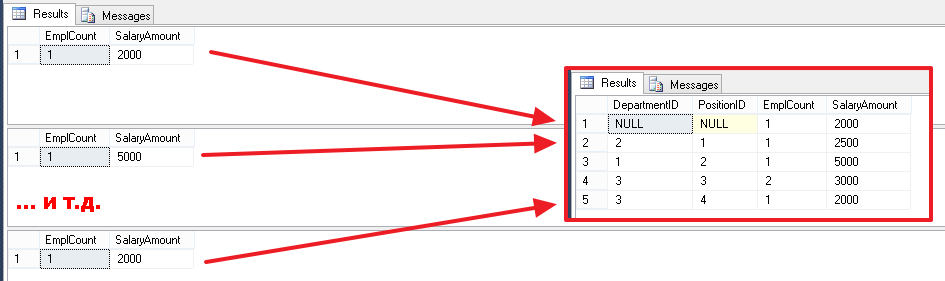
**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=3 **AND** PositionID=4

**Слайд № 30.** А потом все эти результаты объединяются вместе и отдаются нам в виде одного набора:



Из основного, стоит отметить, что в случае группировки (GROUP BY), в перечне колонок в блоке SELECT:

* Мы можем использовать только колонки, перечисленные в блоке GROUP BY
* Можно использовать выражения с полями из блока GROUP BY
* Можно использовать константы, т.к. они не влияют на результат группировки
* Все остальные поля (не перечисленные в блоке GROUP BY) можно использовать только с агрегатными функциями (COUNT, SUM, MIN, MAX, …)
* Не обязательно перечислять все колонки из блока GROUP BY в списке колонок SELECT

**Слайд № 31.** И демонстрация всего сказанного:

**SELECT**

'Строка константа' Const1, *-- константа в виде строки*

1 Const2, *-- константа в виде числа*

*-- выражение с использованием полей участвующих в группировке*

**CONCAT**('Отдел № ',DepartmentID) ConstAndGroupField,

**CONCAT**('Отдел № ',DepartmentID,', Должность № ',PositionID) ConstAndGroupFields,

DepartmentID, *-- поле из списка полей участвующих в группировке*

*-- PositionID, -- поле участвующее в группировке, не обязательно дублировать здесь*

**COUNT**(\*) EmplCount, *-- кол-во строк в каждой группе*

*-- остальные поля можно использовать только с агрегатными функциями: COUNT, SUM, MIN, MAX, …*

**SUM**(Salary) SalaryAmount,

**MIN**(**ID**) MinID

**FROM** Employees

**GROUP** **BY** DepartmentID,PositionID *-- группировка по полям DepartmentID,PositionID*

**Слайд № 32.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Const1** | **Const2** | **ConstAndGroupField** | **ConstAndGroupFields** | **DepartmentID** | **EmplCount** | **SalaryAmount** | **MinID** |
| Строка константа | 1 | Отдел № | Отдел №, Должность № | NULL | 1 | 2000 | 1005 |
| Строка константа | 1 | Отдел № 2 | Отдел № 2, Должность № 1 | 2 | 1 | 2500 | 1002 |
| Строка константа | 1 | Отдел № 1 | Отдел № 1, Должность № 2 | 1 | 1 | 5000 | 1000 |
| Строка константа | 1 | Отдел № 3 | Отдел № 3, Должность № 3 | 3 | 2 | 3000 | 1001 |
| Строка константа | 1 | Отдел № 3 | Отдел № 3, Должность № 4 | 3 | 1 | 2000 | 1003 |

Так же стоит отметить, что группировку можно делать не только по полям, но также и по выражениям.

**Слайд № 33.** Для примера сгруппируем данные по сотрудникам, по годам рождения:

**SELECT**

**CONCAT**('Год рождения - ',**YEAR**(Birthday)) YearOfBirthday,

**COUNT**(\*) EmplCount

**FROM** Employees

**GROUP** **BY** **YEAR**(Birthday)

Ну и конечно же вы можете объединять в блоке GROUP BY выражения с полями:

**SELECT**

DepartmentID,

**CONCAT**('Год рождения - ',**YEAR**(Birthday)) YearOfBirthday,

**COUNT**(\*) EmplCount

**FROM** Employees

**GROUP** **BY** **YEAR**(Birthday),DepartmentID *-- порядок может не совпадать с порядком их использования в блоке SELECT*

**ORDER** **BY** DepartmentID,YearOfBirthday *-- напоследок мы можем применить к результату сортировку*

**GROUP BY** в скупе с агрегатными функциями, одно из основных средств, служащих для получения сводных данных из БД, ведь обычно данные в таком виде и используются, т.к. обычно от нас требуют предоставления сводных отчетов, а не детальных данных (простыней). И конечно же все это крутится вокруг знания базовой конструкции, т.к. прежде чем что-то подытожить (агрегировать), вам нужно первым делом это правильно выбрать, используя «SELECT … WHERE …».

## Слайд № 34. HAVING – наложение условия выборки к сгруппированным данным

**HAVING** – чем-то подобен WHERE, только если WHERE-условие применяется к детальным данным, то HAVING-условие применяется к уже сгруппированным данным. По этой причине в условиях блока HAVING мы можем использовать либо выражения с полями, входящими в группировку, либо выражения, заключенные в агрегатные функции.

Рассмотрим пример:

**SELECT**

DepartmentID,

**SUM**(Salary) SalaryAmount

**FROM** Employees

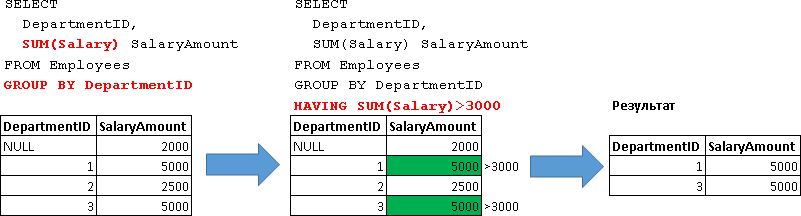
**GROUP** **BY** DepartmentID

**HAVING** **SUM**(Salary)>3000

|  |  |
| --- | --- |
| **DepartmentID** | **SalaryAmount** |
| 1 | 5000 |
| 3 | 5000 |

Т.е. данный запрос вернул нам сгруппированные данные только по тем отделам, у которых сумма ЗП всех сотрудников превышает 3000, т.е. «SUM(Salary)>3000».

**Слайд № 35.** Т.е. здесь в первую очередь происходит группировка и вычисляются данные по всем отделам:



**SELECT**

DepartmentID,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**GROUP** **BY** DepartmentID *-- 1. получаем сгруппированные данные по всем отделам*

А уже к этим данным применяется условие указанно в блоке HAVING:

**SELECT**

DepartmentID,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**GROUP** **BY** DepartmentID *-- 1. получаем сгруппированные данные по всем отделам*

**HAVING** **SUM**(Salary)>3000 *-- 2. условие для фильтрации сгруппированных данных*

**Слайд № 36.** В HAVING-условии так же можно строить сложные условия, используя операторы AND, OR и NOT:

**SELECT**

DepartmentID,

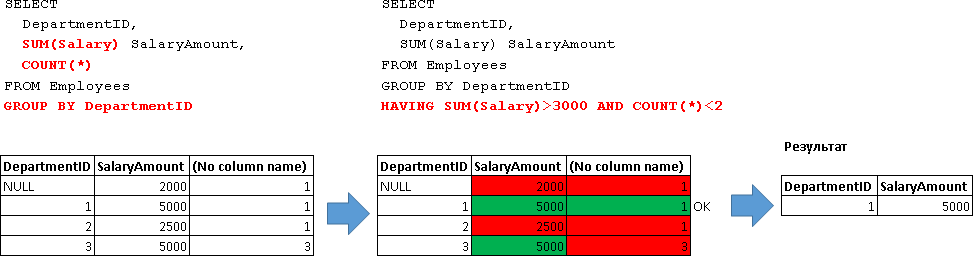
**SUM**(Salary) SalaryAmount

**FROM** Employees

**GROUP** **BY** DepartmentID

**HAVING** **SUM**(Salary)>3000 **AND** **COUNT**(\*)<2 *-- и число людей меньше 2-х*

**Слайд № 37.**



!!! Как можно здесь заметить, **агрегатная функция (см. «COUNT(\*)») может быть указана только в блоке HAVING.**

Соответственно мы можем отобразить только номер отдела, подпадающего под HAVING-условие:

**SELECT**

DepartmentID

**FROM** Employees

**GROUP** **BY** DepartmentID

**HAVING** **SUM**(Salary)>3000 **AND** **COUNT**(\*)<2 *-- и число людей меньше 2-х*

**Слайд № 38.** Пример использования HAVING-условия по полю включенного в GROUP BY:

**SELECT**

DepartmentID,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**GROUP** **BY** DepartmentID *-- 1. сделать группировку*

**HAVING** DepartmentID=3 *-- 2. наложить фильтр на результат группировки*

Это только пример, т.к. в данном случае проверку логичнее было бы сделать через WHERE-условие:

**SELECT**

DepartmentID,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=3 *-- 1. провести фильтрацию детальных данных*

**GROUP** **BY** DepartmentID *-- 2. сделать группировку только по отобранным записям*

Т.е. сначала отфильтровать сотрудников по отделу 3, и только потом сделать расчет.

**Примечание.** На самом деле, несмотря на то, что эти два запроса выглядят по-разному оптимизатор СУБД может выполнить их одинаково.

Вы не сможете использовать агрегатную функцию в предложении WHERE ( если вы не используете подзапрос, описанный позже ), потому что предикаты оцениваются в терминах одиночной строки, а агрегатные функции оцениваются в терминах групп строк. Это означает что вы не сможете сделать что-нибудь подобно следующему:

SELECT snum, odate, MAX (amt)

FROM Oreders

WHERE MAX ((amt)) > 3000.00

GROUP BY snum, odate;

Это будет отклонением от строгой интерпретации ANSI. Чтобы увидеть максимальную стоимость приобретений свыше $3000.00, вы можете использовать предложение HAVING. Предложение HAVING определяет критерии? используемые чтобы удалять определенные группы из вывода, точно также как предложение WHERE делает это для индивидуальных строк. Правильной командой будет следующая:

SELECT snum, odate, MAX ((amt))

FROM Orders

GROUP BY snum, odate

HAVING MAX ((amt)) > 3000.00;

Аргументы в предложении HAVING следуют тем же самым правилам что и в предложении SELECT, состоящей из команд использующих GROUP BY. Они должны иметь одно значение на группу вывода. Следующая команда будет запрещена:

SELECT snum, MAX (amt)

FROM Orders

GROUP BY snum

HAVING odate = 10/03/1988;

Поле оdate не может быть вызвано предложением HAVING, потому что оно может иметь ( и действительно имеет ) больше чем одно значение на группу вывода. Чтобы избегать такой ситуации, предложение HAVING должно ссылаться только на агрегаты и поля выбранные GROUP BY. Имеется правильный способ сделать вышеупомянутый запрос( вывод показывается в Рисунке 6.8 ):

SELECT snum, MAX (amt)

FROM Orders

WHEREodate = 10/03/1990

GROUP BY snum;

Поскольку пол odate нет, не может быть и выбранных полей, значение этих данных меньше чем в некоторых других примерах. Вывод должен вероятно включать что-нибудь такое что говорит - " это - самые большие порядки на 3 Октября." В [Главе 7](http://www.sql.ru/docs/sql/u_sql/ch7.shtml), мы покажем как вставлять текст в ваш вывод. Как и говорилось ранее, HAVING может использовать только аргументы которые имеют одно значение на группу вывода. Практически, ссылки на агрегатные функции - наиболее общие, но и пол выбранные с помощью GROUP BY также допустимы. Например, мы хотим увидеть наибольшие порядки для Serres и Rifkin:

SELECT snum, MAX (amt)

FROM Orders

GROUP BY snum

HAVING snum B (1002,1007);

Вывод для этого запроса показывается в Рисунке 6.9.

=============== SQL Execution Log ==============

| |

| SELECT snum, MAX (amt) |

| FROM Orders |

| GROUP BY snum |

| HAVING snum IN ( 1002, 1007 );

**HAVING** - применяется для фильтрации функций и столбцов сгруппированных при помощи GROUP BY указанных в SELECT.

Другими словами применяется для агрегатных функций(COUNT(), MAX() ...) и столбцов указанных в выражении SELECT и обработанных GROUP BY. Если нужно использовать что-то, что не указанно в SELECT, то лучше использовать WHERE.

Порядок выполнения такой:

1. В SELECT указываем нужные столбцы или агрегатные функции(то с чем будем работать в GROUP BY и HAVING)

2. В GROUP BY пишем по какому столбцу или функции их группируем

3. В HAVING пишем условие фильтровки результата GROUP BY

WHERE - это ограничивающее выражение. Оно выполняется до того, как будет получен результат операции.

HAVING - фильтрующее выражение. Оно применяется к результату операции и выполняется уже после того как этот результат будет получен, в отличии от where.

Выражения WHERE используются вместе с операциями SELECT, UPDATE, DELETE, в то время как HAVING только с SELECT и предложением GROUP BY.

Например, WHERE нельзя использовать таким образом:

SELECT name, SUM(salary) FROM Employees WHERE SUM(salary) > 1000 GROUP BY name

В данном случае больше подходит HAVING:

SELECT name, SUM(salary) FROM Employees GROUP BY name HAVING SUM(salary) > 1000

То есть, использовать WHERE в запросах с агрегатными функциями нельзя, для этого и был введен HAVING.

## Подведем итоги

Рассмотрим конкретное месторасположение каждой изученной нами конструкции и укажем порядок их выполнения:

|  |  |  |
| --- | --- | --- |
| **Конструкция/Блок** | **Порядок выполнения** | **Выполняемая функция** |
| SELECT возвращаемые выражения | 4 | Возврат данных полученных запросом |
| FROM источник | 0 | В нашем случае это пока все строки таблицы |
| WHERE условие выборки из источника | 1 | Отбираются только строки, проходящие по условию |
| GROUP BY выражения группировки | 2 | Создание групп по указанному выражению группировки. Расчет агрегированных значений по этим группам, используемых в SELECT либо HAVING блоках |
| HAVING фильтр по сгруппированным данным | 3 | Фильтрация, накладываемая на сгруппированные данные |
| ORDER BY выражение сортировки результата | 5 | Сортировка данных по указанному выражению |

Конечно же, вы так же можете применить к сгруппированным данным предложения DISTINCT и TOP, изученные во второй части.

Эти предложения в данном случае применятся к окончательному результату:  
**SELECT**

TOP 1 *-- 6. применится в последнюю очередь*

**SUM**(Salary) SalaryAmount

**FROM** Employees

**GROUP** **BY** DepartmentID

**HAVING** **SUM**(Salary)>3000

**ORDER** **BY** DepartmentID *-- 5. сортировка результата*

|  |
| --- |
| **SalaryAmount** |
| 5000 |

**SELECT**

**DISTINCT** *-- показать только уникальные значения SalaryAmount*

**SUM**(Salary) SalaryAmount

**FROM** Employees

**GROUP** **BY** DepartmentID

|  |
| --- |
| **SalaryAmount** |
| 2000 |
| 2500 |
| 5000 |