Простой оператор SELECT

SELECT \* FROM PC PC – таблица в БД

SELECT DISTINCT speed, ram DISTINCT – уникальные записи

FROM PC

ORDER BY ram DESC, speed DESC; ORDER BY – сортировать

Сортировку можно проводить по **возрастанию** (параметр **ASC** принимается по умолчанию) или по **убыванию** (параметр **DESC**). Сортировать так же можно по **нескольким** полям.

Сортировку так же можно проводить по **номеру** столбца в базе.

SELECT DISTINCT speed, ram

FROM PC

WHERE price < 500 WHERE – предикат, из-за которого в выборку ORDER BY 2 DESC; попадут удовлетворяющие условию записи.

В последнем запросе был применен предикат сравнения с использованием операции сравнения «<» (меньше чем). Кроме этой операции сравнения могут использоваться: «=» (равно), «>» (больше), «>=» (больше или равно), «<=» (меньше или равно) и «<>» (не равно). Выражения в предикатах сравнения могут содержать константы и любые поля из таблиц, указанных в предложении **FROM**. Символьные строки и константы типа дата/время записываются в апострофах.

Примеры: **type = ‘laptop’ color <> ’y’ ram – 128 > 0 Price <= speed\*2**

Сортировку можно выполнять даже по столбцам, отсутствующим в списке **SELECT**. Естественно, эти столбцы должны присутствовать на выходе предложения FROM.

SELECT model FROM PC

ORDER BY price DESC;

Обратите внимание, что сама цена (price) не выводится запросом. Исключением является неоднозначная ситуация, возникающая при исключении дубликатов. По той же причине не будет работать запрос с [группировкой](http://www.sql-tutorial.ru/ru/book_group_by_clause.html).

SELECT DISTINCT model FROM PC

ORDER BY price DESC;

Результат: ОШИБКА

Однако если неоднозначность устранить (выполнить сортировку по какому-либо агрегатному значению для группы), то можно "подправить" запрос.

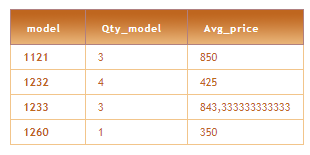
SELECT model FROM PC

GROUP BY model

ORDER BY MAX(price) DESC;

Предложение GROUP BY

Предложение **GROUP BY** используется для определения групп выходных строк, к которым могут применяться агрегатные функции (**COUNT**, **MIN**, **MAX, AVG**и **SUM**). Если это предложение отсутствует, и используются агрегатные функции, то все столбцы с именами, упомянутыми в **SELECT**, должны быть включены в агрегатные функции, и эти функции будут применяться ко всему набору строк, которые удовлетворяют предикату запроса. В противном случае все столбцы списка **SELECT**, не вошедшие в агрегатные функции, должны быть указаны в предложении **GROUP BY**. В результате чего все выходные строки запроса разбиваются на группы, характеризуемые одинаковыми комбинациями значений в этих столбцах. После чего к каждой группе будут применены агрегатные функции. Следует иметь в виду, что для **GROUP BY** все значения **NULL**трактуются как равные, то есть при группировке по полю, содержащему **NULL**-значения, все такие строки попадут в одну группу.

Если при наличии предложения **GROUP BY**, в предложении **SELECT**отсутствуют агрегатные функции, то запрос просто вернет по одной строке из каждой группы. Эту возможность, наряду с ключевым словом **DISTINCT**, можно использовать для исключения дубликатов строк в результирующем наборе.

SELECT model, COUNT(model) AS Qty\_model,

    AVG(price) AS Avg\_price

FROM PC

GROUP BY model;

В этом запросе для каждой модели ПК определяется их количество и средняя стоимость. Все строки с одинаковыми значениями model (номер модели) образуют группу, и на выходе **SELECT**вычисляются количество значений и средняя цена для каждой группы.

Если бы в **SELECT**присутствовал столбец с датой, то можно было бы вычислять эти показатели для каждой конкретной даты. Для этого нужно добавить дату в качестве группирующего столбца, и тогда агрегатные функции вычислялись бы для каждой комбинации значений {модель, дата}.

Существует несколько определенных правил выполнения агрегатных функций.

* Если в результате выполнения запроса не получено ни одной строки (или ни одной строки для данной группы), то исходные данные для вычисления любой из агрегатных функций отсутствуют. В этом случае результатом выполнения функций **COUNT**будет нуль, а результатом всех других функций — **NULL**.
* Аргумент агрегатной функции не может сам содержать агрегатные функции (функция от функции).
* Результат выполнения функции **COUNT**есть целое число (INTEGER). Другие агрегатные функции наследуют типы данных обрабатываемых значений.
* Если при выполнении функции **SUM**будет получен результат, превышающий максимально возможное значение для используемого типа данных, возникает ошибка.

Итак, агрегатные функции, включенные в предложение **SELECT**запроса, не содержащего предложения **GROUP BY**, исполняются над всеми результирующими строками этого запроса. Если же запрос содержит предложение **GROUP BY**, каждый набор строк, который имеет одинаковые значения столбца или группы столбцов, заданных в предложении **GROUP BY**, составляют группу, и агрегатные функции выполняются для каждой группы отдельно.

Предикаты I

Предикаты представляют собой выражения, принимающие истинностное значение. Они могут представлять собой как одно выражение, так и любую комбинацию из неограниченного количества выражений, построенную с помощью булевых операторов **AND**, **OR** или **NOT**. Кроме того, в этих комбинациях может использоваться SQL-оператор **IS**, а также круглые скобки для конкретизации [порядка выполнения операций](http://www.sql-tutorial.ru/ru/book_predicates_1/page2.html).

Предикат в языке [SQL](http://www.sql-tutorial.ru/ru/termins/sql.html) может принимать одно из трех значений TRUE (истина или 1), FALSE (ложь или 0) или **UNKNOWN** (неизвестно или 1/2). Исключение составляют следующие предикаты: **IS NULL** (отсутствие значения),**EXISTS** (существование), **UNIQUE** (уникальность) и **MATCH** (совпадение), которые не могут принимать значение **UNKNOWN**.

Логические операторы при отсутствии скобок, как и арифметические операторы, выполняются в соответствии с их **старшинством (NOT, AND, OR)**. Поменять порядок выполнения логических операторов можно при помощи **скобок**.

Явные операции соединения

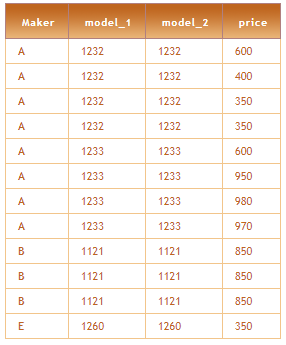
В предложении **FROM**может быть указана **явная операция соединения** двух и более таблиц. Среди ряда операций соединения, описанных в стандарте языка SQL, многими серверами баз данных поддерживается только операция **соединения по предикату**.

FROM <таблица 1>

[INNER]

{{LEFT | RIGHT | FULL } [OUTER]} JOIN <таблица 2>

[ON <предикат>]

Соединение может быть либо внутренним (**INNER**), либо одним из внешних (**OUTER**). Служебные слова **INNER**и **OUTER**можно опускать, поскольку внешнее соединение однозначно определяется его типом — **LEFT**(левое), **RIGHT**(правое) или **FULL**(полное), а просто **JOIN**будет означать внутреннее соединение.

Предикат определяет условие соединения строк из разных таблиц. При этом **INNER JOIN** означает, что в результирующий набор попадут только те соединения строк двух таблиц, для которых значение предиката равно **TRUE**. Как правило, предикат определяет эквисоединение по внешнему и первичному ключам соединяемых таблиц, хотя это не обязательно.

SELECT maker, Product.model AS model\_1,

            PC.model AS model\_2, price

FROM Product INNER JOIN

PC ON PC.model = Product.model

ORDER BY maker, model\_2;

В данном примере в результирующем наборе будут соединяться только те строки из таблиц РС и Product, у которых совпадают номера моделей.

Внешнее соединение**LEFT JOIN** означает, что помимо строк, для которых выполняется условие предиката, в результирующий набор попадут все остальные строки из первой таблицы (левой). При этом отсутствующие значения столбцов из правой таблицы будут заменены **NULL**-значениями.

**Привести все модели ПК, их производителей и цену:**

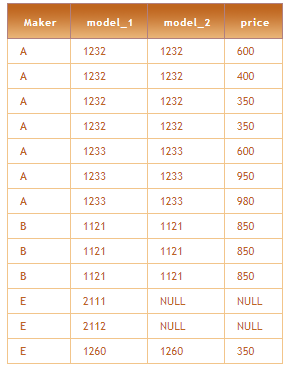
SELECT maker, Product.model AS model\_1, pc.model AS model\_2, price

FROM Product LEFT JOIN

PC ON PC.model = Product.model

WHERE type = 'pc'

ORDER BY maker, PC.model;

Обратите внимание на то, что по сравнению с предыдущим примером пришлось использовать предложение **WHERE**для отбора только производителей ПК. В противном случае в результирующий набор попали бы также и модели портативных компьютеров, и принтеров. В рассмотренном ранее примере это условие было бы излишним, так как соединялись только те строки, у которых совпадали номера моделей, и одной из таблиц была таблица PC, содержащая только модели ПК.

Поскольку моделей 2111 и 2112 из таблицы Product нет в таблице PС, в столбцах из таблицы PС содержится **NULL**.

Соединение **RIGHT JOIN** обратно соединению**LEFT JOIN**, то есть в результирующий набор попадут все строки из второй таблицы, которые будут соединяться только с теми строками из первой таблицы, для которых выполняется условие соединения. В нашем случае левое соединение

Product LEFT JOIN PC ON PC.model = Product.model

будет эквивалентно правому соединению

PC RIGHT JOIN Product ON PC.model = Product.model

Запрос:

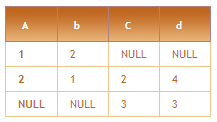
SELECT maker, Product.model AS model\_1, PC.model AS model\_2, price

FROM Product RIGHT JOIN

PC ON PC.model = Product.model

ORDER BY maker, PC.model;

даст те же результаты, что и внутреннее соединение, поскольку в правой таблице (PC) нет таких моделей, которые отсутствовали бы в левой таблице (Product), что вполне естественно для типа связи «один ко многим», которая имеется между таблицами PC и Product.

Наконец, при полном соединении (**FULL JOIN**) в результирующую таблицу попадут не только те строки, которые имеют одинаковые значения в сопоставляемых столбцах, но и все остальные строки исходных таблиц, не имеющие соответствующих значений в другой таблице. В этих строках все столбцы той таблицы, в которой не было найдено соответствия, заполняются **NULL**-значениями. То есть полное соединение представляет собой комбинацию левого и правого внешних соединений.

SELECT A.\*, B.\*

FROM A FULL JOIN

B ON A.a = B.c;

Заметим, что это соединение симметрично, то есть A **FULL JOIN** B эквивалентно B **FULL JOIN** A. Обратите также внимание на обозначение A.\*, что означает вывести все столбцы таблицы А.

Операторы UNION и IN

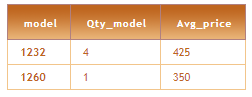
UNION служит для объединения результатов запросов в один.

IN служит для вывода только тех значений, которые входят в указанную область.

Оператор HAVING

Если предложение **WHERE** определяет предикат для фильтрации строк, то предложение **HAVING** применяется после группировки для определения аналогичного предиката, фильтрующего группы по значениям агрегатных функций. Это предложение необходимо для проверки значений, которые получены с помощью агрегатной функции не из отдельных строк источника записей, определенного в предложении **FROM**, а из групп таких строк. Поэтому такая проверка не может содержаться в предложении **WHERE**.

**Получить количество ПК и среднюю цену для каждой модели, средняя цена которой менее $800:**

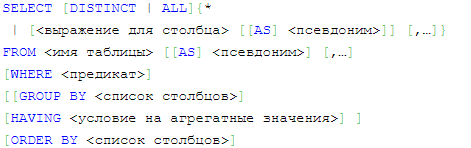
SELECT model, COUNT(model) AS Qty\_model,

   AVG(price) AS Avg\_price

FROM PC

GROUP BY model

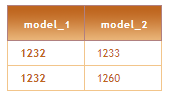
HAVING AVG(price) < 800;

 Заметим, что в предложении **HAVING** нельзя использовать псевдоним (Avg\_price), используемый для именования значений агрегатной функции в предложении **SELECT**. Дело в том, что предложение **SELECT**, формирующее выходной набор запроса, выполняется предпоследним перед предложением **ORDER BY**. Ниже приведен порядок обработки предложений в операторе **SELECT**:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Этот порядок не соответствует синтаксическому порядку общего представления оператора **SELECT**, который ближе к естественному языку.

Использование в запросе нескольких источников записей

Иногда в предложении **FROM** требуется указать одну и ту же таблицу несколько раз. В этом случае обязательным является переименование.

SELECT DISTINCT A.model AS model\_1, B.model AS model\_2

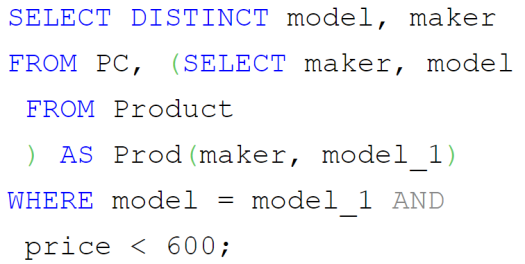
FROM PC AS A, PC B

WHERE A.price = B.price AND

A.model < B.model;

Здесь условие a.model < b.model используется для того, чтобы не выводились одинаковые пары, отличающиеся только перестановкой, например: {1232, 1233} и {1233, 1232}. **DISTINCT**применяется для того, чтобы исключить одинаковые строки, поскольку в таблице PC имеются модели с одинаковыми номерами по одной и той же цене.

Переименование также является обязательным, если в предложении **FROM**используется подзапрос, так как, в противном случае, у нас нет возможности уточнения имени столбца из подзапроса. Так, первый пример можно переписать следующим образом:



SELECT DISTINCT PC.model, maker

FROM PC, (SELECT maker, model

FROM Product

) AS Prod

WHERE PC.model = Prod.model AND

price < 600;

За псевдонимом производного табличного выражения может в скобках стоять список имен столбцов, которые будут использоваться вместо имен табличного выражения. Порядок имен должен, естественно, соответствовать списку столбцов табличного выражения (в нашем случае - списку в предложении SELECT). Это способ позволяет избежать неоднозначности имен и, как следствие, необходимости их уточнения.

Использование операторов SOME (ANY) и ALL с предикатами сравнения

<выражение> <оператор сравнения> SOME | ANY (<подзапрос>)

**SOME**и **ANY**являются синонимами, то есть может использоваться любое из них. Результатом подзапроса является один столбец величин. Если хотя бы для одного значения V, получаемого из подзапроса, результат операции *"<значение выражения> <*[*оператор сравнения*](http://www.sql-tutorial.ru/ru/book_comparison_predicates.html)*> V*" равняется **TRUE**, то предикат **ANY**также равняется **TRUE**.

<выражение> <оператор сравнения> ALL (<подзапрос>)

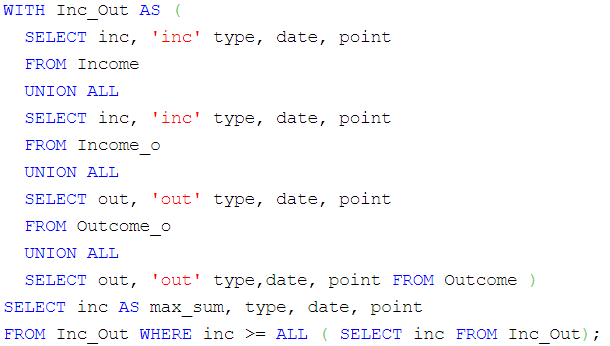
Исполняется так же, как и **ANY**, однако значение предиката **ALL** будет истинным, если для всех значений V, получаемых из подзапроса, предикат "*<значение выражения> <оператор сравнения> V*" дает **TRUE**.

Общие табличные выражения (CTE)

**CTE** играет роль представления, которое создается в рамках одного запроса и, не сохраняется как объект схемы.

**Задача**: Найти максимальную сумму прихода/расхода среди всех 4-х таблиц базы данных "Вторсырье", а также тип операции, дату и пункт приема, когда и где она была зафиксирована.

**Вариант 1** **Вариант 2**

CREATE VIEW Inc\_Out AS

SELECT inc, 'inc' type, date, point

FROM Income

UNION ALL

SELECT inc, 'inc' type, date, point

FROM Income\_o

UNION ALL

SELECT out, 'out' type, date, point

FROM Outcome\_o

UNION ALL

SELECT out, 'out' type,date, point

FROM Outcome;

GO

SELECT inc AS max\_sum, type, date, point

FROM Inc\_Out WHERE inc >= ALL(

SELECT inc FROM Inc\_Out);