String user = "root";

String password = "IRONMAN11";

**SELECT distinct maker FROM product order by maker DESC**;-----выбрать неповторяющиеся (отчетливый) maker из таблицы product (сортировать по) maker от(большего к меньшему) или с конца алфавита к началу. По умолчанию сортировка от меньшего к большему.(descending-нисходящий)

**SELECT distinct maker, model FROM product order by maker desc, model desc;** ----distinct сортирует строки,при этом maker может повторятся из строки в строку но значения model при этом будут отличатся, т.е. строка всё равно будет (distinct) от всех остальных. И здесь сортировка по maker и model, каждая из которых заканчивается desc, если написать только один desc к прим. order by maker, model desc; сортировка будет по одному(последнему), по model;

**select distinct maker, model from product where type!='laptop' order by maker desc;**---- where- это условие (condition) осуществляет выборку по горизонтали и применяется с операторами сравнения =, <, >, <>-не равно, <=, >=,

SELECT \* FROM Planets WHERE Radius BETWEEN 3000 AND 9000 <-- пример предиката \*\*\*BETWEEN\*\*\* может включать в себя предикаты AND, OR, NOT, LIKE, BETWEEN, IS, IN, ключевое слово NULL,или как

в джаве можно использовать выражения типа ( что-то < (speed \* 2) );

символьные строки записываются в одинарных кавычках. В "джава" в 'SQL'

\* SELECT PlanetName, OpeningYear, Opener вывести (PlanetName), (OpeningYear) и имя первооткрывателя (Opener), планет, чье название не начинается или не заканчивается на букву «s»:

\*\* FROM Planets

\*\* WHERE PlanetName NOT LIKE '%s' <-- ЗДЕСЬ МАЛЕНЬКОЕ s ЗНАЧИТ ОБОЗНАЧАЕТ КОНЕЦ СЛОВА

\* AND PlanetName NOT LIKE 'S%' <-- ЗДЕСЬ БОЛЬШОЕ S ЗНАЧИТ ОБОЗНАЧАЕТ НАЧАЛО СЛОВА

ALTER TABLE room AUTO\_INCREMENT = 1

ALTER TABLE visit AUTO\_INCREMENT = 1

DELETE FROM `hotel`.`visit` WHERE (`ID` = '1');

ALTER TABLE visit AUTO\_INCREMENT = 1

ALTER TABLE visit\_ID AUTO\_INCREMENT = 1

ALTER TABLE roomservice\_id AUTO\_INCREMENT = 1

—---------------------------------------------------------**UPDATE**—-----------------------------------------------------------------------------

UPDATE table\_name SET column1 = value1, column2 = value2, … WHERE condition; ←синтаксис!

UPDATE room SET price ='new\_value' WHERE whatever='somevalue'-----------------------------------изменяет значение отдельного столбца в строке

"UPDATE room SET status = ? WHERE room.roomNamber= ? ";----------изменяет значение отдельного столбца в строке c помощью PreparedStatement preparedStatement = con.prepareStatement(sql);

—-------------------------------------------------------------------------------------------------------------------------------------------------------------------

DISТINCT можно задать только один раз для данного предложения SELECT. Если SELECT извлекает

множество полей, то он исключает строки, в которых все выбранные поля идентичны. Строки,

в которых некоторые зна­чения одинаковы, а другие -различны, включаются в результат.

DISТINCT, факти­чески, действует на всю выходную строку, а не на отдельное поле

Предложение WHERE

команды SELECT позволяет опре­делить предикат,(т.е.)условие, которое может

быть либо истинным, либо ложным для ка­ждой строки таблицы. Команда извлекает только те

строки из таблицы, для которых предикат имеет значение "истина".

mysql> SELECT sname FROM salespeople WHERE city='London';

-------------------------------------------------------------------------------------------------------------------------------------------

Стандартные булевы опе­раторы, распознаваемые SQL, -это AND, OR, NОТ.

• AND берет два булевых выражения (в виде А AND В) в качестве аргументов и дает в

результате истину, если они оба истинны.

• OR два булевых выражения (в виде А OR В) в качестве аргументов и оценивает результат

как истину, если хотя бы один из них истинен.

• NOT берет единственное булево выражение (в виде NOT А) в качестве аргу­мента и

изменяет его значение с истинного на ложное или с ложного на истин­ное.

SELECT \* FROM customers WHERE city='San Jose' OR NOT raiting>200;

выбрать всё из customers где city равно 'San Jose' или raiting не больше 200(меньше 200)

+------+----------+----------+---------+------+ РАЗБОР :

| cnum | cname | city | raiting | snum |

+------+----------+----------+---------+------+

| 2001 | Hoffman | London | 100 | 1001 |city!='San Jose' no raiting <200 poetomy vivodim

| 2002 | Giovanni | Rome | 200 | 1003 |city!='San Jose' no raiting <200 poetomy vivodim

| 2003 | Liu | San Jose | 200 | 1002 |city=='San Jose' poetomy vivodim

| 2006 | Clements | London | 100 | 1001 |city!='San Jose' no raiting <200 poetomy vivodim

| 2008 | Cisneros | San Jose | 300 | 1007 |city=='San Jose' poetomy vivodim

| 2007 | Perejra | Rome | 100 | 1004 |city!='San Jose' no raiting <200 poetomy vivodim

+------+----------+----------+---------+------+

+------+----------+----------+---------+------+

| 2004 | Grass | Berlin | 300 | 1002 |city!='San Jose'i raiting>200 poetomy ne vivodim

+------+----------+----------+---------+------+

-------------------------------------------------------------------------------------------------------------------------------------------

SQL понимает круглые скобки следующим образом:

все то, что расположено внут­ри крутлых скобок,

вычисляется прежде всего и рассматривается как единственное выражение по отношению к тому,

что расположено за пределами крутлых скобок (это соответствует стандартной

интерпретации в математике).

-------------------------------------------------------------------------------------------------------------------------------------------

ОПЕРАТОР "IN"

полностью определяет множество (то,что в скобках), которому данное значение может

принадлежать или не принадлежать.т.е. выводит в результат то, что в скобках

(то, что в in);

SELECT \* FROM salespeople WHERE city='BARSELONA' OR city ='London'; <--простой запрос

тоже самле но с "IN"

SELECT \* FROM salespeople WHERE city IN ('BARSELONA','London');

т.е. выведи все строки из salespeople где city = всё-то, что в скобках.

IN определяет множество, элементы которого точно перечисляются в круглых скобках и

разделяются запятыми. Если в поле, имя которого ука­зано слева от IN, есть одно из

перечисленных в списке значений (требуется точное совпадение), то предикат считается

истинным. Если элементы множества имеют чи­словой, а не символьный тип, то одиночные

кавычки непосредственно слева и справа от значения необходимо опустить.

SELECT \* FROM customers WHERE snum IN (1001, 1007, 1004);

—--------------------------------------------------------------------------------------------------------------------------------------------

**REGEXP:**

Регулярное выражение может принимать следующие специальные символы:

^: указывает на начало строки

$: указывает на конец строки

.: соответствует любому одиночному символу

[символы]: соответствует любому одиночному символу из скобок

[начальный\_символ-конечный\_символ]: соответствует любому одиночному символу из диапазона символов

|: отделяет два шаблона строки, и значение должно соответствовать одну из этих шаблонов

Примеры REGEXP:

WHERE ProductName REGEXP 'Phone': строка должна содержать "Phone", например, iPhone X, Nokia Phone N, iPhone

WHERE ProductName REGEXP '^Phone': строка должна начинаться с "Phone", например, Phone 34, PhoneX

WHERE ProductName REGEXP 'Phone$': строка должна заканчиваться на "Phone", например, iPhone, Nokia Phone

WHERE ProductName REGEXP 'iPhone [78]';: строка должна содержать либо iPhone 7, либо iPhone 8

WHERE ProductName REGEXP 'iPhone [6-8]';: строка должна содержать либо iPhone 6, либо iPhone 7, либо iPhone 8

-------------------------------------------------------------------------------------------------------------------------------------------

Оператор BETWEEN

сходен с IN. Вместо перечисления элементов множества, как это делается в IN,

BETWEEN ЗАДАЁТ ГРАНИЦЫ, в которые должно попадать значение, чтобы предикат был истинным.

Используется ключевое слово BETWEEN, за которым следуют начальное значение, ключевое

слово AND и конечное значение. т.е. BETWEEN начальное\_значение and канечное\_значение;

пример: SELECT \* FROM salsepeople WHERE comm BETWEEN 10 AND 12;

Оператор BETWEEN является включающим, т.е. граничные значения

(в данном примере это 10 и 12) включаются в предикат.

интересный пример: SELECT \* FROM salespeopl WHERE (comm BETWEEN 10 AND 12)

AND NOT comm IN (10, 12);

первое булево выражение (comm BETWEEN 10 AND 12);

второе булево выражение AND NOT comm IN (10, 12);

В первом, сравниваются условие и если (comm=10 или 11 или 12 то вернётся true)

дальше второе условие и если сomm =11. То ето строка выводится. т.е условие предиката

(если сomm=10,11,12 и comm не равенн 10, 12 то эту строку выводим)<--ето равносильно

(если comm=11 выводим на экран);

BETWEEN действует на символьных полях, представленных в двоичном (ASCII) эквиваленте,

т.е. для выборки можно вос­пользоваться алфавитным порядком. Следующий запрос выбирает

всех покупателей имена которых попадают в заданный алфавитный диапазон:

SELECT • FROM Customers WHERE cname BEТWEEN 'A' AND 'G';

-------------------------------------------------------------------------------------------------------------------------------------------

LIKE

применим только к полям типа CHAR или VARCHAR, поскольку он исполь­зуется для поиска

подстрок. Другими словами, он осуществляет просмотр строки для выяснения: входит

ли заданная подстрока в указанное поле. С этой же целью используются шаблоны

специальные символы, которые могут обозначать все, что угодно. Существует два типа

шаблонов, используемых с LIКE:

--Символ "подчеркивание" (\_) заменяет один любой символ. Например, образцу 'Ь\_t'

соответствуют 'Ьаt' или 'Ьit', но не соответствует 'Ьrat'. Так как в brat между

b и t (b \_ \_ T) 2-а символа.

--Символ "процент" (%) заменяет последовательность символов произвольной длины, в том

числе и нулевой. Например, образцу '%p%t' соответствуют 'put', 'posit', 'opt', но не

'spite'.

SELECT что-либо FROM откуда-либо WHERE last\_name LIKE 'Ber%'; <-вернёт все имена которые

наченаются на Ber и заканчиваются на любые символы с любой длинной. Что-то типа:Bers,

Berman, Bertran...

SELECT что-либо FROM откуда-либо WHERE last\_name LIKE '%ns%'; <-вернёт имена которые

содержат ns. Что-то тип: Venskii, Ansin, Konski...

SELECT что-либо FROM откуда-либо WHERE last\_name LIKE 'Ber\_ard'; <-вернёт 'Bernard',

'Berzard', 'Bermard', 'Bersard'...

SELECT что-либо FROM откуда-либо WHERE last\_name LIKE '12345\_'; <-вернёт 123456,123454,

123453,123452,123451...

SELECT что-либо FROM откуда-либо WHERE last\_name NOT LIKE 'R%'; Помещая оператора NOT

перед условием LIKE, получаем элементы которые не наченаются на 'R'.

Предположим, вы хотели найти символы % или \_ в условии LIKE. Вы можете сделать это,

используя символ Escape.

Обратите внимание что вы можете определить только escape-символ как один символ(длина=1).

SELECT \* FROM suppliers WHERE supplier\_name LIKE 'B\%';<-- вернёт элементы с именем B%.

По умолчанию Escape символ это '\' соответственно символ следующий после \ будет рас-

сматриватся как литерал и проверятся на наличие или совпадение в искомом элементе.

Также можно переопределить Escape символ используя key-word Escape.

SELECT \* FROM suppliers WHERE supplier\_name LIKE 'Br!%' ESCAPE '!';<--вернёт элементы с

именем 'Br%'

SELECT \* FROM suppliers WHERE supplier\_name LIKE 'H%\%';<--вернёт злементы с именем

'Hello%', 'Hi%', 'Hachapyri%'...

SELECT \* FROM suppliers WHERE supplier\_name LIKE 'H%\\_';<--вернёт элементы с именем

'Hello\_', 'Hi\_', Hachapyri\_'...

Как я это вижу: SELECT что-либо FROM откуда-либо WHERE LIKE 'H%\\_';

выведи строку похожую на 'H%\\_';

можно использовать NOT BETWEEN и NOT LIKE, NOT IN,

-------------------------------------------------------------------------------------------------------------------------------------------

функции агрегирования

• COUNT определяет количество строк (в столбце)или значений поля, выбранных посредством

запроса и не являющихся NULL-значениями;

• SUM вычисляет арифметическую сумму всех выбранных значений данного поля;

• АVG вычисляет среднее значение для всех выбранных значений данного поля;

• МАХ вычисляет наибольшее из всех выбранных значений данного поля;

• MIN вычисляет наименьшее из всех выбранных значений данного поля.

Для SUM и AVG могут использоваться только цифровые поля. Для COUNT, МАХ и

MIN -циф­ровые и символьные поля.

SELECT COUNT(ODATE) FROM orders;<--вернёт число 10

SELECT COUNT(DISTINCT ODATE)FROM orders;<--вернёт число 5(только уникальные значения)

Для подсчета об­щего количества строк в таблице следует использовать функцию COUNT

со звездочкой COUNT со звездочкой включает как NULL-значения, так и повторяющиеся

значения, DISTINCT в этом случае не применим

SELECT COUNT (ALL rating) FROM Customers;<--пощитает строки с учётом дубликатов но не

учитывая null-значения.

Заметка: Если в выражении используются агрегированные ф-ции надо использовать group by, либо

необходимо чтобы все перечисленные столбцы в SELECT использовали агрегированные ф-ции.

----------------------------------------------------------------------------------------------

GROUP BY

используется для объединения результатов выборки по одному или нескольким столбцам.

работает с агрегаторами:COUNT, SUM, MIN, MAX, AVG.

SELECT snum,SUM(amt) AS amt FROM orders GROUP BY snum;<--вернёт уникальные значения

snum и каждому из этих значений сумму из amt значений.

Singer Album Year Sale

The Prodigy Invaders Must Die 2008 1200000

Drowning Pool Sinner 2001 400000

Massive Attack Mezzanine 1998 2300000

The Prodigy Fat of the Land 1997 600000

The Prodigy Music For Generation 1994 1500000

Massive Attack 100th Window 2003 1200000

Drowning Pool Full Circle 2007 800000

Massive Attack Danny The Dog 2004 1900000

Drowning Pool Resilience 2013 500000

SELECT Singer, SUM(Sale) AS AllSales FROM Artists GROUP BY Singer;

результат:

Drowning Pool 1700000

Massive Attack 5400000

The Prodigy 3300000

SELECT Singer, MAX(Year) AS LastAlbumYear FROM Artists GROUP BY Singer;

результат:

Singer LastAlbumYear

Drowning Pool 2013

Massive Attack 2004

The Prodigy 2008

Предложение GROUP ВУ позволяет определять подмножество значений отдельно­го поля в

терминах другого поля и применять функции агрегирования к полученному подмножеству.

я тут почитал некоторое время спустя и честно говоря нихуя не понятно из этого описания, надо писать чтоб хотябы самому потом

понятно было а не заумные слова. Ещо приведём пример с таблицей pc, в ней есть модель, и цена на одинаковую модель может быть разной

в зависимости откомплектации. Надо пощитать сколько штук моделей существует (производится) и какая средняя цена для каждой модели

SELECT pc.model, COUNT(model) as Qty\_model, AVG(price) as Avg\_price FROM PC group by model;

model Qty\_model Avg\_price

1121 3 850.0000

1232 4 425.0000

1233 4 875.0000

1260 1 350.0000

Мы применили group by к колонке model, количество моделей 1121 =3 шт. средняя цена 850.0000 и т.д. Если убрать выражение group by то произойдёт

ошибка. Агрегатные функции используются с group by,

--------------------------------------------------------------------------------------------------------------------------------------------

inner join

An INNER JOIN permits rows from either table to appear in the result if and only if both tables meet the conditions

specified in the ON clause.

SELECT 'nazvani\_colonok' FROM 'nazvanie\_tablici' INNER JOIN 'nazvanie\_vtoroi\_tablici' ON 'nazvanie\_kolonki\_pervoi\_tablici'=

'nazvanie\_kolonko\_vtoroi\_tablici';

Naprimer:

SELECT distinct product.maker, laptop.speed FROM product inner join laptop ON product.model=laptop.model WHERE laptop.hd>=10

left join

snachala vivodit mnozzistvo s levoi tablici (vse) a zatem s pravogo mnozzestva no tolko te kotorie sovpodayt s ysloviem

right join

tozze chto i left no naoborot.

-------------------------------------------------------------------------------------------------------------------------------------------

UNION

The UNION clause combines the results of two SELECT statements into a single result set.

< query 1 >

UNION [ALL]

< query 2 >

-------------------------------------------------------------------------------------------------------------------------------------------

INTERSECT

naxodit odinakovie rezyltati iz pervogo i vtorogo zaprosa. Naprimer: naiti korabli kotorie prisytstvuyt kak v tablice Ships, tak i

v tablice Outcomes: SELECT name FROM Ships INTERSECT SELECT ship FROM Outcomes; <-- vivedet odinakovie korobli iz obeix tablic.

-------------------------------------------------------------------------------------------------------------------------------------------

EXCEPT

vivodit rezyltati iz 'pervoi tblici' zatem stavim except rezyltati viborki vtoroi tablici i esli rzyltati iz pervoi tablici povtoriaytsia

vo vtoroi tablice to oni ne vivodiatsia v rezyltat. T.e. kakbi vivodim vse rezyltati iz pervoi tablici za isklycheniem rezyltatov kotorie

odinakovi s rezyltatami iz vtoroi tablici. Primer:

SELECT Product.maker from Product WHERE type='PC' <--t.e. snachala mi polychim spisok proizvoditelei kotorie proizvodiat pc

EXCEPT

SELECT Product.maker from Product WHERE type='Laptop'<--zatem mi polychim vsex proizvoditelei Laptopov.

i 'otniav' ot proizvoditelei pc proizvoditelei laptopov polychim proizvoditelei pc kotorie ne vipyskayt Laptopi.

-------------------------------------------------------------------------------------------------------------------------------------------------

HAVING

применяется после оператора group by, отрабатывает фильтровку после применения агрегатных ф-ций. Пример:

SELECT model, COUNT(model) AS Qty\_model, AVG(price) AS Avg\_price

FROM PC

GROUP BY model

HAVING AVG(price) < 800;

насамом деле поочерёдность выполнения запроса вот такая:

1)- FROM

2)- WHERE

3)- GROUP BY

4)- HAVING

5)- SELECT

6)- ORDER BY

SELECT [DISTINCT | ALL]{\*

| [<выражение для столбца> [[AS] <псевдоним>]] [,…]}

FROM <имя таблицы> [[AS] <псевдоним>] [,…]

[WHERE <предикат>]

[[GROUP BY <список столбцов>]

[HAVING <условие на агрегатные значения>] ]

[ORDER BY <список столбцов>]

В целом Having можно применять только к результату полученному из GROUP BY, так как данная

инструкция работает ТОЛЬКО с значениями полученными в результате работы агрегатных функций, и по этой же причине их нельзя

применять с WHERE (так - как WHERE работает с значениями из самой таблици, но не с ихними производными которые получаются в результате

от агрегатных ф-ций). Например есть таблица PC c столбцом HD, надо вывести все HD которые повторяются (2 или более раз)

SELECT PC.hd <--vibrat hd

FROM PC

GROUP BY hd <--grypiryem po hd chtobi polychit ynikalnie znacheniia hd (t.e esli hd=10 i hd=10 to polychaem ynikalnoe: hd\_10 = 2, ili hd=5 polychaem hd\_5 = 1, i t.d.)

HAVING count(pc.hd)>1; <-- sravnivaet imenno znachenie polychennoe v rezyltate count(pc.hd)-rezyltat- t.e. ynikalnie znacheniia, esli bolshe chem 1 to popodaet v rezyltat.

-------------------------------------------------------------------------------------------------------------------------------------------

ПОДЗАПРОСЫ представляют выражения SELECT, которые встроены в другие запросы SQL. Primeri:

SELECT \* FROM Products WHERE Price = (SELECT MIN(Price) FROM Products); <--zdes podzapros zdelan k toi-zze tablici

SELECT CreatedAt, Price, (SELECT ProductName FROM Products WHERE Products.Id = Orders.ProductId) AS Product FROM Orders;

SELECT ProductName, Manufacturer, Price, (SELECT AVG(Price) FROM Products AS SubProds WHERE SubProds.Manufacturer=Prods.Manufacturer) AS AvgPrice

FROM Products AS Prods

WHERE Price > (SELECT AVG(Price) FROM Products AS SubProds WHERE SubProds.Manufacturer=Prods.Manufacturer);<--podzaprosov mozzet bit skolko xochesh

----------------------------------------------------------------------------------------------------------------------------------------------------

SOME (ANY) и ALL -операторы КОТОРЫЕ РАБОТАЮТ С ПОДЗАПРОСАМИ. они берут все значения выведенные подзапросом и обрабатывают их как модуль(как одно целое)

соответственно переводу any-это когда хоть чтолибо соответствует условию-поподает в результат. Например:

SELECT \* FROM Salespeople WHERE city = ANY (SELECT city FROM Customers ); <-- Оператор ANY берет все значения выведенные подзапросом, ( для этого

случая - это все значения city в таблице Заказчиков ), и оценивает их как верные если любой (ANY) из их равняется значению города текущей

строки внешнего запроса.

Это означает, что подзапрос должен выбирать значения такого же типа как и те, которые сравниваются в основном предикате.

ВАЖНО: операторы работают с любыми операторами сравнения:!=, =, <, >, <=, >=.

ALL -> compares a value of the first table with all values of the second table and returns the row if there is a match with all values.

SELECT \* FROM Teachers WHERE age > ALL (SELECT age FROM Students);