**SQL STEPIK**

Создать таблицу

CREATE TABLE table\_name

(

column\_name\_1 column\_type\_1,

column\_name\_2 column\_type\_2,

...,

column\_name\_N column\_type\_N,

);

table\_name — имя таблицы;

column\_name — имя столбца;

column\_type — тип данных столбца.

CREATE TABLE book(

book\_id INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(50),

author VARCHAR(30),

price DECIMAL(8,2),

amount INT

);

// Начальное значение для AUTO\_INCREMENT по умолчанию равно 1, и оно будет увеличиваться на 1 для каждой новой записи.

//здесь мы сразу задаем первичный ключ для таблицы

**Для занесения новой записи** в таблицу используется SQL запрос, в котором указывается в какую таблицу, в какие поля заносить новые значения.

INSERT INTO book (title,author,price,amount)

VALUES ("Мастер и Маргарита","Булгаков М.А.",670.99,3);

**Выбрать все записи таблицы book .**

SELECT\*FROM book;

**Выбрать авторов, название книг и их цену из таблицы book.**

SELECT author,title,price FROM book;

**Выбрать названия книг и авторов из таблицы book, для поля title задать имя(псевдоним) Название, для поля author – Автор.**

SELECT title AS Название, author AS Автор

FROM book;

Для упаковки каждой книги требуется один лист бумаги, цена которого 1 рубль 65 копеек. Посчитать стоимость упаковки для каждой книги (сколько денег потребуется, чтобы упаковать все экземпляры книги). В запросе вывести название книги, ее количество и стоимость упаковки, последний столбец назвать **pack**.

SELECT title, amount,

1.65 \* amount AS pack

from book;

В конце года цену всех книг на складе пересчитывают – снижают ее на 30%. Написать SQL запрос, который из таблицы **book** выбирает названия, авторов, количества и вычисляет новые цены книг. Столбец с новой ценой назвать **new\_price**, цену округлить до 2-х знаков после запятой.

SELECT title,author,amount,

ROUND (price - price/100\*30,2) AS new\_price

FROM book;

// Функция SQL ROUND. Округление производится по правилам арифметики

Функция SQL ROUND(результат\_вычислений, n) округляет результат вычислений до n-го знака после запятой. Округление производится по правилам арифметики. Если n - отрицательное число (−n), то округление происходит до n-го знака перед запятой. Таким образом, с помощью функции ROUND можно получить и целое число как результат округления.SELECT ROUND(109.14,-1);

round

-------

110

(1 row)

Округление до целого:

SELECT ROUND(поле) FROM имя\_таблицы WHERE условие

Округление до определенного знака в дробной части:

SELECT ROUND(поле, сколько\_знаков\_оставить) FROM имя\_таблицы WHERE условие

ROUND (price,1) AS new\_price //если не поставить AS выведет столбец с названием ROUND (price,1)

Существует несколько встроенных функций, таких как count(), sum(), avg(), min(), max(), ROUND и др. для выполнения так называемых агрегирующих вычислений данных таблицы или колонки.

**SELECT** COUNT(\*) **AS** records **FROM** users;

// **после SELECT**

Выражение (expression) — это комбинация значений, операторов и функций для оценки (вычисления) значения.

В SQL реализована возможность заносить в поле значение в зависимости от условия. Для этого используется функция IF:

IF (логическое\_выражение, выражение\_1, выражение\_2)

Функция вычисляет логическое\_выражение, если оно истина – в поле заносится значение выражения\_1, в противном случае – значение выражения\_2. Все три параметра IF() являются обязательными.

При анализе продаж книг выяснилось, что наибольшей популярностью пользуются книги Михаила Булгакова, на втором месте книги Сергея Есенина. Исходя из этого решили поднять цену книг Булгакова на 10%, а цену книг Есенина - на 5%. Написать запрос, куда включить автора, название книги и новую цену, последний столбец назвать new\_price. Значение округлить до двух знаков после запятой.

SELECT author, title,

ROUND (IF (author = “Есенин”, price\*1,1, IF (author = “Булгаков”,price\*1,05,price)),2) AS new\_price

FROM book

//действия над тем, что мы выодим, а WHERE была бы выборка, с чем работаем

SELECT author,title,

ROUND (IF(author="Булгаков М.А.",price\*0.1+price,IF(author="Есенин С.А.",price\*0.05+price,price)),2) AS new\_price

FROM book;

Вывести автора, название и цены тех книг, количество которых меньше 10.

SELECT author,title, price

FROM book

WHERE amount<10

Вывести название, автора и стоимость (цена умножить на количество) тех книг, стоимость которых больше 4000 рублей

SELECT title, author, price \* amount AS total

FROM book

WHERE price \* amount > 4000;

Приоритеты операций:

круглые скобки

умножение (\*), деление (/)

сложение (+), вычитание (-)

операторы сравнения (=, >, <, >=, <=, <>)

NOT

AND

OR

Логическое выражение после ключевого слова WHERE может включать операторы  BETWEEN и IN. Приоритет  у этих операторов такой же как у операторов сравнения, то есть они выполняются раньше, чем **NOT**, **AND**, **OR**.

Вывести название, автора, цену и количество всех книг, цена которых меньше 500 или больше 600, а стоимость всех экземпляров этих книг больше или равна 5000.

SELECT title,author,price,amount

FROM book

WHERE (price<500 OR price>600) AND price\*amount >= 5000;

**// >=**

**// после SELECT WHERE**

Оператор  IN  позволяет выбрать данные, соответствующие значениям из списка.

Выбрать названия и цены книг, написанных Булгаковым или Достоевским.

SELECT title, price

FROM book

WHERE author IN ('Булгаков М.А.', 'Достоевский Ф.М.');

*BETWEEN Syntax*

SELECT column\_name(s)

FROM table\_name

WHERE column\_name BETWEEN value1 AND value2;

The BETWEEN operator is inclusive: begin and end values are included.

Вывести название и авторов тех книг, цены которых принадлежат интервалу от 540.50 до 800 (включая границы), а количество или 2, или 3, или 5, или 7 .

SELECT title,author

FROM book

WHERE price BETWEEN 540.50 AND 800 AND amount IN(2,3,5,7);

Вывести автора и название  книг, количество которых принадлежит интервалу от 2 до 14 (включая границы). Информацию отсортировать сначала по авторам (в обратном алфавитном порядке), а затем по названиям книг (по алфавиту).

SELECT author, title

FROM book

WHERE amount BETWEEN 2 AND 14

ORDER BY author DESC, title;

Query result:

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

| author | title |

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

| Достоевский Ф.М. | Братья Карамазовы |

| Достоевский Ф.М. | Идиот |

| Булгаков М.А. | Белая гвардия |

| Булгаков М.А. | Мастер и Маргарита |

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

Общая структура запроса выглядит следующим образом:

**SELECT** ('столбцы или \* для выбора всех столбцов; обязательно')

**FROM** ('таблица; обязательно')

**WHERE** ('условие/фильтрация, например, city = 'Moscow'; необязательно')

**GROUP** **BY** ('столбец, по которому хотим сгруппировать данные; необязательно')

**HAVING** ('условие/фильтрация на уровне сгруппированных данных; необязательно')

**ORDER** **BY** ('столбец, по которому хотим отсортировать вывод; необязательно')

SELECT ИМЯ\_СТОЛБЦА FROM ИМЯ\_ТАБЛИЦЫ

WHERE ЗНАЧЕНИЕ

ПРЕДИКАТ (IN, OR, BETWEEN) (ЗНАЧЕНИЯ, УКАЗЫВАЮЩИЕ ДИАПАЗОН)

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

* The percent sign (%) represents zero, one, or multiple characters
* The underscore sign (\_) represents one, single character

Вывести название и автора тех книг, название которых состоит из двух и более слов, а инициалы автора содержат букву «С». Считать, что в названии слова отделяются друг от друга пробелами и не содержат знаков препинания, между фамилией автора и инициалами обязателен пробел, инициалы записываются без пробела в формате: буква, точка, буква, точка. Информацию отсортировать по названию книги в алфавитном порядке.

Символ % соответствует любому количеству любых символов, а также их отсутствию.

SELECT title, author

FROM book

WHERE title LIKE "%\_ \_%" AND author LIKE "%С.%"

ORDER BY title

// ищет С с точкой

Вывести название книг, состоящих ровно из 5 букв.

Запрос:

SELECT title FROM book

WHERE title LIKE "\_\_\_\_\_"

* 'xyz%' - любые строки, которые начинаются с букв xyz;
* 'xyz\_' - строка длиной в определённое количество символов, которая обязательно начинается с указанных букв;
* '%z' - любая последовательность символов, которая заканчивается символом z;
* '%Word%' - любая последовательность символов, содержащая слово 'Word' в любой позиции строки;
* '% % %' - строка, содержащая не менее двух пробелов.

SELECT title FROM book WHERE title LIKE "\_% и \_%" /\*отбирает слово И внутри названия \*/

OR title LIKE "и \_%" /\*отбирает слово И в начале названия \*/

OR title LIKE "\_% и" /\*отбирает слово И в конце названия \*/  
OR title LIKE "и" /\* отбирает название, состоящее из одного слова И \*/

//\_% и \_%" между и должны быть пробелы и и не просто в составе слова

INSERT INTO book (title,author,price,amount)

VALUES ("Детство, отрочество, юность","Толстой Л.Н.",350,4);

SELECT title AS Название, author AS автор, price AS цена, amount AS количество, price \* amount AS сумма

FROM book;

SELECT title,

ROUND (price,1) AS new\_price

FROM book;

SELECT title, author

FROM book

WHERE amount>5;

SELECT title,author

FROM book

WHERE (price<400 OR price>540) AND price \* amount <2800

ORDER BY price DESC;

SELECT author,price,

ROUND(IF(author="Булгаков М.А.",price\*0.1+price,IF(author="Есенин С.А.",price\*0.2+price,price)),1) AS maxprice

FROM book;

SELECT author,price

FROM book

WHERE price>461

ORDER BY title DESC;

SELECT author

FROM book

WHERE author LIKE "%М.%";

SELECT author

FROM book

WHERE author LIKE "\_\_\_сто%";

SELECT \*

FROM book;

Чтобы отобрать уникальные элементы некоторого столбца используется ключевое слово DISTINCT, которое размещается сразу после SELECT.

SELECT DISTINCT author FROM book;

Другой способ – использование оператора GROUP BY, который группирует данные при выборке, имеющие одинаковые значения в некотором столбце. Столбец, по которому осуществляется группировка, указывается после GROUP BY .

С помощью GROUP BY можно выбрать уникальные элементы столбца, по которому осуществляется группировка. Результат будет точно такой же как при использовании DISTINCT.

SELECT author

FROM book

GROUP BY author;

\* Результаты будут всегда одинаковыми. Но если просто нужно получить неповторяющиеся значения - лучше использовать DISTINCT. Возможности GROUP BY значительно шире, в частности он позволяет производить вычисления над записями, входящими в группу с одинаковыми значениями в указанном после GROUP BY столбце.

Агрегатные функции SQL действуют в отношении значений столбца с целью получения единого результирующего значения. Наиболее часто применяются агрегатные функции SQL SUM, MIN, MAX, AVG и COUNT. Следует различать два случая применения агрегатных функций. Первый: агрегатные функции используются сами по себе и возвращают одно результирующее значение. Второй: агрегатные функции используются с оператором SQL GROUP BY, то есть с группировкой по полям (столбцам) для получения результирующих значений в каждой группе.

### **COUNT() Syntax**

SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

The AVG() function returns the average value of a numeric column.

**Note:** NULL values are not counted.

### **AVG() Syntax**

SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition;

The SUM() function returns the total sum of a numeric column.

**Note:** NULL values are ignored.

### **SUM() Syntax**

SELECT SUM(column\_name)  
FROM table\_name  
WHERE condition;

**Note:** NULL values are ignored.

функцию COUNT() можно применять к любому столбцу, в том числе можно использовать и \*, если таблица не содержит пустых значений. Если же в столбцах есть значения Null, (для группы по автору Есенин в нашем примере), то

* COUNT(\*) —  подсчитывает  все записи, относящиеся к группе, в том числе и со значением NULL;
* COUNT(имя\_столбца) — возвращает количество записей конкретного столбца (только NOT NULL), относящихся к группе.

ВАЖНО.

1. Если столбец указан в SELECT  БЕЗ применения групповой функции, то он обязательно должен быть указан и в GROUP BY. Иначе получим ошибку.

Нельзя:  
SELECT author, amount  
FROM book  
GROUP BY author  
  
Можно:  
SELECT author, SUM(amount)  
FROM book  
GROUP BY author

Посчитать, количество различных книг и количество экземпляров книг каждого автора , хранящихся на складе. Столбцы назвать Автор, Различных\_книг и Количество\_экземпляров соответственно.

SELECT author AS "Автор", COUNT(title) AS "Различных\_книг",SUM(amount) AS "Количество\_экземпляров"

FROM book

GROUP BY author

Порядок выполнения SQL запроса

FROM

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

К групповым функциям SQL относятся: MIN(), MAX() и AVG(), которые вычисляют минимальное, максимальное и среднее значение элементов столбца, относящихся к группе.

Вывести фамилию и инициалы автора, минимальную, максимальную и среднюю цену книг каждого автора . Вычисляемые столбцы назвать **Минимальная\_цена, Максимальная\_цена** и**Средняя\_цена** соответственно.

SELECT title, MIN(price) AS Минимальная­­\_цена, MAX(price) AS Максимальная\_цена, AVG(price) AS Средняя\_цена

FROM book

GROUP BY author;

Для каждого автора вычислить суммарную стоимость книг S (имя столбца Стоимость), а также вычислить налог на добавленную стоимость для полученных сумм (имя столбца НДС ) , который включен в стоимость и составляет k = 18%, а также стоимость книг (Стоимость\_без\_НДС) без него. Значения округлить до двух знаков после запятой. В запросе для расчета НДС(tax) и Стоимости без НДС(S\_without\_tax) использовать следующие формулы:

*tax*=*S*∗*k/100*​​,

*1+k/100*

*S\_without\_tax=S/1+k/100*

SELECT author,ROUND(SUM(price\*amount),2) AS Стоимость,ROUND((SUM(price\*amount)\*18/100)/(1+18/100),2) AS НДС, ROUND((SUM(price\*amount))/(1+18/100),2) AS Стоимость\_без\_НДС

FROM book

GROUP BY author;

Вывести цену самой дешевой книги, цену самой дорогой и среднюю цену уникальных книг на складе. Названия столбцов Минимальная\_цена, Максимальная\_цена, Средняя\_цена соответственно. Среднюю цену округлить до двух знаков после запятой.

FROM book

SELECT DISTINCT MIN(price) AS Минимальная\_цена,MAX(price) AS //distinct один!!! Максимальная\_цена,ROUND(AVG(price),2) AS Средняя\_цена

FROM book;

Чтобы отобрать уникальные элементы некоторого столбца используется ключевое слово DISTINCT, которое размещается сразу после SELECT.

Вычислить среднюю цену и суммарную стоимость тех книг, количество экземпляров которых принадлежит интервалу от 5 до 14, включительно. Столбцы назвать Средняя\_цена и Стоимость, значения округлить до 2-х знаков после запятой.

SELECT ROUND (AVG(price),2) AS Средняя\_цена,ROUND(SUM(price\*amount),2) AS Стоимость

FROM book

WHERE amount BETWEEN 5 AND 14

В запросы с групповыми функциями можно включать условие отбора строк, которое в обычных запросах записывается после WHERE. В запросах с групповыми функциями вместо WHERE используется ключевое слово HAVING , которое размещается после оператора GROUP BY.

WHERE используется для фильтрации строк перед операцией group by, в то время как предложение HAVING используется для фильтрации строк после операции group by.

Пример

Найти минимальную и максимальную цену книг всех авторов, общая стоимость книг которых больше 5000.

SELECT author,

MIN(price) AS Минимальная\_цена,

MAX(price) AS Максимальная\_цена

FROM book

GROUP BY author

HAVING SUM(price \* amount) > 5000;

РАЗНИЦА WHERE и HAVING

SELECT ROUND(AVG(price), 2) AS Средняя\_цена,

SUM(price\*amount) AS Стоимость

FROM book

GROUP BY amount

HAVING amount BETWEEN 5 AND 14;

- Здесь мы сгруппировали по amount, а потом уже среди сгруппированных записей с одинаковым amount вычисляли среднюю цену и стоимость

SELECT ROUND(AVG(price),2) AS Средняя\_цена,

SUM(price\*amount) AS Стоимость

FROM book

WHERE amount BETWEEN 5 and 14;

Здесь мы выбрали цену и стоимость, удовлетворяющих between и уже в столбце вычисляли среднюю цену и стоимость

а теперь результат

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

| Средняя\_цена | Стоимость |

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

| 540.50 | 2702.50 |

| 470.25 | 9405.00 |

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

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

| Средняя\_цена | Стоимость |

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

| 493.67 | 12107.50 |

WHERE и HAVING могут использоваться в одном запросе. При этом необходимо учитывать порядок выполнения SQL запроса на выборку на СЕРВЕРЕ:

FROM

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

Сначала определяется таблица, из которой выбираются данные (FROM), затем из этой таблицы отбираются записи в соответствии с условием WHERE, выбранные данные агрегируются (GROUP BY), из агрегированных записей выбираются те, которые удовлетворяют условию после HAVING. Потом формируются данные результирующей выборки, как это указано после SELECT ( вычисляются выражения, присваиваются имена и пр. ). Результирующая выборка сортируется, как указано после ORDER BY.

Важно! Порядок ВЫПОЛНЕНИЯ запросов - это не порядок ЗАПИСИ ключевых слов в запросе на выборку. Порядок записи (синтаксис запроса) остается таким же, как рассматривался ранее в курсе. Порядок ВЫПОЛНЕНИЯ нужен для того, чтобы понять, почему, например, в WHERE нельзя использовать имена выражений из SELECT. Просто SELECT выполняется компилятором позже, чем WHERE, поэтому ему неизвестно, какое там выражение написано.

Посчитать стоимость всех экземпляров каждого автора без учета книг «Идиот» и «Белая гвардия». В результат включить только тех авторов, у которых суммарная стоимость книг (без учета книг «Идиот» и «Белая гвардия») более 5000 руб. Вычисляемый столбец назвать Стоимость. Результат отсортировать по убыванию стоимости.

SELECT author, SUM(price\*amount) AS Стоимость

FROM book

WHERE title <> "Идиот" AND title <> "Белая гвардия" // не равно одновременно, поэтому AND

GROUP BY author

HAVING SUM(price\*amount)>5000

ORDER BY Стоимость DESC;

//ORDER BY использую Cтоимость , потому что выполняется после Select

SELECT author

FROM book;

SELECT DISTINCT author

FROM book; // вывести только уникальные фио авторов, те без повторов

SELECT author

FROM book

GROUP BY author;

SELECT author, COUNT(author)

FROM book

GROUP BY author;

SELECT COUNT(\*)

FROM book; //количество строк в таблице

SELECT SUM(amount)

FROM book;

SELECT author as "Автор",COUNT(title) AS "Различных книг",SUM(amount) AS "Количество экземпляров" // COUNT – число строк, соответственно считаем кол-во книг автора, SUM – сумма по столбцу

FROM book

GROUP BY author;

SELECT author, MIN(price) //минимальная цена для книг каждого автора

FROM book

GROUP BY author;

SELECT author, ROUND(SUM(price\*amount),1) AS "Суммарная стоимость" // суммарная округленная стоимость всех книг каждого автора

FROM book

GROUP BY author;

SELECT MIN(price), ROUND(AVG(price),2) AS "Средняя цена" // минимальная и средняя всех книг на складе

FROM book;

SELECT author, MIN(price) AS "Мин цена", MAX(price) AS "Макс цена"

FROM book

GROUP BY author

HAVING SUM(price\*amount)>5000; // сначала from, потом group by, потом having, потом select

SELECT AVG(price) AS "Средняя цена", SUM(price\*amount) AS "Стоимость" //средняя цена и стоимость всех книг, где кол-во от 5 до 14

FROM book

WHERE amount BETWEEN 5 AND 14;

SELECT author, SUM(price\*amount) AS "Стоимость"

FROM book

WHERE title <> "Белая гвардия" AND title <> "Идиот"

GROUP BY author

HAVING SUM(price\*amount) >5000

ORDER BY Стоимость DESC //стоимость без кавычек. Стоимость, потому что ORDER BY после SELECT

Вложенные запросы могут включаться в WHERE или HAVING так (в квадратных скобках указаны необязательные элементы, через | – один из элементов):

* [WHERE](https://stepik.org/lesson/297514/step/2?unit=279274) | [HAVING](https://stepik.org/lesson/297515/step/7?unit=279275) выражение оператор сравнения (вложенный запрос);
* [WHERE](https://stepik.org/lesson/297514/step/3?unit=279274) | [HAVING](https://stepik.org/lesson/297515/step/7?unit=279275) выражение, включающее вложенный запрос;
* [WHERE](https://stepik.org/lesson/297514/step/4?unit=279274) | [HAVING](https://stepik.org/lesson/297515/step/7?unit=279275) выражение [NOT] IN (вложенный запрос);
* [WHERE](https://stepik.org/lesson/297514/step/5?unit=279274) | [HAVING](https://stepik.org/lesson/297515/step/7?unit=279275) выражение  оператор\_сравнения  ANY | ALL (вложенный запрос).

Также вложенные запросы могут вставляться в основной запрос после ключевого слова [SELECT](https://stepik.org/lesson/297514/step/6?unit=279274)

Вложенный запрос — это запрос на выборку, который используется внутри инструкции SELECT, INSERT, UPDATE или delete или внутри другого вложенного запроса. Подзапрос может быть использован везде, где разрешены выражения. Пример структуры вложенного запроса SELECT поля\_таблиц FROM список\_таблиц WHERE конкретное\_поле. IN (SELECT поле\_таблицы FROM таблица).

Вложенные подзапросы всегда применяются тогда, когда для выполнения основного запроса необходимо использовать данные, находящиеся в той же или других таблицах, которые невозможно получить при помощи соединения таблиц. Например, чтобы определить, какие хобби имеют степень риска, превышающую среднюю степень риска, необходимо предварительно вычислить эту среднюю степень риска.

Вывести информацию (автора, название и цену) о книгах, цены которых меньше или равны средней цене книг на складе. Информацию вывести в отсортированном по убыванию цены виде. Среднее вычислить как среднее по цене книги.

SELECT author, title, price

FROM book

WHERE price <= ( //сравнение перед вложенным запросом

SELECT AVG(price)

FROM book

)

ORDER BY price DESC;

# Функция ABS () в SQL возвращает модуль числа, то есть выполняет преобразование из отрицательного числа в положительное.

Вывести информацию о книгах, количество экземпляров которых отличается от среднего количества экземпляров книг на складе более чем на 3. То есть нужно вывести и те книги, количество экземпляров которых меньше среднего на 3, или больше среднего на 3.

SELECT title, author, amount

FROM book

WHERE ABS(amount - (SELECT AVG(amount) FROM book)) >3;

Вывести информацию (автора, название и цену) о тех книгах, цены которых превышают минимальную цену книги на складе не более чем на 150 рублей в отсортированном по возрастанию цены виде.

SELECT author, title, price

FROM book

WHERE (price-(SELECT MIN(price) FROM book))<=150

ORDER BY price;

Вывести информацию (автора, книгу и количество) о тех книгах, количество экземпляров которых в таблице book не дублируется.

SELECT author, title, amount

FROM book

WHERE amount IN (

SELECT amount

FROM book

GROUP BY amount

HAVING COUNT(amount)=1);

Вложенный запрос, возвращающий несколько значений одного столбца, можно использовать для отбора записей с помощью операторов ANY и ALL совместно с операциями отношения (=, <>, <=, >=, <, >).

Операторы ANY и ALL используются в SQL для сравнения некоторого значения с результирующим набором вложенного запроса, состоящим из одного столбца. При этом тип данных столбца, возвращаемого вложенным запросом, должен совпадать с типом данных столбца (или выражения), с которым происходит сравнение.

При использовании оператора ANY в результирующую таблицу будут включены все записи, для которых выражение со знаком отношения верно хотя бы для одного элемента результирующего запроса. Как работает оператор ANY:

* amount > ANY (10, 12) эквивалентно amount > 10
* amount < ANY (10, 12) эквивалентно amount < 12
* amount = ANY (10, 12) эквивалентно (amount = 10) OR (amount = 12), а также amount IN  (10,12)
* amount <> ANY (10, 12) вернет все записи с любым значением amount, включая 10 и 12

При использовании оператора ALL в результирующую таблицу будут включены все записи, для которых выражение со знаком отношения верно для всех элементов результирующего запроса. Как работает оператор ALL:

* amount > ALL (10, 12) эквивалентно amount > 12
* amount < ALL (10, 12) эквивалентно amount < 10
* amount = ALL (10, 12) не вернет ни одной записи, так как эквивалентно (amount = 10) AND (amount = 12)
* amount <> ALL (10, 12) вернет все записи кроме тех, в которых amount равно 10 или 12

Важно! Операторы ALL и ANY можно использовать только с вложенными запросами. В примерах выше (10, 12) приводится как результат вложенного запроса просто для того, чтобы показать как эти операторы работают. В запросах так записывать нельзя.

Вывести информацию о книгах(автор, название, цена), цена которых меньше самой большой из минимальных цен, вычисленных для каждого автора.

SELECT author, title, price

FROM book

WHERE price <ANY( // select min (price) пишем столбец, к которому применяем <ANY

SELECT MIN(price)

FROM book

GROUP BY author

);

Вложенный запрос может располагаться после ключевого слова SELECT. В этом случае результат выполнения запроса выводится в отдельном столбце результирующей таблицы. При этом результатом запроса может быть только одно значение, тогда оно будет повторяться во всех строках. Также вложенный запрос может использоваться в выражениях.

FLOOR(salary) – используется для округления дробей до целого в меньшую сторону. 300.90 станет 300

Вывести информацию о книгах, количество экземпляров которых отличается от среднего количества экземпляров книг на складе более чем на 3,  а также указать среднее значение количества экземпляров книг.

SELECT title, author, amount, AVG(amount)

FROM book

WHERE ABS(amount-AVG(amount)>3

SELECT title, author, amount,

(

SELECT AVG(amount)

FROM book

) AS Среднее\_количество

FROM book

WHERE abs(amount - (SELECT AVG(amount) FROM book)) >3;

Среднее количество в виде дробного числа выглядит не очень правильно. Полученное значение можно округлить "вниз" - до ближайшего меньшего целого.

SELECT title, author, amount, FLOOR((SELECT AVG(amount) FROM book)) AS Среднее\_количество FROM book WHERE ABS(amount - (SELECT AVG(amount) FROM book)) >3;

Посчитать сколько и каких экземпляров книг нужно заказать поставщикам, чтобы на складе стало одинаковое количество экземпляров каждой книги, равное значению самого большего количества экземпляров одной книги на складе. Вывести название книги, ее автора, текущее количество экземпляров на складе и количество заказываемых экземпляров книг. Последнему столбцу присвоить имя Заказ. В результат не включать книги, которые заказывать не нужно.

SELECT title, author, amount,

(SELECT MAX(amount) FROM book) - amount AS Заказ

FROM book

WHERE (SELECT MAX(amount) FROM book)-amount>0; // select в скобках

Допускается вставка нескольких записей одновременно, для этого используется SQL запрос следующего вида:

INSERT INTO имя\_таблицы(столбец\_1, столбец\_2, ..., столбец\_N)

VALUES

(значение\_1\_1, значение\_1\_2, ..., значение\_1\_N),

(значение\_2\_1, значение\_2\_2, ..., значение\_2\_N),

...

(значение\_M\_1, значение\_M\_2, ..., значение\_M\_N);

Например, чтобы добавить в таблицу book две новые записи используется запрос:

INSERT INTO book (title, author, price, amount)

VALUES

('Война и мир','Толстой Л.Н.', 1070.20, 2),

('Анна Каренина', 'Толстой Л.Н.', 599.90, 3);

## Задание

Занесите в таблицу**supply** четыре записи, чтобы получилась следующая таблица:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **supply\_id** | **title** | **author** | **price** | **amount** |
| 1 | Лирика | Пастернак Б.Л. | 518.99 | 2 |
| 2 | Черный человек | Есенин С.А. | 570.20 | 6 |
| 3 | Белая гвардия | Булгаков М.А. | 540.50 | 7 |
| 4 | Идиот | Достоевский Ф.М. | 360.80 | 3 |

INSERT INTO supply(title,author, price,amount  
VALUES (“Лирика”,” Пастернак Б.Л.”,”518.99”,”2”),

(“Лирика”,” Пастернак Б.Л.”,”518.99”,”2”),

INSERT INTO supply(title, author, price, amount)

VALUES

("Лирика","Пастернак Б.Л.",518.99,2),

("Черный человек","Есенин С.А.",570.20,6),

("Белая гвардия","Булгаков М.А.",540.50,7),

("Идиот","Достоевский Ф.М.",360.80,3);

SELECT\* FROM supply;

DDL — язык определения данных (Data Definition Language)

N Команда Описание

1 CREATE Создает новую таблицу, представление таблицы или другой объект в БД

2 ALTER Модифицирует существующий в БД объект, такой как таблица

3 DROP Удаляет существующую таблицу, представление таблицы или другой объект в БД

С помощью запроса на добавление можно не только добавить в таблицу конкретные значения (список VALUES), но и записи из другой таблицы, отобранные с помощью запроса на выборку.  В этом случае вместо раздела VALUES записывается запрос на выборку, начинающийся с SELECT.  В нем можно использовать WHERE, GROUP BY, ORDER BY.

Правила соответствия между полями таблицы и вставляемыми значениями из запроса:

1. количество полей в таблице и количество полей в запросе должны совпадать;
2. должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый столбец запроса должен относиться к первому столбцу в списке столбцов таблицы, второй – ко второму столбцу и т.д.
3. типы столбцов запроса должны быть совместимы с типами данных соответствующих столбцов таблицы ( целое число можно занести в поле типа DECIMAL, обратная операция – недопустима).

Занести все книги из таблицы supply в таблицу book.

INSERT INTO book (title, author, price, amount)

SELECT title, author, price, amount

FROM supply;

SELECT \* FROM book;

Добавить из таблицы supply в таблицу book, все книги, кроме книг, написанных Булгаковым М.А. и Достоевским Ф.М.

INSERT INTO book (title, author, price, amount)

SELECT title, author, price, amount

FROM supply

WHERE author <>"Булгаков М.А." AND author <>"Достоевский Ф.М."; //**одновременно** не равно

SELECT \* FROM book;

Занести из таблицы supply в таблицу book только те книги, авторов которых нет в book.

INSERT INTO book

SELECT title, author from supply

WHERE author NOT IN( SELECT author FROM book);

SELECT\* from book

Под обновлением данных подразумевается изменение значений в существующих записях таблицы. При этом возможно как изменение значений полей в группе строк (даже всех строк таблицы), так и правка значения поля отдельной строки.

Изменение записей в таблице реализуется с помощью запроса UPDATE. Простейший запрос на обновление выглядит так:

UPDATE таблица SET поле = выражение

В таблице book необходимо скорректировать значение для покупателя в столбце buy таким образом, чтобы оно не превышало количество экземпляров книг, указанных в столбце amount. А цену тех книг, которые покупатель не заказывал, снизить на 10%.

SELECT\* FROM book;

UPDATE book

SET buy =IF(buy>amount, amount, buy),price = IF(buy=0,price=0.9\*price,price);

SELECT\* FROM book

При необходимости обновлять сразу несколько полей, все поля с их значениями указываются после директивы SET через запятую. Например, нужно изменить название и цену товара с кодом 2 на «утюг», стоимостью 300:

UPDATE goods

SET title = "утюг", price = 300

WHERE num = 2

Такой запрос каждому соответствующему полю в строке назначит его значение. А условие укажет, в какой именно строке будут выполнены изменения.

Для тех книг в таблице **book** , которые есть в таблице supply, не только увеличить их количество в таблице **book** ( увеличить их количество на значение столбца **amount** таблицы **supply**), но и пересчитать их цену (для каждой книги найти сумму цен из таблиц **book** и supply и разделить на 2).

"для каждой книги найти сумму цен из таблиц **book** и **supply** и разделить на 2"

проще для понимания - "Сложить стоимость единицы в таблицах**book** и **supply** и разделить на 2"

UPDATE book,supply

SET book.amount=book.amount+supply.amount, book.price=(book.price+supply.price)/2

WHERE book.title=supply.title AND book.author=supply.author;

SELECT \*FROM book;

Команда UPDATE может обновить значения сразу в нескольких таблицах за один раз. Допустим, мы хотим обновить адрес одного из магазинов, и тут же обновить количество товара в нем:

UPDATE stores, products SET stores.address = 'Пятницкая 23', products.quantity = 3 WHERE stores.id = 4 and products.store\_id = 4;

В запросах на обновление можно использовать несколько таблиц, но тогда

* для столбцов, имеющих одинаковые имена, необходимо указывать имя таблицы, к которой они относятся, например, book.price – столбец price из таблицы book, supply.price – столбец price из таблицы supply;
* все таблицы, используемые в запросе, нужно перечислить после ключевого слова UPDATE;
* в запросе обязательно условие WHERE, в котором указывается условие при котором обновляются данные.

Запрос на удаления позволяет удалить не все записи таблицы, а только те, которые удовлетворяют условию, указанному после ключевого слова WHERE:

DELETE FROM таблица

WHERE условие;

Удалить из таблицы **supply** книги тех авторов, общее количество экземпляров книг которых в таблице **book** превышает 10.

DELETE from supply

WHERE author IN(

SELECT author

FROM book

GROUP BY author

HAVING SUM(amount)>10

);

SELECT\*FROM supply;

Новая таблица может быть создана на основе данных из другой таблицы. Для этого используется запрос SELECT, результирующая таблица которого и будет новой таблицей базы данных. При этом имена столбцов запроса становятся именами столбцов новой таблицы. Запрос на создание новой таблицы имеет вид:

CREATE TABLE имя\_таблицы AS

SELECT ...

Создать таблицу заказ (**ordering**), куда включить авторов и названия тех книг, количество экземпляров которых в таблице **book** меньше 4. Для всех книг указать одинаковое значение - среднее количество экземпляров книг в таблице **book**.

CREATE TABLE ordering AS

SELECT author, title, AVG(amount)

FROM book

WHERE amount>10

CREATE TABLE ordering AS

SELECT author,title,(SELECT ROUND(AVG(amount)) FROM book) AS amount

FROM book

WHERE amount<(SELECT AVG(amount) FROM book);

SELECT\*FROM ordering; // уже новая таблица

CREATE TABLE ordering AS

SELECT author, title,

(

SELECT ROUND(AVG(amount))

FROM book

) AS amount

FROM book

WHERE amount < 4;

SELECT \* FROM ordering;

Создать таблицу заказ (**ordering**), куда включить авторов и названия тех книг, количество экземпляров которых в таблице **book** меньше среднего количества экземпляров книг в таблице **book**. В таблицу включить столбец   **amount,** в котором для всех книг указать одинаковое значение - среднее количество экземпляров книг в таблице **book**.

CREATE TABLE ordering AS

SELECT author,title,(SELECT ROUND(AVG(amount)) FROM book) AS amount

FROM book

WHERE amount<(SELECT AVG(amount) FROM book);

SELECT\*FROM ordering;

Важно понимать, как соотносятся агрегатные функции и SQL-предложения WHERE и HAVING. Основное отличие WHERE от HAVING заключается в том, что WHERE сначала выбирает строки, а затем группирует их и вычисляет агрегатные функции (таким образом, она отбирает строки для вычисления агрегатов), тогда как HAVING отбирает строки групп после группировки и вычисления агрегатных функций. Как следствие, предложение WHERE не должно содержать агрегатных функций; не имеет смысла использовать агрегатные функции для определения строк для вычисления агрегатных функций. Предложение HAVING, напротив, всегда содержит агрегатные функции. (Строго говоря, вы можете написать предложение HAVING, не используя агрегаты, но это редко бывает полезно. То же самое условие может работать более эффективно на стадии WHERE.)

ЗАПРОСЫ КОРРЕКТИРОВКИ ДАННЫХ

INSERT INTO book (title, author,price,amount)

SELECT title,author,price,amount

FROM supply;

SELECT \* FROM book;

INSERT INTO book (title, author, price, amount)

SELECT title, author, price,amount

FROM supply

WHERE author NOT IN (SELECT author FROM book);

SELECT\* FROM book;

UPDATE book

SET price=0.7\*price

WHERE amount<5;

SELECT\* FROM book;

ALTER TABLE book

ADD buy DECIMAL(8,0);

UPDATE book

SET buy=8;

SELECT\* FROM book;

UPDATE book

SET buy=IF (buy>amount,amount,buy), price=IF (buy=0, 0.9\*price, price);

SELECT \*FROM book;

UPDATE book, supply

SET book.amount=book.amount+supply.amount, book.price=(book.price+supply.price)/2

WHERE book.title=supply.title AND book.author=supply.author;

SELECT\* FROM book;

DELETE FROM supply

WHERE author IN (SELECT author

FROM book

GROUP BY author

HAVING SUM(amount)>10);

SELECT\* FROM supply;

CREATE TABLE ordering AS

SELECT author, title , (SELECT ROUND(AVG(amount)) FROM book) AS amount

FROM book

WHERE amount<(SELECT AVG(amount)FROM book);

SELECT\*FROM ordering;

SQL оператор ALTER TABLE используется для добавления, изменения или удаления столбцов в таблице. SQL оператор ALTER TABLE также используется для переименования таблицы.

Синтаксис SQL оператора ALTER TABLE для добавления столбца в таблицу.

ALTER TABLE table\_name  
ADD column\_name column\_definition;

ALTER TABLE book

ADD buy DECIMAL(8,0);

## **CREATE TABLE**

The CREATE TABLE command creates a new table in the database.

The following SQL creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

### **Example**

CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);

## **CREATE TABLE Using Another Table**

A copy of an existing table can also be created using CREATE TABLE.

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

### **Example**

CREATE TABLE TestTable AS  
SELECT customername, contactname  
FROM customers;

## **ALTER TABLE**

The ALTER TABLE command adds, deletes, or modifies columns in a table.

The ALTER TABLE command also adds and deletes various constraints in a table.

The following SQL adds an "Email" column to the "Customers" table:

### **Example**

ALTER TABLE Customers  
ADD Email varchar(255);

The following SQL deletes the "Email" column from the "Customers" table:

### **Example**

ALTER TABLE Customers  
DROP COLUMN Email;

## **DROP TABLE**

The DROP TABLE command deletes a table in the database.

The following SQL deletes the table "Shippers":

### **Example**

DROP TABLE Shippers;

**Note:** Be careful before deleting a table. Deleting a table results in loss of all information stored in the table!

## **TRUNCATE TABLE**

The TRUNCATE TABLE command deletes the data inside a table, but not the table itself.

The following SQL truncates the table "Categories":

### **Example**

TRUNCATE TABLE Categories;

ВЛОЖЕНННЫЕ ЗАПРОСЫ

SELECT title, author, price

FROM book

WHERE price<=(SELECT AVG(price)

FROM book)

ORDER BY price DESC; // без ;

SELECT title, author, price

FROM book

WHERE price-(SELECT MIN(price) FROM book)<=150

ORDER BY price;

SELECT author, title, amount

FROM book

WHERE amount IN (SELECT amount

FROM book

GROUP BY amount

HAVING COUNT(amount)=1);

SELECT title, author, amount, price

FROM book

WHERE amount < ALL (

SELECT AVG(amount)

FROM book

GROUP BY author

);

SELECT author,title, price

FROM book

WHERE price<ANY(SELECT MIN(price)

FROM book

GROUP BY author

);

SELECT title, author, amount,

(SELECT MAX(amount) FROM book)-amount AS Заказ

FROM book

WHERE (SELECT MAX(amount) FROM book)-amount>0; //правильное закрытие скобки после FROM

CREATE TABLE table\_name //создать пустую таблицу

CREATE TABLE table\_name (

column\_name\_1 column\_type\_1,

column\_name\_2 column\_type\_2, ...,

column\_name\_N column\_type\_N, );

CREATE TABLE supply(supply\_id INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(50),

author VARCHAR(30),

price DECIMAL(8,2),

amount INT);

CREATE TABLE IF NOT EXISTS Staff //Если мы вызовем команду CREATE TABLE Staff, а таблица Staff уже есть в базе, команда выдаст ошибку. Поэтому перед созданием разумно проверить, содержит ли уже база таблицу Staff.

INSERT INTO имя\_таблицы(столбец\_1, столбец\_2, ..., столбец\_N)

VALUES

(значение\_1\_1, значение\_1\_2, ..., значение\_1\_N),

(значение\_2\_1, значение\_2\_2, ..., значение\_2\_N),

...

(значение\_M\_1, значение\_M\_2, ..., значение\_M\_N);

Например, чтобы добавить в таблицу **book**две новые записи используется запрос:

INSERT INTO book (title, author, price, amount)

VALUES

('Война и мир','Толстой Л.Н.', 1070.20, 2),

('Анна Каренина', 'Толстой Л.Н.', 599.90, 3);

Вывести из таблицы **trip** информацию о командировках тех сотрудников, фамилия которых заканчивается на букву «а», в отсортированном по убыванию даты последнего дня командировки виде. В результат включить столбцы **name, city, per\_diem, date\_first, date\_last.**

SELECT name,city,per\_diem,date\_first, date\_last

FROM trip

WHERE name LIKE '%а %' // есть пробел (в этом поле есть пробел)

ORDER BY date\_last DESC

Вывести в алфавитном порядке фамилии и инициалы тех сотрудников, которые были в командировке в Москве.

SELECT DISTINCT name

FROM trip

WHERE city="Москва" // не забыть кавычки

ORDER BY name

Для каждого города посчитать, сколько раз сотрудники в нем были.  Информацию вывести в отсортированном в алфавитном порядке по названию городов. Вычисляемый столбец назвать **Количество**.

SELECT city,COUNT(city) AS Количество

FROM trip

GROUP BY city //чтобы посчитать , группируем по городам

ORDER BY city

SELECT author, SUM(amount)  
FROM book  
GROUP BY author

Для ограничения вывода записей в SQL используется оператор LIMIT , после которого указывается количество строк.  Результирующая таблица будет иметь количество строк не более указанного после LIMIT. LIMIT размещается после раздела ORDER BY.

Как правило, этот оператор используется, чтобы отобрать заданное количество отсортированных строк результата запроса.

Вывести два города, в которых чаще всего были в командировках сотрудники. Вычисляемый столбец назвать **Количество**.

SELECT city, COUNT(city) AS Количество

FROM trip

GROUP BY city

ORDER BY COUNT(city) DESC

LIMIT 2

Для вычитания двух дат используется функция **DATEDIFF(дата\_1, дата\_2)**, результатом которой является количество дней между **дата\_1** и **дата\_2**. Например,

DATEDIFF('2020-04-01', '2020-03-28')=4

DATEDIFF('2020-05-09','2020-05-01')=8

DATEDIFF(date\_last, date\_first)

Вывести информацию о командировках во все города кроме Москвы и Санкт-Петербурга (фамилии и инициалы сотрудников, город , длительность командировки в днях, при этом первый и последний день относится к периоду командировки). Последний столбец назвать **Длительность**. Информацию вывести в упорядоченном по убыванию длительности поездки, а потом по убыванию названий городов (в обратном алфавитном порядке).

SELECT name, city,DATEDIFF(date\_last,date\_first)+1 AS Длительность //+1 чтобы включить дату; первая дата больше второй, из первой вычитается вторая

FROM trip

WHERE city NOT IN("Москва","Санкт-Петербург")

ORDER BY Длительность DESC, city DESC //подряд

Вывести информацию о командировках сотрудника(ов), которые были самыми короткими по времени. В результат включить столбцы **name**, **city**, **date\_first**, **date\_last**.

SELECT name, city, date\_first, date\_last

FROM trip

WHERE DATEDIFF(date\_last,date\_first)=(SELECT MIN(DATEDIFF(date\_last,date\_first))FROM trip)

Для того, чтобы выделить номер месяца из даты используется функция MONTH(дата).

Например, MONTH('2020-04-12') = 4.

Если определяется месяц для  значений столбца date\_first, то используется запись MONTH(date\_first)

SELECT name,city,date\_first,date\_last

FROM trip

WHERE MONTH(date\_last)=MONTH(date\_first)

ORDER BY city,name

1. Для того, чтобы выделить название месяца из даты используется функция MONTHNAME(дата), которая возвращает название месяца на английском языке для указанной даты. Например, MONTHNAME('2020-04-12')='April'.
2. Если группировка осуществляется по вычисляемому столбцу (в данном случае «вычисляется» название месяца), то после GROUP BYможно указать как вычисляемое выражение, так и имя столбца, заданное с помощью AS. Важно отметить, что последний вариант (указать имя столбца)  нарушает стандарт по порядку выполнения запросов, но иногда может встречаться на реальных платформах.

Вывести название месяца и количество командировок для каждого месяца. Считаем, что командировка относится к некоторому месяцу, если она началась в этом месяце. Информацию вывести сначала в отсортированном по убыванию количества, а потом в алфавитном порядке по названию месяца виде. Название столбцов – Месяц и Количество.

SELECT MONTHNAME(date\_first) AS Месяц, COUNT(MONTHNAME(date\_first)) AS Количество

FROM trip

GROUP BY MONTHNAME(date\_first)

ORDER BY Количество DESC,MONTHNAME(date\_first) // date\_first без кавычек

1 В SQL есть функции, которые позволяют выделить часть даты: день(DAY()), месяц (MONTH()), год(YEAR()) . Например:

DAY('2020-02-01') = 1

MONTH('2020-02-01') = 2

YEAR('2020-02-01') = 2020

2. Количество дней командировки вычисляется как разница между датами последнего и первого дня командировки плюс 1.

СОЗДАТЬ ТАБЛИЦУ

CREATE TABLE fine(fine\_id INT PRIMARY KEY AUTO\_INCREMENT, // ключевой столбец целого типа с а втоматическим увеличением значения ключа на 1

name VARCHAR(30),

number\_plate VARCHAR(6),

violation VARCHAR(50),

sum\_fine DECIMAL(8,2),

date\_violation DATE,

date\_payment DATE); // столбцы и тип данных через пробел, разные столбцы через запятую

 В таблицу **fine** первые 5 строк уже занесены. Добавить в таблицу записи с ключевыми значениями 6, 7, 8.

INSERT INTO fine (name,number\_plate,violation,sum\_fine,date\_violation,date\_payment)

VALUES("Баранов П.Е.","Р523ВТ","Превышение скорости(от 40 до 60)",Null,"2020-02-14",Null),

("Абрамова К.А.","О111АВ","Проезд на запрещающий сигнал",Null,"2020-02-23",Null),

("Яковлев Г.Р.","Т330ТТ","Проезд на запрещающий сигнал",Null,"2020-03-03",Null)

//строки в скобках через запятую, пустое поле Null, дата в кавычках тоже

Тип**DATE** – позволяет описать дату в формате ГГГГ-ММ-ДД, например, 2020-02-02. При вставке данных в таблицу с помощью INSERT INTO ... VALUES значение даты заключается в кавычки.

Чтобы не писать название таблицы каждый раз, удобно использовать алиасы.

Алиас, это псевдоним, который мы присваивали столбцам после ключевого слова **AS**([шаг](https://stepik.org/lesson/297509/step/4?unit=279269)).  Алиасы так же можно использовать и для таблиц. Это становится актуальным, при увеличении числа используемых таблиц, их иногда может быть и 5 и 10 и более. Псевдонимы помогают сделать запрос чище и читабельнее.

Для присваивания псевдонима существует 2 варианта:

* с использованием ключевого слова **AS**

FROM fine AS f, traffic\_violation AS tv

* а так же и без него

FROM fine f, traffic\_violation tv

После присвоения таблице алиаса, он используется во всех разделах запроса, в котором алиас задан. Например:

WHERE f.violation = tv.violation

Занести в таблицу fine суммы штрафов, которые должен оплатить водитель, в соответствии с данными из таблицы traffic\_violation. При этом суммы заносить только в пустые поля столбца  sum\_fine.

Таблица traffic\_violation создана и заполнена.

Занести в таблицу **fine** суммы штрафов, которые должен оплатить водитель, в соответствии с данными из таблицы **traffic\_violation**. При этом суммы заносить только в пустые поля столбца **sum\_fine**.

**Важно!** Сравнение значения столбца с пустым значением осуществляется с помощью оператора **IS NULL**.

1. После ключевого слова UPDATE кроме обновляемой таблицы**fine** укажите таблицу **traffic\_violation,**для того чтобы запрос видел таблицы источники.  Сначала перечисляем **все источники,** потом выполняем необходимые действия.

2.Обновляйте только те записи таблицы**fine**, у которых значение столбца **violation**совпадает со значением соответствующего столбца таблицы **traffic\_violation**, а также значение столбца **sum\_fine** пусто.

UPDATE fine AS f, traffic\_violation AS t

SET f.sum\_fine=t.sum\_fine

WHERE f.violation=t.violation AND f.sum\_fine IS NULL;

SELECT\* FROM fine

## Группировка данных по нескольким столбцам

В разделе GROUP BY можно указывать несколько столбцов, разделяя их запятыми. Тогда к одной группе будут относиться записи, у которых значения столбцов, входящих в группу, равны. Рассмотрим группировку по нескольким столбцам на примере следующего запроса:

SELECT name, number\_plate, violation, count(\*)

FROM fine

GROUP BY name, number\_plate, violation;

Вывести фамилию, номер машины и нарушение только для тех водителей, которые на одной машине нарушили одно и то же правило   два и более раз. При этом учитывать все нарушения, независимо от того оплачены они или нет. Информацию отсортировать в алфавитном порядке, сначала по фамилии водителя, потом по номеру машины и, наконец, по нарушению.

SELECT name,number\_plate,violation

FROM fine

GROUP BY name,number\_plate,violation // или HAVING count(\*)>=2 одно и то же правило >=2

HAVING count(number\_plate)>=2 // можно сразу >=

ORDER BY name, number\_plate,violation

СОЗДАТЬ ТАБЛИЦУ С ДВУМЯ СТОЛБЦАМИ

|  |  |
| --- | --- |
| **Поле** | **Тип, описание** |
| author\_id | INT PRIMARY KEY AUTO\_INCREMENT |
| name\_author | VARCHAR(50) |

CREATE TABLE author(author\_id INT PRIMARY KEY AUTO\_INCREMENT, name\_author VARCHAR(50))

Заполнить таблицу author. В нее включить следующих авторов:

* Булгаков М.А.
* Достоевский Ф.М.
* Есенин С.А.
* Пастернак Б.Л.

INSERT INTO author(name\_author)

VALUES('Булгаков М.А.'),('Достоевский Ф.М.'),('Есенин С.А.'),('Пастернак Б.Л.'); // строки одно столбца в скобках через запятую)

SELECT\* FROM author

INSERT INTO имя\_таблицы(столбец\_1, столбец\_2, ..., столбец\_N) VALUES (значение\_1\_1, значение\_1\_2, ..., значение\_1\_N), (значение\_2\_1, значение\_2\_2, ..., значение\_2\_N), ... (значение\_M\_1, значение\_M\_2, ..., значение\_M\_N);

**Создание таблицы с внешними ключами**

При создании зависимой таблицы (таблицы, которая содержит внешние ключи) необходимо учитывать, что :

* каждый внешний ключ должен иметь такой же тип данных, как связанное поле главной таблицы (в наших примерах это INT);
* необходимо указать главную для нее таблицу и столбец, по которому осуществляется связь:

FOREIGN KEY (связанное\_поле\_зависимой\_таблицы)

REFERENCES главная\_таблица (связанное\_поле\_главной\_таблицы)

По умолчанию любой столбец, кроме ключевого, может содержать значение NULL. При создании таблицы это можно переопределить,  используя  ограничение NOT NULL для этого столбца:

CREATE TABLE таблица (

    столбец\_1 INT NOT NULL,

    столбец\_2 VARCHAR(10)

);

CREATE TABLE book ( book\_id INT PRIMARY KEY AUTO\_INCREMENT, title VARCHAR(50), author\_id INT NOT NULL, price DECIMAL(8,2), amount INT, FOREIGN KEY (author\_id) REFERENCES author (author\_id) );

Внешний ключ – поле таблицы, предназначенное для хранения значения первичного ключа другой таблицы с целью организации связи между этими таблицами.

Перепишите запрос на создание таблицы **book** , чтобы ее структура соответствовала структуре, показанной на логической схеме (таблица **genre** уже создана, порядок следования столбцов - как на логической схеме в таблице **book**, **genre\_id**  - внешний ключ) . Для **genre\_id** ограничение о недопустимости пустых значений **не задавать**.



CREATE TABLE book(

book\_id INT PRIMARY KEY AUTO\_INCREMENT, title VARCHAR(50),author\_id INT NOT NULL, genre\_id INT, price DECIMAL(8,2),amount INT,FOREIGN KEY (author\_id) REFERENCES author(author\_id),FOREIGN KEY (genre\_id) REFERENCES genre(genre\_id)

);

## **Действия при удалении записи главной таблицы**

С помощью выраженияON DELETE можно установить действия, которые выполняются для записей подчиненной таблицы при удалении связанной строки из главной таблицы. При удалении можно установить следующие опции:

* CASCADE: автоматически удаляет строки из зависимой таблицы при удалении  связанных строк в главной таблице.
* SET NULL: при удалении  связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение NULL. (В этом случае столбец внешнего ключа должен поддерживать установку**NULL**).
* SET DEFAULT похоже на SET NULL за тем исключением, что значение  внешнего ключа устанавливается не в NULL, а в значение по умолчанию для данного столбца.
* RESTRICT: отклоняет удаление строк в главной таблице при наличии связанных строк в зависимой таблице.

**Важно!**Если для столбца установлена опция SET NULL, то при его описании нельзя задать ограничение на пустое значение.

**Пример**

Будем считать, что при удалении автора из таблицы **author**, необходимо удалить все записи о книгах из таблицы **book**, написанные этим автором. Данное действие необходимо прописать при создании таблицы.

CREATE TABLE book (book\_id INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(50), author\_id INT NOT NULL, price DECIMAL(8,2), amount INT, FOREIGN KEY (author\_id) REFERENCES author (author\_id) ON DELETE CASCADE );

Создать таблицу **book** той же структуры, что и на предыдущем шаге. Будем считать, что при удалении автора из таблицы **author**, должны удаляться все записи о книгах из таблицы **book**, написанные этим автором. А при удалении жанра из таблицы **genre** для соответствующей записи **book** установить значение **Null**в столбце **genre\_id**.

CREATE TABLE book(

book\_id INT PRIMARY KEY AUTO\_INCREMENT, title VARCHAR(50),author\_id INT NOT NULL, genre\_id INT, price DECIMAL(8,2),amount INT,FOREIGN KEY (author\_id) REFERENCES author(author\_id) ON DELETE CASCADE,FOREIGN KEY (genre\_id) REFERENCES genre(genre\_id) ON DELETE SET NULL

);

Добавьте три последние записи (с ключевыми значениями 6, 7, 8) в таблицу book, первые 5 записей уже добавлены:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **book\_id** | **title** | **author\_id** | **genre\_id** | **price** | **amount** |
| 1 | Мастер и Маргарита | 1 | 1 | 670.99 | 3 |
| 2 | Белая гвардия | 1 | 1 | 540.50 | 5 |
| 3 | Идиот | 2 | 1 | 460.00 | 10 |
| 4 | Братья Карамазовы | 2 | 1 | 799.01 | 3 |
| 5 | Игрок | 2 | 1 | 480.50 | 10 |
| 6 | Стихотворения и поэмы | 3 | 2 | 650.00 | 15 |
| 7 | Черный человек | 3 | 2 | 570.20 | 6 |
| 8 | Лирика | 4 | 2 | 518.99 | 2 |
|  |  |  |  |  |  |

INSERT INTO book(title,author\_id,genre\_id,price,amount)

VALUES("Стихотворения и поэмы",3,2,650.00,15),("Черный человек",3,2,570.20,6),("Лирика",4,2,518.99,2);

//каждая запись(строка) -это в скобках, строки через запятую

<https://stepik.org/course/63054/info>

The INNER JOIN keyword selects records that have matching values in both tables.

### **INNER JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;

Объединяет записи из двух таблиц, если в связующих полях этих таблиц содержатся одинаковые значения.

Вывести название, жанр и цену тех книг, количество которых больше 8, в отсортированном по убыванию цены виде.

SELECT title,name\_genre,price

FROM

genre INNER JOIN book

ON genre.genre\_id = book.genre\_id

WHERE amount>8

ORDER BY price DESC;

## **Внешнее соединение LEFT и RIGHT OUTER JOIN**

Оператор внешнего соединения LEFT OUTER JOIN  (можно использовать LEFT JOIN) соединяет две таблицы. Порядок таблиц для оператора важен, поскольку оператор не является симметричным.

SELECT

...

FROM

таблица\_1 LEFT JOIN таблица\_2

ON условие

...

Результат запроса формируется так:

1. в результат включается внутреннее соединение (INNER JOIN) первой и второй таблицы в соответствии с условием;
2. затем в результат добавляются те записи первой таблицы, которые не вошли во внутреннее соединение на шаге 1, для таких записей соответствующие поля второй таблицы заполняются значениями NULL.

Соединение RIGHT JOIN действует аналогично, только в пункте 2 первая таблица меняется на вторую и наоборот.

Вывести название всех книг каждого автора, если книг некоторых авторов в данный момент нет на складе – вместо названия книги указать Null.

SELECT name\_author, title

FROM

author LEFT JOIN book

ON author.author\_id = book.author\_id

ORDER BY name\_author;

Вывести все жанры, которые не представлены в книгах на складе.

SELECT name\_genre

FROM genre LEFT JOIN book

ON genre.genre\_id=book.genre\_id

WHERE title IS NULL

Необходимо в каждом городе провести выставку книг каждого автора в течение 2020 года. Дату проведения выставки выбрать случайным образом. Создать запрос, который выведет город, автора и дату проведения выставки. Последний столбец назвать **Дата**. Информацию вывести, отсортировав сначала в алфавитном порядке по названиям городов, а потом по убыванию дат проведения выставок.

SELECT name\_city, name\_author, DATE\_ADD('2020-01-01',INTERVAL FLOOR(RAND() \* 365) DAY) AS Дата

FROM author CROSS JOIN city

ORDER BY name\_city, Дата DESC

**Запросы на выборку из нескольких таблиц**

Запрос на выборку может выбирать данные из двух и более таблиц базы данных. При этом таблицы должны быть логически связаны между собой. Для каждой пары таблиц, включаемых в запрос, необходимо указать свой оператор соединения. Наиболее распространенным является внутреннее соединение INNER JOIN, поэтому в примерах будем использовать его.

Вывести информацию о тех книгах, их авторах и жанрах, цена которых принадлежит интервалу от 500  до 700 рублей  включительно.

SELECT title, name\_author, name\_genre, price, amount

FROM author INNER JOIN book ON author.author\_id = book.author\_id

INNER JOIN genre ON genre.genre\_id = book.genre\_id

WHERE price BETWEEN 500 AND 700;

Вывести информацию о книгах (жанр, книга, автор), относящихся к жанру, включающему слово «роман» в отсортированном по названиям книг виде.

SELECT name\_genre, title, name\_author

FROM book

INNER JOIN genre ON book.genre\_id=genre.genre\_id

INNER JOIN author ON book.author\_id=author.author\_id

WHERE name\_genre LIKE "%роман%"

ORDER BY title

## **Запросы для нескольких таблиц с группировкой**

В запросах с групповыми функциями могут использоваться несколько таблиц, между которыми используются различные типы соединений.

**Пример**

Вывести количество различных книг каждого автора. Информацию отсортировать в алфавитном порядке по фамилиям  авторов.

SELECT title, author, COUNT(amount) AS количество

FROM book INNER JOIN author ON author.author\_id=book.author\_id

GROUP BY author

ORDER BY name

SELECT name\_author, count(title) AS Количество

FROM author INNER JOIN book on author.author\_id = book.author\_id GROUP BY name\_author ORDER BY name\_author;

При использовании соединения INNER JOIN мы не можем узнать, что книг Лермонтова на складе нет, но предполагается, что они могут быть. Чтобы автор Лермонтов был включен в результат, нужно изменить соединение таблиц.

SELECT name\_author, count(title) AS Количество

FROM

author LEFT JOIN book

on author.author\_id = book.author\_id

GROUP BY name\_author

ORDER BY name\_author;

Посчитать количество экземпляров  книг каждого автора из таблицы **author**.  Вывести тех авторов,  количество книг которых меньше 10, в отсортированном по возрастанию количества виде. Последний столбец назвать **Количество**.

SELECT name\_author,SUM(amount) AS Количество

FROM author LEFT JOIN book

ON author.author\_id=book.author\_id

GROUP BY name\_author

HAVING SUM(amount)<10 OR SUM(amount) IS NULL

ORDER BY SUM(amount);

// HAVING /\*пишем условия отбора в результирующий запрос после группировки: "количество книг которых меньше 10". при этом NULL (пустота) не равно 0 и пустоту нельзя сравнить с числом при помощи оператора сравнения, а вывести данные строчки нужно. Поэтому в условие отбора необходимо добавить  OR SUM(amount) is NULL