

SQL-Урок 1. Установка SQLite Studio. Выгрузка БД больших объёмов данных.

Язык SQL. Основные понятия.

SQLiteStudio - это менеджер баз данных SQLite со следующими функциями:

- Портативный - не нужно устанавливать или удалять. Просто скачайте, распакуйте и запустите.
- Интуитивно понятный интерфейс ,
- Мощный , но легкий и быстрый ,
- Все функции SQLite3 и SQLite2, заключенные в простой графический интерфейс ,
- Кроссплатформенность - работает на Windows 9x / 2k / XP / 2003 / Vista / 7, Linux, MacOS X и должна работать на других Unixes (еще не проверенных).
- Экспорт в различные форматы (операторы SQL, CSV, HTML, XML, PDF, JSON),
- Импорт данных из различных форматов (CSV, пользовательские текстовые файлы [регулярные выражения]),
- Многочисленные небольшие дополнения, такие как форматирование кода , история запросов, выполняемых в окнах редактора, проверка синтаксиса на лету и многое другое,
- Поддержка Unicode ,
- Skinnable (интерфейс может выглядеть как родной для Windows 9x / XP, KDE, GTK, Mac OS X или рисовать виджеты для других сред, WindowMaker и т. Д.),
- Настраиваемые цвета, шрифты и ярлыки.
- Открытый и бесплатный - выпущен под лицензией GPLv3 .

Для того, чтобы начать изучать **SQL** нам нужно сначала понять, что такое база данных.

1. Что такое База Данных

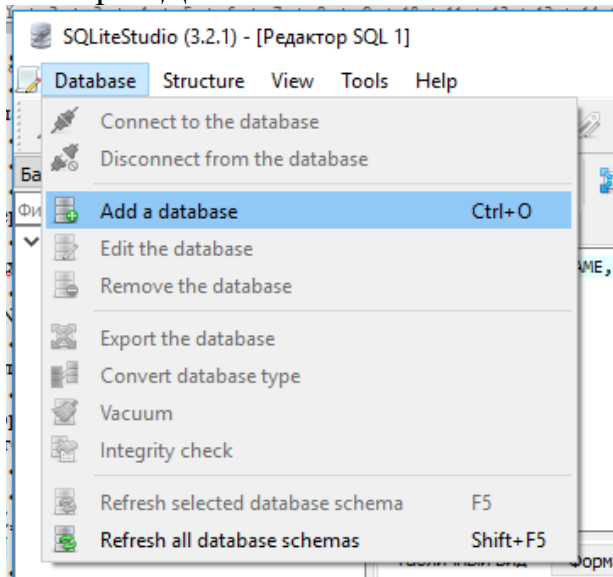
База данных (БД) - упорядоченный набор логически взаимосвязанных данных, используемых совместно, и которые хранятся в одном месте. Если коротко, то простейшая БД это обычная таблица со строками и столбцами в которой хранится разного рода информация (примером может служить таблица в **Excel**). Так, часто, с БД нераздельно связывают **Системы управления базами данных (СУБД)**, которые предоставляют функционал для работы с БД. Язык **SQL** как раз и является частью СУБД, которая осуществляет управление информацией в БД. Мы будем считать БД набором обычных таблиц, которые хранятся в отдельных файлах.

2. Что такое SQL

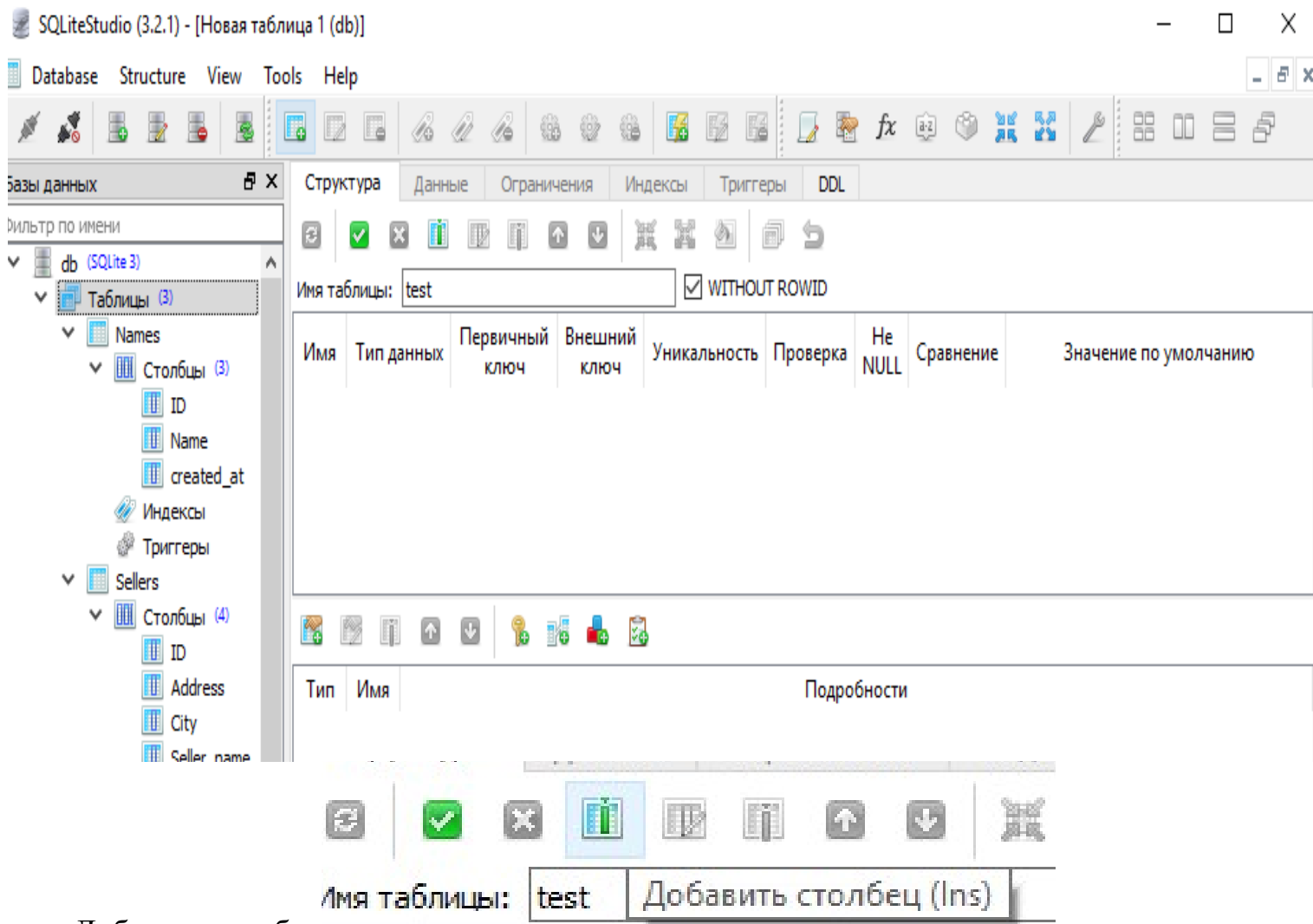
Итак, переходим к **SQL**.

SQL - простой язык программирования, который имеет немного команд и которой может научиться любой желающий. Расшифровывается как **Structured Query Language** - язык структурированных запросов, который был разработан для работы с БД, а именно, чтобы получать /добавлять /изменять данные, иметь возможность обрабатывать большие массивы информации и быстро получать структурированную и сгруппированную информацию. Есть много вариантов языка **SQL**, но у них всех основные команды почти одинаковы. Также существует и много СУБД, но основными из них являются: **Microsoft Access, Microsoft SQL Server, MySQL, Oracle SQL, IBM DB2 SQL, PostgreSQL, MySQL, SQLite**. Чтобы работать с **SQL** кодом, нам понадобится одна из вышеперечисленных СУБД. Для обучения мы будем использовать СУБД **SQLite** с помощью **SQLiteStudio**

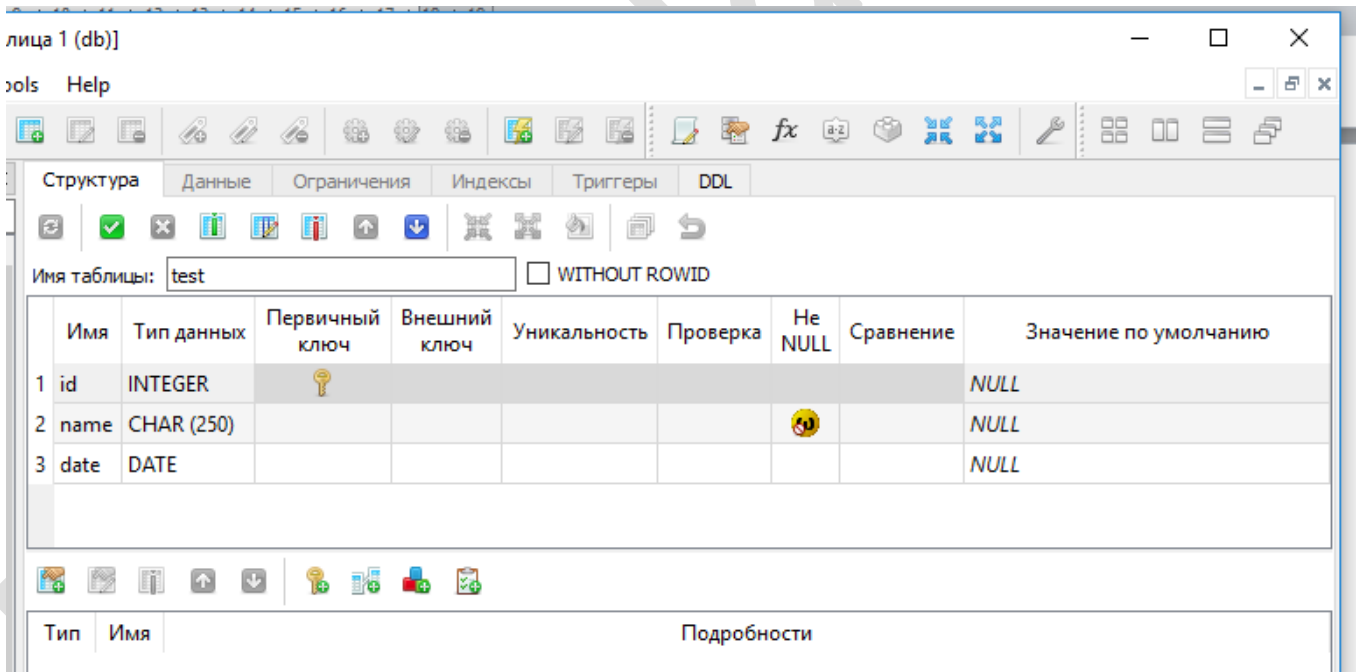
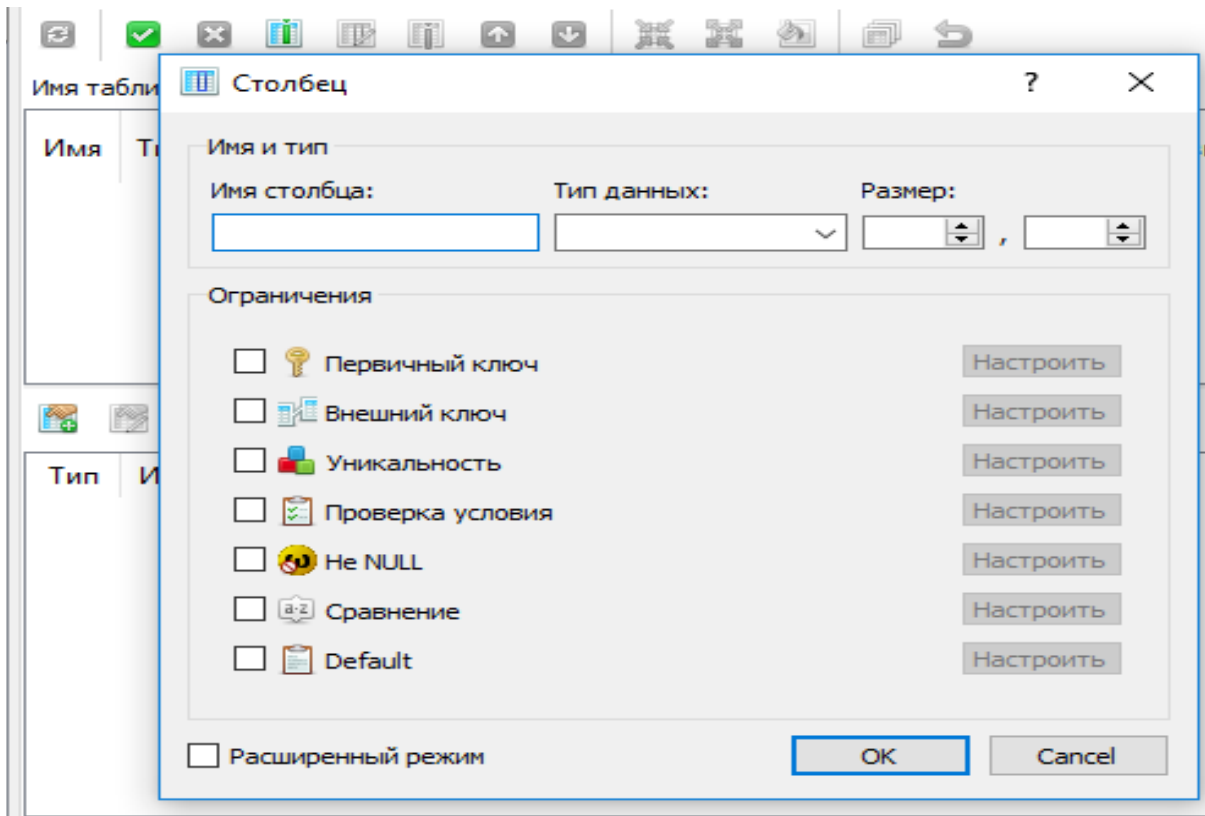
Импорт БД



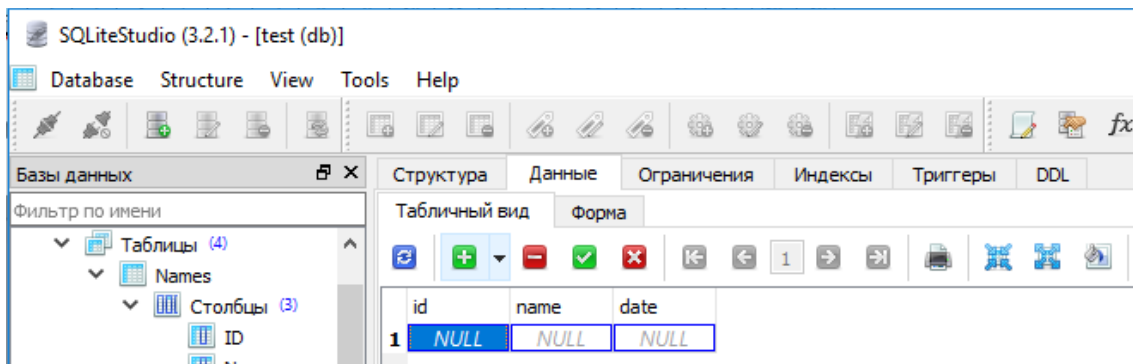
Добавить таблицу



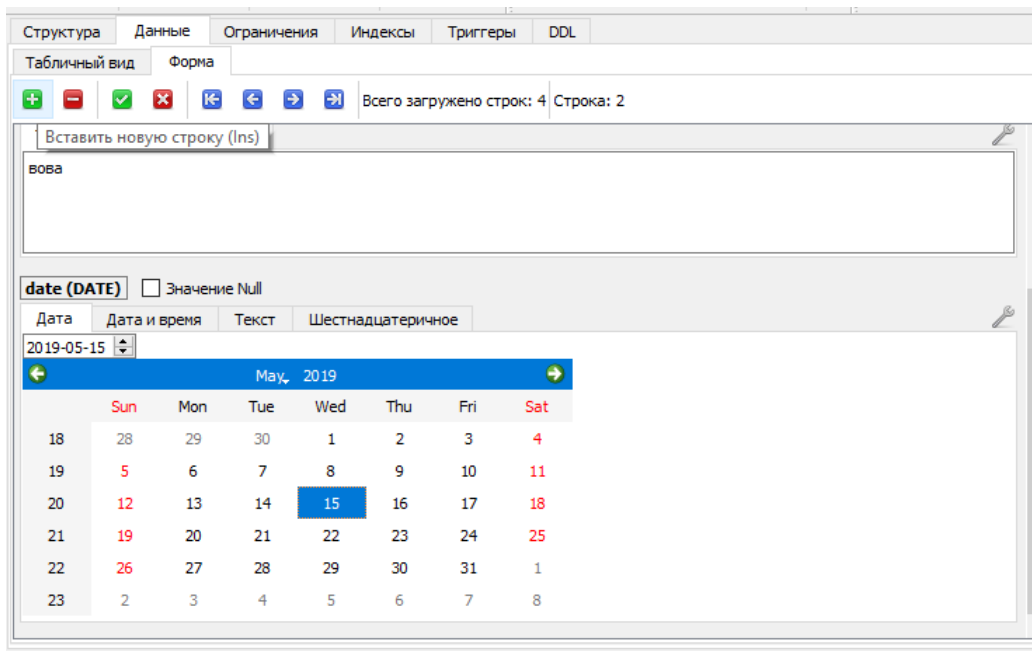
Добавить столбец



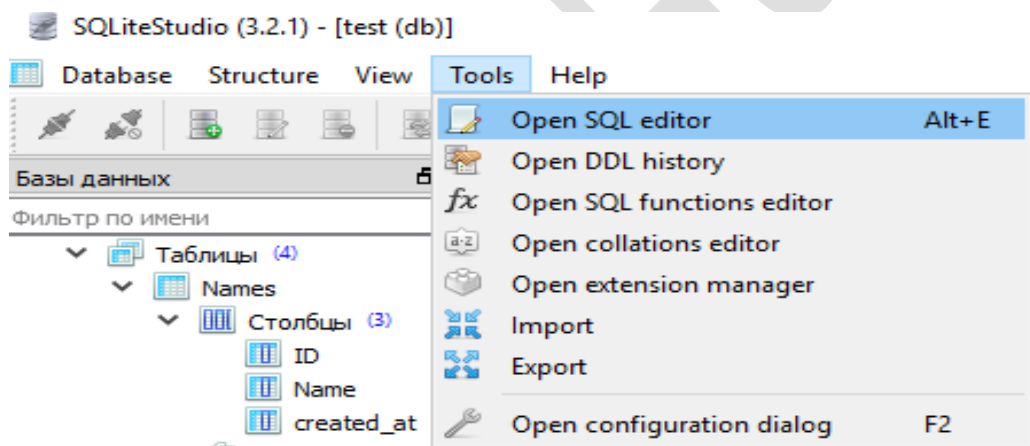
Заполнить таблицу данными перейдите на вкладку данные и нажмите +



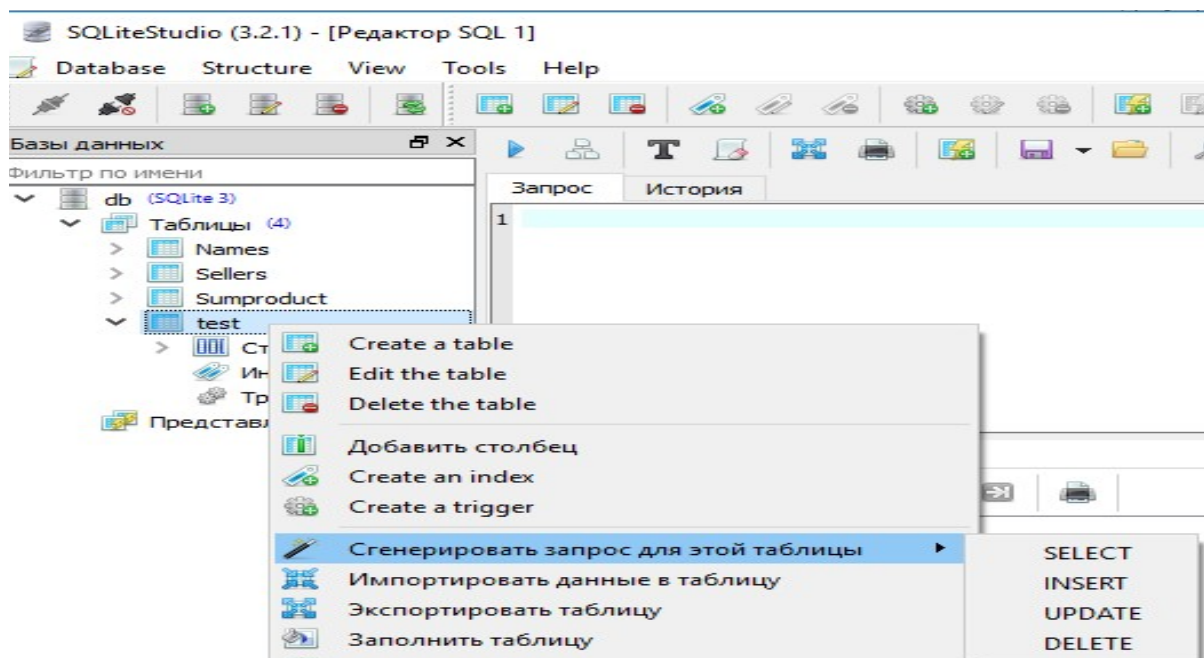
Перейдите во вкладку форма и заполните поля с помощью формы



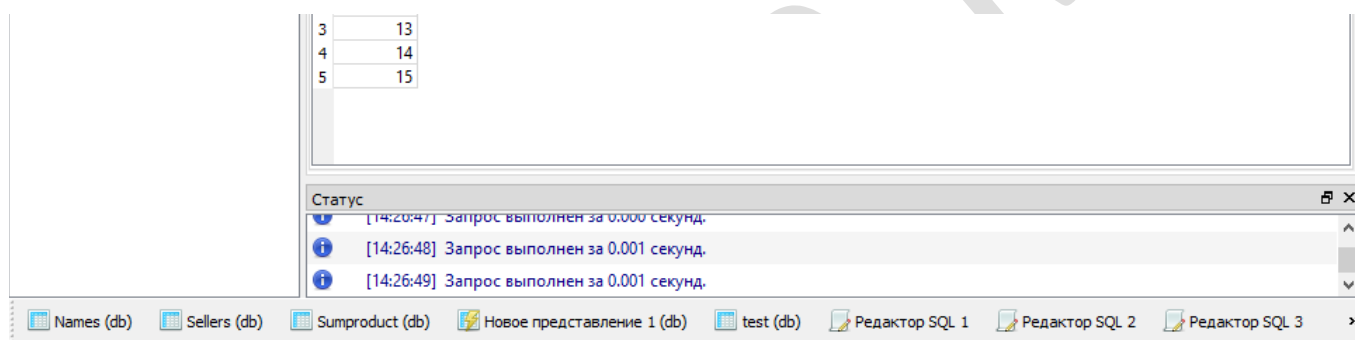
Чтобы вызвать SQL редактор нажмите Tools=> open SQL editor



Автогенератор SQL кода



Переход между SQL и таблицами внизу окна



SQL как и другие языки программирования имеет свои команды (операторы), с помощью которых отдаются инструкции для выборки данных. Чтобы рассмотреть как работают операторы SQL, мы будем использовать мнимую БД с информацией о реализованной продукции по городам и месяцам:

	ID	Month	Product	City	Quantity	Amount
1	1	Январь	Шоколадные конфеты	Горловка	216	924.12
2	2	Январь	Молоко	Горловка	915	747.67
3	3	Январь	Шоколад	Горловка	249	368.94
4	4	Январь	Соль	Горловка	574	473.52
5	5	Январь	Сахар	Горловка	665	632.56
6	6	Январь	Йогурт	Горловка	380	257.47
7	7	Январь	Помидоры	Горловка	132	298.8
8	8	Январь	Рыба	Горловка	913	455.44
9	9	Январь	Бананы	Горловка	251	671.99
10	10	Январь	Мандарины	Горловка	353	653.7
11	11	Январь	Сыр	Горловка	517	562.3
12	12	Январь	Чай	Горловка	607	667.9
13	13	Январь	Сметана	Горловка	74	598.26
14	14	Январь	Огурцы	Горловка	756	605.39

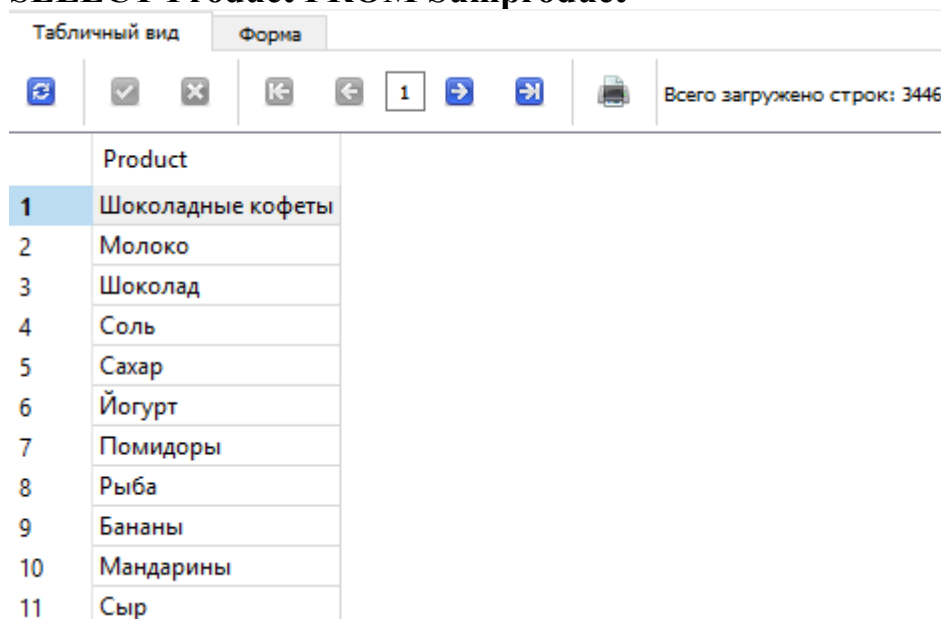
Задание: изучить основные элементы управления. Создать 2 таблицы, наполнить по 5 полей тестовыми данными, создать одну из таблиц с помощью формы. Изучит основные переходы между вкладками.

SQL-Урок 2. Выборка данных (SELECT)

Самым первым и главным оператором в SQL является *SELECT*. С его помощью мы можем отбирать необходимые нам поля данных в таблице.

1. Выборка отдельных полей.

SELECT Product FROM Sumproduct



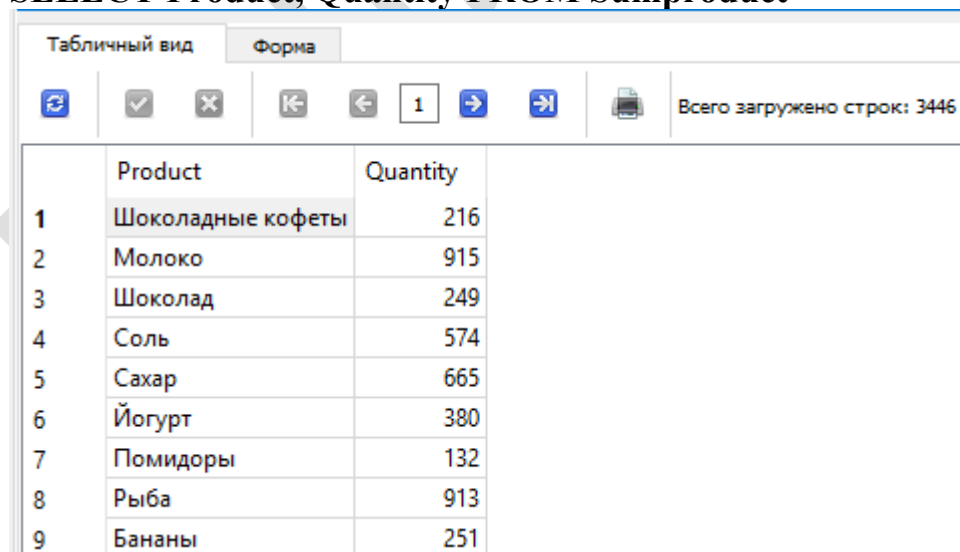
	Product
1	Шоколадные кофеты
2	Молоко
3	Шоколад
4	Соль
5	Сахар
6	Йогурт
7	Помидоры
8	Рыба
9	Бананы
10	Мандарины
11	Сыр

Видим, что наш SQL запрос отобрал колонку **Product** из таблицы **Sumproduct**.

2. Выборка нескольких полей.

Допустим, нам необходимо выбрать название и количество реализованного товара. Для этого просто перечисляем необходимые поля через запятую:

SELECT Product, Quantity FROM Sumproduct



	Product	Quantity
1	Шоколадные кофеты	216
2	Молоко	915
3	Шоколад	249
4	Соль	574
5	Сахар	665
6	Йогурт	380
7	Помидоры	132
8	Рыба	913
9	Бананы	251

3. Выборка всех столбцов.

Если же нам необходимо получить всю таблицу со всеми полями, тогда просто ставим знак звездочка (*):

SELECT * FROM Sumproduct

	ID	Month	Product	City	Quantity	Amount
1	1	Январь	Шоколадные кофеты	Горловка	216	924.12
2	2	Январь	Молоко	Горловка	915	747.67
3	3	Январь	Шоколад	Горловка	249	368.94
4	4	Январь	Соль	Горловка	574	473.52
5	5	Январь	Сахар	Горловка	665	632.56
6	6	Январь	Йогурт	Горловка	380	257.47
7	7	Январь	Помидоры	Горловка	132	298.8
8	8	Январь	Рыба	Горловка	913	455.44
9	9	Январь	Бананы	Горловка	251	671.99

Все операторы в **SQL** нечувствительны к регистру, поэтому вы можете писать как большими буквами, так и маленькими (как правило, их принято писать большими буквами, чтобы различать от названий полей и таблиц). Названия же таблиц и полей является наоборот чувствительными к регистру и должны писаться точно как в БД.

Задание:

1. Выберите все записи из **Authors**
2. Выберите все записи из **Books**
3. Выберите все записи из **Publishers**
4. Выберите поле **name** из **Authors**
5. Выберите поля в следующем порядке **name, id** из **Publishers**
6. Выберите поля в следующем порядке: Дата публикации, Название, Описание, Цена из **Books**

SQL-Урок 3. Сортировка (ORDER BY)

В будущем нам может понадобиться сортировать нашу выборку - в алфавитном порядке для текста или по возрастанию/убыванию - для цифровых значений. Для таких целей в **SQL** есть специальный оператор **ORDER BY**.

1. Сортировка выбранных данных.

Давайте всю нашу таблицу отсортируем по сумме реализации продукции, а именно по столбцу **Amount**.

```
SELECT * FROM Sumproduct ORDER BY Amount
```

	ID	Month	Product	City	Quantity	Amount
1	2158	Август	Мороженое	Шахтёрск	665	100.58
2	296	Февраль	Апельсины	Макеевка	211	100.59
3	110	Январь	Бананы	Донецк	850	101.05
4	42	Январь	Укроп	Кировское	910	101.14
5	3081	Ноябрь	Кофе	Ясиноватая	863	101.15
6	190	Январь	Яйца	Харцызск	61	101.37
7	1479	Июнь	Шоколад	Снежное	572	101.98
8	3213	Декабрь	Сахар	Иловайск	395	102.13

Видим, что запрос отсортировал записи по возрастанию в поле **Amount**. Обязательно нужно соблюдать последовательность расположения операторов, т.е. оператор *ORDER BY* должен идти в самом конце запроса. В противном случае будет получено сообщение об ошибке.

Также особенностью оператора *ORDER BY* является то, что он может сортировать данные по полю, которого мы не выбирали в запросе, то есть достаточно, чтобы оно вообще было в БД.

2. Сортировка по нескольким полям.

Теперь отсортируем наш пример дополнительно за еще одним полем. Пусть это будет поле **Product**, которое отображает место реализации продукции.

SELECT * FROM Sumproduct ORDER BY Product, Amount

	ID	Month	Product	City	Quantity	Amount
1	296	Февраль	Апельсины	Макеевка	211	100.59
2	1603	Июнь	Апельсины	Иловайск	106	104.1
3	2472	Сентябрь	Апельсины	Кировское	540	127.29
4	1557	Июнь	Апельсины	Енакиево	427	132.02
5	2601	Октябрь	Апельсины	Енакиево	345	160.96
6	2789	Октябрь	Апельсины	Ясиноватая	908	168.59
7	2268	Август	Апельсины	Углегорск	648	173.41
8	592	Март	Апельсины	Ясиноватая	1	175.02
9	3163	Ноябрь	Апельсины	Ждановка	987	180.83

Очередность сортировки будет зависеть от порядка расположения полей в запросе. То есть, в нашем случае сначала данные будут рассортированы по колонке **Product**, а затем по **Amount**

3. Направление сортировки.

Несмотря на то, что по умолчанию оператор *ORDER BY* сортирует по возрастанию, мы можем также прописать сортировки значений по убыванию. Для этого в конце каждого поля проставляем оператор *DESC* (что является сокращением от слова *DESCENDING*).

SELECT * FROM Sumproduct ORDER BY City DESC, Amount

	ID	Month	Product	City	Quantity	Amount
1	3081	Ноябрь	Кофе	Ясиноватая	863	101.15
2	1403	Июнь	Мандарины	Ясиноватая	127	107.32
3	515	Февраль	Сливочное масло	Ясиноватая	969	108.19
4	935	Апрель	Яблоки	Ясиноватая	153	108.27
5	2280	Август	Шоколадные конфеты	Ясиноватая	650	116.46
6	927	Апрель	Укроп	Ясиноватая	766	117.6
7	1883	Июль	Мороженое	Ясиноватая	768	119.87

В данном примере, значение в поле **City** были отсортированы по убыванию, а в поле **Amount** - по возрастанию. Оператор **DESC** применяется только для одного столбца, поэтому при необходимости его нужно прописывать после каждого поля, которое принимает участие в сортировке.

Задание:

Задание:

1. Выберите Книги по дате публикации в возрастающем порядке с полями название, дата публикации, цена
2. Выберите Книги по названию в убывающем порядке с полями цена, название
3. Выберите название авторов книг (Authors) по убыванию со всеми полями
4. Выберите название Издателей по возрастанию со всеми полями
5. Выберите Книги по цене в убывающем порядке, названию в возрастающем порядке со всеми полями кроме id_publisher

SQL-Урок 4. Фильтрация данных (WHERE)

В большинстве случаев необходимо получать не все записи, а только те, которые соответствуют определенным критериям. Поэтому для осуществления фильтрации выборки в SQL есть специальный оператор **WHERE**.

1. Простое фильтрование оператором WHERE.

Давайте из нашей таблицы, например, отберем записи, относящиеся только к определенному товару. Для этого мы укажем дополнительный параметр отбора, который будет фильтровать значение по колонке **Product**.

Пример запроса для отбора текстовых значений:

```
SELECT * FROM Sumproduct WHERE Product = 'Кофе'
```

	ID	Month	Product	City	Quantity	Amount
1	23	Январь	Кофе	Горловка	206	131.39
2	38	Январь	Кофе	Кировское	688	367.67
3	59	Январь	Кофе	Зугрэс	593	362.58
4	105	Январь	Кофе	Дебальцево	490	738.53
5	108	Январь	Кофе	Донецк	14	182.12
6	153	Январь	Кофе	Шахтёрск	219	601.42
7	178	Январь	Кофе	Докучаевск	625	196.08
8	184	Январь	Кофе	Харцызск	938	884.13
9	209	Январь	Кофе	Енакиево	449	196.1

Как видим, условие отбора взято в одинарные кавычки, что является обязательным при фильтровании текстовых значений. При фильтровании числовых значений кавычки не нужны.

Пример запроса для отбора числовых значений:

SELECT * FROM Sumproduct WHERE Amount > 800 ORDER BY Amount

	ID	Month	Product	City	Quantity	Amount
1	2800	Октябрь	Петрушка	Ясиноватая	767	800.21
2	3083	Ноябрь	Кефир	Харцызск	686	800.26
3	1404	Июнь	Чеснок	Ясиноватая	802	800.35
4	2421	Сентябрь	Сметана	Зугрэс	472	800.39
5	2770	Октябрь	Кофе	Докучаевск	930	800.45
6	3074	Ноябрь	Яйца	Ясиноватая	941	800.66
7	924	Апрель	Рыба	Ясиноватая	786	800.95
8	2548	Сентябрь	Помидоры	Докучаевск	419	801.03

В этом примере мы отобрали записи, в которых выручка от реализации составила более **800** и, дополнительно, все записи отсортировали по возрастанию по полю **Amount**.

В таблице ниже указан перечень условных операторов, поддерживаемых SQL:

Знак операции	Значение
=	Равно
<>	Не равно
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
BETWEEN	Между двумя значениями

IS NULL

Отсутствует запись

2. Фильтрация по диапазону значений (BETWEEN).

Для отбора данных, которые лежат в определенном диапазоне, используется оператор **BETWEEN**. В следующем запросе будут отображены все значения, лежащие в пределах от **850 \$** в **860 \$** включительно, в поле **Amount**.

SELECT * FROM Sumproduct WHERE Amount BETWEEN 800 AND 860

	ID	Month	Product	City	Quantity	Amount
1	19	Январь	Яйца	Горловка	123	858.55
2	49	Январь	Молоко	Кировское	10	812.23
3	56	Январь	Чай	Зугрэс	405	829
4	65	Январь	Шоколад	Зугрэс	578	803.63
5	79	Январь	Груши	Зугрэс	16	808.71
6	97	Январь	Шоколад	Дебальцево	738	827.21
7	111	Январь	Капуста	Донецк	323	844.21
8	211	Январь	Сливочное масло	Енакиево	504	806.06

3. Выборка пустых записей (IS NULL).

В **SQL** существует специальный оператор для выборки пустых записей (называется **NULL**). Пустой записью считается любая ячейка в таблице, в которую не введены какие-либо символы. Если в ячейку введен **0** или **пробел**, то считается, что поле заполнено.

Для начала удалим **City** у первой записи

	ID	Month	Product	City	Quantity	Amount
1	1	Январь	Шоколадные кофеты	Горловка	216	924.12
2	2	Январь	Молоко	Горловка	915	747.67
3	3	Январь	Шоколад	Горловка	249	368.94
4	4	Январь	Соль	Горловка	574	473.52
5	5	Январь	Сахар	Горловка	665	632.56

	ID	Month	Product	City	Quantity	Amount
1	1	Январь	Шоколадные кофеты	NULL	216	924.12
2	2	Январь	Молоко	Горловка	915	747.67
3	3	Январь	Шоколад	Горловка	249	368.94
4	4	Январь	Соль	Горловка	574	473.52
5	5	Январь	Сахар	Горловка	665	632.56

Выберем **City** с значением **NULL**

SELECT * FROM Sumproduct WHERE City IS NULL

ID	Month	Product	City	Quantity	Amount
1	1 Январь	Шоколадные кофеты	NULL	216	924.12

В примере выше, мы нарочно удалили значение в поле **City**, чтобы продемонстрировать работу оператора **NULL**.

Верните значение City для первой записи

ID	Month	Product	City	Quantity	Amount
1	1 Январь	Шоколадные кофеты	Горловка	216	924.12
2	2 Январь	Молоко	Горловка	915	747.67
3	3 Январь	Шоколад	Горловка	249	368.94
4	4 Январь	Соль	Горловка	574	473.52

4. Расширенная фильтрация (AND, OR).

Язык **SQL** не ограничивается фильтрацией по одному условию, для собственных целей вы можете использовать достаточно сложные конструкции для выборки данных одновременно по многим критериям. Для этого в **SQL** есть дополнительные операторы, которые расширяют возможности оператора **WHERE**. Такими операторами являются: **AND, OR, IN, NOT**. Приведем несколько примеров работы данных операторов.

SELECT * FROM Sumproduct WHERE Amount > 900 AND City = 'Докучаевск'

ID	Month	Product	City	Quantity	Amount
1	579 Март	Груши	Докучаевск	304	965.85
2	582 Март	Яблоки	Докучаевск	195	974.28
3	1017 Апрель	Апельсины	Докучаевск	424	939.39
4	1023 Апрель	Огурцы	Докучаевск	736	917.06
5	1210 Май	Лук	Докучаевск	906	904.99
6	1820 Июль	Кофе	Докучаевск	154	953.14

SELECT * FROM Sumproduct WHERE Month = 'Сентябрь' OR Month = 'Декабрь'

ID	Month	Product	City	Quantity	Amount	
274	2568	Сентябрь	Шоколадные кофеты	Углегорск	762	617.41
275	2569	Сентябрь	Укроп	Углегорск	407	924
276	2570	Сентябрь	Петрушка	Углегорск	191	654.88
277	2571	Сентябрь	Лук	Углегорск	6	974.07
278	2572	Сентябрь	Сахар	Углегорск	772	698.75
279	3179	Декабрь	Сливочное масло	Макеевка	155	870.54
280	3180	Декабрь	Картофель	Макеевка	100	889.24
281	3181	Декабрь	Петрушка	Макеевка	444	819.84
282	3182	Декабрь	Яблоки	Макеевка	727	764.48
283	3183	Декабрь	Кофе	Макеевка	892	444.65

Давайте объединим операторы **AND** и **OR**. Для этого сделаем выборку продуктов *Рыба* и *Мясо*, которые были проданы в марте (*Март*).

SELECT * FROM Sumproduct WHERE Product = 'Рыба' OR Product = 'Мясо' AND Month= 'Март'

ID	Month	Product	City	Quantity	Amount	
34	793	Март	Мясо	Донецк	782	663.54
35	805	Март	Рыба	Донецк	664	818.15
36	820	Март	Рыба	Зугрэс	460	589.64
37	839	Март	Мясо	Зугрэс	679	403.68
38	845	Март	Рыба	Макеевка	101	491.84
39	859	Март	Мясо	Макеевка	12	623.34
40	878	Март	Мясо	Енакиево	109	962.92
41	889	Апрель	Рыба	Харцызск	35	665.92
42	924	Апрель	Рыба	Ясиноватая	786	800.95
43	964	Апрель	Рыба	Ждановка	944	281.32
44	1012	Апрель	Рыба	Иловыйск	956	396.83
45	1060	Апрель	Рыба	Дебальцево	46	342.37
46	1115	Май	Рыба	Енакиево	389	606.06
47	1151	Май	Рыба	Ясиноватая	911	120.43

Видим, что в нашу выборку попало за много значений (кроме марта (**Март**), также апрель (**Апрель**), май (**Май**) и т.д.. В чем же причина? А в том, что **SQL** имеет приоритеты выполнения команд. То есть оператор **AND** имеет более высокий приоритет, чем оператор **OR**, поэтому сначала были отобраны записи с мясом, которые проданы в марте, а потом все записи, касающиеся рыбы.

Итак, чтобы получить правильную выборку, нам нужно изменить приоритеты выполнения команд. Для этого используем **скобки**, как в математике. Тогда, сначала будут обработаны операторы в скобках, а затем - все остальные.

SELECT * FROM Sumproduct WHERE (Product = 'Рыба' OR Product = 'Мясо') AND Month= 'Март'

ID	Month	Product	City	Quantity	Amount	
1	524	Март	Мясо	Дебальцево	782	136.53
2	562	Март	Мясо	Снежное	104	188.15
3	588	Март	Мясо	Докучаевск	809	775.47
4	589	Март	Рыба	Докучаевск	84	314.7
5	625	Март	Мясо	Ждановка	437	524.78
6	630	Март	Рыба	Ждановка	891	569.04
7	646	Март	Мясо	Иловайск	843	546.19
8	651	Март	Рыба	Иловайск	761	721.93
9	668	Март	Мясо	Углегорск	334	142.78
10	678	Март	Рыба	Углегорск	899	954.16
11	708	Март	Мясо	Торез	189	475.02
12	731	Март	Рыба	Горловка	102	821.51
13	745	Март	Мясо	Шахтёрск	320	935.92
14	750	Март	Рыба	Шахтёрск	54	421.69
15	781	Март	Мясо	Харцызск	304	705.09
16	785	Март	Рыба	Харцызск	121	644.84

5. Расширенная фильтрация (оператор IN).

SELECT * FROM Sumproduct WHERE ID IN (4, 12, 58, 67, 927, 2132)

ID	Month	Product	City	Quantity	Amount	
1	4	Январь	Соль	Горловка	574	473.52
2	12	Январь	Чай	Горловка	607	667.9
3	58	Январь	Яйца	Зугрэс	920	714
4	67	Январь	Апельсины	Зугрэс	172	191.36
5	927	Апрель	Укроп	Ясиноватая	766	117.6
6	2132	Август	Укроп	Донецк	957	832.04

Оператор **IN** выполняет ту же функцию, что и **OR**, однако имеет ряд преимуществ:

- При работе с длинными списками, предложение с **IN** легче читать;
- Используется меньшее количество операторов, что ускоряет обработку запроса;
- Самое важное преимущество **IN** в том, что в его конструкции можно использовать дополнительную конструкцию **SELECT**, что открывает большие возможности для создания сложных подзапросов.

6. Расширенная фильтрация (оператор NOT).

SELECT * FROM Sumproduct WHERE NOT City IN ('Донецк', 'Ясиноватая')

Табличный вид		Форма				
		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Всего загружено строк: 3042
ID	Month	Product	City	Quantity	Amount	
1	1 Январь	Шоколадные кофеты	Горловка	216	924.12	
2	2 Январь	Молоко	Горловка	915	747.67	
3	3 Январь	Шоколад	Горловка	249	368.94	
4	4 Январь	Соль	Горловка	574	473.52	
5	5 Январь	Сахар	Горловка	665	632.56	
6	6 Январь	Йогурт	Горловка	380	257.47	
7	7 Январь	Помидоры	Горловка	132	298.8	

Ключевое слово **NOT** позволяет убрать ненужные значения из выборки. Также его особенностью является то, что оно проставляется перед названием столбца, участвующего в фильтровании, а не после

Задание 4:

2. Выбрать Книги с ценой от 950.62 включительно с полями название, описание, цена отсортированных по названию в убывающем порядке
 1. Выбрать Авторы с именем Adella Hagenes
3. Выбрать Книги где цена от 300 до 400 и дата публикации с начала 2001 года. Отсортировать от самой старой даты публикации
4. Выбрать Книги с полями id, имя, цена по условию где id может быть одним из след значений 140066, 3431, 78360, 186656, 197115 или имя Robbie Maggio. Отсортировать все по имени в убывающем порядке
5. Выбрать Издателей id которых не 1, 5, 22, 33. Отсортировать название в убывающем порядке
6. Выбрать Авторы имя которых или Jeffry Champlin или Kari Will и id больше 15 или выбрать id из диапазона 20-50. Не использовать оператор IN
7. Выбрать Авторы имя которых или Vicenta Shields или Kari Will или Randall Stehr Sr. и id больше 12 или выбрать id из диапазона 80 до 100. Не использовать оператор IN
9. Выбрать Книги с датой публикации между мартом 12 и апрелем 21 2010 года. Отсортировать книги от самых новых публикаций
8. Выбрать Книги цена которых больше 440.02 но меньше 500.00 и описание пустое с полями: название, описание, цена. С сортировкой по цене от меньшего к большему и названию по возрастанию.

SQL-Урок 5. Символы подстановки (LIKE)

Часто, для фильтрации данных, нам нужно будет осуществить выборку не по точному совпадению условия, а по приближенному значению. То есть когда, например, мы ищем товар, название которого соответствует определенному шаблону или содержит определенные символы или слова. Для таких целей в SQL существует оператор LIKE, который ищет приближенные значения. Для конструирования такого шаблона используются метасимволы

1. Метасимвол знак процента (%)

```
SELECT * FROM Sumproduct WHERE Product LIKE 'Шоколад%'
```

Всего загружено строк: 219

ID	Month	Product	City	Quantity	Amount
1	1 Январь	Шоколадные конфеты	Горловка	216	924.12
2	3 Январь	Шоколад	Горловка	249	368.94
3	37 Январь	Шоколад	Кировское	364	717.49

Как видим, СУБД отобрала только те записи, где в колонке **Product** были товары, начинающиеся на слово **Шоколад**

SELECT * FROM Sumproduct WHERE Product LIKE '%ко'

Всего загружено строк: 115

ID	Month	Product	City	Quantity	Amount
1	2 Январь	Молоко	Горловка	915	747.67
2	49 Январь	Молоко	Кировское	10	812.23
3	51 Январь	Молоко	Зугрэс	989	539.66

Отобрали записи, где в колонке **Product** были товары, заканчивающиеся на **ко**

SELECT * FROM Sumproduct WHERE Product LIKE '%ко%'

Всего загружено строк: 445

ID	Month	Product	City	Quantity	Amount
1	1 Январь	Шоколадные кофе...	Горловка	216	924.12
2	2 Январь	Молоко	Горловка	915	747.67
3	3 Январь	Шоколад	Горловка	249	368.94
4	25 Январь	Морковь	Горловка	990	228.48
5	37 Январь	Шоколад	Кировское	364	717.49
6	39 Январь	Шоколадные кофе...	Кировское	28	529.7

СУБД отобрала только те записи, где в колонке **Product** были товары, содержащие **ко** в названии.

2. Метасимвол знак подчеркивания (_)

Знак подчеркивания применяется для того, чтобы заменить один символ в слове. Давайте найдем слово состоящие из пяти букв которое заканчивается на **p**:

SELECT * FROM Sumproduct WHERE Product LIKE '____p'

Всего загружено строк: 225

ID	Month	Product	City	Quantity	Amount
1	5 Январь	Сахар	Горловка	665	632.56
2	26 Январь	Кефир	Горловка	68	764.93
3	55 Январь	Кефир	Зугрэс	285	126.27
4	72 Январь	Сахар	Зугрэс	632	562.98
5	86 Январь	Сахар	Лебальцево	825	150.65

Задание:

1.Найти описание книг где начинается со слова 'mouse'

2. Найти книги название которых содержит 'cortez' или 'kirlin' и 'si'; по цене больше 500. Отсортировать по самым новым публикациям

3. Найти книги название которых заканчиваются на 'en' и в описание содержит 'ready' и 'to' или 'and sometimes'. Вывести поля цена, дата публикации, название, описание.

4. Найти книги название которых состоят из 12 букв где последние две буквы это 'es'. Сортировка названия по возрастанию

5. Найти книги которые в названии содержат слово 'ickl' или заканчиваются на 'dds' или состоят из 12 букв где вторая и третья буква это всегда 'ud'. Цена между 100 и 800. Отсортировать название по алфавиту в убывающем порядке

6. Найти книги где описание содержит СЛОВО 'ready'
Т.к. мы не используем регулярные выражения –

SQL-Урок 6. Вычисляемые поля

Для чего нужно использовать расчетные поля? Как правило, информация в БД представлена в разрезе отдельных фрагментов, поскольку так легче структурировать данные и оперировать ими. Однако нам часто будет нужно использовать не отдельные части данных, а уже соединенную и обработанную информацию. Например, часто необходимо сочетать имя и фамилию клиентов, сочетать элементы адресов, которые находятся в разных столбцах таблицы, обрабатывать текст и отдельные слова, буквы и символы, суммировать общую стоимость покупки, отображать статистику по информации, находящейся в БД. Данные обычно хранятся отдельными "кусками", что требует их дополнительной обработки на стороне клиентского приложения. Однако есть возможность получать уже обработанную информацию с помощью СУБД. Именно в этом случае помогают расчетные поля. Они автоматически создаются при выполнении запроса и имеют вид и свойства обычных столбцов, которые уже имеются в таблице. Единственное отличие заключается в том, что физически расчетных полей нет, поэтому они не занимают дополнительного места в БД, а временно существуют в "оперативной памяти" СУБД. Преимуществом выполнения операций на стороне СУБД является скорость обработки данных.

1. Выполнение математических операций

Одним из способов использования расчетных полей является выполнение математических операций над выбранными данными. Давайте на примере рассмотрим как это происходит, используя снова нашу таблицу **Sumproduct**. Предположим, нам нужно вычислить среднюю цену приобретения каждого товара. Для этого нужно переделить колонку **Amount** (сумма) на **Quantity** (количество):

```
SELECT *, Amount/Quantity FROM Sumproduct
```

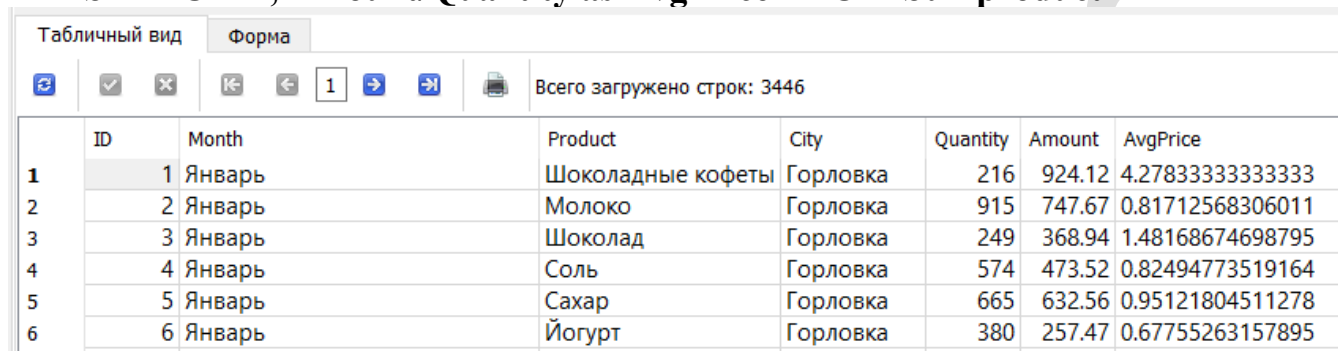
ID	Month	Product	City	Quantity	Amount	Amount / Quantity
1	1 Январь	Шоколадные конфеты	Горловка	216	924.12	4.27833333333333
2	2 Январь	Молоко	Горловка	915	747.67	0.81712568306011
3	3 Январь	Шоколад	Горловка	249	368.94	1.48168674698795
4	4 Январь	Соль	Горловка	574	473.52	0.82494773519164
5	5 Январь	Сахар	Горловка	665	632.56	0.95121804511278
6	6 Январь	Йогурт	Горловка	380	257.47	0.67755263157895
7	7 Январь	Помидоры	Горловка	132	298.8	2.26363636363636

Как видим, СУБД отобрала все наименования товаров и отобразила их среднюю стоимость в отдельном столбце, который был создан во время выполнения запроса.

2. Использование псевдонимов

В предыдущем примере мы рассчитывали среднюю стоимость покупки и отображали значение в расчетном столбце. Однако в дальнейшем, нам неудобно обращаться к этому полю, так как его название выбрала СУБД. (СУБД дала название полю Amount / Quantity). Однако мы можем назвать поле самостоятельно, заранее указав его название в запросе, то есть дать псевдоним. Давайте перепишем предыдущий пример и укажем псевдонима для расчетного поля:

```
SELECT *, Amount/Quantity as AvgPrice FROM Sumproduct
```



ID	Month	Product	City	Quantity	Amount	AvgPrice
1	1 Январь	Шоколадные конфеты	Горловка	216	924.12	4.27833333333333
2	2 Январь	Молоко	Горловка	915	747.67	0.81712568306011
3	3 Январь	Шоколад	Горловка	249	368.94	1.48168674698795
4	4 Январь	Соль	Горловка	574	473.52	0.82494773519164
5	5 Январь	Сахар	Горловка	665	632.56	0.95121804511278
6	6 Январь	Йогурт	Горловка	380	257.47	0.67755263157895

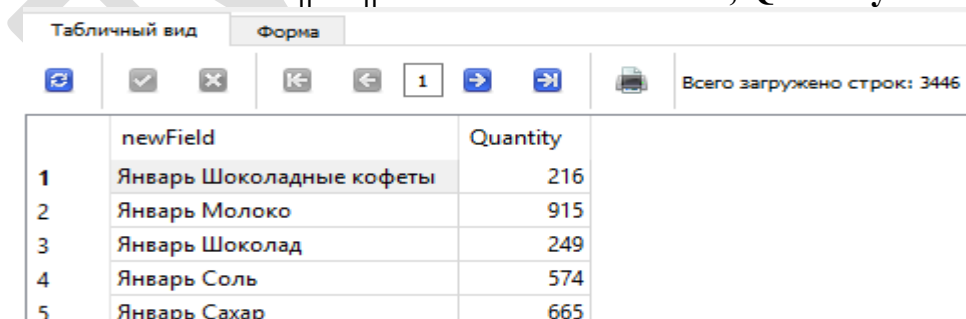
Видим, наше расчетное поле получило собственное название **AvgPrice**. Для этого мы использовали оператор **AS**, после которого указали необходимое нам название. Стоит отметить, что в SQL поддерживаются только основные математические операции: сложение (+), вычитание (-), умножение (*), деление (/). Также для изменения очередности выполнения операции можно использовать круглые скобки.

Часто псевдонимы используют не только чтобы называть расчетные поля, но и для переименования действующих. Это может быть необходимым, если действующее поле имеет длинное название или название не достаточно информативным.

3. Соединение полей (конкатенация)

Кроме математических операций мы можем объединять текст и выводить его в отдельном поле. Давайте рассмотрим, каким образом можно осуществить склеивание (конкатенацию) текста. Имеем такой пример:

```
SELECT Month || ' ' || Product AS newField, Quantity FROM Sumproduct
```



	newField	Quantity
1	Январь Шоколадные конфеты	216
2	Январь Молоко	915
3	Январь Шоколад	249
4	Январь Соль	574
5	Январь Сахар	665

В этом примере мы соединили значение в двух столбцах и вывели результат в новое поле **NewField**

Задание:

1. Показать книги цена которых больше 500. Вывести цену рядом с основной ценой меньше на 25. Поля: название, цена, цена с вычетом

2. Показать цены книг если они подражали бы в два раза. Дата публикации с 2000 года.

3. Найти книги цена которых больше 900 и вывести рядом дополнительную цифру: цена + налог 25 процентов. Использовать для этого поле псевдоним. Отсортировать по дополнительному полю где самые высокие цены с налогом в начале

4. Найдите все книги название которых начинается на 'wai' и добавьте поле в начало состоящее из идентификатора, знака дефис и названия. Назовите это поле как newField.

5. Выберите все книги с названием по шаблону '{название}:{остаток} = {цена} + {цена налога} - {скидка}'

Нужно посчитать и вывести подставив цифры остатка, цена налога, скидка где цена налога 3 процента; скидка 10 единиц.

SQL-Урок 7. Функции обработки данных

Как и в большинстве языков программирования, в SQL существуют функции для обработки данных. Стоит отметить, что в отличие от SQL-операторов, функции не стандартизованы для всех видов СУБД, то есть для выполнения одних и тех же операции над данными, разные СУБД имеют свои собственные имена функций. Это означает, что код запроса написан в одной СУБД может не работать в другой, и это нужно учитывать в дальнейшем. Больше всего это касается функций для обработки текстовых значений, преобразования типов данных и манипуляций над датами.

Обычно СУБД поддерживается стандартный набор типов функций, а именно:

- Текстовые функции, которые используются для обработки текста (выделение части символов в тексте, определение длины текста, перевод символов в верхний или нижний регистр ...)
- Числовые функции. Используются для выполнения математических операций над числовыми значениями
- Функции даты и времени (осуществляют манипулирования датой и временем, рассчитывают период между датами, проверяют даты на корректность и т.п.)
- Статистические функции (для вычисления максимальных /минимальных значений, средних значений, подсчет количества и суммы ...)
- Системные функции (предоставляют разного рода служебную информацию о СУБД, пользователе и др..).

1. Функции SQL для обработки текста

Реализация SQL в СУБД Access имеет следующие функции для обработки текста:

<u>SUBSTR</u>	Извлечь и вернуть подстроку с предварительно определенной длиной, начиная с указанной позиции в исходной строке
<u>TRIM</u>	Вернуть копию строки, в которой указанные символы удалены из начала и конца строки.
<u>LTRIM</u>	Вернуть копию строки, в которой указанные символы удалены из начала строки.

<u>RTRIM</u>	Вернуть копию строки, в которой указанные символы удалены из конца строки.
<u>LENGTH</u>	Вернуть количество символов в строке или количество байтов в BLOB.
<u>REPLACE</u>	Вернуть копию строки, в которой каждый экземпляр подстроки заменяется другой подстрокой.
<u>UPPER</u>	Вернуть копию строки со всеми символами, преобразованными в верхний регистр.
<u>LOWER</u>	Вернуть копию строки со всеми символами, преобразованными в нижний регистр.
<u>INSTR</u>	Найти подстроку в строке и вернуть целое число, указывающее позицию первого вхождения подстроки.

SELECT Name as original, **UPPER**(Name), **LOWER**(Name) **FROM** Names

	original	UPPER(Name)	LOWER(Name)
1	Nikita	NIKITA	nikita
2	Alex	ALEX	alex
3	Karina	KARINA	karina
4	Lilya	LILYA	lilya
5	Petya	PETYA	petya
6	Sveta	SVETA	sveta

SELECT Name as original, **TRIM**(Name, 'a'), **LTRIM**(Name, 'A'), **RTRIM**(Name, 'a') **FROM** Names

Всего загружено строк: 6

	original	TRIM(Name, 'a')	LTRIM(Name, 'A')	RTRIM(Name, 'a')
1	Nikita	Nikit	Nikita	Nikit
2	Alex	Alex	lex	Alex
3	Karina	Karin	Karina	Karin
4	Lilya	Lily	Lilya	Lily
5	Petya	Pety	Petya	Pety
6	Sveta	Svet	Sveta	Svet

SELECT Name as original, **LENGTH**(Name), **REPLACE**(Name, 'a', 'Z'), **INSTR**(Name, 'i'), **SUBSTR**(Name, 3, 2) **FROM** Names

original	LENGTH(Name)	REPLACE(Name, 'a', 'Z')	INSTR(Name, 'i')	SUBSTR(Name, 3, 2)
Nikita	6	NikitZ	2	ki
Alex	4	Alex	0	ex
Karina	6	KZrinZ	4	ri
Lilya	5	LilyZ	2	ly
Petya	5	PetyZ	0	ty
Sveta	5	SvetZ	0	et

2. Функции SQL для обработки чисел

Функции обработки чисел предназначены для выполнения математических операций над числовыми данными.

<u>ABS</u>	Возвращает абсолютное значение числа
<u>RANDOM</u>	Возвращает случайное значение с плавающей запятой между минимальным и максимальным целочисленными значениями.
<u>ROUND</u>	Округлить плавающее значение с заданной точностью.

SELECT ABS(-1000);

ABS(- 1000)
1 1000

SELECT ABS(10-200);

ABS(10 - 200)
1 190

SELECT RANDOM();

random()
3260037847516095343

SELECT ROUND(1929.236, 2);

round(1929.236, 2)
1929.24

SELECT Amount, ROUND(Amount, 1), ROUND(ABS(ROUND(Amount, 1) - Amount), 2) FROM Sumproduct

	Amount	ROUND(Amount, 1)	ROUND(ABS(ROUND(Amount, 1) - Amount), 2)
1	924.12	924.1	0.02
2	747.67	747.7	0.03
3	368.94	368.9	0.04
4	473.52	473.5	0.02
5	632.56	632.6	0.04
6	257.47	257.5	0.03
7	298.8	298.8	0
8	455.44	455.4	0.04
9	671.99	672	0.01
10	653.7	653.7	0

В этом примере во втором столбце округлили до одного знака после запятой Amount. В третьем столбце СУБД сначала округлила Amount - **ROUND(Amount, 1)** и далее отняла от суммы Amount. Получилось отрицательное число, которое с помощью ABS преобразовалось в абсолютное число, которое округлили до 2 знаков.

3. Функции SQL для обработки даты и времени

Функции манипулирования датой и временем являются одними из важнейших и часто используемых функций SQL. В базах данных значения дат и времени хранятся в специальном формате, поэтому их невозможно использовать напрямую без дополнительной обработки. Каждая СУБД имеет свой набор функций для обработки дат, что, к сожалению, не позволяет переносить их на другие платформы и реализации SQL.

<u>DATE</u> (timestring, modifier, modifier,...)	Вычисляет дату на основе нескольких модификаторов даты.
<u>TIME</u>	Вычисляет время на основе нескольких модификаторов даты.
<u>DATETIME</u>	Вычисляет дату и время на основе одного или нескольких модификаторов даты.
<u>STRFTIME</u>	Форматирует дату на основе указанной строки формата.

Форматы строк даты и времени

1	YYYY-MM-DD
2	YYYY-MM-DD HH:MM
3	YYYY-MM-DD HH:MM:SS
4	YYYY-MM-DD HH:MM:SS.SSS
5	YYYY-MM-DDTHH:MM
6	YYYY-MM-DDTHH:MM:SS
7	YYYY-MM-DDTHH:MM:SS.SSS
8	HH:MM
9	HH:MM:SS
10	HH:MM:SS.SSS
11	now
12	DDDDDDDDDD

Модификаторы даты

1	NNN days	Добавьте ± NNN дней к дате и времени
2	NNN hours	Добавьте ± NNN часов к дате и времени
3	NNN minutes	Добавьте ± NNN минут к дате и времени
4	NNN.NNNN seconds	Добавьте ± NNN секунд к / от даты и времени
5	NNN months	Добавьте ± NNN месяцев к дате и времени

6	NNN years	Добавить ± NNN год к дате и времени
7	start of month	Назад к началу месяца
8	start of year	назад к началу года
9	start of day	Обратно к началу дня
10	weekday N	Перенесите дату вперед на следующую дату, где номером дня недели будет N
11	unixepoch	Unix time
12	localtime	Вернуть местное время
13	utc	Время возврата по UTC

построения формата

%d	день месяца: 01-31
%f	доли секунды: SS.SSS
%H	час: 00-24
%j	день года: 001-366
%J	Юлианский день номер
%m	месяц: 01-12
%M	минута: 00-59
%s	секунд с 1970-01-01
%S	секунд: 00-59
%w	день недели 0-6 с воскресеньем == 0
%W	неделя года: 00-53
%Y	год: 0000-9999
%%	%

SELECT DATE('now', '-1 day');

Табличный вид		Форма	
Всего загружено строк: 1			
date('now', '-1 day')			
1	2019-03-17		

Текущая дата минус 1 день

SELECT DATE('2018-11-01', '-1 month');

Табличный вид		Форма	
date('2018-11-01', '-1 month')		Всего загружено строк: 1	
1	2018-10-01		

Текущая дата минус один месяц

SELECT DATE('now', 'start of month', '+1 month', '-1 day');

Табличный вид		Форма	
DATE('now', 'start of month', '+1 month', '-1 day')		Всего загружено строк: 1	
1	2019-03-31		

- `now` это строка времени, которая указывает текущую дату.
- `start of month`, `+1 month` и `-1 day` являются модификаторы.

Функция работает следующим образом:

- Сначала `start of month` применяется к текущей дате, указанной в `now` строке времени, поэтому результатом является первый день текущего месяца.
- Во-вторых, `+1 month` применяется к первому дню текущего месяца, который соответствует первому дню следующего месяца.
- В-третьих, `-1 day` применяется к первому дню следующего месяца, то есть к последнему дню предыдущего месяца.

SELECT TIME('10:20:30', '+2 hours');

Табличный вид		Форма	
time('10:20:30', '+2 hours')		Всего загружено строк: 1	
1	12:20:30		

Добавили 2 часа времени

SELECT ID, NAME, created_at, STRFTIME('%d', created_at) as day, STRFTIME('%m', created_at) as month, DATETIME (created_at, '+16 hour') FROM Names WHERE created at > DATE('now', '-1 day')

ID	NAME	created_at	day	month	datetime(created_at, '+16 hour')
1	4 Nikita	2019-03-19 10:15:14	19	03	2019-03-20 02:15:14
2	5 Alex	2019-03-20 10:16:31	20	03	2019-03-21 02:16:31
3	7 Lilya	2019-03-20 09:17:25	20	03	2019-03-21 01:17:25
4	8 Petya	2019-03-20 08:14:27	20	03	2019-03-21 00:14:27
5	9 Sveta	2019-03-20 10:12:20	20	03	2019-03-21 02:12:20

4. Статистические функции SQL

Статистические функции помогают нам получить готовые данные без их выборки. SQL-запросы с этими функциями часто используются для анализа и создания

различных отчетов. Примером таких выборок может быть: определение количества строк в таблице, получение суммы значений по определенному полю, поиск наибольшего /наименьшего или среднего значения в указанном столбце таблицы. Также отметим, что статистические функции поддерживаются всеми СУБД без особых изменений в написании.

COUNT()	Возвращает число строк в таблице или столбце
SUM()	Возвращает сумму значений в столбце
MIN()	Возвращает наименьшее значение в столбце
MAX()	Возвращает наибольшее значение в столбце
AVG()	Возвращает среднее значение в столбце

Примеры использования функции COUNT():

SELECT COUNT(*) AS Count1 FROM Sumproduct - возвращает количество всех строк в таблице

The screenshot shows a database interface with a toolbar at the top containing icons for refresh, check, close, navigation, and a page indicator showing '1' of 1 rows. Below the toolbar, the result is displayed in a table with one column named 'Count1' and one row containing the value '3446'.

Count1
3446

Примеры использования функции SUM():

SELECT SUM(Quantity) AS Sum1 FROM Sumproduct WHERE Month = 'Апрель'

The screenshot shows a database interface with a toolbar at the top. Below the toolbar, the result is displayed in a table with one column named 'Sum1' and one row containing the value '110896'.

Sum1
110896

Здесь отразили общее количество проданного товара в апреле.

SELECT SUM(Quantity*Amount) AS Sum2 FROM Sumproduct WHERE Month = 'Апрель'

The screenshot shows a database interface with a toolbar at the top. Below the toolbar, the result is displayed in a table with one column named 'Sum2' and one row containing the value '59134395.659999974'.

Sum2
59134395.659999974

Как видим, в статистических функциях мы можем осуществлять вычисления над несколькими столбцами с использованием стандартных математических операторов.

Пример использования функции MIN():

SELECT MIN(Amount) AS Min1 FROM Sumproduct

Табличный вид		Форма	
		Всего загружено строк: 1	
Min1			
1	100.58		

Пример использования функции MAX():

SELECT MAX(Amount) AS Max1 FROM Sumproduct

Табличный вид		Форма	
		Всего загружено строк: 1	
Max1			
1	999.75		

Пример использования функции AVG():

SELECT AVG(Amount) AS Avg1 FROM Sumproduct

Табличный вид		Форма	
		Всего загружено строк: :	
Avg1			
1	548.6985432385371		

Задание часть 1:

- Вывести название Издательств в верхнем регистре и посчитать кол-во символов в названии - использовать псевдоним, где это необходимо
- Вывести всех авторов отрезав первую букву 'D' если она есть у автора, заменить имя 'Kari' на 'Danil' если есть. Вывести поля имя, без буквы в начале, с замененным словом.
Использовать псевдоним, где это необходимо
- Найти книги где
 - длина названия не больше 6 символов или содержит 'ni' и 'mr'
 - описание содержит 'voice' и позиция этого слова не позже 30 символа
 - цена от 200 до 900
 - id в таблице больше 20000
 Вывести поля
 - названия в нижнем регистре
 - описание в верхнем регистре
 - id имеет псевдоним 'индикатор'
 Отсортировать все по длине название где самые длинные названия в начале
- Вывести Книги, где есть поля названия, цена, целая часть цены, целая часть цены минус 800 по модулю. Отсортировать по возрастанию последнего поля.

5. Заменить все буквы 'e' на 'x' в названии Книг, длина названия от 6 до 8 символа включительно и индикатор из этого списка: 63719 или 154928 или 5860 или 12123 или 15491 или 22314 или 1323 или 2345 или 3425 или 43234
Все поля в верхнем регистре. Использовать псевдоним, где это необходимо. Отсортировать по длине названия, где самые длинные вверху.

Задание часть2:

1. Вывести все книги, которые издались пол года назад.
2. Вывести все книги, где дата разбита отдельно на год, месяц, день, часы, минуты, секунды.
3. Прибавить к дате издания 9 месяцев и вывести те, которые больше текущей даты
4. Вывести книги за 2 последних високосных года. Отсортировать по убыванию цены.
5. Вывести минимальную цену книги
6. Вывести разницу между максимальной и текущей ценой.
7. Вывести среднюю цену по издателю с id_publisher 11. Округлить цену до целого числа
8. Вывести общую сумму по издателю с id_publisher 15. Округлить сумму до 2 знаков после запятой
9. Вывести общее кол-во книг где цена больше 500

SQL-Урок 8 Группировка данных (GROUP BY)

Группировка данных позволяет разделить все данные на логические наборы, благодаря чему становится возможным выполнение статистических вычислений отдельно в каждой группе.

1. Создание групп (GROUP BY)

Группы создаются с помощью предложения **GROUP BY** оператора **SELECT**. Рассмотрим на примере.

```
SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY Product
```

	Product	Product_num
1	Апельсины	53027
2	Бананы	52053
3	Груши	51389
4	Йогурт	46125
5	Капуста	58647
6	Карамель	54141
7	Картофель	56620

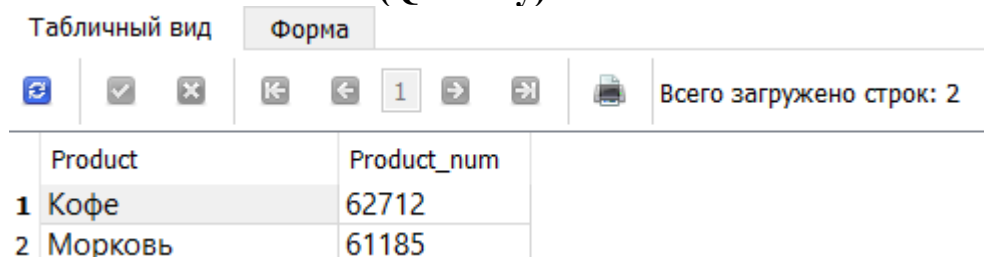
Данным запросом мы извлекли информацию о количестве реализованной продукции в каждом месяце. Оператор **SELECT** приказывает вывести два столбца

Product - название продукта и **Product_num** - расчетное поле, которое мы создали для отображения количества реализованной продукции (формула поля **SUM (Quantity)**). Предложение **GROUP BY** указывает СУБД сгруппировать данные по столбцу **Product**. Стоит также отметить, что **GROUP BY** должен идти после предложения **WHERE** и перед **ORDER BY**.

2. Фильтрующие группы (HAVING)

Так же, как мы фильтровали строки в таблице, мы можем осуществлять фильтрацию по сгруппированным данным. Для этого в SQL существует оператор **HAVING**. Возьмем предыдущий пример и добавим фильтрацию по группам.

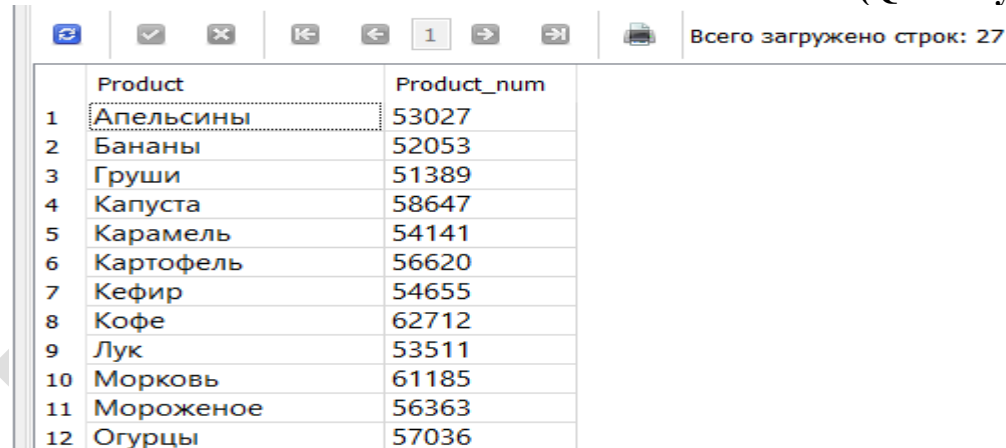
SELECT Product,SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY Product HAVING SUM(Quantity)>60000



	Product	Product_num
1	Кофе	62712
2	Морковь	61185

Как видим, оператор **HAVING** очень похож на оператора **WHERE**, однако между собой они имеют существенное отличие: **WHERE** фильтрует данные до того, как они будут сгруппированы, а **HAVING** - осуществляет фильтрацию после группировки. Таким образом, строки, которые были изъяты предложением **WHERE** НЕ будут включены в группу. Итак, операторы **WHERE** и **HAVING** могут использоваться в одном предложении. Рассмотрим пример:

SELECT Product, SUM(Quantity) AS Product_num FROM Sumproduct WHERE Product<>'Молоко' GROUP BY Product HAVING SUM(Quantity)>50000



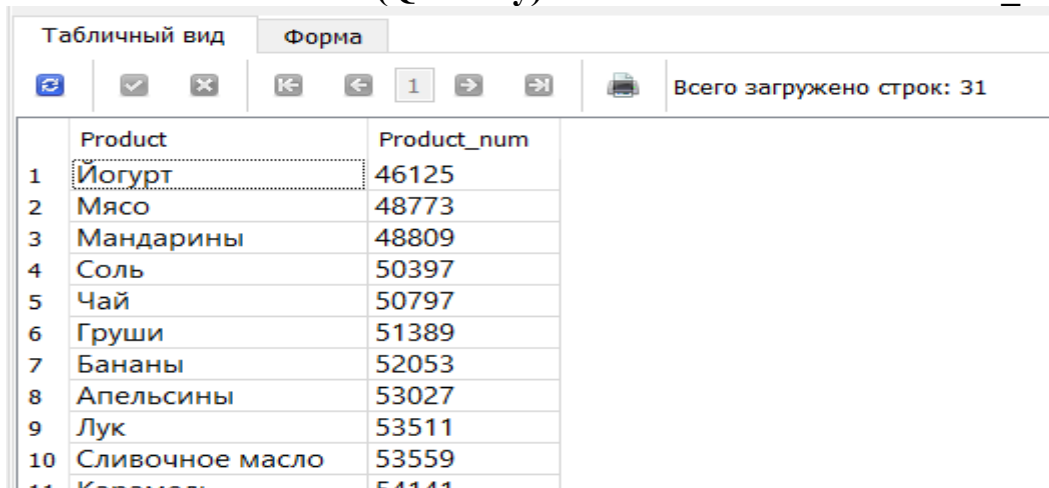
	Product	Product_num
1	Апельсины	53027
2	Бананы	52053
3	Груши	51389
4	Капуста	58647
5	Карамель	54141
6	Картофель	56620
7	Кефир	54655
8	Кофе	62712
9	Лук	53511
10	Морковь	61185
11	Мороженое	56363
12	Огурцы	57036

Мы к предыдущему примеру добавили оператор **WHERE**, где указали товар **Молоко**, что в свою очередь повлияло на группирование оператором **HAVING**. Как результат видим, что товар **Молоко** не попал в перечень групп с количеством реализованной продукции больше 50000 шт.

3. Группировка и сортировка

Как и при обычной выборке данных, мы можем сортировать группы после группировки оператором **HAVING**. Для этого мы можем использовать уже знакомый нам оператор **ORDER BY**. В данной ситуации его применения аналогичное предыдущим примерам. К примеру:

SELECT Product,SUM(Quantity) AS Product_num FROM Sumproduct GROUP BY Product HAVING SUM(Quantity)>45000 ORDER BY Product_num



	Product	Product_num
1	Йогурт	46125
2	Мясо	48773
3	Мандарины	48809
4	Соль	50397
5	Чай	50797
6	Груши	51389
7	Бананы	52053
8	Апельсины	53027
9	Лук	53511
10	Сливочное масло	53559
11	Карамель	54141

Видим, что для сортировки сводных результатов нам нужно просто прописать предложения с **ORDER BY** после оператора **HAVING**. Можно использовать псевдоним (alias) **AS**.

Задание :

1. Сгруппировать книги по издателям. Вывести общую сумму, среднюю сумму по каждому из них. Округлить до двух знаков суммы.
2. Вывести минимальную стоимость книги по годам
3. Вывести максимальную сумму книг по дням за январь 2019 года. Сумму округлить до одного знака после запятой
4. Вывести кол-во книг в каждом месяце если кол-во больше 60

SQL-Урок 9. Подзапросы

До сих пор мы получали данные из базы данных с помощью простых запросов и одного оператора **SELECT**. Однако, все же, чаще нам нужно будет выбирать данные, соответствующие многим условиям, и здесь не обойтись без расширенных запросов. Для этого в **SQL** существуют подзапросы или вложенные подзапросы, когда один оператор **SELECT** укладывается в другой.

1. Фильтрация с помощью подзапросов

Таблицы баз данных, которые используются в СУБД **Sqlite** являются реляционными таблицами, т.е. все таблицы можно связать между собой по общим полям. Допустим у нас хранятся данные в двух разных таблицах и нам нужно выбрать данные в одной из них, в зависимости от того, какие данные в другой. Для этого создадим еще одну таблицу в нашей базе данных. Это будет, например, таблица **Sellers** с информацией о поставщиках:

ID	Address	City	Seller_name
1	396780, г. Старая Кулатка, ул. Купавенский	Донецк	Иван Степко
2	412402, г. Богородское, ул. Вокзальная Площадь	Докучаевск	Воронина Тереза
3	347435, г. Домново, ул. Новоспасский Переулок	Дебальцево	Арджеванидзе Лаврентий
4	391825, г. Боровичи, ул. Ижорская	Горловка	Игнатъев Фадей
5	446966, г. Ставрополь, ул. Селигерская	Енакиево	Козлов Руслан
6	164182, г. Покровское, ул. Берников Переулок	Кировское	Мионова Элиза
7	632715, г. Ольховатка, ул. Колобовский 3-й Переулок	Шахтёрск	Леонов Ким
8	98530, г. Советск, ул. Марьиной Роци 12-й Проезд	Макеевка	Баландин Ким

Теперь мы имеем две таблицы - **Sumproduct** и **Sellers**, которые имеют одинаковое поле **City**. Предположим, нам нужно посчитать сколько товаров было продано только конкретным поставщиком. Сделать это нам помогут подзапросы. Итак, сначала напишем запрос для выборки городов где есть поставщик

```
SELECT City FROM Sellers WHERE Seller_name = 'Иван Степко'
```

City
1 Донецк
2 Горловка

Теперь передадим эти данные в следующий запрос, который будет выбирать данные из таблицы **Sumproduct**:

```
SELECT SUM(Quantity) FROM Sumproduct WHERE City IN ('Донецк','Горловка')
```

SUM(Quantity)
1 187802

Также мы можем объединить эти два запроса в один. Таким образом, один запрос, который выводит данные будет главным, а второй запрос, который передает входные данные, будет вспомогательным (подзапросом). Для вложения подзапроса используем конструкцию **WHERE ... IN (...)**

```
SELECT SUM(Quantity) FROM Sumproduct WHERE City IN (SELECT City FROM Sellers WHERE Seller_name = 'Иван Степко')
```

2. Использование подзапросов в качестве расчетных полей

Мы также можем использовать подзапросы в качестве расчетных полей. Отразим, например, количество реализованной продукции по каждому продавцу с помощью следующего запроса:

```
SELECT Seller_name, (SELECT SUM(Quantity) FROM Sumproduct WHERE Sellers.City = Sumproduct.City) AS Qty FROM Sellers
```

	Seller_name	Qty
1	Иван Степко	75299
2	Воронина Тереза	135118
3	Арджеванидзе Лаврентий	91241
4	Игнатъев Фадей	112503
5	Козлов Руслан	93419
6	Миронова Элиза	104438
7	Леонов Ким	92922
8	Баландин Ким	119242
9	Иван Степко	112503

Первый оператор **SELECT** отражает два столбца - **Seller_name** и **Qty**. Поле **Qty** является расчетным, оно формируется в результате выполнения подзапроса, который взят в круглые скобки. Этот подзапрос выполняется по одному разу для каждой записи в поле **Seller_name** и в общем будет выполнен девять раз, поскольку выбрано имена девяти продавцов.

Также в подзапросе, предложение **WHERE** выполняет функцию объединения, поскольку с помощью **WHERE** мы соединили две таблицы по полю **City**, используя полные названия столбцов

Задание :

1. Вывести сумму издателя 'Usborne Publishing'.
2. Посчитать сумму по издателям с названием 'Book Works', 'Robson Books' и 'Leaf Books'. Отсортировать сумму по возрастанию
3. Вывести максимальную сумму по каждому издателя используя подзапрос. Поля: имя издателя, сумма - до двух знаков после запятой. Отсортировать по убыванию
4. Найти издателя Polity. Показать минимальную и максимальную сумму по книгам, округлить до двух знаков после запятой. Отсортировать по названию издателя.

SQL-Урок 10. Объединение таблиц (INNER JOIN)

Наиболее мощной особенностью языка SQL есть возможность сочетать различные таблицы в оперативной памяти СУБД при выполнении запросов. Объединение очень часто используется для анализа данных. Как правило, данные находятся в разных таблицах, что позволяет их более эффективно хранить (поскольку информация НЕ дублируется), упрощает обработку данных и позволяет масштабировать базу данных (возможно добавлять новые таблицы с дополнительной информацией). Таблицы баз данных, которые используются в СУБД **Sqlite** являются реляционными таблицами, т.е. все таблицы можно связать между собой по общим полям.

1. Создание объединения таблиц

Объединение таблиц очень простая процедура. Нужно указать все таблицы, которые будут включены в объединение и "объяснить" СУБД, как они будут связаны между собой. Объединение делается с помощью слова **WHERE**, например:

```
SELECT DISTINCT Seller_name, Product FROM Sellers, Sumproduct WHERE Sellers.City = Sumproduct.City
```

	Seller_name	Product
1	Иван Степко	Апельсины
2	Иван Степко	Бананы
3	Иван Степко	Груши
4	Иван Степко	Йогурт
5	Иван Степко	Капуста
6	Иван Степко	Карамель
7	Иван Степко	Картофель
8	Иван Степко	Кефир
9	Иван Степко	Кофе
10	Иван Степко	Лук
11	Иван Степко	Мандарины
12	Иван Степко	Молоко
13	Иван Степко	Морковь
14	Иван Степко	Мороженое








DISTINCT исключает повторяющиеся/дублирующиеся **СТРОКИ** в результате выборки данных.

Соединив две таблицы, мы смогли увидеть какие товары реализует каждый продавец. Рассмотрим код запроса подробнее, поскольку он немного отличается от обычного запроса. Оператор **SELECT** начинается с указанием столбцов, которые мы хотим вывести, однако эти поля находятся в разных таблицах, предложение **FROM** содержит две таблицы, которые мы хотим объединить в операторе **SELECT**, таблицы объединяются с помощью слова **WHERE**, указывающее столбцы для объединения. Обязательно нужно указывать полное название поля (*Таблица.Поле*), поскольку поле **City** есть в обеих таблицах.

2. Внутреннее объединение

В предыдущем примере для объединения таблиц мы использовали слово **WHERE**, которое осуществляет проверку на основе эквивалентности двух таблиц. Объединение такого типа называется также "*внутренним объединением*". Существует также и другой способ объединения таблиц, который явно указывает на тип объединения. Рассмотрим следующий пример:

```
SELECT DISTINCT Seller_name, Product FROM Sellers INNER JOIN Sumproduct ON Sellers.City = Sumproduct.City
```


Табличный вид		Форма
      		Всего загружено строк: 248
	Seller_name	Product
1	Иван Степко	Апельсины
2	Иван Степко	Бананы
3	Иван Степко	Груши
4	Иван Степко	Йогурт
5	Иван Степко	Капуста
6	Иван Степко	Карамель
7	Иван Степко	Картофель
8	Иван Степко	Кефир

В этом запросе вместо **WHERE** мы использовали конструкцию **INNER JOIN ... ON ...**, которая дала аналогичный результат. Несмотря на то, что объединение с предложением **WHERE** короче, все же лучше использовать **INNER JOIN**, поскольку она является более гибкой

SQL-Урок 11. Расширенное объединение таблиц (OUTER JOIN)

!!! RIGHT JOIN только SQL Lite не поддерживает

В предыдущем разделе мы рассмотрели самые простые способы объединения таблиц - с помощью предложений **WHERE** и **INNER JOIN**. Эти объединения называются внутренними объединениями или объединениями по эквивалентности. Однако SQL имеет в своем арсенале гораздо больше возможностей объединить таблицы, а именно существуют также и другие виды объединений: внешние объединения, природные объединения и самообъединения. Но для начала рассмотрим, каким образом мы можем присваивать таблицам псевдонимы, поскольку в дальнейшем, мы будем вынуждены использовать полные названия полей, которыми без сокращений будет очень трудно оперировать из-за их большой длины.

1. Использование псевдонимов таблиц

В предыдущем разделе мы узнали, как можно использовать псевдонимы для ссылки на определенные поля таблицы или на расчетные поля. SQL так же дает нам возможность использовать псевдонимы вместо имен таблиц. Это дает нам такие преимущества, как более короткий синтаксис SQL и позволяет много раз использовать одну и ту же таблицу в операторе **SELECT**.

```
SELECT Seller_name, SUM(Amount) AS Sum1
FROM Sellers AS S, Sumproduct AS SP
WHERE S.City = SP.City
GROUP BY Seller_name
```

	Seller_name	Sum1
1	Арджеванидзе Лаврентий	100706.80000000002
2	Баландин Ким	137236.98
3	Воронина Тереза	145511.87000000005
4	Иван Степко	208840.16999999993
5	Игнатъев Фадей	125353.27999999997
6	Козлов Руслан	112723.83000000007
7	Леонов Ким	102031.12
8	Миронова Элиза	120428.40999999997

Мы отобразили общую сумму реализованного товара по каждому продавцу. В нашем SQL запросе мы использовали такие псевдонимы: для расчетного поля **SUM (Amount)** псевдоним *Sum1* для таблицы **Sellers** псевдоним *S* и для **Sumproduct** псевдоним *SP*. Заметим, что псевдонимы таблиц могут быть применены и в других предложениях, как **ORDER BY**, **GROUP BY** и других.

2. Самообъединения

Рассмотрим пример. Предположим, нам нужно узнать продукты которые есть в том городе что и конкретная сумма. Для этого создадим такой запрос:

```
SELECT * FROM Sumproduct WHERE City = (SELECT City FROM Sumproduct WHERE Amount = '924.12')
```

	ID	Month	Product	City	Quantity	Amount
1	1	Январь	Шоколадные кофеты	Горловка	216	924.12
2	2	Январь	Молоко	Горловка	915	747.67
3	3	Январь	Шоколад	Горловка	249	368.94
4	4	Январь	Соль	Горловка	574	473.52
5	5	Январь	Сахар	Горловка	665	632.56
6	6	Январь	Йогурт	Горловка	380	257.47
7	7	Январь	Помидоры	Горловка	132	298.8
8	8	Январь	Рыба	Горловка	913	455.44

Также, эту задачу мы можем решить и через самообъединения, прописав следующий код:

```
SELECT S1.* FROM Sumproduct S1, Sumproduct S2
WHERE S1.City = S2.City AND S2.Amount = '924.12'
```

	ID	Month	Product	City	Quantity	Amount
1	2073	Август	Апельсины	Горловка	729	502.68
2	2083	Август	Бананы	Горловка	632	618.74
3	2088	Август	Груши	Горловка	430	567.17
4	2092	Август	Йогурт	Горловка	573	284.7
5	2095	Август	Капуста	Горловка	532	563.66
6	2097	Август	Картофель	Горловка	236	204.51
7	2082	Август	Кефир	Горловка	913	600.55

Для решения этой задачи использовались псевдонимы. Первый раз для таблицы Sumproduct

присвоили псевдоним **S1**, второй раз - псевдоним **S2**. После этого эти псевдонимы можно применять в качестве имен таблиц. В операторе **WHERE** мы с названием каждого поля добавляем префикс **S1**, для того, чтобы СУБД понимала поля которой таблицы нужно выводить (поскольку мы из одной таблицы сделали две виртуальные). Предложение **WHERE** сначала объединяет таблицы, а затем фильтрует данные второй таблицы по полю **Amount**, чтобы вернуть только необходимые значения.

Самообъединения часто используют для замены подзапросов, которые выбирают данные из той же таблицы, что и внешний оператор **SELECT**. Хотя конечный результат получается тем самым, многие СУБД обрабатывают объединения гораздо быстрее подзапросов. Стоит поэкспериментировать, чтобы определить, какой запрос работает быстрее.

3. Естественное объединение

Естественное объединение - это объединение, в котором вы выбираете только те столбцы, которые не повторяются. Обычно это делается с помощью записи (**SELECT ***) для одной таблицы и указанием перечня полей - для остальных таблиц. Например:

```
SELECT SP.*, S.Seller_name
FROM Sumproduct AS SP, Sellers AS S
WHERE SP.City = S.City
```

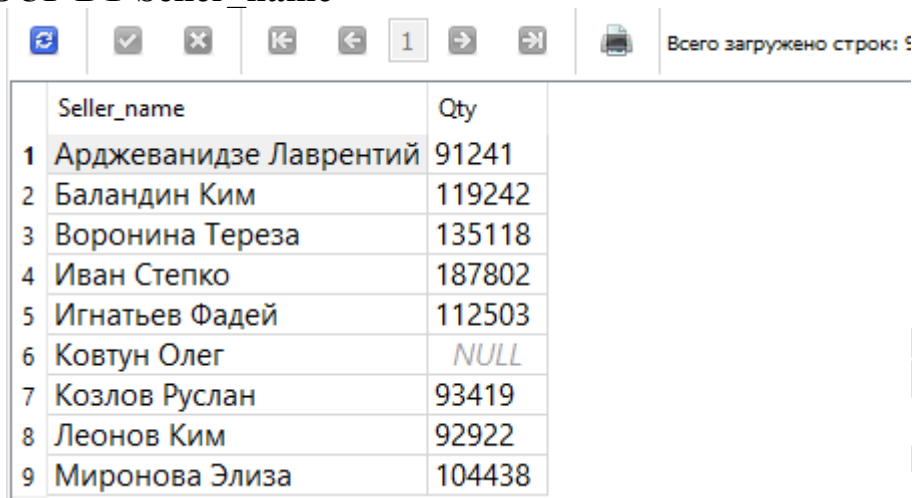
	ID	Month	Product	City	Quantity	Amount	Seller_name
1	1	Январь	Шоколадные кофеты	Горловка	216	924.12	Иван Степко
2	1	Январь	Шоколадные кофеты	Горловка	216	924.12	Игнатъев Фадей
3	2	Январь	Молоко	Горловка	915	747.67	Иван Степко
4	2	Январь	Молоко	Горловка	915	747.67	Игнатъев Фадей
5	3	Январь	Шоколад	Горловка	249	368.94	Иван Степко
6	3	Январь	Шоколад	Горловка	249	368.94	Игнатъев Фадей
7	4	Январь	Соль	Горловка	574	473.52	Иван Степко

4. Внешнее объединение (OUTER JOIN)

Обычно при объединении связывают строки одной таблицы с соответствующими строками другой, однако в некоторых случаях может потребоваться включить в результат строки, не имеющие связанных строк в другой таблице (т.е. выбираются совершенно все строки из одной таблицы и добавляются только связанные строки из другой). Объединение такого типа называется внешним. Для этого используются ключевые слова **OUTER JOIN ... ON ...** с приставкой **LEFT** или **RIGHT**. Рассмотрим

пример, предварительно добавив в таблицу **Sellers** нового продавца - *Ковтун Олег*, который не имеет продаж:

```
SELECT Seller_name, SUM(Quantity) AS Qty
FROM Sellers LEFT OUTER JOIN Sumproduct ON Sellers.City=Sumproduct.City
GROUP BY Seller_name
```



	Seller_name	Qty
1	Арджеванидзе Лаврентий	91241
2	Баландин Ким	119242
3	Воронина Тереза	135118
4	Иван Степко	187802
5	Игнатъев Фадей	112503
6	Ковтун Олег	NULL
7	Козлов Руслан	93419
8	Леонов Ким	92922
9	Миронова Элиза	104438

Данным запросом мы вытащили перечень всех продавцов в базе и подсчитали для них общее количество проданного товара за все месяцы. Видим что по новому продавцу отсутствуют продажи. Если бы мы использовали внутреннее объединение, то нового продавца мы бы не увидели, поскольку он не имеет записей в таблице **Sumproduct**. Мы можем изменять направление объединения не только прописывая **LEFT** или **RIGHT**, но и просто меняя порядок таблиц (т. е. две записи будут давать одинаковый результат: **Sellers LEFT OUTER JOIN Sumproduct** та **Sumproduct RIGHT OUTER JOIN Sellers**).

Задание :

ИСПОЛЬЗОВАТЬ ТОЛЬКО ЧЕРЕЗ LEFT JOIN

- 1. Выбрать издателей с книгами и отсортировать по издателям, которые не имеют книг в первую очередь**
- 2. Выбрать издателей с книгами и показать издателей, у которых нет книг**
- 3. Выбрать издателей с книгами, показать издателей у которых есть книги, вывести количество авторов у каждой книги. Книги должны быть с описанием.**

Авторов должно быть не больше двух у книги. Имена авторов содержит 'Angela' и добавить издателей у которых нет книг вообще.

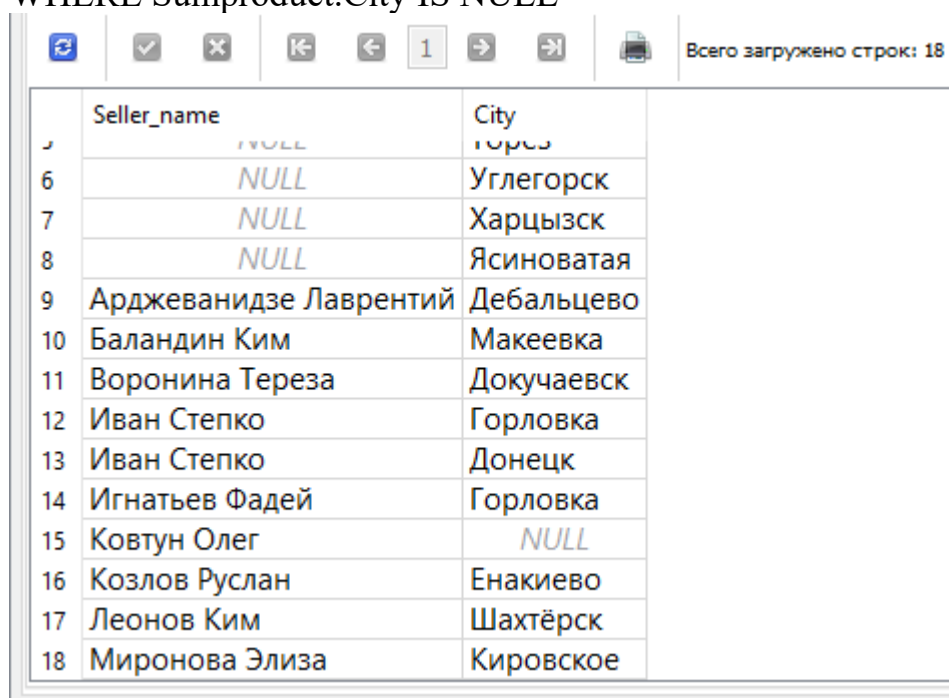
SQL-Урок 12. Комбинированные запросы (UNION)

В большинстве **SQL-запросов** используется один оператор, с помощью которого возвращаются данные из одной или нескольких таблиц. **SQL** также позволяет выполнять одновременно несколько отдельных запросов и отображать результат в виде единого набора данных. Такие комбинированные запросы обычно называют *сочетаниями* или *сложными запросами*.

1. Использование оператора UNION

Запросы в языке **SQL** комбинируются с помощью оператора **UNION**. Для этого необходимо указать каждый запрос **SELECT** и разместить между ними ключевое слово **UNION**. Ограничений по количеству использованного оператора **UNION** в одном общем запросе нет. В предыдущем разделе мы отмечали, что **Access** не имеет возможности создавать *полное внешнее объединение*, теперь мы посмотрим, как можно этого достичь через оператор **UNION**.

```
SELECT Sellers.Seller_name, Sumproduct.City
FROM Sumproduct LEFT JOIN Sellers ON Sumproduct.City = Sellers.City
UNION
SELECT Sellers.Seller_name, Sumproduct.City
FROM Sellers LEFT JOIN Sumproduct ON Sumproduct.City = Sellers.City
WHERE Sumproduct.City IS NULL
```



	Seller_name	City
5	NULL	Горловка
6	NULL	Углегорск
7	NULL	Харцызск
8	NULL	Ясиноватая
9	Арджеванидзе Лаврентий	Дебальцево
10	Баландин Ким	Макеевка
11	Воронина Тереза	Докучаевск
12	Иван Степко	Горловка
13	Иван Степко	Донецк
14	Игнатъев Фадей	Горловка
15	Ковтун Олег	NULL
16	Козлов Руслан	Енакиево
17	Леонов Ким	Шахтёрск
18	Миронова Элиза	Кировское

Видим, что запрос отобразил как все колонки из первой таблицы - так и с другой, независимо от того, все ли записи имеют соответствия в другой таблице.

Также стоит отметить, что во многих случаях вместо **UNION** мы можем использовать предложение **WHERE** со многими условиями, и получать аналогичный результат. Однако из-за **UNION** записи выглядят более лаконичными и понятными. Также необходимо соблюдать определенные правила при написании комбинированных запросов:

- запрос **UNION** должен включать два и более операторов **SELECT**, отделенных между собой ключевым словом **UNION** (т.е. если в запросе используется четыре оператора **SELECT**, то должно быть три ключевых слова **UNION**)
- каждый запрос в операторе **UNION** должен иметь одни и те же столбцы, выражения или статистические функции, которые, к тому же, должны быть перечислены в одинаковом порядке
- типы данных столбцов должны быть совместимыми. Они не обязательно должны быть одного типа, однако обязаны иметь подобный тип, чтобы **СУБД** могла их однозначно преобразовать (например, это могут быть различные числовые типы данных или различные типы даты).

2. Включение или выключение повторяющихся строк

Запрос с **UNION** автоматически удаляет все повторяющиеся строки из набора результатов запроса (то есть, ведет себя как предложения **WHERE** с несколькими условиями в одном операторе **SELECT**). Такое поведение оператора **UNION** по умолчанию, но при желании мы можем изменить это. Для этого нам следует использовать оператор **UNION ALL** вместо **UNION**.

3. Сортировка результатов комбинированных запросов

Результаты выполнения оператора **SELECT** сортируются с помощью предложения **ORDER BY**. При комбинировании запросов с помощью **UNION** только одно предложение **ORDER BY** может быть использовано, и оно должно быть проставлено в последнем операторе **SELECT**. Действительно, на практике нет особого смысла часть результатов сортировать в одном порядке, а другую часть - в другом. Поэтому несколько предложений **ORDER BY** применять не разрешается.

SQL-Урок 16. Добавление данных (INSERT INTO)

В предыдущих разделах мы рассматривали работу по получению данных с заранее созданных таблиц. Теперь пора разобраться, каким же образом мы можем создавать/удалять таблицы, добавлять новые записи и удалять старые. Для этих целей в **SQL** существуют такие операторы, как: **CREATE** - создает таблицу, **ALTER** - изменяет структуру таблицы, **DROP** - удаляет таблицу или поле, **INSERT** - добавляет данные в таблицу. Начнем знакомство с данной группой операторов из оператора **INSERT**.

1. Добавление целых строк

Как видно из названия, оператор **INSERT** используется для вставки (добавления) строк в таблицу базы данных. Добавление можно осуществить несколькими способами:

- - добавить одну полную строку
- - добавить часть строки
- - добавить результаты запроса.

Итак, чтобы добавить новую строку в таблицу, нам необходимо указать название таблицы, перечислить названия колонок и указать значение для каждой колонки с помощью конструкции **INSERT INTO название_таблицы (поле1, поле2 ...) VALUES(значение1, значение2 ...)**. Рассмотрим на примере.

INSERT INTO Sellers (ID, Address, City, Seller_name) VALUES ('12', ' 455034, г. Лянтор, ул. Комиссариатский Переулок', 'Донецк', ' Щербакова Берта')

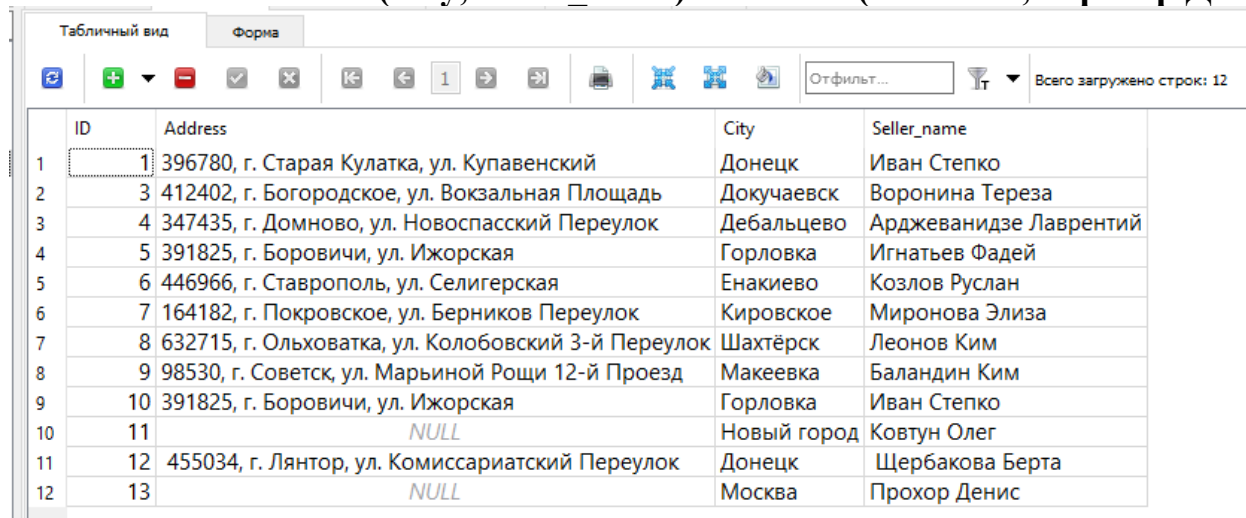
	ID	Address	City	Seller_name
1	1	396780, г. Старая Кулатка, ул. Купавенский	Донецк	Иван Степко
2	3	412402, г. Богородское, ул. Вокзальная Площадь	Докучаевск	Воронина Тереза
3	4	347435, г. Домново, ул. Новоспасский Переулок	Дебальцево	Арджеванидзе Лаврентий
4	5	391825, г. Боровичи, ул. Ижорская	Горловка	Игнатъев Фадей
5	6	446966, г. Ставрополь, ул. Селигерская	Енакиево	Козлов Руслан
6	7	164182, г. Покровское, ул. Берников Переулок	Кировское	Миронова Элиза
7	8	632715, г. Ольховатка, ул. Колобовский 3-й Переулок	Шахтёрск	Леонов Ким
8	9	98530, г. Советск, ул. Марьиной Рощи 12-й Проезд	Макеевка	Баландин Ким
9	10	391825, г. Боровичи, ул. Ижорская	Горловка	Иван Степко
10	11	NULL	Новый город	Ковтун Олег
11	12	455034, г. Лянтор, ул. Комиссариатский Переулок	Донецк	Щербакова Берта

Также можно изменять порядок указания названий колонок, однако одновременно нужно менять и порядок значений в параметре **VALUES**.

2. Добавление части строк

В предыдущем примере при использовании оператора **INSERT** мы явно отмечали имена столбцов таблицы. Используя данный синтаксис, мы можем пропустить некоторые столбцы. Это значит, что вы вводите значение для одних столбцов но не предлагаете их для других. Например:

INSERT INTO Sellers (City, Seller_name) VALUES ('Москва', 'Проход Денис')



ID	Address	City	Seller_name
1	396780, г. Старая Кулатка, ул. Купавенский	Донецк	Иван Степко
2	412402, г. Богородское, ул. Вокзальная Площадь	Донецк	Воронина Тереза
3	347435, г. Домново, ул. Новоспасский Переулок	Дебальцево	Арджеванидзе Лаврентий
4	391825, г. Боровичи, ул. Ижорская	Горловка	Игнатъев Фадей
5	446966, г. Ставрополь, ул. Селигерская	Енакиево	Козлов Руслан
6	164182, г. Покровское, ул. Берников Переулок	Кировское	Мионова Элиза
7	632715, г. Ольховатка, ул. Колобовский 3-й Переулок	Шахтёрск	Леонов Ким
8	98530, г. Советск, ул. Марьиной Роци 12-й Проезд	Макеевка	Баландин Ким
9	391825, г. Боровичи, ул. Ижорская	Горловка	Иван Степко
10	11	Новый город	Ковтун Олег
11	455034, г. Лянтор, ул. Комиссариатский Переулок	Донецк	Щербакова Берта
12	13	Москва	Проход Денис

В данном примере мы не указали значение поле адрес и id. Вы можете исключать некоторые столбцы из оператора **INSERT INTO**, если это позволяет производить определение таблицы. В этом случае должно соблюдаться одно из условий: этот столбец определен как допускающий значение **NULL** (отсутствие какого-либо значения) или в определение таблицы указанное значение по умолчанию. Это означает, что, если не указано никакое значение, будет использовано значение по умолчанию. Если вы пропускаете столбец таблицы, которая не допускает появления в своих строках значений **NULL** и не имеет значения, определенного для использования по умолчанию, СУБД выдаст сообщение об ошибке, и эта строка не будет добавлена. Поле id является автоинкрементированным поэтому его не указали

3. Добавление отобранных данных

В предыдущей примерах мы вставляли данные в таблицы, прописывая их вручную в запросе. Однако оператор **INSERT INTO** позволяет автоматизировать этот процесс, если мы хотим вставлять данные из другой таблицы. Для этого в SQL существует такая конструкция как **INSERT INTO ... SELECT ...**. Данная конструкция позволяет одновременно выбирать данные из одной таблицы, и вставить их в другую. Предположим мы имеем еще одну таблицу **Sellers_EU** с перечнем продавцов нашего товара в Европе и нам нужно их добавить в общую таблицу **Sellers**. Структура этих таблиц одинакова (то же количество колонок и те же их названия), однако другие данные. Для этого мы можем прописать следующий запрос:

INSERT INTO Sellers (ID, Address, City, Seller_name, Country) SELECT ID, Address, City, Seller_name, Country FROM Sellers_EU

Нужно обратить внимание, чтобы значение внутренних ключей не повторялись (поле **ID**), в противном случае произойдет ошибка. Оператор **SELECT** также может включать предложения **WHERE** для фильтрации данных. Также следует отметить, что СУБД не обращает внимания на названия колонок, которые содержатся в операторе **SELECT**, для нее важно только порядок их расположения. Поэтому данные в первом

указанном столбце, что были выбраны из-за **SELECT**, будут в любом случае заполнены в первый столбец таблицы **Sellers**, указанной после оператора **INSERT INTO**, независимо от названия поля.

SQL-Урок 17-18. Создание таблиц (CREATE TABLE)

Язык **SQL** используется не только для обработки информации, но и предназначена для выполнения всех операций с базами данных и таблицами, включая также создание таблиц и работа с ними. Существует два способа создания таблиц: 1) большинство СУБД обладают визуальным интерфейсом для интерактивного создания таблиц и управление ими; 2) таблицами можно манипулировать, используя операторы **SQL**. Стоит отметить, что, когда вы используете интерактивный инструментарий СУБД, на самом деле вся работа выполняется операторами **SQL**, т.е. интерфейс сам создает эти команды незаметно для пользователя (это подобно на запись макроса в **Excel**, когда макрорекодер записывает ваши действия и преобразует их в команды **VBA**).

1. Создание таблиц

Для создания таблиц программным способом используют оператор **CREATE TABLE**. Для этого нужно указать следующие данные:

```
CREATE TABLE [IF NOT EXISTS] table_name (  
    column_1 data_type PRIMARY KEY,  
    column_2 data_type NOT NULL,  
    column_3 data_type DEFAULT 0,  
    table_constraint  
) [WITHOUT ROWID];
```

Так мы сначала указываем название новой таблицы, затем в скобках перечисляем столбцы, которые будем создавать, причем их названия не могут повторяться в пределах одной таблицы. После названий столбцов указывается тип данных для каждого поля, затем отмечаем может ли поле содержать пустые значения (**NULL** или **NOT NULL**), а также нужно указать поле, которое будет первичным ключом (**Primary key**).

Язык **SQL** также позволяет определять для каждого поля значение по умолчанию, то есть, если пользователь не укажет значение определенного поля - оно будет автоматически проставлено СУБД. Значение по умолчанию определяется ключевым словом **DEFAULT** при определении столбцов оператором **CREATE TABLE**.

По умолчанию, строка в таблице имеет неявный столбец **WITHOUT ROWID**, который может быть передан в качестве `rowid`, `oid` или `rowid` столбца. В `rowid` столбце хранится 64-разрядный целочисленный ключ со знаком, который однозначно идентифицирует строку в таблице. Если вы не хотите, чтобы **SQLite** создавал `rowid` столбец, вы можете указать эту **WITHOUT ROWID** опцию в **CREATE TABLE** операторе.

IF NOT EXISTS позволяет игнорировать созданию таблицы если такая уже существует

Классы данных вместо типов данных в SQLite3

SQLite3 вместо типов данных использует классы данных. **Классы данных в SQLite3** – это более широкое понятие, нежели тип. Любое значение в SQLite3 может иметь один из пяти классов:

- **NULL** – пустое значение или его отсутствие. Значение NULL в SQLite3, как и во многих других СУБД и языках программирования уникально, то есть оно не равно ни другому NULL ни еще какому-либо значению. На NULL можно только проверить;
- **INTEGER** – класс данных INTEGER в SQLite3 используется для хранения целочисленных значений, которые хранятся в 1, 2, 3, 4, 6 или 8 байтах, в зависимости от самого значения;
- **REAL** – класс данных REAL в SQLite3 предназначен для хранения вещественных чисел, данные в классе REAL хранятся в формате восьмибайтного числа IEEE с плавающей точкой (вспомните школьную математику, где вам рассказывали про мантиссу и порядок);
- **TEXT** – класс данных TEXT в SQLite3 используется для хранения строковых значений, в базе данных данные с классом TEXT хранятся с использованием кодировки UTF-8 или UTF-16 (это можно настроить, рекомендуем использовать UTF-8);
- **BLOB** – класс данных BLOB в SQLite3 используется для хранения бинарных данных, данные с классом BLOB хранятся в том виде, в котором они были введены.

Все остальные типы данных столбца приводятся к 5 типам данных описанных выше. Они были созданы для совместимости с другими СУБД.

Тип данных столбца, который используется при создании таблицы	Аффинированный тип данных
INT	INTEGER
INTEGER	
TINYINT	
SMALLINT	
MEDIUMINT	
BIGINT	
UNSIGNED BIG INT	
INT2	
INT8	
CHARACTER (20)	TEXT
VARCHAR (255)	
VARYING CHARACTER (255)	
NCHAR (55)	
NATIVE CHARACTER (70)	
NVARCHAR (100)	
TEXT	
CLOB	
BLOB	BLOB
<i>без явного указания типа данных</i>	

REAL	REAL
DOUBLE	
DOUBLE PRECISION	
FLOAT	
NUMERIC	NUMERIC
DECIMAL (10,5)	
BOOLEAN	
DATE	
DATETIME	

Создадим таблицу:

```
CREATE TABLE groups (
group_id integer PRIMARY KEY,
name text NOT NULL
);
```

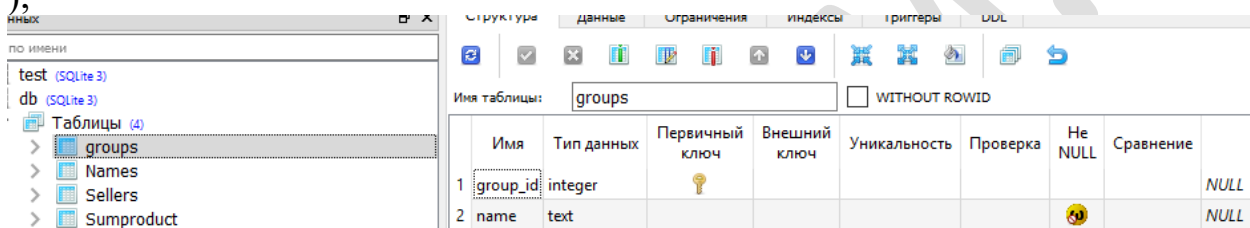
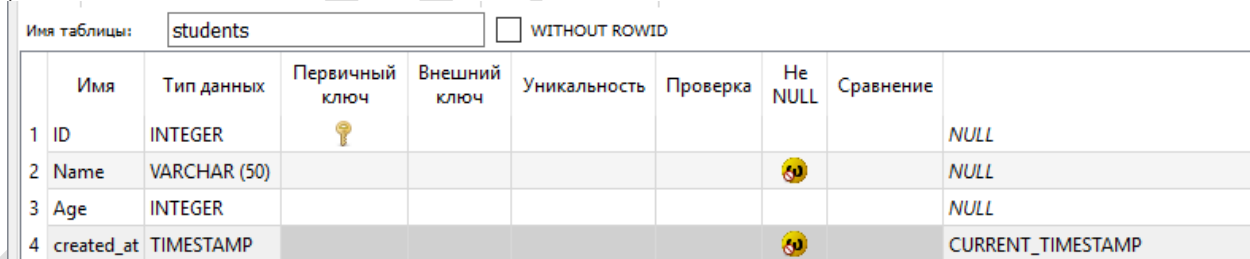


Таблица называется groups. Содержит поля primary key group_id с типом integer как индикатор строки в таблице и name типа text для записи названия группы.

```
CREATE TABLE students (
ID INTEGER PRIMARY KEY AUTOINCREMENT,
Name VARCHAR (50) NOT NULL,
Age INTEGER,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL
);
```



В таблице students ID имеет AUTOINCREMENT. Поле Age можно не указывать т.к. NOT NULL нет. Тип TIMESTAMP по умолчанию CURRENT_TIMESTAMP для поля created_at позволяет при создании записи автоматически подставлять текущее время.

2. Обновление и удаление таблиц

Для того, чтобы изменить таблицу в SQL используется оператор ALTER TABLE. При использовании данного оператора необходимо ввести следующую информацию:

- - имя таблицы, которую мы хотим изменить
- - перечень изменений, которые мы хотим сделать.

Для примера давайте добавим новую колонку в таблицу Sellers, в которой будем указывать телефон реализатора:

```
ALTER TABLE Sellers ADD Phone CHAR (20)
```

ID	Address	City	Seller_name	Phone
1	396780, г. Старая Кулатка, ул. Купавенский	Донецк	Иван Степко	NULL
2	3 412402, г. Богородское, ул. Вокзальная Площадь	Докучаевск	Воронина Тереза	NULL
3	4 347435, г. Домново, ул. Новоспасский Переулок	Дебальцево	Арджеванидзе Лаврентий	NULL
4	5 391825, г. Боровичи, ул. Ижорская	Горловка	Игнатъев Фадей	NULL
5	6 446966, г. Ставрополь, ул. Селигерская	Енакиево	Козлов Руслан	NULL
6	7 164182, г. Покровское, ул. Берников Переулок	Кировское	Миронова Элиза	NULL

Кроме добавления столбцов, мы можем их удалять. Давайте теперь удалим поле **Phone**. Для этого пропишем следующий запрос:

```
ALTER TABLE Sellers DROP COLUMN Phone
```

Это работает для большинства СУБД. Но не для SQLite. Есть другой путь:

```
CREATE TABLE sqlitestudio_temp_table AS SELECT *
```

```
FROM Sellers;
```

```
DROP TABLE Sellers;
```

```
CREATE TABLE Sellers (
```

```
ID INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
Address VARCHAR (100),
```

```
City VARCHAR (50),
```

```
Seller_name VARCHAR (50)
```

```
);
```

```
INSERT INTO Sellers (
```

```
ID,
```

```
Address,
```

```
City,
```

```
Seller_name
```

```
)
```

```
SELECT ID,
```

```
Address,
```

```
City,
```

```
Seller_name
```

```
FROM sqlitestudio_temp_table;
```

```
DROP TABLE sqlitestudio_temp_table;
```

Для начала нужно создать временную таблицу и перенести все данные из Sellers, удалить таблицу Sellers и создать новую с нужными полями. После создания добавить все записи из временной таблицы и удалить ее. Где **DROP TABLE** Название таблицы – удаляет таблицу из базы данных.