Глеб Зайцев

SQL на практике. 60 задач с решениями на данных магазина

Глеб Зайцев

SQL на практике. 60 задач с решениями на данных магазина

«Автор»
2026

Зайцев Г.

SQL на практике. 60 задач с решениями на данных магазина / Г. Зайцев — «Автор», 2026

Практикум для тех, кто знает основы SELECT и хочет научиться собирать запрос из условия задачи. 60 упражнений на одной учебной SQLite-базе магазина: покупатели, товары, заказы, позиции заказов и платежи.Шесть глав: фильтрация, агрегирование, JOIN, подзапросы, CTE и оконные функции. Сначала — задания; затем — подсказки, решения, ожидаемое число строк и пример результата. Отдельно разобраны регистр кириллицы, расхождения между заказом и оплатой, пропущенные даты и отличие трёх строк от трёх календарных дней.Полный код создания базы и инструмент запуска запросов включены в книгу. Для практики нужен компьютер с Python 3 и встроенным модулем sqlite3. Отдельный архив, платный сервис или аккаунт не требуются. Данные синтетические; это учебные упражнения, а не готовая финансовая отчётность.

Содержание

Глеб Зайцев
SQL на практике. 60 задач с решениями на данных магазина

SQL на практике

60 задач с решениями на данных магазина

Глеб Зайцев

SQLite · самостоятельная практика

Первое книжное издание 2026

Для выполнения заданий нужен компьютер с Python 3 и модулем sqlite3. Весь код создания учебной базы включён в книгу. Отдельный архив, платный сервис и доступ к чужому серверу не требуются.

Попробуйте до покупки: почему JOIN увеличил сумму заказа

Сначала маленький результат

Вы уже знаете SELECT, но после соединения таблиц получаете неожиданную сумму? Разберём один такой случай на трёх учебных заказах. Для этого примера не нужны приложение в конце книги, скачанная база или регистрация в сервисе. Достаточно Python 3 со встроенным модулем sqlite3 на компьютере.

В таблице заказов одна строка означает один заказ. В таблице позиций строка означает одну позицию заказа, которая может включать несколько единиц товара. У первого заказа две позиции, поэтому после JOIN его полная сумма встречается дважды. SQL не ошибается: мы сложили значения на неподходящем уровне детализации.

Учебные данные:

Заказ: 1; Статус: paid; Полная сумма, #: 1000; Количество позиций: 2.

Заказ: 2; Статус: paid; Полная сумма, #: 600; Количество позиций: 1.

Заказ: 3; Статус: cancelled; Полная сумма, #: 900; Количество позиций: 1.

Задача — найти сумму оплаченных заказов, у которых есть хотя бы одна позиция. Правильный результат: 1000 + 600 = 1600 #. Отменённый заказ не включаем. Статусы и суммы здесь вымышлены; мы тренируем запрос, а не составляем бухгалтерский отчёт.

Запустите пример целиком

Сохраните следующий блок в обычный текстовый файл `join_demo.py` в кодировке UTF-8. Если при копировании из читалки в начале строк появились пробелы, в том числе неразрывные, удалите их: каждая строка этого примера может начинаться с первого непробельного символа. В этом конкретном примере нет Python-блоков с обязательными отступами. Пробелы внутри строк и переносы строк сохраняйте; кавычки должны быть прямыми.

Откройте терминал в папке, куда сохранён файл, и выполните `python join_demo.py`. Если Python у вас запускается командой `python3` или `py`, замените только первое слово. База создаётся в памяти и исчезает после завершения программы. Файлы базы и сетевые подключения не используются.

```
import sqlite3

db = sqlite3.connect(":memory:")
db.execute("PRAGMA foreign_keys = ON")
db.executescript("""
CREATE TABLE demo_orders (
order_id INTEGER PRIMARY KEY,
status TEXT NOT NULL,
total_rub INTEGER NOT NULL CHECK (total_rub >= 0)
);
CREATE TABLE demo_items (
order_id INTEGER NOT NULL REFERENCES demo_orders(order_id),
line_no INTEGER NOT NULL,
quantity INTEGER NOT NULL CHECK (quantity > 0),
```

```
    unit_price_rub INTEGER NOT NULL CHECK (unit_price_rub >= 0),
    PRIMARY KEY (order_id, line_no)
    );
    INSERT INTO demo_orders VALUES
    (1, 'paid', 1000), (2, 'paid', 600), (3, 'cancelled', 900);
    INSERT INTO demo_items VALUES
    (1, 1, 1, 400), (1, 2, 2, 300),
    (2, 1, 1, 600), (3, 1, 1, 900);
    """)

    wrong_sql = """
    SELECT SUM(o.total_rub) AS paid_total_rub
    FROM demo_orders AS o
    JOIN demo_items AS i ON i.order_id = o.order_id
    WHERE o.status = 'paid';
    """

    correct_sql = """
    SELECT COALESCE(SUM(o.total_rub), 0) AS paid_total_rub
    FROM demo_orders AS o
    WHERE o.status = 'paid'
    AND EXISTS (
    SELECT 1 FROM demo_items AS i
    WHERE i.order_id = o.order_id
    );
    """

    print("После ошибочного JOIN:",
db.execute(wrong_sql).fetchone()[0])
    print("Без повторного учёта:",
db.execute(correct_sql).fetchone()[0])
    db.close()
```

Ожидаемый вывод:

```
После ошибочного JOIN: 2600
Без повторного учёта: 1600
```

Почему второй запрос работает

В первом запросе оплаченные заказы после JOIN дают три строки: две строки заказа 1 и одну строку заказа 2. Поэтому складываются 1000 + 1000 + 600.

Во втором запросе мы остаёмся в таблице заказов, где каждый заказ представлен одной строкой. `EXISTS` проверяет наличие позиции, но не добавляет её к результату как отдельную строку. Сколько бы позиций ни было у заказа, его сумма учитывается один раз. Если подходящих заказов нет, `SUM` возвращает `NULL`; `COALESCE` заменяет его на ноль.

Здесь нельзя лечить ошибку заменой `SUM` на `SUM(DISTINCT o.total_rub)`. Два разных заказа могут стоить одинаково. Если добавить ещё один оплаченный заказ на 1000 # с одной позицией, правильная сумма станет 2600 #, а `SUM(DISTINCT ...)` ошибочно оставит 1600 #: он различает суммы, не заказы.

Проверьте себя

До строки с первым `print` вставьте эти команды и снова запустите файл:

```
db.execute("INSERT INTO demo_orders VALUES (4, 'paid', 1000)")
db.execute("INSERT INTO demo_items VALUES (4, 1, 1, 1000)")
```

Теперь первый запрос даст 3600, второй — 2600. Если получилось иначе, сначала сравните вставленные строки и убедитесь, что команды расположены до вычисления результатов, а не после `db.close()`.

В полном практикуме этот подход развивается на одной базе магазина: условия задачи, самостоятельный запрос, подсказка, решение и контроль результата. Далее идут 60 заданий по выборкам, группировкам, соединениям, датам, подзапросам, CTE и оконным функциям. Полный код основной базы включён в книгу; этот маленький пример от неё не зависит.

Как работать с практикумом

Эта книга поможет потренировать SQL на одной связанной базе интернет-магазина. В ней 60 задач: от отбора товаров до рейтингов и накопительных сумм. Сначала решайте самостоятельно, затем сверяйтесь с подсказкой и эталонным запросом во второй части.

Практикум предназначен для тех, кто уже знаком с таблицами и базовым SELECT. Он не заменяет полный учебник по проектированию баз. Краткие введения объясняют подход к каждой группе задач, а разборы обращают внимание на ошибки в определении метрик.

Все имена, товары, заказы и платежи в наборе учебные. Это не данные реальных покупателей и не выгрузка продавца. Распределения намеренно простые и не отражают рынок. Регистрации и заказы создаются независимо; встречаются заказы раньше регистрации, что отдельно учтено в задании об интервале до первой покупки.

Задачи не требуют менять исходные строки. Сохраните свой запрос в отдельный файл answer.sql и выполняйте его только на учебной базе. Приведённая в приложении команда открывает базу в режиме чтения.

Примеры рассчитаны на диалект SQLite. Перенос на PostgreSQL, MySQL или другую СУБД потребует адаптации функций дат и некоторых правил группировки. Все 60 эталонов проверены на SQLite 3.53.1. Это техническая проверка примеров, а не обещание трудоустройства или уровня квалификации.

Содержание

Подготовка учебной базы

Создайте отдельную пустую папку для практикума. В текстовом редакторе создайте в ней файл create_shoplab.py в кодировке UTF-8. Скопируйте в него весь код из приложения А: от первой строки импорта до последнего print. Не включайте заголовки книги и номера страниц. Сохраняйте обычные прямые кавычки и переносы строк. В этой электронной версии каждая строка Python начинается с левого края: код не зависит от абзацных отступов читалки.

В терминале, открытом в этой папке, выполните команду ниже. Если на вашем компьютере Python запускается как python3 или py, замените только первое слово команды.

```
python create_shoplab.py
```

Программа создаст shoplab.sqlite3 рядом с файлом и выведет контрольные количества строк. Если база уже существует, она остановится и не перезапишет её. Для повторного чистого запуска используйте новую пустую папку.

customers — 30 строк.

products — 18 строк.

orders — 72 строк.

order_items — 144 строк.

payments — 60 строк.

Python и встроенная в него SQLite могут иметь разные версии. Для проверки поддержки оконных функций выполните следующий короткий пример в отдельном файле check_sqlite.py. Он не создаёт файлов базы и должен напечатать [(1,)].

```
import sqlite3
db = sqlite3.connect(":memory:")
print(db.execute("SELECT row_number() OVER ()").fetchall())
db.close()
```

Если Python не найден, установите актуальную стабильную версию Python 3 с официального сайта python.org. Если проверка выдаёт ошибку синтаксиса у OVER, используемая библиотека SQLite слишком старая для шестой главы. Текст книги можно читать с телефона, но выполнять практические задания удобнее на компьютере.

Первый запрос

Создайте query.py из приложения Б. В файле answer.sql сохраните один запрос без обрамляющих кавычек:

```
SELECT COUNT(*) AS customers_count
FROM customers;
```

```
python query.py answer.sql
```

Ожидается заголовок customers_count, значение 30 и строка Строк: 1. Затем заменяйте содержимое answer.sql своим решением. Команда показывает весь результат и число строк; автоматическую оценку она не выставляет.

В некоторых читалках копирование длинного кода неудобно. Откройте приобретённую книгу на компьютере в доступном текстовом формате. При переносе проверяйте переносы строк и прямые кавычки. Если читалка добавила отступы абзацев, уберите пробелы только в начале строк кода; внутри строк пробелы сохраняйте. Код нельзя копировать как картинку.

Схема и правила расчётов

customers

Покупатели. customer_id — первичный ключ, customer_name — учебное имя, city — город, segment — retail, pro или premium, signup_date — дата регистрации. email_opt_in равен 1 при согласии на рассылку и 0 иначе.

products

Товары. product_id — ключ, product_name — название, category — office, home, electronics или sport. price_rub — текущая цена в целых рублях, stock_qty — учебный остаток, is_active — признак активности 0 или 1.

orders

Заказы. order_id — ключ, customer_id ссылается на покупателя, order_date — дата. status принимает paid, shipped, delivered или cancelled. channel принимает market, site или partner.

order_items

Строки заказов. Составной ключ order_id и product_id допускает товар в заказе один раз. quantity — количество, unit_price_rub — цена за единицу именно в этом заказе. Она может отличаться от текущей цены товара.

payments

Платежи. payment_id — ключ; уникальный order_id означает не более одного платежа на заказ. paid_at — дата, amount_rub — сумма в целых рублях, method — card, sbp или invoice.

Связи: один покупатель имеет несколько заказов; один заказ имеет несколько товарных строк и ноль или один платёж. Одна строка заказа относится к одному товару. Внешние ключи включены при создании базы.

В этом наборе отменённые заказы не имеют платежей; остальные оплачены целиком. Возвраты, налоги, комиссии и себестоимость не моделируются. Слово «выручка» в заданиях означает только сумму учебных платежей либо равную ей стоимость строк неотменённых заказов. Чистая прибыль здесь не рассчитывается.

Если условие не исключает отменённые заказы, учитывайте их. В задачах на заказы и товарные строки это специально оговорено. Платёж не следует суммировать после размножающего его соединения с товарными строками.

Платежи охватывают январь — август 2026 года; август неполный. Дни и месяцы без строк не добавляются автоматически. Временем отсчёта для «последних 30 дней» служит максимальная дата набора, а не сегодняшний день.

Глава 1 Базовые выборки

Начните с набора строк. WHERE оставляет только строки, удовлетворяющие условию, а SELECT задаёт столбцы результата. ORDER BY нужен, если важна последовательность; без него порядок не обещан.

В LIMIT сначала определите сортировку. Для устойчивого результата добавляйте идентификатор после основной метрики, если условие не требует сохранять ничьи. NULL проверяют через IS NULL, а не через равенство.

Подсказки и эталоны находятся во второй части под теми же номерами. Поля в строке «Результат» задают форму ответа; выводите их в указанном порядке. Если в условии не указан порядок строк, используйте сортировку из эталона для удобства сравнения.

Задача 1 1

Выведите все активные товары: идентификатор, название, категорию и цену. Отсортируйте сначала по категории, затем по цене по возрастанию.

Результат: product_id, product_name, category, price_rub.

Задача 1 2

Найдите покупателей из Москвы и Казани. Покажите имя, город и сегмент, отсортируйте по городу и имени.

Результат: customer_name, city, segment.

Задача 1 3

Покажите идентификатор заказа, покупателя, дату и статус для заказов за февраль 2026 года. Включите весь месяц. Отсортируйте по дате и идентификатору заказа.

Результат: order_id, customer_id, order_date, status.

Задача 1 4

Получите список уникальных городов покупателей в алфавитном порядке.

Результат: city.

Задача 1 5

Найдите пять самых дорогих активных товаров.

Результат: product_name, category, price_rub.

Задача 1 6

Выведите товары с остатком меньше 10 единиц, но больше нуля.

Результат: product_id, product_name, stock_qty.

Задача 1 7

Найдите товары, в названии которых содержится подстрока «набор» точно в указанном регистре. Выведите product_id, product_name и price_rub, отсортировав по product_id.

Результат: product_id, product_name, price_rub.

Задача 1 8

Покажите все отменённые заказы из канала site.

Результат: order_id, customer_id, order_date.

Задача 1 9

Разделите товары на ценовые группы: цена меньше 500 рублей — low; от 500 включительно до 1000 не включительно — mid; от 1000 включительно — high. Выведите название, цену и группу, отсортировав по цене и product_id.

Результат: product_name, price_rub, price_band.

Задача 1 10

Выведите покупателей, которые согласились на email-рассылку и зарегистрировались не раньше 1 марта 2026 года.

Результат: customer_id, customer_name, signup_date.

Глава 2 Агрегации

GROUP BY меняет единицу наблюдения: вместо отдельных строк вы получаете одну строку на группу. WHERE фильтрует исходные строки, HAVING — уже посчитанные группы.

Перед SUM определите, что именно складывается: стоимость строк заказа, полученные платежи или количество единиц. COUNT(*) считает строки, COUNT(поле) пропускает NULL. Для долей используйте 100.0, чтобы избежать целочисленного деления.

Подсказки и эталоны находятся во второй части под теми же номерами. Поля в строке «Результат» задают форму ответа; выводите их в указанном порядке. Если в условии не указан порядок строк, используйте сортировку из эталона для удобства сравнения.

Задача 2 1

Посчитайте количество покупателей в каждом городе.
Результат: city, customers_count.

Задача 2 2

Найдите среднюю цену активных товаров в каждой категории, округлив до двух знаков.
Результат: category, avg_price_rub.

Задача 2 3

Посчитайте число заказов каждого статуса.
Результат: status, orders_count.

Задача 2 4

Посчитайте сумму полученных платежей по каждому способу оплаты.

Конец ознакомительного фрагмента.