

Материал подготовлен с помощью ИИ.

НАБЛЮДЕНИЕ / ГИПОТЕЗА / ПРОВЕРКА

Сбой найден

Два инцидента для знакомства с форматом.

Практикум диагностики backend-инцидентов. Условия, трассировки, гипотезы, исправления и проверяемые результаты.

Уровень: базовый SQL. Ориентировочное время: 20 минут на сценарий.

F1 / Один экран, 101 запрос

Список заказов работает медленно, хотя каждый запрос к базе быстрый.

F2 / Заказ появился дважды

Сортировка есть, но на второй странице повторяется последний заказ с первой.

Формат самостоятельной практики. Индивидуальная проверка, видеолекции и обещания карьерного результата не входят в материал.

Как устроена практика

01 / Условие

На странице задачи представлены только исходные факты и варианты гипотез. До чтения разбора полезно письменно зафиксировать причину, предполагаемое исправление и проверяемый инвариант.

02 / Разбор

Следующая страница объясняет правильный вариант, недостатки альтернатив и границы решения. Подсказки не являются штрафом: цель --- выстроить причинную цепочку.

03 / Эксперимент

Проверки количества запросов и пагинации входят в воспроизводимые модели. Условия и разборы также доступны на демонстрационной странице.

Оценка собственного решения

По одному баллу за каждый пункт: причина связана с фактами; сформулирован инвариант; исправление защищает этот инвариант; проверено поведение при сбое или конкурентном вызове. Максимум --- 4 балла на задачу. Выбор правильного варианта без объяснения не равен полному решению.

Границы лаборатории

Это учебные модели, а не готовые компоненты платежной системы. Нагрузочные характеристики, провайдеры авторизации, реальные брокеры и сетевые отказы требуют отдельной проверки. В исходных примерах намеренно присутствуют ошибочные алгоритмы; их назначение явно указано.

F1 / SQL / N+1 / ~20 МИНУТ

Один экран, 101 запрос

API возвращает 100 заказов вместе с именами покупателей. На странице присутствуют 100 разных покупателей. Сериализация выполняется последовательно. Время сети до базы включено во время каждого SQL-вызова. Все данные и трассировки в задаче синтетические.

Наблюдения

```
GET /orders?limit=100    total=842 ms
SQL calls=101           SQL total=808 ms
Application + JSON=34 ms
```

```
SELECT id, customer_id FROM orders
ORDER BY id DESC LIMIT 100;      -- 8 ms
SELECT name FROM customers WHERE id=9001; -- 8 ms
SELECT name FROM customers WHERE id=9002; -- 8 ms
... еще 98 отдельных SELECT по первичному ключу
```

```
Схема: orders(id PK, customer_id FK NOT NULL),
customers(id PK, name NOT NULL).
План каждого поиска покупателя использует PK.
```

Какое изменение непосредственно устраняет показанную причину задержки, сохраняя 100 заказов и имена покупателей?

- A. Добавить еще один индекс на customers.id
- B. Получить заказы и покупателей одним JOIN
- C. Увеличить пул соединений в 10 раз

До разбора: причина → инвариант → исправление → проверка.

Место для гипотезы

F1 / РАЗБОР

Один экран, 101 запрос

Правильный вариант: В

Это N+1: один запрос получает список, затем отдельный запрос обслуживает каждый элемент. Быстрый план отдельного запроса не устраняет суммарную стоимость повторов. Цель исправления --- сократить число обращений при сохранении состава результата.

Исправление

1. Для данного списка выбираются только необходимые поля одним JOIN. Альтернатива --- два запроса: заказы и пакет покупателей по уникальным идентификаторам.
2. Если ORM загружает связи лениво, проверяется фактический SQL во время сериализации. Исправление должно относиться к конкретному экрану, а не включать eager loading для всех связей.
3. Для связей один-ко-многим прямой JOIN с LIMIT требует отдельной проверки: ограничение может начать действовать на строки JOIN, а не на число заказов.

```
SELECT o.id, c.name
FROM orders AS o
JOIN customers AS c ON c.id = o.customer_id
ORDER BY o.id DESC
LIMIT 100;
```

Альтернативные гипотезы

- А. Поиск уже использует первичный ключ. Второй индекс не сокращает число сетевых обращений.
- С. Текущий код выполняет запросы последовательно. Большой пул сам по себе не меняет этот алгоритм и может увеличить нагрузку.

F1 / ПРОВЕРКА И ГРАНИЦЫ

Как доказать исправление

1. Сравниваются идентификаторы, порядок и имена в обоих результатах: они совпадают.
2. На выборках из 1, 10 и 100 заказов исправленный вариант выполняет один запрос, а исходный --- 2, 11 и 101.
3. На реальном стенде отдельно измеряется р95 при одинаковой нагрузке. Учебная трассировка не является обещанием конкретного ускорения.

Ограничения

Лабораторный скрипт проверяет количество запросов и эквивалентность результата в SQLite. Он не моделирует сеть, ORM и производительность PostgreSQL.

Подсказки к исходной задаче

1. Полезно сравнить число заказов и число SQL-вызовов.
2. Время одного запроса невелико; основная цена повторяется 100 раз.
3. В данной схеме у каждого заказа ровно один покупатель. JOIN не размножает строки заказов.

Вопрос на развитие

Как изменится решение, если нужно показать три последние позиции каждого заказа? Какие строки должен ограничивать LIMIT?

Связанный эксперимент

Идентификатор модели: n_plus_one. Python / SQLite.

Первичная документация

PostgreSQL: [table expressions / JOIN](#)

<https://www.postgresql.org/docs/18/queries-table-expressions.html>

F2 / API / пагинация / ~20 МИНУТ

Заказ появился дважды

Лента сортируется по неизменяемому уникальному id по убыванию. Первая страница и следующая запрашиваются в разных транзакциях. Между запросами может появиться новый заказ. Требование --- продолжить просмотр старых заказов без повторов из-за вставок в начало ленты.

Наблюдения

До первого запроса: id = [106, 105, 104, 103, 102, 101]

Страница 1: LIMIT 3 OFFSET 0 -> [106, 105, 104]

Между запросами добавлен заказ id=107.

Страница 2: LIMIT 3 OFFSET 3 -> [104, 103, 102]

```
SELECT id FROM orders
ORDER BY id DESC
LIMIT 3 OFFSET :offset;
```

Как продолжить ленту после id=104 без сдвига границы при добавлении id=107?

A. Еще раз указать ORDER BY id DESC

B. Убирать повторы только на клиенте

C. Запросить id < 104 с тем же порядком и LIMIT

До разбора: причина → инвариант → исправление → проверка.

Место для гипотезы

F2 / РАЗБОР

Заказ появился дважды

Правильный вариант: C

OFFSET отсчитывает позиции в новом наборе. Вставка в начало сдвигает границу. Для последовательной ленты подходит курсорная пагинация по стабильному уникальному ключу.

Исправление

1. Первая страница возвращает курсор последней строки. Следующая использует строгое условие `id < cursor_id`.
2. Если сортировка идет по `created_at`, к ней добавляется `id` для однозначности: `ORDER BY created_at DESC, id DESC`; курсор содержит оба значения, условие `--- (created_at, id) < (:time, :id)`. Оба поля должны быть `NOT NULL`.
3. Фильтры, направление сортировки и область доступа сохраняются между страницами. Курсор проходит проверку формата; значения передаются в SQL параметрами.

```
SELECT id FROM orders
WHERE id < :cursor_id
ORDER BY id DESC
LIMIT :page_size;
-- cursor_id = 104, page_size = 3
-- результат: [103, 102, 101]
```

Альтернативные гипотезы

- A. Порядок уже однозначен. Ошибка возникает из-за изменения позиции строк между запросами.
- B. Это скрывает симптом, но не определяет устойчивую границу страницы и не устраняет пропуски при удалениях перед OFFSET.

F2 / ПРОВЕРКА И ГРАНИЦЫ

Как доказать исправление

1. После вставки 107 вторая страница остается [103, 102, 101].
2. После удаления уже показанного заказа граница не сдвигается; дубли не появляются.
3. Для составного курсора проверяются одинаковые временные метки. Все строки должны обходиться ровно один раз в неизменяемой выборке.

Ограничения

Курсор не создает снимок всей ленты. Изменение ключа сортировки способно привести к повтору или пропуску; для стабильной выгрузки нужен отдельный контракт снимка. Произвольный переход к странице 50 становится иной задачей.

Подсказки к исходной задаче

1. ORDER BY определяет порядок внутри одного запроса, но не фиксирует весь набор на время просмотра.
2. После вставки 107 строка 104 перемещается с позиции 3 на позицию 4.
3. Продолжение можно выразить через значение последнего увиденного ключа.

Вопрос на развитие

Как обеспечить выгрузку одного фиксированного набора, если записи продолжают меняться во время обхода?

Связанный эксперимент

Идентификатор модели: pagination. Python / SQLite.

Первичная документация

PostgreSQL: LIMIT and OFFSET

<https://www.postgresql.org/docs/18/queries-limit.html>