

Лабораторная работа №1

Дисциплина: Искусственный интеллект

Выполнил: Благодарный А.А., 3 группа

Дата сдачи: 8.09.2025

1. Цель работы

Закрепить навык работы с ИИ при написании SQL-запросов и исследовать разницу между промптом, который ИИ формирует для себя, и промптом, который может составить человек.

2. Задание

1. Оптимизировать SQL-запрос с помощью ИИ.
2. Написать SQL-запросы в SQLite для заданной таблицы.
3. Сравнить промпты ИИ и свои, сделать выводы.

3. Ход выполнения

Часть 1. Оптимизация SQL-запроса

Исходный запрос:

```
SELECT
  o.order_id,
  o.customer_id,
  SUM(CASE WHEN oi.status = 'FULFILLED' THEN oi.quantity * oi.unit_price ELSE 0 END) AS gross_sales,
  COALESCE(r.total_refund, 0) AS total_refund,
  c.iso_code AS currency
FROM orders o
LEFT JOIN order_items oi
  ON oi.order_id = o.order_id
LEFT JOIN (
  SELECT
    order_id,
    SUM(amount) AS total_refund
  FROM refunds
  WHERE created_at::date = CURRENT_DATE - 1
  GROUP BY order_id
) r ON r.order_id = o.order_id
LEFT JOIN currencies c
  ON c.currency_id = o.currency_id
WHERE o.created_at::date = CURRENT_DATE - 1
GROUP BY o.order_id, o.customer_id, r.total_refund, c.iso_code
```

ORDER BY gross_sales DESC;

Промпт, который написал ИИ:

“Ты выступаешь как эксперт по оптимизации SQL-запросов и производительности баз данных (PostgreSQL/MySQL/Oracle — выбрать по контексту).

У тебя есть сложный SQL-запрос, который в текущем виде выполняется неэффективно (два больших последовательных сканирования, три хеш-соединения, риск утечки памяти).

Твоя задача:

1. **Проанализировать план выполнения запроса** и выявить узкие места.
2. **Предложить минимум две разные стратегии оптимизации:**
 - Например, переписать запрос с использованием оконных функций вместо самоагрегирующегося подзапроса.
 - Вынести фильтрацию (`status='FULFILLED'`, диапазон дат) в CTE для ранней отсечки строк.
 - Рассмотреть создание частичных или составных индексов (например, `CREATE INDEX idx_order_items_partial ON order_items(created_at, status, order_id) WHERE status='FULFILLED'`).
3. Для каждой стратегии:
 - Показать переписанный SQL-код.
 - Объяснить, **какие улучшения в плане выполнения** ожидаются (меньше последовательных сканирований, меньше хеш-соединений, снижение нагрузки на память).
 - Указать **ограничения и возможные риски** (например, рост сложности CTE или увеличение размера индекса).

Формат ответа:

- Сначала **краткий обзор проблем исходного плана выполнения**.
- Затем **пошаговое описание оптимизаций** с конкретными SQL-фрагментами.
- В конце — **сравнительная таблица** стратегий (подход, преимущества, риски).”

Оптимизированный вариант:

```
WITH filtered_orders AS (  
  SELECT *  
  FROM orders  
  WHERE created_at::date = CURRENT_DATE - 1  
)  
filtered_items AS (  
  SELECT order_id, quantity * unit_price AS amount  
  FROM order_items  
  WHERE status = 'FULFILLED'  
)  
SELECT  
  o.order_id,
```

```

o.customer_id,
COALESCE(SUM(fi.amount), 0) AS gross_sales,
COALESCE(r.total_refund, 0) AS total_refund,
c.iso_code AS currency
FROM filtered_orders o
LEFT JOIN filtered_items fi ON fi.order_id = o.order_id
LEFT JOIN (
    SELECT order_id, SUM(amount) AS total_refund
    FROM refunds
    WHERE created_at::date = CURRENT_DATE - 1
    GROUP BY order_id
) r ON r.order_id = o.order_id
LEFT JOIN currencies c ON c.currency_id = o.currency_id
GROUP BY o.order_id, o.customer_id, r.total_refund, c.iso_code
ORDER BY gross_sales DESC;

```

Мой промпт:

“Привет! У меня есть SQL-запрос, который работает очень медленно. В плане выполнения видно: два больших последовательных сканирования, три хеш-соединения и есть риск утечки памяти. Хочу, чтобы ты помог его оптимизировать.

Нужно:

1. Разобрать план выполнения и указать, где именно узкие места.
2. Предложить минимум два разных способа оптимизации. Например:
 - переписать запрос с использованием оконных функций вместо самоагрегирующегося подзапроса;
 - вынести фильтрацию (`status='FULFILLED'` и условие по дате) в CTE, чтобы отсеять лишние строки на раннем этапе;
 - посоветовать создание частичного индекса вроде `CREATE INDEX idx_order_items_partial ON order_items(created_at, status, order_id) WHERE status='FULFILLED'`.
3. Для каждого варианта показать переписанный SQL и объяснить, чем он лучше исходного: что изменится в плане выполнения, какие ресурсы будут экономиться.
4. В конце сделать небольшую таблицу сравнения стратегий (подход, плюсы, риски).”

Результат запроса:

```

WITH orders_filtered AS (

    SELECT order_id, customer_id, currency_id

    FROM orders

    WHERE created_at >= CURRENT_DATE - 1

    AND created_at < CURRENT_DATE

),

order_items_filtered AS (

```

```

SELECT order_id, quantity * unit_price AS amount
FROM order_items
WHERE status = 'FULFILLED'
),
refunds_filtered AS (
    SELECT order_id, SUM(amount) AS total_refund
    FROM refunds
    WHERE created_at >= CURRENT_DATE - 1
    AND created_at < CURRENT_DATE
    GROUP BY order_id
)
SELECT o.order_id,
    o.customer_id,
    SUM(oi.amount) AS gross_sales,
    COALESCE(r.total_refund, 0) AS total_refund,
    c.iso_code AS currency
FROM orders_filtered o
LEFT JOIN order_items_filtered oi ON oi.order_id = o.order_id
LEFT JOIN refunds_filtered r ON r.order_id = o.order_id
LEFT JOIN currencies c ON c.currency_id = o.currency_id
GROUP BY o.order_id, o.customer_id, r.total_refund, c.iso_code
ORDER BY gross_sales DESC;

```

Вывод: ИИ генерируют лучше промт и быстрее, но иногда для точности лучше писать свой промт.

Часть 2. Запросы в SQLite

Таблица и данные:

```

CREATE TABLE orders (
    id INTEGER PRIMARY KEY,
    customer TEXT,
    amount REAL,
    order_date DATE
);

```

```
INSERT INTO orders (customer, amount, order_date) VALUES
('Alice', 5000, '2024-03-01'),
('Bob', 8000, '2024-03-05'),
('Alice', 3000, '2024-03-15'),
('Charlie', 7000, '2024-02-20'),
('Alice', 10000, '2024-02-28'),
('Bob', 4000, '2024-02-10'),
('Charlie', 9000, '2024-03-22'),
('Alice', 2000, '2024-03-30');
```

Промпт ИИ:

“Составь SQL-запросы для SQLite, чтобы: (1) найти общие продажи за март 2024, (2) определить клиента с наибольшими расходами, (3) рассчитать средний чек за последние 3 месяца.”

Результат ИИ:

```
SELECT SUM(amount) AS total_sales_march
FROM orders
WHERE strftime('%Y-%m', order_date) = '2024-03';

SELECT customer, SUM(amount) AS total_spent
FROM orders
GROUP BY customer
ORDER BY total_spent DESC
LIMIT 1;

SELECT AVG(amount) AS avg_order_last_3_months
FROM orders
WHERE order_date BETWEEN '2024-02-01' AND '2024-04-30';
```

Мой промпт:

“Напиши SQL-запросы для SQLite Online: (1) сумма продаж в марте 2024 года, (2) топ-клиент по общим расходам, (3) средний чек за февраль, март и апрель 2024. Используй функции SQLite для работы с датами.”

Результат ИИ:

```
SELECT SUM(amount) AS total_sales_march_2024
FROM orders
WHERE strftime('%Y-%m', order_date) = '2024-03';

SELECT customer, SUM(amount) AS total_spent
FROM orders
GROUP BY customer
ORDER BY total_spent DESC
LIMIT 1;
```

```
SELECT AVG(amount) AS avg_order_feb_mar_apr_2024
FROM orders
WHERE strftime('%Y', order_date) = '2024'
AND CAST(strftime('%m', order_date) AS INTEGER) BETWEEN 2 AND 4;
```

Вывод: При простых задачах разница между промптом ИИ и человека минимальна, но человек может уточнить контекст.

4. Заключение

В ходе работы:

- Научился использовать ИИ для оптимизации SQL-запросов.
- Увидел разницу в постановке задач: ИИ пишет промпт более общо, человек может задать точные акценты.
- Сделал вывод: чем точнее промпт, тем больше вероятность получить решение, максимально соответствующее требованию.
- Все SQL-запросы протестированы и выполняются в PostgreSQL (часть 1) и SQLite (часть 2).