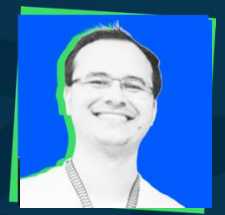
OZON tech

Перестаньте оптимизировать C# и займитесь вашей БД



Дмитрий Орлов
Руководитель подгруппы постингов, Ozon


LEGO CITY
6-12
LEGO CITY
LEGO CITY

AGENDA

01 **Почему перестать?** [easy]

02 **Как искать медленные запросы?** [easy]

03 **Устройство и планы запросов PostgreSQL** [average]

04 **Причины медленных запросов и варианты решения** [hard]

01 { Почему перестать?

Моделируем ситуацию

Тимлид прислал код и сказал, что он тормозит

```
public async IAsyncEnumerable<OrderResponse> GetOrders(...)
{
    var orders = await _ordersRepository.GetOrders(...);
    var items = await _itemsService.GetItems(...);

    foreach (var order in orders)
    {
        var orderItems = items.Where(i => order.ItemIds.Contains(i.Id));

        yield return new OrderResponse(order, orderItems);
    }
}
```

Моделируем ситуацию

```
public async IAsyncEnumerable<OrderResponse> GetOrders(...)
{
    var orders = await _ordersRepository.GetOrders(...);
    var items = await _itemsService.GetItems(...);

    foreach (var order in orders)
    {
        var orderItems = items.Where(i => order.ItemIds.Contains(i.Id));

        yield return new OrderResponse(order, orderItems);
    }
}
```

Моделируем ситуацию

```
public async IAsyncEnumerable<OrderResponse> GetOrders(...)
{
    var orders = await _ordersRepository.GetOrders(...);
    var items = await _itemsService.GetItems(...).ToDictionary(...);

    foreach (var order in orders)
    {
        var orderItems = order.ItemIds.Select(i => items[i]);

        yield return new OrderResponse(order, orderItems);
    }
}
```

Моделируем ситуацию

Тимлид благодарит
и показывает график

Latency Numbers Every Programmer Should Know ©

```
L1 cache reference ..................... 0.5 ns
Branch mispredict ....................... 5 ns
L2 cache reference ...................... 7 ns
Mutex lock/unlock ....................... 25 ns
Main memory reference ..................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

Профилируйте!

02

Как искать медленные запросы?

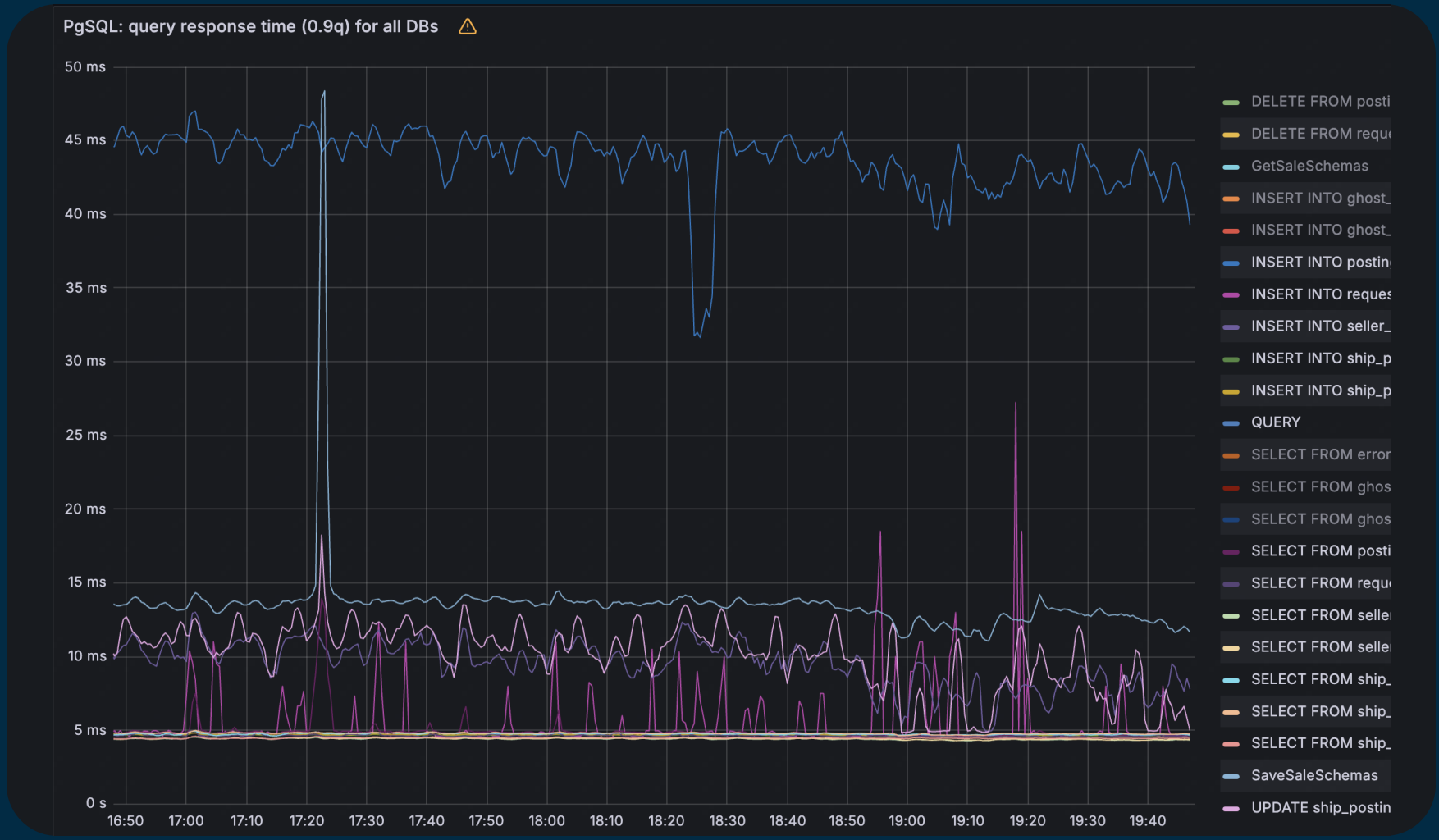
EF Core — TagWith

```
1 var orders = await context.Orders
2     .Where(o => o.ClientId == clientId)
3     .TagWith("GetClientOrders");
```

https://learn.microsoft.com/ru-ru/ef/core/querying/tags

Метрики и трассировки

- Prometheus
- OpenTelemetry
- OpenTelemetry.
  Npgsql



Средства PostgreSQL

```
select * from pg_stat_acivity;
select * from pg_stat_statements;
```

```
log_min_duration_statement = n (ms)
```

```
-- Модуль pg_store_plans
select * from pg_store_plans;
```

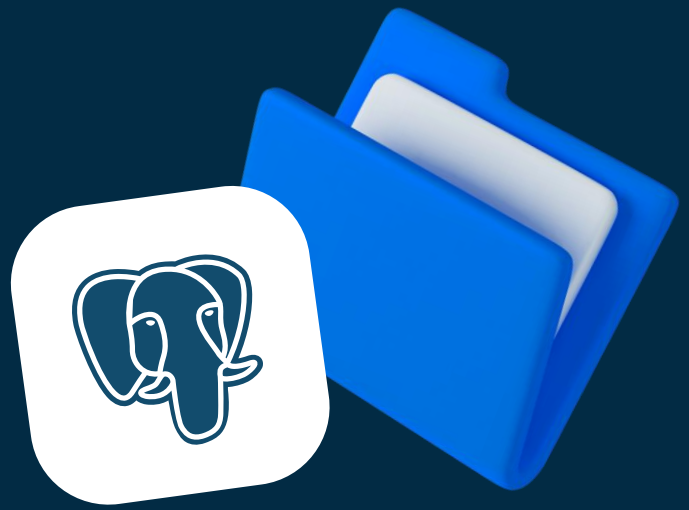
```
select * from pg_stats;
```

03

Устройство и планы запросов PostgreSQL

Как PostgreSQL хранит данные

- Данные в PostgreSQL лежат в таблицах-кучах (heap)
- Данные хранятся страницами по 8 kb (по умолчанию)
- Когда вы обновляете данные, PostgreSQL создаёт новую запись, а старую помечает удалённой — MVCC
- Индексы лежат отдельно и содержат ссылки на страницы (кластерных нет)
- Реальной очисткой рулит процесс vacuum



Vacuum

01 Удаляет мертвые кортежи (dead tuples)

02 Обновляет статистику

03 Обновляет visibility map

04 Отдаёт ОС пустые страницы с конца таблицы

05 …FULL — отдаёт все, требует ACCESS EXCLUSIVE LOCK

06 Запускается периодически сам (auto vacuum)

Как смотреть план?

EXPLAIN

…ANALYZE — выполнит запрос и даст actual (time, rows)

… BUFFERS — поглядеть на IO

BEGIN EXPLAIN ANALYZE … ROLLBACK

https://postgrespro.ru/docs/postgresql/16/using-explain

```
1 explain (analyze, buffers)
2   select *
3   from ship_postings
4   where posting_id in (
5     23624730760,
6     24987564205);
```

```
Index Scan using ship_postings_pkey on ship_postings
(cost=0.57..4.26 rows=2 width=259)
(actual time=3.300..5.676 rows=2 loops=1)
  Index Cond: (posting_id = ANY ('{23624730760,24987564205}'::bigint[]))
  Buffers: shared hit=1 read=9
Planning:
  Buffers: shared hit=4
Planning Time: 1.720 ms
Execution Time: 5.737 ms
```

4 CPU
4 GB RAM
ship_postings: 1 млрд
ship_posting_schemas: 1.5 млрд

```
1 explain (analyze, buffers)
2     select *
3     from ship_postings
4         join ship_posting_schemas using (posting_id)
5     where posting_id in (24557310091, 24557311091)
6       and shipped_at is not null;
```

```
Nested Loop
  (cost=1.15..9.45 rows=1 width=280)
  (actual time=4.099..4.116 rows=1 loops=1)
  Buffers: shared hit=9 read=8
  ->  Index Scan using ship_postings_pkey on ship_postings
      (cost=0.57..4.26 rows=2 width=259)
      (actual time=2.309..2.324 rows=1 loops=1)
        Index Cond: (posting_id = ANY ('{24557310091,24557311091}'::bigint[]))
        Buffers: shared hit=8 read=4
  ->  Index Scan using ship_posting_schemas_pkey on ship_posting_schemas
      (cost=0.58..2.59 rows=1 width=29)
      (actual time=1.783..1.783 rows=1 loops=1)
        Index Cond: (posting_id = ship_postings.posting_id)
        Filter: (shipped_at IS NOT NULL)
        Buffers: shared hit=1 read=4
Planning:
  Buffers: shared hit=216 read=32
Planning Time: 16.503 ms
Execution Time: 5.720 ms
```

Cost vs Time

Cost

Измеряется
в попугаях

**1 чтение с диска*

Time

Измеряется
в миллисекундах

Что ещё есть в плане

Width

Средний размер строки в байтах

Buffers

1 = 8 килобайт

Rows

Число строк по плану

Actual rows

Число строк по факту

04

Причины медленных запросов

Ситуации

01 **Плохой запрос**

Который видно сразу

02 **Нормальный запрос**

Без индексов

03 **Нормальный запрос**

Индексы не используются

04 **Нормальный запрос**

Индексы есть

Плохой запрос

```sql
SELECT * FROM (
    SELECT * FROM (
        SELECT * FROM (
            SELECT * FROM table
                JOIN (SELECT * FROM otherTable WHERE )
                    a ON table.Id = a.OtherID
                    ...
                    ...
)))))))
```

Может возникнуть, если автор запроса ORM, ChatGPT, построитель запросов, человек, никогда не видевший БД

Решение — переписать, отказаться от ORM, накинуть железа

Ситуации

01 **Плохой запрос**
Который видно сразу

02 **Нормальный запрос**
Без индексов

03 **Нормальный запрос**
Индексы не используются

04 **Нормальный запрос**
Индексы есть

Нормальный запрос

01 Смотрим план, видим SeqScan

02 Добавляем индекс (concurrently, если не хочется блокировок)

03 Смотрим план, видим Index Scan/Index Only Scan

P.S. Выбор полей для индекса — нетривиальная задача

Ситуации

01 **Плохой запрос**
Который видно сразу

02 **Нормальный запрос**
Без индексов

03 **Нормальный запрос**
Индексы не используются

04 **Нормальный запрос**
Индексы есть

Индексы есть, но не используются

01 Неселективный индекс — на bool/enum

02 Функция на поле — lower(email) = ‘cto@ozon.ru’

03 Операторы, которые не поддерживаются индексом — ILIKE

04 Нет статистики по индексу — auto vacuum не справляется

05 Без индекса и правда быстрее — таблица маленькая
или нужно перебрать почти все записи

Ситуации

01 **Плохой запрос**
Который видно сразу

02 **Нормальный запрос**
Без индексов

03 **Нормальный запрос**
Индексы не используются

04 **Нормальный запрос**
Индексы есть

Индексы есть и используются

01 rows и actual rows сильно разнятся

02 Index Only Scan (Heap Fetches > 0)

03 Buffers: shared read=100500

Actual Rows != Rows

01 Планировщик думает одно, а по факту — другое

02 Планировщик опирается на статистику
(число строк, уникальных значений, распределение…)

Решение — тюнинг auto vacuum

Index Only Scan Heap Fetches > 0

01 Index Scan ходит в кучу за данными

02 Index Only Scan не ходит…

03 …кроме случаев, когда его visibility map устарела

04 Биты устанавливаются при vacuum

05 Биты сбрасываются при любом изменении страницы

Решение — тюнинг auto vacuum

ozon tech

Buffers: shared read = 100500

01 OFFSET… LIMIT пагинация (используйте WHERE id > previous_id LIMIT)

02 Проверяйте index bloat, table bloat

03 Ваши лучшие друзья — fill factor, auto vacuum, pg_repack, pg_compacttable и правильные удаления

Как правильно удалять данные

- Простой советский DELETE
- DELETE, а затем pg_repack/pg_compacttable
- CREATE → INSERT FROM SELECT → RENAME → DROP
- Партиционирование и DROP PARTITION
- TRUNCATE TABLE
- DROP TABLE или даже DATABASE

Что почитать про индексы

https://habr.com/ru/companies/postgrespro/articles/326096

https://postgrespro.ru/docs/postgresql/16/indexes

Выводы

1. Профилируйте и оптимизируйте самые узкие места
2. Читайте планы запросов, ищите проблемы
3. Работайте с большими данными правильно — поменьше обновлений, побольше правильных удалений
4. Настройте метрики и алерты (query time, table + index bloat)
5. Регулярно дружите с vacuum, pg_repack, pg_compacttable

OZON tech

Спасибо за внимание! Ваши вопросы?

Дмитрий Орлов
Руководитель подгруппы постингов, Ozon
dmorlov@ozon.ru