PostgreSQL и JDBC: выжимаем все соки

Владимир Ситников

PgConf 2016

NetCracker®

О себе

- Владимир Ситников, @VladimirSitnikv
- Инженер по производительности в NetCracker
- 10 лет опыта с Java/SQL
- PgJDBC committer
- English slides: http://www.slideshare.net/VladimirSitnikv

Explain (analyze, buffers) PostgreSQL и JDBC

- Выборка данных
- Вставка данных
- Производительность
- Подводные грабли

Вступление

На выборку одной строки по первичному ключу уходит 20мс. Localhost. База в памяти

Вступление

На выборку одной строки по первичному ключу уходит 20мс. Localhost. База в памяти

A. Это норма

B. Норма это 1сек

C. Да вы что? 1мс же!

D. 100мкс

Много запросов – проблема

Если один запрос занимает 10мс, то 100 запросов это уже секунда *

* Ваш Капитан

Сетевой протокол PostgreSQL

- Simple query

Сетевой протокол PostgreSQL

- Simple query
  - 'Q' + длина + текст_запроса

Сетевой протокол PostgreSQL

- Simple query
  - 'Q' + длина + текст_запроса
- Extended query

Сетевой протокол PostgreSQL

- Simple query
  - 'Q' + длина + текст_запроса
- Extended query
  - Команды Parse, Bind, Execute

Сетевой протокол PostgreSQL

Super extended query

https://github.com/pgjdbc/pgjdbc/pull/478

«что бы нам хотелось от» backend protocol

Сетевой протокол PostgreSQL

Simple query

- Неплохо для одноразовых запросов

Сетевой протокол PostgreSQL

Simple query

- Неплохо для одноразовых запросов
- Не поддерживает бинарный формат

Extended query

- Экономит время планирования

Сетевой протокол PostgreSQL

Extended query

- Экономит время планирования
- Поддерживает бинарный формат передачи

PreparedStatement

```
Connection con = ...;
PreparedStatement ps =
  con.prepareStatement("SELECT...");
  ...
ps.close();
```


PreparedStatement

```
Connection con = ...;
PreparedStatement ps =
  con.prepareStatement("SELECT...");
  ...
ps.close();
```


Работа с PostgreSQL курильщика

```
PARSE S_1 as ...; // con.prepareStmt
  BIND/EXEC
DEALLOCATE // ps.close()
PARSE S_2 as ...;
  BIND/EXEC
DEALLOCATE // ps.close()
```

Работа с PostgreSQL здорового человека

```
PARSE S_1 as ...;
  BIND/EXEC
  BIND/EXEC
  BIND/EXEC
  BIND/EXEC
  BIND/EXEC
  ...
DEALLOCATE
```

Работа с PostgreSQL здорового человека

```
PARSE S_1 as ...; ← 1 раз в жизни
  BIND/EXEC ← обработка REST
  BIND/EXEC
  BIND/EXEC ← ещё REST
  BIND/EXEC
  BIND/EXEC
  ...
DEALLOCATE ← желательно «никогда»
```

Счастливые statement’ов не закрывают

Вывод №1: чтобы работало быстрее, закрывать statement’ы не нужно

Счастливые statement’ов не закрывают

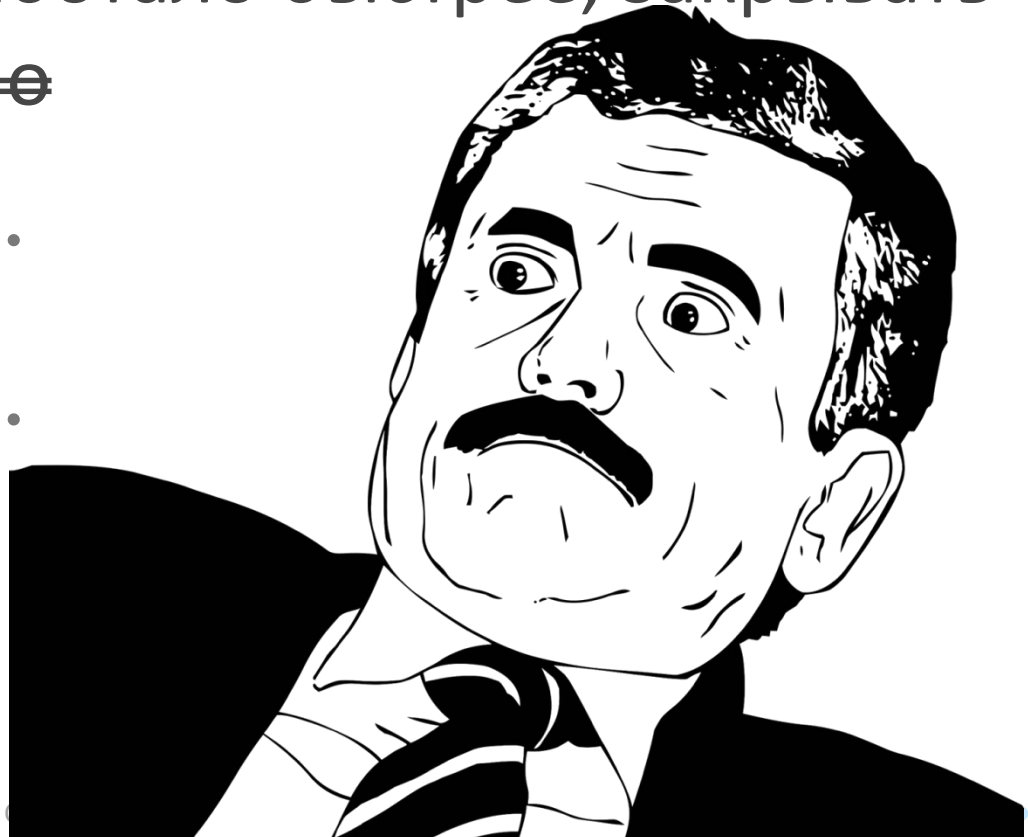
Вывод №1: чтобы работало быстрее, закрывать statement’ы не нужно

```
ps = con.prepareStatement(...)
ps.execueQuery();
ps = con.prepareStatement(...)
ps.execueQuery();
...
```

Счастливые statement’ов не закрывают

Вывод №1: чтобы работало быстрее, ~~закрывать statement’ы не нужно~~

```
ps = con.prepare...
ps.execueQuery();
ps = con.prepare...
ps.execueQuery();
...
```



Что будет, если statement'ы не закрывать

```
@Benchmark
public Statement leakStatement() {
  return con.createStatement();
}
```

Что будет, если statement'ы не закрывать

```
@Benchmark
public Statement leakStatement() {
  return con.createStatement();
}
```

```
pgjdbc < 9.4.1202, -Xmx128m, OracleJDK 1.8u40
# Warmup Iteration 1: 1147,070 ns/op
# Warmup Iteration 2: 12101,537 ns/op
# Warmup Iteration 3: 90825,971 ns/op
# Warmup Iteration 4: <failure>
java.lang.OutOfMemoryError: GC overhead limit exceeded
```

Что будет, если statement'ы не закрывать

```
@Benchmark
public Statement leakStatement() {
  return con.createStatement();
}
```

```
pgjdbc >= 9.4.1202, -Xmx128m, OracleJDK 1.8u40
# Warmup Iteration 1: 30 ns/op
# Warmup Iteration 2: 27 ns/op
# Warmup Iteration 3: 30 ns/op
...
```

Суровая реальность

- В реальности, приложения всегда закрывают statement’ы

Суровая реальность

- В реальности, приложения всегда закрывают statement’ы
- PostgreSQL не имеет общего кэша запросов

Суровая реальность

- В реальности, приложения всегда закрывают statement’ы
- PostgreSQL не имеет общего кэша запросов
- А тратить время на разбор и планирование не хотим

Server-prepared statements

Что делать?

- Завернуть все запросы в PL/PgSQL
  - Это помогает, но у нас 100500 SQL запросов

Server-prepared statements

Что делать?

- Завернуть все запросы в PL/PgSQL
  - Это помогает, но у нас 100500 SQL запросов
- Сделать кэш запросов на уровне JDBC

Кэш запросов в PgJDBC

- Кэш запросов появился в версии 9.4.1202 (2015-08-27)
  см. https://github.com/pgjdbc/pgjdbc/pull/319

Кэш запросов в PgJDBC

- Кэш запросов появился в версии 9.4.1202 (2015-08-27)
  см. https://github.com/pgjdbc/pgjdbc/pull/319
- Работает прозрачно для приложения

Кэш запросов в PgJDBC

- Кэш запросов появился в версии 9.4.1202 (2015-08-27)
  см. https://github.com/pgjdbc/pgjdbc/pull/319
- Работает прозрачно для приложения
- Скорость такая, что PL/PgSQL не нужен

Кэш запросов в PgJDBC

- Кэш запросов появился в версии 9.4.1202 (2015-08-27)
  см. https://github.com/pgjdbc/pgjdbc/pull/319
- Работает прозрачно для приложения
- Скорость такая, что PL/PgSQL не нужен
- Server-prepare активируется после 5-го выполнения (prepareThreshold)

Цифры где?

- Конечно, затраты planning time напрямую зависят от сложности запросов

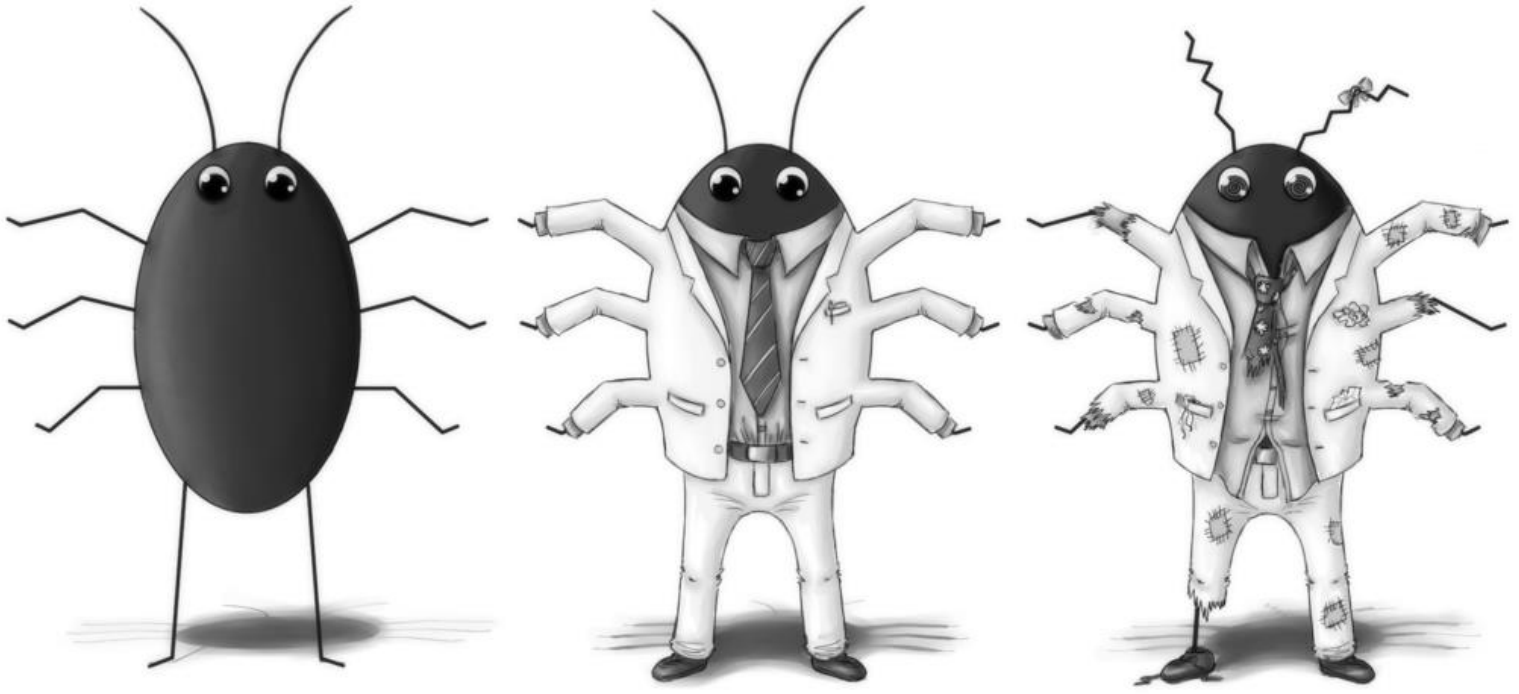
Цифры где?

- Конечно, затраты planning time напрямую зависят от сложности запросов
- У нас доходило до **20мс+ planning time** на OLTP запросах: 10КиБ запрос, 170 строк explain

Цифры где?

- Конечно, затраты planning time напрямую зависят от сложности запросов
- У нас доходило до **20мс+ planning time** на OLTP запросах: 10КиБ запрос, 170 строк explain
- Стало ~0мс

Час расплаты



Генерируемые запросы – зло

- Если запрос генерируется динамически

Генерируемые запросы – зло

- Если запрос генерируется динамически
- То это каждый раз новый объект java.lang.String

Генерируемые запросы – зло

- Если запрос генерируется динамически
- То это каждый раз новый объект java.lang.String
- Значит, приходится заново вычислять hashCode

Типы параметров

Если типы параметров меняются, то server-prepared statement приходится менять

Типы параметров

Если типы параметров меняются, то server-prepared statement приходится менять

```
ps.setInt(1, 42);
```

Типы параметров

Если типы параметров меняются, то server-prepared statement приходится менять

```
ps.setInt(1, 42);

...
ps.setNull(1, Types.VARCHAR);
```

Типы параметров

Если типы параметров меняются, то server-prepared statement приходится менять

```
ps.setInt(1, 42);

...
ps.setNull(1, Types.VARCHAR);
```


Типы параметров

Если типы параметров меняются, то server-prepared statement приходится менять

```
ps.setInt(1, 42);

...
ps.setNull(1, Types.VARCHAR);
```

Это приводит к DEALLOCATE → PREPARE

Тип параметров изменять нельзя

Вывод №1

- Даже у NULL’ов должен быть верный тип

Нежданчик

Перешли на prepared, и запрос замедлился в 5'000 раз. Как так?

Нежданчик

Перешли на prepared, и запрос замедлился в 5'000 раз. Как так?

A. Бага

B. Фича

C. Фича

D. Бага

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

```
select *
  from plan_flipper -- <- таблица
 where skewed = 0 -- 1 млн строк
   and non_skewed = 42 -- 20 строк
```

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

0.1мс ← 1-е выполнение

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

0.1мс ← 1-е выполнение

0.05мс ← 2-е выполнение

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

0.1мс ← 1-е выполнение

0.05мс ← 2-е выполнение

0.05мс ← 3-е выполнение

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

0.1мс ← 1-е выполнение

0.05мс ← 2-е выполнение

0.05мс ← 3-е выполнение

0.05мс ← 4-е выполнение

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

0.1мс ← 1-е выполнение

0.05мс ← 2-е выполнение

0.05мс ← 3-е выполнение

0.05мс ← 4-е выполнение

0.05мс ← 5-е выполнение

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

0.1мс ← 1-е выполнение

0.05мс ← 2-е выполнение

0.05мс ← 3-е выполнение

0.05мс ← 4-е выполнение

0.05мс ← 5-е выполнение

250 мс ← 6-е выполнение

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

0.1мс ← 1-е выполнение

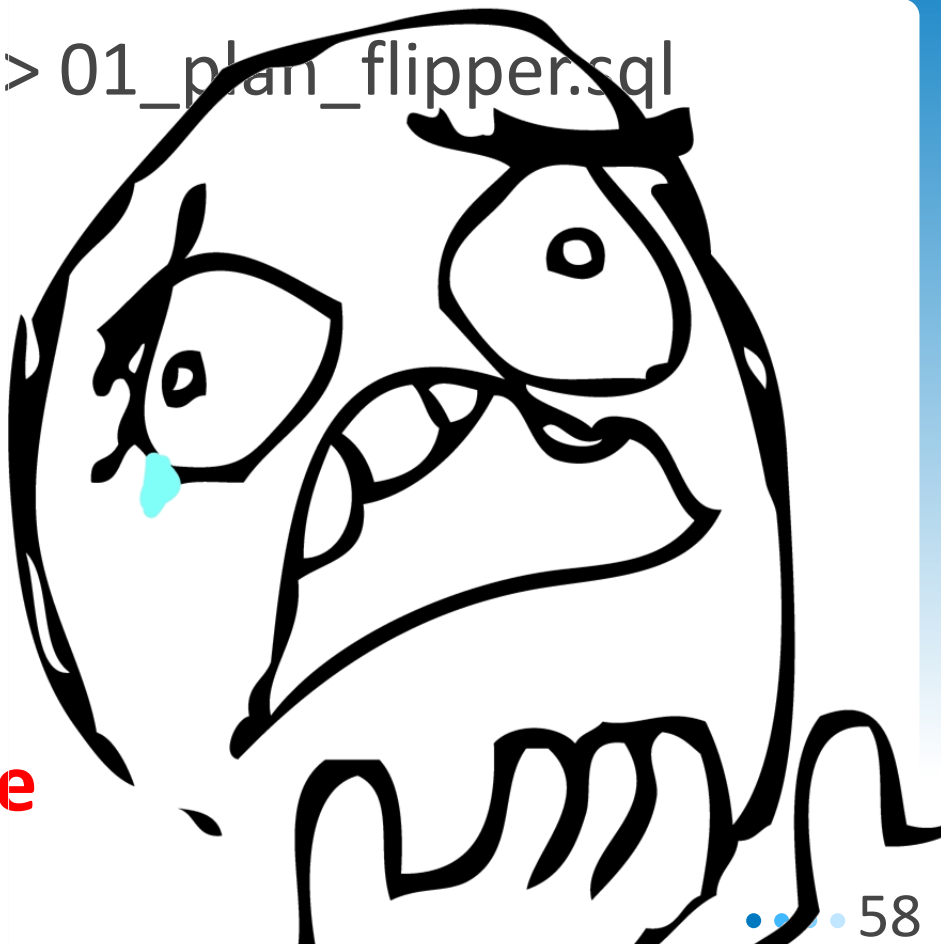
0.05мс ← 2-е выполнение

0.05мс ← 3-е выполнение

0.05мс ← 4-е выполнение

0.05мс ← 5-е выполнение

250 мс ← 6-е выполнение



Нежданчик

- Кто виноват?

Нежданчик

- Кто виноват?
  - PostgreSQL переходит на generic plan после 5-го выполнения server-prepared statement’а

Нежданчик

- Кто виноват?
  - PostgreSQL переходит на generic plan после 5-го выполнения server-prepared statement’а
- Что делать?

Нежданчик

- Кто виноват?
  - PostgreSQL переходит на generic plan после 5-го выполнения server-prepared statement’а
- Что делать?
  - Добавлять `+0`, `OFFSET 0`, и далее по списку

Нежданчик

- Кто виноват?
  - PostgreSQL переходит на generic plan после 5-го выполнения server-prepared statement’а
- Что делать?
  - Добавлять `+0`, `OFFSET 0`, и далее по списку
  - Внимательнее проверять планы

Нежданчик

- Кто виноват?
  - PostgreSQL переходит на generic plan после 5-го выполнения server-prepared statement’а
- Что делать?
  - Добавлять `+0`, `OFFSET 0`, и далее по списку
  - Внимательнее проверять планы
  - Обсуждать в pgsql-hackers

Нежданчик

https://gist.github.com/vlsi -> 01_plan_flipper.sql

Запрещаем использование индекса через +0:

```
select *
  from plan_flipper
 where skewed+0 = 0 ← ~ /*+no_index*/
   and non_skewed = 42
```

Explain explain explain explain

Правило 6-и explain’ов:

Explain explain explain explain

Правило 6-и explain’ов:

```
prepare x(number) as select ...;
```

Explain explain explain explain

Правило 6-и explain’ов:

```
prepare x(number) as select ...;
explain analyze execute x(42); -- 1ms
```

Explain explain explain explain

Правило 6-и explain’ов:

```
prepare x(number) as select ...;
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
```

Explain explain explain explain

Правило 6-и explain’ов:

```
prepare x(number) as select ...;
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
```

Explain explain explain explain

Правило 6-и explain’ов:

```
prepare x(number) as select ...;
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
```

Explain explain explain explain

Правило 6-и explain’ов:

```
prepare x(number) as select ...;
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
```

Explain explain explain explain

Правило 6-и explain’ов:

```
prepare x(number) as select ...;
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 1ms
explain analyze execute x(42); -- 10 sec
```

Везде баг

Проблема выбора

Есть схема **A** с таблицей **Ы**, и схема **Б** с таблицей **Ы**. Что вернёт запрос `select * from Ы`?

Проблема выбора

Есть схема **А** с таблицей **Ы**, и схема **Б** с таблицей **Ы**. Что вернёт запрос `select * from Ы`?

А.Ы

Б.Ы

В. Ошибку

Г. Всё упомянутое выше

Search_path

Есть схема **A** с таблицей **Ы**, и схема **Б** с таблицей **Ы**. Что вернёт запрос `select * from Ы`?

- Для определения схемы используется параметр search_path

Search_path

Есть схема **А** с таблицей **Ы**, и схема **Б** с таблицей **Ы**. Что вернёт запрос `select * from Ы`?

- Для определения схемы используется параметр search_path
- server-prepared statements не подозревают, что search_path может кто-то менять ⍰ может быть что угодно

Search_path может пойти не так

- 9.1 просто выполнит server-prepared со старой таблицей
- 9.2-9.5 могут упасть с ошибкой "cached plan must not change result type"

Search_path

Как лечить?

- Не менять search_path

Search_path

Как лечить?

- Не менять search_path
- Обсуждать в pgsql-hackers
  - Set search_path + server-prepared statements = cached plan must not change result type

Search_path

Как лечить?

- Не менять search_path
- Обсуждать в pgsql-hackers
  - Set search_path + server-prepared statements = cached plan must not change result type
- В PL/pgSQL та же самая проблема ☺

Проблема выбора

Нужно выбрать **1млн** строк по **1КиБ**, -Xmx**128m**

```
while (resultSet.next())
  resultSet.getString(1);
```

Проблема выбора

Нужно выбрать **1млн** строк по **1КиБ**, -Xmx**128m**

```
while (resultSet.next())
  resultSet.getString(1);
```

A. Сработает

B. OutOfMemory

C. Без LIMIT/OFFSET никуда

D. Нужно autoCommit(false)

Выборка данных

- По умолчанию, PgJDBC выбирает все строки

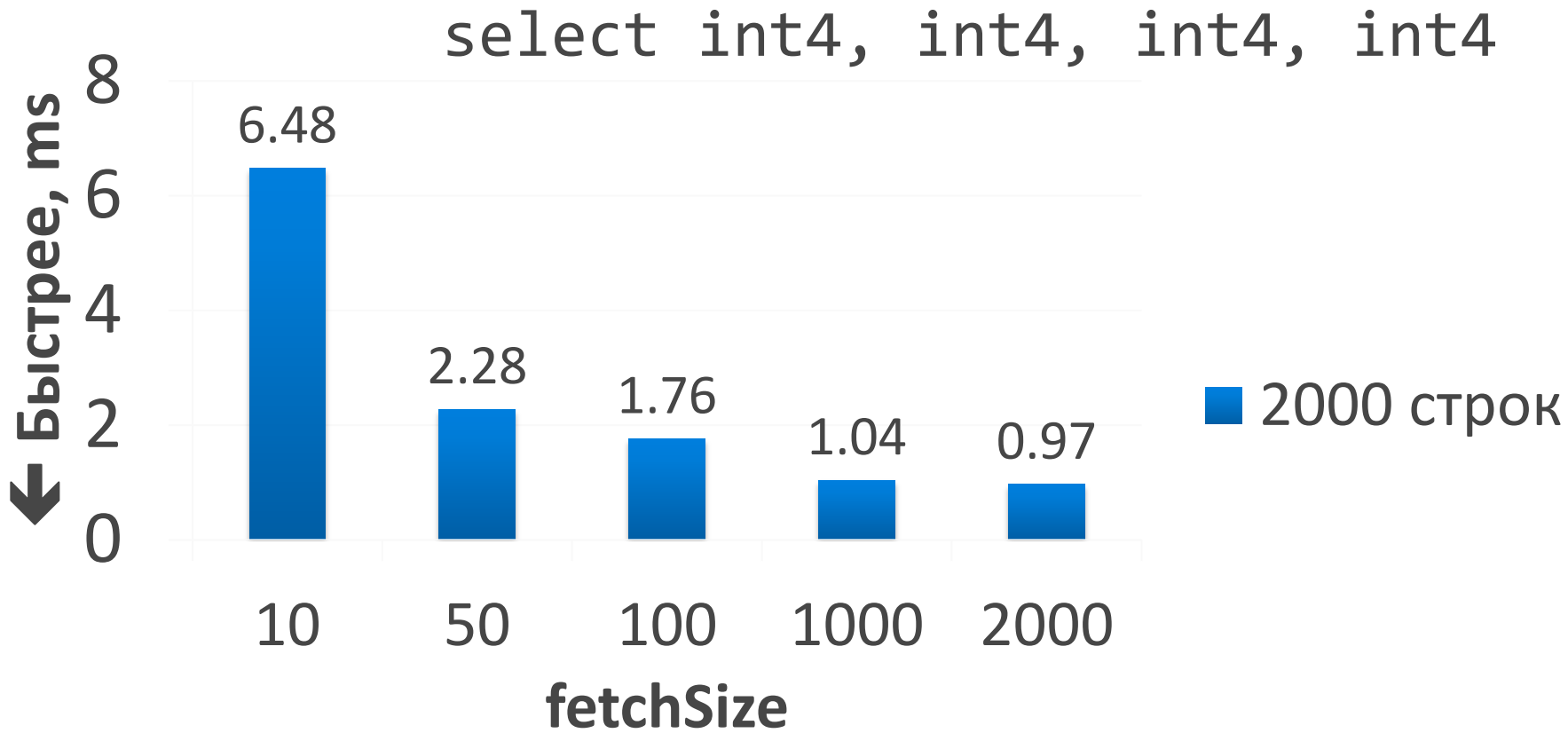
Выборка данных

- По умолчанию, PgJDBC выбирает все строки
- Для выборки по частям, нужно вызвать `Statement.setFetchSize` и `connection.setAutoCommit(false)`

Выборка данных

- По умолчанию, PgJDBC выбирает все строки
- Для выборки по частям, нужно вызвать `Statement.setFetchSize` и `connection.setAutoCommit(false)`
- Глобальная настройка – `defaultRowFetchSize (9.4.1202+)`

Влияние fetchSize на время выборки



FetchSize – добро

Вывод №2:

- Для защиты от переполнения памяти указываем `defaultRowFetchSize >= 100`

PostgreSQL вставляет

Для вставки нужно использовать

PostgreSQL вставляет

Для вставки нужно использовать

- `INSERT() VALUES()`

PostgreSQL вставляет

Для вставки нужно использовать

- `INSERT() VALUES()`
- `INSERT() SELECT ?, ?, ?`

PostgreSQL вставляет

Для вставки нужно использовать

- `INSERT() VALUES()`
- `INSERT() SELECT ?, ?, ?`
- `INSERT() VALUES()` → `executeBatch`

PostgreSQL вставляет

Для вставки нужно использовать

- `INSERT() VALUES()`
- `INSERT() SELECT ?, ?, ?`
- `INSERT() VALUES() → executeBatch`
- `INSERT() VALUES(), (), () → executeBatch`

PostgreSQL вставляет

Для вставки нужно использовать

- `INSERT() VALUES()`
- `INSERT() SELECT ?, ?, ?`
- `INSERT() VALUES() → executeBatch`
- `INSERT() VALUES(), (), () → executeBatch`
- `COPY`

Batch INSERT здорового человека

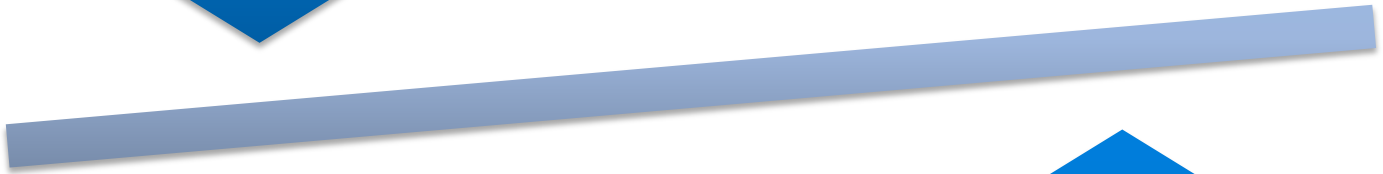
```
PARSE S_1 as ...;
  BIND/EXEC
  BIND/EXEC
  BIND/EXEC
  BIND/EXEC
  BIND/EXEC
  ...
DEALLOCATE
```

TCP наносит ответный удар

JDBC занято отправкой запросов, и ответы ещё не читали

TCP наносит ответный удар

JDBC занято отправкой запросов, и ответы ещё не читали



База не может читать запросы, т.к. занята отправкой ответов

Batch INSERT в реальности

```
PARSE S_1 as ...;
  BIND/EXEC
  BIND/EXEC
SYNC ← flush & ожидание ответа
  BIND/EXEC
  BIND/EXEC
SYNC ← flush & ожидание ответа
  ...
```

TCP deadlock avoidance

- PgJDBC разбавляет batch операции командой SYNC

TCP deadlock avoidance

- PgJDBC разбавляет batch операции командой SYNC
- Больше SYNC’ов → медленнее работает

Ужасы нашего городка

Меняем 1 строку, и скорость работы batch insert возрастает в 10 раз:

https://github.com/pgjdbc/pgjdbc/pull/380

```
- static int QUERY_FORCE_DESCRIBE_PORTAL = 128;
+ static int QUERY_FORCE_DESCRIBE_PORTAL = 512;
```

Ужасы нашего городка

Меняем 1 строку, и скорость работы batch insert возрастает в 10 раз:

https://github.com/pgjdbc/pgjdbc/pull/380

```
- static int QUERY_FORCE_DESCRIBE_PORTAL = 128;
+ static int QUERY_FORCE_DESCRIBE_PORTAL = 512;
...
// оказалось, значение 128 уже было занято
  static int QUERY_DISALLOW_BATCHING = 128;
```

Доверяй, но замеряй

- Java 1.8u40+
- Core i7 2.6Ghz
- Java microbenchmark harness
- PostgreSQL 9.5

Тестируемые запросы: INSERT

pgjdbc/ubenchmark/InsertBatch.java

```
insert into batch_perf_test(a, b, c)
  values (?, ?, ?)
```

Тестируемые запросы: INSERT

pgjdbc/ubenchmark/InsertBatch.java

```
insert into batch_perf_test(a, b, c)
  values (?, ?, ?)
```

Тестируемые запросы: INSERT

pgjdbc/ubenchmark/InsertBatch.java

```
insert into batch_perf_test(a, b, c)
  values (?, ?, ?), (?, ?, ?), (?, ?,
?), (?, ?, ?), (?, ?, ?), (?, ?, ?),
(?, ?, ?), (?, ?, ?), (?, ?, ?), ...;
```

Тестируемые запросы: COPY

pgjdbc/ubenchmark/InsertBatch.java

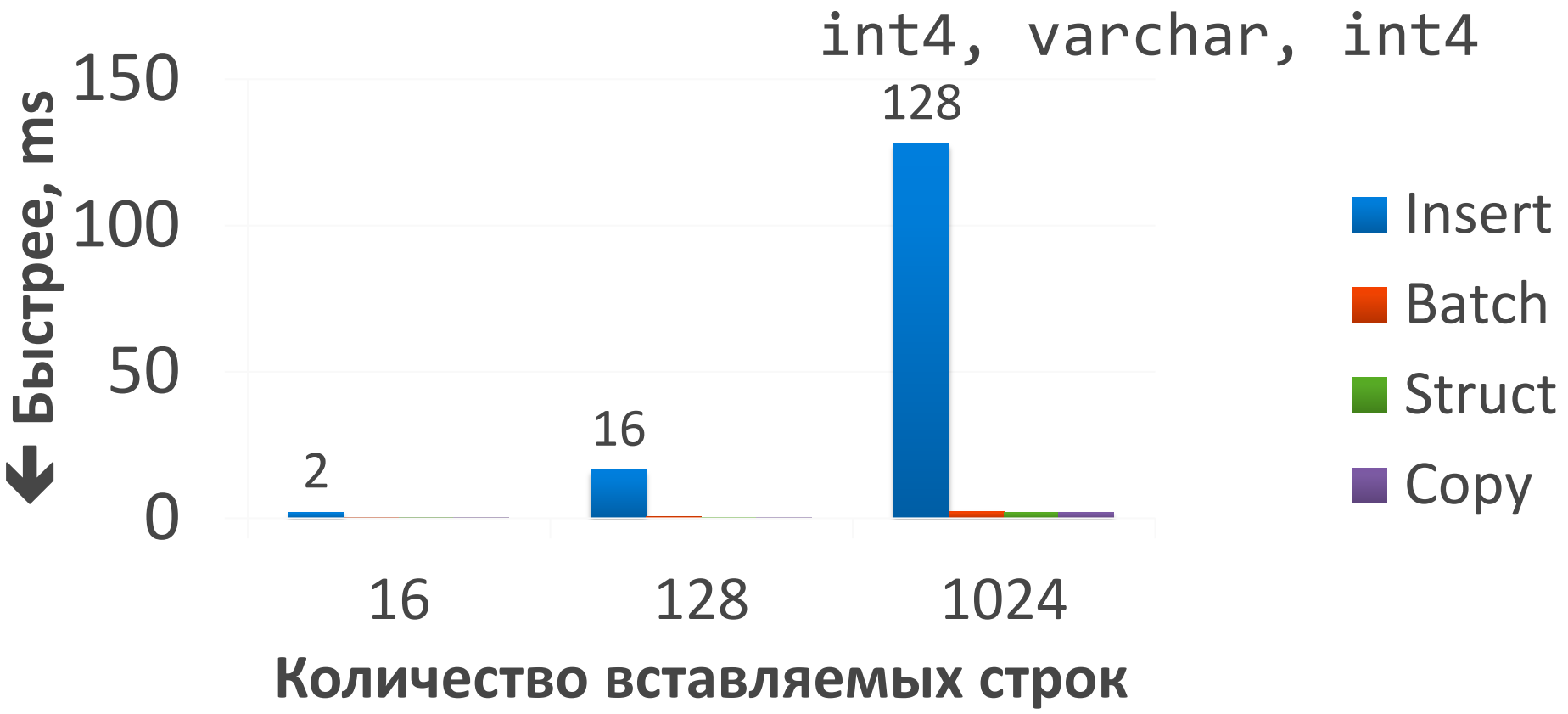
```
COPY batch_perf_test FROM STDIN
1    s1   1
2    s2   2
3    s3   3
...
```

Тестируемые запросы: ручные структуры

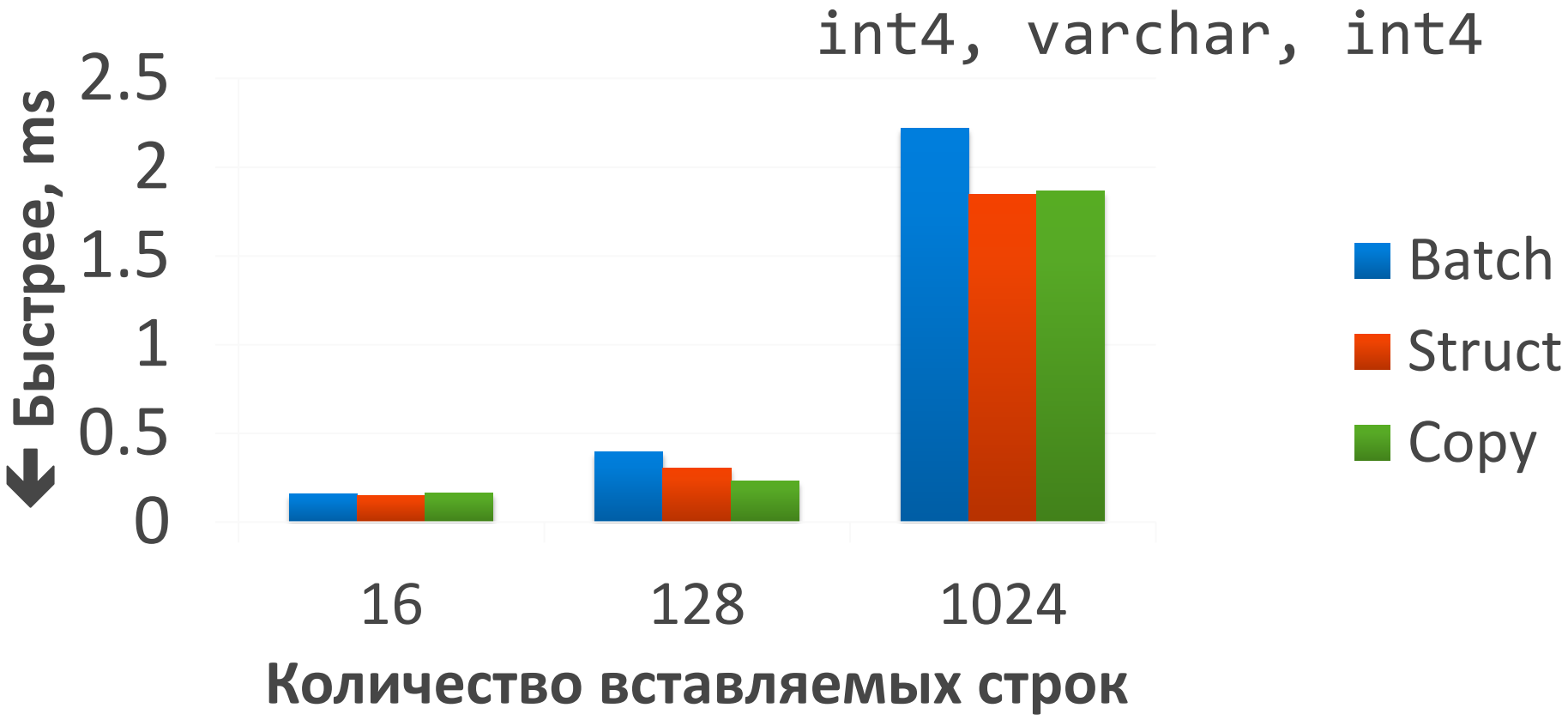
pgjdbc/ubenchmark/InsertBatch.java

```
insert into batch_perf_test
 select * from
 unnest('{"(1,s1,1)","(2,s2,2)",
  "(3,s3,3)"}'::batch_perf_test[])
```

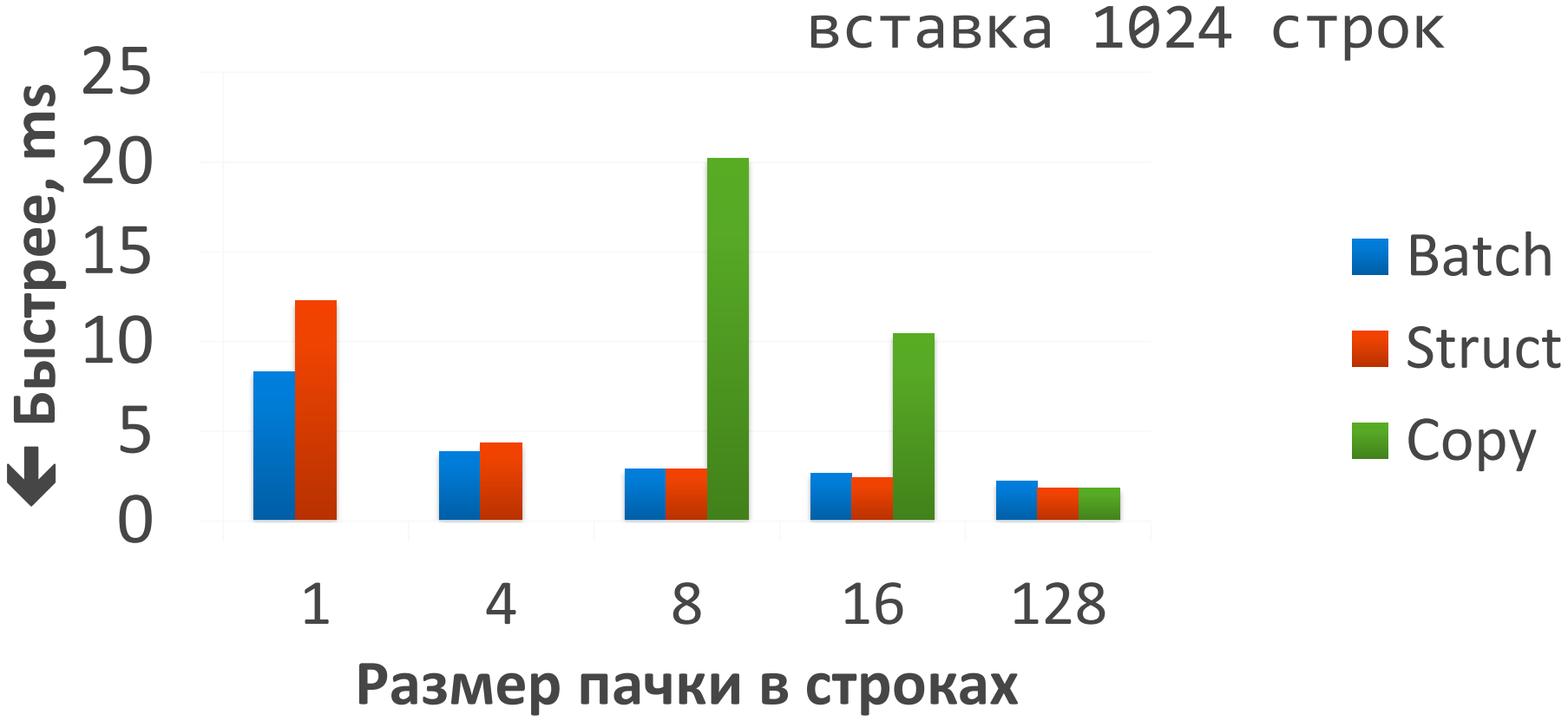
Нужно использовать batch, ваш К.О.



COPY – это хорошо



COPY – это хорошо



Заключение

- `PreparedStatement` – наше всё
- `EXPLAIN ANALYZE` нужно делать 6 раз
- `+0` и `OFFSET 0` по вкусу

О себе

- Владимир Ситников, @VladimirSitnikv
- Инженер по производительности в NetCracker
- 10 лет опыта с Java/SQL
- PgJDBC committer

Вопросы?

NetCracker®

Владимир Ситников,
PgConf 2016