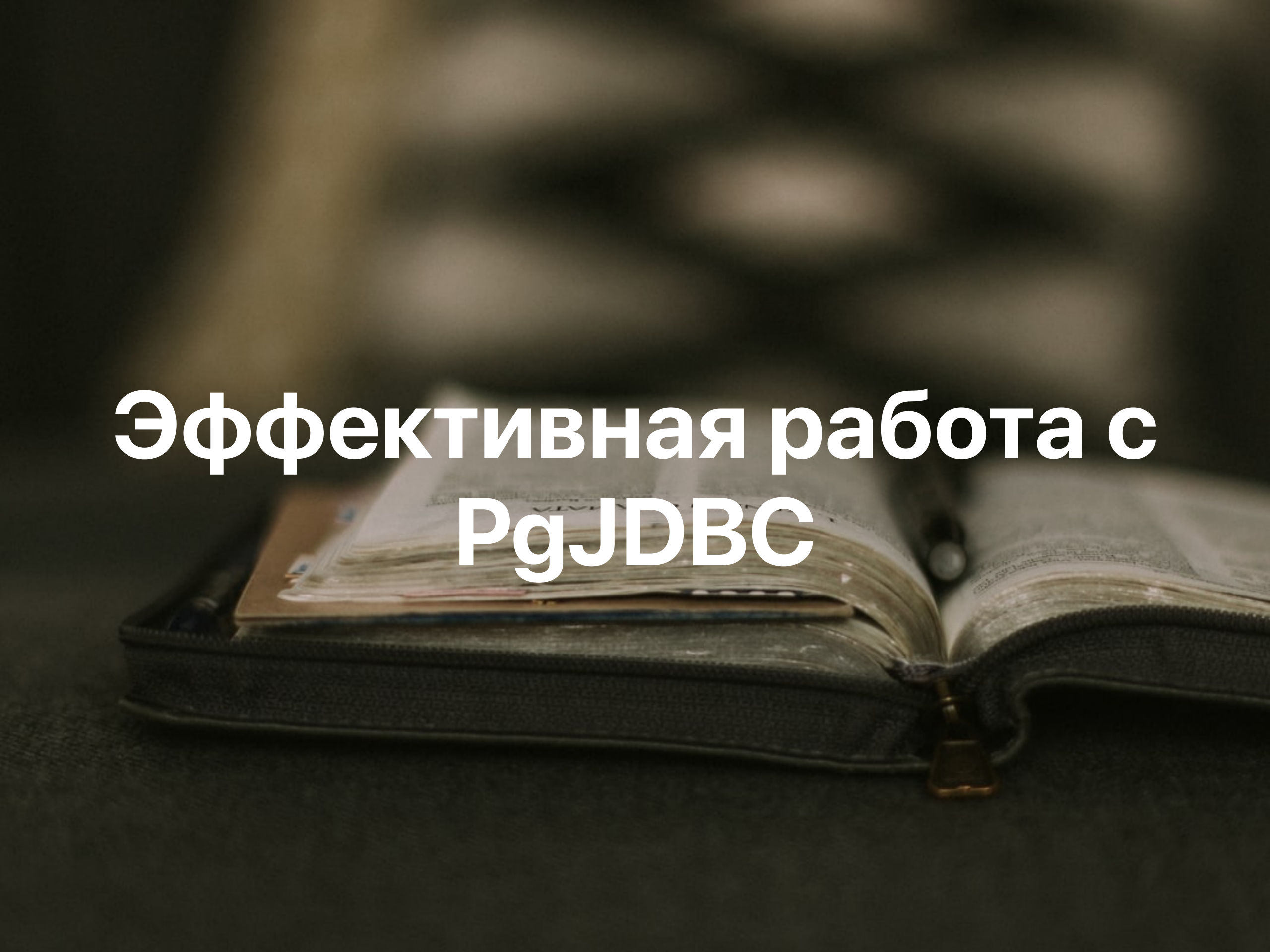
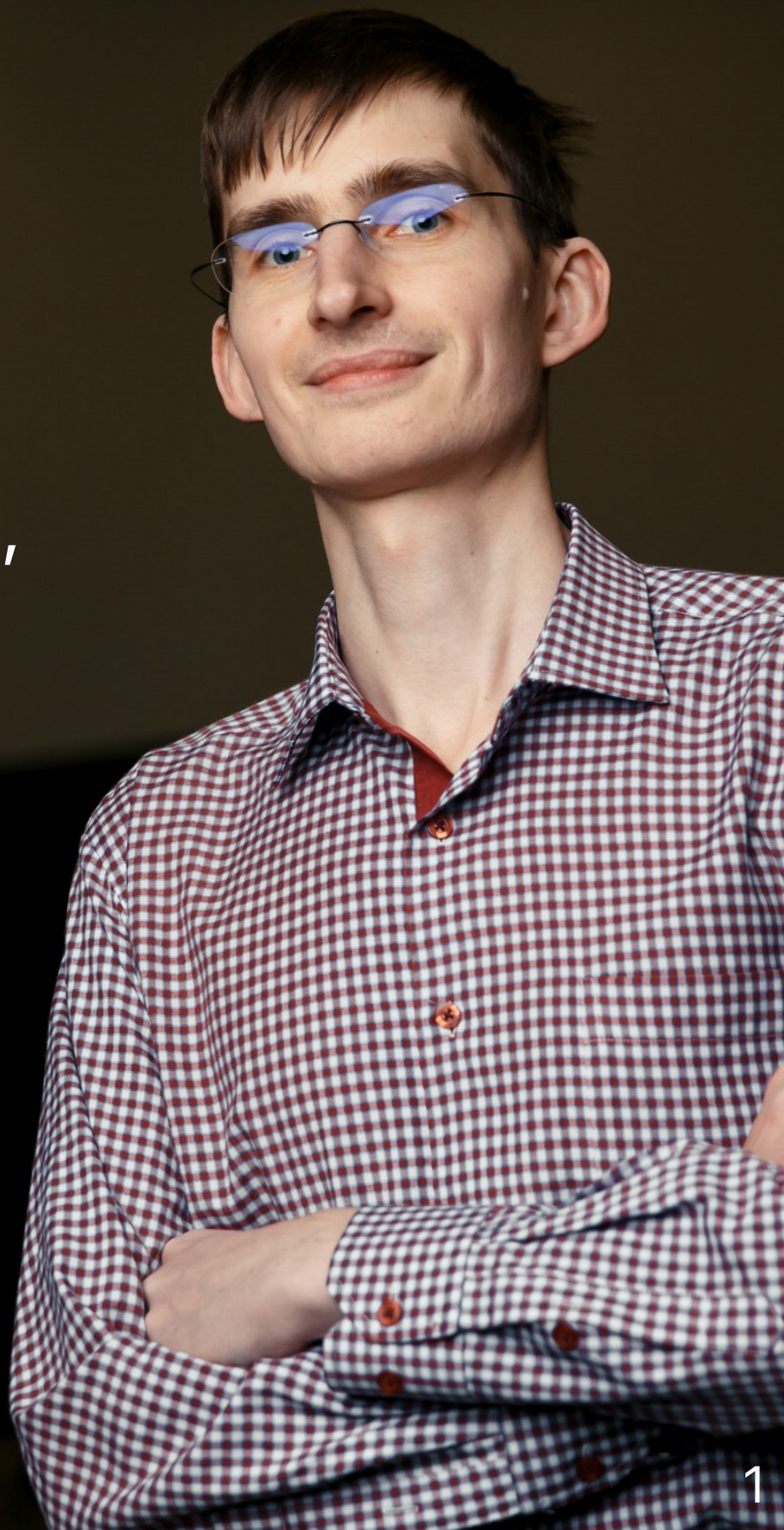


Эффективная работа с PgJDBC

Владимир Ситников

Performance engineer
Pgjdbc, JMeter committer
Член программных коммитетов
JPoint, Joker, Heisenbug, DevOops,
SmartData

VladimirSitnikv
VladimirSitnikv
sitnikov.vladimir@gmail.com



- Что делает драйвер PgJDBC

- Что делает драйвер PgJDBC
- Как он это делает

- Что делает драйвер PgJDBC
- Как он это делает
- Что он мог бы делать

Подключаемся

```
org.postgresql.util.PSQLException:
Connection attempt timed out.
```

```
org.postgresql.util.PSQLException:
Connection attempt timed out.
```

- Не смогли подключиться к серверу?

```
org.postgresql.util.PSQLException:
Connection attempt timed out.
```

- Не смогли подключиться к серверу?
- Подключились, но сервер не ответил?

```
org.postgresql.util.PSQLException:
Connection attempt timed out.
```

- Не смогли подключиться к серверу?
- Подключились, но сервер не ответил?
- Попробовали несколько реплик, но всё равно нет ответа?

```
DriverManager.getConnection(
  "jdbc:postgresql://db:5432/postgres"
)
```

```
DriverManager.getConnection(
  "jdbc:postgresql://db:5432/postgres"
)

DriverManager.getConnection(
  "jdbc:postgresql://db1:5432,db2:5432/postgres"
)
```

```
DriverManager.getConnection(
  "jdbc:postgresql://db:5432/postgres"
)

// loginTimeout — общий лимит, со всеми репликами
//   default: 0 (unlim, sec)
DriverManager.getConnection(
  "jdbc:postgresql://db1:5432,db2:5432/postgres"
)
```

```
DriverManager.getConnection(
  "jdbc:postgresql://db:5432/postgres"
)

// loginTimeout — общий лимит, со всеми репликами
//   default: 0 (unlim, sec)
DriverManager.getConnection(
  // connectTimeout — лимит на socket.connect()
  //   default: 10 (sec)
  "jdbc:postgresql://db1:5432,db2:5432/postgres"
)
```

```
)

// loginTimeout — общий лимит, со всеми репликами
//   default: 0 (unlim, sec)
DriverManager.getConnection(
  // connectTimeout — лимит на socket.connect()
  //   default: 10 (sec)
  "jdbc:postgresql://db1:5432,db2:5432/postgres"
    // socketTimeout — лимит на socket.read/write
    //   default: 0 (unlim, sec)
)
```

```
DriverManager.getConnection(
  "jdbc:postgresql://db:5432/postgres"
)

// loginTimeout — общий лимит, со всеми репликами
//   default: 0 (unlim, sec)
DriverManager.getConnection(
  // connectTimeout — лимит на socket.connect()
  //   default: 10 (sec)
  "jdbc:postgresql://db1:5432,db2:5432/postgres"
    // socketTimeout — лимит на socket.read/write
    //   default: 0 (unlim, sec)
)
```

```
DriverManager.getConnection(
  "jdbc:postgresql://db:5432/postgres"
)

// loginTimeout — общий лимит, со всеми репликами
//   default: 0 (unlim, sec)
// HikariCP connectionTimeout, default: 30s
DriverManager.getConnection(
  // connectTimeout — лимит на socket.connect()
  //   default: 10 (sec)

  "jdbc:postgresql://db1:5432,db2:5432/postgres"
    // socketTimeout — лимит на socket.read/write
    //   default: 0 (unlim, sec)
)
```

```
DriverManager.getConnection(
  "jdbc:postgresql://db:5432/postgres"
)

// loginTimeout — общий лимит, со всеми репликами
//   default: 0 (unlim, sec)
// HikariCP connectionTimeout, default: 30s
// loginTimeout должно хватать на все реплики
DriverManager.getConnection(
  // connectTimeout — лимит на socket.connect()
  //   default: 10 (sec)
  "jdbc:postgresql://db1:5432,db2:5432/postgres"
    // socketTimeout — лимит на socket.read/write
    //   default: 0 (unlim, sec)
)
```

Куда подключимся из `db1:5432,db2:5432`?

Куда подключимся из db1:5432,db2:5432?

- primary?

Куда подключимся из `db1:5432,db2:5432`?

- `primary`?
- `secondary`?

Куда подключимся из `db1:5432,db2:5432`?

- `primary?`
- `secondary?`
- `any?`

`targetServerType` выбирает куда подключимся из `db1:5432,db2:5432`

`targetServerType` выбирает куда подключимся из `db1:5432,db2:5432`

- `primary`, `preferPrimary`

targetServerType выбирает куда подключимся из db1:5432,db2:5432

- primary, preferPrimary
- secondary, preferSecondary

`targetServerType` выбирает куда подключимся из `db1:5432,db2:5432`

- `primary`, `preferPrimary`
- `secondary`, `preferSecondary`
- `any` ⬅ default 🤷

Как найти primary сервер?

Как найти primary сервер?

Как найти primary сервер?

- Перебираем в указанном порядке?

Как найти primary сервер?

- Перебираем в указанном порядке?
- Запоминаем кто был `primary` прошлый раз?

PgJDBC запоминает состояние `primary`, `secondary` на 10 секунд (`hostRecheckSeconds`) и пробует хосты в указанном порядке

PgJDBC запоминает состояние `primary`, `secondary` на 10 секунд (`hostRecheckSeconds`) и пробует хосты в указанном порядке

- Наверное, вы захотите указать `loadBalanceHosts=true`

PgJDBC запоминает состояние `primary`, `secondary` на 10 секунд (`hostRecheckSeconds`) и пробует хосты в указанном порядке

- Наверное, вы захотите указать `loadBalanceHosts=true`
- Возможно, подключаться стоит в параллель: https://github.com/pgjdbc/pgjdbc/issues/3197

Так что же значит
`Connection attempt timed out?`

Так что же значит `Connection attempt timed out`?

Ошибку нужно сделать более понятной:
https://github.com/pgjdbc/pgjdbc/issues/3189

Выполняем запросы

```
var ps = con.prepareStatement(
  "select * from users where name = ?"
);
```

```
var ps = con.prepareStatement(
  "select * from users where name = ?"
);

ps.setString(1, "Джек");
```

```
var ps = con.prepareStatement(
  "select * from users where name = ?"
);

ps.setString(1, "Джек");

ResultSet rs = ps.executeQuery();
```

Основные API для выполнения запросов

Основные API для выполнения запросов

- simple query protocol

Основные API для выполнения запросов

- simple query protocol
- extended query protocol

Simple Query

Отправляем SQL на сервер, и ждём результат

Simple Query

Simple Query

- Каждый раз отправляем SQL на сервер

Simple Query

- Каждый раз отправляем SQL на сервер
- База его каждый раз анализирует

Extended Query

Extended Query

- parse — отправляем SQL на сервер

Extended Query

- parse — отправляем SQL на сервер
- bind — указываем параметры и формат ответа

Extended Query

- parse — отправляем SQL на сервер
- bind — указываем параметры и формат ответа
- describe — выясняем "набор результирующих полей"

Extended Query

- parse — отправляем SQL на сервер
- bind — указываем параметры и формат ответа
- describe — выясняем "набор результирующих полей"
- execute — выполняем запрос

Как посмотреть что драйвер отправляет в базу?

Как посмотреть что драйвер отправляет в базу?

- Wireshark

Как посмотреть что драйвер отправляет в базу?

- Wireshark
- Логи pgjdbc: https://jdbc.postgresql.org/documentation/logging/

```
# Specify the handler, the handlers will be installed during VM startup
handlers = java.util.logging.FileHandler

# Default global logging level
.level = OFF

# Default file output is in user's home directory
java.util.logging.FileHandler.pattern = %h/pgjdbc%u.log
java.util.logging.FileHandler.limit = 5000000
java.util.logging.FileHandler.count = 20

java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.FileHandler.level = FINEST

java.util.logging.SimpleFormatter.format = %1$tY-%1$tm-%1$td %1$tH:%1$tM:%1$tS %4$s %2$s %5$s%6$s%n

# Facility specific properties.
org.postgresql.level = FINEST
```

Extended Query

Extended Query

- быстрее — не шлём SQL каждый раз

Extended Query

- быстрее — не шлём SQL каждый раз
- быстрее — поддерживает бинарные форматы

Extended Query

- быстрее — не шлём SQL каждый раз
- быстрее — поддерживает бинарные форматы
- надёжнее — меньше риск SQL injection

По умолчанию, PgJDBC использует `extended` протокол, но можно настроить через `preferQueryMode`:

По умолчанию, PgJDBC использует `extended` протокол, но можно настроить через `preferQueryMode`:

- `extended` — только extended (default)

По умолчанию, PgJDBC использует `extended` протокол, но можно настроить через `preferQueryMode`:

- `extended` — только extended (default)
- `simple` — только simple, даже для `PreparedStatement`

По умолчанию, PgJDBC использует `extended` протокол, но можно настроить через `preferQueryMode`:

- `extended` — только extended (default)
- `simple` — только simple, даже для `PreparedStatement`
- `extendedForPrepared` — extended только для `PreparedStatement`

CVE 10: CVE-2024-1597

Для воспроизведения CVE-2024-1597 нужно

Для воспроизведения CVE-2024-1597 нужно

- `preferQueryMode=simple`

Для воспроизведения CVE-2024-1597 нужно

- `preferQueryMode=simple`
- Весьма необычный запрос

```
select *
  from users
 where id = ? and status = ?
```

```
select *
  from users
 where id = ? and status = ?

// preferQueryMode=simple
select *
  from users
 where id = 42 and status = 'active'
```

Нормальный же запрос?

```
select *
  from users
 where id = ? and status = ?

// preferQueryMode=simple
select *
  from users
 where id = 42 and status = 'active'
```

Нормальный же запрос?

```
 where id = 42 and status = 'active'

// CVE: если передать отрицательное число,
//      то получится комментарий
select *
  from users
 where id = -? and status = ?
```

CVE исправили во всех версиях

- 42.7.2
- 42.6.1
- 42.5.5
- 42.4.4
- 42.3.9
- 42.2.28

Схема нумерации версий PgJDBC

Схема нумерации версий PgJDBC

- Приоритет — обратная совместимость

Схема нумерации версий PgJDBC

- Приоритет — обратная совместимость
- Добавили фичу — будет minor version

Схема нумерации версий PgJDBC

- Приоритет — обратная совместимость
- Добавили фичу — будет minor version
- Поправили ошибку, но возможны несовместимости? — minor version

Схема нумерации версий PgJDBC

- Приоритет — обратная совместимость
- Добавили фичу — будет minor version
- Поправили ошибку, но возможны несовместимости? — minor version
- Security — во всех версиях (Java 7+)

Server-side statement причиняет много радости:

Server-side statement причиняет много радости:

- `prepared statement "S_1" does not exist`

Server-side statement причиняет много радости:

- `prepared statement "S_1" does not exist`
- `cached plan must not change result type`

Server-side statement причиняет много радости:

- `prepared statement "S_1" does not exist`
- `cached plan must not change result type`
- `OutOfMemoryError`

Extended Query vs OutOfMemory

Extended Query vs OutOfMemory

- Запросы и планы хранятся на сервере

Extended Query vs OutOfMemory

- Запросы и планы хранятся на сервере
- У каждого соединения память своя

Extended Query vs OutOfMemory

- Запросы и планы хранятся на сервере
- У каждого соединения память своя
- Много сложных запросов * много соединений = много памяти

PgJDBC создаёт server-prepared только для частых запросов

PgJDBC создаёт server-prepared только для частых запросов

- Запрос нужно запустить 5+ раз (`prepareThreshold`)

PgJDBC создаёт server-prepared только для частых запросов

- Запрос нужно запустить 5+ раз (`prepareThreshold`)
- Размера кэша на клиенте должно хватить
  - `preparedStatementCacheQueries` (256) — количество запросов
  - `preparedStatementCacheSizeMiB` (5) — суммарная длина SQL

prepareThreshold

prepareThreshold

- 5 — на 6-ой раз создадим server-side statement

prepareThreshold

- 5 — на 6-ой раз создадим server-side statement
- 4 — на 5-ой раз создадим server-side statement

prepareThreshold

- 5 — на 6-ой раз создадим server-side statement
- 4 — на 5-ой раз создадим server-side statement
- 0 — вообще не будем создавать server-side statement

prepareThreshold

- `5` — на 6-ой раз создадим server-side statement
- `4` — на 5-ой раз создадим server-side statement
- `0` — вообще не будем создавать server-side statement
- `-1` — на первом же выполнении создадим server-side statement

Память на стороне PostgreSQL можно отслеживать по

Память на стороне PostgreSQL можно отслеживать по

- PG 17+ EXPLAIN (memory)

Память на стороне PostgreSQL можно отслеживать по

- PG 17+ `EXPLAIN (memory)`
- `gdb` + `MemoryContextStats()`:

https://wiki.postgresql.org/wiki/Developer_FAQ#Examining_backend_memory_use

Память на стороне PostgreSQL можно отслеживать по

- PG 17+ `EXPLAIN (memory)`
- `gdb` + `MemoryContextStats()`:
  https://wiki.postgresql.org/wiki/Developer_FAQ#Examining_backend_memory_use
- `ps aux | grep postgres`

Память на стороне PostgreSQL можно отслеживать по

- PG 17+ `EXPLAIN (memory)`
- `gdb` + `MemoryContextStats()`:
  https://wiki.postgresql.org/wiki/Developer_FAQ#Examining_backend_memory_use
- `ps aux | grep postgres`
- Падению базы

cached plan must not change result type

cached plan must not change result type

- Подготовили server-side statement: `select id, name ...`

cached plan must not change result type

- Подготовили server-side statement: `select id, name ...`
- Кто-то сделал `alter table modify column`

cached plan must not change result type

- Подготовили server-side statement: `select id, name ...`
- Кто-то сделал `alter table modify column`
- Выполняем запрос ещё раз и получаем ошибку

Решение cached plan must not ...

Решение cached plan must not ...

- Не делать alter table

Решение cached plan must not ...

- Не делать `alter table`
- Выполнить `deallocate all` (`pjgdbc` сбросит свой кэш)

При использовании server-prepared statement, у базы есть выбор:

При использовании server-prepared statement, у базы есть выбор:

- Смотреть на значения параметров и строить лучший план

При использовании server-prepared statement, у базы есть выбор:

- Смотреть на значения параметров и строить лучший план
- Не смотреть на значения параметров и строить какой-то план

При использовании server-prepared statement, у базы есть выбор:

- Смотреть на значения параметров и строить лучший план
- Не смотреть на значения параметров и строить какой-то план
- PostgreSQL первые 5 раз всегда смотрит

При использовании server-prepared statement, у базы есть выбор:

- Смотреть на значения параметров и строить лучший план
- Не смотреть на значения параметров и строить какой-то план
- PostgreSQL первые 5 раз всегда смотрит
- На 6+ выполнении всегда будет либо custom либо generic план

```sql
create table plan_flipper(
    id int,
    skewed int,
    non_skewed int,
    padding varchar
);
```

```
create table plan_flipper(
    id int,
    skewed int,
    non_skewed int,
    padding varchar
);

insert into plan_flipper(id, skewed, non_skewed, padding)
  select g.i as id
       , 0 as skewed
       , mod(g.i, 100000) as non_skewed
       , lpad(' ', 100, ' ')
    from generate_series(1,1000000) as g(i);
```

```
);

insert into plan_flipper(id, skewed, non_skewed, padding)
  select g.i as id
       , 0 as skewed
       , mod(g.i, 100000) as non_skewed
       , lpad(' ', 100, ' ')
    from generate_series(1,1000000) as g(i);

insert into plan_flipper(id, skewed, non_skewed, padding)
  select g.i as id
       , g.i as skewed
       , mod(g.i, 100000) as non_skewed
       , lpad(' ', 100, ' ')
    from generate_series(1000001,2000000) as g(i);
```

```
    skewed int,
    non_skewed int,
    padding varchar
);

insert into plan_flipper(id, skewed, non_skewed, padding)
  select g.i as id
       , 0 as skewed
       , mod(g.i, 100000) as non_skewed
       , lpad(' ', 100, ' ')
    from generate_series(1,1000000) as g(i);

insert into plan_flipper(id, skewed, non_skewed, padding)
  select g.i as id
       , g.i as skewed
       , mod(g.i, 100000) as non_skewed
       , lpad(' ', 100, ' ')
    from generate_series(1000001,2000000) as g(i);
```

```
select *
  from plan_flipper
 where skewed = 0
   and non_skewed = 42
```

```
explain
select *
  from plan_flipper
 where skewed = 0
   and non_skewed = 42
```

```
explain
select *
  from plan_flipper
 where skewed = 0
   and non_skewed = 42

...
...... Seq Scan on plan_flipper
```

```
create index skewed__flipper ON plan_flipper(skewed);
```

```
create index skewed__flipper ON plan_flipper(skewed);
create index non_skewed__flipper ON plan_flipper(non_skewed);
```

```
create index skewed__flipper ON plan_flipper(skewed);
create index non_skewed__flipper ON plan_flipper(non_skewed);
commit;
vacuum analyze plan_flipper;
```

```
explain (analyze, costs off)
  select *
  from plan_flipper
  where skewed = 0
    and non_skewed = 42
```

```
explain (analyze, costs off)
  select *
  from plan_flipper
  where skewed = 0
    and non_skewed = 42

Bitmap Heap Scan on plan_flipper
       (actual time=0.020..0.047 rows=10 loops=1)
  Recheck Cond: (non_skewed = 42)
  Filter: (skewed = 0)
  Rows Removed by Filter: 10
  Heap Blocks: exact=20
  ->  Bitmap Index Scan on non_skewed__flipper
             (actual time=0.013..0.013 rows=20 loops=1)
        Index Cond: (non_skewed = 42)
Execution Time: 0.061 ms
```

```
explain (analyze, costs off)
  select *
  from plan_flipper
  where skewed = 0
    and non_skewed = 42

Bitmap Heap Scan on plan_flipper
       (actual time=0.020..0.047 rows=10 loops=1)
  Heap Blocks: exact=20
  ->  Bitmap Index Scan on non_skewed__flipper
            (actual time=0.013..0.013 rows=20 loops=1)
        Index Cond: (non_skewed = 42)
Execution Time: 0.061 ms
```

```
prepare stmt(int, int)
  select *
    from plan_flipper
   where skewed = $1
     and non_skewed = $2
```

```
prepare stmt(int, int) as
  select *
    from plan_flipper
   where skewed = $1
     and non_skewed = $2

explain (analyze, buffers, costs off)
  execute flipper(0, 42);
```

```
prepare stmt(int, int) as
  select *
    from plan_flipper
   where skewed = $1
     and non_skewed = $2

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);
```

```
   where skewed = $1
     and non_skewed = $2

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);
```

```
explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);
```

```
explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);
```

```
explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);
```

```
explain (analyze, buffers, costs off)
  execute flipper(0, 42);

explain (analyze, buffers, costs off)
  execute flipper(0, 42);

Index Scan using skewed__flipper on plan_flipper
      (actual time=0.106..152.385 rows=10 loops=1)
  Index Cond: (skewed = $1)
  Filter: (non_skewed = $2)
  Rows Removed by Filter: 999990
  Buffers: shared hit=10 read=19016
Planning Time: 0.131 ms
Execution Time: 152.418 ms
```

Как фиксировать custom или generic план?

Как фиксировать custom или generic план?

- plan_cache_mode=auto, force_custom_plan, force_generic_plan

Как фиксировать custom или generic план?

- `plan_cache_mode=auto, force_custom_plan, force_generic_plan`
- Удаляем лишние индексы

Как фиксировать custom или generic план?

- `plan_cache_mode=auto, force_custom_plan, force_generic_plan`
- Удаляем лишние индексы
- Добавляем `skewed+0`, чтобы не использовался индекс

Итого: 5 + 5 = 10

Итого: 5 + 5 = 10

- PgJDBC первые 5 выполнений не использует server-prepared

Итого: 5 + 5 = 10

- PgJDBC первые 5 выполнений не использует server-prepared
- PostgreSQL первые 5 раз анализирует generic vs custom

Server-prepared statement зависит от типов параметров

```
var ps = con.prepareStatement(
  "select * from users where name = ?"
);

ps.setString(1, "Джек");

ResultSet rs = ps.executeQuery();
```

```
);

ps.setString(1, "Джек");

ResultSet rs = ps.executeQuery();

ps.setInt(1, 42);


rs = ps.executeQuery();
```

Кто же так делает?

```
void setParameterValue(
        PreparedStatement ps,
        int index,
        Object value) {

}
```

```
void setParameterValue(
      PreparedStatement ps,
      int index,
      Object value) {
  if (value instanceof String str) {
     ps.setString(index, str);
  }
}
```

```
void setParameterValue(
        PreparedStatement ps,
        int index,
        Object value) {
   if (value instanceof String str) {
      ps.setString(index, str);
   } else if (value instanceof Integer int4) {
      ps.setInt(index, int4);
   }
}
```

```
void setParameterValue(
      PreparedStatement ps,
      int index,
      Object value) {
  if (value instanceof String str) {
     ps.setString(index, str);
  } else if (value instanceof Integer int4) {
     ps.setInt(index, int4);
  } else {
     ps.setNull(index, Types.OTHER);
  }
}
```

Types.OTHER приводит к чехарде с типами

```
void setParameterValue(
       PreparedStatement ps,
       int index,
       Object value) {
  if (value instanceof String str) {
     ps.setString(index, str);
  } else if (value instanceof Integer int4) {
     ps.setInt(index, int4);
  } else {
     ps.setNull(index, Types.OTHER);
  }
}
```

```sql
create table users(
  id bigint primary key,
  name text
);
```

```sql
create table users(
  id bigint primary key,
  name text
);

insert into users(id, name)
  select g.i, g.i
    from generate_series(1, 100000) as s(x);
```

```
);

insert into users(id, name)
  select g.i, g.i
    from generate_series(1, 100000) as s(x);

prepare s_num(numeric) as
  select name
    from users
   where id = $1;

explain (analyze, buffers, costs off)
  execute s_num(42);
```

```
    from users
   where id = $1;

explain (analyze, buffers, costs off)
  execute s_num(42);

Seq Scan on users
    (actual time=0.026..82.212 rows=1 loops=1)
  Filter: ((id)::numeric = '42'::numeric)
  Rows Removed by Filter: 999999
  Buffers: shared hit=5406
Planning Time: 0.116 ms
Execution Time: 82.228 ms
```

```
prepare s_num(numeric) as
  select name
    from users
   where id = $1;

Seq Scan on users
    (actual time=0.026..82.212 rows=1 loops=1)
  Filter: ((id)::numeric = '42'::numeric)
  Rows Removed by Filter: 999999
  Buffers: shared hit=5406
Execution Time: 82.228 ms
```

От типов зависит план выполнения

При "внезапных" сменах типов PgJDBC пересоздаёт server-prepared statement

Откуда же брать типы параметров?

Откуда же брать типы параметров?

- По-хорошему, типы нужно указывать явно

Откуда же брать типы параметров?

- По-хорошему, типы нужно указывать явно
- Особенно часто проблема на `null`

Откуда же брать типы параметров?

- По-хорошему, типы нужно указывать явно
- Особенно часто проблема на `null`
- Иногда можно спросить у базы: `PreparedStatement.getParameterMetaData()`

```
select name from users where id = ?;
```

```
select name from users where id = ?;

select abs(?);
```

```sql
select name from users where id = ?;

select abs(?); -- int4? double?

select ?; -- :)
```

PreparedStatement.getParameterMetaData()

`PreparedStatement.getParameterMetaData()`

- Обычно может вернуть типы парамеров

`PreparedStatement.getParameterMetaData()`

- Обычно может вернуть типы парамеров
- Типы могут зависеть от уже установленных параметров

`PreparedStatement.getParameterMetaData()`

- Обычно может вернуть типы парамеров
- Типы могут зависеть от уже установленных параметров
- PgJDBC сейчас не кэширует `parameterMetadata`
  - https://github.com/pgjdbc/pgjdbc/issues/621

`PreparedStatement.getParameterMetaData()`

- Обычно может вернуть типы парамеров
- Типы могут зависеть от уже установленных параметров
- PgJDBC сейчас не кэширует `parameterMetadata`
  - https://github.com/pgjdbc/pgjdbc/issues/621
- Spring вызывает `getParameterMetaData` для каждого неизвестного `null`

Timestamp

Timestamp

- java.sql.Date

Timestamp

- java.sql.Date
- java.sql.Timestamp

Timestamp

- `java.sql.Date`
- `java.sql.Timestamp`
- `setTimestamp(...)`, `setTimestamp(...)`

Timestamp

- `java.sql.Date`
- `java.sql.Timestamp`
- `setTimestamp(...)`, `setTimestamp(...)`
- Эти методы очень плохо подходят для `local timestamp` (без часового пояса)

Timestamp

- `java.sql.Date`
- `java.sql.Timestamp`
- `setTimestamp(...)`, `setTimestamp(...)`
- Эти методы очень плохо подходят для `local timestamp` (без часового пояса)
- Например, непонятно что хотят от `setTimestamp`: `date`, `timestamp`, ...

Timestamp: рекомендации

Timestamp: рекомендации

- `java.time.LocalDate`: дата без времени и без часового пояса

Timestamp: рекомендации

- `java.time.LocalDate`: дата без времени и без часового пояса
- `java.time.LocalDateTime`: дата со временем, без часового пояса

Timestamp: рекомендации

- `java.time.LocalDate`: дата без времени и без часового пояса
- `java.time.LocalDateTime`: дата со временем, без часового пояса
- `java.time.OffsetDateTime`: "момент во времени"

Timestamp: рекомендации

- `java.time.LocalDate`: дата без времени и без часового пояса
- `java.time.LocalDateTime`: дата со временем, без часового пояса
- `java.time.OffsetDateTime`: "момент во времени"
- При получении указываем желаемый тип: `rs.getObject(1, OffsetDateTime.class)`

Batch insert

```
var ps = con.prepareStatement(...);
for(...) {
    ps.setInt(1, ...);
    ps.setString(2, ...);
    ps.addBatch();
}
ps.executeBatch();
```

Batch insert

```
    ps.setInt(1, ...);
    ps.setString(2, ...);
    ps.addBatch();
}
ps.executeBatch();

// На сервер отправится
```

PARSE

Batch insert

```
ps.executeBatch();

// На сервер отправится

PARSE

BIND
EXEC
BIND
EXEC
```

Batch insert

```
PARSE

BIND
EXEC
BIND
EXEC

SYNC ← тут мы ждём ответа от базы
```

Batch insert

```
EXEC

SYNC ← тут мы ждём ответа от базы

// Обрабатываем ответы сервера,
// и отправляем следующую пачку данных
BIND
EXEC
BIND
EXEC

SYNC ← тут мы ждём ответа от базы
```

Для снижения накладных расходов на bind, exec придумали reWriteBatchedInserts

```
insert into users(id, mame) values(?, ?);
```

```
insert into users(id, mame) values(?, ?);

// reWriteBatchedInserts ⇒

insert into users(id, mame)
    values (?, ?), (?, ?);
```

```
insert into users(id, mame) values(?, ?);

// reWriteBatchedInserts ⇒

insert into users(id, mame)
    values (?, ?), (?, ?);

insert into users(id, mame)
    values (?, ?), (?, ?), (?, ?), (?, ?);
```

```
insert into users(id, mame) values(?, ?);

// reWriteBatchedInserts ⟹

insert into users(id, mame)
    values (?, ?), (?, ?);

insert into users(id, mame)
    values (?, ?), (?, ?), (?, ?), (?, ?);
```

```
// Сейчас
insert into users(id, mame)
    values (?, ?), (?, ?), (?, ?), (?, ?);
```

```
// Сейчас
insert into users(id, mame)
    values (?, ?), (?, ?), (?, ?), (?, ?);

// Услышано на pgconf 2024:
insert into users(id, mame)
    select unnest(?), unnest(?);
```

```
insert into users(id, mame)
    values (?, ?), (?, ?), (?, ?), (?, ?);

// Услышано на pgconf 2024:
insert into users(id, mame)
    select unnest(?), unnest(?);

// Но есть проблемные запросы
insert into users(id, mame)
    values (? + ?, DEFAULT);
```

https://github.com/pgjdbc/pgjdbc/issues/3195

OutOfMemory при выборке

```
ResultSet rs = ps.executeQuery();
while (rs.next()) {
  //
}
```

```
// Выполняем запрос
ResultSet rs = ps.executeQuery();
// Получаем результаты
while (rs.next()) {
  //
}
```

В PG есть возможность выбирать данные постепенно

```
// Выполняем запрос
ResultSet rs = ps.executeQuery();
while (rs.next()) {
  //
}
```

Но после commit server-side statement превращается в тыкву

```
// Выполняем запрос
ResultSet rs = ps.executeQuery();
while (rs.next()) {
  //
}
```

По умолчанию в JDBC autoCommit=true

```
// Где сработает autocommit?
ResultSet rs = ps.executeQuery();
while (rs.next()) {
  //
}
```

По умолчанию в JDBC autoCommit=true

```
// Где сработает autocommit?
// В момент .executeQuery?
ResultSet rs = ps.executeQuery();
while (rs.next()) {
  //
}
// При закрытии statement?
```

```
// Сейчас autocommit сработает при executeQuery.
ResultSet rs = ps.executeQuery();
while (rs.next()) {
  //
}
```

```
// Сейчас autocommit сработает при executeQuery.
// По спецификации — после завершения выборки.
ResultSet rs = ps.executeQuery();
while (rs.next()) {
  //
}
```

Выбираем большое количество строк

Выбираем большое количество строк

- Отключаем `autoCommit`

Выбираем большое количество строк

- Отключаем `autoCommit`
- Указываем `ps.setFetchSize(...)`, `rs.setFetchSize(..)`

Выбираем большое количество строк

- Отключаем `autoCommit`
- Указываем `ps.setFetchSize(...)`, `rs.setFetchSize(..)`
- Можно указывать настройку `defaultRowFetchSize` (42.3+) на уровне соединения

Выбираем большое количество строк

- Отключаем `autoCommit`
- Указываем `ps.setFetchSize(...)`, `rs.setFetchSize(..)`
- Можно указывать настройку `defaultRowFetchSize` (42.3+) на уровне соединения
- Можно указать `adaptiveFetch=true` (42.3+) и драйвер будет адаптировать количество выбираемых строк

Выводы

- Обновляйте pgjdbc

Выводы

- Обновляйте pgjdbc
- Активируйте новые фичи (без SMS!)

Выводы

- Обновляйте pgjdbc
- Активируйте новые фичи (без SMS!)
- Используйте `prepareThreshold ≠ 0`

Выводы

- Обновляйте pgjdbc
- Активируйте новые фичи (без SMS!)
- Используйте `prepareThreshold ≠ 0`
- Делитесь идеями — вместе мы сильнее

Владимир Ситников

Performance engineer
Pgjdbc, JMeter committer
Член программных коммитетов
JPoint, Joker, Heisenbug, DevOops,
SmartData

VladimirSitnikv
VladimirSitnikv
sitnikov.vladimir@gmail.com