Разгоняем вставку Spring + PostgreSQL

Пару слов о себе:

- Фатов Дмитрий
- Более 12 лет в ИТ
- Пишу на Kotlin
- Работаю в Газпромбанке
- Создаем и строим решения на платформе G2

О чем доклад:

Как ускорить вставку данных в PostgreSQL

- Настройки Spring

- Создание собственной прослойки для вставки данных в БД

- Использование кастомных методов PostgreSQL

- Распараллеливание процесса вставки

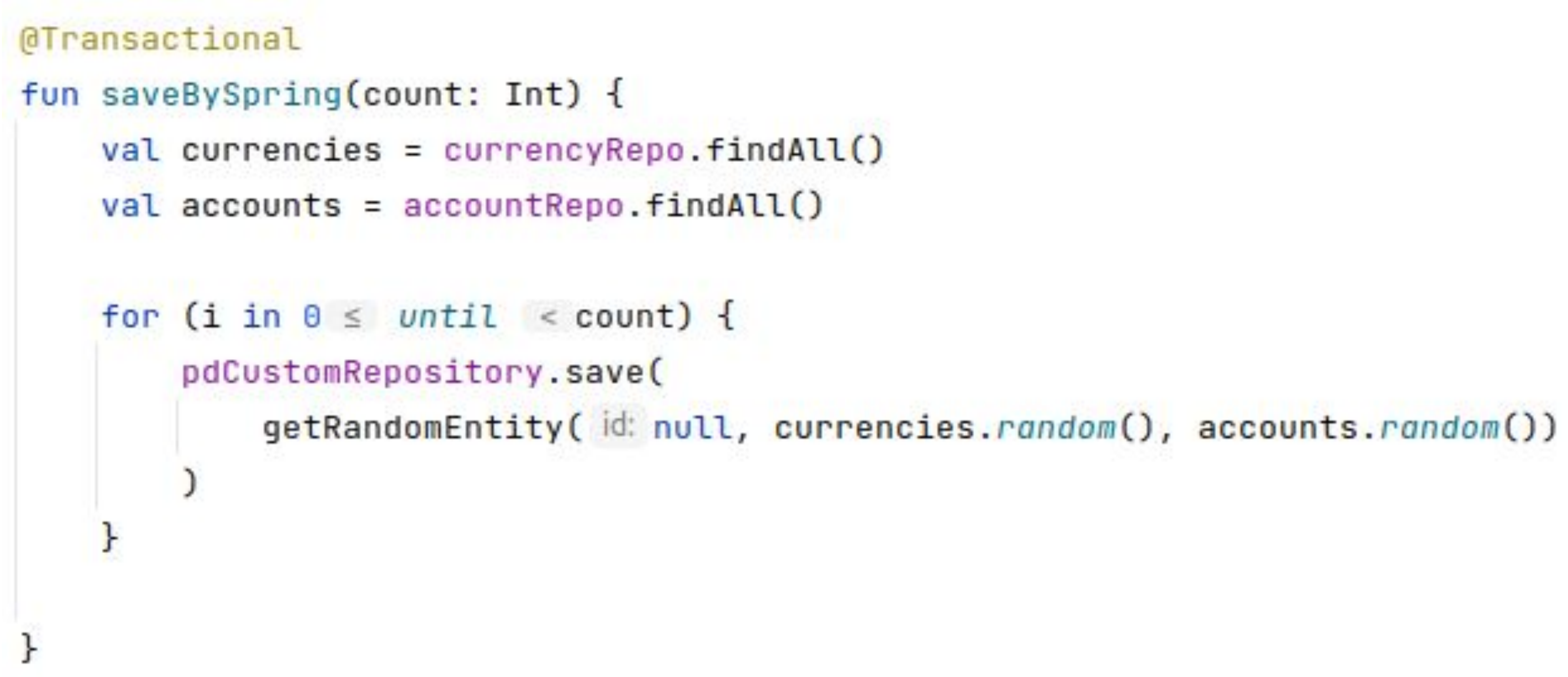
Немного предыстории:

- Разрабатываем SaaS
- Есть внешние интеграции через xml
- Было < 30 000 документов в одной выгрузке
- Стало ~ 2 000 000 документов в одной выгрузке = 4 000 000 записей в БД
- SLA < 5 минут на обмен данными со сторонними системами

Application

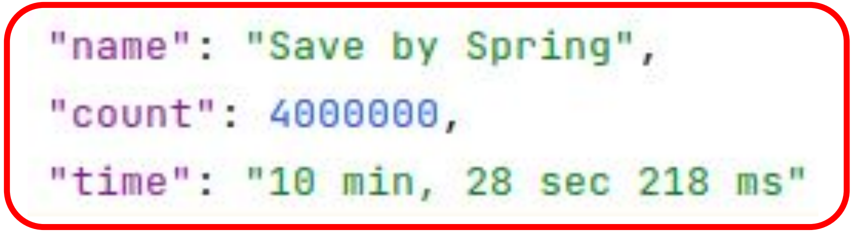
- Code and benchmars: https://github.com/FatovDI/acceleration-insertion-postgresql-joker2023
- Подготовленная БД, размер БД 32 Гб
- 100_000_000 строк в основной таблице с индексами
- Будем тестировать вставку на 4_000_000 записей
- Замеры: 3 итерации прогрева, 5 итераций замеров
- Окружение: java 17, PostgreSQL 14.5

Spring

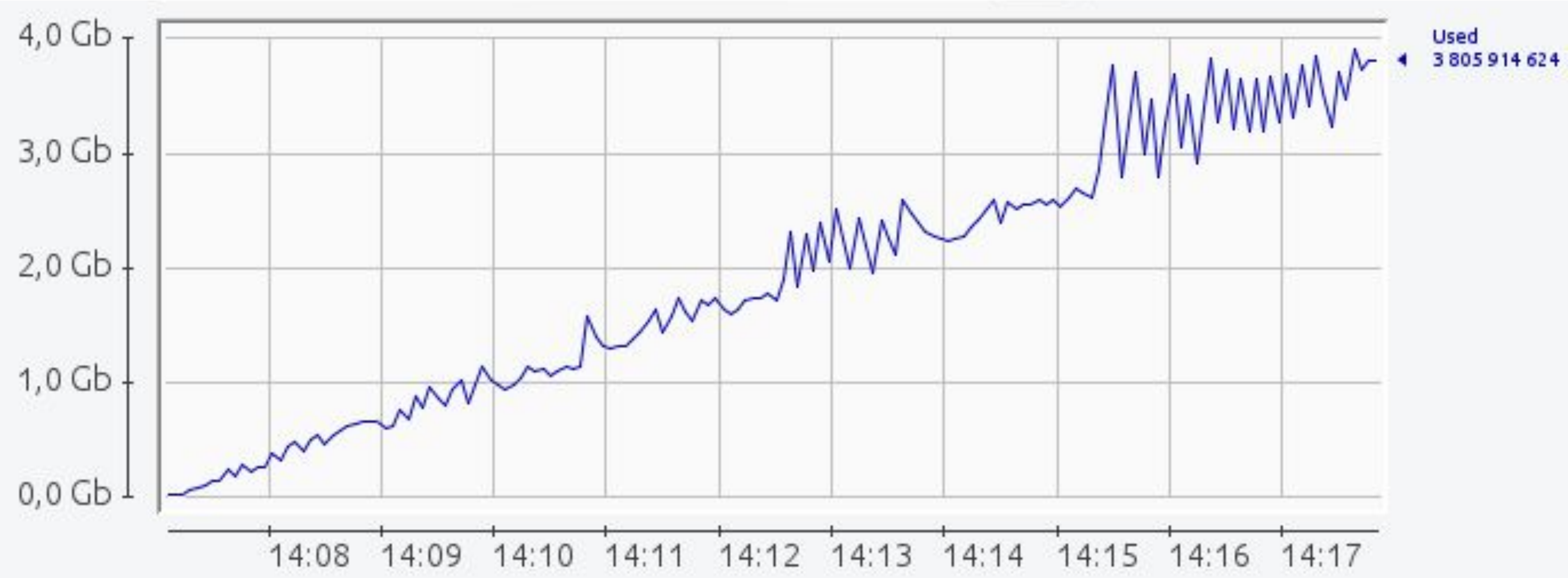


```
@Transactional
fun saveBySpring(count: Int) {
    val currencies = currencyRepo.findAll()
    val accounts = accountRepo.findAll()

    for (i in 0 ≤ until < count) {
        pdCustomRepository.save(
            getRandomEntity( id: null, currencies.random(), accounts.random())
        )
    }

}
```

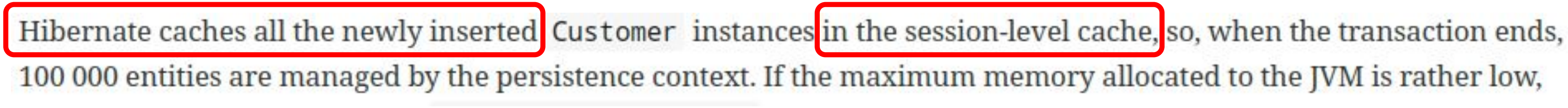
Spring



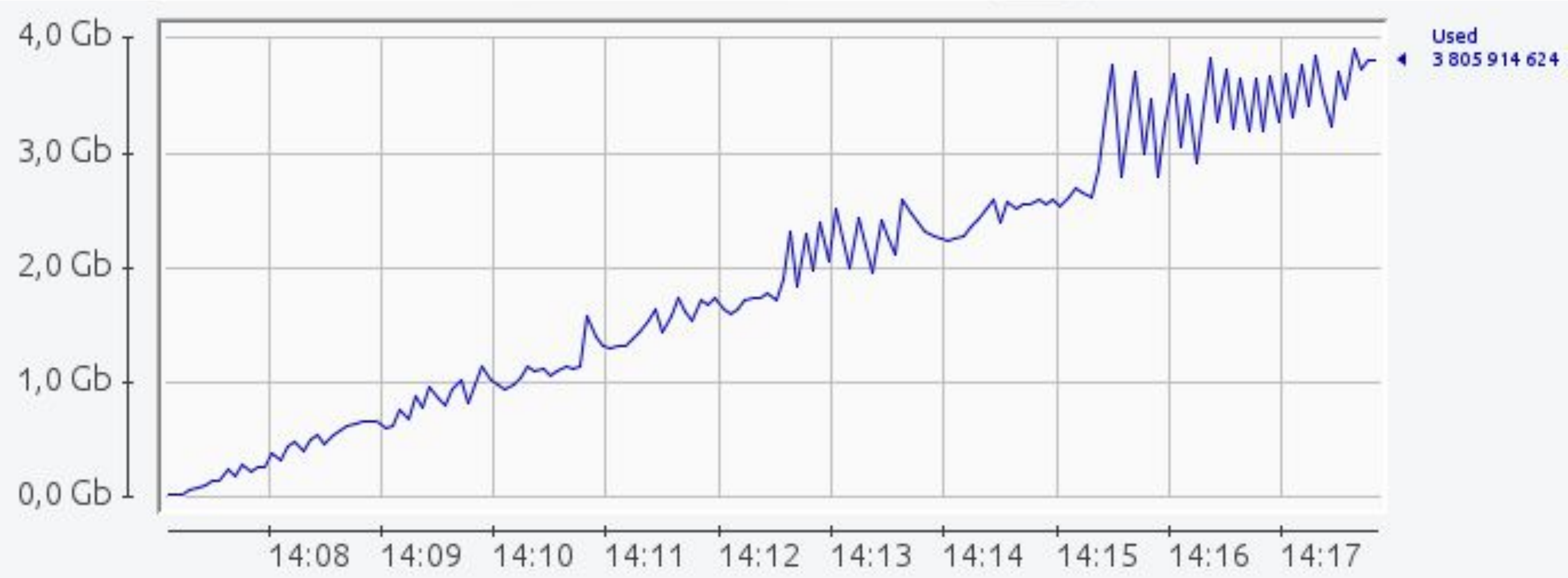
```
"name": "Save by Spring",
"count": 4000000,
"time": "10 min, 28 sec 218 ms"
```



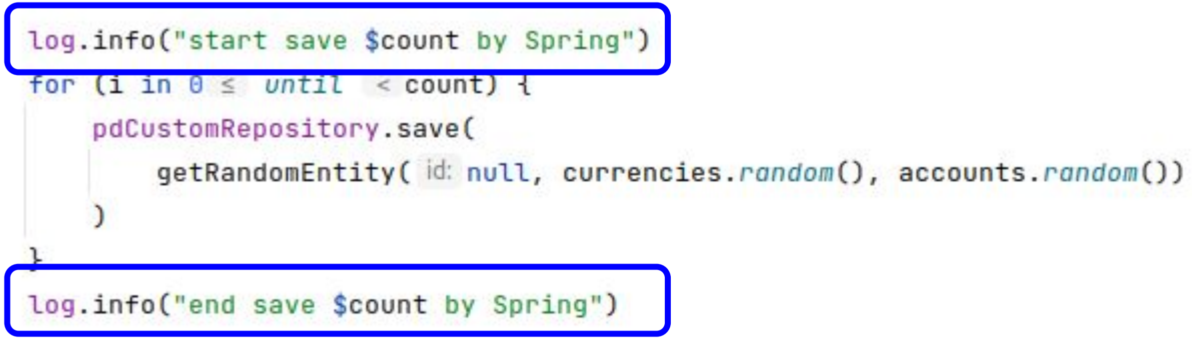
Spring



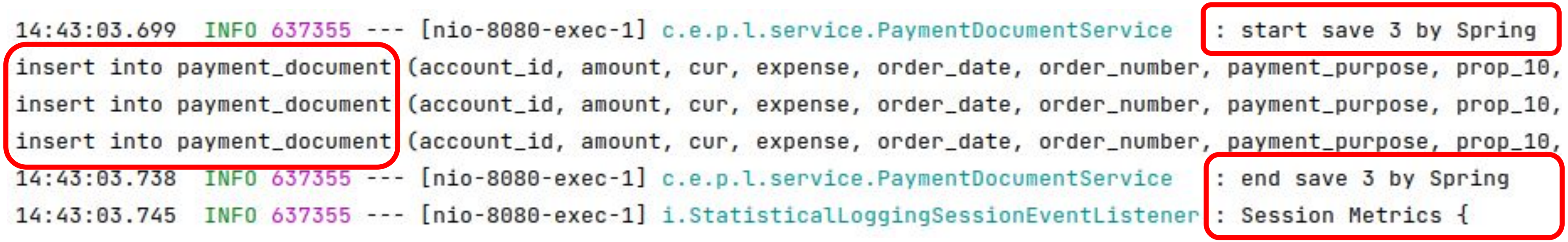
Hibernate caches all the newly inserted `Customer` instances in the session-level cache, so, when the transaction ends, 100 000 entities are managed by the persistence context. If the maximum memory allocated to the JVM is rather low,



Spring

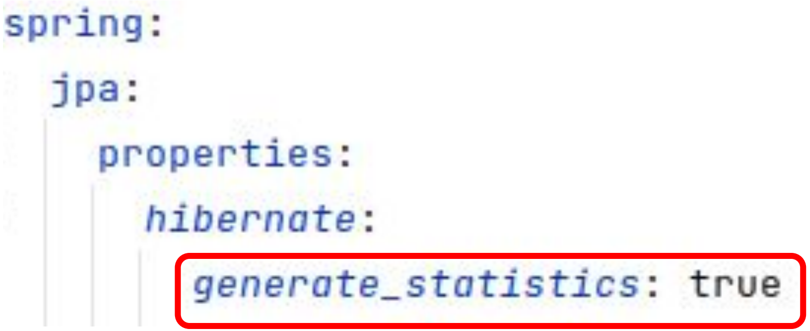


```
log.info("start save $count by Spring")
for (i in 0 ≤ until < count) {
    pdCustomRepository.save(
        getRandomEntity( id: null, currencies.random(), accounts.random())
    )
}
log.info("end save $count by Spring")
```

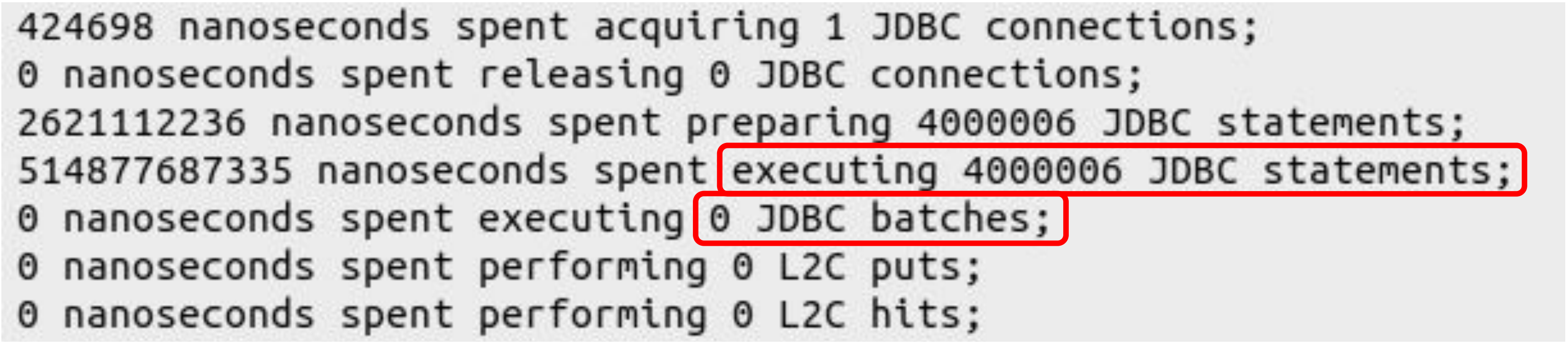


```
14:43:03.699  INFO 637355 --- [nio-8080-exec-1] c.e.p.l.service.PaymentDocumentService   : start save 3 by Spring
insert into payment_document (account_id, amount, cur, expense, order_date, order_number, payment_purpose, prop_10,
insert into payment_document (account_id, amount, cur, expense, order_date, order_number, payment_purpose, prop_10,
insert into payment_document (account_id, amount, cur, expense, order_date, order_number, payment_purpose, prop_10,
14:43:03.738  INFO 637355 --- [nio-8080-exec-1] c.e.p.l.service.PaymentDocumentService   : end save 3 by Spring
14:43:03.745  INFO 637355 --- [nio-8080-exec-1] i.StatisticalLoggingSessionEventListener : Session Metrics {
```

Spring



```yaml
spring:
  jpa:
    properties:
      hibernate:
        generate_statistics: true
```

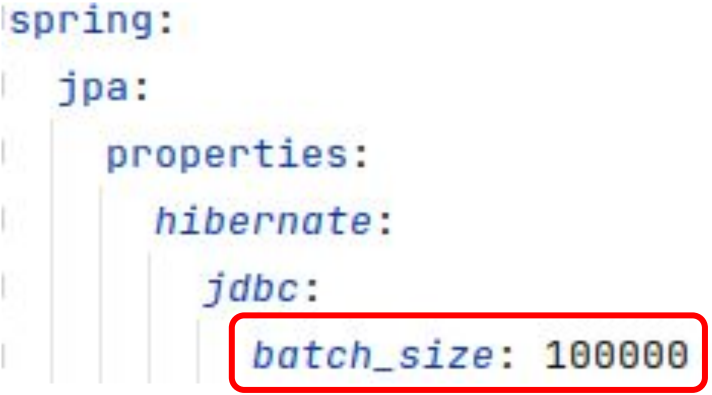


```
424698 nanoseconds spent acquiring 1 JDBC connections;
0 nanoseconds spent releasing 0 JDBC connections;
262112236 nanoseconds spent preparing 4000006 JDBC statements;
51487687335 nanoseconds spent executing 4000006 JDBC statements;
0 nanoseconds spent executing 0 JDBC batches;
0 nanoseconds spent performing 0 L2C puts;
0 nanoseconds spent performing 0 L2C hits;
```

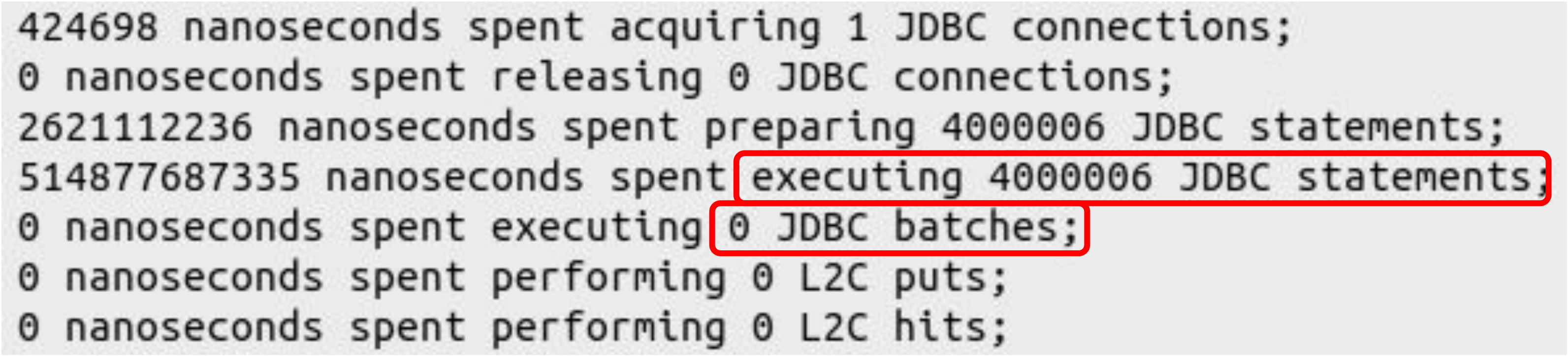
Spring

Как объединить данные в батчи?

Spring

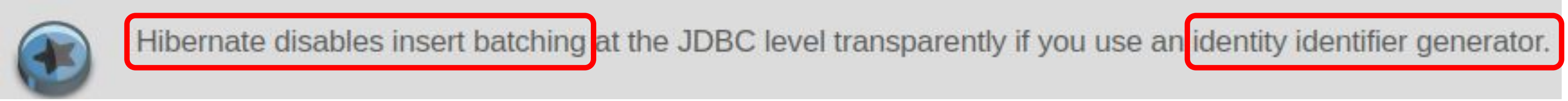


```yaml
spring:
  jpa:
    properties:
      hibernate:
        jdbc:
          batch_size: 100000
```

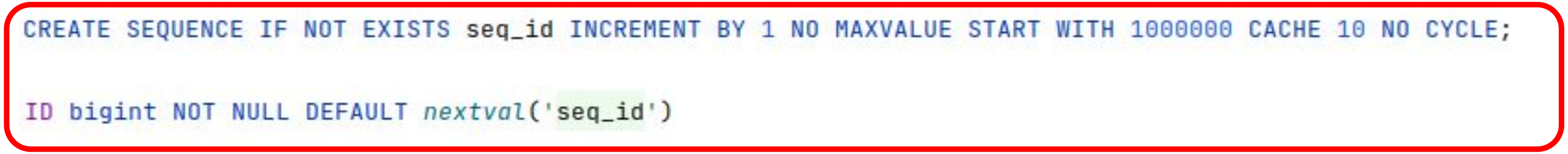


```
424698 nanoseconds spent acquiring 1 JDBC connections;
0 nanoseconds spent releasing 0 JDBC connections;
2621112236 nanoseconds spent preparing 4000006 JDBC statements;
514877687335 nanoseconds spent executing 4000006 JDBC statements;
0 nanoseconds spent executing 0 JDBC batches;
0 nanoseconds spent performing 0 L2C puts;
0 nanoseconds spent performing 0 L2C hits;
```

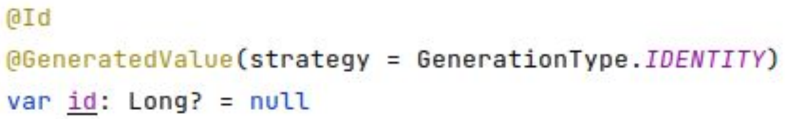
Spring



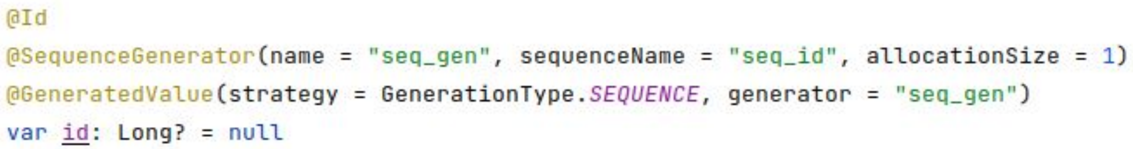
Hibernate disables insert batching at the JDBC level transparently if you use an identity identifier generator.



```
CREATE SEQUENCE IF NOT EXISTS seq_id INCREMENT BY 1 NO MAXVALUE START WITH 1000000 CACHE 10 NO CYCLE;

ID bigint NOT NULL DEFAULT nextval('seq_id')
```

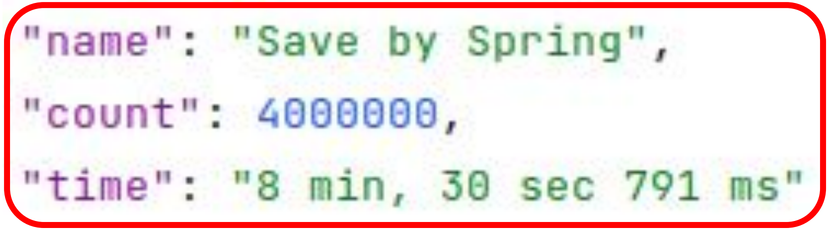


```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
var id: Long? = null
```

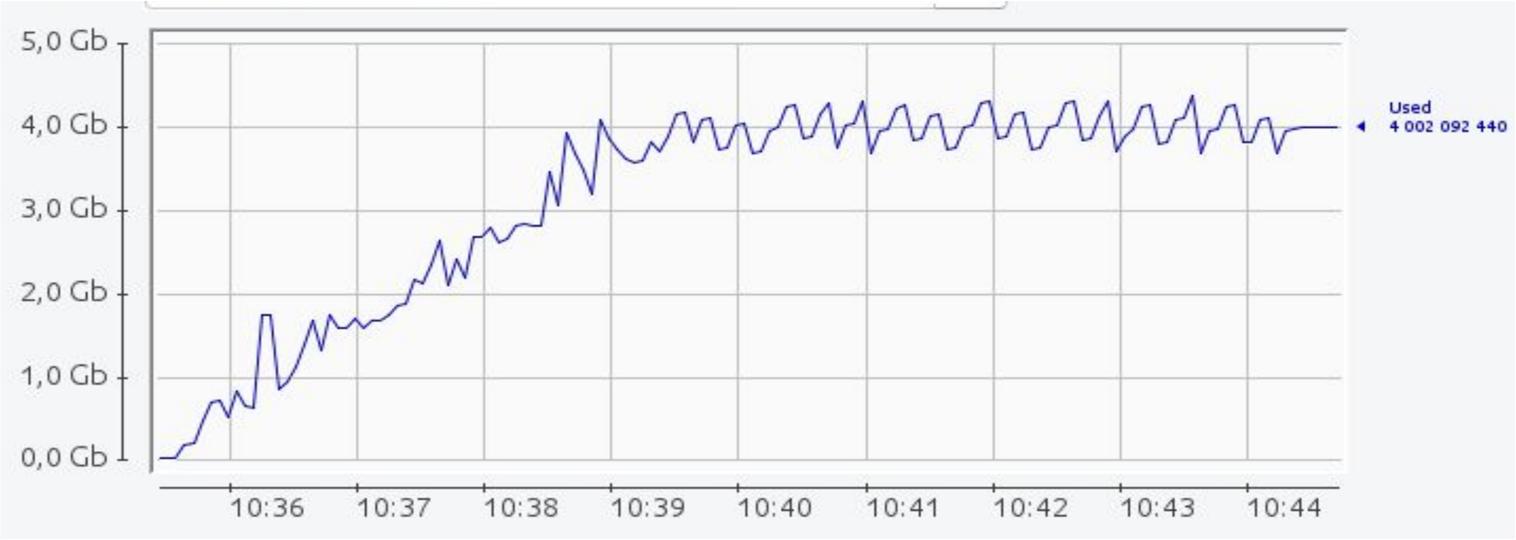


```
@Id
@SequenceGenerator(name = "seq_gen", sequenceName = "seq_id", allocationSize = 1)
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seq_gen")
var id: Long? = null
```

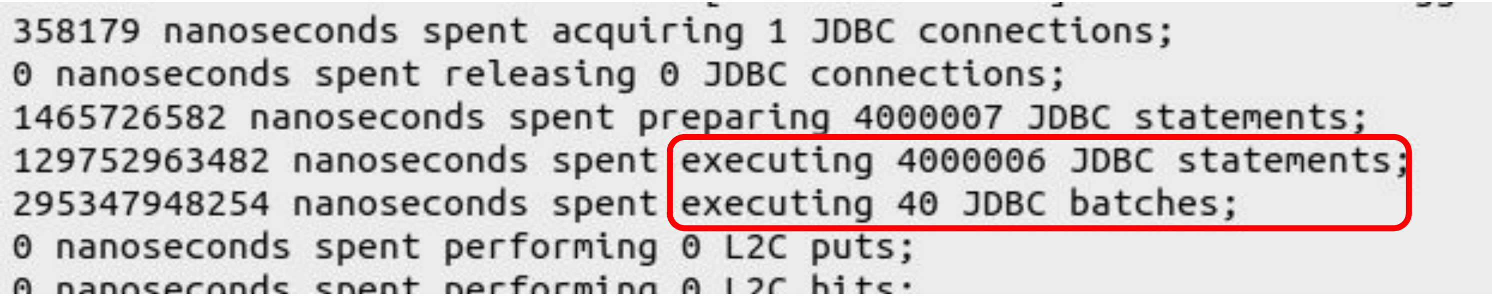
Spring with batch size 100k



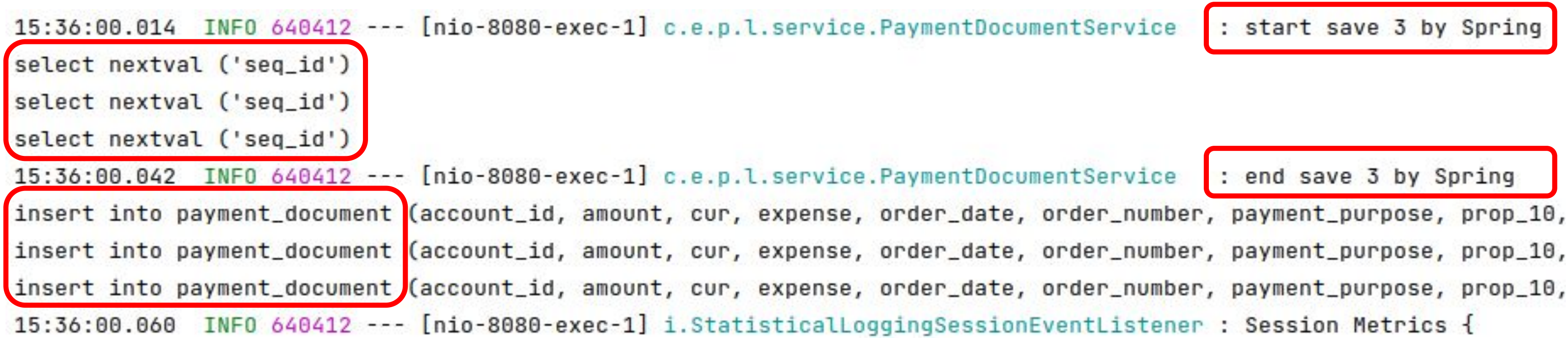
```
"name": "Save by Spring",
"count": 4000000,
"time": "8 min, 30 sec 791 ms"
```



Spring with batch size 100k

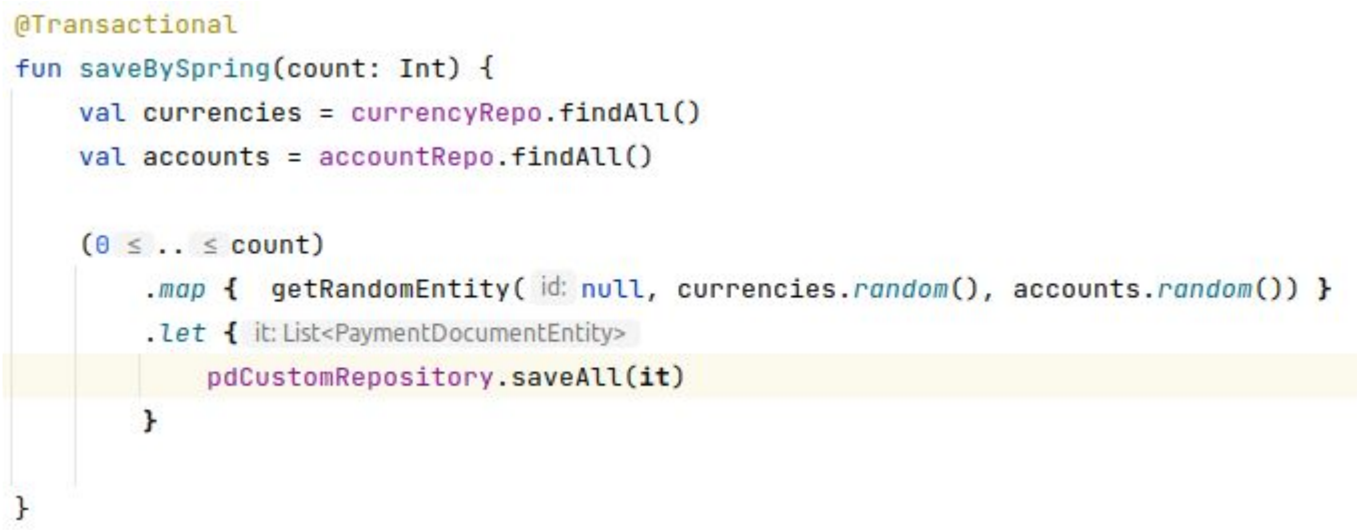


```
358179 nanoseconds spent acquiring 1 JDBC connections;
0 nanoseconds spent releasing 0 JDBC connections;
1465726582 nanoseconds spent preparing 4000007 JDBC statements;
129752963482 nanoseconds spent executing 4000006 JDBC statements;
295347948254 nanoseconds spent executing 40 JDBC batches;
0 nanoseconds spent performing 0 L2C puts;
0 nanoseconds spent performing 0 L2C hits;
```



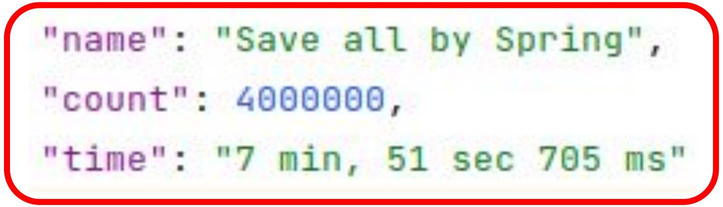
```
15:36:00.014  INFO 640412 --- [nio-8080-exec-1] c.e.p.l.service.PaymentDocumentService   : start save 3 by Spring
select nextval ('seq_id')
select nextval ('seq_id')
select nextval ('seq_id')
15:36:00.042  INFO 640412 --- [nio-8080-exec-1] c.e.p.l.service.PaymentDocumentService   : end save 3 by Spring
insert into payment_document (account_id, amount, cur, expense, order_date, order_number, payment_purpose, prop_10,
insert into payment_document (account_id, amount, cur, expense, order_date, order_number, payment_purpose, prop_10,
insert into payment_document (account_id, amount, cur, expense, order_date, order_number, payment_purpose, prop_10,
15:36:00.060  INFO 640412 --- [nio-8080-exec-1] i.StatisticalLoggingSessionEventListener : Session Metrics {
```

Spring with batch size 100k. Method saveAll()

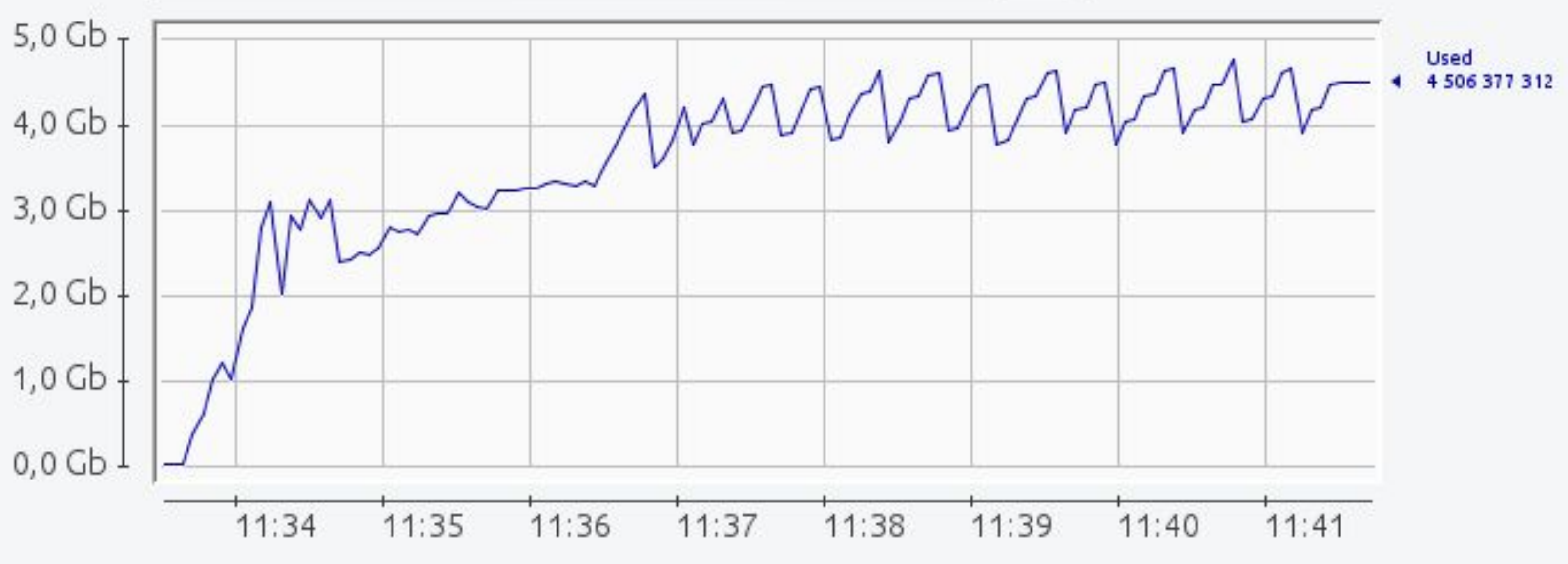


```
@Transactional
fun saveBySpring(count: Int) {
    val currencies = currencyRepo.findAll()
    val accounts = accountRepo.findAll()

    (0 ≤ .. ≤ count)
        .map {  getRandomEntity( id: null, currencies.random(), accounts.random()) }
        .let { it: List<PaymentDocumentEntity>
            pdCustomRepository.saveAll(it)
        }

}
```

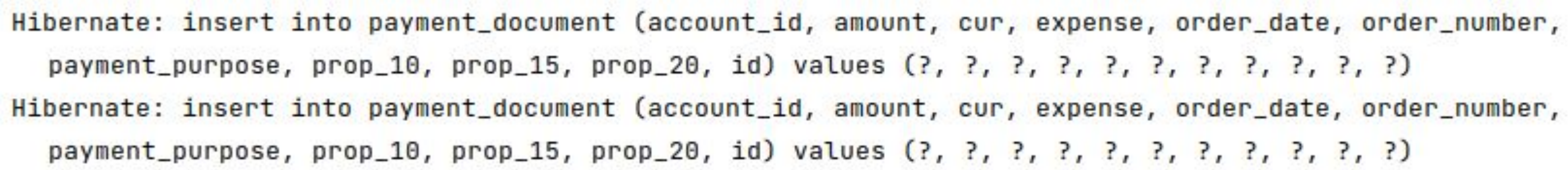
Spring with batch size 100k. Method saveAll()



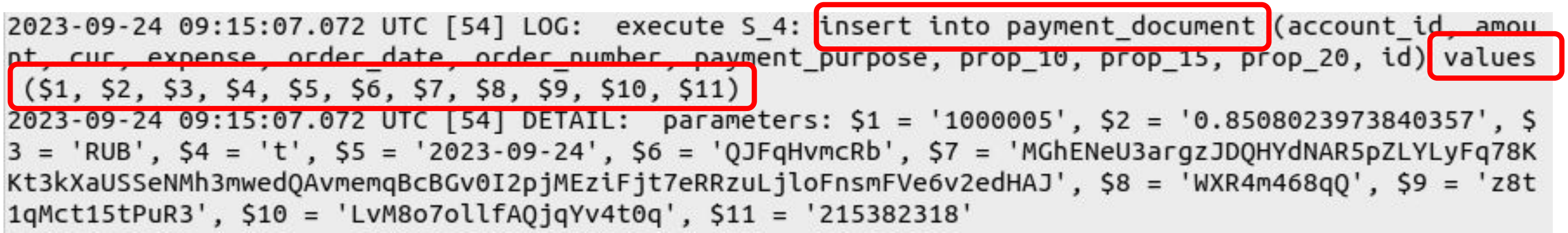
```
"name": "Save all by Spring",
"count": 4000000,
"time": "7 min, 51 sec 705 ms"
```



Spring

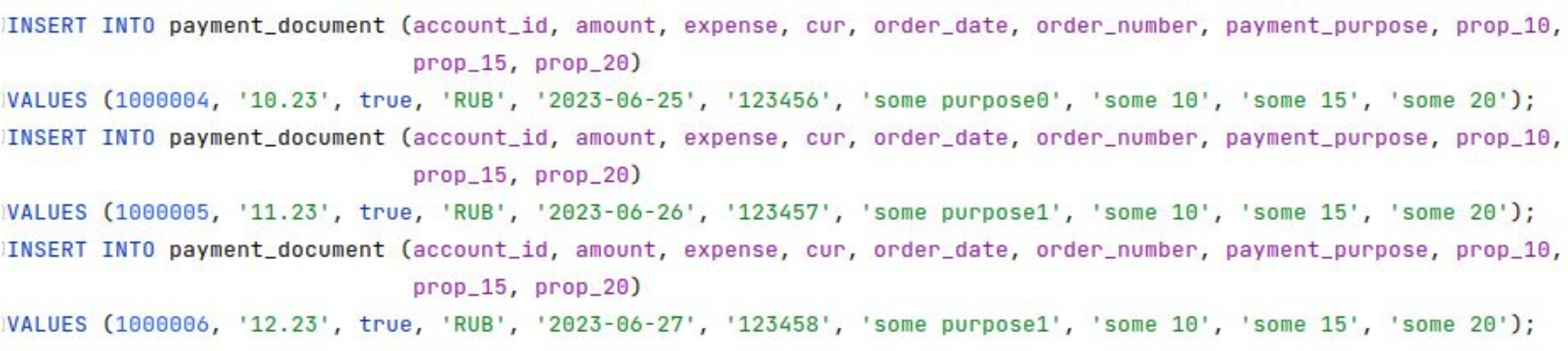


```
Hibernate: insert into payment_document (account_id, amount, cur, expense, order_date, order_number,
  payment_purpose, prop_10, prop_15, prop_20, id) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into payment_document (account_id, amount, cur, expense, order_date, order_number,
  payment_purpose, prop_10, prop_15, prop_20, id) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

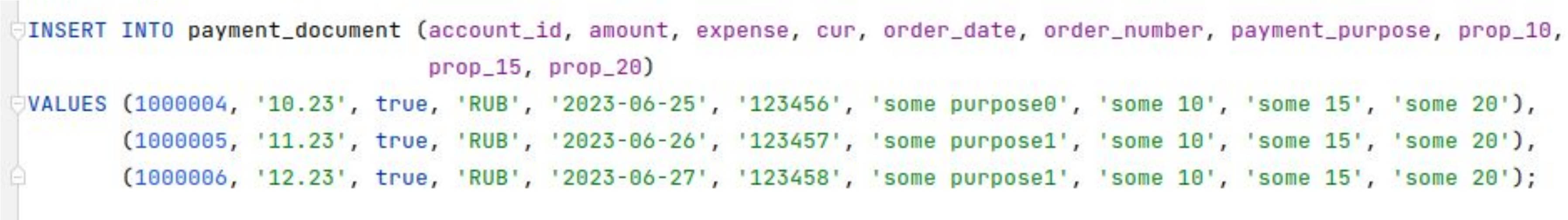


```
2023-09-24 09:15:07.072 UTC [54] LOG:  execute S_4: insert into payment_document (account_id, amou
nt, cur, expense, order_date, order_number, payment_purpose, prop_10, prop_15, prop_20, id) values
 ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11)
2023-09-24 09:15:07.072 UTC [54] DETAIL:  parameters: $1 = '1000005', $2 = '0.850802397384035 7', $
3 = 'RUB', $4 = 't', $5 = '2023-09-24', $6 = 'QJFqHvmcRb', $7 = 'MGhENeU3argzJDQHYdNAR5pZLYLyFq78K
Kt3kXaUSSeNMh3mwedQAvmemqBcBGv0I2pjMEziFjt7eRRzuLjloFnsmFVe6v2edHAJ', $8 = 'WXR4m468qQ', $9 = 'z8t
1qMct15tPuR3', $10 = 'LvM8o7ollfAQjqYv4t0q', $11 = '215382318'
```

Spring

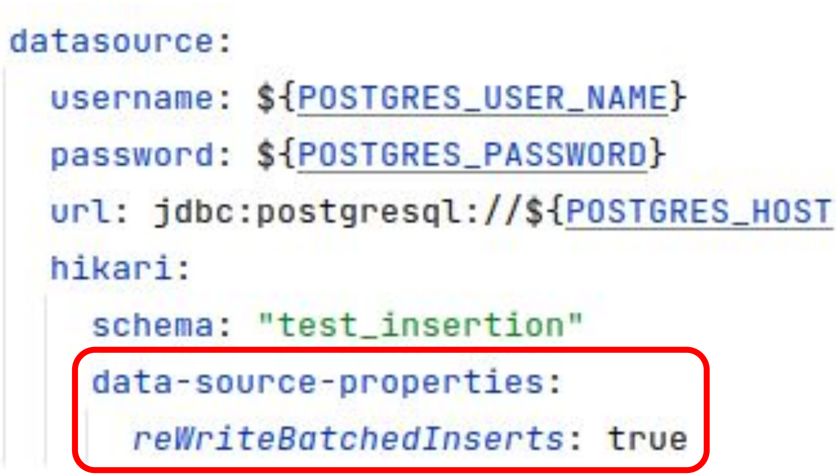


```sql
INSERT INTO payment_document (account_id, amount, expense, cur, order_date, order_number, payment_purpose, prop_10,
                              prop_15, prop_20)
VALUES (1000004, '10.23', true, 'RUB', '2023-06-25', '123456', 'some purpose0', 'some 10', 'some 15', 'some 20');
INSERT INTO payment_document (account_id, amount, expense, cur, order_date, order_number, payment_purpose, prop_10,
                              prop_15, prop_20)
VALUES (1000005, '11.23', true, 'RUB', '2023-06-26', '123457', 'some purpose1', 'some 10', 'some 15', 'some 20');
INSERT INTO payment_document (account_id, amount, expense, cur, order_date, order_number, payment_purpose, prop_10,
                              prop_15, prop_20)
VALUES (1000006, '12.23', true, 'RUB', '2023-06-27', '123458', 'some purpose1', 'some 10', 'some 15', 'some 20');
```

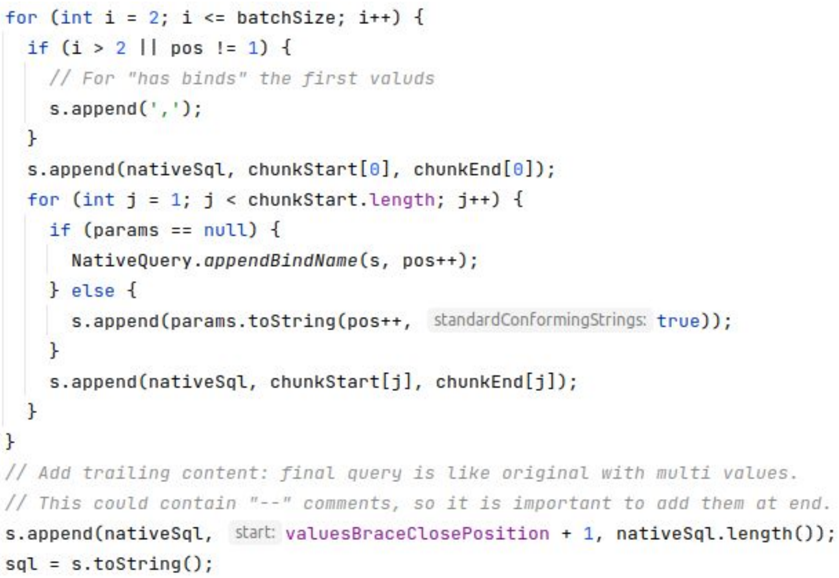


```sql
INSERT INTO payment_document (account_id, amount, expense, cur, order_date, order_number, payment_purpose, prop_10,
                              prop_15, prop_20)
VALUES (1000004, '10.23', true, 'RUB', '2023-06-25', '123456', 'some purpose0', 'some 10', 'some 15', 'some 20'),
       (1000005, '11.23', true, 'RUB', '2023-06-26', '123457', 'some purpose1', 'some 10', 'some 15', 'some 20'),
       (1000006, '12.23', true, 'RUB', '2023-06-27', '123458', 'some purpose1', 'some 10', 'some 15', 'some 20');
```

Spring reWriteBatchedInserts

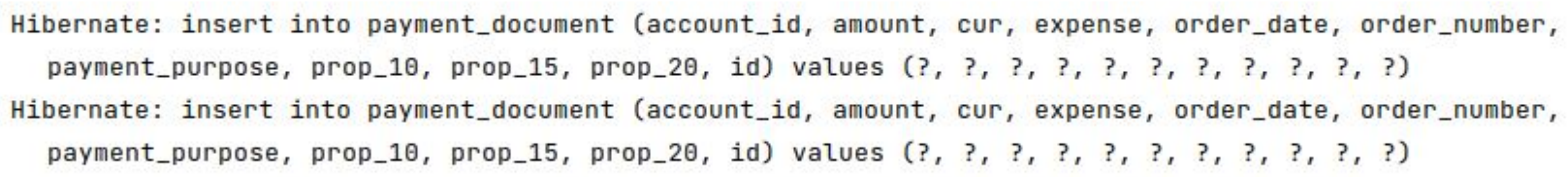


```yaml
datasource:
  username: ${POSTGRES_USER_NAME}
  password: ${POSTGRES_PASSWORD}
  url: jdbc:postgresql://${POSTGRES_HOST
  hikari:
    schema: "test_insertion"
    data-source-properties:
      reWriteBatchedInserts: true
```

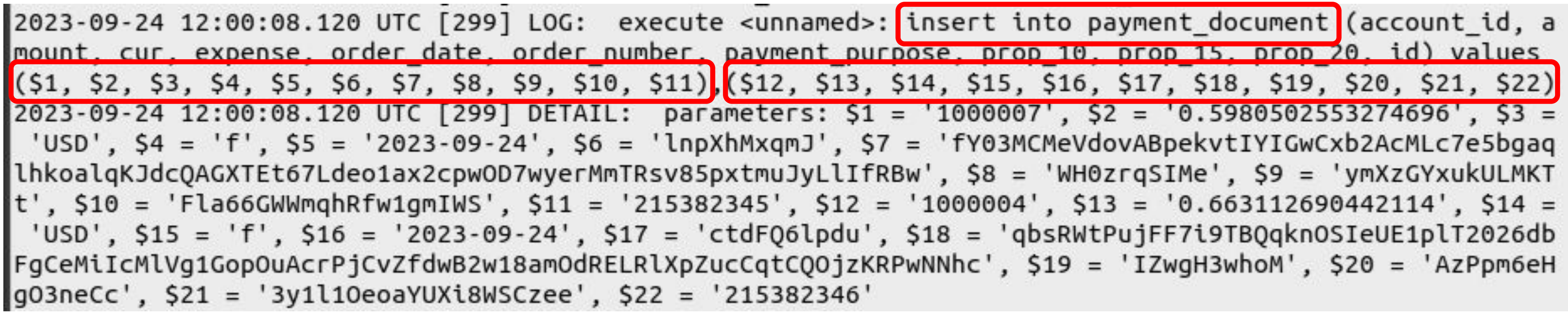


```java
for (int i = 2; i <= batchSize; i++) {
  if (i > 2 || pos != 1) {
    // For "has binds" the first valuds
    s.append(',');
  }
  s.append(nativeSql, chunkStart[0], chunkEnd[0]);
  for (int j = 1; j < chunkStart.length; j++) {
    if (params == null) {
      NativeQuery.appendBindName(s, pos++);
    } else {
      s.append(params.toString(pos++,  standardConformingStrings: true));
    }
    s.append(nativeSql, chunkStart[j], chunkEnd[j]);
  }
}
// Add trailing content: final query is like original with multi values.
// This could contain "--" comments, so it is important to add them at end.
s.append(nativeSql,  start: valuesBraceClosePosition + 1, nativeSql.length());
sql = s.toString();
```

Spring reWriteBatchedInserts

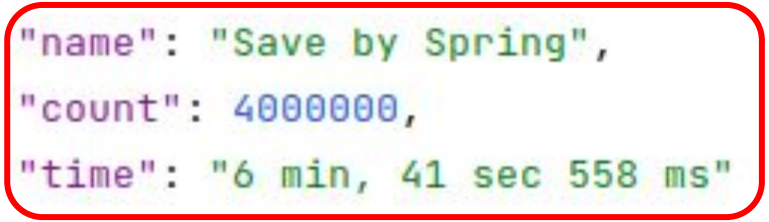


```
Hibernate: insert into payment_document (account_id, amount, cur, expense, order_date, order_number,
   payment_purpose, prop_10, prop_15, prop_20, id) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into payment_document (account_id, amount, cur, expense, order_date, order_number,
   payment_purpose, prop_10, prop_15, prop_20, id) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

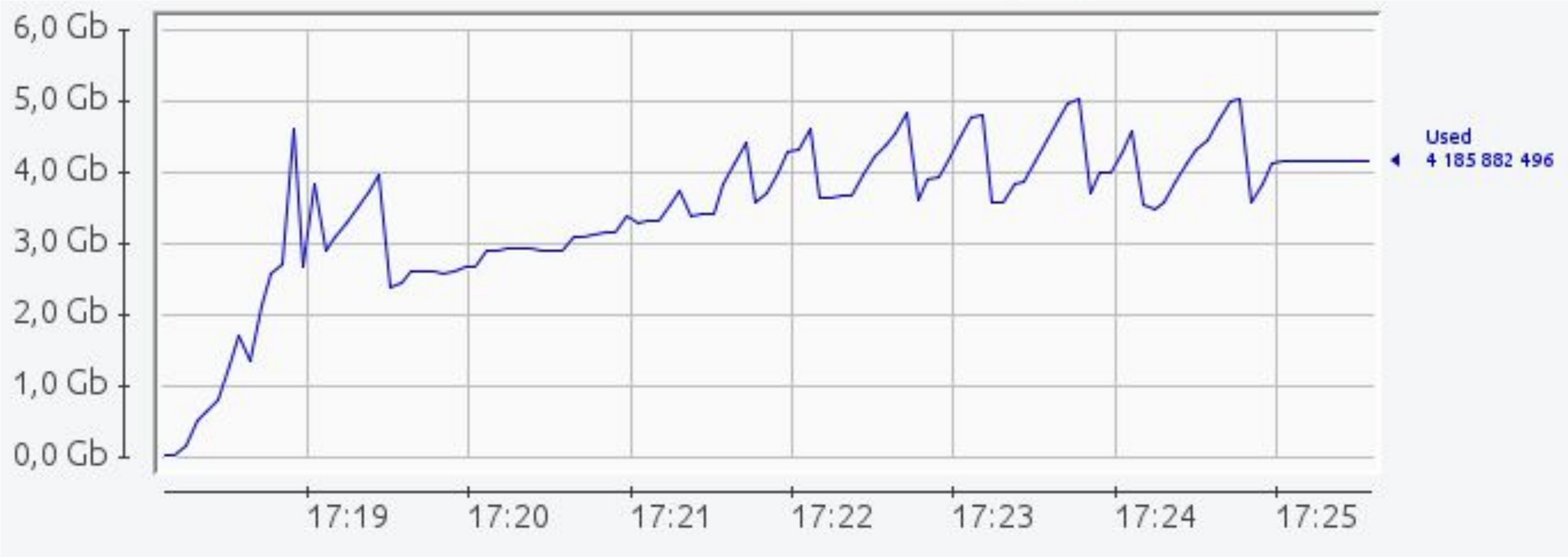


```
2023-09-24 12:00:08.120 UTC [299] LOG:  execute <unnamed>: insert into payment_document (account_id, a
mount, cur, expense, order_date, order_number, payment_purpose, prop_10, prop_15, prop_20, id) values
($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11),($12, $13, $14, $15, $16, $17, $18, $19, $20, $21, $22)
2023-09-24 12:00:08.120 UTC [299] DETAIL:  parameters: $1 = '1000007', $2 = '0.5980502553274696', $3 =
 'USD', $4 = 'f', $5 = '2023-09-24', $6 = 'lnpXhMxqmJ', $7 = 'fY03MCMeVdovABpekvtIYIGwCxb2AcMLc7e5bgaq
lhkoalqKJdcQAGXTEt67Ldeo1ax2cpwOD7wyerMmTRsv85pxtmuJyLlIfRBw', $8 = 'WH0zrqSIMe', $9 = 'ymXzGYxukULMKT
t', $10 = 'Fla66GWWmqhRfw1gmIWS', $11 = '215382345', $12 = '1000004', $13 = '0.663112690442114', $14 =
 'USD', $15 = 'f', $16 = '2023-09-24', $17 = 'ctdFQ6lpdu', $18 = 'qbsRWtPujFF7i9TBQqknOSIeUE1plT2026db
FgCeMiIcMlVg1GopOuAcrPjCvZfdwB2w18amOdRELRlXpZucCqtCQOjzKRPwNNhc', $19 = 'IZwgH3whoM', $20 = 'AzPpm6eH
gO3neCc', $21 = '3y1l1OeoaYUXi8WSCzee', $22 = '215382346'
```

Spring reWriteBatchedInserts



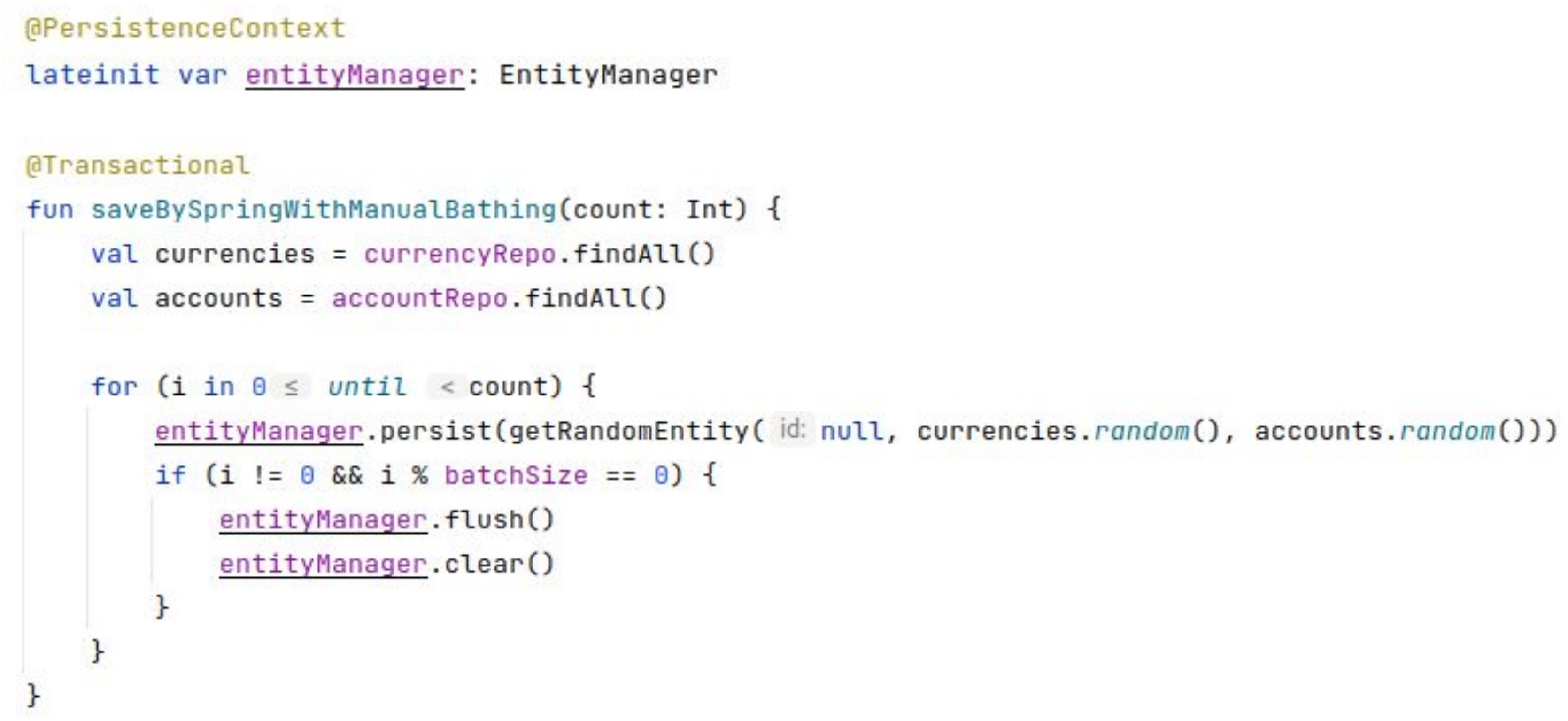
```
"name": "Save by Spring",
"count": 4000000,
"time": "6 min, 41 sec 558 ms"
```



Spring

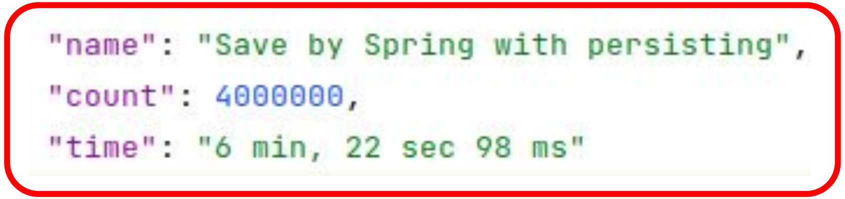
Как сократить объем памяти?

Spring with manual persisting

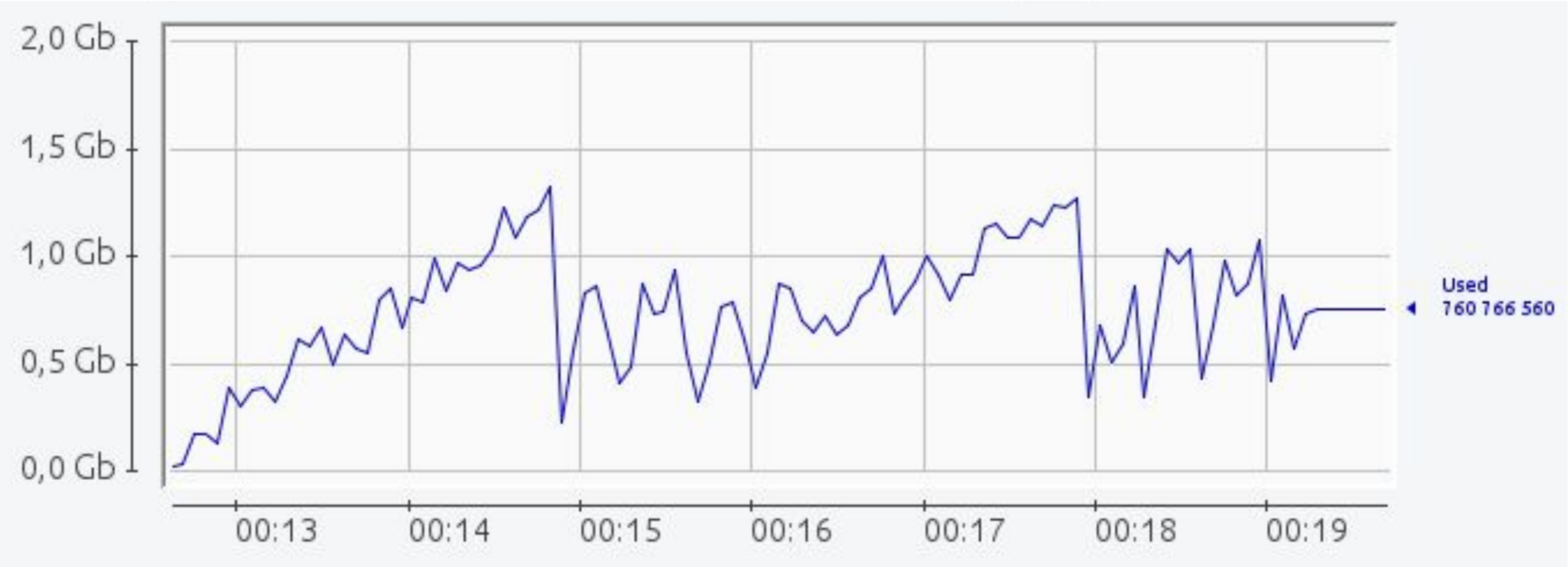


```
@PersistenceContext
lateinit var entityManager: EntityManager

@Transactional
fun saveBySpringWithManualBathing(count: Int) {
    val currencies = currencyRepo.findAll()
    val accounts = accountRepo.findAll()

    for (i in 0 ≤ until < count) {
        entityManager.persist(getRandomEntity( id: null, currencies.random(), accounts.random()))
        if (i != 0 && i % batchSize == 0) {
            entityManager.flush()
            entityManager.clear()
        }
    }
}
```

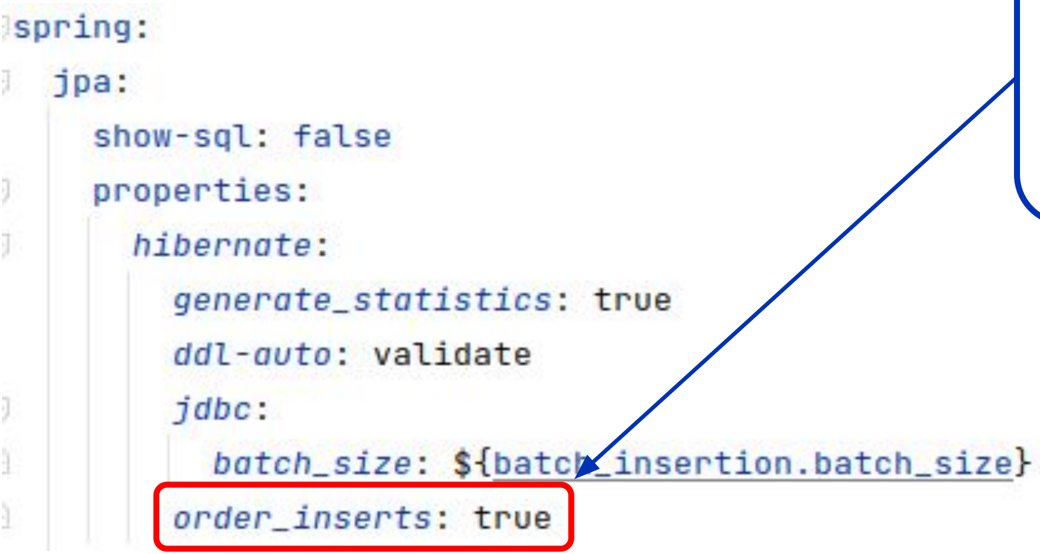
Spring with manual persisting and reWriteBatchedInserts



```
"name": "Save by Spring with persisting",
"count": 4000000,
"time": "6 min, 22 sec 98 ms"
```



Spring



```yaml
spring:
  jpa:
    show-sql: false
    properties:
      hibernate:
        generate_statistics: true
        ddl-auto: validate
        jdbc:
          batch_size: ${batch_insertion.batch_size}
        order_inserts: true
```

Устанавливает порядок при сохранении нескольких типов сущностей в одной транзакции. Позволяет батчингу работать стабильно.

Spring. Итого:

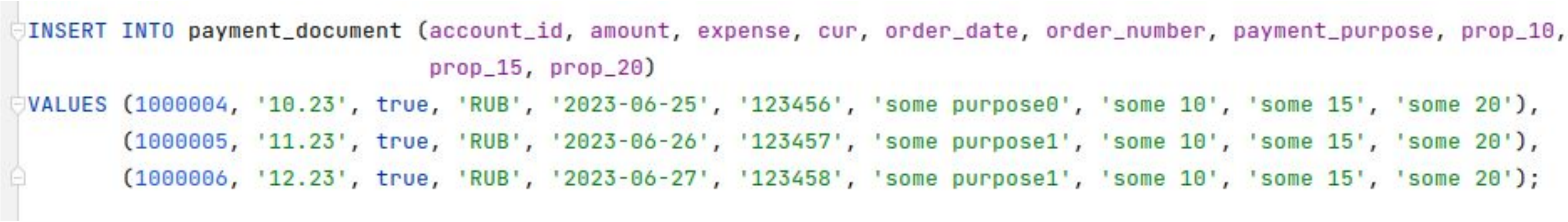
- Настройка batch_size позволяет отправлять данные батчам, не работает с GenerationType.IDENTITY
- Hibernate хранит все созданные данные в session-level cache до окончания транзакции. Можно очищать через EntityManager
- Настройка order_inserts позволяет батчингу стабильно работать.
- Удалось увеличить скорость вставки примерно на 40%

Spring

Может ну его, этот Hibernate?

Spring

Может ну его, этот hibernate?

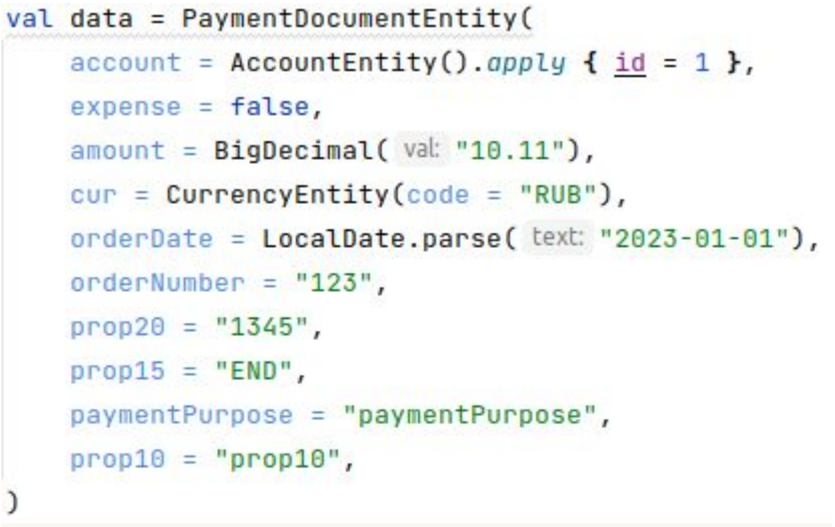


```
INSERT INTO payment_document (account_id, amount, expense, cur, order_date, order_number, payment_purpose, prop_10,
                              prop_15, prop_20)
VALUES (1000004, '10.23', true, 'RUB', '2023-06-25', '123456', 'some purpose0', 'some 10', 'some 15', 'some 20'),
       (1000005, '11.23', true, 'RUB', '2023-06-26', '123457', 'some purpose1', 'some 10', 'some 15', 'some 20'),
       (1000006, '12.23', true, 'RUB', '2023-06-27', '123458', 'some purpose1', 'some 10', 'some 15', 'some 20');
```

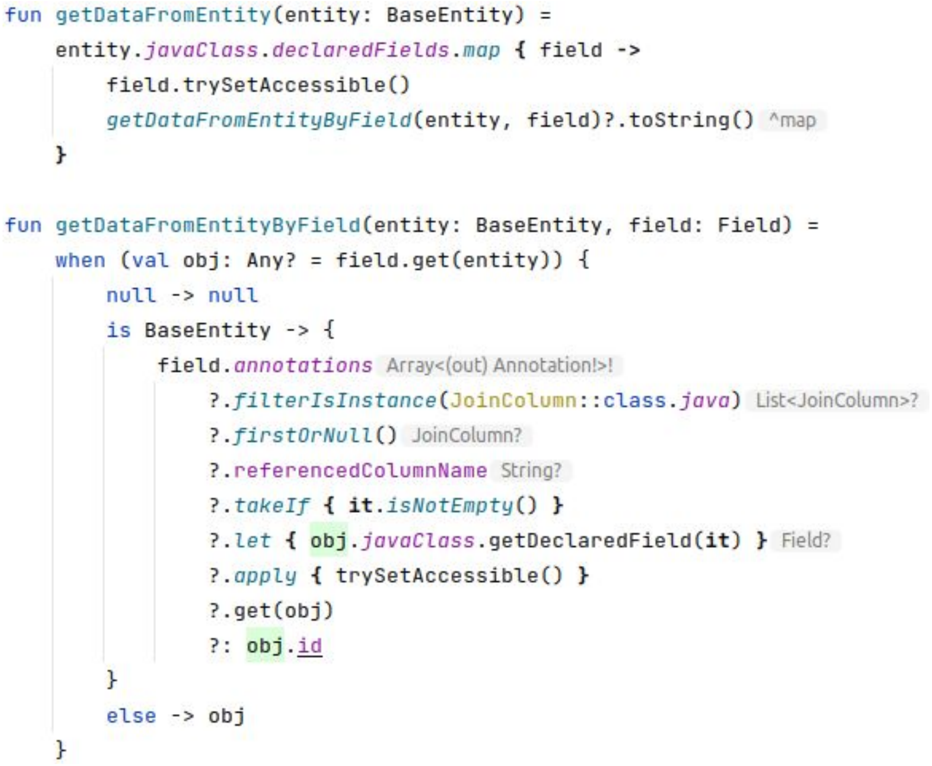
Reflection vs jpa-modelgen vs KProperty

Как?

Reflection vs jpa-modelgen vs KProperty



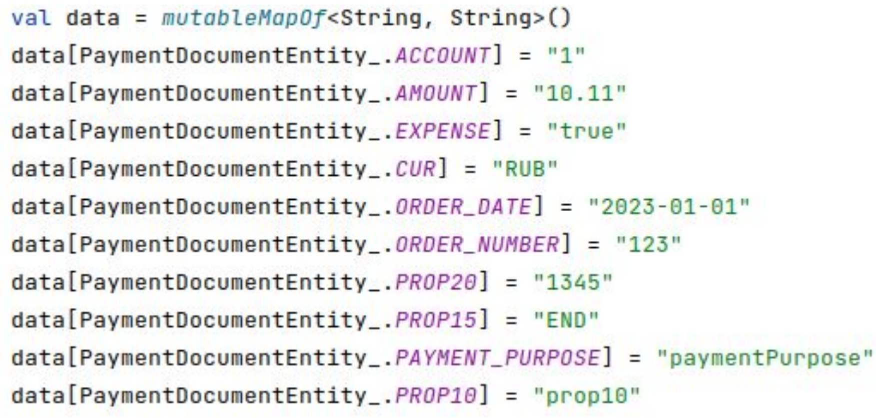
```
val data = PaymentDocumentEntity(
    account = AccountEntity().apply { id = 1 },
    expense = false,
    amount = BigDecimal( val: "10.11"),
    cur = CurrencyEntity(code = "RUB"),
    orderDate = LocalDate.parse( text: "2023-01-01"),
    orderNumber = "123",
    prop20 = "1345",
    prop15 = "END",
    paymentPurpose = "paymentPurpose",
    prop10 = "prop10",
)
```



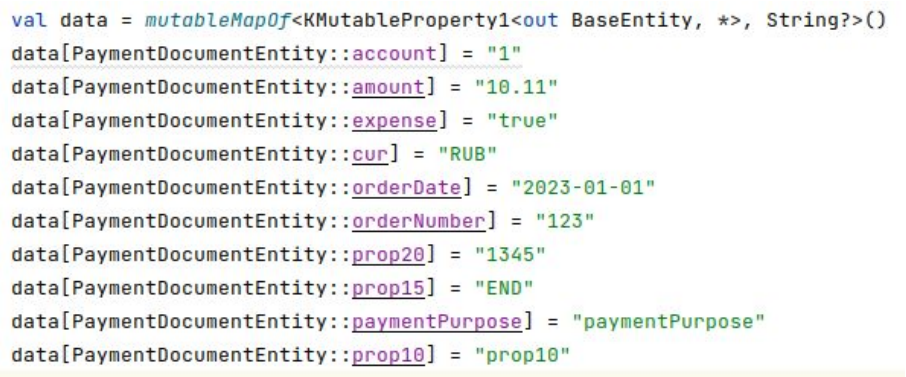
```
fun getDataFromEntity(entity: BaseEntity) =
    entity.javaClass.declaredFields.map { field ->
        field.trySetAccessible()
        getDataFromEntityByField(entity, field)?.toString() ^map
    }

fun getDataFromEntityByField(entity: BaseEntity, field: Field) =
    when (val obj: Any? = field.get(entity)) {
        null -> null
        is BaseEntity -> {
            field.annotations Array<(out) Annotation!>!
                ?.filterIsInstance(JoinColumn::class.java) List<JoinColumn>?
                ?.firstOrNull() JoinColumn?
                ?.referencedColumnName String?
                ?.takeIf { it.isNotEmpty() }
                ?.let { obj.javaClass.getDeclaredField(it) } Field?
                ?.apply { trySetAccessible() }
                ?.get(obj)
                ?: obj.id
        }
        else -> obj
    }
```

Reflection vs jpa-modelgen vs KProperty

hibernate-jpamodelgen vs KProperty Kotlin



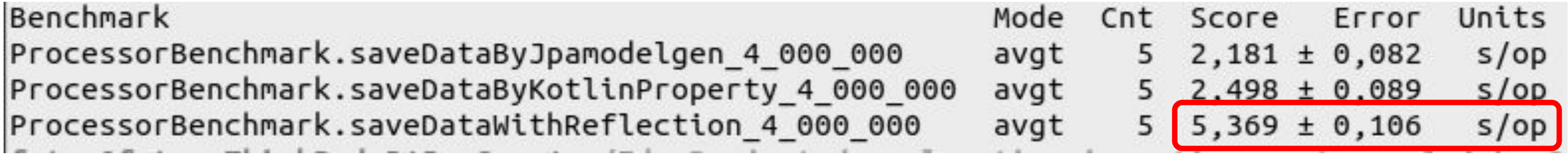
```
val data = mutableMapOf<String, String>()
data[PaymentDocumentEntity_.ACCOUNT] = "1"
data[PaymentDocumentEntity_.AMOUNT] = "10.11"
data[PaymentDocumentEntity_.EXPENSE] = "true"
data[PaymentDocumentEntity_.CUR] = "RUB"
data[PaymentDocumentEntity_.ORDER_DATE] = "2023-01-01"
data[PaymentDocumentEntity_.ORDER_NUMBER] = "123"
data[PaymentDocumentEntity_.PROP20] = "1345"
data[PaymentDocumentEntity_.PROP15] = "END"
data[PaymentDocumentEntity_.PAYMENT_PURPOSE] = "paymentPurpose"
data[PaymentDocumentEntity_.PROP10] = "prop10"
```



```
val data = mutableMapOf<KMutableProperty1<out BaseEntity, *>, String?>()
data[PaymentDocumentEntity::account] = "1"
data[PaymentDocumentEntity::amount] = "10.11"
data[PaymentDocumentEntity::expense] = "true"
data[PaymentDocumentEntity::cur] = "RUB"
data[PaymentDocumentEntity::orderDate] = "2023-01-01"
data[PaymentDocumentEntity::orderNumber] = "123"
data[PaymentDocumentEntity::prop20] = "1345"
data[PaymentDocumentEntity::prop15] = "END"
data[PaymentDocumentEntity::paymentPurpose] = "paymentPurpose"
data[PaymentDocumentEntity::prop10] = "prop10"
```

https://hibernate.org/orm/tooling/

Reflection vs jpa-modelgen vs KProperty

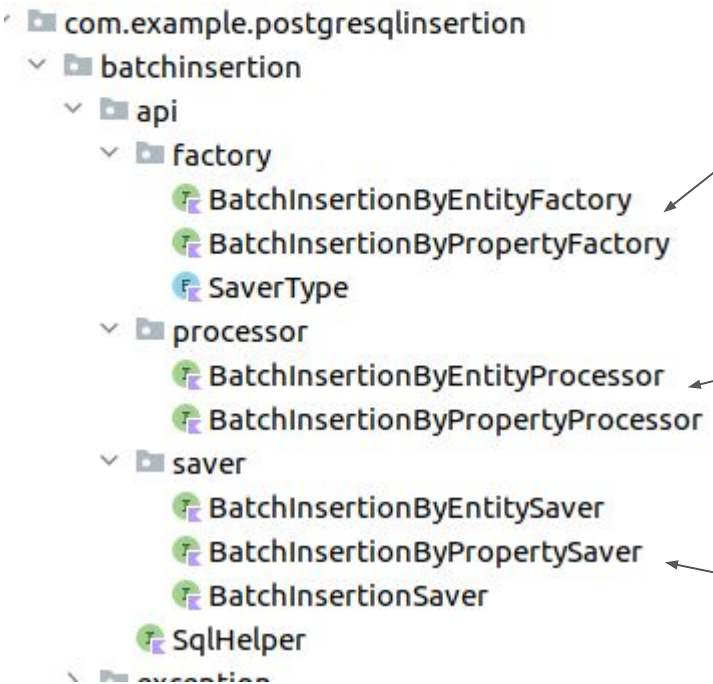


```
Benchmark                                          Mode  Cnt  Score   Error  Units
ProcessorBenchmark.saveDataByJpamodelgen_4_000_000  avgt    5  2,181 ± 0,082   s/op
ProcessorBenchmark.saveDataByKotlinProperty_4_000_000  avgt    5  2,498 ± 0,089   s/op
ProcessorBenchmark.saveDataWithReflection_4_000_000  avgt    5  5,369 ± 0,106   s/op
```

Application

Как это работает?

Application



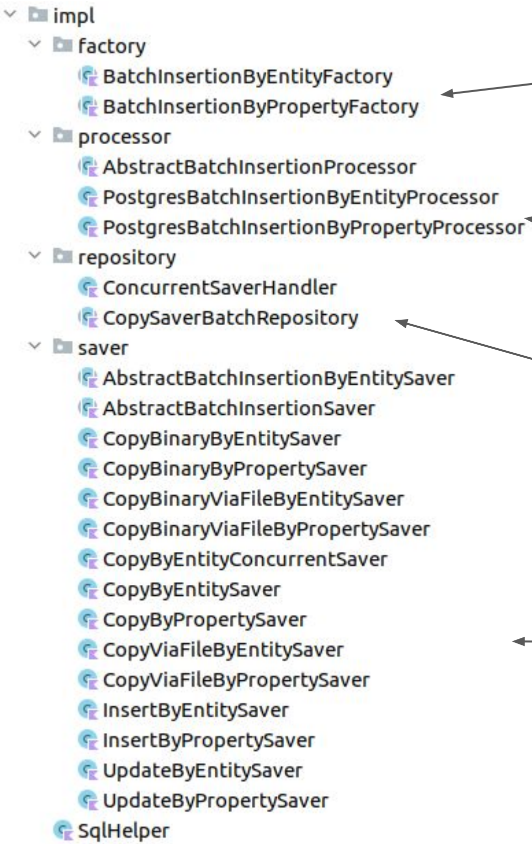
```
com.example.postgresqlinsertion
  batchinsertion
    api
      factory
        BatchInsertionByEntityFactory
        BatchInsertionByPropertyFactory
        SaverType
      processor
        BatchInsertionByEntityProcessor
        BatchInsertionByPropertyProcessor
      saver
        BatchInsertionByEntitySaver
        BatchInsertionByPropertySaver
        BatchInsertionSaver
      SqlHelper
    exception
```

Получает нужный saver по типу

Логика преобразования данных для сохранения в БД

Управление процессом сохранения

Application



```
impl
  factory
    BatchInsertionByEntityFactory
    BatchInsertionByPropertyFactory
  processor
    AbstractBatchInsertionProcessor
    PostgresBatchInsertionByEntityProcessor
    PostgresBatchInsertionByPropertyProcessor
  repository
    ConcurrentSaverHandler
    CopySaverBatchRepository
  saver
    AbstractBatchInsertionByEntitySaver
    AbstractBatchInsertionSaver
    CopyBinaryByEntitySaver
    CopyBinaryByPropertySaver
    CopyBinaryViaFileByEntitySaver
    CopyBinaryViaFileByPropertySaver
    CopyByEntityConcurrentSaver
    CopyByEntitySaver
    CopyByPropertySaver
    CopyViaFileByEntitySaver
    CopyViaFileByPropertySaver
    InsertByEntitySaver
    InsertByPropertySaver
    UpdateByEntitySaver
    UpdateByPropertySaver
  SqlHelper
```

Заготовки для создания класса компонента.

Используется в saver для сохранения данных.

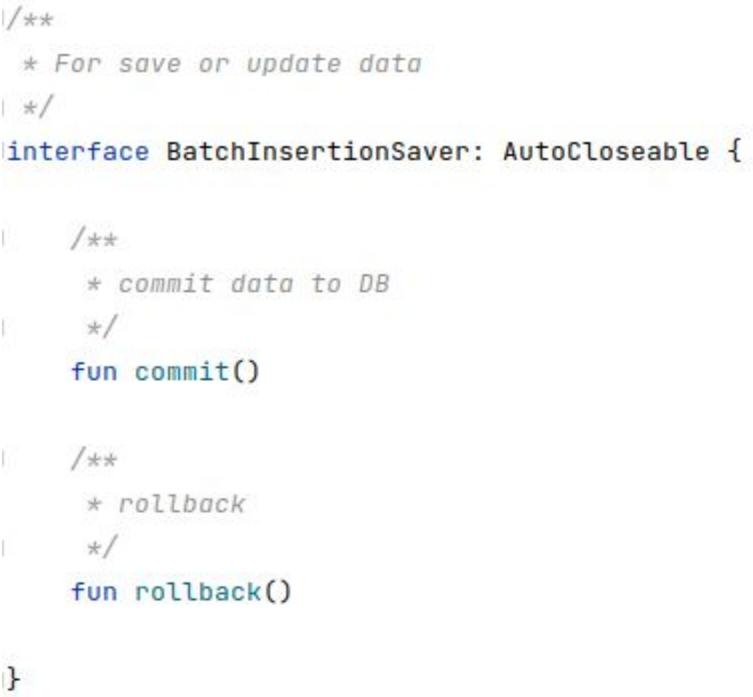
Репозиторий для интеграции со Spring

Реализации saver.

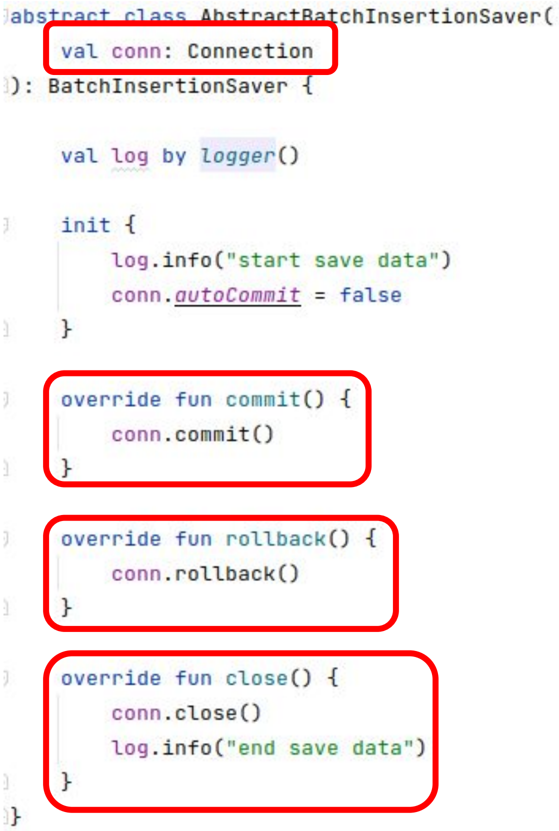
Application

Что внутри?

Application

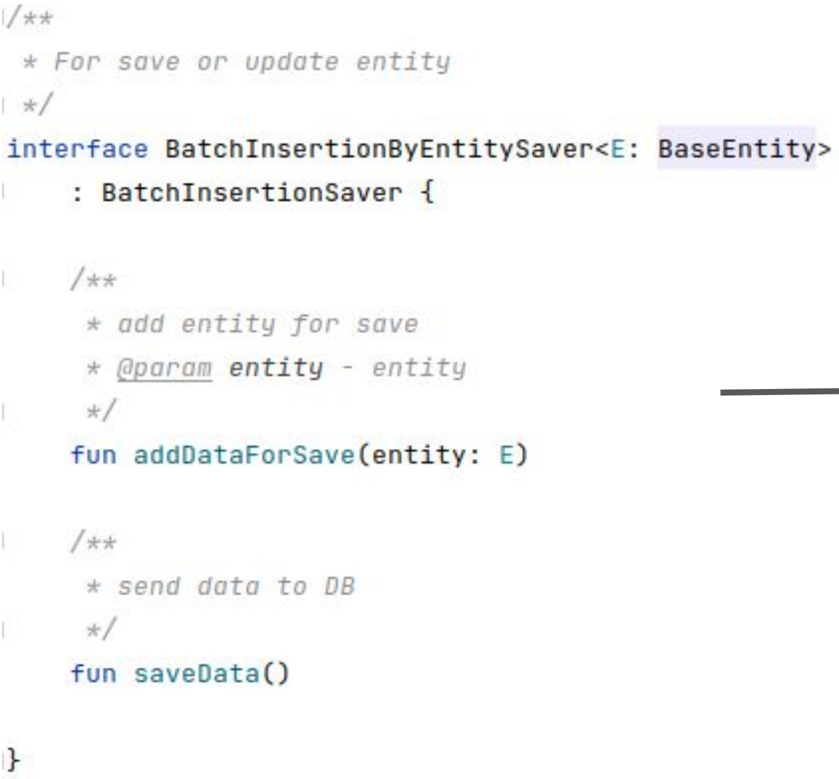


```
/**
 * For save or update data
 */
interface BatchInsertionSaver: AutoCloseable {

    /**
     * commit data to DB
     */
    fun commit()

    /**
     * rollback
     */
    fun rollback()

}
```

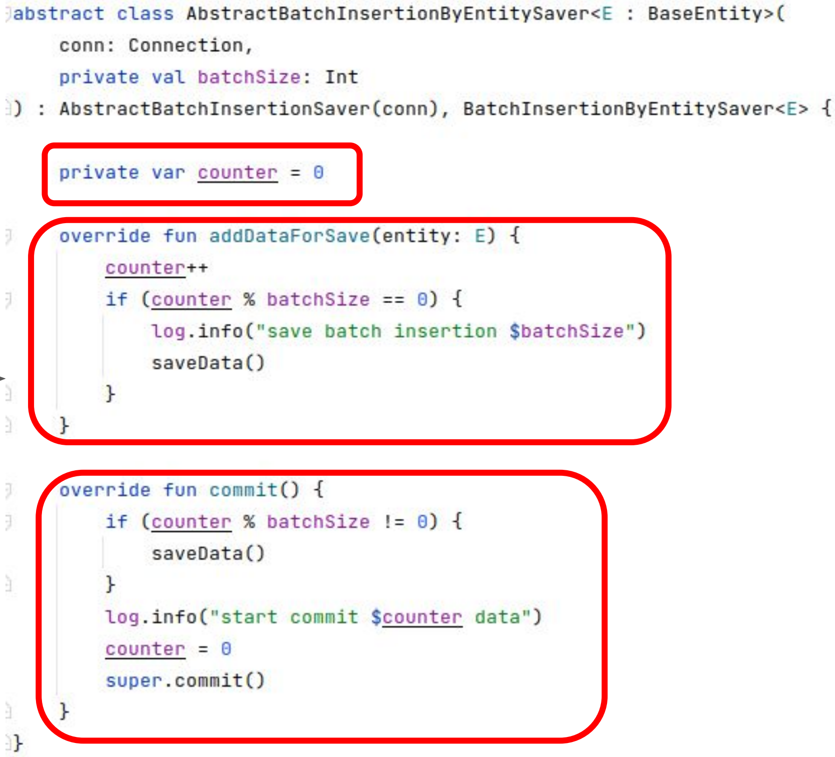


```
abstract class AbstractBatchInsertionSaver(
    val conn: Connection
): BatchInsertionSaver {

    val log by logger()

    init {
        log.info("start save data")
        conn.autoCommit = false
    }

    override fun commit() {
        conn.commit()
    }

    override fun rollback() {
        conn.rollback()
    }

    override fun close() {
        conn.close()
        log.info("end save data")
    }
}
```

Application

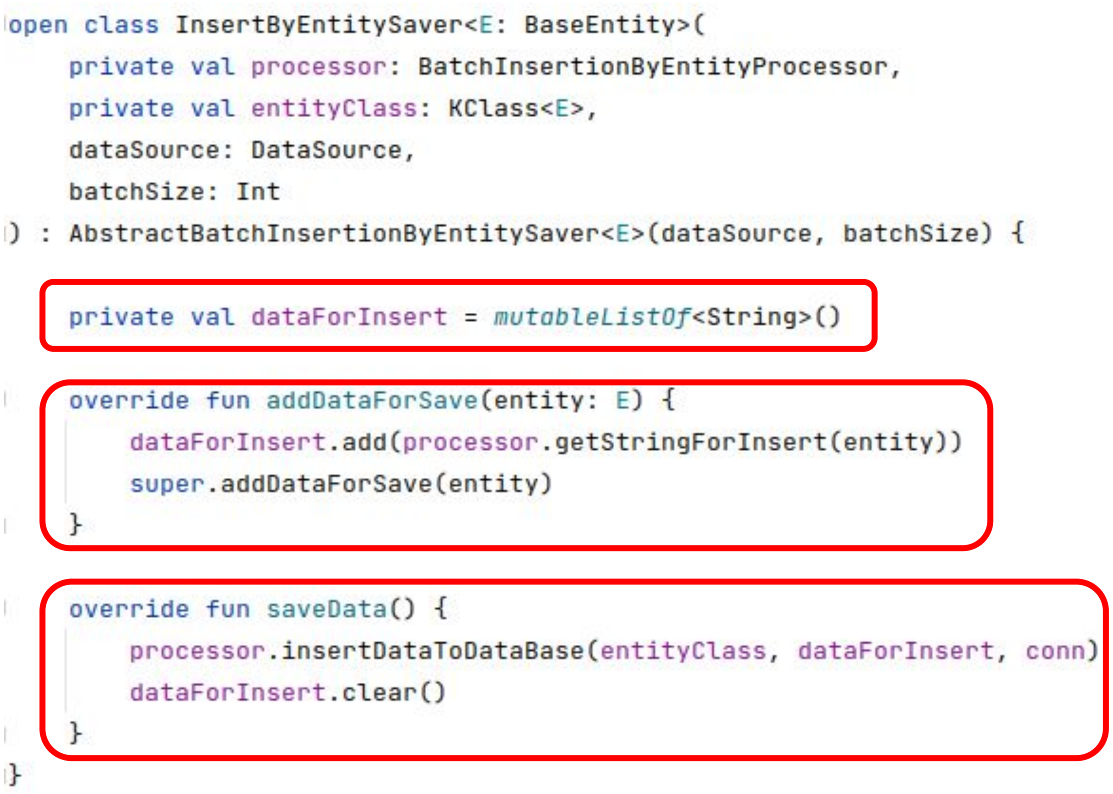


```
/**
 * For save or update entity
 */
interface BatchInsertionByEntitySaver<E: BaseEntity>
    : BatchInsertionSaver {

    /**
     * add entity for save
     * @param entity - entity
     */
    fun addDataForSave(entity: E)

    /**
     * send data to DB
     */
    fun saveData()

}
```



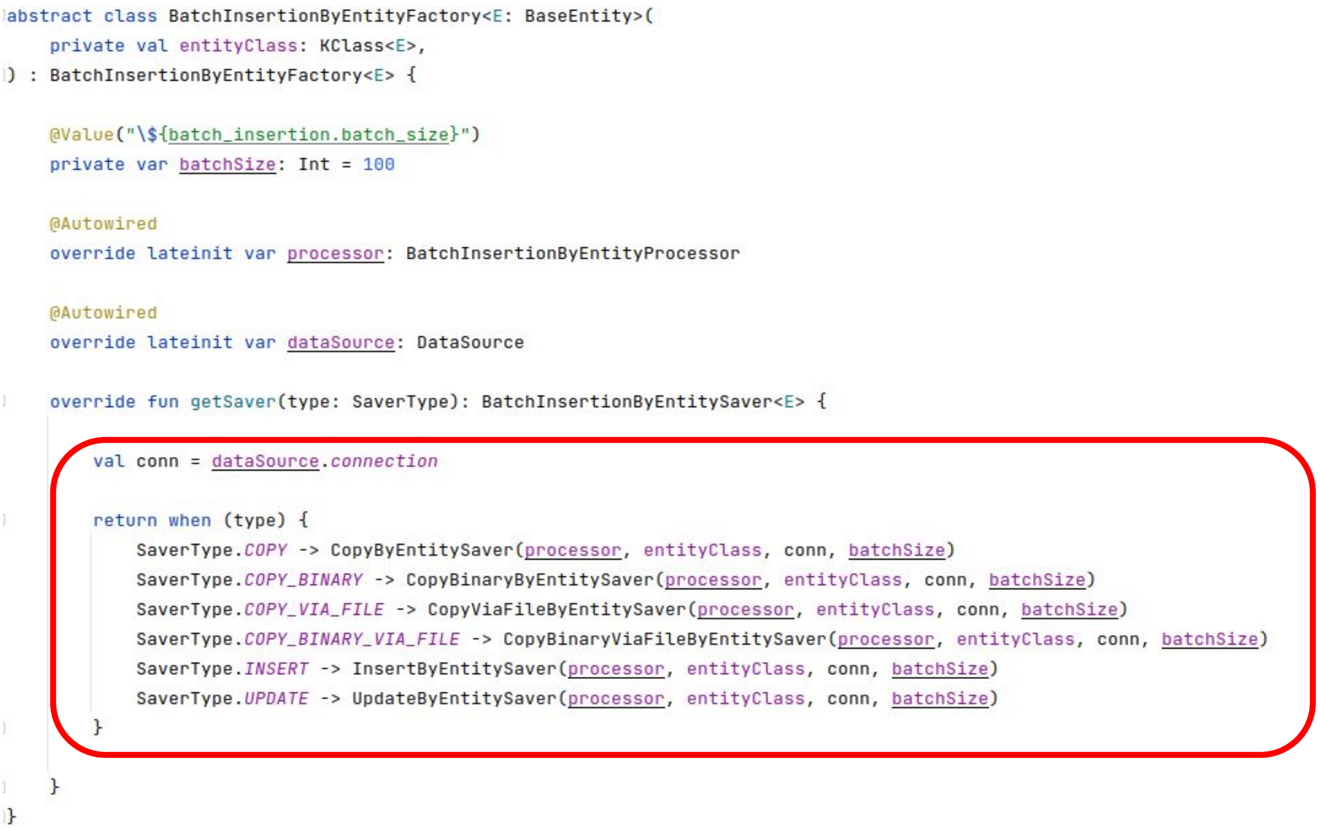
```
abstract class AbstractBatchInsertionByEntitySaver<E : BaseEntity>(
    conn: Connection,
    private val batchSize: Int
) : AbstractBatchInsertionSaver(conn), BatchInsertionByEntitySaver<E> {

    private var counter = 0

    override fun addDataForSave(entity: E) {
        counter++
        if (counter % batchSize == 0) {
            log.info("save batch insertion $batchSize")
            saveData()
        }
    }

    override fun commit() {
        if (counter % batchSize != 0) {
            saveData()
        }
        log.info("start commit $counter data")
        counter = 0
        super.commit()
    }
}
```

Application



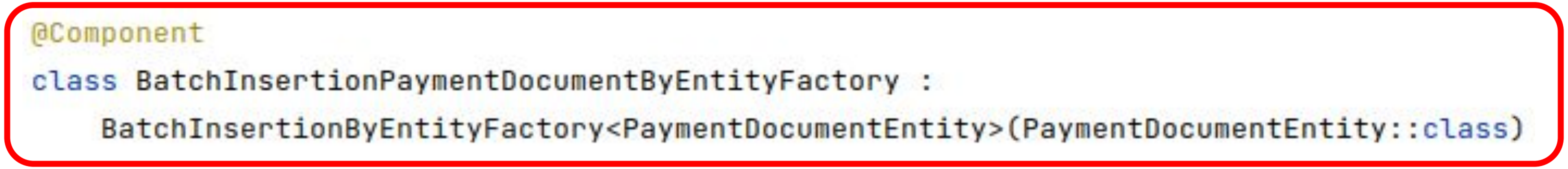
```
open class InsertByEntitySaver<E: BaseEntity>(
    private val processor: BatchInsertionByEntityProcessor,
    private val entityClass: KClass<E>,
    dataSource: DataSource,
    batchSize: Int
) : AbstractBatchInsertionByEntitySaver<E>(dataSource, batchSize) {

    private val dataForInsert = mutableListOf<String>()

    override fun addDataForSave(entity: E) {
        dataForInsert.add(processor.getStringForInsert(entity))
        super.addDataForSave(entity)
    }

    override fun saveData() {
        processor.insertDataToDataBase(entityClass, dataForInsert, conn)
        dataForInsert.clear()
    }
}
```

Application

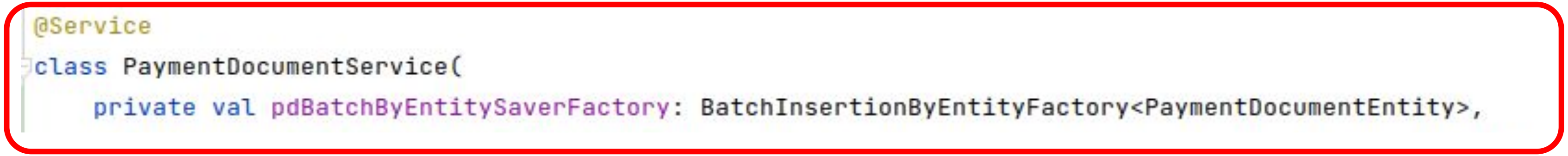


```
abstract class BatchInsertionByEntityFactory<E: BaseEntity>(
    private val entityClass: KClass<E>,
) : BatchInsertionByEntityFactory<E> {

    @Value("\${batch_insertion.batch_size}")
    private var batchSize: Int = 100

    @Autowired
    override lateinit var processor: BatchInsertionByEntityProcessor

    @Autowired
    override lateinit var dataSource: DataSource

    override fun getSaver(type: SaverType): BatchInsertionByEntitySaver<E> {

        val conn = dataSource.connection

        return when (type) {
            SaverType.COPY -> CopyByEntitySaver(processor, entityClass, conn, batchSize)
            SaverType.COPY_BINARY -> CopyBinaryByEntitySaver(processor, entityClass, conn, batchSize)
            SaverType.COPY_VIA_FILE -> CopyViaFileByEntitySaver(processor, entityClass, conn, batchSize)
            SaverType.COPY_BINARY_VIA_FILE -> CopyBinaryViaFileByEntitySaver(processor, entityClass, conn, batchSize)
            SaverType.INSERT -> InsertByEntitySaver(processor, entityClass, conn, batchSize)
            SaverType.UPDATE -> UpdateByEntitySaver(processor, entityClass, conn, batchSize)
        }

    }
}
```

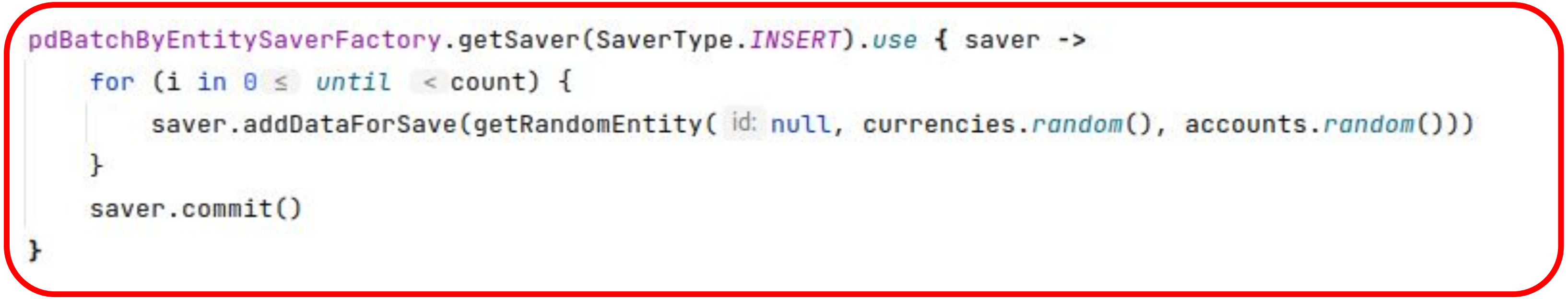
Application



```kotlin
@Component
class BatchInsertionPaymentDocumentByEntityFactory :
    BatchInsertionByEntityFactory<PaymentDocumentEntity>(PaymentDocumentEntity::class)
```



```kotlin
@Service
class PaymentDocumentService(
    private val pdBatchByEntitySaverFactory: BatchInsertionByEntityFactory<PaymentDocumentEntity>,
```

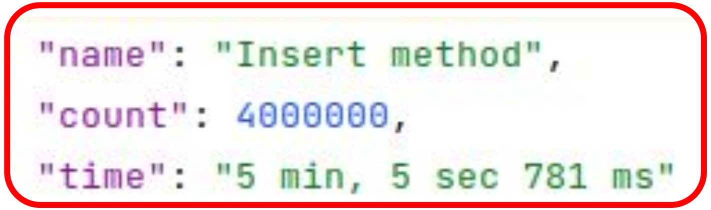


```kotlin
pdBatchByEntitySaverFactory.getSaver(SaverType.INSERT).use { saver ->
    for (i in 0 ≤ until < count) {
        saver.addDataForSave(getRandomEntity( id: null, currencies.random(), accounts.random()))
    }
    saver.commit()
}
```

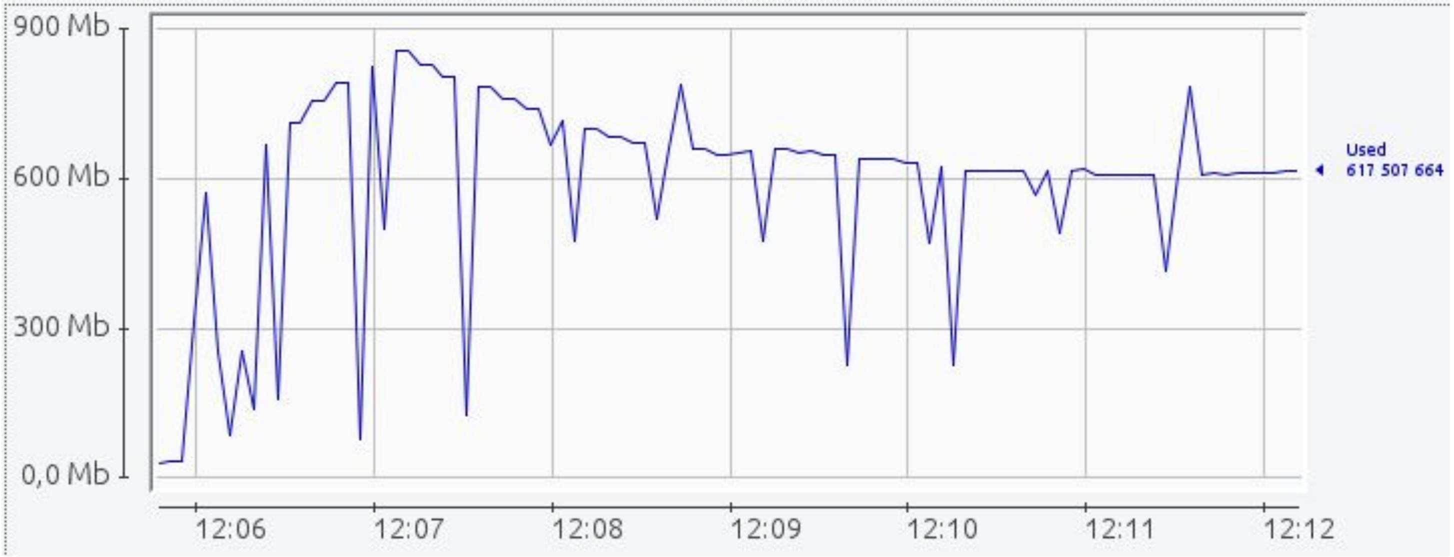
Application

Перейдем к замерам.

Insert with batch size 100k



```
"name": "Insert method",
"count": 4000000,
"time": "5 min, 5 sec 781 ms"
```



PostgreSQL

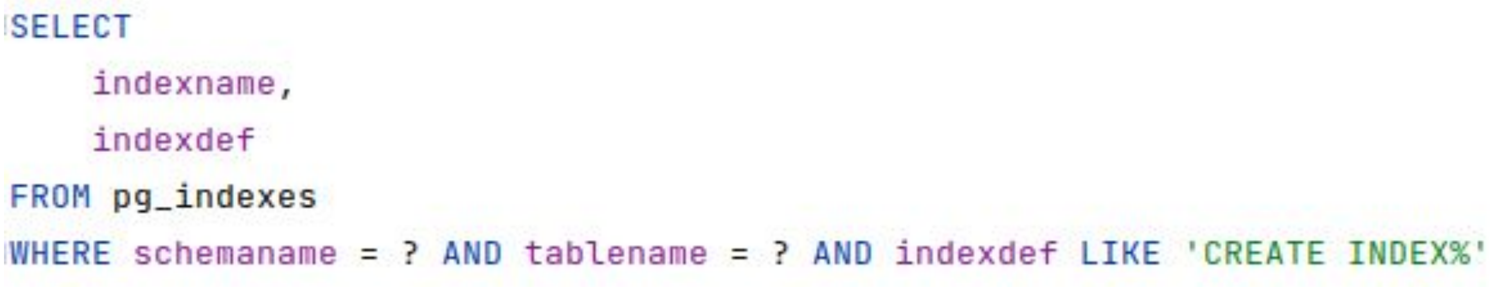
Что может мешать вставке больших объемов данных?

- Триггеры
- Индексы

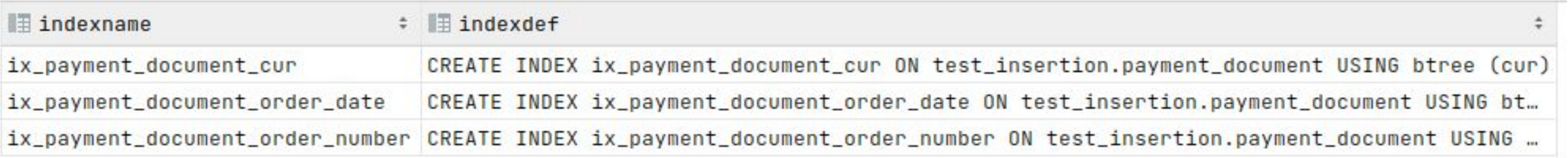
Индексы

- Можно удалить перед вставкой
- Затем вернуть

PostgreSQL

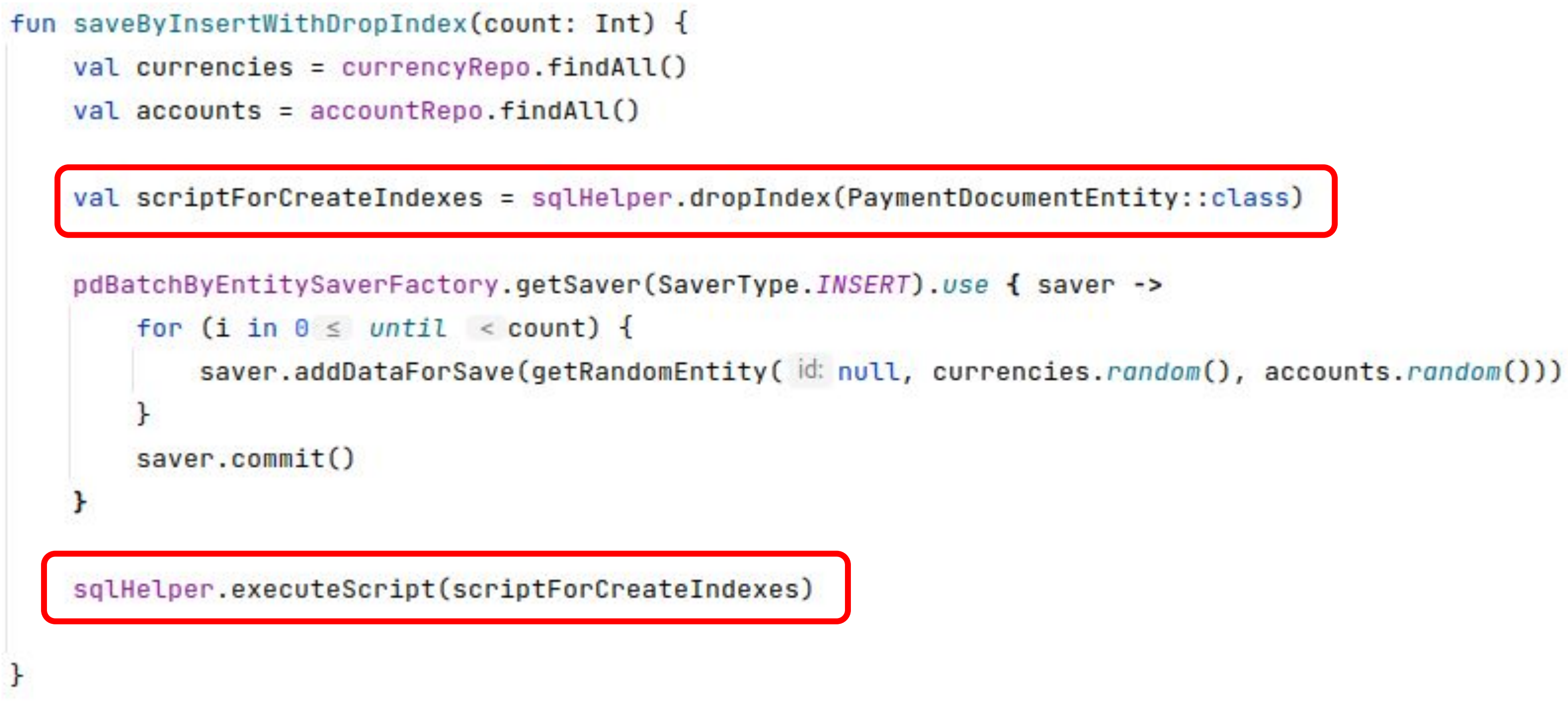


```
SELECT
    indexname,
    indexdef
FROM pg_indexes
WHERE schemaname = ? AND tablename = ? AND indexdef LIKE 'CREATE INDEX%'
```



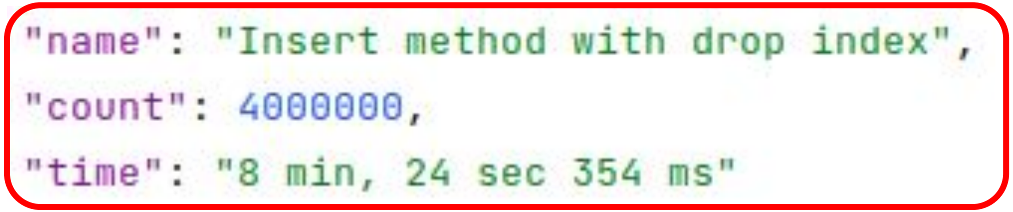
indexname	indexdef
ix_payment_document_cur	CREATE INDEX ix_payment_document_cur ON test_insertion.payment_document USING btree (cur)
ix_payment_document_order_date	CREATE INDEX ix_payment_document_order_date ON test_insertion.payment_document USING bt…
ix_payment_document_order_number	CREATE INDEX ix_payment_document_order_number ON test_insertion.payment_document USING …

Insert with drop index

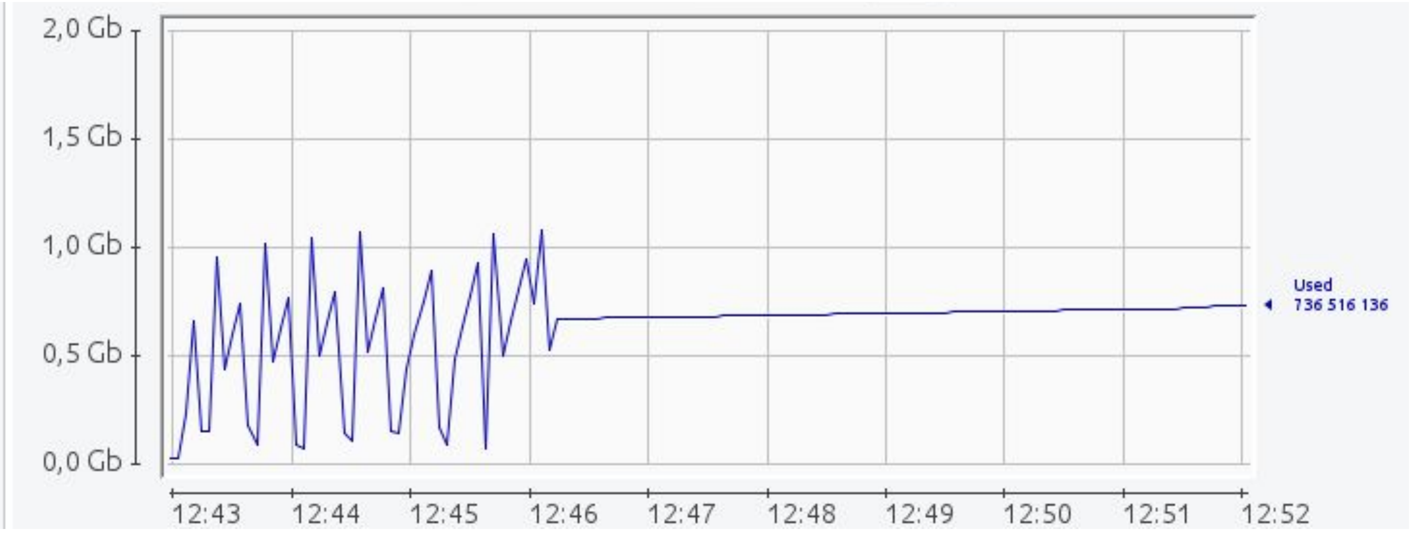


```
fun saveByInsertWithDropIndex(count: Int) {
    val currencies = currencyRepo.findAll()
    val accounts = accountRepo.findAll()

    val scriptForCreateIndexes = sqlHelper.dropIndex(PaymentDocumentEntity::class)

    pdBatchByEntitySaverFactory.getSaver(SaverType.INSERT).use { saver ->
        for (i in 0 ≤ until < count) {
            saver.addDataForSave(getRandomEntity( id: null, currencies.random(), accounts.random()))
        }
        saver.commit()
    }

    sqlHelper.executeScript(scriptForCreateIndexes)

}
```

Insert with drop index



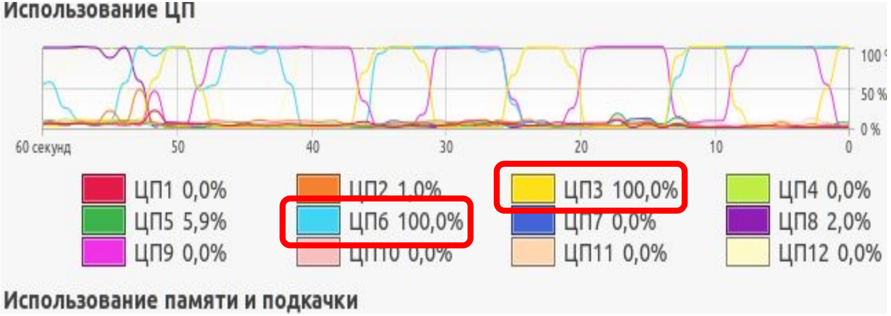
```
"name": "Insert method with drop index",
"count": 4000000,
"time": "8 min, 24 sec 354 ms"
```

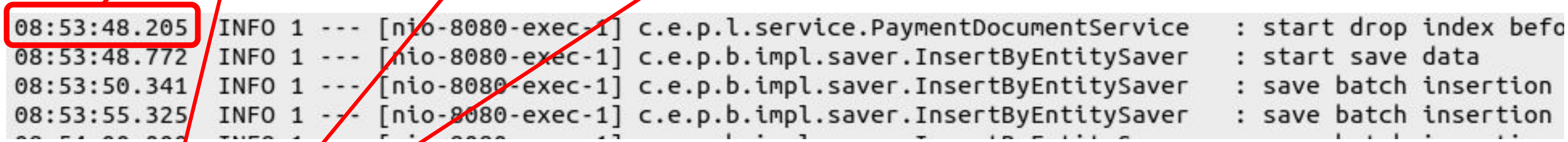


Insert with drop index

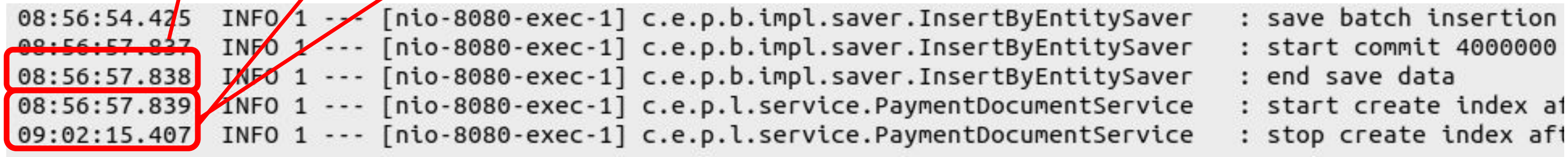
3 мин 9 сек
вставка без
индексов

5 мин 12 сек
восстановление
индексов





```
08:53:48.205  INFO 1 --- [nio-8080-exec-1] c.e.p.l.service.PaymentDocumentService    : start drop index befo
08:53:48.772  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.InsertByEntitySaver    : start save data
08:53:50.341  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.InsertByEntitySaver    : save batch insertion
08:53:55.325  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.InsertByEntitySaver    : save batch insertion
```



```
08:56:54.425  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.InsertByEntitySaver    : save batch insertion
08:56:57.837  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.InsertByEntitySaver    : start commit 4000000
08:56:57.838  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.InsertByEntitySaver    : end save data
08:56:57.839  INFO 1 --- [nio-8080-exec-1] c.e.p.l.service.PaymentDocumentService    : start create index af
09:02:15.407  INFO 1 --- [nio-8080-exec-1] c.e.p.l.service.PaymentDocumentService    : stop create index aft
```

Insert with drop index

Секционированные таблицы.

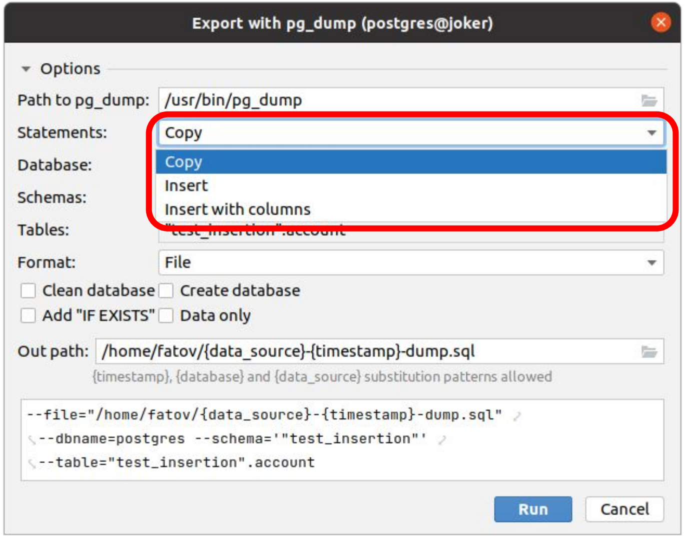
- Позволяет делать секции по диапазону, списку, хешу
- Можно добавлять секцию при массовой вставке
- В новой секции вставка будет осуществляться как в пустую таблицу
- https://postgrespro.ru/docs/postgresql/13/ddl-partitioning?ysclid=lnkccn1z6b239693409

PostgreSQL

Что еще ты умеешь, PostgreSQL?

PostgreSQL

Как еще PostgreSQL может вставлять данные в БД?



Export with pg_dump (postgres@joker)

▾ Options

Path to pg_dump: /usr/bin/pg_dump

Statements: Copy

- Copy
- Insert
- Insert with columns

Database:

Schemas:

Tables: "test_insertion".account

Format: File

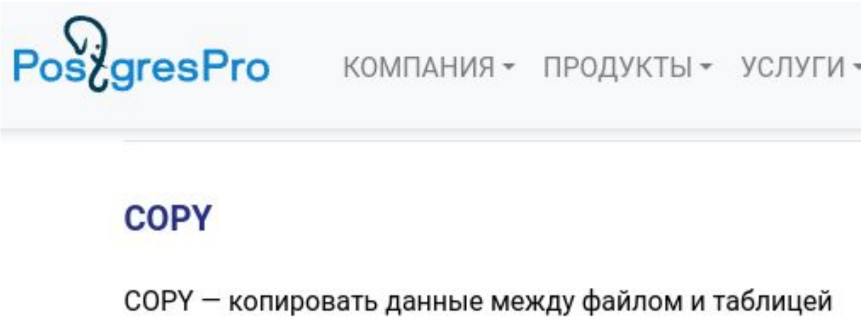
□ Clean database □ Create database

□ Add "IF EXISTS" □ Data only

Out path: /home/fatov/{data_source}-{timestamp}-dump.sql

{timestamp}, {database} and {data_source} substitution patterns allowed

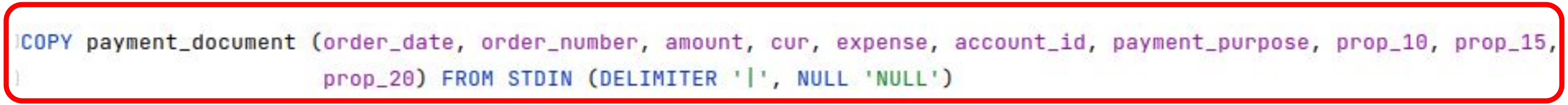
```
--file="/home/fatov/{data_source}-{timestamp}-dump.sql"
--dbname=postgres --schema='"test_insertion"'
--table="test_insertion".account
```



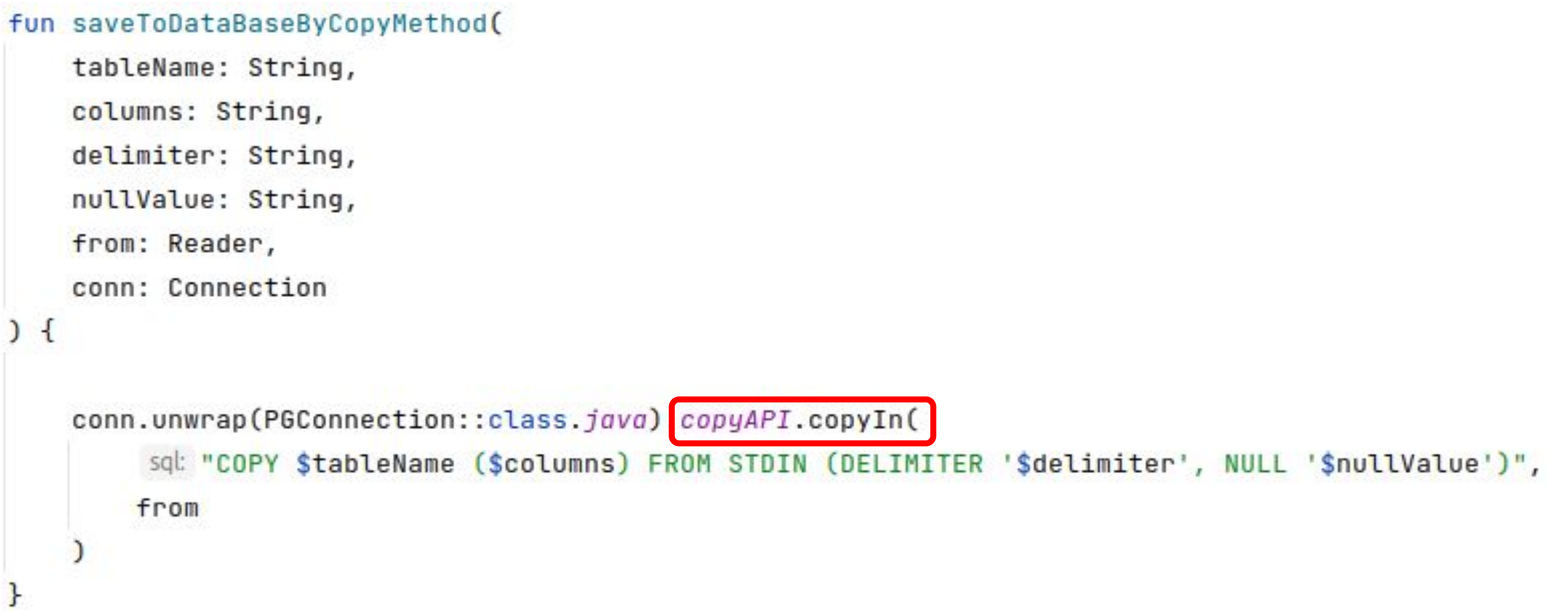
COPY

COPY — копировать данные между файлом и таблицей

Copy

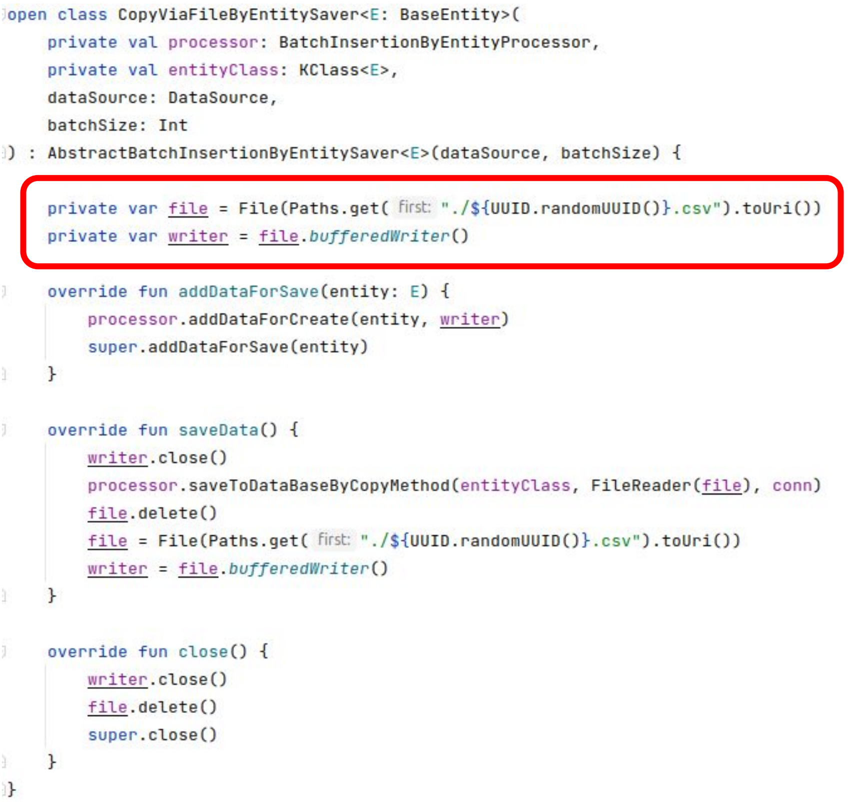


```
COPY payment_document (order_date, order_number, amount, cur, expense, account_id, payment_purpose, prop_10, prop_15,
                       prop_20) FROM STDIN (DELIMITER '|', NULL 'NULL')
```



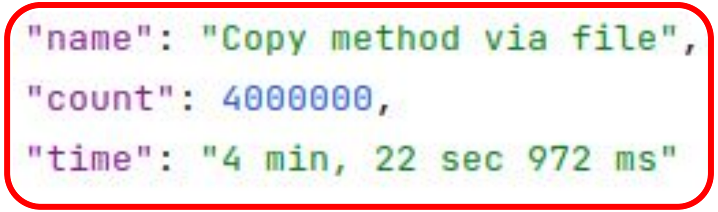
```
fun saveToDataBaseByCopyMethod(
    tableName: String,
    columns: String,
    delimiter: String,
    nullValue: String,
    from: Reader,
    conn: Connection
) {

    conn.unwrap(PGConnection::class.java) copyAPI.copyIn(
        sql: "COPY $tableName ($columns) FROM STDIN (DELIMITER '$delimiter', NULL '$nullValue')",
        from
    )
}
```

Copy



```
open class CopyViaFileByEntitySaver<E: BaseEntity>(
    private val processor: BatchInsertionByEntityProcessor,
    private val entityClass: KClass<E>,
    dataSource: DataSource,
    batchSize: Int
) : AbstractBatchInsertionByEntitySaver<E>(dataSource, batchSize) {

    private var file = File(Paths.get( first: "./${UUID.randomUUID()}.csv").toUri())
    private var writer = file.bufferedWriter()

    override fun addDataForSave(entity: E) {
        processor.addDataForCreate(entity, writer)
        super.addDataForSave(entity)
    }

    override fun saveData() {
        writer.close()
        processor.saveToDataBaseByCopyMethod(entityClass, FileReader(file), conn)
        file.delete()
        file = File(Paths.get( first: "./${UUID.randomUUID()}.csv").toUri())
        writer = file.bufferedWriter()
    }

    override fun close() {
        writer.close()
        file.delete()
        super.close()
    }
}
```

Copy via file



```
"name": "Copy method via file",
"count": 4000000,
"time": "4 min, 22 sec 972 ms"
```

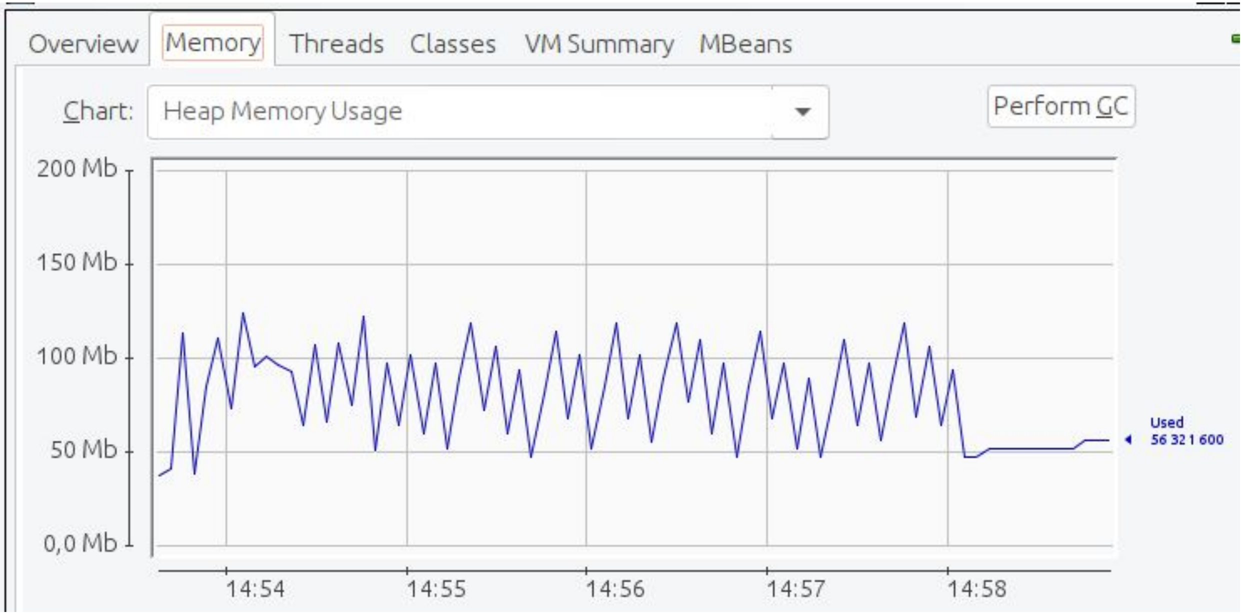
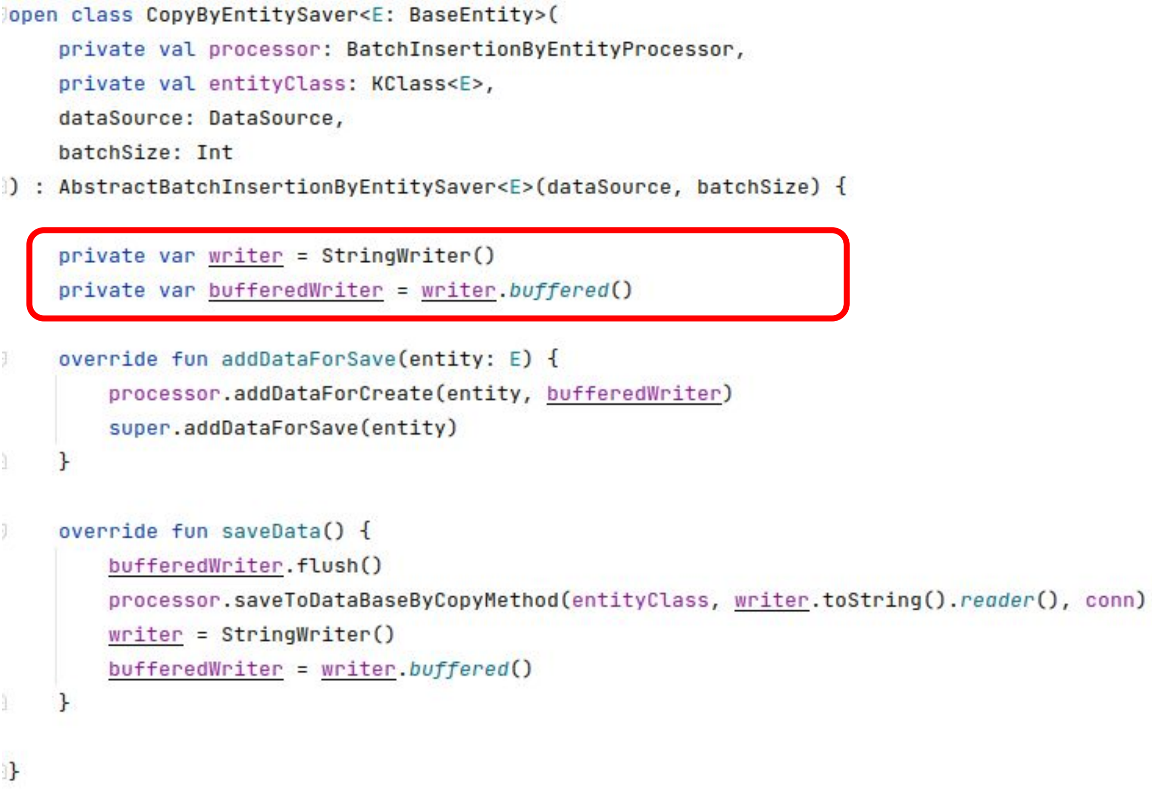


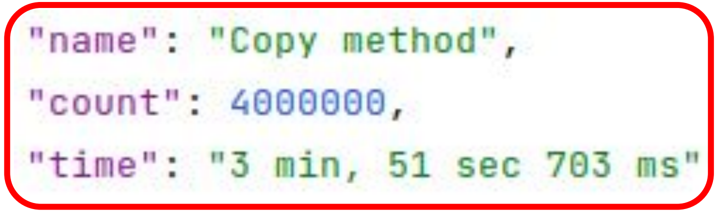
Chart: Heap Memory Usage

Copy

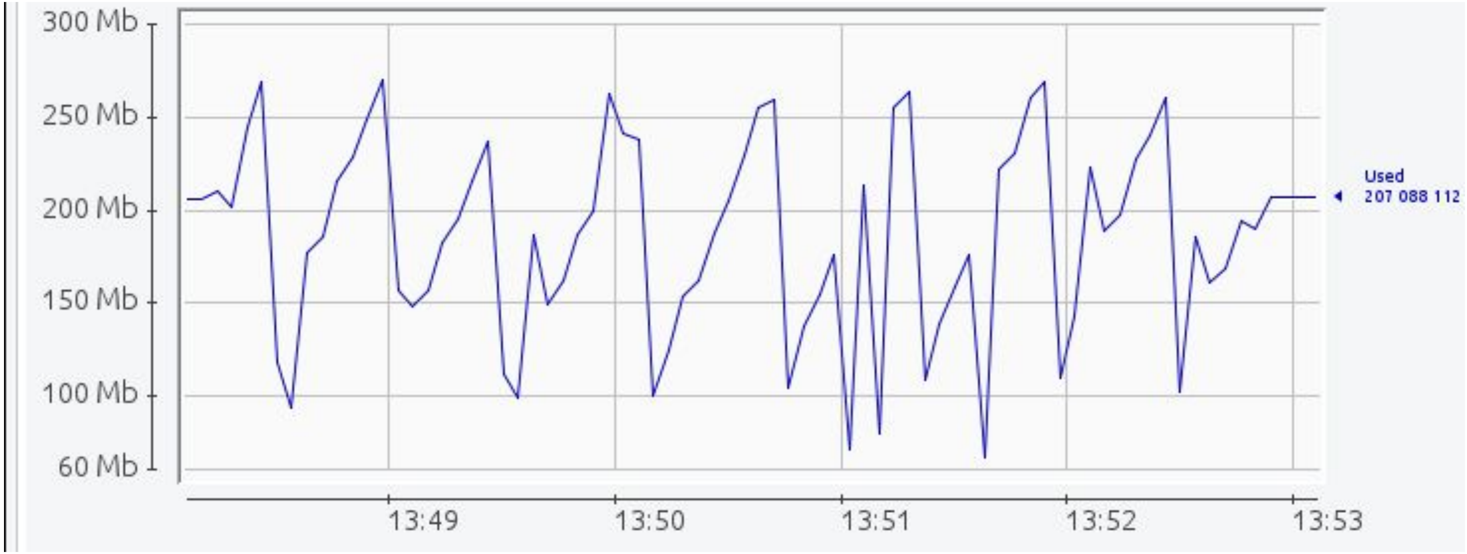


```kotlin
open class CopyByEntitySaver<E: BaseEntity>(
    private val processor: BatchInsertionByEntityProcessor,
    private val entityClass: KClass<E>,
    dataSource: DataSource,
    batchSize: Int
) : AbstractBatchInsertionByEntitySaver<E>(dataSource, batchSize) {

    private var writer = StringWriter()
    private var bufferedWriter = writer.buffered()

    override fun addDataForSave(entity: E) {
        processor.addDataForCreate(entity, bufferedWriter)
        super.addDataForSave(entity)
    }

    override fun saveData() {
        bufferedWriter.flush()
        processor.saveToDataBaseByCopyMethod(entityClass, writer.toString().reader(), conn)
        writer = StringWriter()
        bufferedWriter = writer.buffered()
    }

}
```

Copy with batch size 100_000



```
"name": "Copy method",
"count": 4000000,
"time": "3 min, 51 sec 703 ms"
```

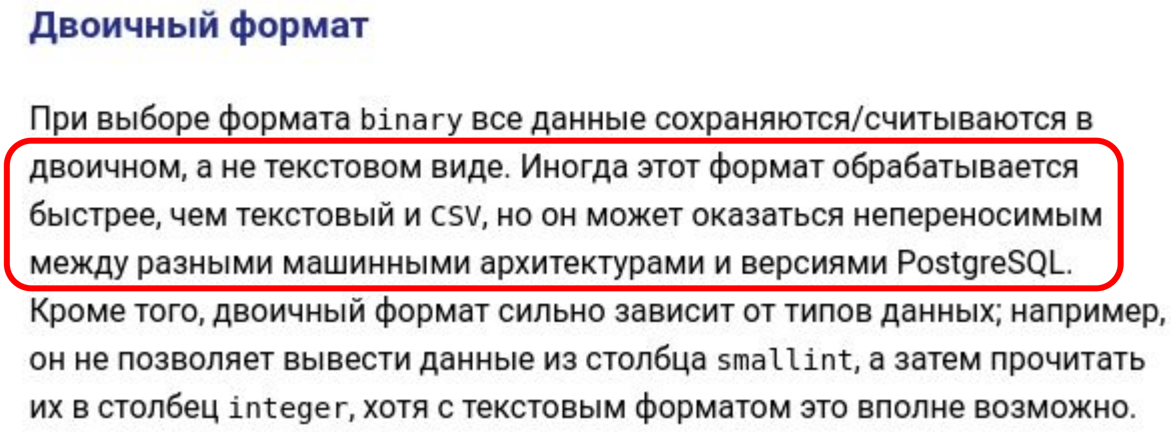


Copy

Что ещё?

Copy

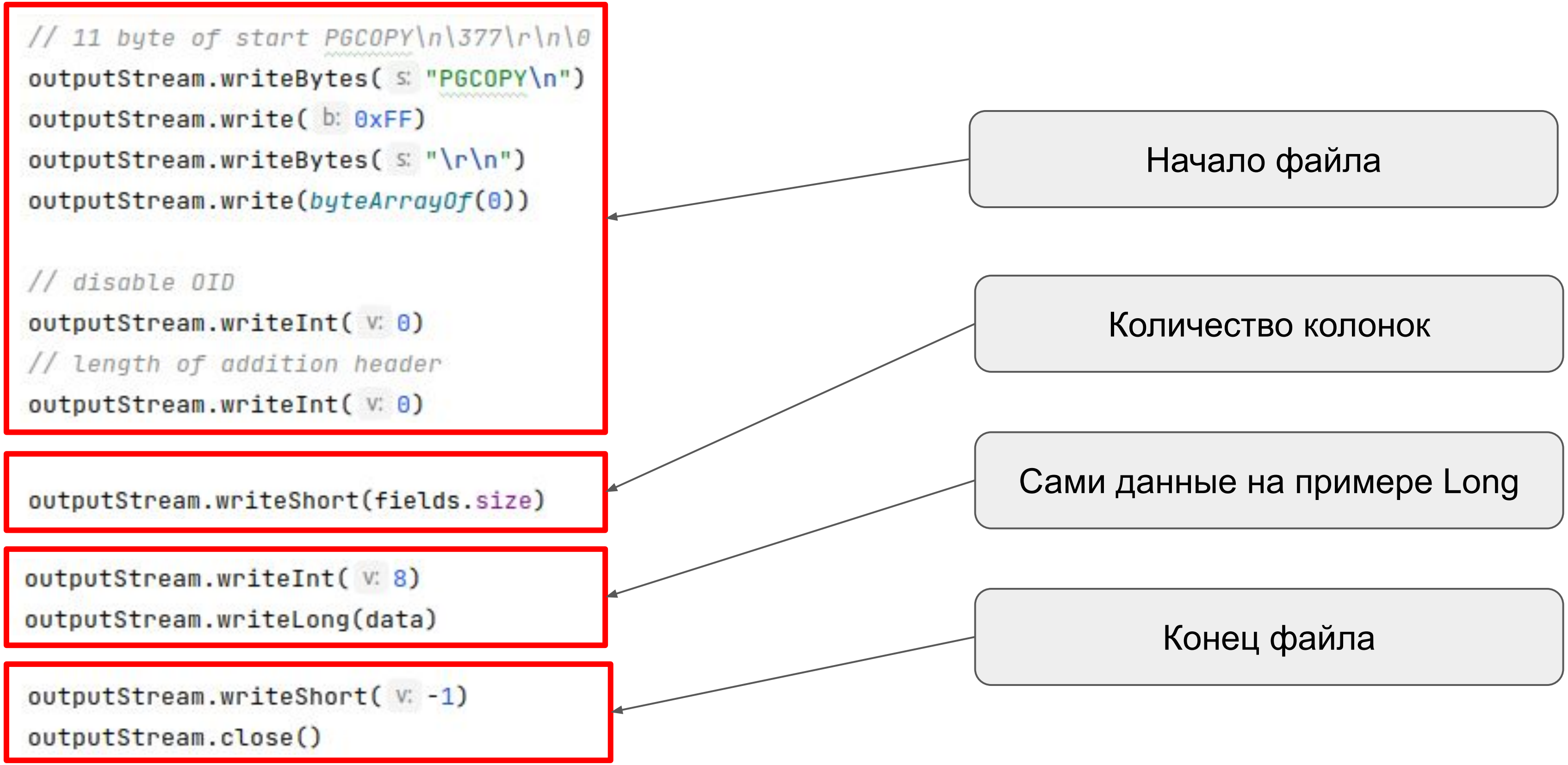
Что еще?



Двоичный формат

При выборе формата `binary` все данные сохраняются/считываются в двоичном, а не текстовом виде. Иногда этот формат обрабатывается быстрее, чем текстовый и `CSV`, но он может оказаться непереносимым между разными машинными архитектурами и версиями PostgreSQL. Кроме того, двоичный формат сильно зависит от типов данных; например, он не позволяет вывести данные из столбца `smallint`, а затем прочитать их в столбец `integer`, хотя с текстовым форматом это вполне возможно.

Copy binary



```
// 11 byte of start PGCOPY\n\377\r\n\0
outputStream.writeBytes( s: "PGCOPY\n")
outputStream.write( b: 0xFF)
outputStream.writeBytes( s: "\r\n")
outputStream.write(byteArrayOf(0))

// disable OID
outputStream.writeInt( v: 0)
// length of addition header
outputStream.writeInt( v: 0)
```

Начало файла

```
outputStream.writeShort(fields.size)
```

Количество колонок

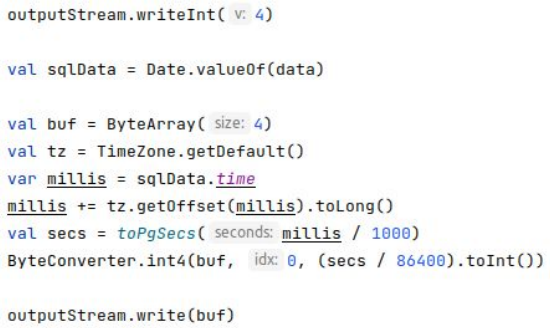
```
outputStream.writeInt( v: 8)
outputStream.writeLong(data)
```

Сами данные на примере Long

```
outputStream.writeShort( v: -1)
outputStream.close()
```

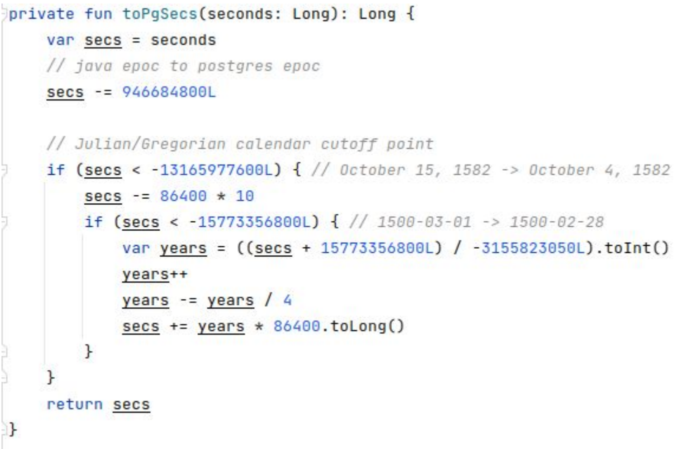
Конец файла

Copy binary. LocalDate



```
outputStream.writeInt( v: 4)

val sqlData = Date.valueOf(data)

val buf = ByteArray( size: 4)
val tz = TimeZone.getDefault()
var millis = sqlData.time
millis += tz.getOffset(millis).toLong()
val secs = toPgSecs( seconds: millis / 1000)
ByteConverter.int4(buf,  idx: 0, (secs / 86400).toInt())

outputStream.write(buf)
```

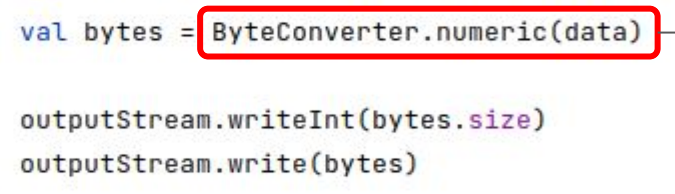
Преобразование LocalDate в binary



```
private fun toPgSecs(seconds: Long): Long {
    var secs = seconds
    // java epoc to postgres epoc
    secs -= 946684800L

    // Julian/Gregorian calendar cutoff point
    if (secs < -13165977600L) { // October 15, 1582 -> October 4, 1582
        secs -= 86400 * 10
        if (secs < -15773356800L) { // 1500-03-01 -> 1500-02-28
            var years = ((secs + 15773356800L) / -3155823050L).toInt()
            years++
            years -= years / 4
            secs += years * 86400.toLong()
        }
    }
    return secs
}
```

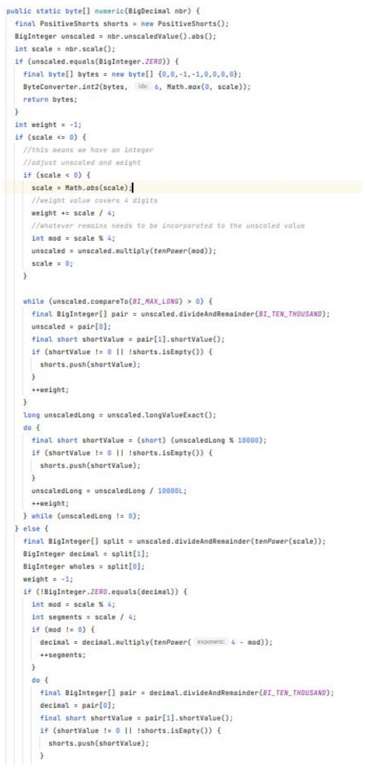
Взято из драйвера PostgreSQL

Copy binary. BigDecimal



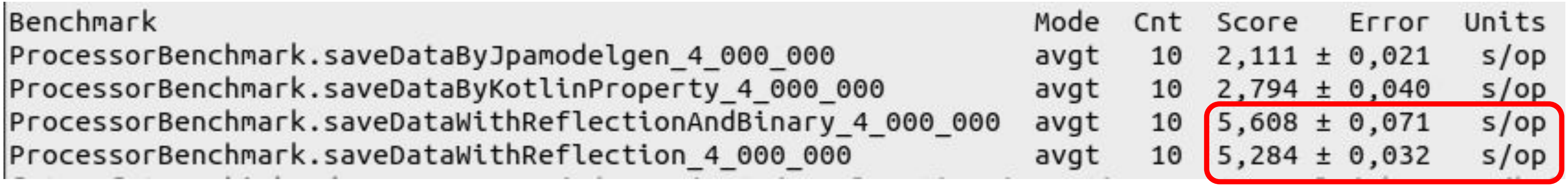
```
val bytes = ByteConverter.numeric(data)

outputStream.writeInt(bytes.size)
outputStream.write(bytes)
```

110 строк кода



```
public static byte[] numeric(BigDecimal nbr) {
  final PositiveShorts shorts = new PositiveShorts();
  BigInteger unscaled = nbr.unscaledValue().abs();
  int scale = nbr.scale();
  if (unscaled.equals(BigInteger.ZERO)) {
    final byte[] bytes = new byte[] {0,0,-1,-1,0,0,0,0};
    ByteConverter.int2(bytes,  idx: 6, Math.max(0, scale));
    return bytes;
  }
  int weight = -1;
  if (scale <= 0) {
    //this means we have an integer
    //adjust unscaled and weight
    if (scale < 0) {
      scale = Math.abs(scale);
      //weight value covers 4 digits
      weight += scale / 4;
      //whatever remains needs to be incorporated to the unscaled value
      int mod = scale % 4;
      unscaled = unscaled.multiply(tenPower(mod));
      scale = 0;
    }

    while (unscaled.compareTo(BI_MAX_LONG) > 0) {
      final BigInteger[] pair = unscaled.divideAndRemainder(BI_TEN_THOUSAND);
      unscaled = pair[0];
      final short shortValue = pair[1].shortValue();
      if (shortValue != 0 || !shorts.isEmpty()) {
        shorts.push(shortValue);
      }
      ++weight;
    }
    long unscaledLong = unscaled.longValueExact();
    do {
      final short shortValue = (short) (unscaledLong % 10000);
      if (shortValue != 0 || !shorts.isEmpty()) {
        shorts.push(shortValue);
      }
      unscaledLong = unscaledLong / 10000L;
      ++weight;
    } while (unscaledLong != 0);
  } else {
    final BigInteger[] split = unscaled.divideAndRemainder(tenPower(scale));
    BigInteger decimal = split[1];
    BigInteger wholes = split[0];
    weight = -1;
    if (!BigInteger.ZERO.equals(decimal)) {
      int mod = scale % 4;
      int segments = scale / 4;
      if (mod != 0) {
        decimal = decimal.multiply(tenPower( exponent: 4 - mod));
        ++segments;
      }
      do {
        final BigInteger[] pair = decimal.divideAndRemainder(BI_TEN_THOUSAND);
        decimal = pair[0];
        final short shortValue = pair[1].shortValue();
        if (shortValue != 0 || !shorts.isEmpty()) {
          shorts.push(shortValue);
        }
```

Benchmark. Reflection vs jpa-modelgen vs KProperty vs Binary file

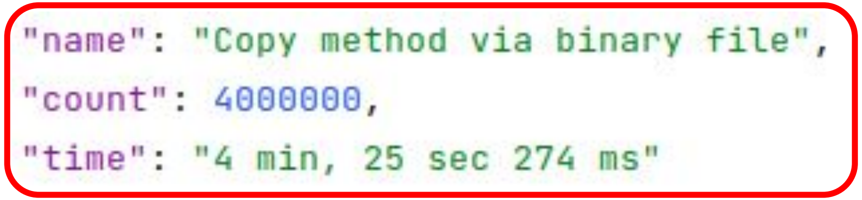


```
Benchmark                                                    Mode  Cnt  Score   Error  Units
ProcessorBenchmark.saveDataByJpamodelgen_4_000_000           avgt   10  2,111 ± 0,021   s/op
ProcessorBenchmark.saveDataByKotlinProperty_4_000_000        avgt   10  2,794 ± 0,040   s/op
ProcessorBenchmark.saveDataWithReflectionAndBinary_4_000_000 avgt   10  5,608 ± 0,071   s/op
ProcessorBenchmark.saveDataWithReflection_4_000_000          avgt   10  5,284 ± 0,032   s/op
```

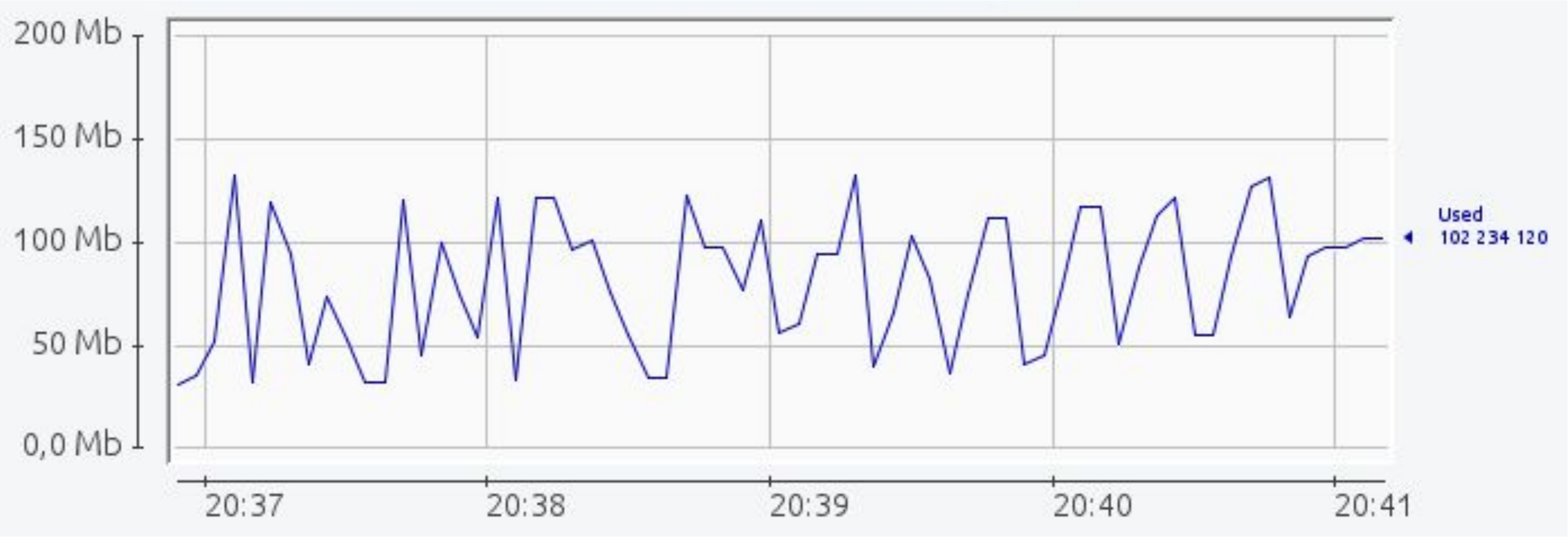
Copy binary

Перейдем к замерам.

Copy binary via file

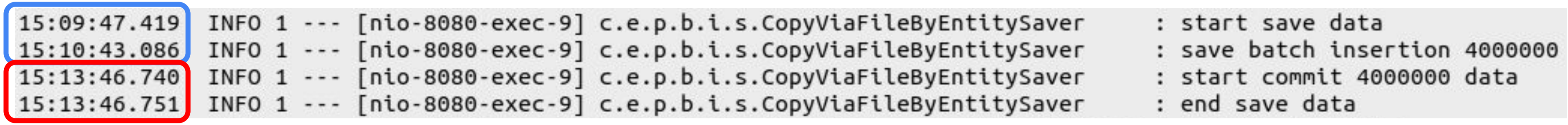


```
"name": "Copy method via binary file",
"count": 4000000,
"time": "4 min, 25 sec 274 ms"
```

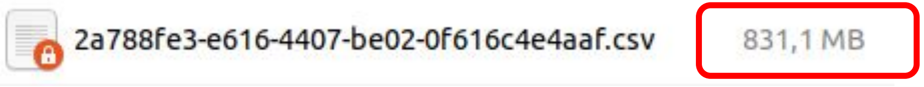


Copy. csv vs binary

CSV. Размер файла: 831 мб. Создание файла: 56 сек, вставка в БД: 3 мин 3 сек.

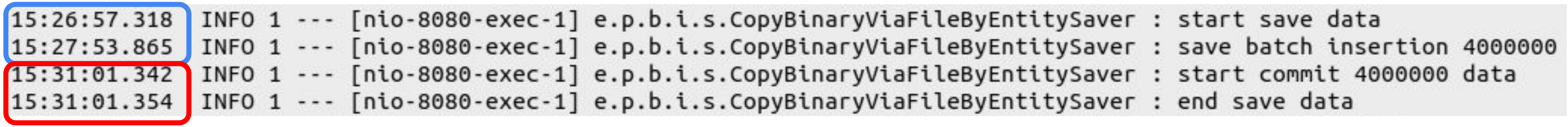


```
15:09:47.419  INFO 1 --- [nio-8080-exec-9] c.e.p.b.i.s.CopyViaFileByEntitySaver     : start save data
15:10:43.086  INFO 1 --- [nio-8080-exec-9] c.e.p.b.i.s.CopyViaFileByEntitySaver     : save batch insertion 4000000
15:13:46.740  INFO 1 --- [nio-8080-exec-9] c.e.p.b.i.s.CopyViaFileByEntitySaver     : start commit 4000000 data
15:13:46.751  INFO 1 --- [nio-8080-exec-9] c.e.p.b.i.s.CopyViaFileByEntitySaver     : end save data
```

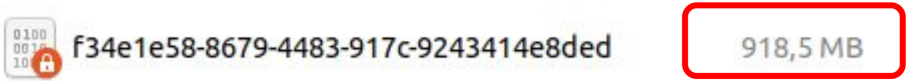


2a788fe3-e616-4407-be02-0f616c4e4aaf.csv 831,1 MB

Binary. Размер файла: 918 мб. Создание файла: 56 сек, вставка в БД: 3 мин 8 сек.

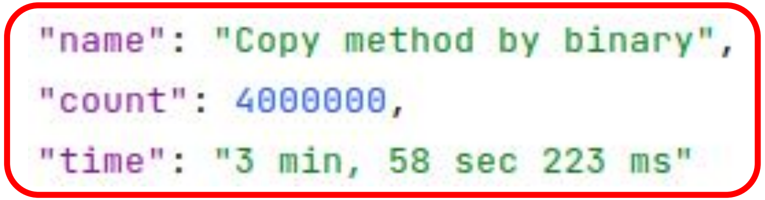


```
15:26:57.318  INFO 1 --- [nio-8080-exec-1] e.p.b.i.s.CopyBinaryViaFileByEntitySaver : start save data
15:27:53.865  INFO 1 --- [nio-8080-exec-1] e.p.b.i.s.CopyBinaryViaFileByEntitySaver : save batch insertion 4000000
15:31:01.342  INFO 1 --- [nio-8080-exec-1] e.p.b.i.s.CopyBinaryViaFileByEntitySaver : start commit 4000000 data
15:31:01.354  INFO 1 --- [nio-8080-exec-1] e.p.b.i.s.CopyBinaryViaFileByEntitySaver : end save data
```

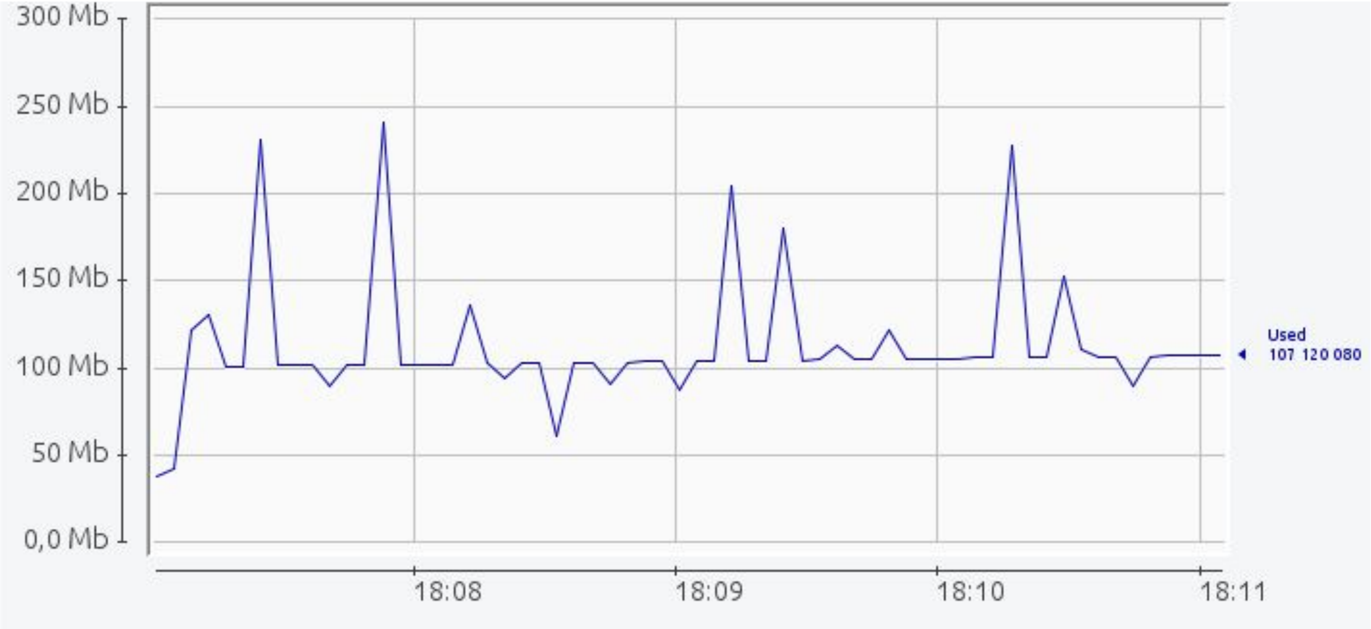


f34e1e58-8679-4483-917c-9243414e8ded 918,5 MB

Copy binary with batch size 100_000



```
"name": "Copy method by binary",
"count": 4000000,
"time": "3 min, 58 sec 223 ms"
```



Copy

Как прикрутить это к транзакциям Spring?

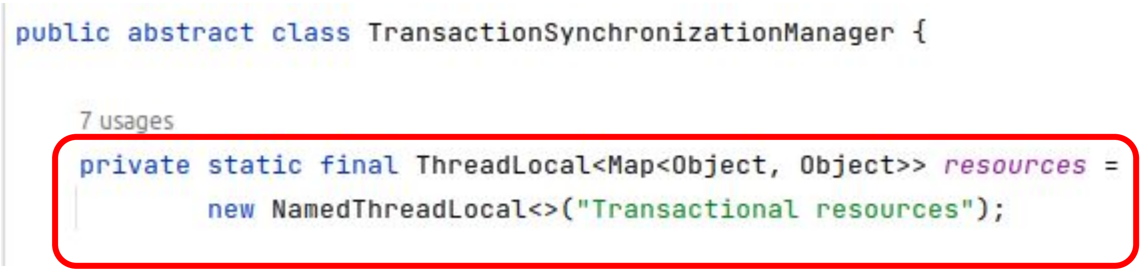
Copy

Как прикр м Spring?

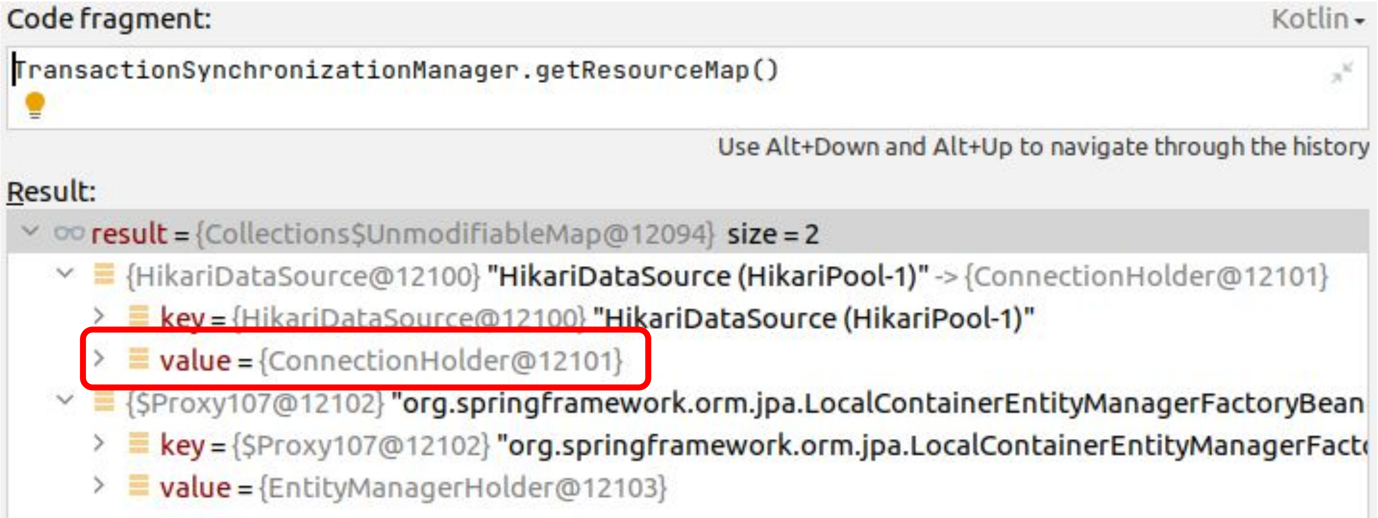
Copy

```
@Transactional
fun saveByCopyViaSpring(count: Int) {
    val currencies = currencyRepo.findAll()
    val accounts = accountRepo.findAll()

    for (i in 0 ≤ until < count) {
        pdCustomRepository.saveByCopy(
            getRandomEntity( id: null, currencies.random(), accounts.random())
        )
    }

}
```

Transaction synchronization manager



```
public abstract class TransactionSynchronizationManager {

    7 usages
    private static final ThreadLocal<Map<Object, Object>> resources =
            new NamedThreadLocal<>("Transactional resources");
```



Code fragment:

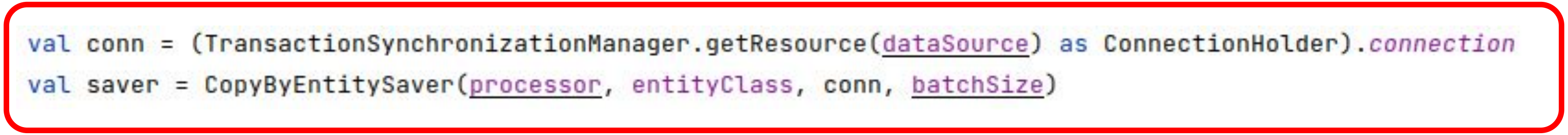
```
TransactionSynchronizationManager.getResourceMap()
```

Use Alt+Down and Alt+Up to navigate through the history

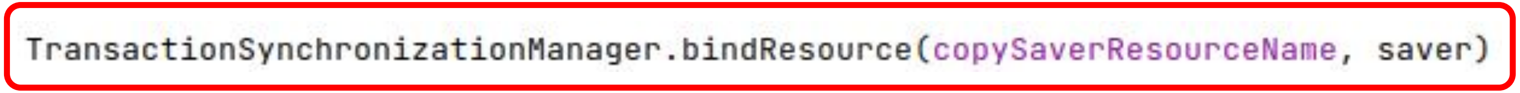
Result:

```
result = {Collections$UnmodifiableMap@12094} size = 2
  {HikariDataSource@12100} "HikariDataSource (HikariPool-1)" -> {ConnectionHolder@12101}
    key = {HikariDataSource@12100} "HikariDataSource (HikariPool-1)"
    value = {ConnectionHolder@12101}
  {$Proxy107@12102} "org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean
    key = {$Proxy107@12102} "org.springframework.orm.jpa.LocalContainerEntityManagerFact
    value = {EntityManagerHolder@12103}
```

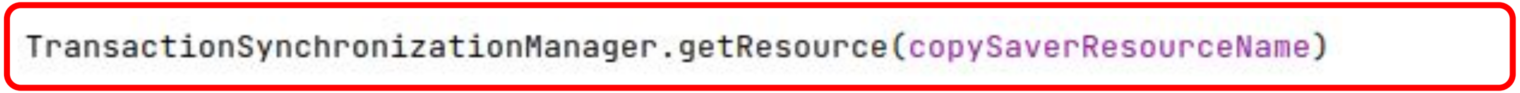
Transaction synchronization manager



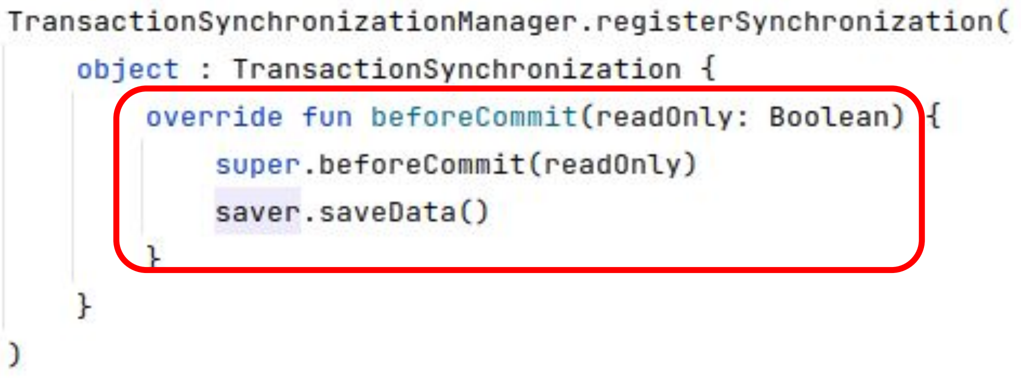
```
val conn = (TransactionSynchronizationManager.getResource(dataSource) as ConnectionHolder).connection
val saver = CopyByEntitySaver(processor, entityClass, conn, batchSize)
```



```
TransactionSynchronizationManager.bindResource(copySaverResourceName, saver)
```

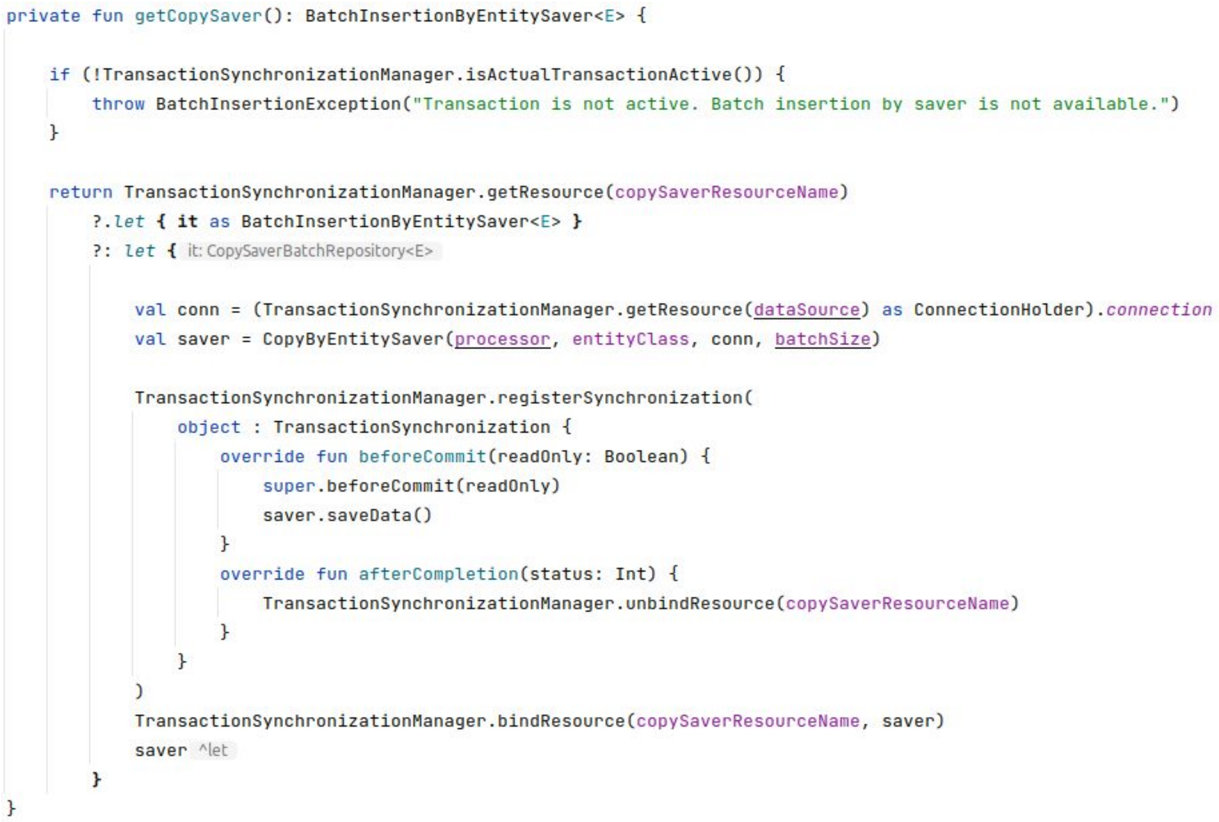


```
TransactionSynchronizationManager.getResource(copySaverResourceName)
```



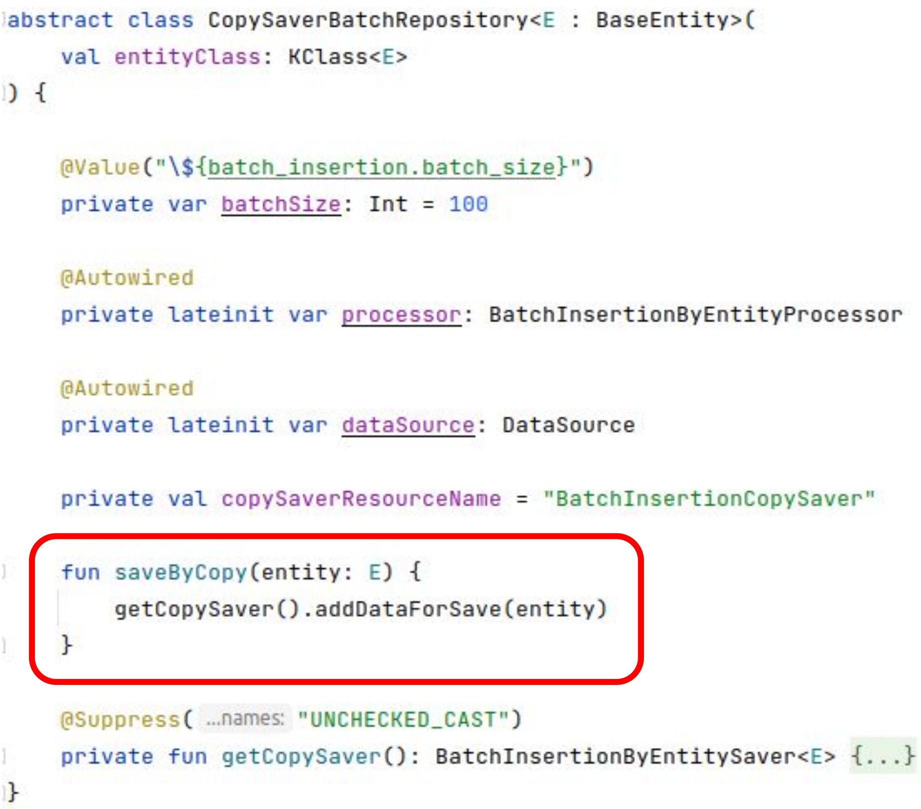
```
TransactionSynchronizationManager.registerSynchronization(
    object : TransactionSynchronization {
        override fun beforeCommit(readOnly: Boolean) {
            super.beforeCommit(readOnly)
            saver.saveData()
        }
    }
)
```

JPA repository with copy saver



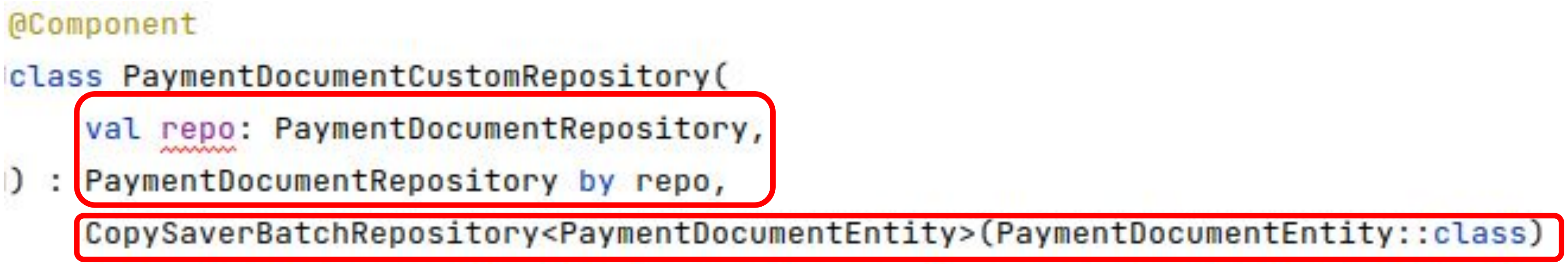
```
private fun getCopySaver(): BatchInsertionByEntitySaver<E> {

    if (!TransactionSynchronizationManager.isActualTransactionActive()) {
        throw BatchInsertionException("Transaction is not active. Batch insertion by saver is not available.")
    }

    return TransactionSynchronizationManager.getResource(copySaverResourceName)
        ?.let { it as BatchInsertionByEntitySaver<E> }
        ?: let { it: CopySaverBatchRepository<E>

            val conn = (TransactionSynchronizationManager.getResource(dataSource) as ConnectionHolder).connection
            val saver = CopyByEntitySaver(processor, entityClass, conn, batchSize)

            TransactionSynchronizationManager.registerSynchronization(
                object : TransactionSynchronization {
                    override fun beforeCommit(readOnly: Boolean) {
                        super.beforeCommit(readOnly)
                        saver.saveData()
                    }
                    override fun afterCompletion(status: Int) {
                        TransactionSynchronizationManager.unbindResource(copySaverResourceName)
                    }
                }
            )
            TransactionSynchronizationManager.bindResource(copySaverResourceName, saver)
            saver ^let
        }
}
```

JPA repository with copy saver



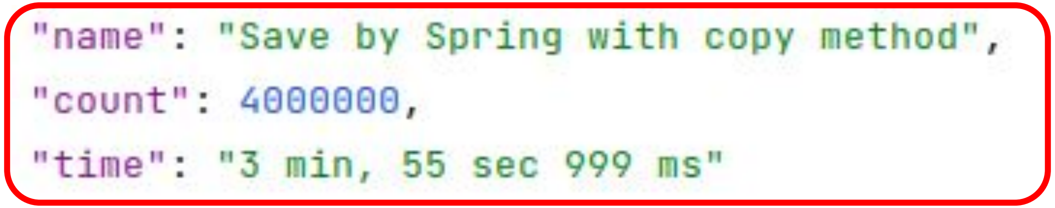
```kotlin
abstract class CopySaverBatchRepository<E : BaseEntity>(
    val entityClass: KClass<E>
) {

    @Value("\${batch_insertion.batch_size}")
    private var batchSize: Int = 100

    @Autowired
    private lateinit var processor: BatchInsertionByEntityProcessor

    @Autowired
    private lateinit var dataSource: DataSource

    private val copySaverResourceName = "BatchInsertionCopySaver"

    fun saveByCopy(entity: E) {
        getCopySaver().addDataForSave(entity)
    }

    @Suppress( ...names: "UNCHECKED_CAST")
    private fun getCopySaver(): BatchInsertionByEntitySaver<E> {...}
}
```

JPA repository with copy saver

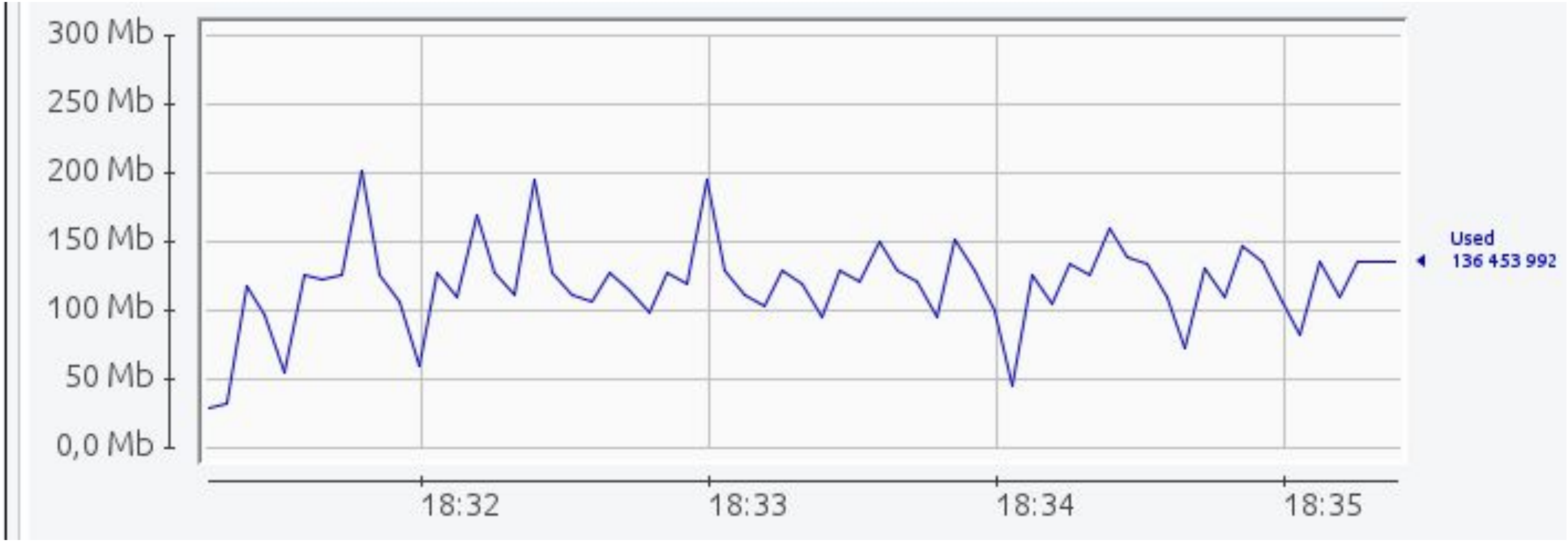


```
@Component
class PaymentDocumentCustomRepository(
    val repo: PaymentDocumentRepository,
) : PaymentDocumentRepository by repo,
    CopySaverBatchRepository<PaymentDocumentEntity>(PaymentDocumentEntity::class)
```

Save by custom repository with copy saver



```
"name": "Save by Spring with copy method",
"count": 4000000,
"time": "3 min, 55 sec 999 ms"
```



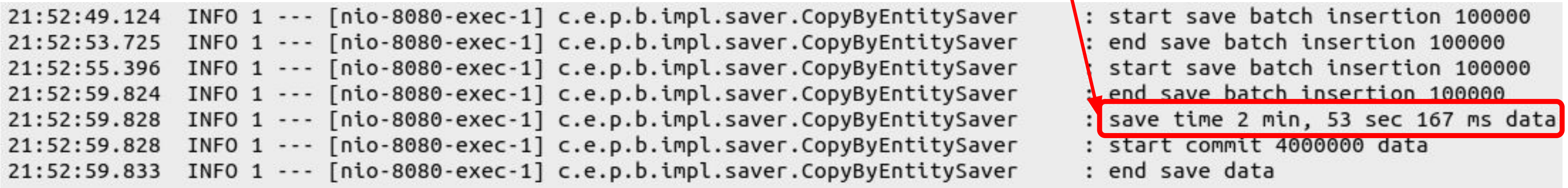
Copy

Как распараллелить вставку данных?

Copy

Где время?

⅔ от общего времени



```
21:52:49.124  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.CopyByEntitySaver     : start save batch insertion 100000
21:52:53.725  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.CopyByEntitySaver     : end save batch insertion 100000
21:52:55.396  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.CopyByEntitySaver     : start save batch insertion 100000
21:52:59.824  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.CopyByEntitySaver     : end save batch insertion 100000
21:52:59.828  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.CopyByEntitySaver     : save time 2 min, 53 sec 167 ms data
21:52:59.828  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.CopyByEntitySaver     : start commit 4000000 data
21:52:59.833  INFO 1 --- [nio-8080-exec-1] c.e.p.b.impl.saver.CopyByEntitySaver     : end save data
```

Copy

Где время?

```
21:52:49.124  INFO 1 --- [nio-8          start save batch insertion 100000
21:52:53.725  INFO 1 --- [nio-8          end save batch insertion 100000
21:52:55.396  INFO 1 --- [nio-8          start save batch insertion 100000
21:52:59.824  INFO 1 --- [nio-8          end save batch insertion 100000
21:52:59.828  INFO 1 --- [nio-8          save time 2 min, 53 sec 167 ms data
21:52:59.828  INFO 1 --- [nio-8          start commit 4000000 data
21:52:59.833  INFO 1 --- [nio-8          end save data
```

Copy with concurrent saver

Что для это сделаем:

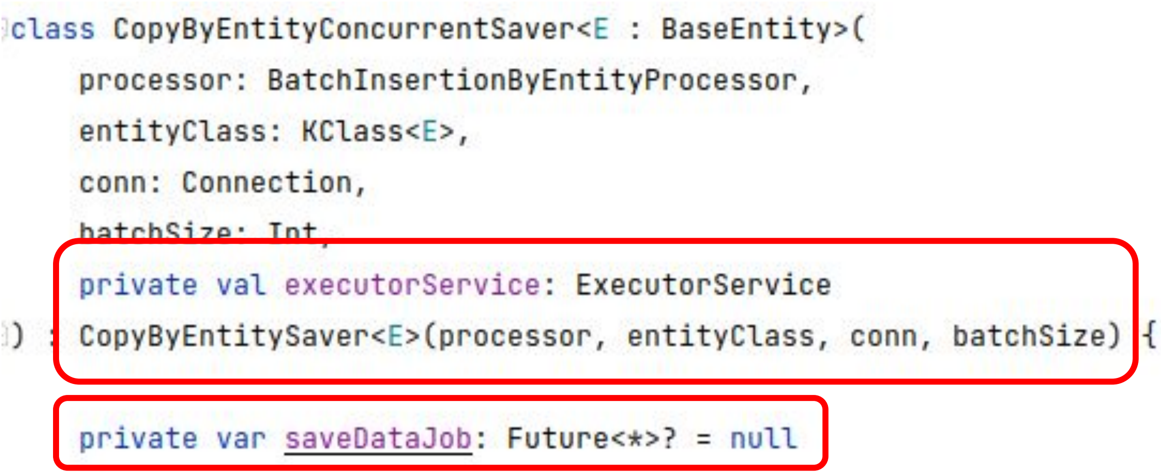
- Отдельный ThreadPoolExecutor для управления задачами
- ConcurrentSaverHandler который будет распределять нагрузку по нескольким Saver
- Расширим функциональность CopyByEntitySaver для неблокирующей отправки данных в БД

Copy with concurrent saver

Какие минусы у этого подхода:

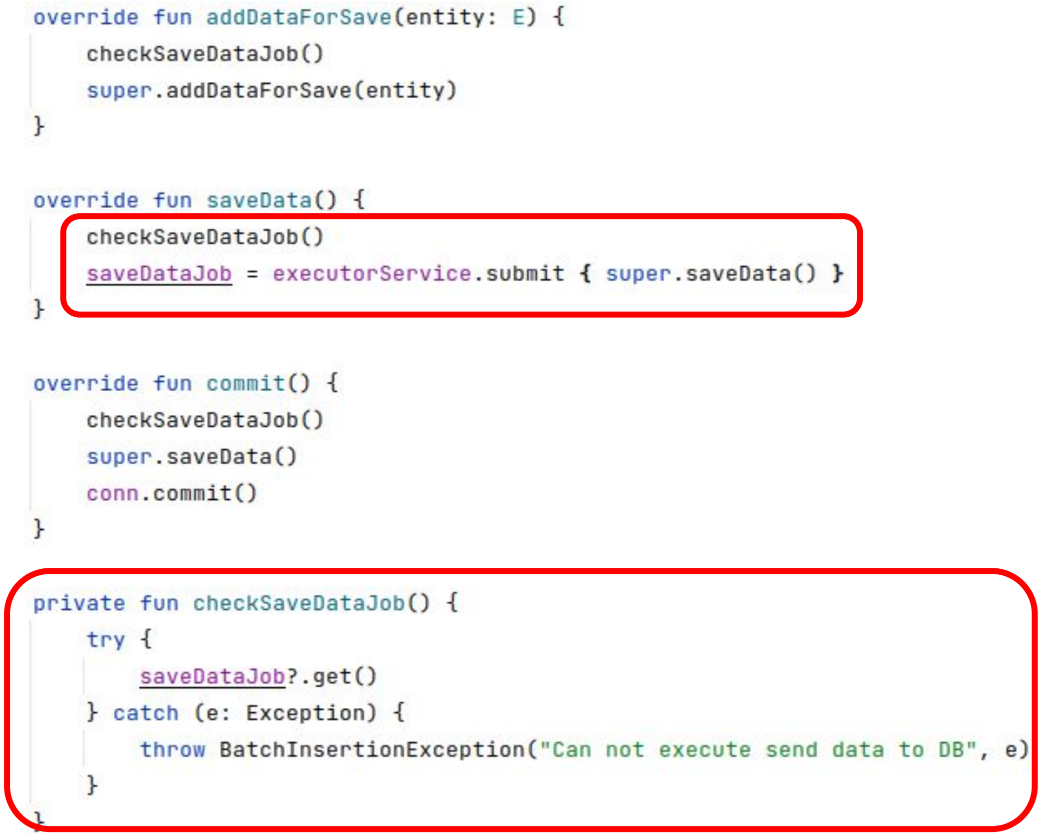
- Неконтролируемое потребление коннектов при больших нагрузках
- Транзакция для параллельных saver перестанет быть атомарной

Concurrent saver



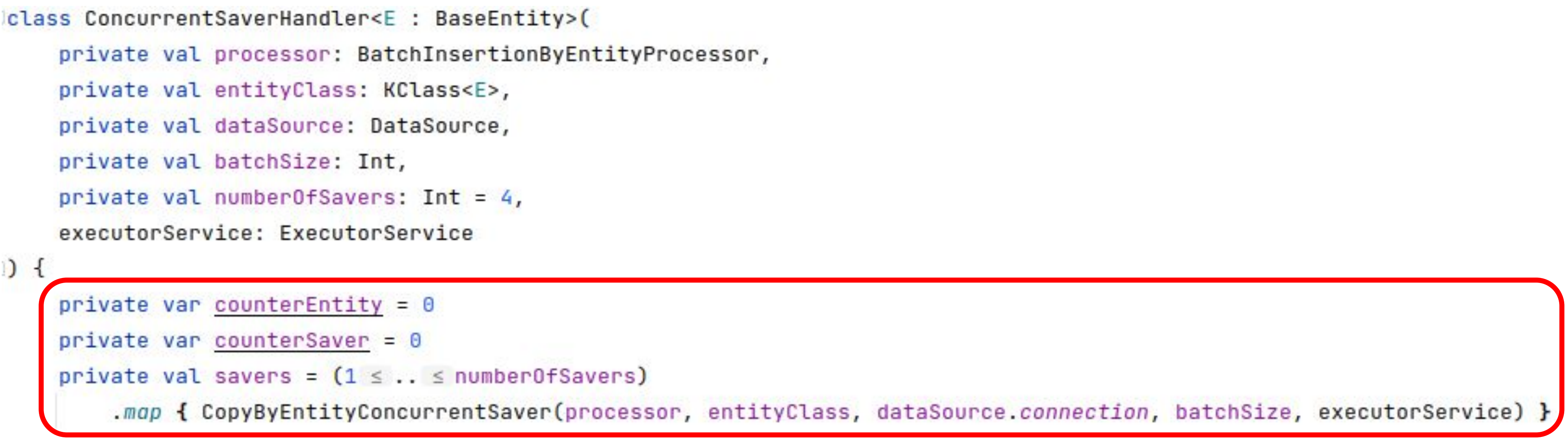
```
class CopyByEntityConcurrentSaver<E : BaseEntity>(
    processor: BatchInsertionByEntityProcessor,
    entityClass: KClass<E>,
    conn: Connection,
    batchSize: Int,
    private val executorService: ExecutorService
) : CopyByEntitySaver<E>(processor, entityClass, conn, batchSize) {

    private var saveDataJob: Future<*>? = null
```

Concurrent saver



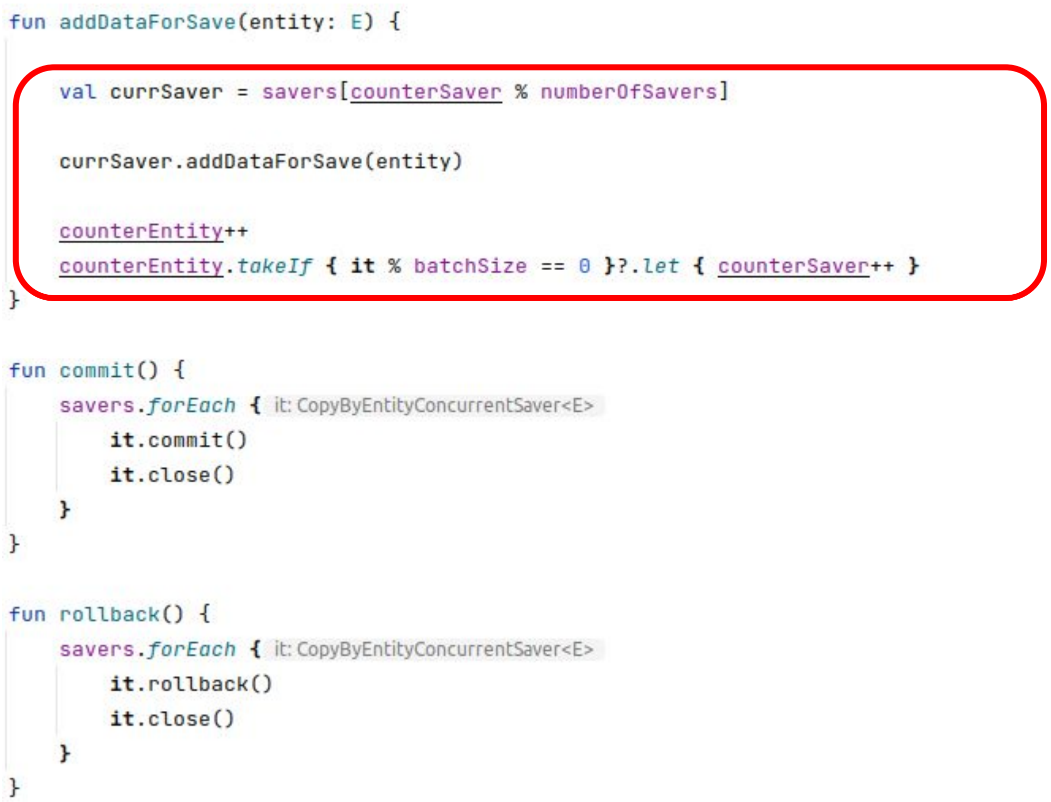
```
override fun addDataForSave(entity: E) {
    checkSaveDataJob()
    super.addDataForSave(entity)
}

override fun saveData() {
    checkSaveDataJob()
    saveDataJob = executorService.submit { super.saveData() }
}

override fun commit() {
    checkSaveDataJob()
    super.saveData()
    conn.commit()
}

private fun checkSaveDataJob() {
    try {
        saveDataJob?.get()
    } catch (e: Exception) {
        throw BatchInsertionException("Can not execute send data to DB", e)
    }
}
```

Concurrent saver handler



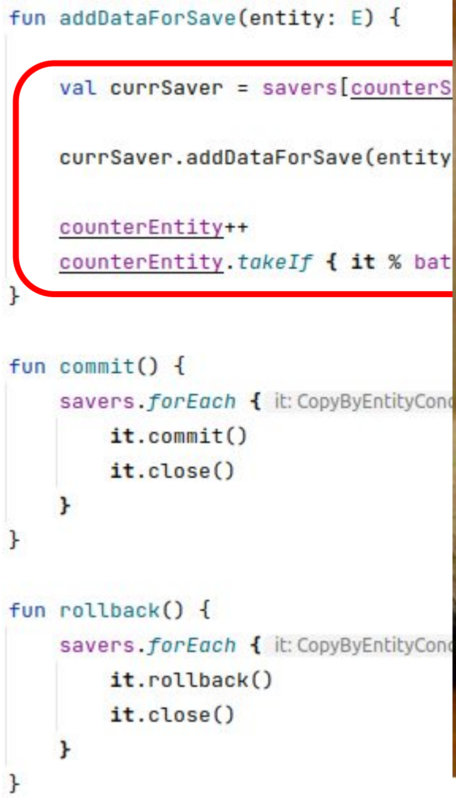
```kotlin
class ConcurrentSaverHandler<E : BaseEntity>(
    private val processor: BatchInsertionByEntityProcessor,
    private val entityClass: KClass<E>,
    private val dataSource: DataSource,
    private val batchSize: Int,
    private val numberOfSavers: Int = 4,
    executorService: ExecutorService
) {
    private var counterEntity = 0
    private var counterSaver = 0
    private val savers = (1 ≤ .. ≤ numberOfSavers)
        .map { CopyByEntityConcurrentSaver(processor, entityClass, dataSource.connection, batchSize, executorService) }
```

Concurrent saver handler



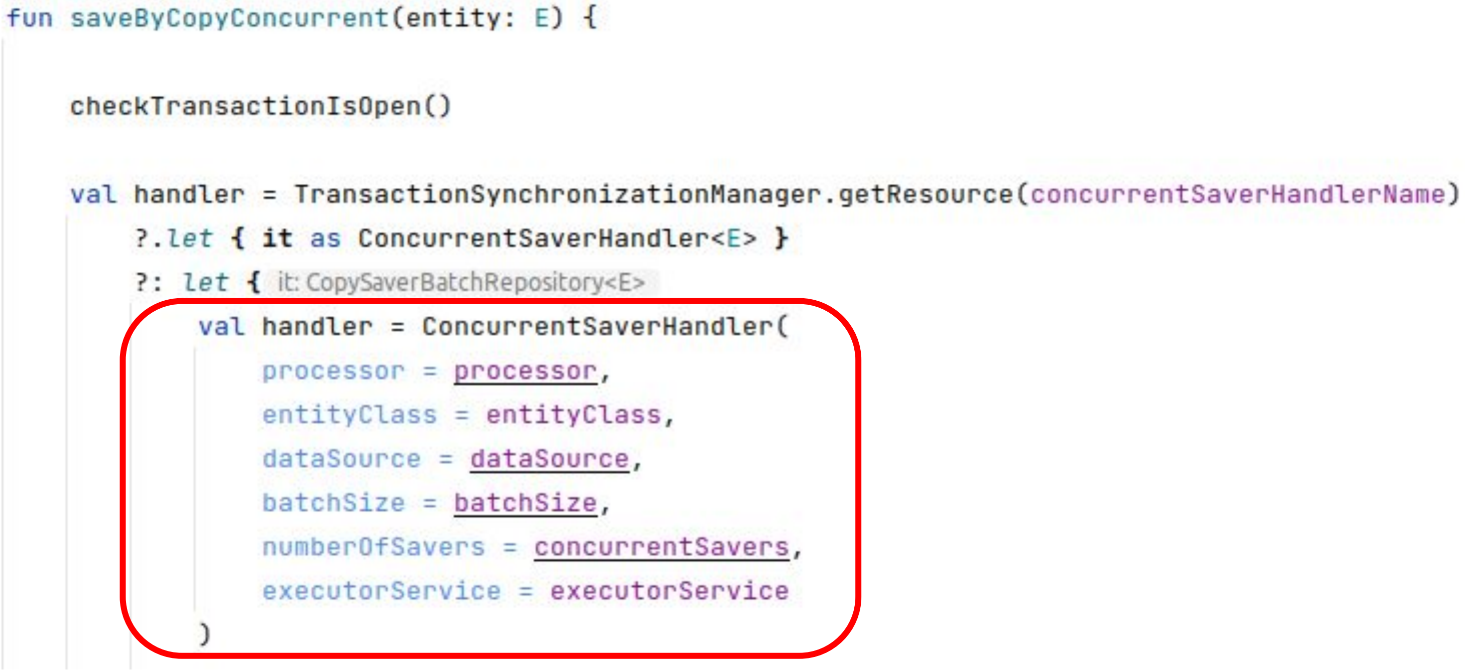
```
fun addDataForSave(entity: E) {

    val currSaver = savers[counterSaver % numberOfSavers]

    currSaver.addDataForSave(entity)

    counterEntity++
    counterEntity.takeIf { it % batchSize == 0 }?.let { counterSaver++ }
}

fun commit() {
    savers.forEach { it: CopyByEntityConcurrentSaver<E>
        it.commit()
        it.close()
    }
}

fun rollback() {
    savers.forEach { it: CopyByEntityConcurrentSaver<E>
        it.rollback()
        it.close()
    }
}
```

Concurrent saver handler



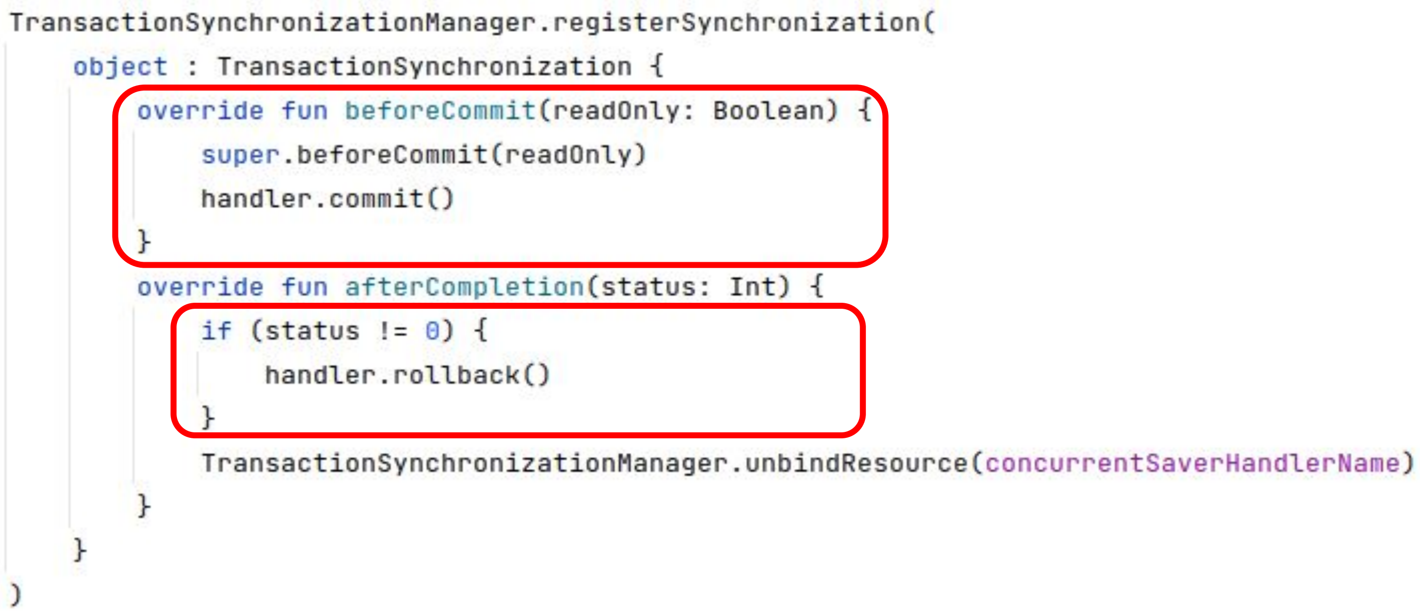
```
fun addDataForSave(entity: E) {

    val currSaver = savers[counterS

    currSaver.addDataForSave(entity

    counterEntity++
    counterEntity.takeIf { it % bat
}

fun commit() {
    savers.forEach { it: CopyByEntityCond
        it.commit()
        it.close()
    }
}

fun rollback() {
    savers.forEach { it: CopyByEntityCond
        it.rollback()
        it.close()
    }
}
```

Save by copy method with concurrent



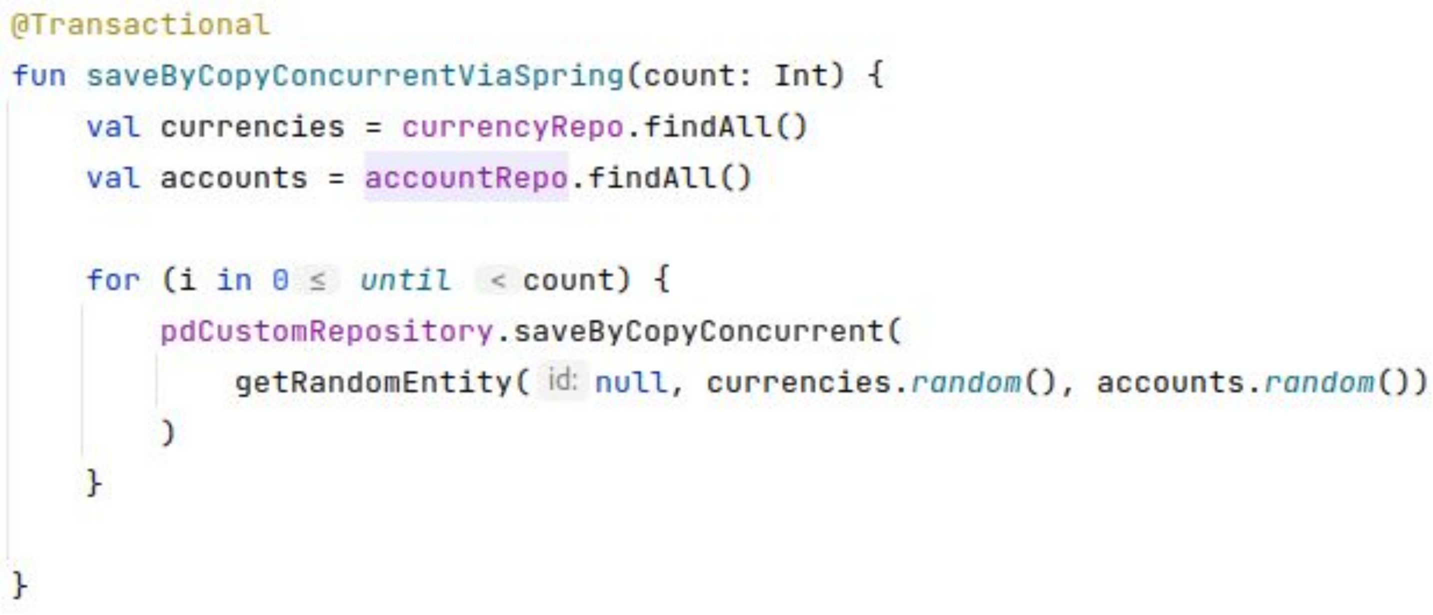
```
fun saveByCopyConcurrent(entity: E) {

    checkTransactionIsOpen()

    val handler = TransactionSynchronizationManager.getResource(concurrentSaverHandlerName)
        ?.let { it as ConcurrentSaverHandler<E> }
        ?: let { it: CopySaverBatchRepository<E>
            val handler = ConcurrentSaverHandler(
                processor = processor,
                entityClass = entityClass,
                dataSource = dataSource,
                batchSize = batchSize,
                numberOfSavers = concurrentSavers,
                executorService = executorService
            )
```

Save by copy method with concurrent



```
TransactionSynchronizationManager.registerSynchronization(
    object : TransactionSynchronization {
        override fun beforeCommit(readOnly: Boolean) {
            super.beforeCommit(readOnly)
            handler.commit()
        }
        override fun afterCompletion(status: Int) {
            if (status != 0) {
                handler.rollback()
            }
            TransactionSynchronizationManager.unbindResource(concurrentSaverHandlerName)
        }
    }
)
```

Save by copy method with concurrent

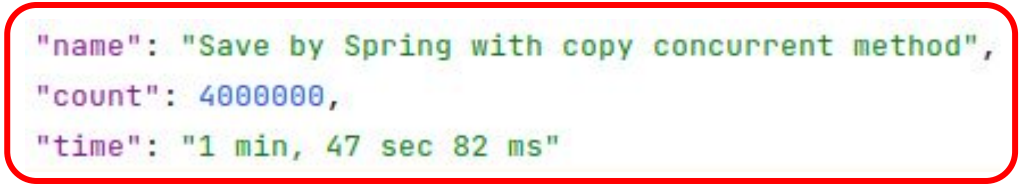


```
@Transactional
fun saveByCopyConcurrentViaSpring(count: Int) {
    val currencies = currencyRepo.findAll()
    val accounts = accountRepo.findAll()

    for (i in 0 ≤ until < count) {
        pdCustomRepository.saveByCopyConcurrent(
            getRandomEntity( id: null, currencies.random(), accounts.random())
        )
    }

}
```

Save by copy method with concurrent

Перейдем к замерам.

Save by copy method with concurrent



```
"name": "Save by Spring with copy concurrent method",
"count": 4000000,
"time": "1 min, 47 sec 82 ms"
```

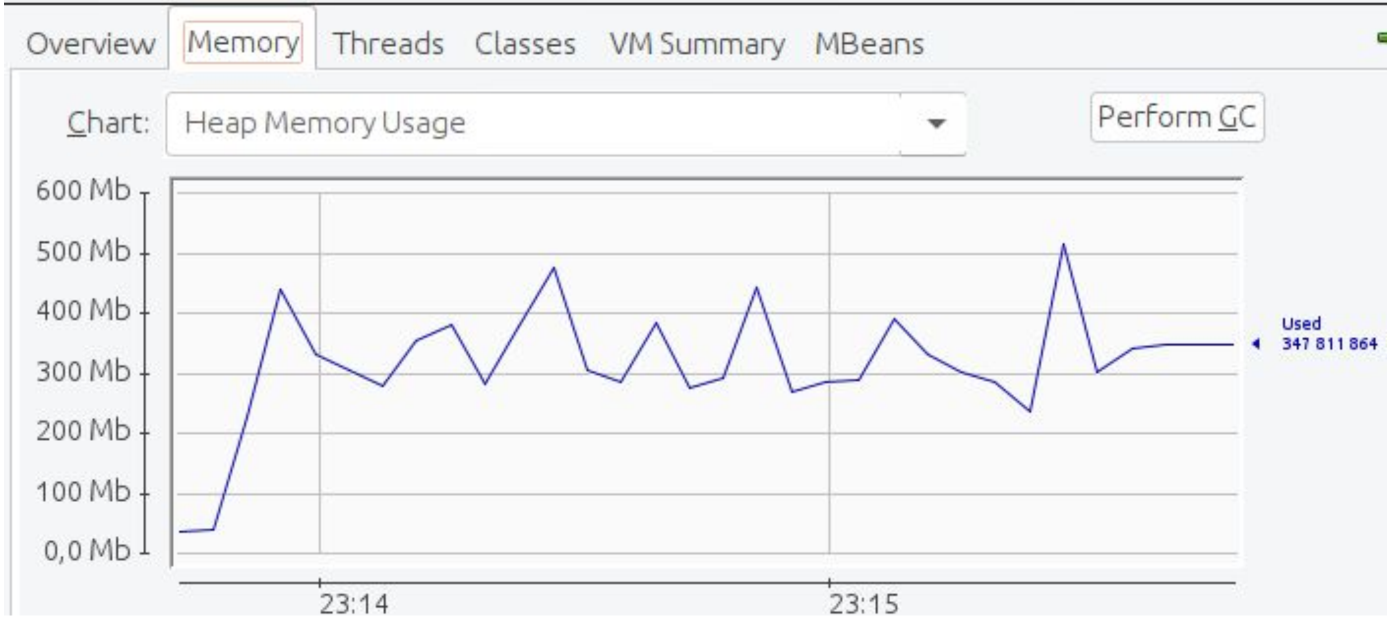


Chart: Heap Memory Usage

Used 347 811 864

Заключение:

- Spring и Hibernate делает свой overhead, можно увеличить скорость вставки данных ~ 40%
- Своя реализация прослойки. Быстрее, можно подружить со Spring, распараллелить процесс вставки, плюшки спринга не работают.
- Copy метод быстрее insert ~ 30%. Работает только с PostgreSQL
- Copy binary vs csv. Сомнительный профит по скорости, возможны проблемы с переходом на новые версии
- Удалось ускорить процесс вставки данных в 6 раз.

Спасибо за внимание!

Телеграм: @FatovDI

Репозиторий:
https://github.com/FatovDI/acceleration-insertion-postgresql-joker2023