NNMeetup 2025

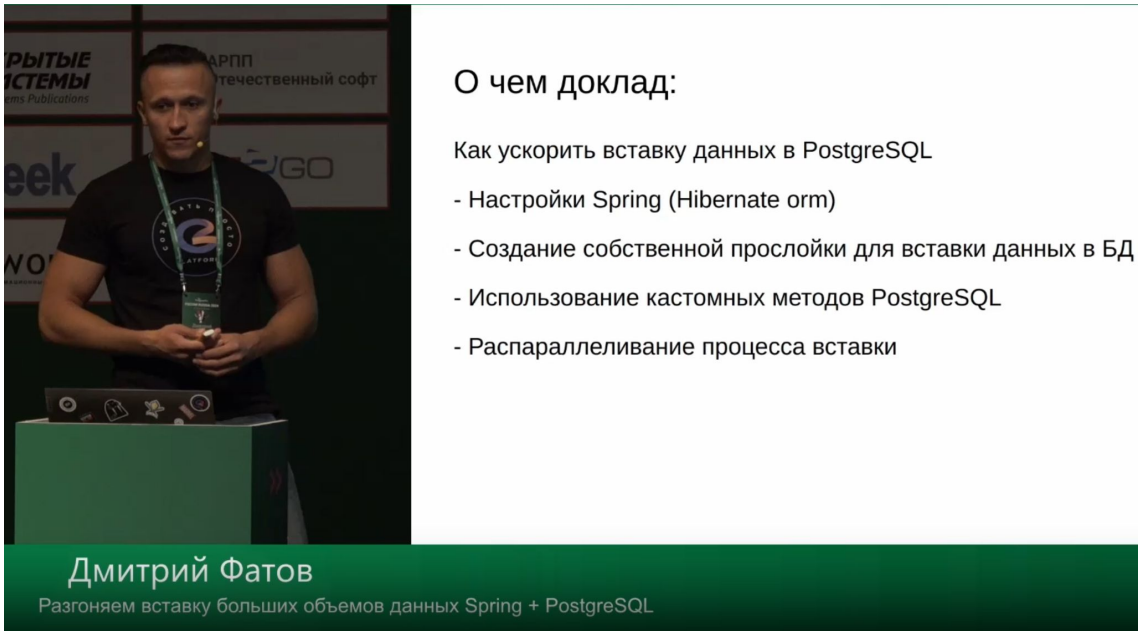
Фатов Дмитрий

Многопоточная вставка данных в БД: от скорости к атомарности. Spring + PostgreSQL

Пару слов о себе:

- Фатов Дмитрий
- Более 13 лет в ИТ
- Пишу на Kotlin
- Работаю в Газпромбанке
- Создаем и строим решения на платформе G2

Предыстория:



https://pgconf.ru/talk/1621867

О чем доклад:

Делаем многопоточную вставку данных в PostgreSQL атомарной

- Как реализовать многопоточную вставку данных в Spring
- Batch update. Как ускорить обновление данных в Hibernate и PostgreSQL
- Признак атомарности в отдельной таблицы. Реализация и Benchmark.
- Возможные проблемы изложенных подходов.
- Патч PostgreSQL позволяющий сохранить атомарность на стороне БД?

Приложение:

- Code: https://github.com/FatovDI/atomic-multithreaded-insertion-postgresql
- 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
- Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 12 ядер, 32gb ОЗУ

Настройки приложения.

```
spring.jpa.properties.hibernate.jdbc.batch_size=5000
```

```
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;
```

2 min 24 sec (23%)

Настройки приложения

```
ALTER SEQUENCE seq_id INCREMENT BY 50;
```

```
@Id
@SequenceGenerator(name = "seq_gen", sequenceName = "seq_id", allocationSize = 50)
```

```
23:08:39.918  INFO 383623 --- [nio-8080-exec-3] c.e.p.l.service.PaymentDocumentService    : start save 99 by Spring
select nextval ('seq_id')
select nextval ('seq_id')
23:08:39.947  INFO 383623 --- [nio-8080-exec-3] c.e.p.l.service.PaymentDocumentService    : end save 99 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,
```

2 min 22 sec (30%)

Настройки приложения

```
spring.datasource.hikari.data-source-properties.reWriteBatchedInserts=true
```

```
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 (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

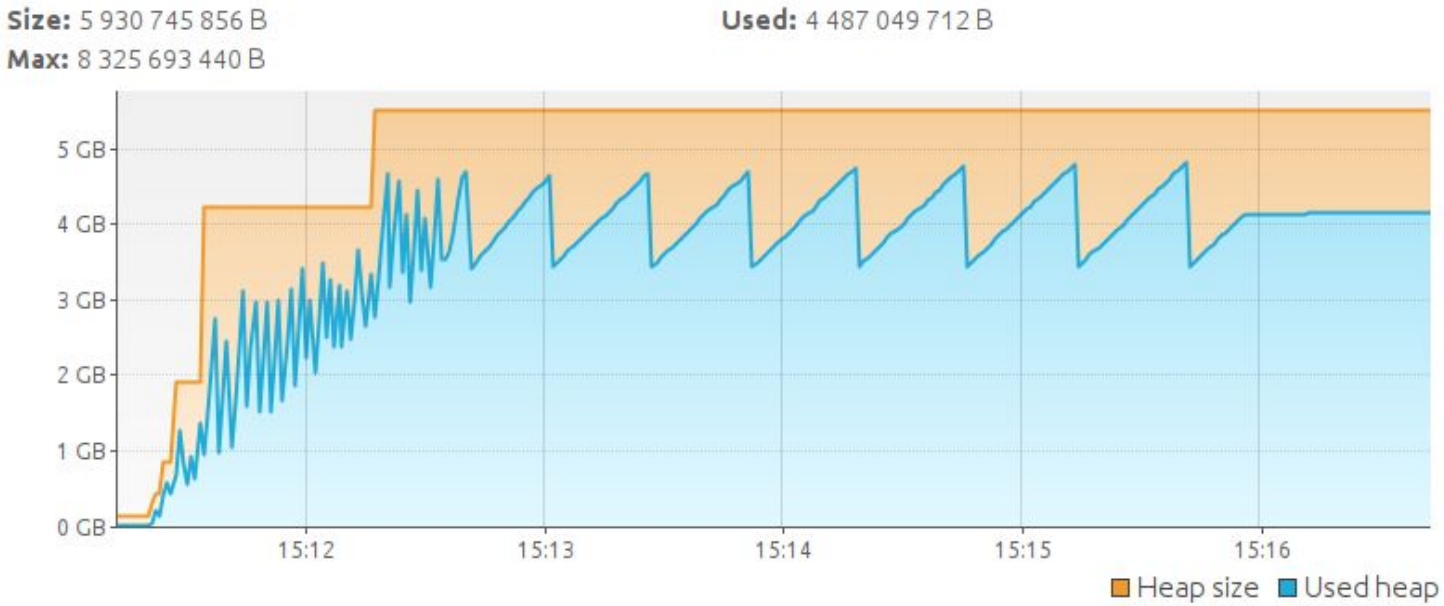


```
2023-09-24 12:00:08.120 UTC [299] LOG:  execute <unnamed>: insert into payment_document (account_id, a
mount, cur, expense, order_date, order_number, payment_purpose, prop_10, prop_15, prop_20, id) values
($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11),($12, $13, $14, $15, $16, $17, $18, $19, $20, $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
```

2 min 24 sec (23%)

Однопоточная вставка данных

```
"name": "Save by Spring",
"count": 4000000,
"time": "4 min, 37 sec 44 ms"
```



Делаем вставку данных асинхронной

Делаем вставку данных асинхронной

```
@Configuration
@EnableAsync
class SpringAsyncConfig {

    @Value("8")
    private var poolSize: Int = 8

    @Bean(name = ["threadPoolAsyncInsertExecutor"])
    fun threadPoolAsyncInsertExecutor(): Executor {
        return Executors.newFixedThreadPool(poolSize)
    }

}
```

Делаем вставку данных асинхронной

```
@Component
class PaymentDocumentSaver(
    val paymentDocumentRepository: PaymentDocumentRepository,
) {

    @Async("threadPoolAsyncInsertExecutor")
    fun saveBatchAsync(entities: List<PaymentDocumentEntity>): Future<List<PaymentDocumentEntity>> {
        val savedEntities = paymentDocumentRepository.saveAll(entities)
        return CompletableFuture.completedFuture(savedEntities)
    }

}
```

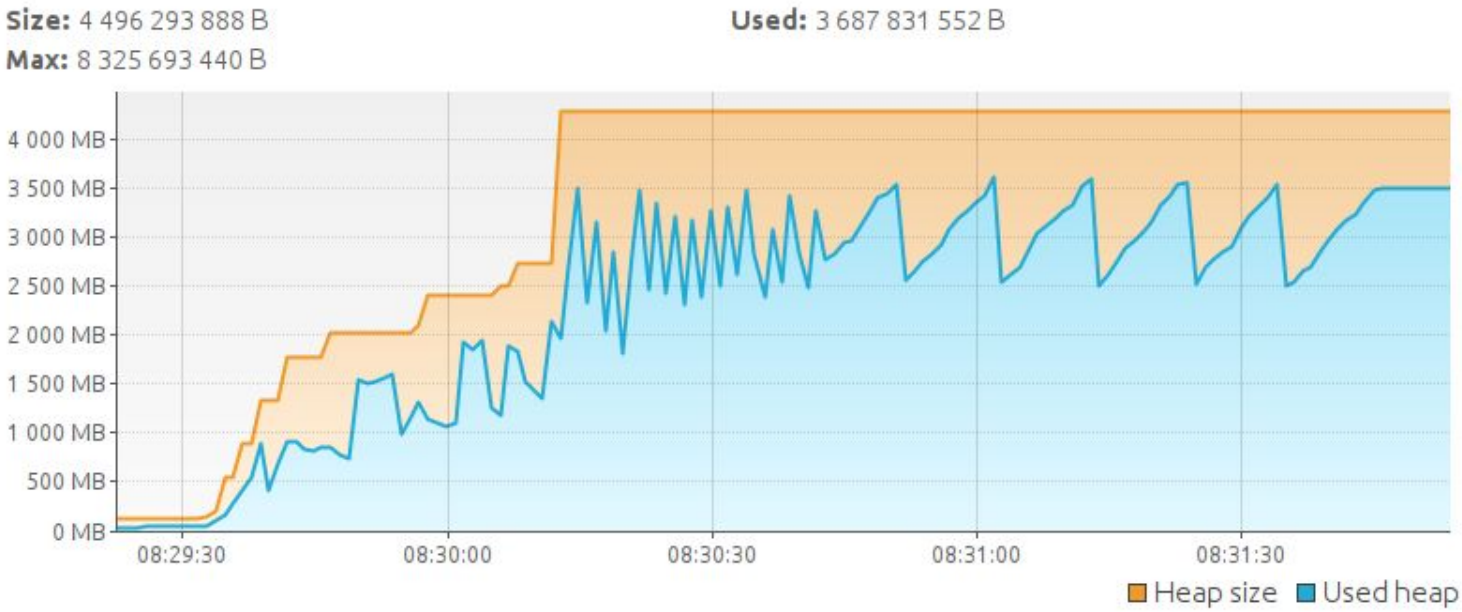
Делаем вставку данных асинхронной

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

    var listForSave = mutableListOf<PaymentDocumentEntity>()
    val saveTasks = mutableListOf<Future<List<PaymentDocumentEntity>>>()
    (1 ≤ .. ≤ count).forEach { it: Int
        listForSave.add(getRandomEntity( id: null, currencies.random(), accounts.random()))
        if (it != 0 && it % batchSize == 0) {
            saveTasks.add(saver.saveBatchAsync(listForSave))
            listForSave = mutableListOf()
        }
    }
    listForSave.takeIf { it.isNotEmpty() }?.let { saveTasks.add(saver.saveBatchAsync(it)) }
    saveTasks.forEach { it.get() }
}
```

Многопоточная вставка данных

```
"name": "Save by spring with async",
"count": 4000000,
"time": "1 min, 56 sec 179 ms"
```

≈ 2 min 41 sec (58%)



Как сделать вставку атомарной?

Атомарность через дополнительный флаг

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@Where(clause = "ready_to_read = true")
abstract class BaseAsyncInsertEntity : BaseEntity() {

    var readyToRead: Boolean = true

}
```

```
Hibernate: select paymentdoc0_.id as id1_1_, paymentdoc0_.ready_to_read as ready_to2_1_,
  amount1_3_, paymentdoc0_.cur as cur10_3_, paymentdoc0_.expense as expense2_3_, paymentdo
  order_nu4_3_, paymentdoc0_.payment_purpose as payment_5_3_, paymentdoc0_.prop_10 as prop
  prop_8_3_ from payment_document paymentdoc0_ where ( paymentdoc0_.ready_to_read = true)
```

Атомарность через дополнительный флаг

```
(1 ≤ .. ≤ count).forEach { it: Int
    listForSave.add(
        getRandomEntity( id: null, currencies.random(), accounts.random()) apply { readyToRead = false }
    )
    if (it != 0 && it % batchSize == 0) {
        saveTasks.add(saver.saveBatchAsync(listForSave))
        listForSave = mutableListOf()
    }
}
listForSave.takeIf { it.isNotEmpty() }?.let { saveTasks.add(saver.saveBatchAsync(it)) }

val savedPD = saveTasks.flatMap { it.get() }
savedPD.forEach { it.readyToRead = true }
repository.saveAll(savedPD)
```

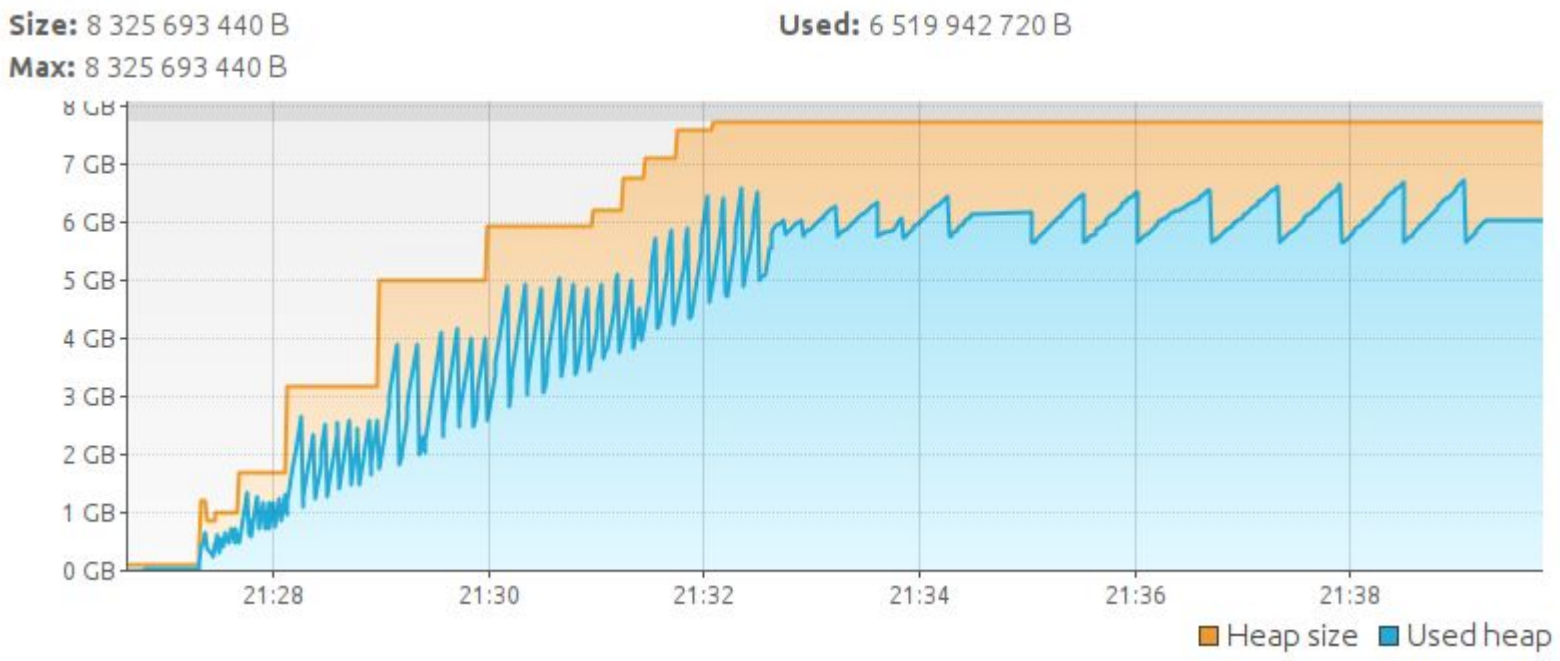
Обновление это быстро?

Обновление через JpaRepository

```
@Transactional
fun updateBySpring(count: Int) {
    val listId = sqlHelper.getIdListForUpdate(count, PaymentDocumentEntity::class)
    val currencies = currencyRepo.findAll()
    val accounts = accountRepo.findAll()

    for (i in 0 ≤ until < count) {
        repository.save(getRandomEntity(listId[i], currencies.random(), accounts.random()))
    }
}
```

Обновление через JpaRepository

```
"name": "Update via spring",
"count": 4000000,
"time": "10 min, 21 sec 348 ms"
```

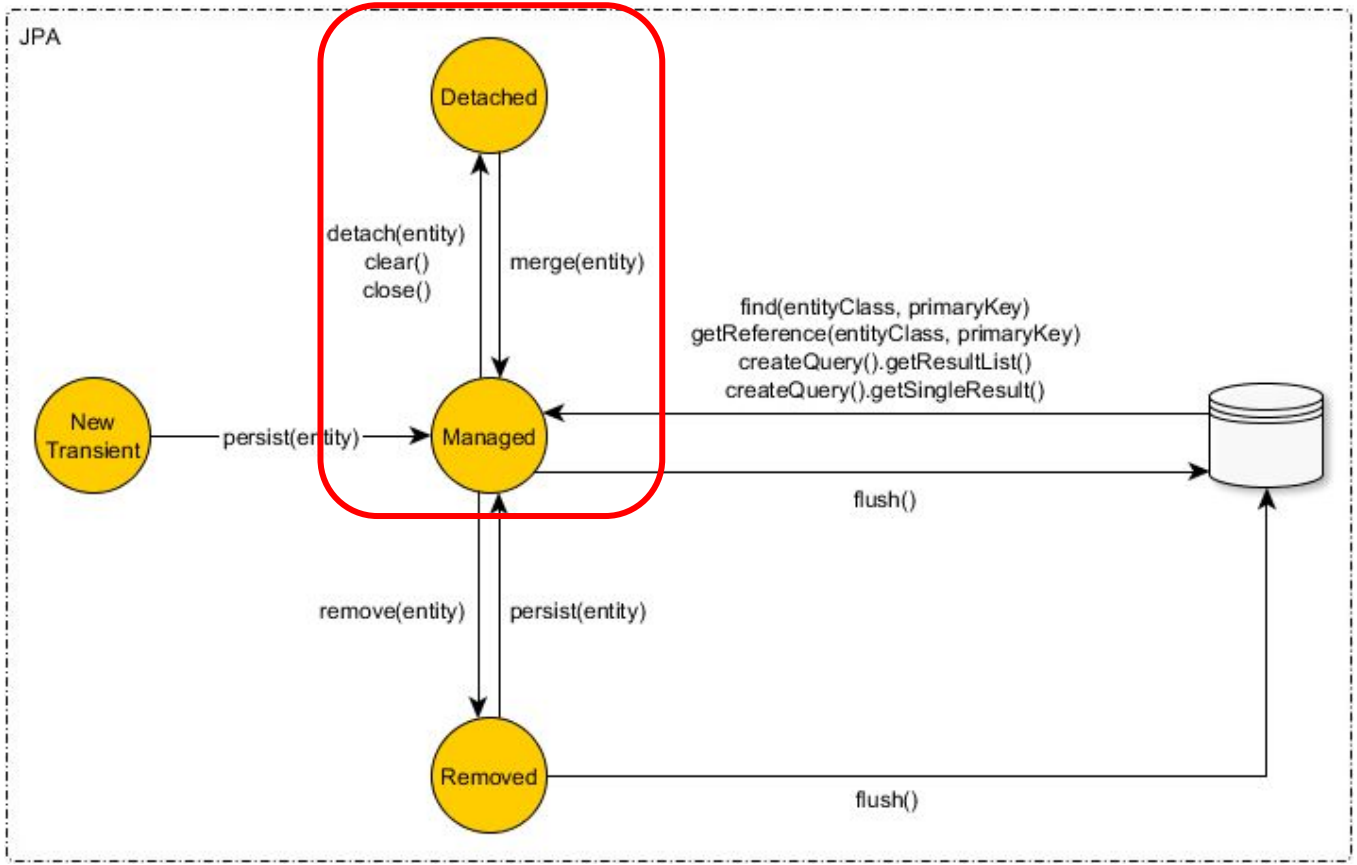


Обновление чер

Обновление через JpaRepository

```
Hibernate: select paymentdoc0_.id as id1_2_0_, paymentdoc0_.account_id as account10_2_0_, paymentd
  .order_date as order_da4_2_0_, paymentdoc0_.order_number as order_nu5_2_0_, paymentdoc0_.payment
  .prop_20 as prop_9_2_0_ from payment_document paymentdoc0_ where paymentdoc0_.id=?
2023-09-21 23:33:37.901  INFO 156134 --- [nio-8080-exec-2] c.e.p.l.service.PaymentDocumentService
Hibernate: update payment_document set account_id=?, amount=?, cur=?, expense=?, order_date=?, ord
```

Обновление через JpaRepository



Обновление через JpaRepository

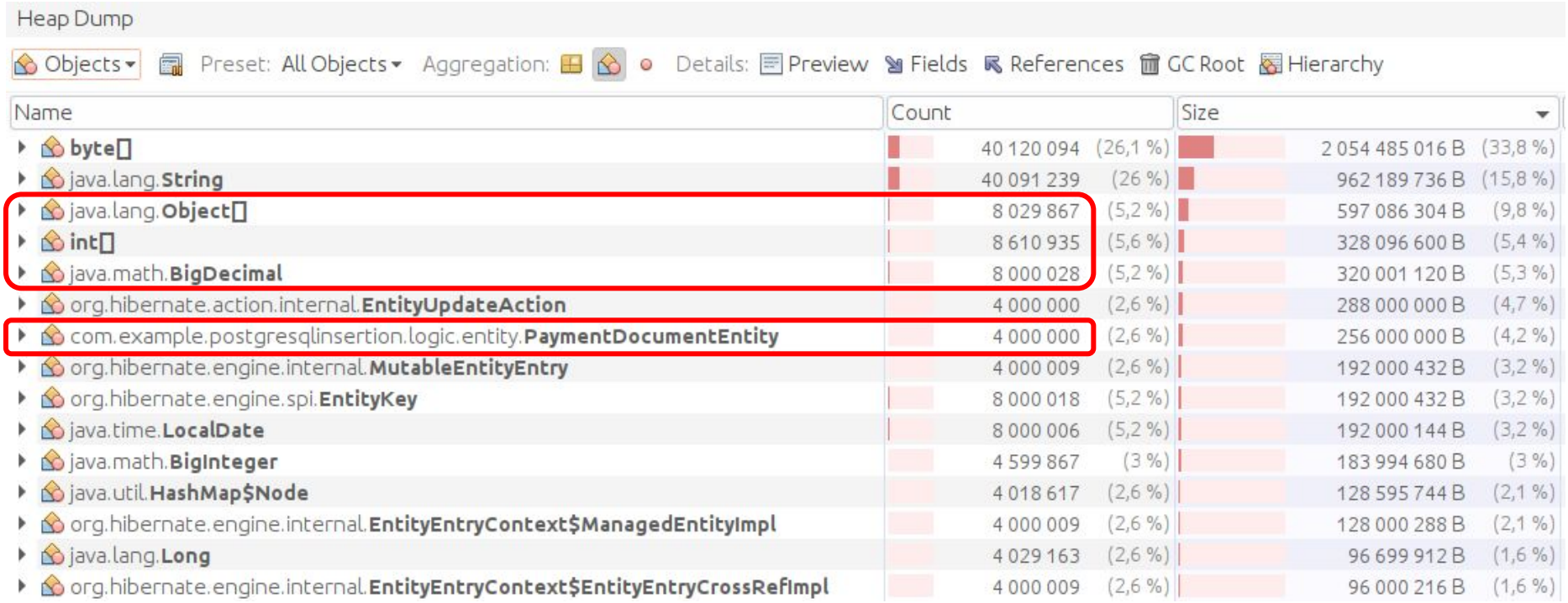
```
public <S extends T> S save(S entity) {

    Assert.notNull(entity, message: "Entity must not be null.");

    if (entityInformation.isNew(entity)) {
        em.persist(entity);
        return entity;
    } else {
        return em.merge(entity);
    }
}
```

Метод entityIsDetached класс DefaultMergeEventListener

```
String previousFetchProfile = source.getLoadQueryInfluencers().getInternalFetchProfil
source.getLoadQueryInfluencers().setInternalFetchProfile( "merge" );

//we must clone embedded composite identifiers, or
//we will get back the same instance that we pass in
final Serializable clonedIdentifier = (Serializable) persister.getIdentifierType().de
final Object result = source.get( entityName, clonedIdentifier );

source.getLoadQueryInfluencers().setInternalFetchProfile( previousFetchProfile );
```

Почему потребление памяти X2?

Update через JpaRepository:

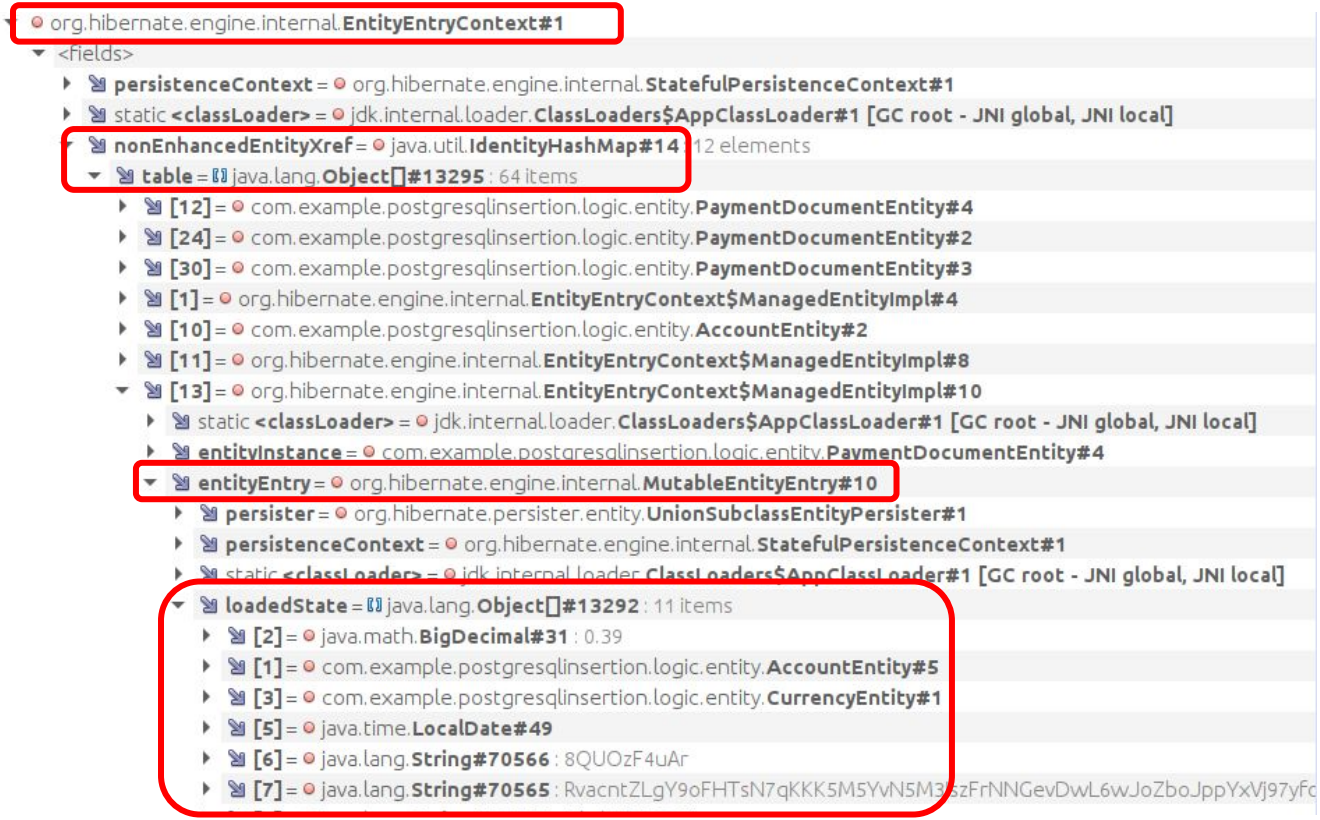


Name	Count	Size
byte[]	40 120 094 (26,1 %)	2 054 485 016 B (33,8 %)
java.lang.String	40 091 239 (26 %)	962 189 736 B (15,8 %)
java.lang.Object[]	8 029 867 (5,2 %)	597 086 304 B (9,8 %)
int[]	8 610 935 (5,6 %)	328 096 600 B (5,4 %)
java.math.BigDecimal	8 000 028 (5,2 %)	320 001 120 B (5,3 %)
org.hibernate.action.internal.EntityUpdateAction	4 000 000 (2,6 %)	288 000 000 B (4,7 %)
com.example.postgresqlinsertion.logic.entity.PaymentDocumentEntity	4 000 000 (2,6 %)	256 000 000 B (4,2 %)
org.hibernate.engine.internal.MutableEntityEntry	4 000 009 (2,6 %)	192 000 432 B (3,2 %)
org.hibernate.engine.spi.EntityKey	8 000 018 (5,2 %)	192 000 432 B (3,2 %)
java.time.LocalDate	8 000 006 (5,2 %)	192 000 144 B (3,2 %)
java.math.BigInteger	4 599 867 (3 %)	183 994 680 B (3 %)
java.util.HashMap$Node	4 018 617 (2,6 %)	128 595 744 B (2,1 %)
org.hibernate.engine.internal.EntityEntryContext$ManagedEntityImpl	4 000 009 (2,6 %)	128 000 288 B (2,1 %)
java.lang.Long	4 029 163 (2,6 %)	96 699 912 B (1,6 %)
org.hibernate.engine.internal.EntityEntryContext$EntityEntryCrossRefImpl	4 000 009 (2,6 %)	96 000 216 B (1,6 %)

Метод entityIsDetached класс DefaultMergeEventListener

```
    // cascade first, so that all unsaved objects get their
    // copy created before we actually copy
    cascadeOnMerge( source, persister, entity, copyCache );
    copyValues( persister, entity, target, source, copyCache );

    //copyValues works by reflection, so explicitly mark the entity instance dirty
    markInterceptorDirty( entity, target, persister );

    event.setResult( result );
}
```

Update через JpaRepository:



Как избавиться от select при update

Update через Session:

```
fun batchUpdateBySession(entities: List<PaymentDocumentEntity>): List<PaymentDocumentEntity> {
    sessionFactory.openStatelessSession().use { session ->
        val transaction = session.beginTransaction()
        try {
            entities.forEach { session.update(it) }
            transaction.commit()
        } catch (e: Exception) {
            transaction.rollback()
            throw e
        }
    }
    return entities
}
```

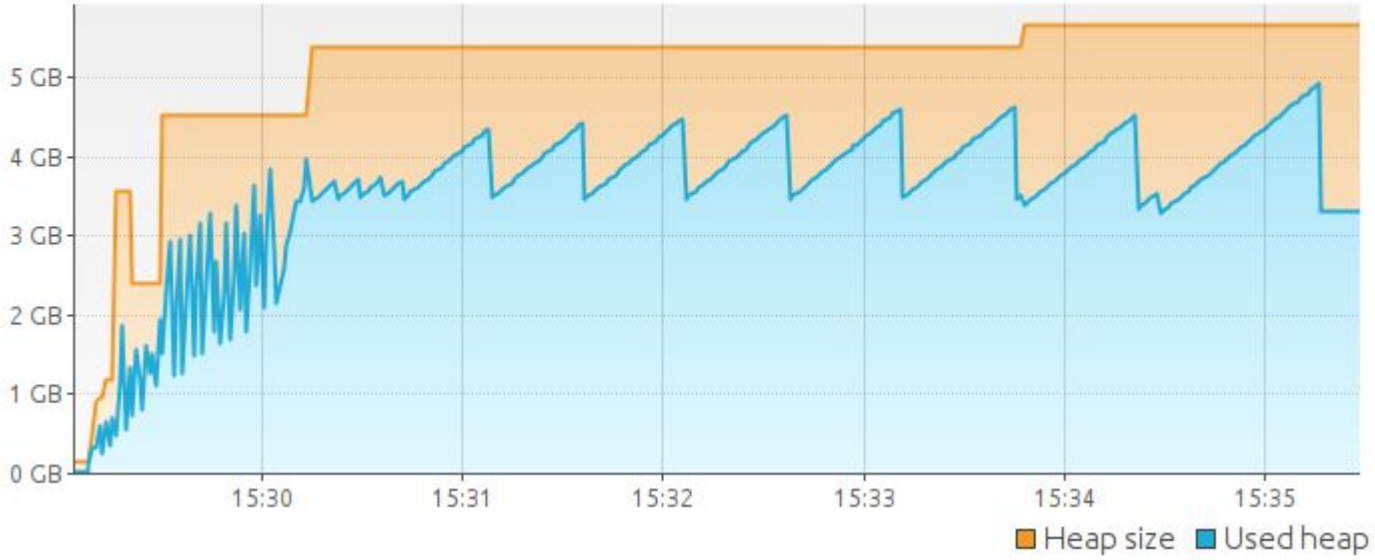
```
Hibernate: update payment_document set account_id=?, amount=?, cur=?, expense=?, orde
Hibernate: update payment_document set account_id=?, amount=?, cur=?, expense=?, orde
Hibernate: update payment_document set account_id=?, amount=?, cur=?, expense=?, orde
```

Update через Session:

```
"name": "Update via spring by session",
"count": 4000000,
"time": "5 min, 59 sec 573 ms"
```

≈ 5 min 12 sec (49%)

Size: 6 098 518 016 B
Max: 8 325 693 440 B
Used: 3 588 832 560 B



Изменение только одного поля. JdbcTemplate

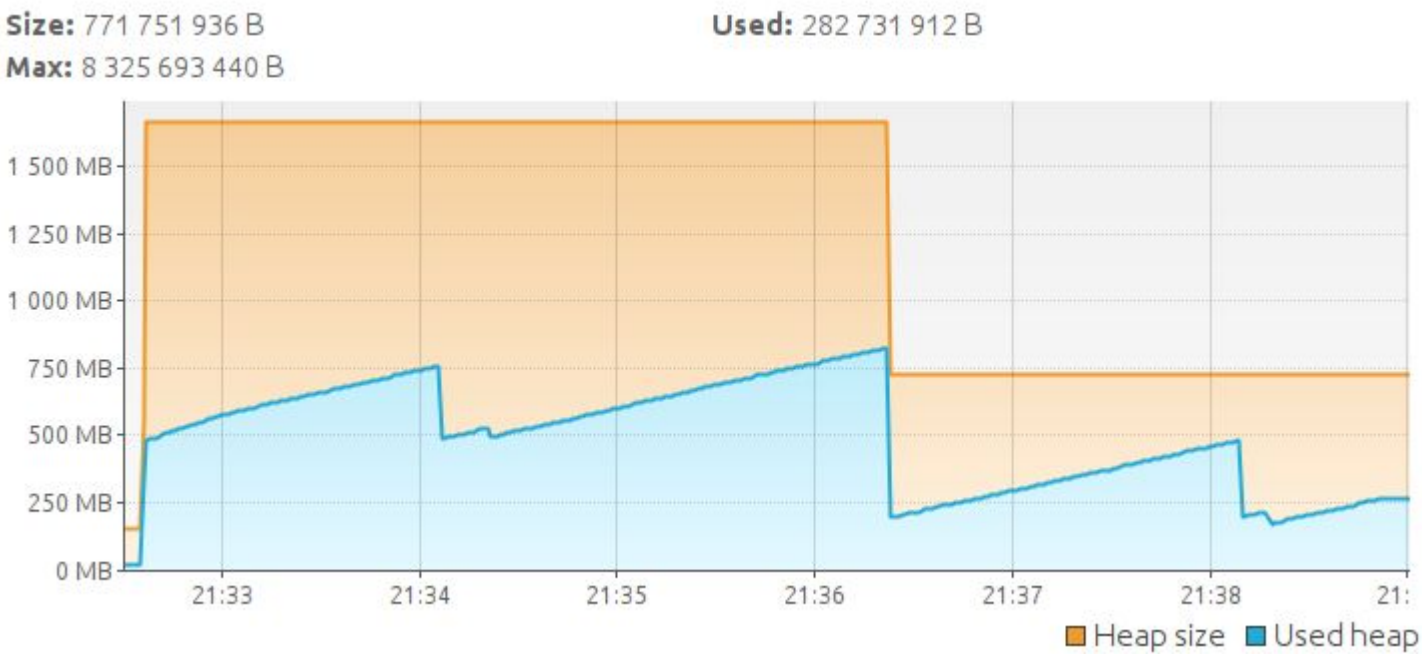
```
private var batchSize: Int = 5000

private val jdbcTemplate = JdbcTemplate(dataSource)

fun setReadyToRead(idList: List<Long>) {
    jdbcTemplate.batchUpdate(
        sql: "update payment_document set ready_to_read = true where id = ?",
        idList, batchSize) {  ps, argument ->
        ps.setLong( parameterIndex: 1, argument)
    }
}
```

```
: Executing prepared SQL statement [update payment_document set ready_to_read = true where id = ?]
: Sending SQL batch update #1 with 5000 items
: Sending SQL batch update #2 with 5000 items
```

Изменение только одного поля. JdbcTemplate

```
"name": "Set ready to read jdbcTemplate",
"count": 4000000,
"time": "5 min, 51 sec 255 ms"
```

≈ 8 sec (2%)



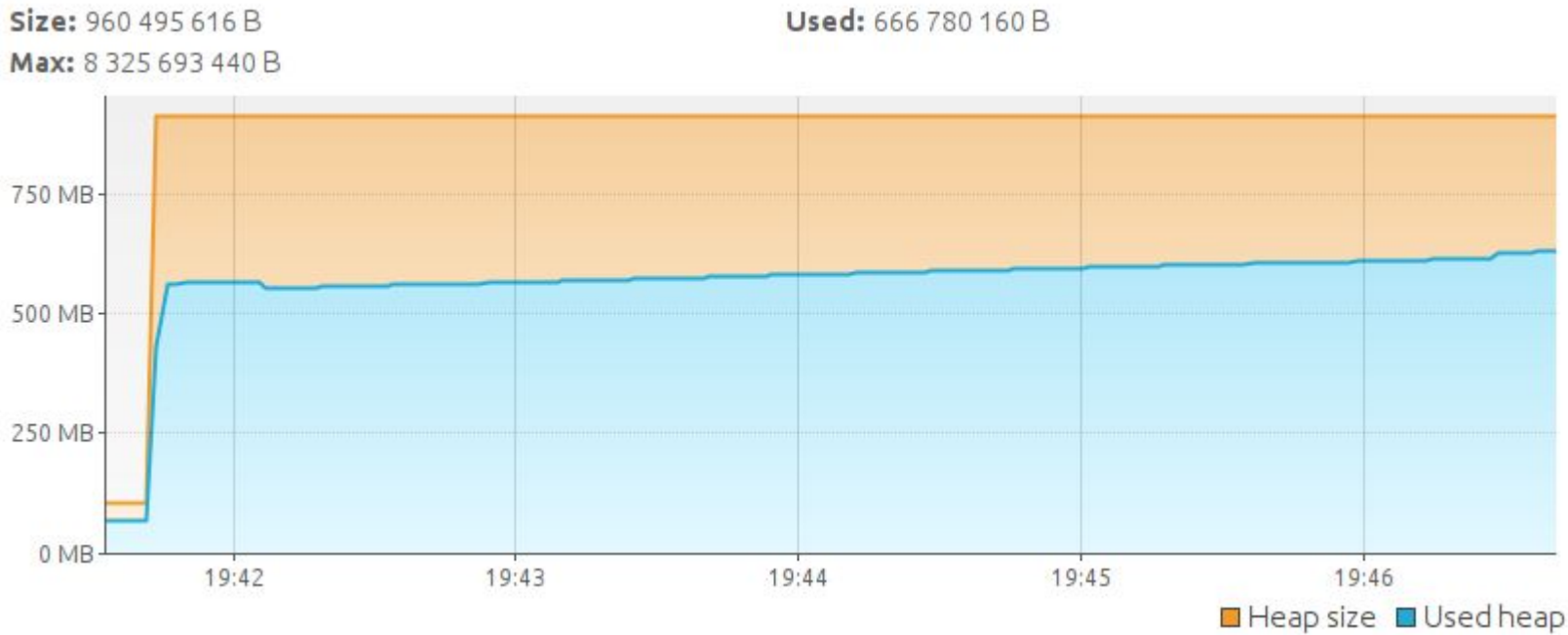
Изменение только одного поля. Any([])

```
fun setReadyToReadArray(idList: List<Long>): Int {
    return jdbcTemplate.update(
        sql: "update payment_document set ready_to_read = true where id = any (?)"
    ) {  ps ->
        ps.setObject( parameterIndex: 1, idList.toTypedArray())
    }
}
```

Изменение только одного поля. Any([])

```
"name": "Set ready to read array",
"count": 4000000,
"time": "4 min, 5 sec 116 ms"
```

≈ 1 min 46 sec (30%)



Добавление идентификатора транзакции

```
ALTER TABLE PAYMENT_DOCUMENT ADD COLUMN transaction_id uuid DEFAULT null;
CREATE INDEX IX_PAYMENT_DOCUMENT_transaction_id
    on PAYMENT_DOCUMENT (transaction_id) where payment_document.transaction_id is not null;
```

```
fun removeTransactionId(transactionId: UUID): Int {
    return jdbcTemplate.update(
        sql: "update payment_document set transaction_id = null where transaction_id = ?"
    ) {  ps ->
        ps.setObject( parameterIndex: 1, transactionId)
    }
}
```

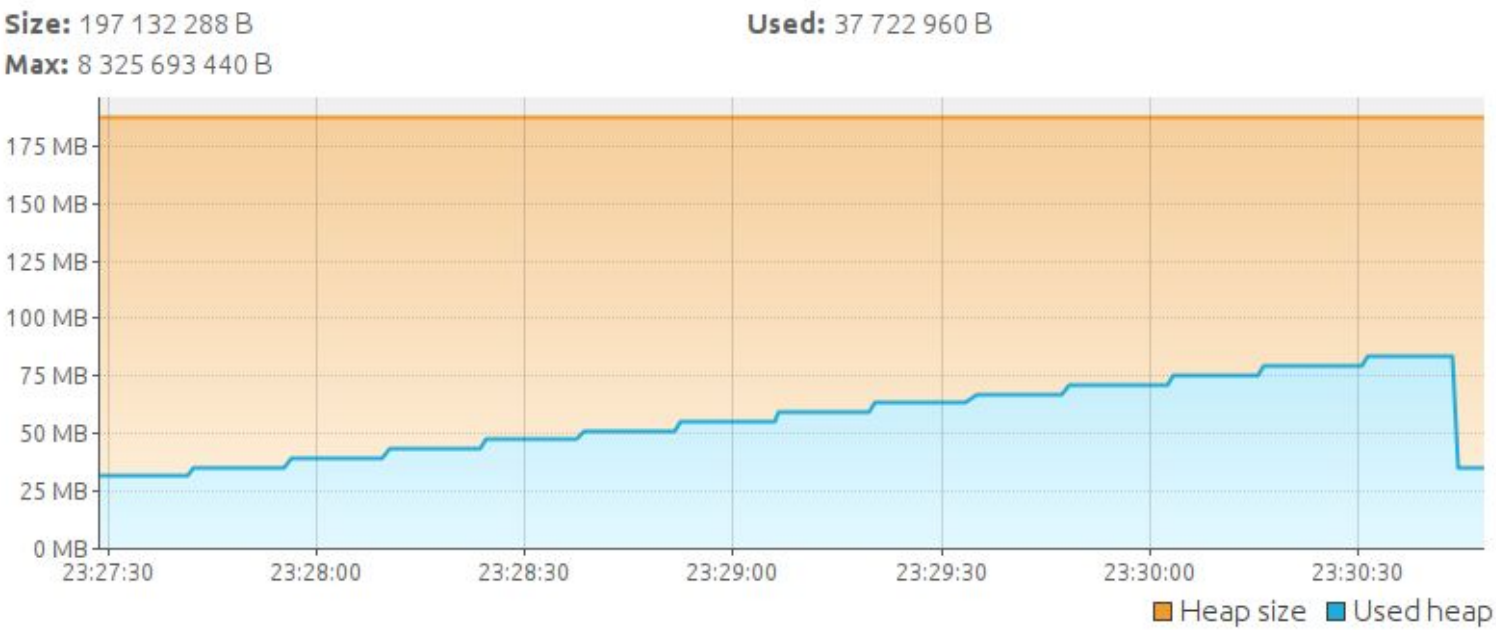
Добавление идентификатора транзакции

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@Where(clause = "transaction_id is null")
abstract class BaseAsyncInsertEntity : BaseEntity() {

    var transactionId: UUID? = null

}
```

Set transaction id = null

```
"name": "Remove transaction id after insert"
"count": 4000000,
"time": "2 min, 50 sec 498 ms"
```

≈ 1 min 15 sec (30%)



Update в PostgreSQL это быстро?

Как работает PostgreSQL?

Рогов Е. В.

PostgreSQL 17 изнутри. — М.: ДМК Пресс, 2025. — 668 с.

ISBN 978-5-93700-372-0

https://postgrespro.ru/education/books/internals

Как работает PostgreSQL?

- MVCC (Multiversion Concurrency Control) Многоверсионное управление конкурентным доступом
- Заголовки строк (24 байта) содержат информацию о версиях.
- Посмотреть данные страницы индексов и данных:

```
CREATE EXTENSION pageinspect;

select * from heap_page_items(get_raw_page('payment_document', 0));

select * from bt_page_items('pk_payment_document', 1);
```

Обновление индексного поля.

```
update payment document set transaction id = ? where id = 1000010;
update payment_document set transaction_id = ? where id = 1000010;
```

Страница с индексом ПК

itemoffset	htid
1	(0,1)
2	(0,2)
3	(0,3)

Страница с данными

lp	t_xmin	t_xmax
1	8242139	8242140
2	8242140	8242141
3	8242141	0

Ссылка на данные в таблице (индекс страницы, номер строки)

Номер строки

Транзакция создавшая строку

Транзакция изменившая строку

Оптимизация HOT (Heap-Only Tuple) обновления

Позволяет:

- Не создавать дополнительные ссылки индексов на строки данных

Применим когда:

- На странице есть место для новой строки
- На обновляемых столбцах нет индексов
- Обновляемые столбцы есть в индексах, но их значения не меняются.

HOT (Heap-Only Tuple) обновления.

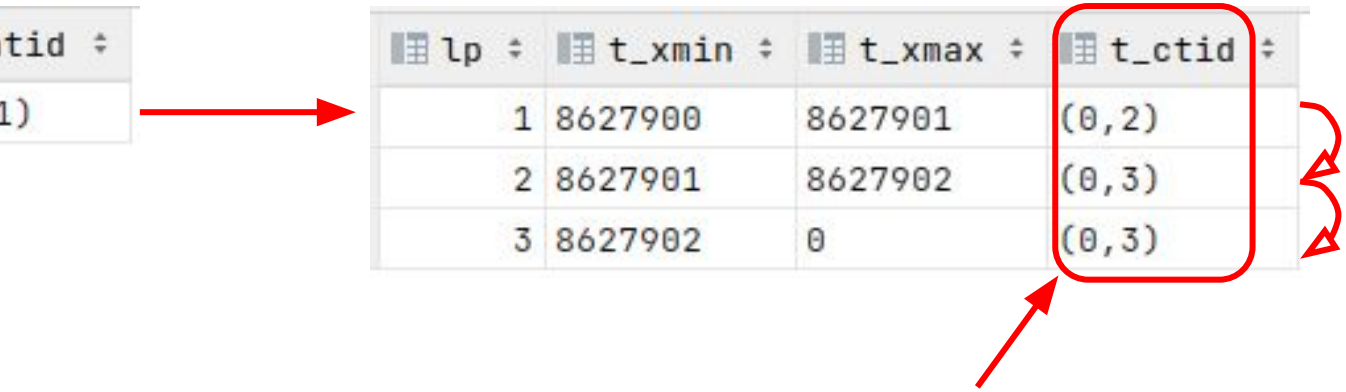
```
update payment document  set ready to read = true  where id = 1000010;
update payment_document  set ready_to_read = false where id = 1000010;
```

Страница с индексом ПК

itemoffset	htid
1	(0,1)

Страница с данными

lp	t_xmin	t_xmax	t_ctid
1	8627900	8627901	(0,2)
2	8627901	8627902	(0,3)
3	8627902	0	(0,3)



Ссылка на следующую строку в цепочке

Наши обновления были HOT?

Оптимизация HOT (Heap-Only Tuple) обновления

Позволяет:

- Не создавать дополнительные ссылки индексов на строки данных

Применим когда:

- На странице есть место для новой строки
- На обновляемых столбцах нет индексов
- Обновляемые столбцы есть в индексах, но их значения не меняются.

Оптимизация HOT (Heap-Only Tuple) обновления

```
select * from page_header(get_raw_page('payment_document', 0));
```

lower	upper	pagesize
160	304	8192

304 - 160 = 144 (свободное место на странице)

lp	lp_len	t_xmin	t_xmax
1	225	8627900	8627901
2	225	8627901	8627902
3	225	8627902	0

lp_len - размер размер строки вместе с заголовком = 225 байт

Как оставить место под обновления?

Fillfactor для HOT обновлений

- Fillfactor - процент наполнения страниц при вставке данных
- При достижении указанного порога вставка осуществляется на новую страницу
- Не заполненное пространство остается под HOT update
- Установить можно следующим способом:

```
alter table payment_document set (fillfactor = 80);
vacuum full payment_document;
```

Set ready to read by transaction id.

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@Where(clause = "ready_to_read = true")
abstract class BaseAsyncInsertEntity : BaseEntity() {

    var transactionId: UUID? = null
    var readyToRead: Boolean = true

}

fun setReadyToRead(transactionId: UUID): Int {
    return jdbcTemplate.update(
        sql: "update payment_document set ready_to_read = true  where transaction_id = ?") { ps ->
        ps.setObject( parameterIndex: 1, transactionId)
    }
}
```

Fillfactor для HOT обновлений

Обновление 4 млн. строк

Fillfactor, %	Время обновления	Размер таблицы, Gb*
100	2 min, 50 sec 262 ms	31
90	2 min, 47 sec 729 ms	34
80	2 min, 27 sec 284 ms	37
70	1 min, 55 sec 763 ms	41
60	1 min, 23 sec 521 ms	48
50	**0 min, 16 sec 454 ms**	58

* `SELECT pg_size_pretty(pg_total_relation_size('test_insertion.payment_document' ));`

Fillfactor = 50. Update 4M rows by id vs transaction_id

```
update payment_document
    set ready_to_read = true
where id = any (?)
```

VS

```
update payment_document
    set ready_to_read = true
where transaction_id = ?
```

1 min, 11 sec 646 ms

0 min, 16 sec 454 ms

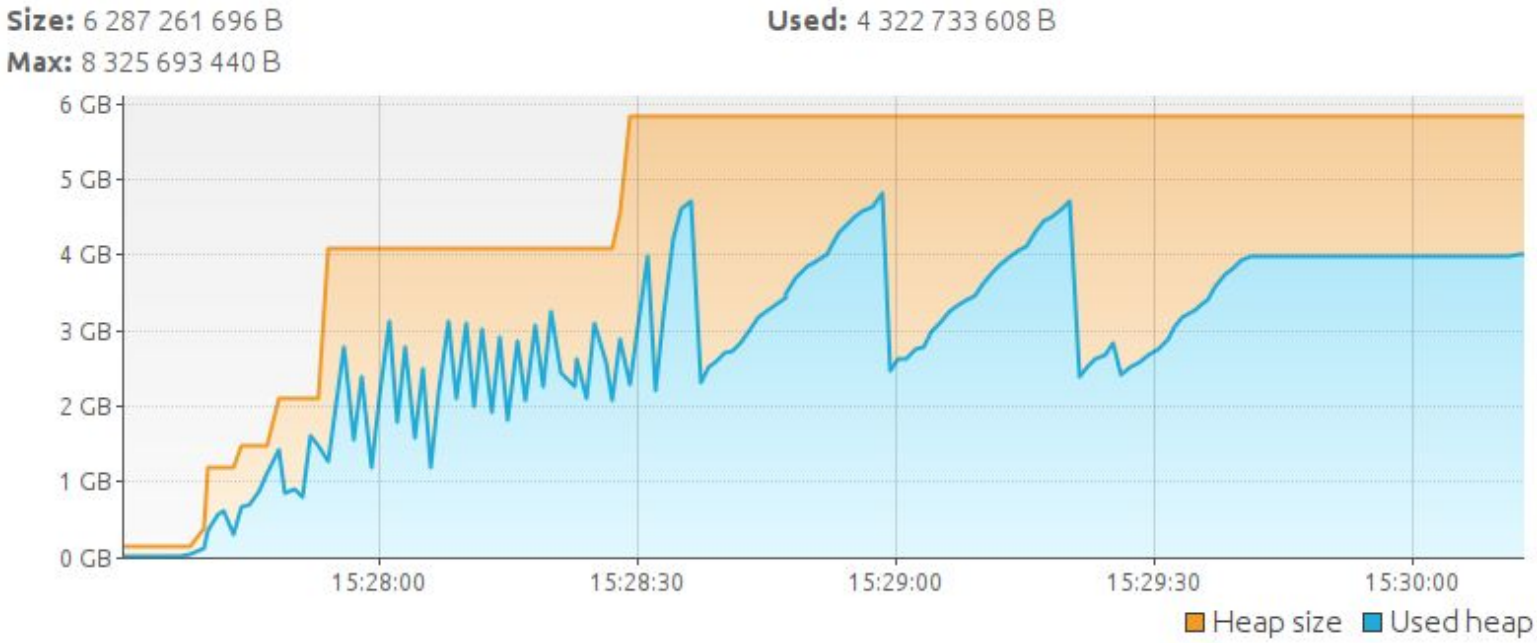
Атомарная вставка с fillfactor = 50

```
val transactionId = Generators.timeBasedEpochGenerator().generate()
(1 ≤ .. ≤ count).forEach { it: Int
    listForSave.add(
        getRandomEntity( id: null, currencies.random(), accounts.random(), transactionId)
            .apply { readyToRead = false }
    )
    if (it != 0 && it % batchSize == 0) {
        saveTasks.add(saver.saveBatchAsync(listForSave))
        listForSave = mutableListOf()
    }
}
listForSave.takeIf { it.isNotEmpty() }?.let { saveTasks.add(saver.saveBatchAsync(it)) }

saveTasks.flatMap { it.get() }
saver.setReadyToRead(transactionId)
```

Атомарная вставка с fillfactor = 50

```
"name": "save concurrent and atomic by tr. ID",
"count": 4000000,
"time": "2 min, 22 sec 921 ms"
```

≈ 2 min 15 sec (49%)



Итоги. Атомарность через обновление.

- Hibernate для detached entity загружает ее состояние из БД и хранит его до окончания транзакции
- Обновление по общему полю работает быстрее
- fillfactor помогает HOT обновлениям, но требует дополнительное место на диске
- В 2 раза удалось ускорить вставку данных в БД с сохранением атомарности

А что если вынести признак атомарности в отдельную таблицу?

Transaction ID в отдельной таблице.

Вариант 1

```
CREATE TABLE ACTIVE_TRANSACTION
(
    ID bigint NOT NULL,
    transaction_id uuid NOT NULL
);
ALTER TABLE ACTIVE_TRANSACTION ADD CONSTRAINT PK_ACTIVE_TRANSACTION PRIMARY KEY (ID);
CREATE INDEX IX_ACTIVE_TRANSACTION_transaction_id on ACTIVE_TRANSACTION (transaction_id);
```

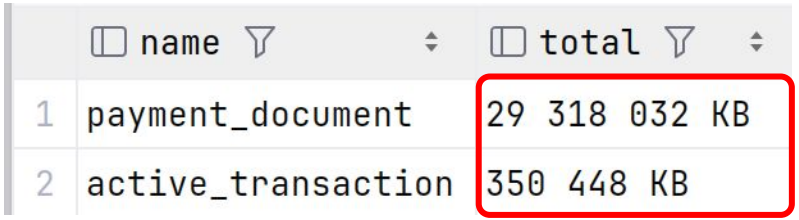
VS

Вариант 2

```
ALTER TABLE PAYMENT_DOCUMENT ADD COLUMN transaction_id uuid DEFAULT null;
CREATE TABLE ACTIVE_TRANSACTION
(
    transaction_id uuid NOT NULL
);
CREATE INDEX IX_ACTIVE_TRANSACTION_transaction_id on ACTIVE_TRANSACTION (transaction_id);
```

Transaction ID в отдельной таблице.

Вариант 1

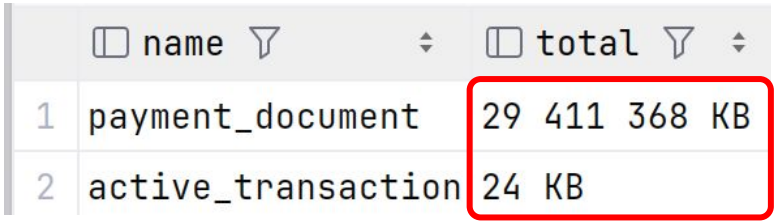


	name	total
1	payment_document	29 318 032 KB
2	active_transaction	350 448 KB

+ 4 млн. строк в active_transaction и 342 МБ

VS

Вариант 2



	name	total
1	payment_document	29 411 368 KB
2	active_transaction	24 KB

+ 1 строка в active_transaction и 91 МБ в основной таблице

Вариант 1. Transaction ID в отдельной таблице.

Вариант 1. Transaction ID в отдельной таблице.

```
@Entity
@Table(name = "active_transaction")
class PaymentDocumentActiveTransactionEntity(
    @Id
    var id: Long? = null,
    @OneToOne(cascade = [CascadeType.PERSIST])
    @MapsId
    @JoinColumn(name = "id")
    var paymentDocument: PaymentDocumentEntity? = null,
    var transactionId: UUID? = null,
)
```

Вариант 1. Transaction ID в отдельной таблице.

```
@Async("threadPoolAsyncInsertExecutor")
fun saveBatchAsync(
    entities: List<PaymentDocumentEntity>,
    transactionId: UUID
): Future<List<PaymentDocumentEntity>> {
    val savedEntities = entities
        .map { PaymentDocumentActiveTransactionEntity(paymentDocument = it, transactionId = transactionId) }
        .let { activeTransactionRepository.saveAll(it) } (Mutable)List<PaymentDocumentActiveTransactionEntity!>
        .mapNotNull { it.paymentDocument }

    return CompletableFuture.completedFuture(savedEntities)
}
```

Вариант 1. Transaction ID в отдельной таблице.

```
var listForSave = mutableListOf<PaymentDocumentEntity>()
val saveTasks = mutableListOf<Future<List<PaymentDocumentEntity>>>()
val transactionId = Generators.timeBasedEpochGenerator().generate()
(1 ≤ .. ≤ count).forEach { it: Int
    listForSave.add(
        getRandomEntity( id: null, currencies.random(), accounts.random())
    )
    if (it != 0 && it % batchSize == 0) {
        saveTasks.add(saver.saveBatchAsync(listForSave, transactionId))
        listForSave = mutableListOf()
    }
}
listForSave.takeIf { it.isNotEmpty() }?.let { saveTasks.add(saver.saveBatchAsync(it, transactionId)) }

val docs = saveTasks.flatMap { it.get() }
activeTransactionRepository.deleteAllByTransactionId(transactionId)
```

Удаление 4 млн записей

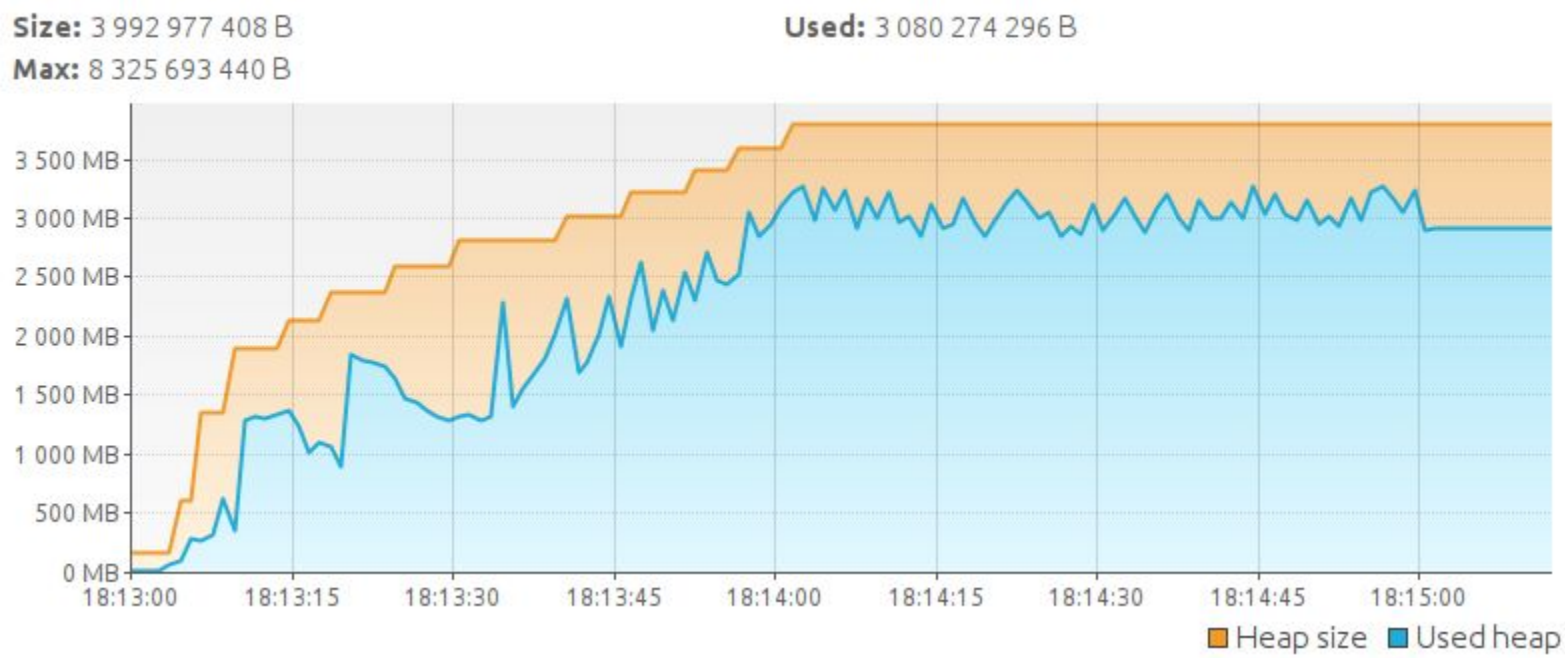
```
@Transactional
@Modifying
@Query("delete from PaymentDocumentActiveTransactionEntity where transactionId = :transactionId")
fun deleteAllByTransactionId(transactionId: UUID)
```

```
Delete on active_transaction  (cost=0.00..75483.00 rows=0 width=0) (actual time=2
  ->  Seq Scan on active_transaction  (cost=0.00..75483.00 rows=4000000 width=6)
        Filter: (transaction_id = '15ba5973-0f5d-4ff3-895f-ecb6b025f6b6'::uuid)
Planning Time: 0.240 ms
Execution Time: 2357.062 ms
```

Вариант 1. Атомарная вставка с отдельной таблицей

```
"name": "save concurrent and atomic by tr. ID
"count": 4000000,
"time": "2 min, 1 sec 559 ms"
```

≈ 21 sec (15%)



Вариант 2. Transaction ID в двух таблицах.

Вариант 2. Transaction ID в двух таблицах.

```
@Entity   fatov *
@Table(name = "active_transaction")
class ActiveTransactionEntity(
    @Id
    var transactionId: UUID? = null,
)
```

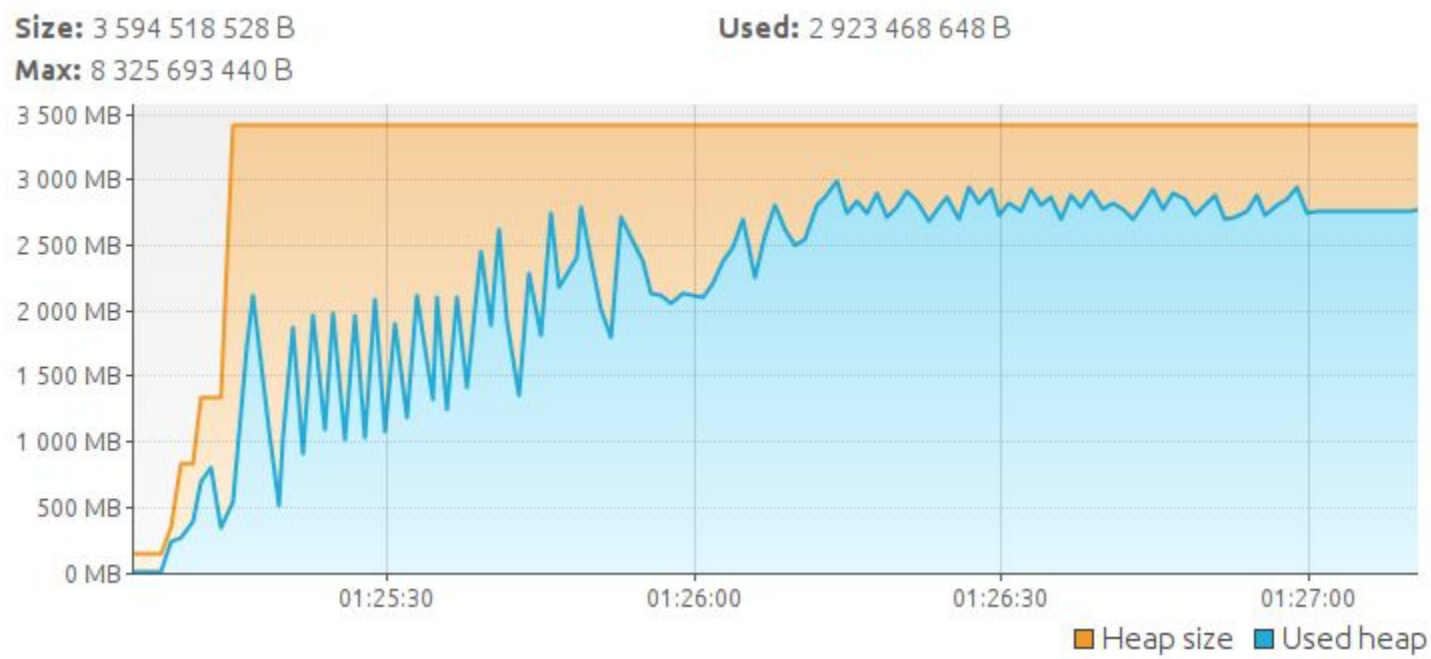
Вариант 2. Transaction ID в двух таблицах.

```
var listForSave = mutableListOf<PaymentDocumentEntity>()
val saveTasks = mutableListOf<Future<List<PaymentDocumentEntity>>>()
val transactionId = Generators.timeBasedEpochGenerator().generate()
activeTransactionRepository.saveAndFlush(ActiveTransactionEntity(transactionId = transactionId))
(1 ≤ .. ≤ count).forEach {
    listForSave.add(
        getRandomEntity( id: null, currencies.random(), accounts.random(), transactionId)
    )
    if (it != 0 && it % batchSize == 0) {
        saveTasks.add(saver.saveBatchAsync(listForSave))
        listForSave = mutableListOf()
    }
}
listForSave.takeIf { it.isNotEmpty() }?.let { saveTasks.add(saver.saveBatchAsync(it)) }

val docs = saveTasks.flatMap { it.get() }
activeTransactionRepository.deleteById(transactionId)
```

Вариант 2. Transaction ID в двух таблицах.


```
"name": "save concurrent and atomic by tr. ID in two table",
"count": 4000000,
"time": "1 min, 57 sec 167 ms"
```


≈ 4 sec (3%)



Как сделать чтение только актуальных данных?

Условие по NOT IN. Вариант 1

```
@Entity
@Table(name = "payment_document")
@Where(clause = "id not in (select at.id from active_transaction as at)")
class PaymentDocumentEntity(
```

```
from payment_document paymentdoc0_
where (paymentdoc0_.id not in (select at.id from active_transaction as at))
```

Условие по NOT IN. Вариант 1

```
from payment_document paymentdoc0_
where (paymentdoc0_.id not in (select at.id from active_transaction as at))
```

```
Index Scan using pk_payment_document on payment_document paymentdoc0_ (actual time=1174.130..1174.
  Index Cond: (id = 1000071)
  Filter: (NOT (SubPlan 1))
  SubPlan 1
    ->  Materialize (actual time=0.068..921.638 rows=4000000 loops=1)
          ->  Seq Scan on active_transaction at (actual time=0.062..303.874 rows=4000000 loops=1)
Planning Time: 0.253 ms
Execution Time: 1182.752 ms
```

Условие по NOT IN. Вариант 1

```
from payment_document paymentdoc0_
where (paymentdoc0_.id not in (select at.id from active_transaction as at))
```

```
Index Scan using pk_payment_document on payment_document paymentdoc0_ (actual time=1174.130..1174.
  Index Cond: (id = 1000071)
  Filter: (NOT (SubPlan 1))
  SubPlan 1
    ->  Materialize (actual time=0.068..921.638 rows=4000000 loops=1)
          ->  Seq Scan on active_transaction at (actual time=0.062..303.87 rows=4000000 loops=1)
Planning Time: 0.253 ms
Execution Time: 1182.752 ms
```

Условие NOT EXISTS. Вариант 1

```
@Entity
@Table(name = "payment_document")
@Where(clause = "NOT EXISTS (SELECT * FROM active_transaction at WHERE at.id = id)")
class PaymentDocumentEntity(
```

```
from payment_document paymentdoc0_
where (NOT EXISTS(SELECT * FROM active_transaction at WHERE at.id = paymentdoc0_.id))
```

Условие NOT EXISTS. Вариант 1

```
from payment_document paymentdoc0_
where (NOT EXISTS(SELECT * FROM active_transaction at WHERE at.id = paymentdoc0_.id))
```

```
Nested Loop Anti Join (actual time=0.075..0.080 rows=1 loops=1)
  Join Filter: (at.id = paymentdoc0_.id)
  ->  Index Scan using pk_payment_document on payment_document paymentdoc0_ (actual time=0.044..0.047 rows=1 loops=1)
        Index Cond: (id = 1000071)
  ->  Index Only Scan using pk_active_transaction on active_transaction at (actual time=0.022..0.022 rows=0 loops=1)
        Index Cond: (id = 1000071)
        Heap Fetches: 0
Planning Time: 0.437 ms
Execution Time: 0.157 ms
```

Условие NOT EXISTS. Вариант 2

```
@Entity
@Table(name = "payment_document")
@Where(clause = "NOT EXISTS (SELECT * FROM active_transaction at WHERE at.transaction_id = transaction_id)")
class PaymentDocumentEntity(
```

```
from payment_document paymentdoc0_
where (NOT EXISTS(SELECT * FROM active_transaction at WHERE at.transaction_id = paymentdoc0_.transaction_id))
```

Условие NOT EXISTS. Вариант 2

```
from payment_document paymentdoc0_
where (NOT EXISTS(SELECT * FROM active_transaction at WHERE at.transaction_id = paymentdoc0_.transaction_id))
```

```
Nested Loop Anti Join (actual time=0.050..0.055 rows=1 loops=1)
  Join Filter: (at.transaction_id = paymentdoc0_.transaction_id)
  Rows Removed by Join Filter: 1
  ->  Index Scan using pk_payment_document on payment_document paymentdoc0_ (actual time=0.032..0.035 rows=1 loops=1)
        Index Cond: (id = 1000071)
  ->  Seq Scan on active_transaction at (actual time=0.008..0.009 rows=1 loops=1)
Planning Time: 0.291 ms
Execution Time: 0.106 ms
```

Условие с LEFT JOIN. Вариант 1

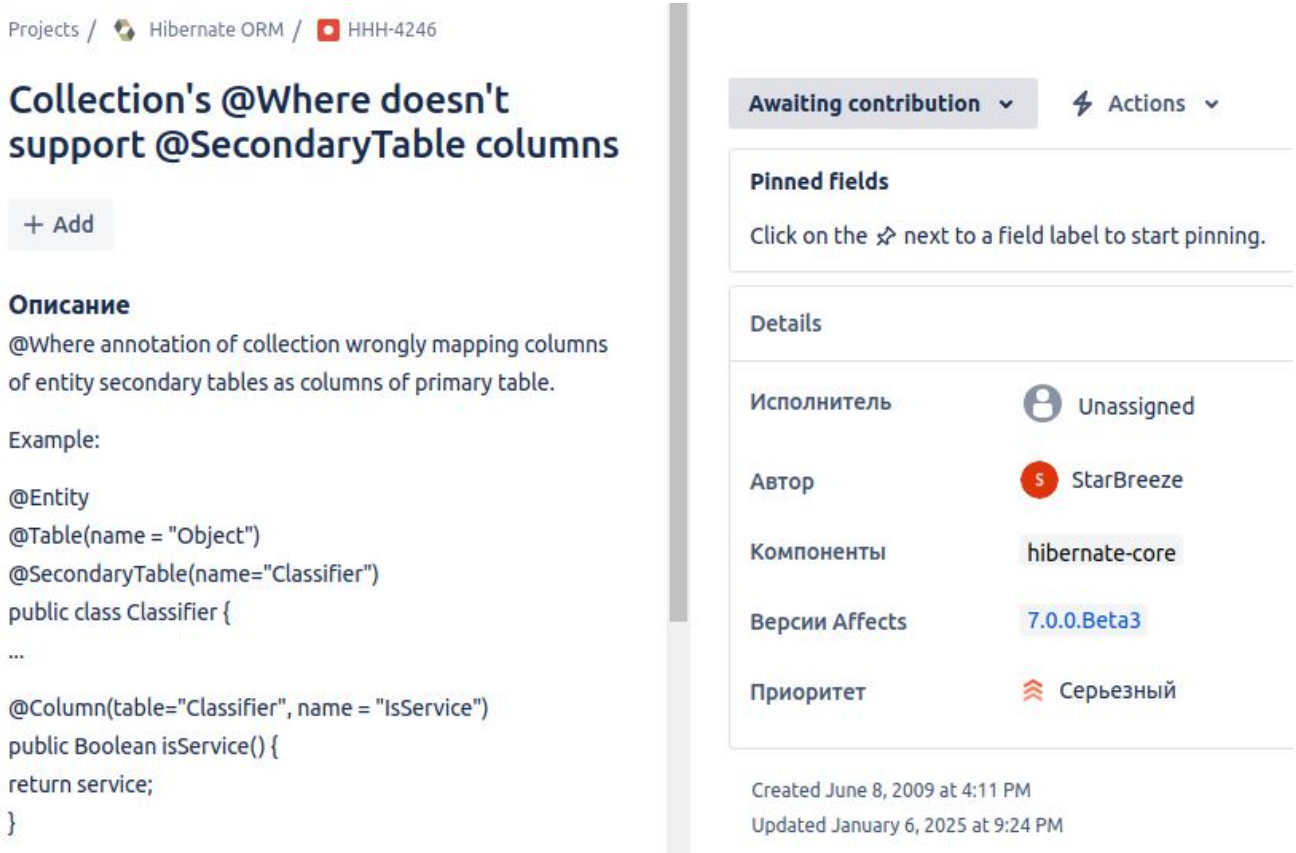
```
@Entity
@SecondaryTable(name = "active_transaction",
    pkJoinColumns = [PrimaryKeyJoinColumn(name = "id", referencedColumnName = "id")]
)
@Where(clause = "transaction_id is null")
@Table(name = "payment_document")
class PaymentDocumentEntity(


    var prop20: String? = null,
    @Column(table = "active_transaction")
    var transactionId: UUID? = null,
) : BaseEntity()
```

Условие с LEFT JOIN. Вариант 1

```
select paymentdoc0_.id                as id1_4_,
       paymentdoc0_.account_id        as account10_4_,
       paymentdoc0_.amount            as amount2_4_,
       paymentdoc0_.cur               as cur11_4_,
       paymentdoc0_.expense           as expense3_4_,
       paymentdoc0_.order_date        as order_da4_4_,
       paymentdoc0_.order_number      as order_nu5_4_,
       paymentdoc0_.payment_purpose   as payment_6_4_,
       paymentdoc0_.prop_10           as prop_7_4_,
       paymentdoc0_.prop_15           as prop_8_4_,
       paymentdoc0_.prop_20           as prop_9_4_,
       paymentdoc0_1_.transaction_id  as transact2_1_
from payment_document paymentdoc0_
         left outer join active_transaction paymentdoc0_1_ on paymentdoc0_.id = paymentdoc0_1_.id
where (paymentdoc0_.transaction_id is null)
```

Условие с LEFT JOIN. Вариант 1

HHH-4246



Projects / Hibernate ORM / HHH-4246

Collection's @Where doesn't support @SecondaryTable columns

+ Add

Описание

@Where annotation of collection wrongly mapping columns of entity secondary tables as columns of primary table.

Example:

```
@Entity
@Table(name = "Object")
@SecondaryTable(name="Classifier")
public class Classifier {
...

@Column(table="Classifier", name = "IsService")
public Boolean isService() {
return service;
}
```

Awaiting contribution Actions

Pinned fields

Click on the next to a field label to start pinning.

Details

Исполнитель	Unassigned
Автор	StarBreeze
Компоненты	hibernate-core
Версии Affects	7.0.0.Beta3
Приоритет	Серьезный

Created June 8, 2009 at 4:11 PM
Updated January 6, 2025 at 9:24 PM

Условие с LEFT JOIN. Вариант 1

```
@SecondaryTable(name = "active_transaction",
    pkJoinColumns = [PrimaryKeyJoinColumn(name = "id", referencedColumnName = "id")]
)
@Table(name = "payment_document")
@Where(clause = "#{active_transaction_condition}")
class PaymentDocumentEntity(
```

Условие с LEFT JOIN. Вариант 1

```
class SqlInterceptor: StatementInspector {

    override fun inspect(sql: String?): String? {

        return sql
            ?.takeIf { it.contains( other: "#{active_transaction_condition}" ) }
            ?.let { s ->
                val joinName = "join active_transaction\\s+(\\w+)".toRegex().findAll(s)
                var finallySql = s
                joinName.forEach { it: MatchResult
                    finallySql = s.replaceFirst(
                        oldValue: "#{active_transaction_condition}",
                        newValue: "${it.groupValues[1]}.transaction_id is null"
                    )
                }
                finallySql ^let
            } ?: sql
    }
}
```

Условие с LEFT JOIN. Вариант 1

```
select paymentdoc0_.id                as id1_4_,
       paymentdoc0_.account_id        as account10_4_,
       paymentdoc0_.amount            as amount2_4_,
       paymentdoc0_.cur               as cur11_4_,
       paymentdoc0_.expense           as expense3_4_,
       paymentdoc0_.order_date        as order_da4_4_,
       paymentdoc0_.order_number      as order_nu5_4_,
       paymentdoc0_.payment_purpose   as payment_6_4_,
       paymentdoc0_.prop_10           as prop_7_4_,
       paymentdoc0_.prop_15           as prop_8_4_,
       paymentdoc0_.prop_20           as prop_9_4_,
       paymentdoc0_1_.transaction_id as transact2_1_
from payment_document paymentdoc0_
         left outer join active_transaction paymentdoc0_1_ on paymentdoc0_.id = paymentdoc0_1_.id
where (paymentdoc0_1_.transaction_id is null)
```

Условие с LEFT JOIN. Вариант 1

```
from payment_document paymentdoc0_
         left outer join active_transaction paymentdoc0_1_ on paymentdoc0_.id = paymentdoc0_1_.id
where (paymentdoc0_1_.transaction_id is null)
```

```
Nested Loop Left Join (actual time=0.091..0.092 rows=1 loops=1)
  Join Filter: (paymentdoc0_.id = paymentdoc0_1_.id)
  Filter: (paymentdoc0_1_.transaction_id IS NULL)
  ->  Index Scan using pk_payment_document on payment_document paymentdoc0_ (actual time=0.067..0.067 rows=1 loops=1)
        Index Cond: (id = 1000071)
  ->  Index Scan using pk_active_transaction on active_transaction paymentdoc0_1_ (actual time=0.020..0.020 rows=0 loops=1)
        Index Cond: (id = 1000071)
Planning Time: 3.752 ms
Execution Time: 0.137 ms
```

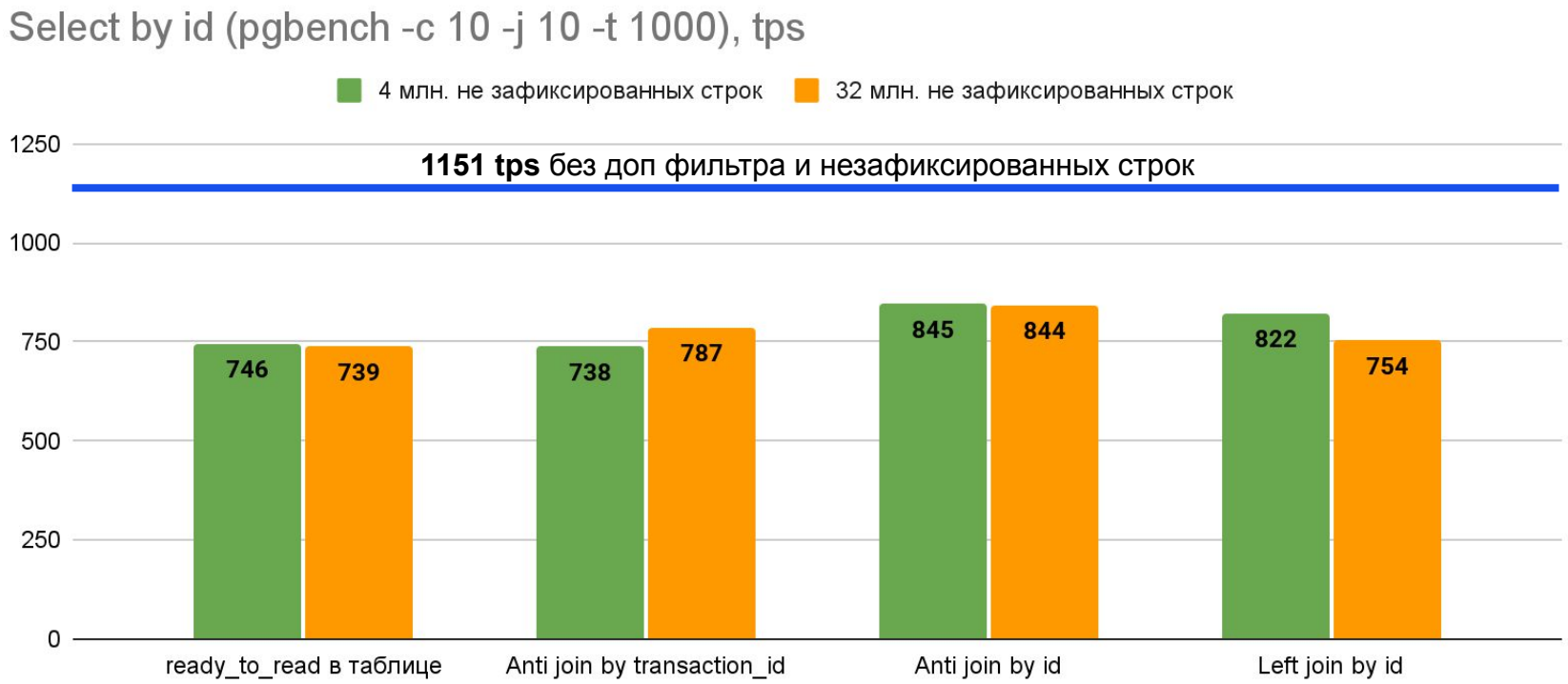
Условие с LEFT JOIN. Вариант 2

```
from payment_document paymentdoc0_
        left outer join active_transaction paymentdoc0_1_ on paymentdoc0_.transaction_id = paymentdoc0_1_.transaction_id
where (paymentdoc0_1_.transaction_id is null)
```

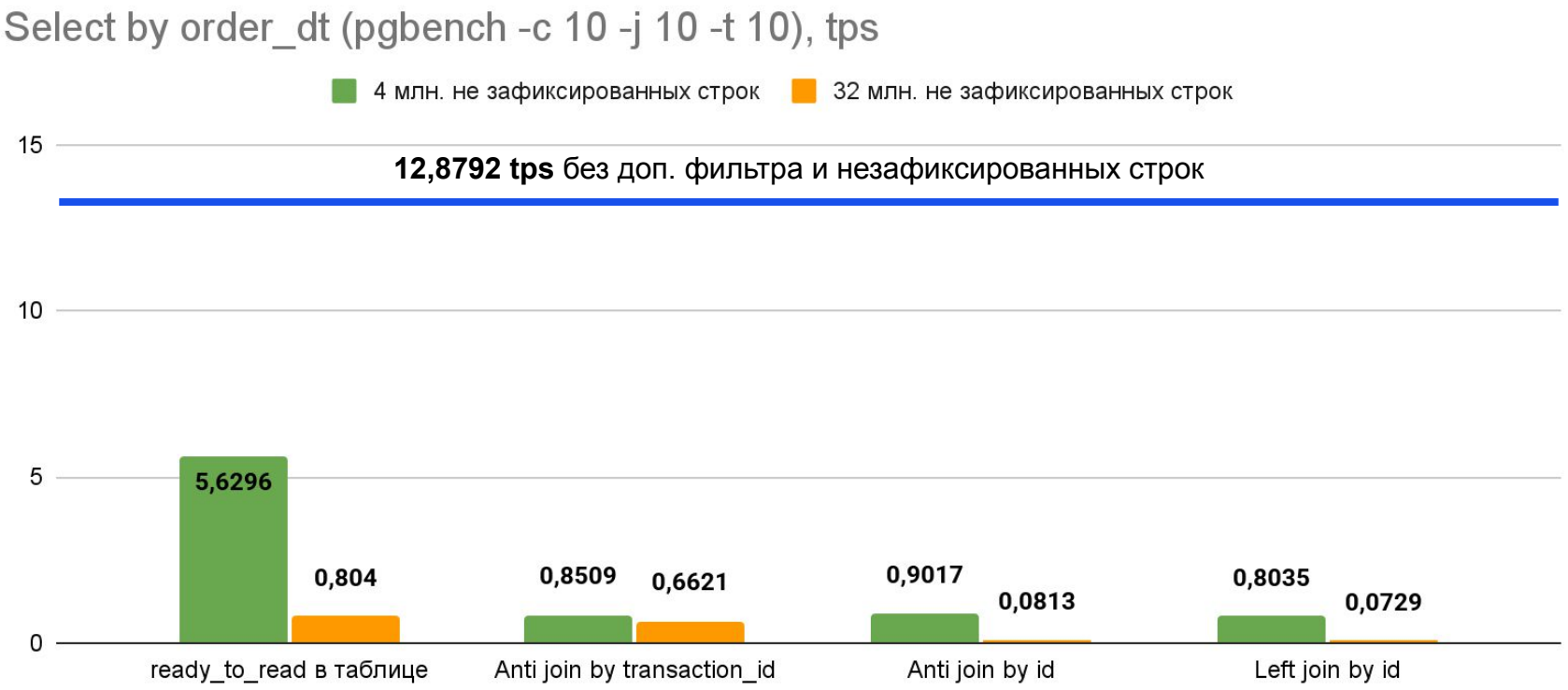
```
Nested Loop Anti Join (actual time=0.051..0.056 rows=1 loops=1)
  Join Filter: (paymentdoc0_.transaction_id = paymentdoc0_1_.transaction_id)
  Rows Removed by Join Filter: 1
  ->  Index Scan using pk_payment_document on payment_document paymentdoc0_ (actual time=0.032..0.035 rows=1 loops=1)
        Index Cond: (id = 1000071)
  ->  Seq Scan on active_transaction paymentdoc0_1_ (actual time=0.008..0.009 rows=1 loops=1)
Planning Time: 0.310 ms
Execution Time: 0.108 ms
```

У кого больше TPS?

Выборка 1 записи по первичному ключу (id)



Выборка 10к записей по индексному полю (order_dt)



Итоги. transaction_id в смежных таблицах

- Удобство использования. Не нужно тюнить postgres.
- Можно обойтись без внесения изменений в основную таблицу
- Более чем на 15% работает быстрее чем с дополнительным полем в основной таблице
- Anti join vs Left join. Приблизительно одинаковые показатели.
- Нужно следить за количеством незафиксированных данных.

Какие проблемы могут быть с @Where

Проблема 1. Соединение по первичному ключу

```
CREATE TABLE STATEMENT
(
    payment_document_id bigint NULL,
    ID bigint NOT NULL DEFAULT nextval('seq_id')
);
ALTER TABLE STATEMENT ADD CONSTRAINT PK_STATEMENT PRIMARY KEY (ID);
ALTER TABLE STATEMENT ADD CONSTRAINT FK_STATEMENT_payment_document_id FOREIGN KEY (payment_document_id) REFERENCES PAYM
```

```
@Entity
@Table(name = "statement")
class Statement(
    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "payment_document_id", referencedColumnName = "id")
    var paymentDocument: PaymentDocumentEntity? = null,
) : BaseEntity()
```

Проблема 1. Соединение по первичному ключу

```
statementRepo.findByPaymentDocument(paymentDocument)
```

```
select statement0_.id as id1_5_, statement0_.payment_document_id as payment_2_5_
from statement statement0_
where statement0_.payment_document_id=?;

from payment_document paymentdoc0_
        left outer join account accountent1_ on paymentdoc0_.account_id = accountent1_.id
        left outer join currency currencyen2_ on accountent1_.cur = currencyen2_.code
        left outer join currency currencyen3_ on paymentdoc0_.cur = currencyen3_.code
where paymentdoc0_.id=?
  and (NOT EXISTS(SELECT * FROM active_transaction at WHERE at.id = paymentdoc0_.id));
```

Проблема 1. Соединение по первичному ключу

```
statementRepo.getById(id)
```

```
select statement0_.id as id1_5_0_, statement0_.payment_document_id as payment_2_5_0_,
       paymentdoc1_.id as id1_4_1_, paymentdoc1_.account_id as account10_4_1_, paymentdoc1_.amount as amount
       accountent2_.id as id1_0_2_, accountent2_.cur as cur4_0_2_, accountent2_.name as name2_0_2_, accounte
       currencyen3_.id as id1_3_3_, currencyen3_.code as code2_3_3_, currencyen3_.name as name3_3_3_,
       currencyen4_.id as id1_3_4_, currencyen4_.code as code2_3_4_, currencyen4_.name as name3_3_4_
from statement statement0_
        left outer join payment_document paymentdoc1_ on statement0_.payment_document_id = paymentdoc1_.id
        left outer join account accountent2_ on paymentdoc1_.account_id = accountent2_.id
        left outer join currency currencyen3_ on accountent2_.cur = currencyen3_.code
        left outer join currency currencyen4_ on paymentdoc1_.cur = currencyen4_.code
where statement0_.id=?;
```

Проблема 1. Соединение по первичному ключу

```
@Override
public String getOnCondition(
        String alias,
        SessionFactoryImplementor factory,
        Map enabledFilters,
        Set<String> treatAsDeclarations) {
    if ( isReferenceToPrimaryKey() && ( treatAsDeclarations == null || treatAsDeclarations.isEmpty() ) ) {
        return "";
    }
    else {
        return getAssociatedJoinable( factory ).filterFragment( alias, enabledFilters, treatAsDeclarations );
    }
}
```

* Класс EntityType

Проблема 1. Соединение по первичному ключу

```
@Override
public String getOnCondition(
        String alias,
        SessionFactoryImplementor factory,
        Map enabledFilters,
        Set<String> treatAsDeclarations
    if ( isReferenceToPrimaryKey( treatAsDeclarations ) || treatAsDeclarations == null || treatAsDeclarations.isEmpty() ) ) {
        return "";
    }
    else {
        return getAssociatedJoinable( factory ).filterFragment( alias, enabledFilters, treatAsDeclarations );
    }
}
```

FIXED SINCE 3.1.0 VERSION

* Класс EntityType

Проблема 2. Соединение НЕ по первичному ключу

```
ALTER TABLE PAYMENT_DOCUMENT ADD COLUMN statements bigint DEFAULT nextval('seq_id');
CREATE TABLE STATEMENT
(
    payment_document_collection_id bigint NULL,
    ID bigint NOT NULL DEFAULT nextval('seq_id')
);
```

```
@Entity
@Table(name = "statement")
class Statement(
    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "payment_document_collection_id", referencedColumnName = "statements")
    var paymentDocument: PaymentDocumentEntity? = null,
) : BaseEntity()
```

Проблема 2. Соединение НЕ по первичному ключу

```
select statement0_.id as id1_5_0_, statement0_.payment_document_collection_id as payment_2_5_0_,
       paymentdoc1_.id as id1_4_1_, paymentdoc1_.account_id as account11_4_1_, paymentdoc1_.amount as amount2
       accountent2_.id as id1_0_2_, accountent2_.cur as cur4_0_2_, accountent2_.name as name2_0_2_, accounter
       currencyen3_.id as id1_3_3_, currencyen3_.code as code2_3_3_, currencyen3_.name as name3_3_3_,
       currencyen4_.id as id1_3_4_, currencyen4_.code as code2_3_4_, currencyen4_.name as name3_3_4_
from statement statement0_
        left outer join payment_document paymentdoc1_
                        on statement0_.payment_document_collection_id = paymentdoc1_.statements and
                           (NOT EXISTS(SELECT * FROM active_transaction at WHERE at.id = paymentdoc1_.id))
        left outer join account accountent2_ on paymentdoc1_.account_id = accountent2_.id
        left outer join currency currencyen3_ on accountent2_.cur = currencyen3_.code
        left outer join currency currencyen4_ on paymentdoc1_.cur = currencyen4_.code
where statement0_.id=?;
```

Проблема 2. Соединение НЕ по первичному ключу

```
select statement0_.id as id1_5_0_, statement0_.payment_document_collection_id as payment_2_5_0_,
       paymentdoc1_.id as id1_4_1_, paymentdoc1_.account_id as account11_4_1_, paymentdoc1_.amount as
       paymentdoc1_1_.transaction_id as transact2_1_1_,
       accountent2_.id as id1_0_2_, accountent2_.cur as cur4_0_2_, accountent2_.name as name2_0_2_, ac
       currencyen3_.id as id1_3_3_, currencyen3_.code as code2_3_3_, currencyen3_.name as name3_3_3_,
       currencyen4_.id as id1_3_4_, currencyen4_.code as code2_3_4_, currencyen4_.name as name3_3_4_
from statement statement0_
        left outer join payment_document paymentdoc1_
                        on statement0_.payment_document_collection_id = paymentdoc1_.statements and
                           (paymentdoc1_1_.transaction_id is null)
        left outer join active_transaction paymentdoc1_1_ on paymentdoc1_.id = paymentdoc1_1_.id
        left outer join account accountent2_ on paymentdoc1_.account_id = accountent2_.id
        left outer join currency currencyen3_ on accountent2_.cur = currencyen3_.code
        left outer join currency currencyen4_ on paymentdoc1_.cur = currencyen4_.code
where statement0_.id=?;
```

Проблема 2. Соединение НЕ по первичному ключу

```
select statement0_.id as id1_5_0_, statement0_.payment_document_collection_id
       paymentdoc1_.id as id1_4_1_, paymentdoc1_.account_id as account1
       paymentdoc1_1_.transaction_id as transact2_1_1_,
       accountent2_.id as id1_0_2_, accountent2_.cur as cu
       currencyen3_.id as id1_3_3_, currencyen3_.co
       currencyen4_.id as id1_3_4_, currency
from statement statement0_
        left outer join paym
                                                    ayment_document_collection_id = paymentdoc1_.statements and
                                                    ntdoc1_1_.transaction_id is null)
        le                                          tive_transaction paymentdoc1_1_ on paymentdoc1_.id = paymentdoc1_1_.id
        le                                          join account accountent2_ on paymentdoc1_.account_id = accountent2_.id
        left outer join currency currencyen3_ on accountent2_.cur = currencyen3_.code
        left outer join currency currencyen4_ on paymentdoc1_.cur = currencyen4_.code
where statement0_.id=?;
```

Пример запроса из версии spring boot 3.1.0

```
select s1_0.id,p1_0.id,a1_0.id,c1_0.id,c1_0.code,c1_0.name,a1_0.name,a1_0.number,p1_0.amount,
       c2_0.id,c2_0.code,c2_0.name,p1_0.expense,p1_0.order_date,p1_0.order_number,
       p1_0.payment_purpose,p1_0.prop_10,p1_0.prop_15,p1_0.prop_20,p1_1.transaction_id
from statement s1_0
         left join (payment_document p1_0 left join active_transaction p1_1 on p1_0.id = p1_1.id)
                   on p1_0.id = s1_0.payment_document_id and (p1_1.transaction_id is null)
         left join account a1_0 on a1_0.id = p1_0.account_id
         left join currency c1_0 on c1_0.code = a1_0.cur
         left join currency c2_0 on c2_0.code = p1_0.cur
where s1_0.id=?;
```

Реализация атомарности параллельных вставок на стороне PostgreSQL возможна?

Prepare transaction

PREPARE TRANSACTION

PREPARE TRANSACTION — подготовить текущую транзакцию для двухфазной фиксации

Синтаксис

```
PREPARE TRANSACTION id_транзакции
```

Описание

PREPARE TRANSACTION подготавливает текущую транзакцию для двухфазной фиксации. После этой команды транзакция перестаёт быть связанной с текущим сеансом; её состояние полностью сохраняется на диске, и есть очень большая вероятность, что она будет успешно зафиксирована, даже если до этого времени работа базы данных будет прервана аварийно.

https://postgrespro.ru/docs/postgrespro/16/sql-prepare-transaction

Prepare transaction

```
conn1.createStatement().use { stmt ->
    stmt.execute( sql: "insert into payment_document (prop_10, prop_15, prop_20) values ('$p10', '$p15', '$p20')")
    stmt.execute( sql: "prepare transaction '$transactionId1'")
}
conn1.commit()

conn2.createStatement().use { stmt ->
    stmt.execute( sql: "insert into payment_document (prop_10, prop_15, prop_20) values ('$p10', '$p15', '$p20')")
    stmt.execute( sql: "prepare transaction '$transactionId2'")
}
conn2.commit()


conn3.createStatement().use { stmt ->
    stmt.execute( sql: "commit prepared '$transactionId1'")
    stmt.execute( sql: "commit prepared '$transactionId2'")
}
conn3.commit()
```

Prepare transaction

```
conn3.createStatement().use { stmt ->
    stmt.execute( sql: "commit prepared '$transactionId1'")
    stmt.execute( sql: "commit prepared '$transactionId2'")
}
conn3.commit()
```

```
ERROR: COMMIT PREPARED cannot run inside a transaction block
org.postgresql.util.PSQLException Create breakpoint : ERROR: COMMIT PREPARED cannot run inside a transaction block
    at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2713)
    at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:2401)
    at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:368)
    at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:498)
    at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:415)
```

Prepare transaction

```
conn1.createStatement().use { stmt ->
    stmt.execute( sql: "insert into payment_document (prop_10, prop_15, prop_20) values ('$p10', '$p15', '$p20')")
    stmt.execute( sql: "prepare transaction '$transactionId1'")
}
conn1.commit()

conn2.createStatement().use { stmt ->
    stmt.execute( sql: "insert into payment_document (prop_10, prop_15, prop_20) values ('$p10', '$p15', '$p20')")
    stmt.execute( sql: "prepare transaction '$transactionId1'")
}
conn2.commit()


conn3.autoCommit = true
conn3.createStatement().use { stmt ->
    stmt.execute( sql: "commit prepared '$transactionId1'")
}
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql: "insert into payment_document (prop_10, prop_15, prop_20) values ('$p10', '$p15', '$p20')")
    stmt.execute( sql: "prepare transaction '$transactionId1'")
}
conn2.commit()
```

```
ERROR: transaction identifier "aa0b56ff-3978-4818-8550-6b2648d77a34" is already in use
org.postgresql.util.PSQLException Create breakpoint : ERROR: transaction identifier "aa0b56ff-3978-4818-8550-6b2648d77a34" is already in use
    at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2713)
    at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:2401)
    at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:368)
    at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:498)
    at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:415)
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql:
    stmt.execute( sql:
}
conn2.commit()
```

```
ERROR: transaction identi
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql:
    stmt.execute( sql:
}
conn2.commit()
```

```
ERROR: transaction identi
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql:
    stmt.execute( sql:
}
conn2.commit()
```

```
ERROR: transaction identif
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql:
    stmt.execute( sql:
}
conn2.commit()
```

```
ERROR: transaction identif
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql:
    stmt.execute( sql:
}
conn2.commit()
```

```
ERROR: transaction identif
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql:
    stmt.execute( sql:
}
conn2.commit()
```

```
ERROR: transaction identif
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql:
    stmt.execute( sql:
}
conn2.commit()
```

```
ERROR: transaction identi
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

Prepare transaction

```
conn2.createStatement().use { stmt ->
    stmt.execute( sql:
    stmt.execute( sql:
}
conn2.commit()
```

```
ERROR: transaction identif
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

```
10', '$p15', '$p20')")
```

```
3d77a34" is already in use
```

Prepare tr

```
conn2.createStatement(
    stmt.execute( sql: '
    stmt.execute( sql: '
}
conn2.commit()
```

```
ERROR: transaction identif
org.postgresql.util.PSQLEx
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.core
    at org.postgresql.jdbc
    at org.postgresql.jdbc
```

```
10', '$p15', '$p20')")
```

```
8d77a34" is already in use
```

Vadim Lakt
last seen 2 minutes ago

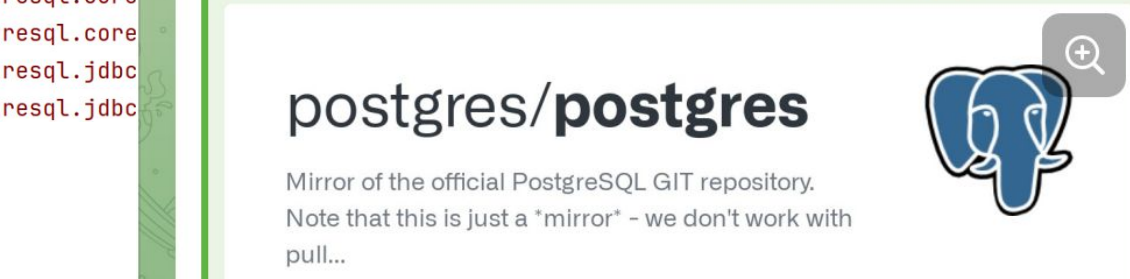
v1-0001-Introduced-the-abi...ecify-several-prepare.patch
38.9 KB 11:42

git checkout e215166c9c810950cff101cc098e66c8758538fa

git apply ~/v1-0001-Introduced-the-ability-to-specify-several-prepare.patch
edited 11:44

https://github.com/postgres/postgres

GitHub
GitHub - postgres/postgres: Mirror of the official PostgreSQL GIT repository. Note that this is just a *mirror* - we don't wo...
Mirror of the official PostgreSQL GIT repository. Note that this is just a *mirror* - we don't work with pull requests on github. To...



Prepare transaction

```
conn1.createStatement().use { stmt ->
    stmt.execute( sql = "insert into payment_document (prop_10, prop_15, prop_20) values ('$p10', '$p15', '$p20')")
    stmt.execute( sql = "prepare transaction '$transactionId1'")

}

conn2.createStatement().use { stmt ->
    stmt.execute( sql = "insert into payment_document (prop_10, prop_15, prop_20) values ('$p10', '$p15', '$p20')")
    stmt.execute( sql = "prepare transaction '$transactionId2'")
}
conn2.commit()

conn3.createStatement().use { stmt ->
    stmt.execute( sql = "commit prepared '$transactionId1', '$transactionId2'")
}
```

Prepare tran

```
conn1.createStatement().
    stmt.execute( sql = "i                                                    $p10', '$p15', '$p20')")
    stmt.execute( sql = "p

}

conn2.createStatement().
    stmt.execute( sql = "i                                                    $p10', '$p15', '$p20')")
    stmt.execute( sql = "p
}
conn2.commit()

conn3.createStatement().
    stmt.execute( sql = "c
}
```

Итоги доклада:

- Реализации атомарности в самой таблице. Нужно тюнить postgres, за скорость платим местом на диске.
- Реализация атомарности в смежных таблицах. Удобство использования, на 15% быстрее чем первый вариант.
- @Where корректно работает начиная с версии Spring Boot 3.1.0
- Все native query должны учитывать реализацию атомарности на уровне бизнес-логики.

Телеграм: @FatovDI

Спасибо за внимание!

Репозиторий:
https://github.com/FatovDI/atomic-multithreaded-insertion-postgresql