



Дружим .NET и PostgreSQL Serializable с параллельностью

Черняев Антон
TeamLead в Altenar

АНТОН ЧЕРНЯЕВ

Tg: @anchernyaev

6 лет занимаюсь разработкой
на .NET

Суммарно более 3 лет руковожу
командами Backend-разработки

Как разработчик решал множество
проблем с Serializable в PostgreSQL



ПРО КОМПАНИЮ



ALTENAR

Международная IT-компания, основанная в 2011 году.

Специализация — разработка высоконагруженного ПО для лицензированных игровых операторов (B2B).

Используемые SaaS-решения:

- **real time конвейеры для производства вероятностей в спортивных мероприятиях;**
- платформы для интеграции с банковскими системами;
- платформы приёма спортивных ставок;
- мобильные и терминальные решения.

Основной стек технологий: .NET, React, PHP, Go & Flutter.

КТО ЗНАКОМ С РЕЛЯЦИОННЫМИ БАЗАМИ И ЗНАЕТ SQL?

ПЛАН ДОКЛАДА:

ПЛАН ДОКЛАДА:

- Предметная область

ПЛАН ДОКЛАДА:

- Предметная область
- Задача с параллельными операциями, которую мы решаем

ПЛАН ДОКЛАДА:

- Предметная область
- Задача с параллельными операциями, которую мы решаем
- Способы решения задачи при помощи PostgreSQL

ПЛАН ДОКЛАДА:

- Предметная область
- Задача с параллельными операциями, которую мы решаем
- Способы решения задачи при помощи PostgreSQL
- Serializable, как выбранный способ решения

ПЛАН ДОКЛАДА:

- Предметная область
- Задача с параллельными операциями, которую мы решаем
- Способы решения задачи при помощи PostgreSQL
- Serializable, как выбранный способ решения
- Проблемы с Serializable

ПЛАН ДОКЛАДА:

- Предметная область
- Задача с параллельными операциями, которую мы решаем
- Способы решения задачи при помощи PostgreSQL
- Serializable, как выбранный способ решения
- Проблемы с Serializable
- Выводы

ПЛАН ДОКЛАДА:

- Предметная область
- Задача с параллельными операциями, которую мы решаем
- Способы решения задачи при помощи PostgreSQL
- Serializable, как выбранный способ решения
- Проблемы с Serializable
- Выводы
- Ссылки на презентацию и полезную литературу

ПРЕДМЕТНАЯ ОБЛАСТЬ

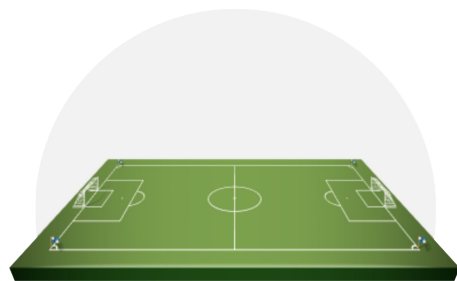


ПРЕДМЕТНАЯ ОБЛАСТЬ



Altenar Data Feed

ПРЕДМЕТНАЯ ОБЛАСТЬ



Футбольный стадион

ГОЛ



Altenar Data Feed

ПРЕДМЕТНАЯ ОБЛАСТЬ



ПРЕДМЕТНАЯ ОБЛАСТЬ



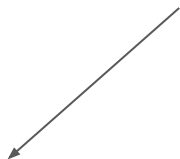
Потребитель



ПРЕДМЕТНАЯ ОБЛАСТЬ

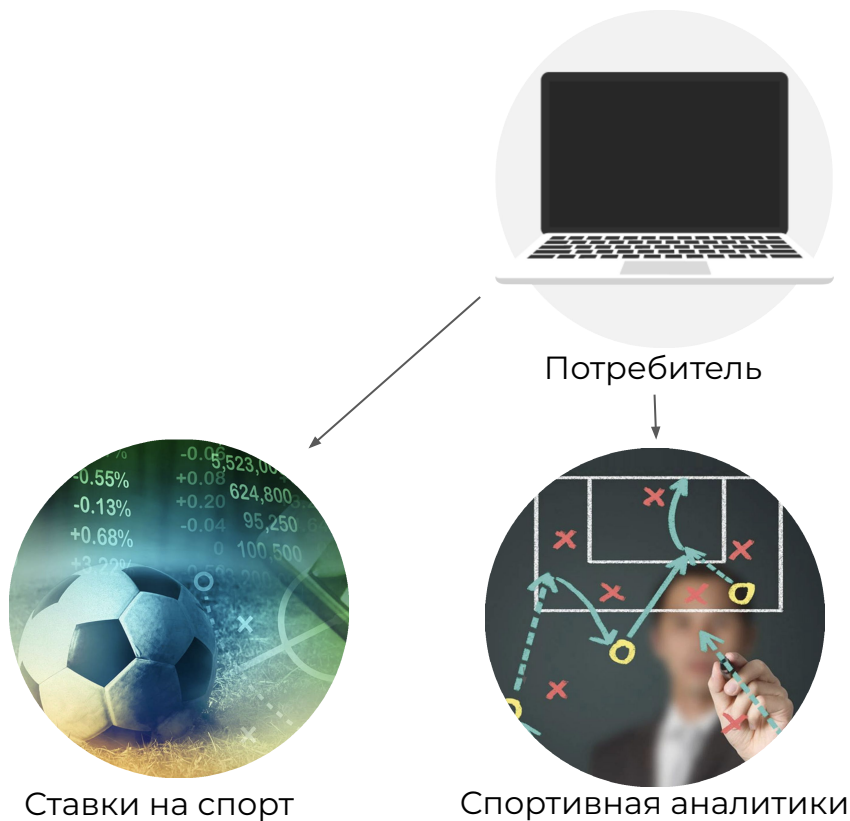


Потребитель

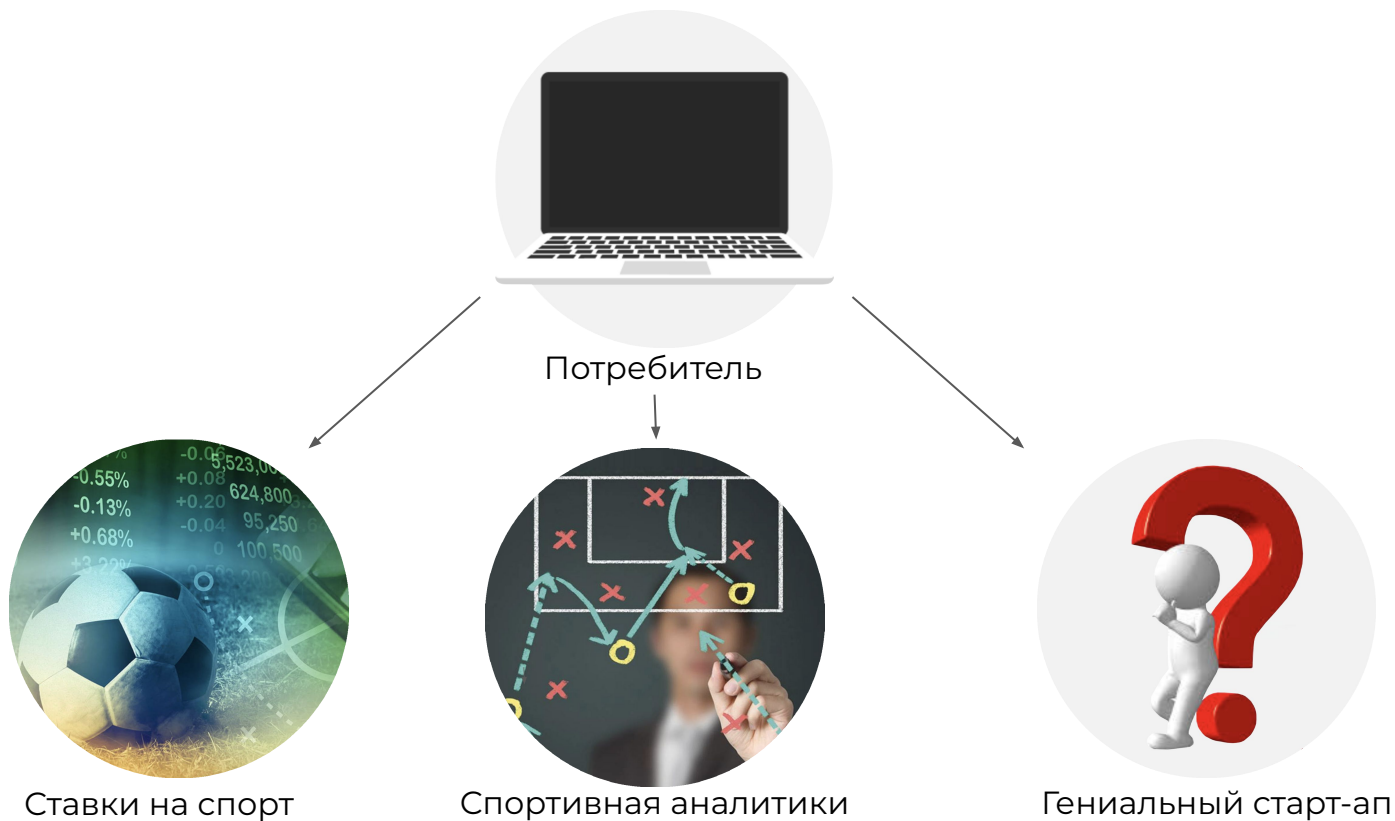


Ставки на спорт

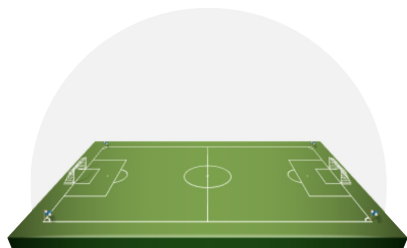
ПРЕДМЕТНАЯ ОБЛАСТЬ



ПРЕДМЕТНАЯ ОБЛАСТЬ



ПРЕДМЕТНАЯ ОБЛАСТЬ



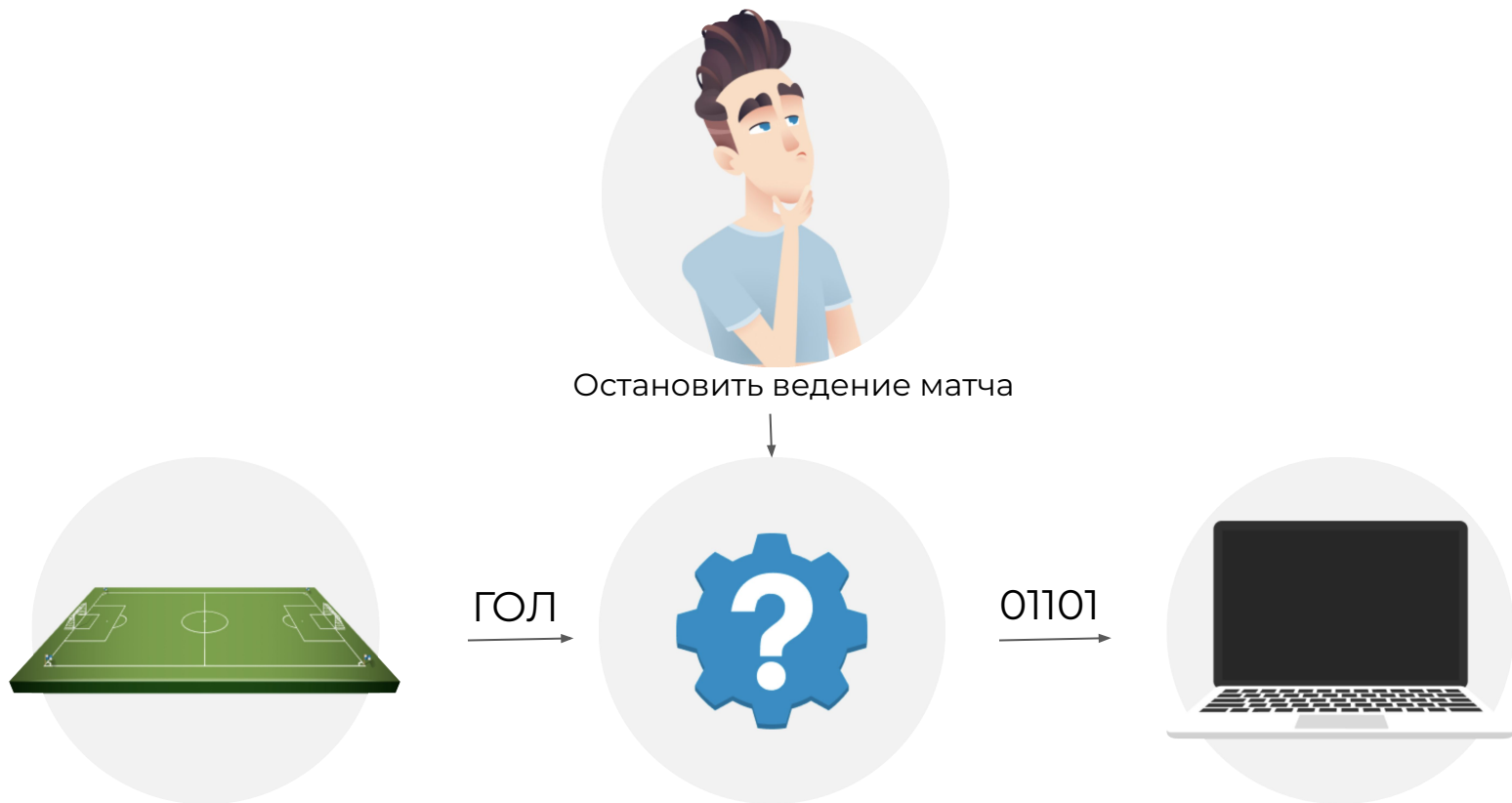
ГОЛ
→



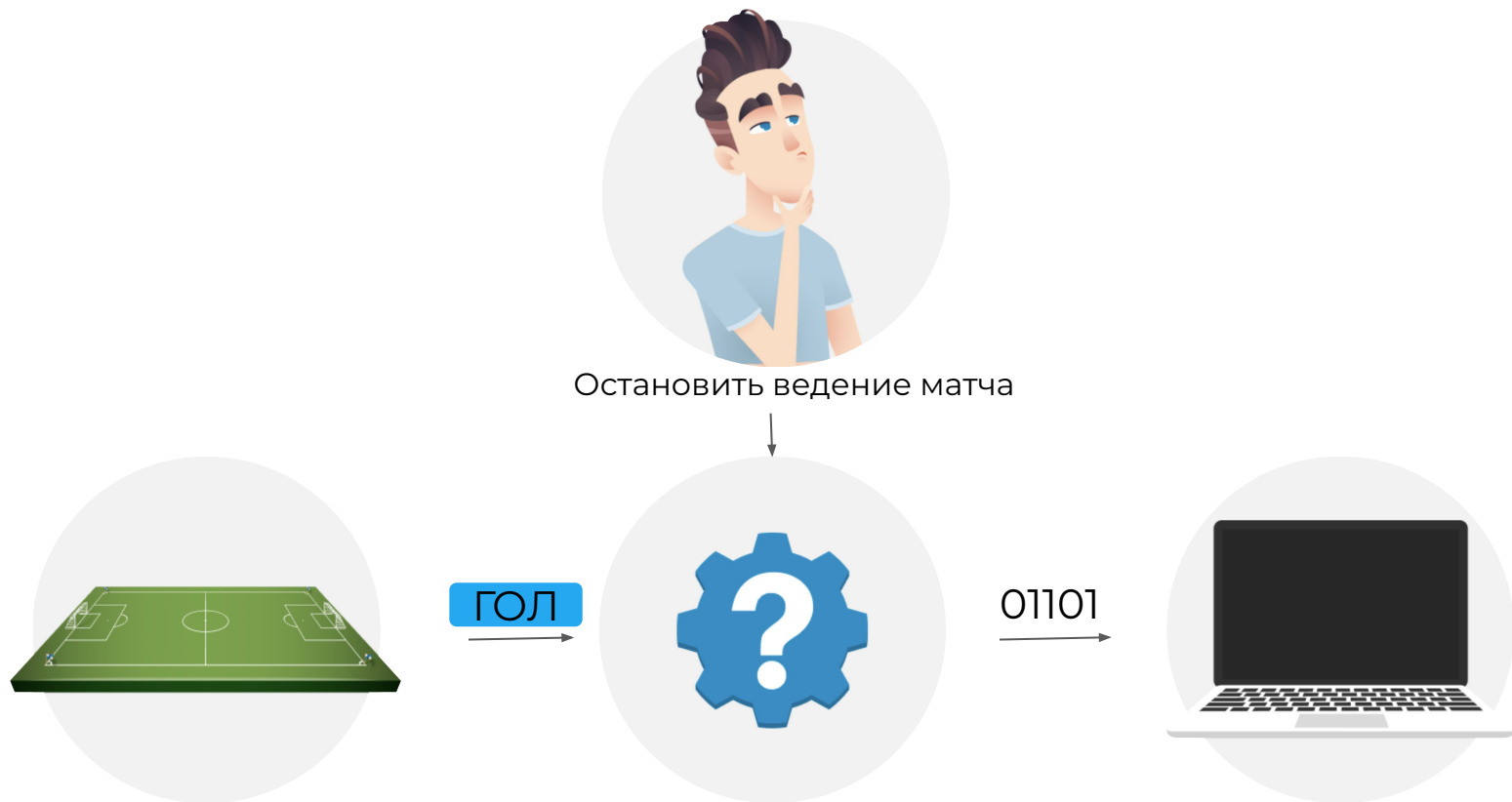
01101
→



ПРЕДМЕТНАЯ ОБЛАСТЬ



ПРЕДМЕТНАЯ ОБЛАСТЬ



ПРЕДМЕТНАЯ ОБЛАСТЬ



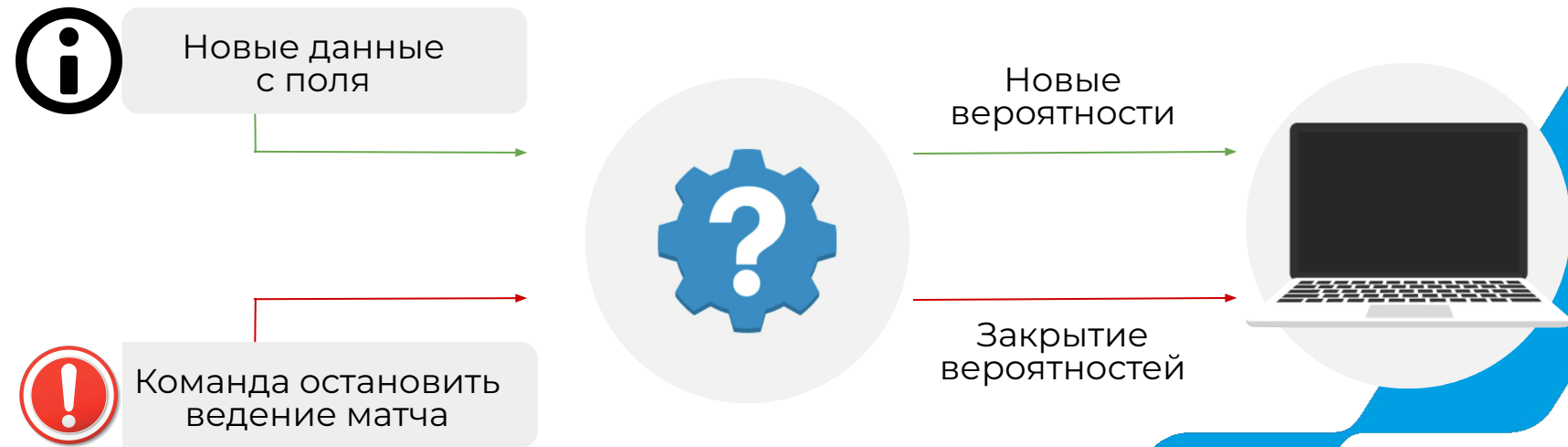
ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



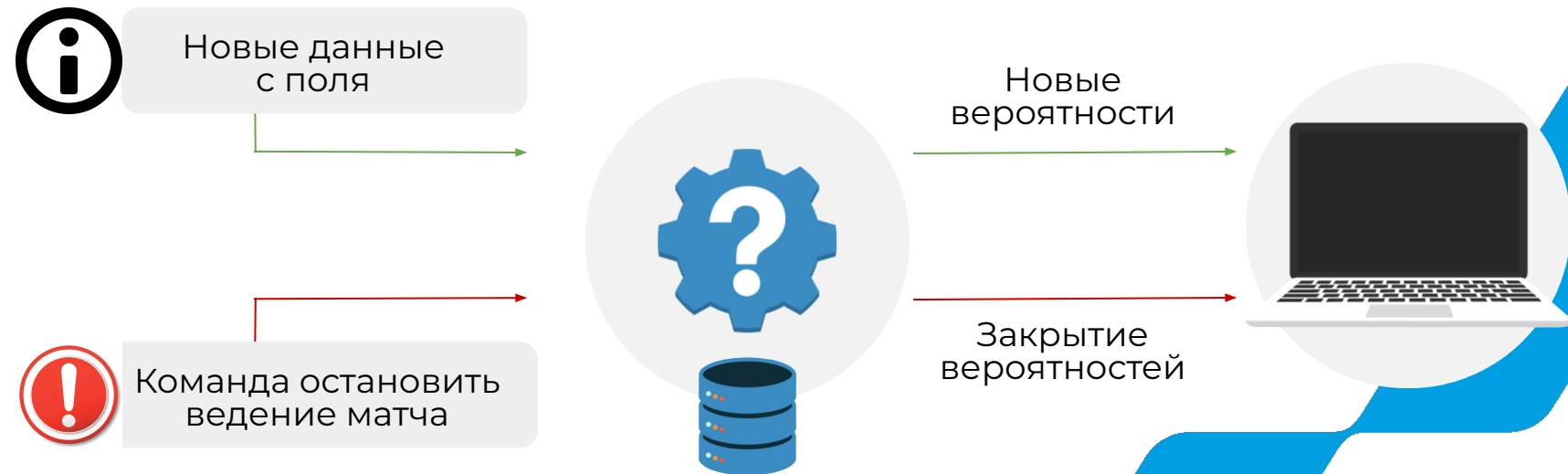
ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



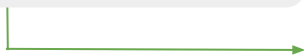
ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



Новые данные
с поля



Команда остановить
ведение матча



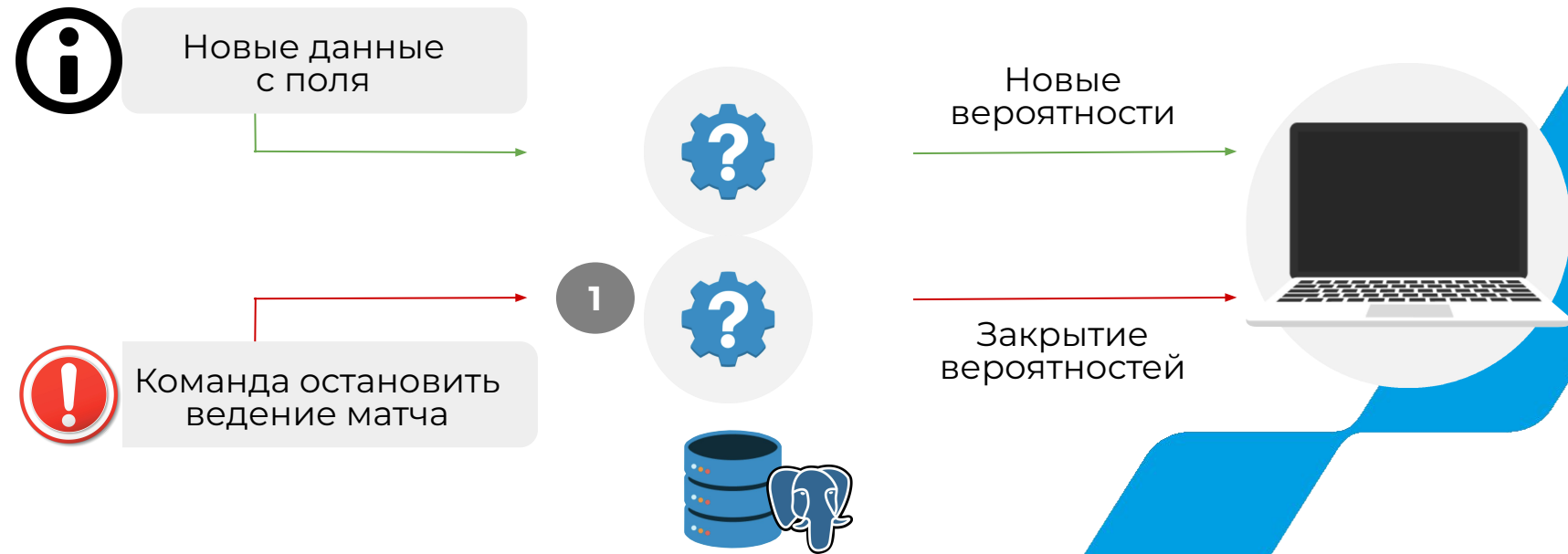
Новые
вероятности



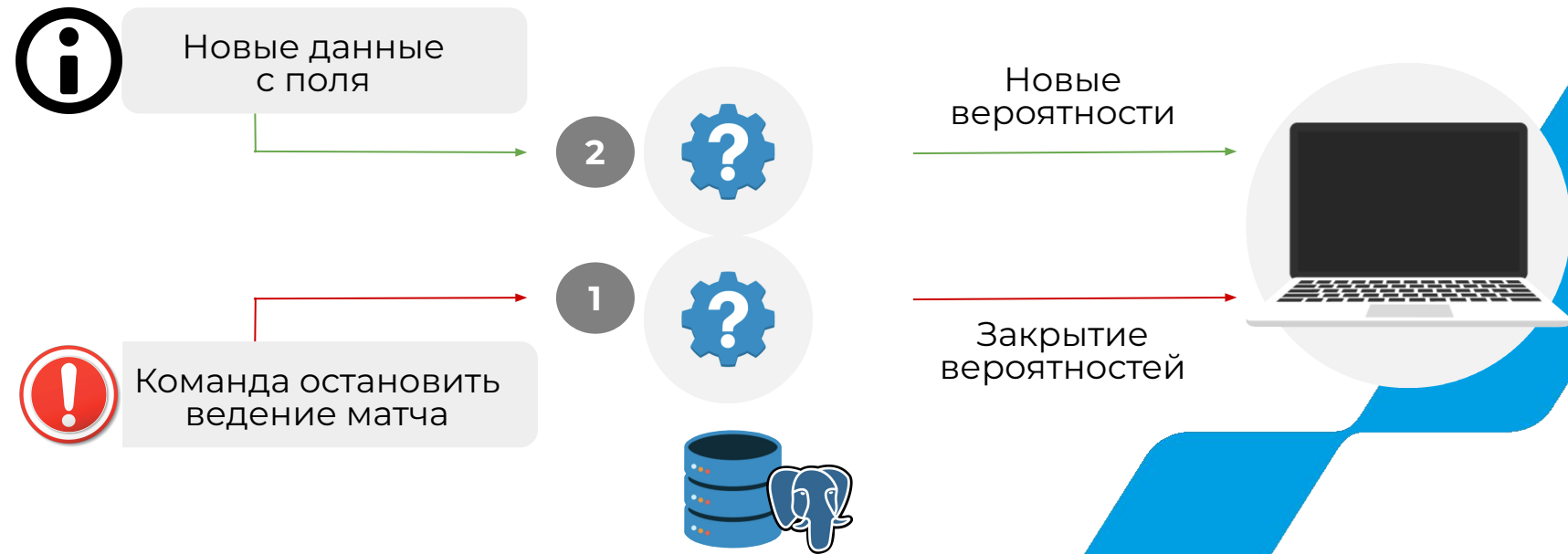
Закрытие
вероятностей



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



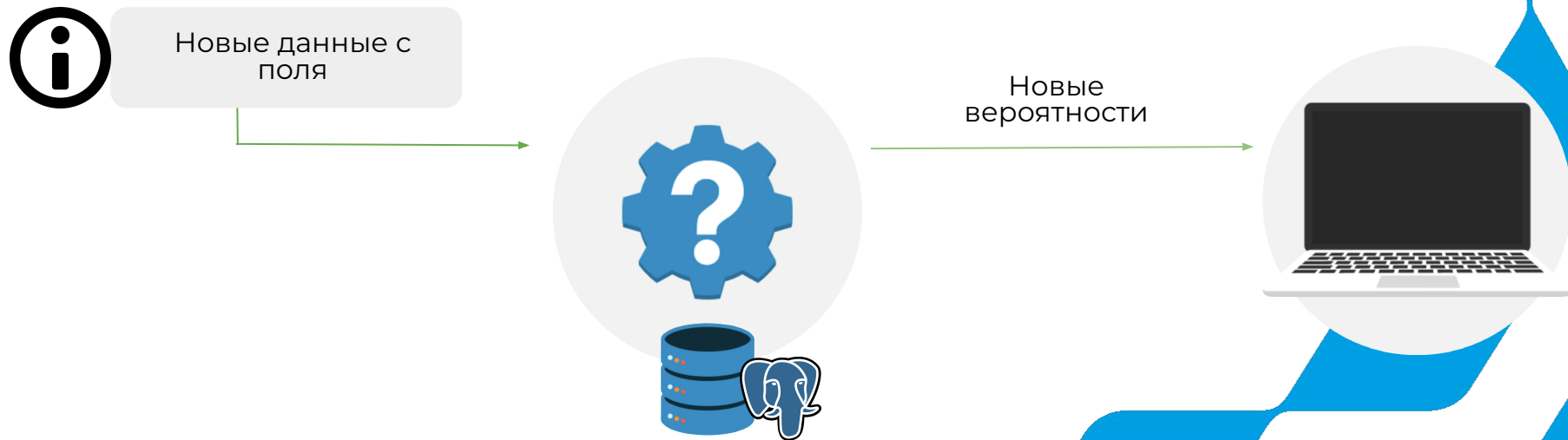
ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



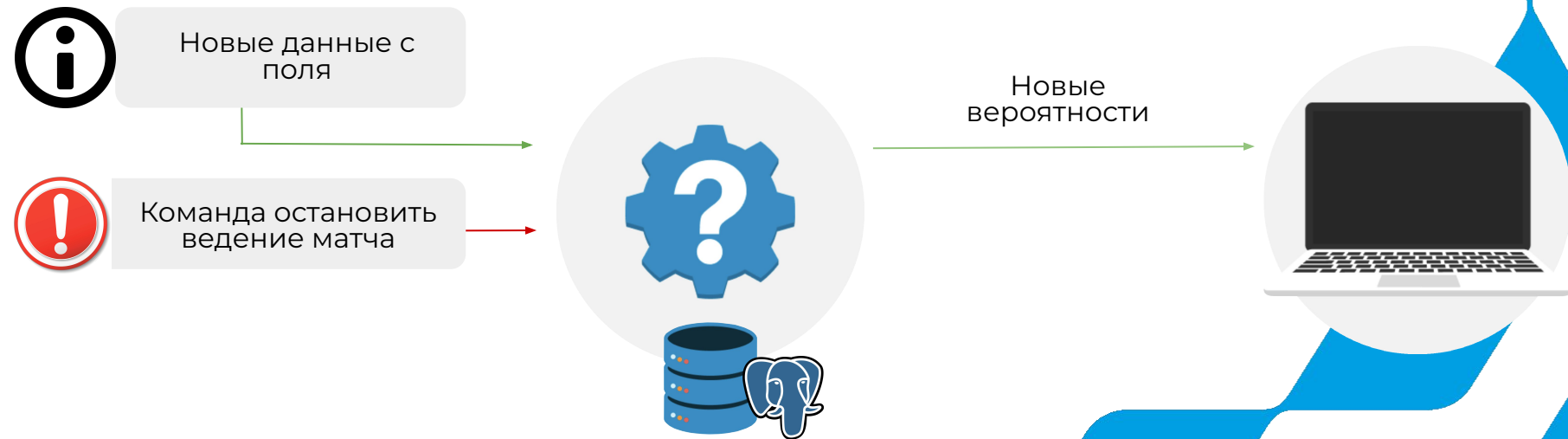
Новые данные с
поля



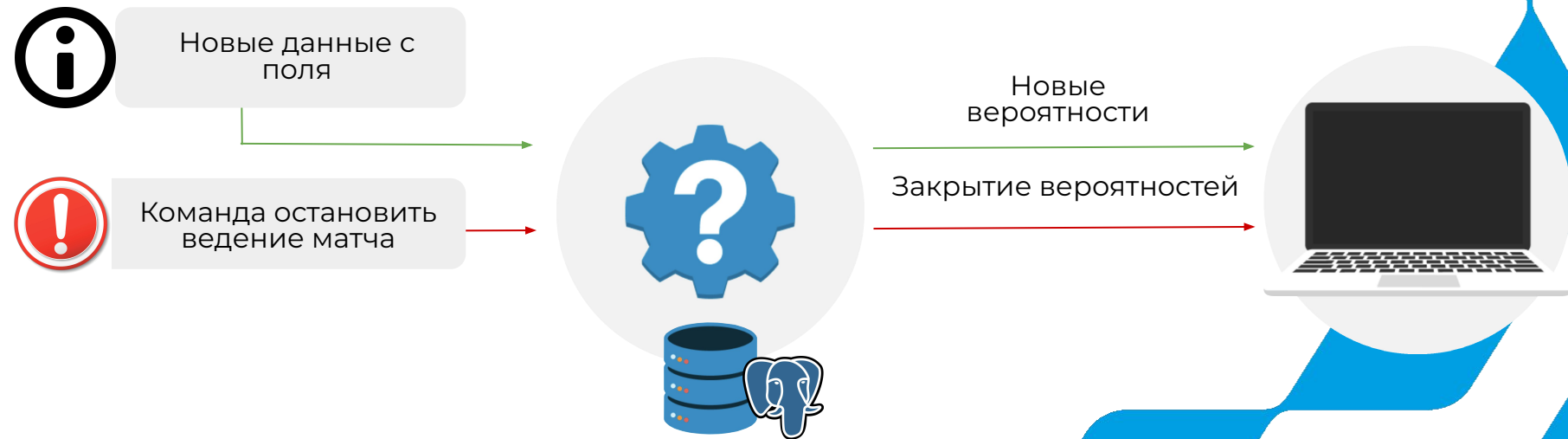
ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



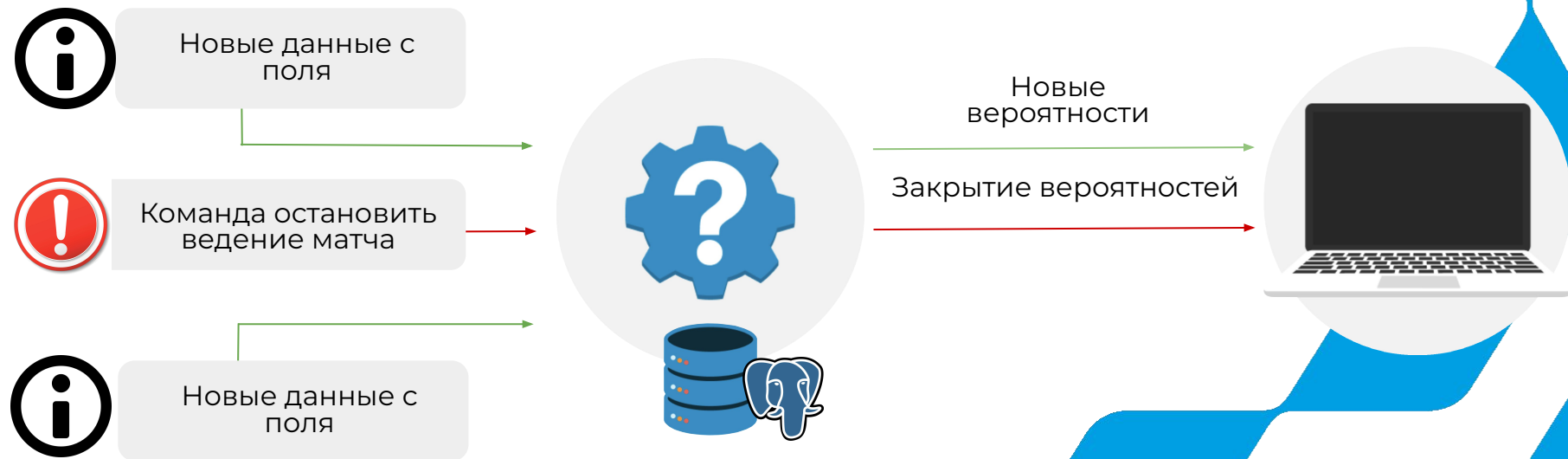
ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ

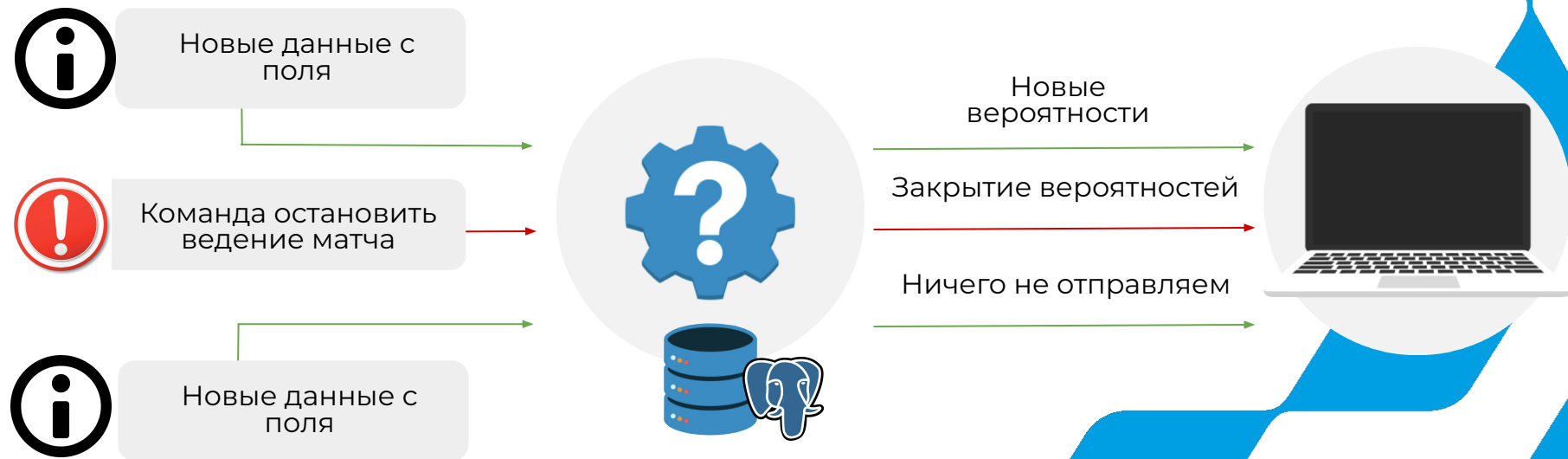


СХЕМА БАЗЫ ДАННЫХ



СХЕМА БАЗЫ ДАННЫХ

matches	
PK	match_id : bigint
	status : text (active / stopped)

СХЕМА БАЗЫ ДАННЫХ

matches	
PK	match_id : bigint
	status : text (active / stopped)



probabilities	
PK	probability_id : bigint
FK	match_id : bigint
	value: decimal

СХЕМА БАЗЫ ДАННЫХ

matches	
PK	match_id : bigint
	status : text (active / stopped)



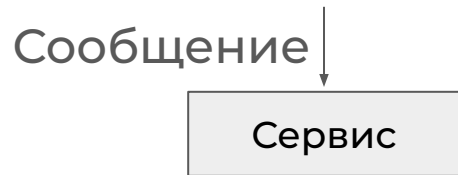
probabilities	
PK	probability_id : bigint
FK	match_id : bigint
	value: decimal

outbox	
PK	sequence_number : bigint
	match_id : bigint report : json

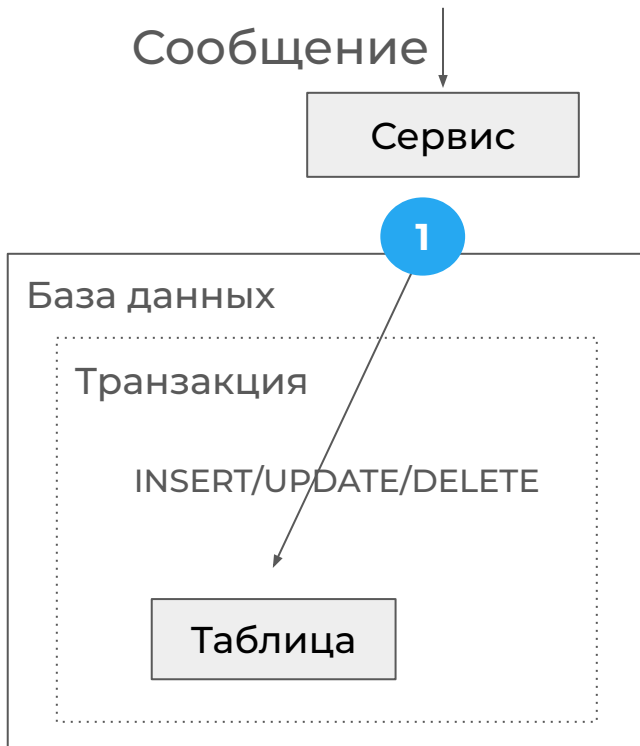


PATTERN “TRANSACTIONAL OUTBOX”

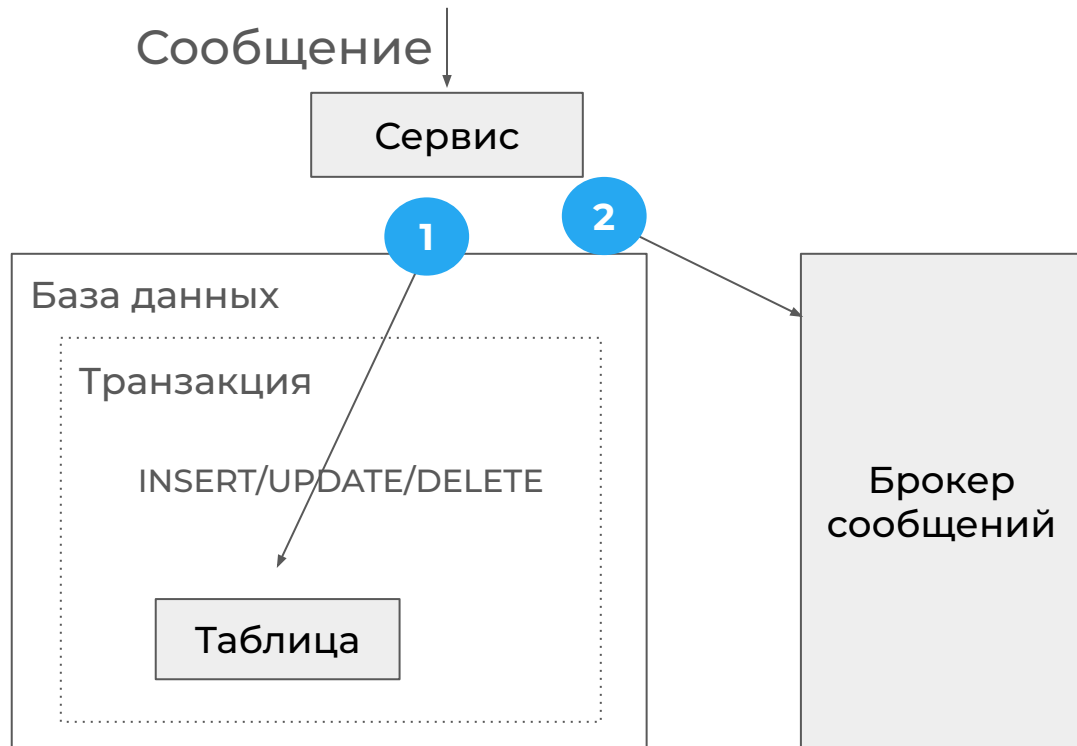
ПАТТЕРН “TRANSACTIONAL OUTBOX”



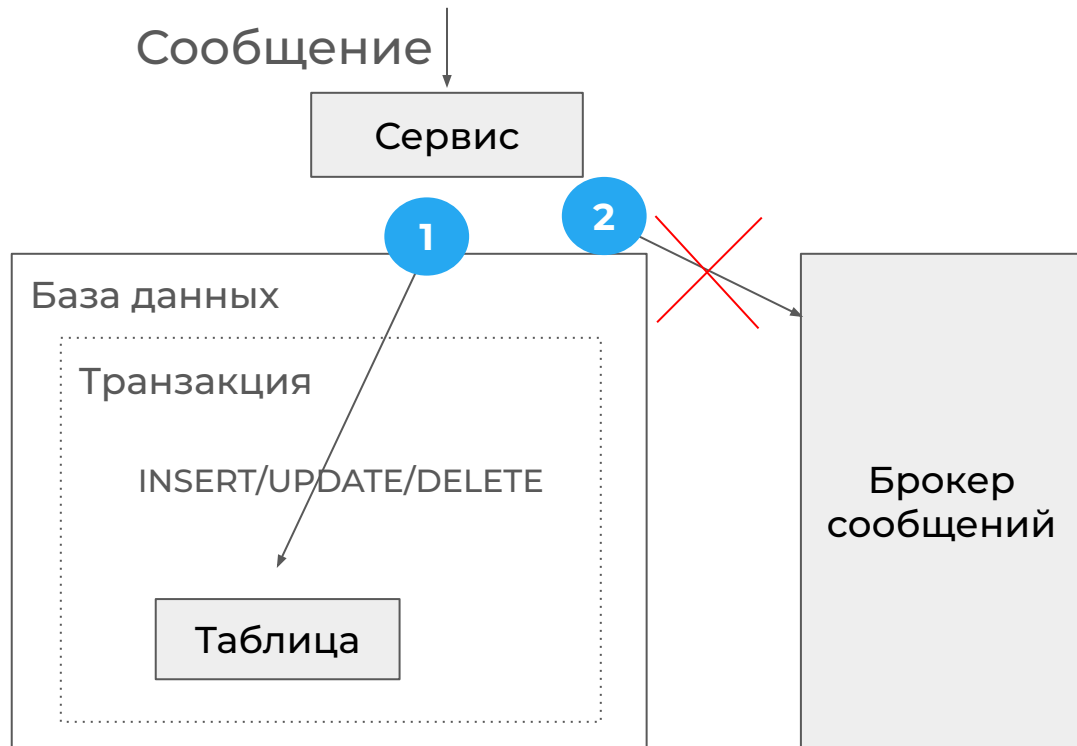
ПАТТЕРН “TRANSACTIONAL OUTBOX”



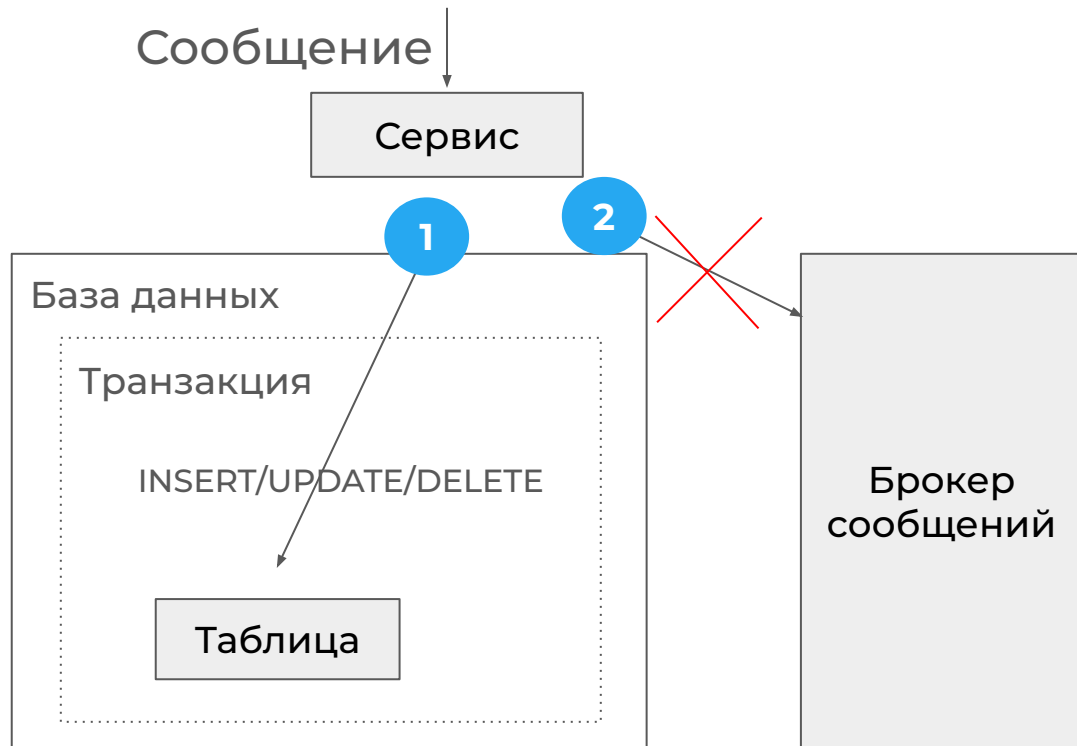
ПАТТЕРН “TRANSACTIONAL OUTBOX”



ПАТТЕРН “TRANSACTIONAL OUTBOX”



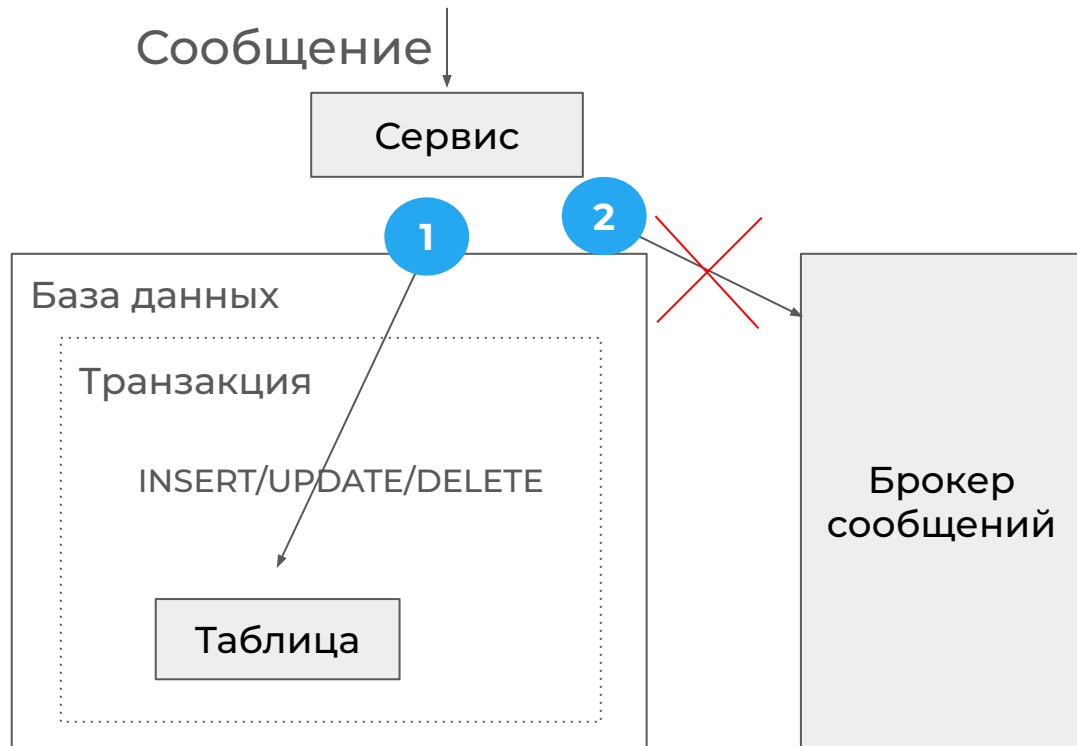
ПАТТЕРН “TRANSACTIONAL OUTBOX”



Проблема:

- Изменения в базе уже сохранились

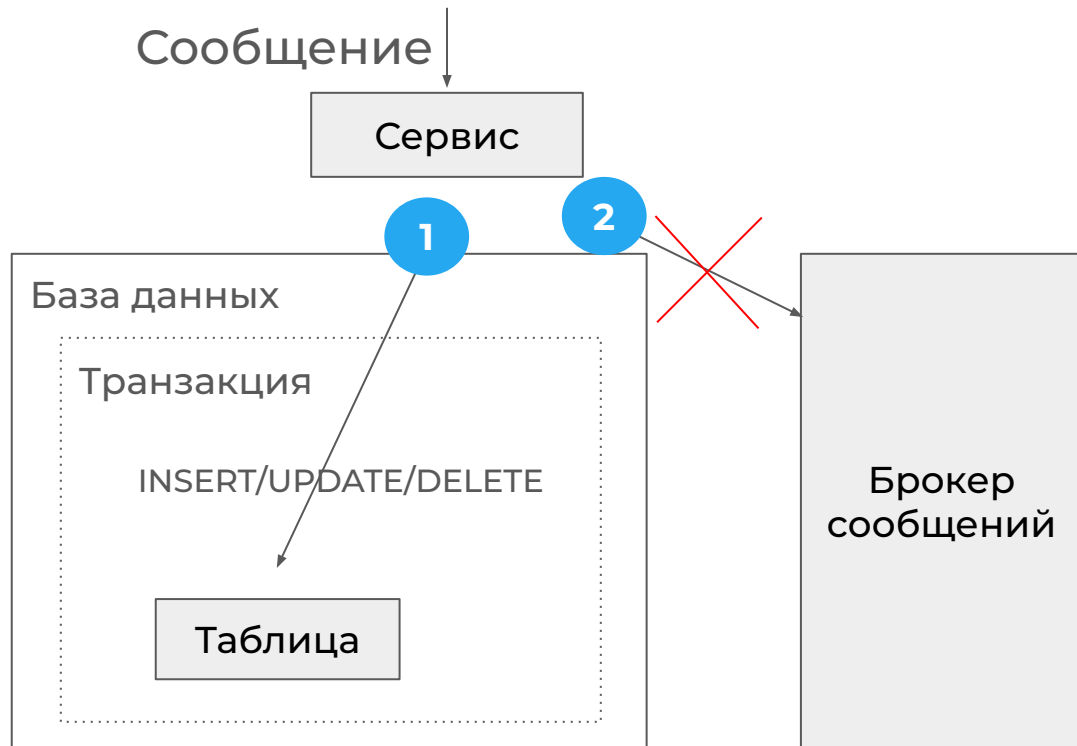
ПАТТЕРН “TRANSACTIONAL OUTBOX”



Проблема:

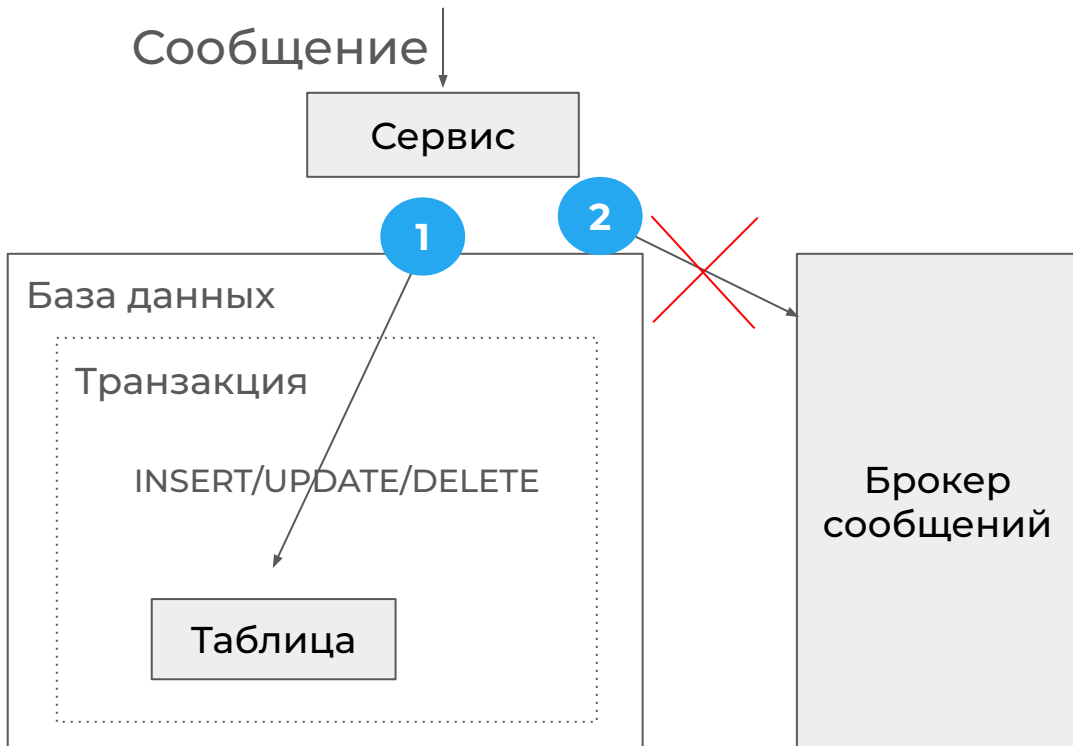
- Изменения в базе уже сохранились
- Сообщение дальше не отправилось

ПАТТЕРН “TRANSACTIONAL OUTBOX”



Возможные решения:

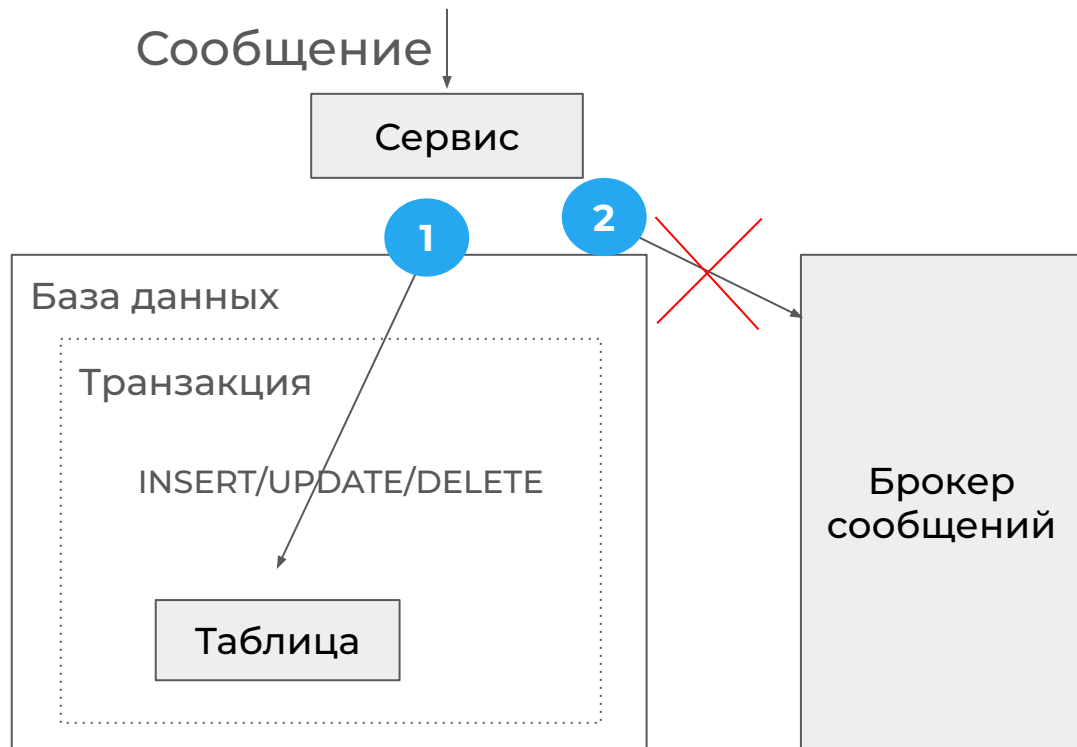
ПАТТЕРН “TRANSACTIONAL OUTBOX”



Возможные решения:

- Пытаться контролировать кейс в логике сервиса

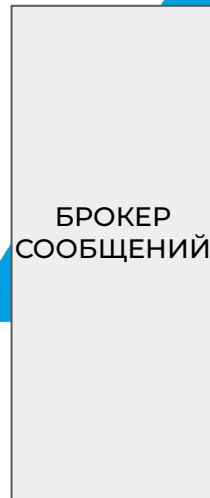
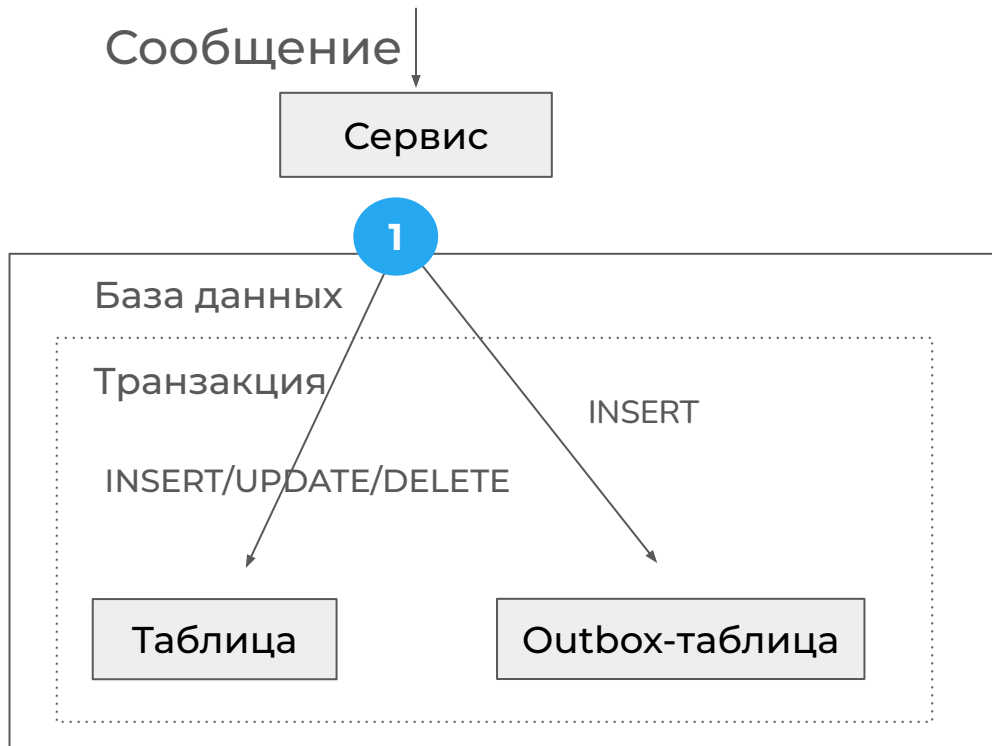
ПАТТЕРН “TRANSACTIONAL OUTBOX”



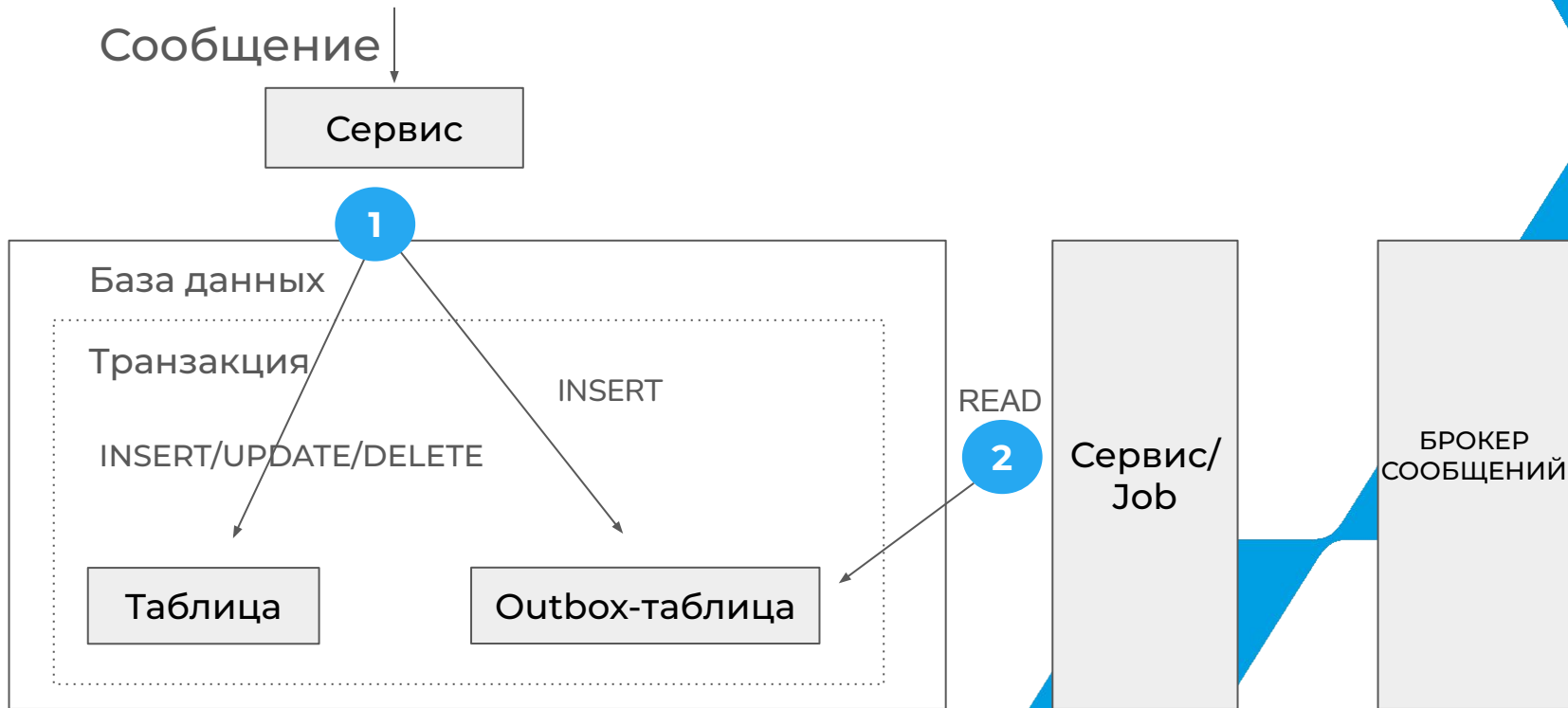
Возможные решения:

- Пытаться контролировать кейс в логике сервиса
- Использовать двухфазных коммитов между БД и брокером

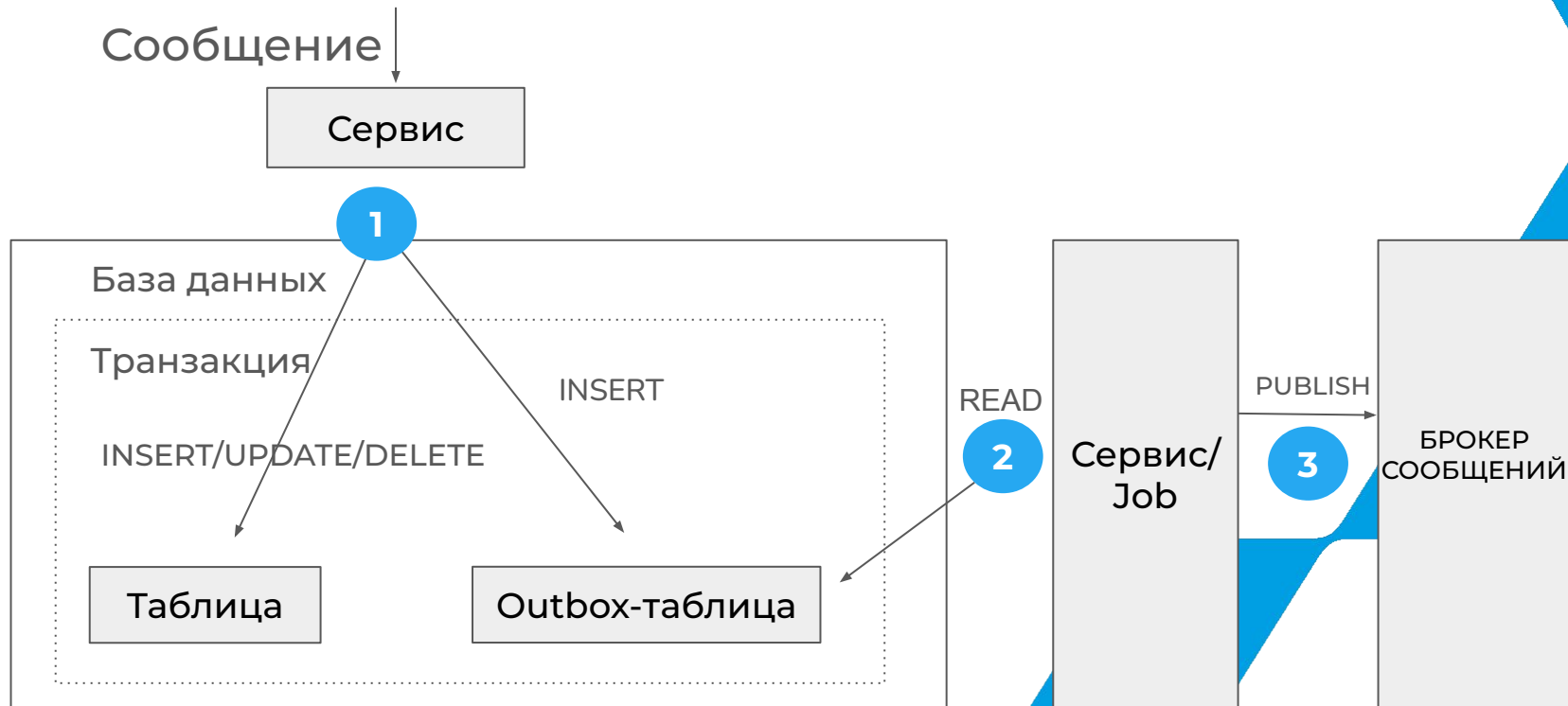
ПАТТЕРН “TRANSACTIONAL OUTBOX”



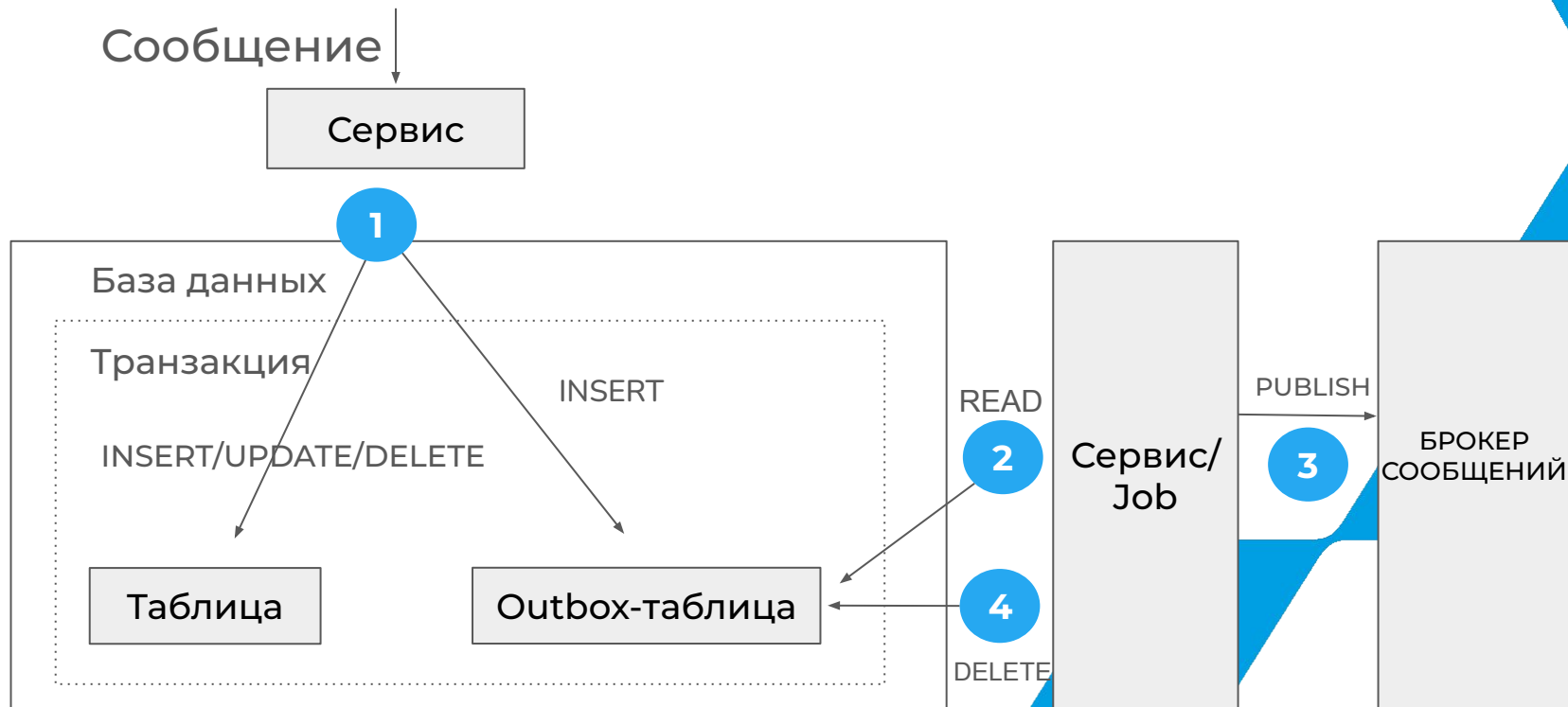
ПАТТЕРН “TRANSACTIONAL OUTBOX”



ПАТТЕРН “TRANSACTIONAL OUTBOX”



ПАТТЕРН “TRANSACTIONAL OUTBOX”



ПАТТЕРН “TRANSACTIONAL OUTBOX”

- Достигаем консистентного изменения данных в БД и формирования исходящего сообщения

ПАТТЕРН “TRANSACTIONAL OUTBOX”

- Достигаем консистентного изменения данных в БД и формирования исходящего сообщения
- Соблюдаем гарантию отправки “at least once”

ПАТТЕРН “TRANSACTIONAL OUTBOX”

- Достигаем консистентного изменения данных в БД и формирования исходящего сообщения
- Соблюдаем гарантию отправки “at least once”
- Достаточно легко обрабатываем недоступность брокера

СХЕМА БАЗЫ ДАННЫХ

matches	
PK	match_id : bigint
	status : text (active / stopped)



probabilities	
PK FK1 PK	match_id : bigint probability_id : bigint
	value: decimal

outbox	
PK	sequence_number : bigint
	match_id : bigint report : json

РЕАЛИЗАЦИЯ НА C#

```
public class Match
{
    public long MatchId { get; set; }

    public Status Status { get; set; }
}
```

```
public enum Status
{
    Active,
    Stopped
}
```

matches	
PK	match_id : bigint
	status : text (active / stopped)

РЕАЛИЗАЦИЯ НА C#

```
public class Probability
{
    public long MatchId { get; set; }

    public long ProbabilityId { get; set; }

    public decimal Value { get; set; }
}
```

probabilities	
PK	probability_id : bigint
FK	match_id : bigint
	value: decimal



РЕАЛИЗАЦИЯ НА C#

```
public class Outbox
{
    public long SequenceNumber
    { get; set; }

    public long MatchId { get; set; }

    public string Report { get; set; }
}
```

outbox	
PK	sequence_number : bigint
	match_id : bigint report : json



РЕАЛИЗАЦИЯ НА C#

```
public class ServiceDbContext(  
    DbContextOptions<ServiceDbContext> contextOptions)  
    : DbContext(contextOptions)  
{  
    public virtual DbSet<Match> Matches { get; set; }  
  
    public virtual DbSet<Probability> Probabilities { get; set; }  
  
    public virtual DbSet<Outbox> Outbox { get; set; }  
  
    ...  
}
```



РЕАЛИЗАЦИЯ НА C#

```
// Описывает операцию, например,  
// остановка ведения матча или обработка новых данных с поля  
public interface IOperation  
{  
    Task ExecuteAsync(CancellationToken token);  
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        await using var transaction =
            await dbContext.Database.BeginTransactionAsync(token);

        // Операция остановки матча или новых данных с поля
        await operation.ExecuteAsync(token);

        await dbContext.SaveChangesAsync(token);
        await transaction.CommitAsync(token);
    }
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        await using var transaction =
            await dbContext.Database.BeginTransactionAsync(token);

        // Операция остановки матча или новых данных с поля
        await operation.ExecuteAsync(token);

        await dbContext.SaveChangesAsync(token);
        await transaction.CommitAsync(token);
    }
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler (ServiceDbContext dbContext)
{
    public async Task HandleAsync (
        IOperation operation,
        CancellationToken token)
    {
        await using var transaction =
            await dbContext.Database.BeginTransactionAsync (token);

        // Операция остановки матча или новых данных с поля
        await operation.ExecuteAsync (token);

        await dbContext.SaveChangesAsync (token);
        await transaction.CommitAsync (token);
    }
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        await using var transaction =
            await dbContext.Database.BeginTransactionAsync(token);

        // Операция остановки матча или новых данных с поля
        await operation.ExecuteAsync(token);

        await dbContext.SaveChangesAsync(token);
        await transaction.CommitAsync(token);
    }
}
```



ТРАНЗАКЦИИ

Остановка ведения матча

0 BEGIN TRANSACTION;

Новые данные с поля

0 BEGIN TRANSACTION;



ТРАНЗАКЦИИ

Остановка ведения матча

```
0 BEGIN TRANSACTION;  
1 SELECT STATUS FROM matches  
   WHERE match_id = 10; //  
   'active'
```

Новые данные с поля

```
0 BEGIN TRANSACTION;  
2 SELECT STATUS FROM matches  
   WHERE match_id = 10; //  
   'active'
```



ТРАНЗАКЦИИ

Остановка ведения матча

- 0 BEGIN TRANSACTION;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities
WHERE match_id = 10;

Новые данные с поля

- 0 BEGIN TRANSACTION;
- 2 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'

ТРАНЗАКЦИИ

Остановка ведения матча

- 0 BEGIN TRANSACTION;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities
WHERE match_id = 10;
- 4 UPDATE matches
SET STATUS = 'stopped'
WHERE match_id = 10;

Новые данные с поля

- 0 BEGIN TRANSACTION;
- 2 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'

ТРАНЗАКЦИИ

Остановка ведения матча

- 0 BEGIN TRANSACTION;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities
WHERE match_id = 10;
- 4 UPDATE matches
SET STATUS = 'stopped'
WHERE match_id = 10;
- 5 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');
- 6 COMMIT;

Новые данные с поля

- 0 BEGIN TRANSACTION;
- 2 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'

ТРАНЗАКЦИИ

Остановка ведения матча

- 0 BEGIN TRANSACTION;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities
WHERE match_id = 10;
- 4 UPDATE matches
SET STATUS = 'stopped'
WHERE match_id = 10;
- 5 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');
- 6 COMMIT;

Новые данные с поля

- 0 BEGIN TRANSACTION;
- 2 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 7 UPDATE probabilities
SET value = 0.5
WHERE match_id = 10;

ТРАНЗАКЦИИ

Остановка ведения матча

- 0 BEGIN TRANSACTION;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities
WHERE match_id = 10;
- 4 UPDATE matches
SET STATUS = 'stopped'
WHERE match_id = 10;
- 5 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');
- 6 COMMIT;

Новые данные с поля

- 0 BEGIN TRANSACTION;
- 2 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 7 UPDATE probabilities
SET value = 0.5
WHERE match_id = 10;
- 8 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');

ТРАНЗАКЦИИ

Остановка ведения матча

- 0 BEGIN TRANSACTION;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities
WHERE match_id = 10;
- 4 UPDATE matches
SET STATUS = 'stopped'
WHERE match_id = 10;
- 5 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');
- 6 COMMIT;

Новые данные с поля

- 0 BEGIN TRANSACTION;
- 2 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 7 UPDATE probabilities
SET value = 0.5
WHERE match_id = 10;
- 8 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');
- 9 COMMIT;

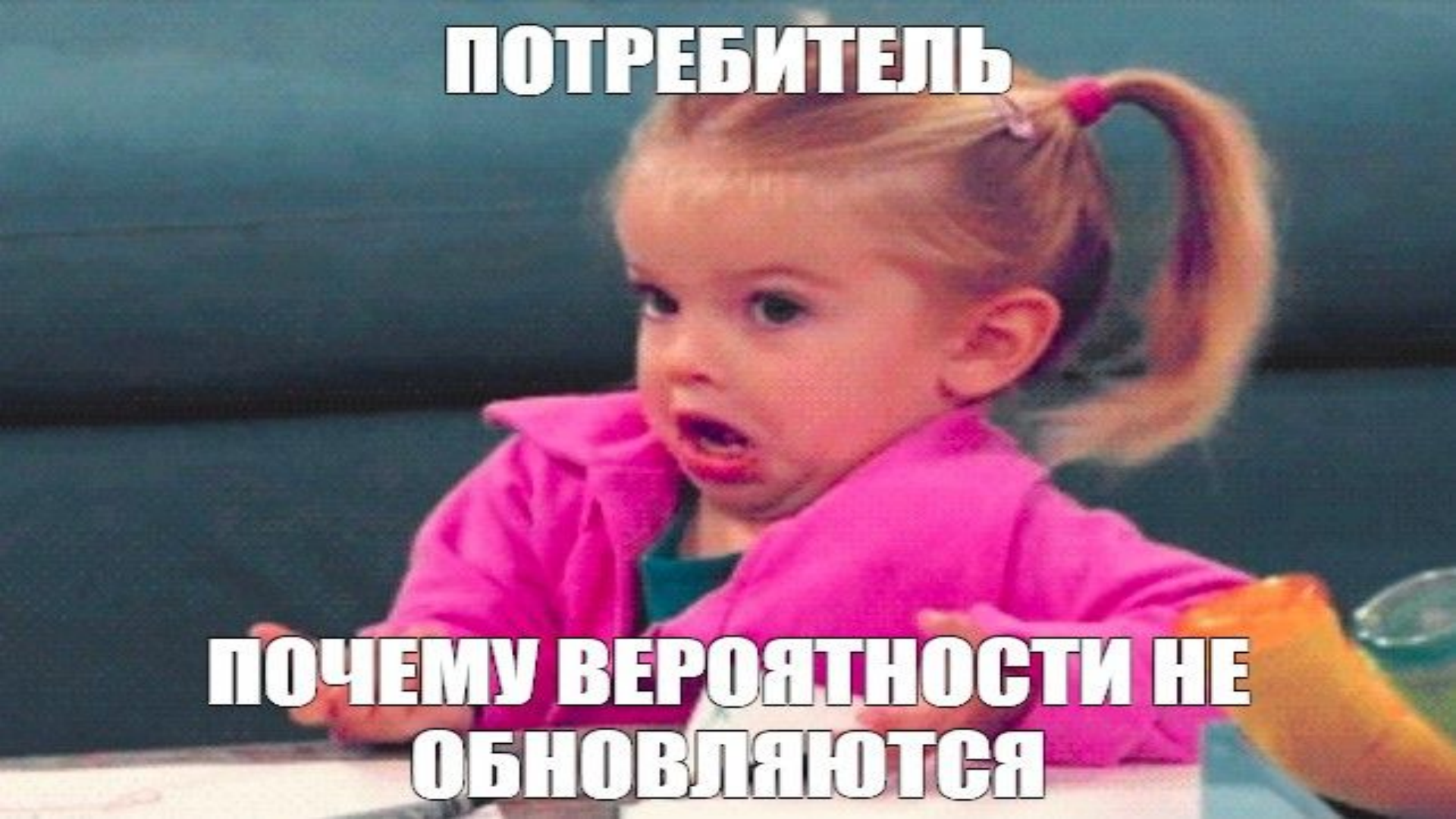
РЕЗУЛЬТАТЫ ОПЕРАЦИЙ

outbox		
sequence_number	match_id	report
1	10	Закрытие вероятностей
2	10	Новые вероятности

ПЛОХО

ПОТРЕБИТЕЛЬ

**ПОЧЕМУ ВЕРОЯТНОСТИ НЕ
ОБНОВЛЯЮТСЯ**



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ

Остановка ведения матча и обработка
новых данных с поля могут начаться
одновременно и использовать
одинаковые данные из базы



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ

Остановка ведения матча и обработка новых данных с поля могут начаться одновременно и использовать одинаковые данные из базы

Остановка ведения может завершиться раньше, чем производство новых вероятностей



ЗАДАЧА, КОТОРУЮ МЫ РЕШАЕМ

Остановка ведения матча и обработка новых данных с поля могут начаться одновременно и использовать одинаковые данные из базы

Остановка ведения может завершиться раньше, чем производство новых вероятностей

Параллельные операции могут выполняться в разных инстансах приложения



ВЫБРАННОЕ РЕШЕНИЕ



ВАРИАНТЫ СИНХРОНИЗАЦИИ НА УРОВНЕ БД



ВАРИАНТЫ СИНХРОНИЗАЦИИ НА УРОВНЕ БД

Ручные блокировки в БД
(например, на строки или таблицы)

ВАРИАНТЫ СИНХРОНИЗАЦИИ НА УРОВНЕ БД

Ручные блокировки в БД
(например, на строки или таблицы)

Рекомендательные блокировки в
БД (Advisory lock)

ВАРИАНТЫ СИНХРОНИЗАЦИИ НА УРОВНЕ БД

Ручные блокировки в БД
(например, на строки или таблицы)

Рекомендательные блокировки в
БД (Advisory lock)

Использование различных уровней
изолированности транзакций

РУЧНЫЕ БЛОКИРОВКИ В БД (например, на строки или таблицы)

- + Есть в PostgreSQL (например, `SELECT FOR UPDATE`)



РУЧНЫЕ БЛОКИРОВКИ В БД (например, на строки или таблицы)

- + Есть в PostgreSQL (например, `SELECT FOR UPDATE`)
- + Минимальные накладные расходы на производительность



РУЧНЫЕ БЛОКИРОВКИ В БД (например, на строки или таблицы)

- + Есть в PostgreSQL (например, `SELECT FOR UPDATE`)
 - + Минимальные накладные расходы на производительность
-
- Не самое простое в реализации (нет поддержки в ORM-фреймворках)

РУЧНЫЕ БЛОКИРОВКИ В БД (например, на строки или таблицы)

- + Есть в PostgreSQL (например, `SELECT FOR UPDATE`)
 - + Минимальные накладные расходы на производительность
-
- Не самое простое в реализации (нет поддержки в ORM-фреймворках)
 - Повышенная внимательность к коду



РЕКОМЕНДАТЕЛЬНЫЕ БЛОКИРОВКИ В БД (Advisory Lock)

- + Есть в PostgreSQL (Advisory lock)

РЕКОМЕНДАТЕЛЬНЫЕ БЛОКИРОВКИ В БД (Advisory Lock)

- + Есть в PostgreSQL (Advisory lock)
- + Достаточно простое в реализации

РЕКОМЕНДАТЕЛЬНЫЕ БЛОКИРОВКИ В БД (Advisory Lock)

- + Есть в PostgreSQL (Advisory lock)
- + Достаточно простое в реализации
- + Обладает высокой производительностью

РЕКОМЕНДАТЕЛЬНЫЕ БЛОКИРОВКИ В БД (Advisory Lock)

- + Есть в PostgreSQL (Advisory lock)
 - + Достаточно простое в реализации
 - + Обладает высокой производительностью
-
- Повышенная внимательность к коду

РЕКОМЕНДАТЕЛЬНЫЕ БЛОКИРОВКИ В БД (Advisory Lock)

- + Есть в PostgreSQL (Advisory lock)
 - + Достаточно простое в реализации
 - + Обладает высокой производительностью
-
- Повышенная внимательность к коду
 - Мы упустили этот вариант изначально

ИСПОЛЬЗОВАНИЕ РАЗЛИЧНЫХ УРОВНЕЙ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

+ Есть в PostgreSQL



ИСПОЛЬЗОВАНИЕ РАЗЛИЧНЫХ УРОВНЕЙ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

- + Есть в PostgreSQL
- + Достаточно простое в реализации



ИСПОЛЬЗОВАНИЕ РАЗЛИЧНЫХ УРОВНЕЙ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

- + Есть в PostgreSQL
- + Достаточно простое в реализации
- + Не требует повышенного внимания к коду сторону разработчика



ИСПОЛЬЗОВАНИЕ РАЗЛИЧНЫХ УРОВНЕЙ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

- + Есть в PostgreSQL
 - + Достаточно простое в реализации
 - + Не требует повышенного внимания к коду сторону разработчика
-
- Снижает производительность запросов



УРОВНИ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

Read uncommitted (нет в PostgreSQL)

Read committed

Repeatable read

Serializable

Аномалия — нарушение целостности данных в результате выполнения нескольких параллельных транзакций



УРОВНИ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

Уровень	“Грязное” чтение	Неповторяемое чтение	Фантомное чтение	Аномалия сериализации
Read uncommitted (нет в PostgreSQL)	Возможно, но не в PostgreSQL	Возможно	Возможно	Возможно
Read committed	Невозможно	Возможно	Возможно	Возможно
Repeatable read	Невозможно	Невозможно	Возможно, но не в PostgreSQL	Возможно
Serializable	Невозможно	Невозможно	Невозможно	Невозможно

УРОВНИ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

Уровень	“Грязное” чтение	Неповторяемое чтение	Фантомное чтение	Аномалия сериализации
Read uncommitted (нет в PostgreSQL)	Возможно, но не в PostgreSQL	Возможно	Возможно	Возможно
Read committed	Невозможно	Возможно	Возможно	Возможно
Repeatable read	Невозможно	Невозможно	Возможно, но не в PostgreSQL	Возможно
Serializable	Невозможно	Невозможно	Невозможно	Невозможно

SERIALIZABLE B PostgreSQL



Serializable в PostgreSQL

Академический подход - SSI
(Serializable Snapshot Isolation) от 2008 г.
Появился в PostgreSQL в 2012 г.

Serializable в PostgreSQL

Академический подход - SSI
(Serializable Snapshot Isolation) от 2008 г.
Появился в PostgreSQL в 2012 г.

Эмулирует последовательное
исполнение транзакций, но
не блокирует их



Serializable в PostgreSQL

Академический подход - SSI
(Serializable Snapshot Isolation) от 2008 г.
Появился в PostgreSQL в 2012 г.

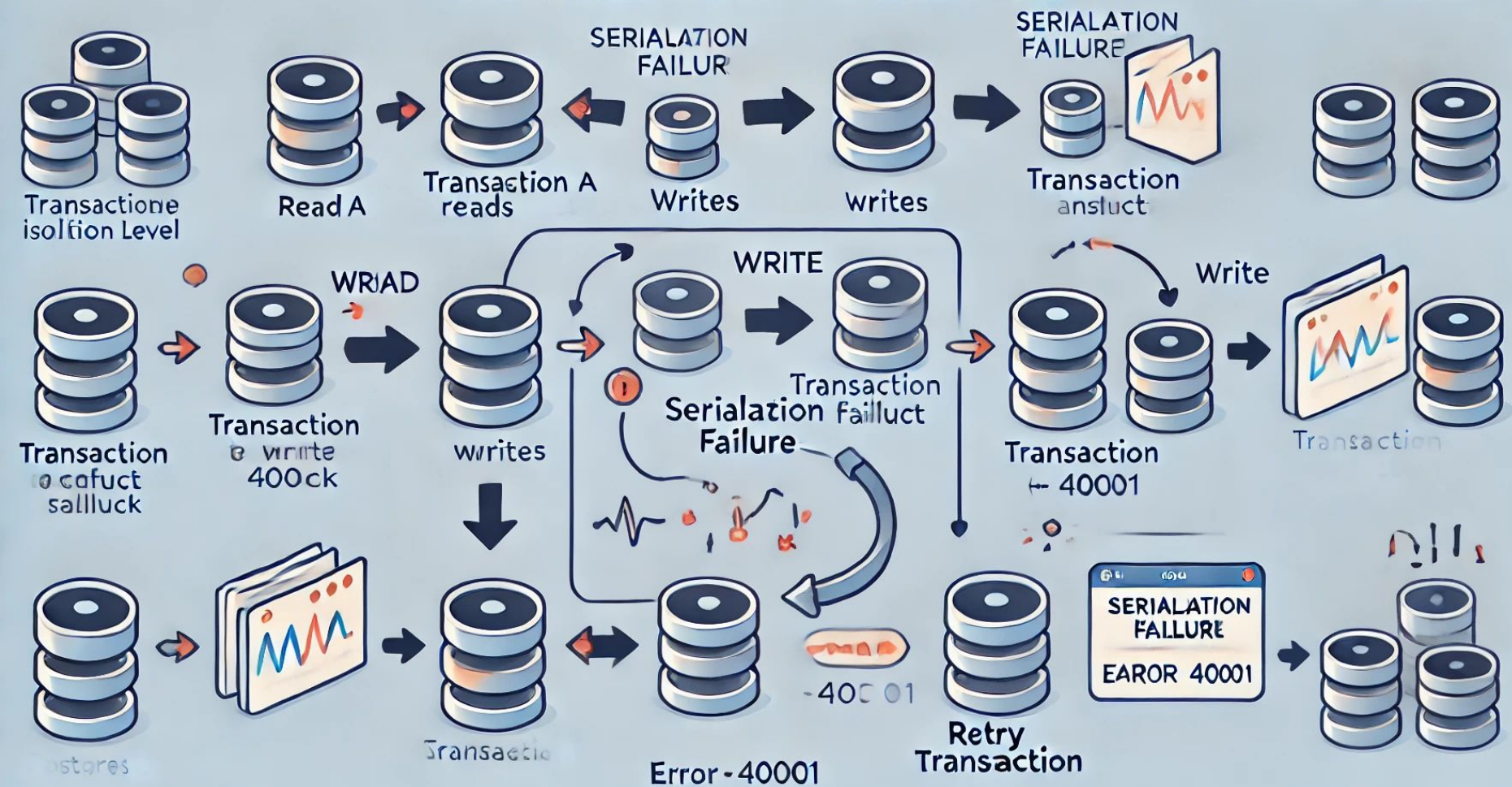
Эмулирует последовательное
исполнение транзакций, но
не блокирует их

Использует `predicate_lock`'и для
отслеживания возможных
аномалий

SERIALIZABLE в PostgreSQL

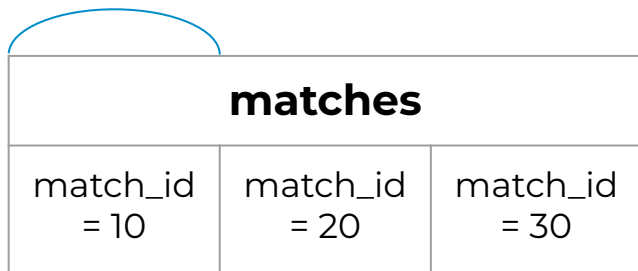


- PostGresSL



Serializable в PostgreSQL

Predicate
lock 1




matches		
match_id = 10	match_id = 20	match_id = 30

- Чтение создает “predicate_lock” на прочитанные данные



Serializable в PostgreSQL

Predicate
lock 1

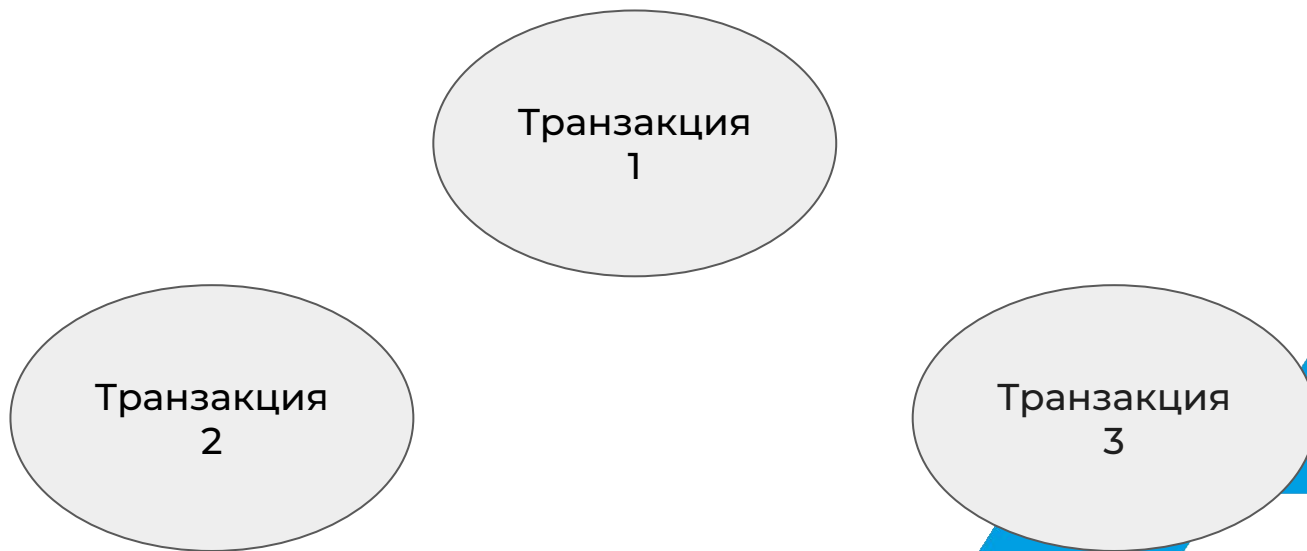


matches		
match_id = 10	match_id = 20	match_id = 30

- Чтение создает “predicate_lock” на прочитанные данные
- Модификация данных, на которые наложен “predicate_lock” вызывает конфликт

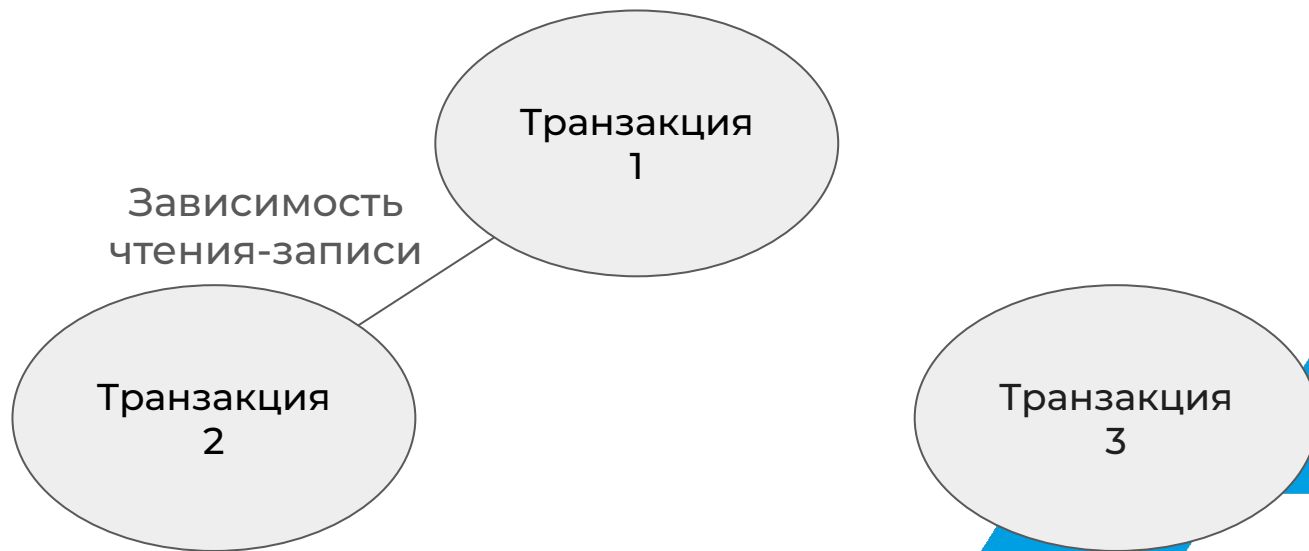
Serializable в PostgreSQL

Граф зависимостей



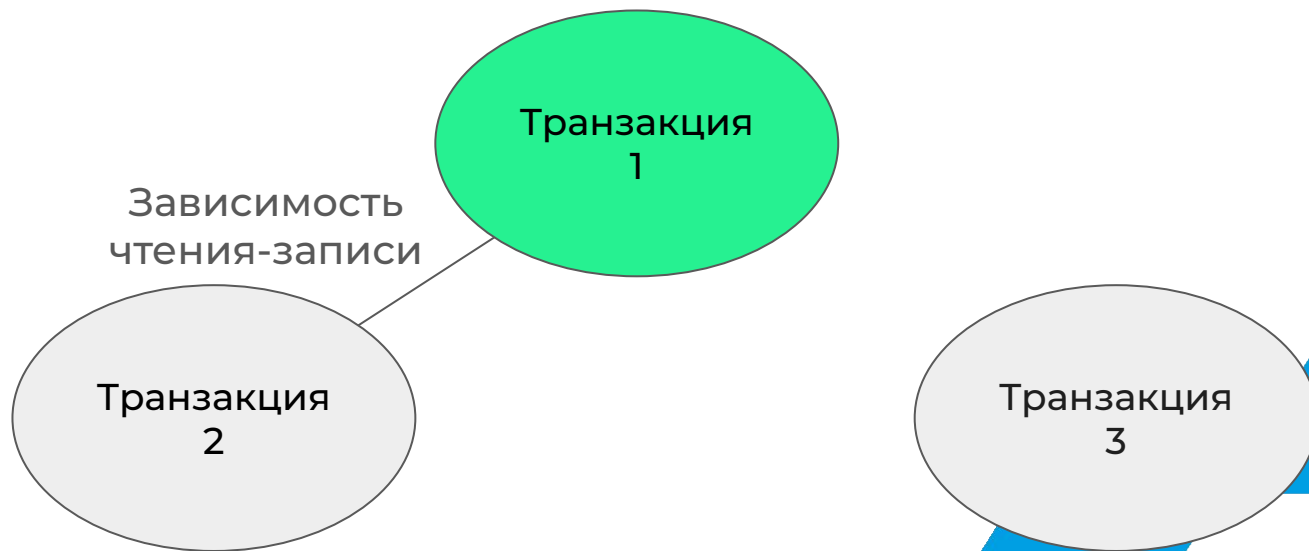
Serializable в PostgreSQL

Граф зависимостей



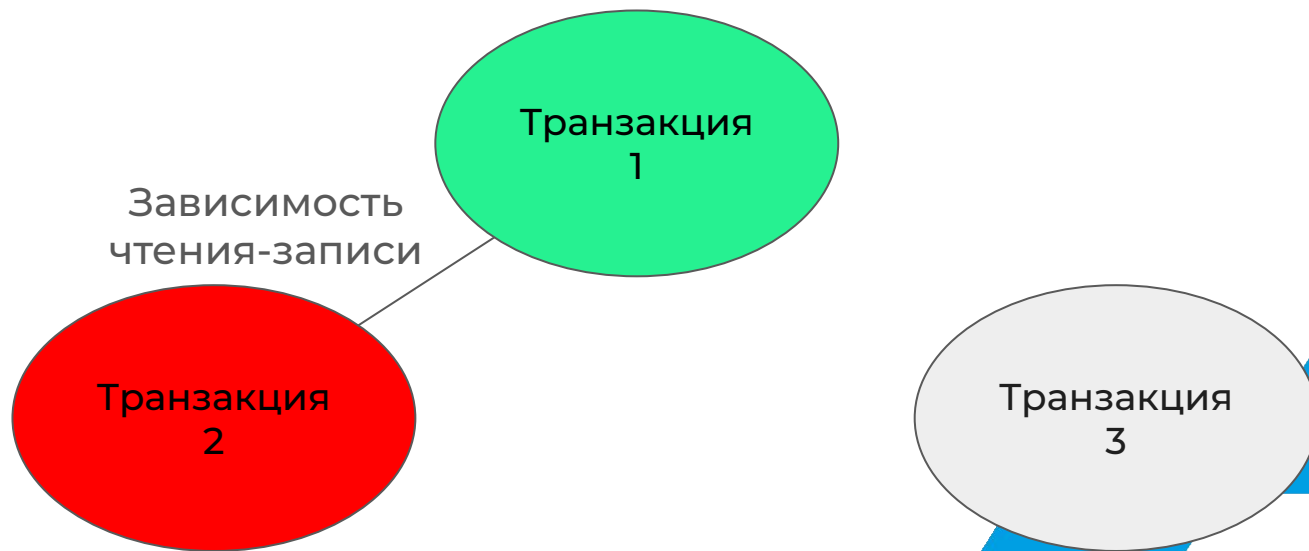
Serializable в PostgreSQL

Граф зависимостей



Serializable в PostgreSQL

Граф зависимостей



SERIALIZABLE. ПРОИЗВОДИТЕЛЬНОСТЬ



Serializable.

Производительность

- 2000 TPM (~ 35 TPS) нагруженных изменяющих данные транзакций (~250 футбольных матчей с важным сообщением раз в 10 секунд)

Serializable.

Производительность

- 2000 TPM (~ 35 TPS) нагруженных изменяющих данные транзакций (~250 футбольных матчей с важным сообщением раз в 10 секунд)
- Время выполнения транзакций с Serializable ~100 ms

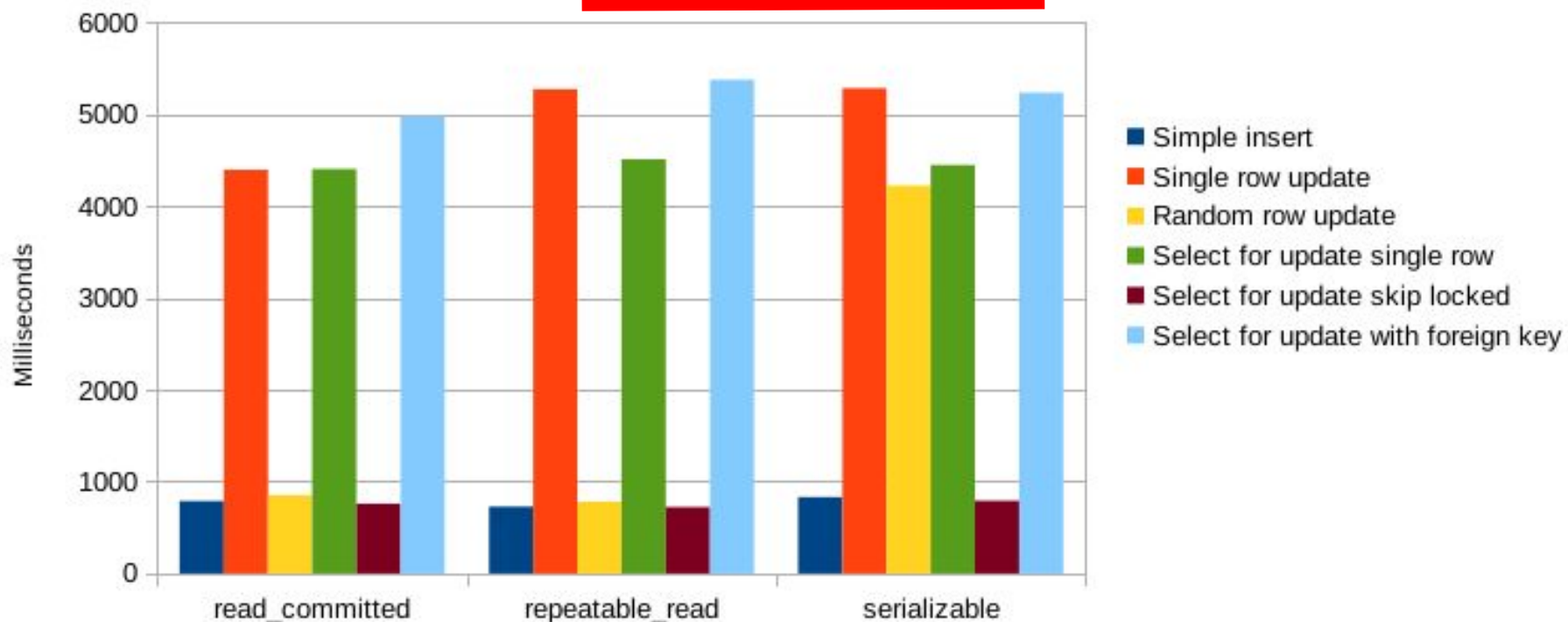
Serializable.

Производительность

- 2000 TPM (~ 35 TPS) нагруженных изменяющих данные транзакций (~250 футбольных матчей с важным сообщением раз в 10 секунд)
- Время выполнения транзакций с Serializable ~100 ms
- Время выполнения с ReadCommitted + SELECT FOR UPDATE ~80 ms

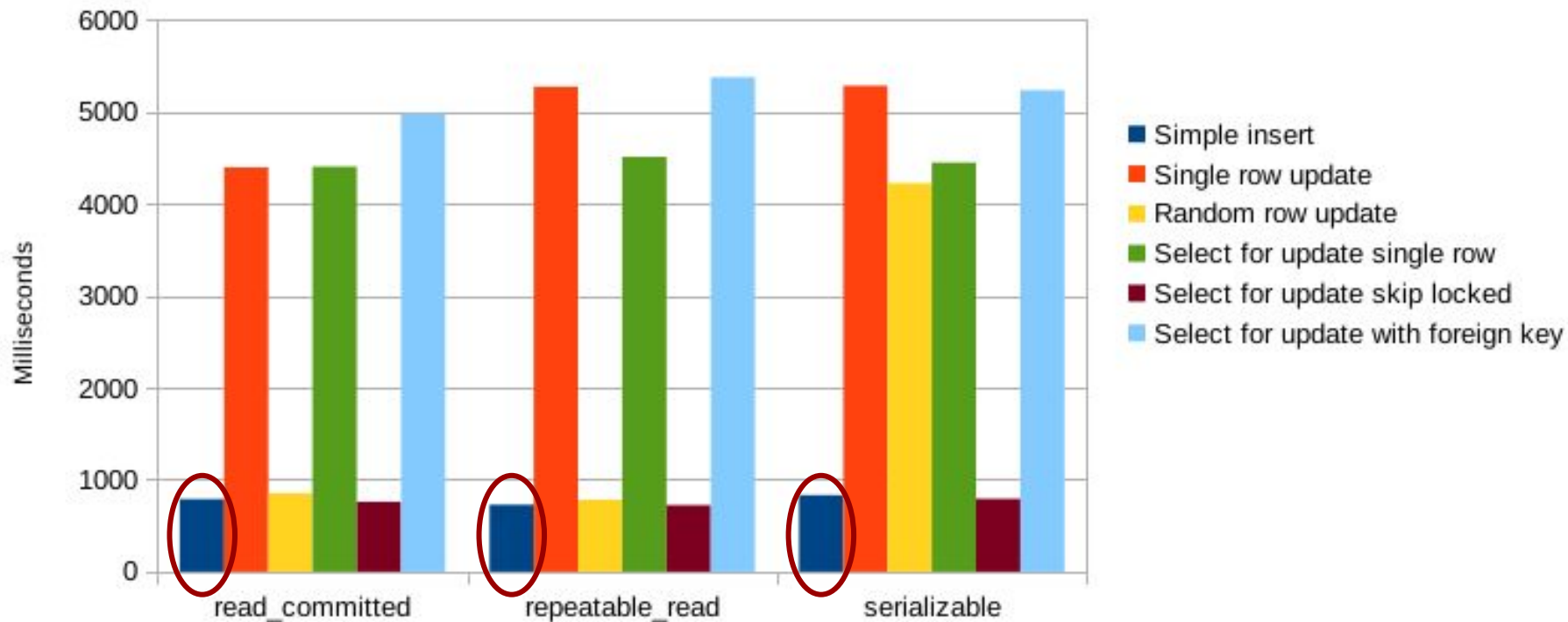
Tests at different transaction isolation levels

1600 threads, 10 connections



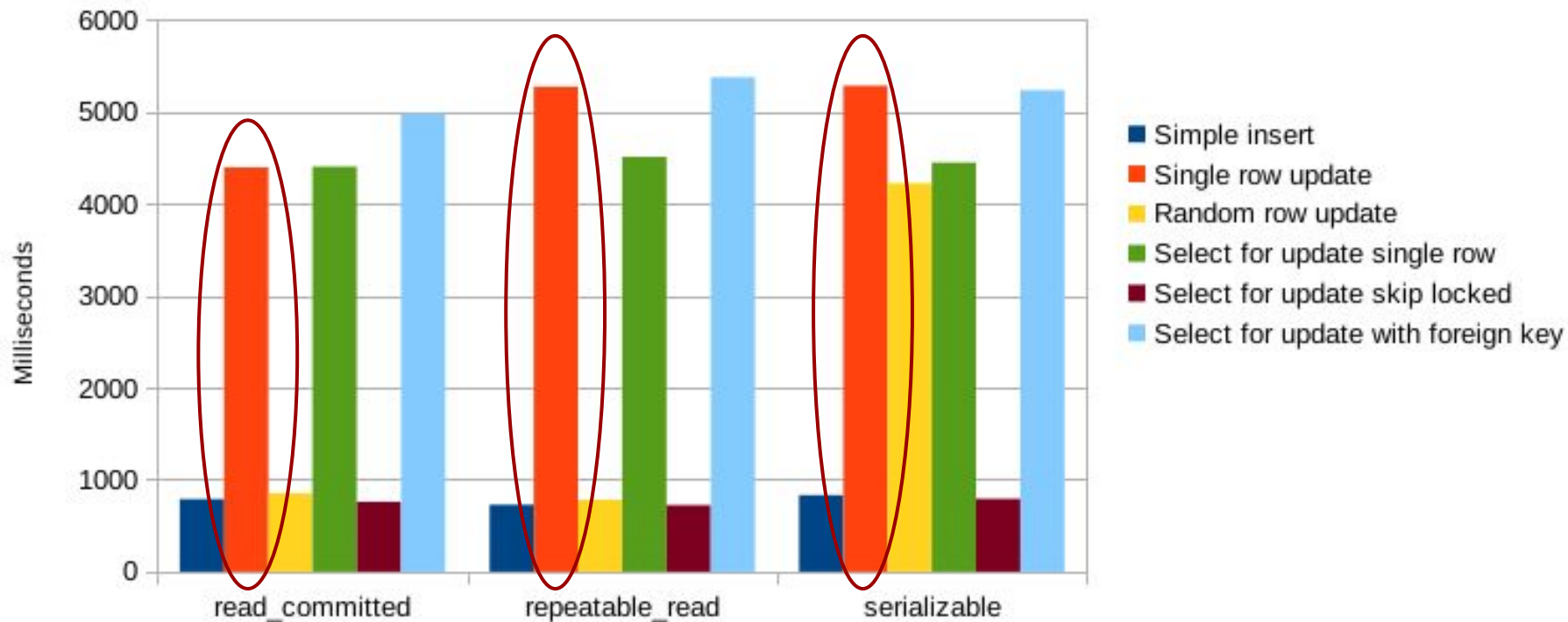
Tests at different transaction isolation levels

1600 threads, 10 connections



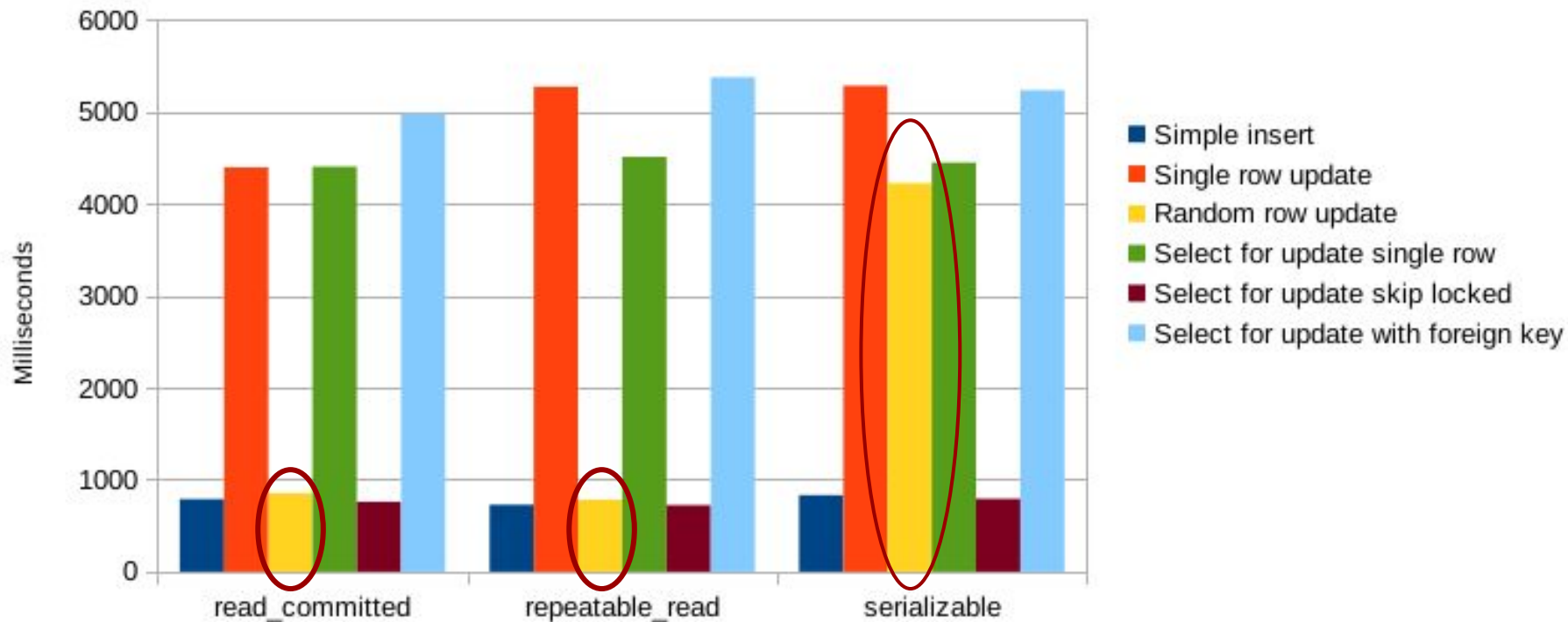
Tests at different transaction isolation levels

1600 threads, 10 connections



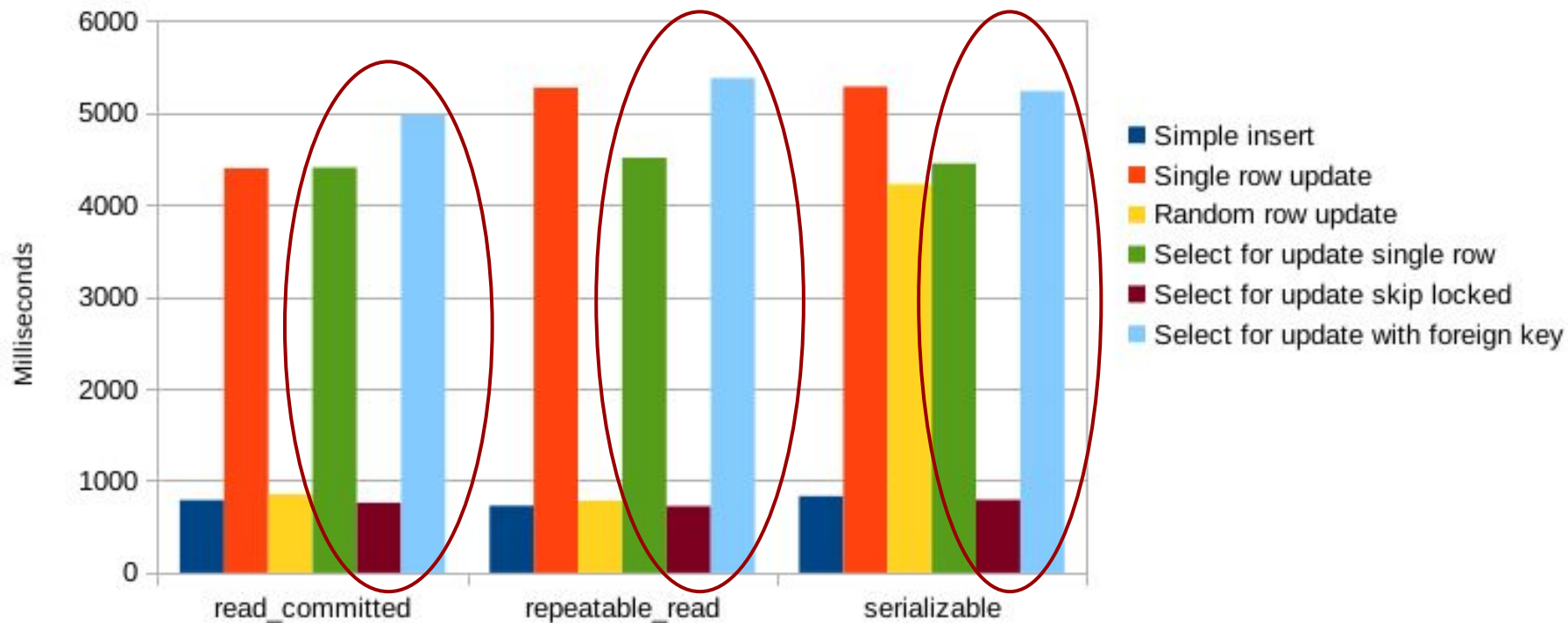
Tests at different transaction isolation levels

1600 threads, 10 connections



Tests at different transaction isolation levels

1600 threads, 10 connections



Serializable.

Производительность

- Производительность снижается относительно других уровней изолированности при обновлении данных
- Производительность чтения почти не меняется
- Полезно провести свои нагрузочные тесты, чтобы определить будет ли снижение критично для вашего проекта

ВАЖНЫЕ МОМЕНТЫ, КОТОРЫЕ НУЖНО УЧИТЫВАТЬ ПРИ ИСПОЛЬЗОВАНИИ SERIALIZABLE



ВАЖНЫЕ МОМЕНТЫ, КОТОРЫЕ НУЖНО УЧИТЫВАТЬ ПРИ ИСПОЛЬЗОВАНИИ Serializable

Все транзакции должны
использовать Serializable уровень
изоляции

ВАЖНЫЕ МОМЕНТЫ, КОТОРЫЕ НУЖНО УЧИТЫВАТЬ ПРИ ИСПОЛЬЗОВАНИИ Serializable

Все транзакции должны
использовать Serializable уровень
изолированности

Транзакции, которые читают
данные, желательно пометить READ
ONLY

ВАЖНЫЕ МОМЕНТЫ, КОТОРЫЕ НУЖНО УЧИТЫВАТЬ ПРИ ИСПОЛЬЗОВАНИИ Serializable

Все транзакции должны
использовать Serializable уровень
изолированности

Транзакции, которые читают
данные, желательно пометить READ
ONLY

Приложение должно быть готово
повторять транзакции при ошибках
от PostgreSQL

ВАЖНЫЕ МОМЕНТЫ, КОТОРЫЕ НУЖНО УЧИТЫВАТЬ ПРИ ИСПОЛЬЗОВАНИИ Serializable

[40001] ОШИБКА: не удалось
сериализовать доступ из-за
зависимостей чтения/записи
между транзакциями

Hint: Транзакция может
завершиться успешно при
следующей попытке.

РЕАЛИЗАЦИЯ НА C#

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        await using var transaction =
            await dbContext.Database.BeginTransactionAsync(
                IsolationLevel.Serializable,
                token);

        // Операция остановки матча или новых данных с поля
        await operation.ExecuteAsync(token);

        await dbContext.SaveChangesAsync(token);
        await transaction.CommitAsync(token);
    }
}
```


РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        await ExecuteOperationAsync(operation, token);
    }
}
```

```
private async Task ExecuteOperationAsync(
    IOperation operation, CancellationToken token)
{
    await using var transaction = await dbContext.Database.BeginTransactionAsync(
        IsolationLevel.Serializable, token);
    // Операция остановки матча или новых данных с поля
    await operation.ExecuteAsync(token);
    await dbContext.SaveChangesAsync(token);
    await transaction.CommitAsync(token);
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        await ExecuteOperationAsync(operation, token);
    }
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext, ILogger<OperationHandler> logger)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        var attempt = 0;
        while (attempt < 3)
        {
            try
            {
                await ExecuteOperationAsync(operation, token);
                return;
            }
            catch (Exception ex) when (IsConcurrentException(ex))
            {
                logger.LogWarning(ex, "Concurrency detected");
                attempt++;
            }
        }
    }
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext, ILogger<OperationHandler> logger)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        var attempt = 0;
        while (attempt < 3)
        {
            try
            {
                await ExecuteOperationAsync(operation, token);
                return;
            }
            catch (Exception ex) when (IsConcurrentException(ex))
            {
                logger.LogWarning(ex, "Concurrency detected");
                attempt++;
            }
        }
    }
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext, ILogger<OperationHandler> logger)
{
    ...

    private static bool IsConcurrentException(Exception e)
    {
        ...
    }
}
```



РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext, ILogger<OperationHandler> logger)
{
    ...

    private static bool IsConcurrentException(Exception e)
    {
        if (e.InnerException is not DbUpdateException dbUpdateException)
        {
            return false;
        }

        ...
    }
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext, ILogger<OperationHandler> logger)
{
    ...

    private static bool IsConcurrentException(Exception e)
    {
        if (e.InnerException is not DbUpdateException dbUpdateException)
        {
            return false;
        }

        if (dbUpdateException.InnerException is not PostgresException pgEx)
        {
            return false;
        }

        ...
    }
}
```

РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext, ILogger<OperationHandler> logger)
{
    ...

    private static bool IsConcurrentException(Exception e)
    {
        if (e.InnerException is not DbUpdateException dbUpdateException)
        {
            return false;
        }

        if (dbUpdateException.InnerException is not PostgresException pgEx)
        {
            return false;
        }

        return pgEx.SqlState == "40001";
    }
}
```


РЕАЛИЗАЦИЯ НА C#

```
public class OperationHandler(ServiceDbContext dbContext, ILogger<OperationHandler> logger)
{
    public async Task HandleAsync(
        IOperation operation,
        CancellationToken token)
    {
        var attempt = 0;
        while (attempt < 3)
        {
            try
            {
                await ExecuteOperationAsync(operation, token);
                return;
            }
            catch (Exception ex) when (IsConcurrentException(ex))
            {
                logger.LogWarning(ex, "Concurrency detected");
                attempt++;
            }
        }
    }
}
```

РЕАЛИЗАЦИЯ НА C#

- Добавили уровень изолированности Serializable во все транзакции
- Поддержали логику повторения запросов

ТРАНЗАКЦИИ C Serializable

Остановка ведения матча

0

```
BEGIN TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;
```

Новые данные с поля

0

```
BEGIN TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;
```

ТРАНЗАКЦИИ C Serializable

Остановка ведения матча

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`

Новые данные с поля

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`

ТРАНЗАКЦИИ C Serializable

Остановка ведения матча

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `SELECT * FROM probabilities
WHERE match_id = 10;`

Новые данные с поля

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`

ТРАНЗАКЦИИ С Serializable

Остановка ведения матча

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `SELECT * FROM probabilities
WHERE match_id = 10;`
- 4 `UPDATE matches SET STATUS =
'stopped' WHERE match_id = 10;`

Новые данные с поля

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`

ТРАНЗАКЦИИ С Serializable

Остановка ведения матча

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `SELECT * FROM probabilities
WHERE match_id = 10;`
- 4 `UPDATE matches SET STATUS =
'stopped' WHERE match_id = 10;`
- 5 `INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');`
- 6 `COMMIT;`

Новые данные с поля

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`

ТРАНЗАКЦИИ C Serializable

Остановка ведения матча

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `SELECT * FROM probabilities
WHERE match_id = 10;`
- 4 `UPDATE matches SET STATUS =
'stopped' WHERE match_id = 10;`
- 5 `INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');`
- 6 `COMMIT;`

Новые данные с поля

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 7 `UPDATE probabilities SET
value = 2
WHERE match_id = 10;`

ТРАНЗАКЦИИ C Serializable

Остановка ведения матча

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `SELECT * FROM probabilities
WHERE match_id = 10;`
- 4 `UPDATE matches SET STATUS =
'stopped' WHERE match_id = 10;`
- 5 `INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');`
- 6 `COMMIT;`

Новые данные с поля

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 7 `UPDATE probabilities SET
value = 2
WHERE match_id = 10;`
- 8 `>>> SERIALIZATION ERROR`

ТРАНЗАКЦИИ C Serializable

Остановка ведения матча

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `SELECT * FROM probabilities
WHERE match_id = 10;`
- 4 `UPDATE matches SET STATUS =
'stopped' WHERE match_id = 10;`
- 5 `INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');`
- 6 `COMMIT;`

Новые данные с поля

- 8 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 8 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'stopped'`

ТРАНЗАКЦИИ C Serializable

Остановка ведения матча

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `SELECT * FROM probabilities
WHERE match_id = 10;`
- 4 `UPDATE matches SET STATUS =
'stopped' WHERE match_id = 10;`
- 5 `INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');`
- 6 `COMMIT;`

Новые данные с поля

- 8 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 8 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'stopped'`
- 9 `COMMIT;`

ТРАНЗАКЦИИ С Serializable

outbox		
sequence_number	match_id	report
1	10	Закрытие вероятностей

ХОРОШО



A close-up shot of Leonardo DiCaprio from the chest up. He is wearing a black tuxedo with a white shirt and a dark bow tie. He is holding a glass of champagne in his right hand, which is raised towards the camera. He has a slight, knowing smile on his face, looking directly at the viewer. The background is dark and out of focus, filled with numerous small, bright, bokeh lights in shades of blue, white, and gold, suggesting a festive or party atmosphere.

ЭТО

УСПЕХ

A close-up shot of Leonardo DiCaprio from the chest up. He is wearing a black tuxedo with a white shirt and a dark bow tie. He is holding a flute glass filled with a bubbly liquid, likely champagne, and is looking directly at the camera with a slight, knowing smile. His hair is neatly combed. The background is dark and out of focus, showing the lights and ornaments of a Christmas tree, creating a bokeh effect with warm, golden light spots.

ЭТО

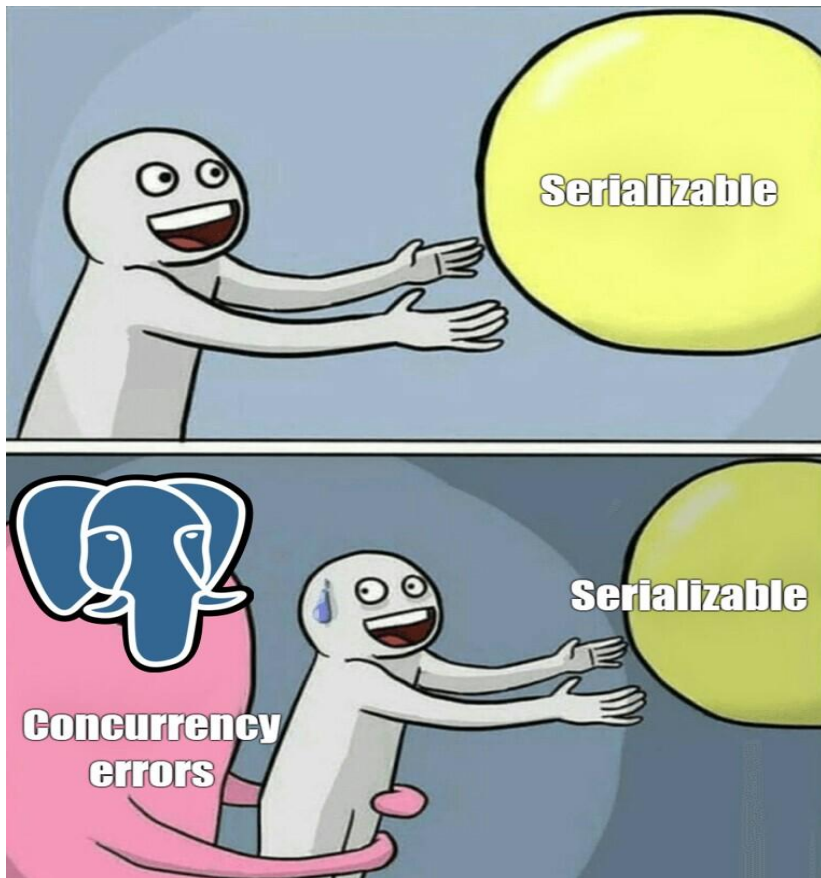
УСПЕХ

?????

ПРОБЛЕМЫ







Проблемы

Детектирование аномалий, где их концептуально нет

Необходимость повторять запросы в таких ситуациях

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1



ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

Новые данные с поля по матчу 10

0

```
BEGIN TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;
```

Новые данные с поля по матчу 20

0

```
BEGIN TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;
```

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

Новые данные с поля по матчу 10

```
0 BEGIN TRANSACTION ISOLATION  
  LEVEL SERIALIZABLE;  
  
1 SELECT STATUS FROM matches  
  WHERE match_id = 10; //  
  'active'
```

Новые данные с поля по матчу 20

```
0 BEGIN TRANSACTION ISOLATION  
  LEVEL SERIALIZABLE;  
  
2 SELECT STATUS FROM matches  
  WHERE match_id = 20; //  
  'active'
```

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

Новые данные с поля по матчу 10

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities;

Новые данные с поля по матчу 20

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 2 SELECT STATUS FROM matches
WHERE match_id = 20; //
'active'
- 4 SELECT * FROM probabilities;

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

Новые данные с поля по матчу 10

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities;
- 5 UPDATE probabilities SET
value = 2 WHERE match_id = 10;

Новые данные с поля по матчу 20

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 2 SELECT STATUS FROM matches
WHERE match_id = 20; //
'active'
- 4 SELECT * FROM probabilities;

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

Новые данные с поля по матчу 10

```
0 BEGIN TRANSACTION ISOLATION
  LEVEL SERIALIZABLE;

1 SELECT STATUS FROM matches
  WHERE match_id = 10; //
  'active'

3 SELECT * FROM probabilities;

5 UPDATE probabilities SET
  value = 2 WHERE match_id = 10;

6 INSERT INTO outbox(match_id,
  report) VALUES
  (10, '{"key": "value"}');

7 COMMIT;
```

Новые данные с поля по матчу 20

```
0 BEGIN TRANSACTION ISOLATION
  LEVEL SERIALIZABLE;

2 SELECT STATUS FROM matches
  WHERE match_id = 20; //
  'active'

4 SELECT * FROM probabilities;
```

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

Новые данные с поля по матчу 10

```
0 BEGIN TRANSACTION ISOLATION
  LEVEL SERIALIZABLE;

1 SELECT STATUS FROM matches
  WHERE match_id = 10; //
  'active'

3 SELECT * FROM probabilities;

5 UPDATE probabilities SET
  value = 2 WHERE match_id = 10;

6 INSERT INTO outbox(match_id,
  report) VALUES
  (10, '{"key": "value"}');

7 COMMIT;
```

Новые данные с поля по матчу 20

```
0 BEGIN TRANSACTION ISOLATION
  LEVEL SERIALIZABLE;

2 SELECT STATUS FROM matches
  WHERE match_id = 20; //
  'active'

4 SELECT * FROM probabilities;

8 UPDATE probabilities SET
  value = 2 WHERE match_id = 20;
```


ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

Новые данные с поля по матчу 10

```
0 BEGIN TRANSACTION ISOLATION
  LEVEL SERIALIZABLE;

1 SELECT STATUS FROM matches
  WHERE match_id = 10; //
  'active'

3 SELECT * FROM probabilities;

5 UPDATE probabilities SET
  value = 2 WHERE match_id = 10;

6 INSERT INTO outbox(match_id,
  report) VALUES
  (10, '{"key": "value"}');

7 COMMIT;
```

Новые данные с поля по матчу 20

```
0 BEGIN TRANSACTION ISOLATION
  LEVEL SERIALIZABLE;

2 SELECT STATUS FROM matches
  WHERE match_id = 20; //
  'active'

4 SELECT * FROM probabilities;

8 UPDATE probabilities SET
  value = 2 WHERE match_id = 20;

9 >>> SERIALIZATION ERROR
```

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

Новые данные с поля по матчу 10

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 SELECT * FROM probabilities;
- 5 UPDATE probabilities SET
value = 2 WHERE match_id = 10;
- 6 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');
- 7 COMMIT;

Новые данные с поля по матчу 20

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 2 SELECT STATUS FROM matches
WHERE match_id = 20; //
'active'
- 4 SELECT * FROM probabilities;
- 8 UPDATE probabilities SET
value = 2 WHERE match_id = 20;
- 9 >>> SERIALIZATION ERROR

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

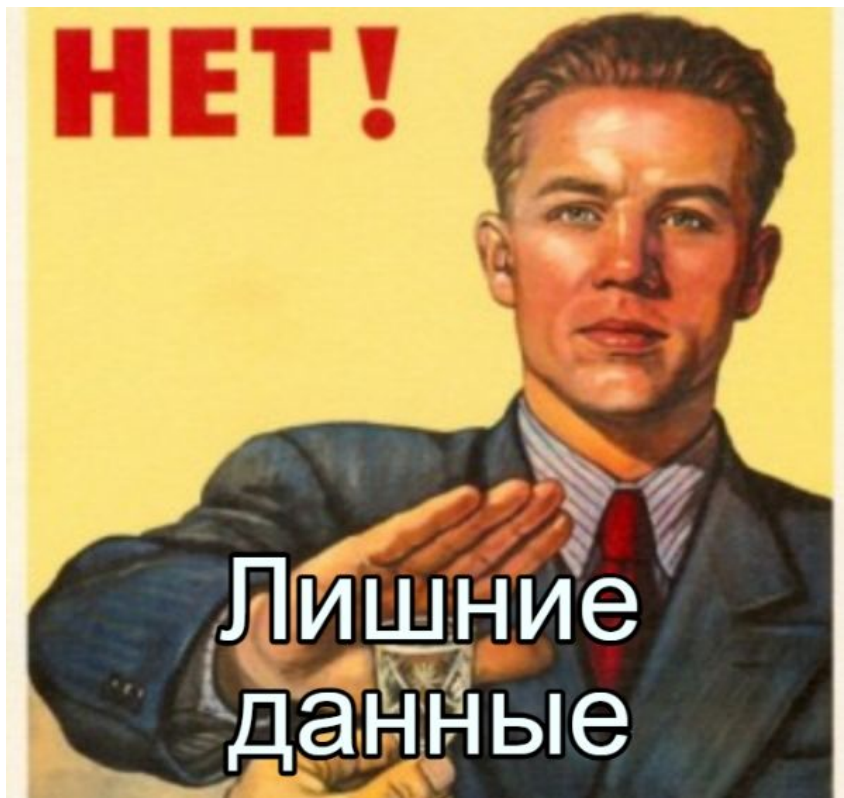
```
var probabilities = await dbContext.Probabilities
    .ToListAsync(token);

var filteredProbabilities = probabilities
    .Where(probability => probability.MatchId ==
matchToUpdate.MatchId)
    .ToList();
```

ЗАПРОС ЛИШНИХ ДАННЫХ

Проблема 1

```
var filteredProbabilities = await dbContext.Probabilities
    .Where(probability => probability.MatchId ==
matchToUpdate.MatchId)
    .ToListAsync(token);
```



РЕШЕНИЕ

Не запрашивать данные,
которые не нужны для
исполнения транзакции

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2



ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Новые данные с поля по матчу 10

0

```
BEGIN TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;
```

Новые данные с поля по матчу 20

0

```
BEGIN TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;
```

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Новые данные с поля по матчу 10

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'`

Новые данные с поля по матчу 20

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 20; //
'active'`

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Новые данные с поля по матчу 10

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `UPDATE probabilities SET
value = 2 WHERE match_id = 10;`

Новые данные с поля по матчу 20

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 20; //`
`'active'`

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Новые данные с поля по матчу 10

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT STATUS FROM matches
WHERE match_id = 10; //`
`'active'`
- 3 `UPDATE probabilities SET
value = 2 WHERE match_id = 10;`
- 5 `INSERT INTO outbox(match_id,
report) VALUES
(10, '{ "key": "value" });`
- 6 `COMMIT;`

Новые данные с поля по матчу 20

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT STATUS FROM matches
WHERE match_id = 20; //`
`'active'`

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Новые данные с поля по матчу 10

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 UPDATE probabilities SET
value = 2 WHERE match_id = 10;
- 5 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');
- 6 COMMIT;

Новые данные с поля по матчу 20

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 2 SELECT STATUS FROM matches
WHERE match_id = 20; //
'active'
- 7 UPDATE probabilities SET
value = 2 WHERE match_id = 20;

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Новые данные с поля по матчу 10

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 1 SELECT STATUS FROM matches
WHERE match_id = 10; //
'active'
- 3 UPDATE probabilities SET
value = 2 WHERE match_id = 10;
- 5 INSERT INTO outbox(match_id,
report) VALUES
(10, '{"key": "value"}');
- 6 COMMIT;

Новые данные с поля по матчу 20

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 2 SELECT STATUS FROM matches
WHERE match_id = 20; //
'active'
- 7 UPDATE probabilities SET
value = 2 WHERE match_id = 20;
- 8 >>> SERIALIZATION ERROR

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Новые данные с поля по матчу 10

- 0 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
- 1 SELECT STATUS FROM matches WHERE match_id = 10; // 'active'
- 3 UPDATE probabilities SET value = 2 WHERE match_id = 10;
- 5 INSERT INTO outbox(match_id, report) VALUES (10, '{"key": "value"}');
- 6 COMMIT;

Новые данные с поля по матчу 20

- 0 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
- 2 SELECT STATUS FROM matches WHERE match_id = 20; // 'active'
- 7 UPDATE probabilities SET value = 2 WHERE match_id = 20;
- 8 >>> SERIALIZATION ERROR



ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Страница таблицы Probabilities							
match_id = 10	match_id = 10	match_id = 10	match_id = 10	match_id = 10	match_id = 20	match_id = 20	

8 KB

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Predicate lock	Predicate lock	Predicate lock	Predicate lock	Predicate lock				
Страница таблицы Probabilities								
match_id = 10	match_id = 10	match_id = 10	match_id = 10	match_id = 10	match_id = 20	match_id = 20		

8 KB

ИЗМЕНЕНИЕ БОЛЬШОГО КОЛИЧЕСТВА СТРОК В РАМКАХ ОДНОЙ ТРАНЗАКЦИИ

Проблема 2

Predicate lock

Страница таблицы Probabilities								
match_id = 10	match_id = 10	match_id = 10	match_id = 10	match_id = 10	match_id = 20	match_id = 20		

8 KB

РЕШЕНИЕ

Проблема 2

Параметр в postgresql.conf -
max_pred_locks_per_page

РЕШЕНИЕ

Проблема 2

Параметр в postgresql.conf -
max_pred_locks_per_page

max_pred_locks_per_page = 128
(2 по умолчанию)

РЕШЕНИЕ

Проблема 2

Параметр в postgresql.conf -
max_pred_locks_per_page

max_pred_locks_per_page = 128
(2 по умолчанию)

Цена: больше расход памяти

КОНКУРЕНЦИЯ ПРИ ДОБАВЛЕНИИ НОВЫХ ЗАПИСЕЙ

Проблема 3

КОНКУРЕНЦИЯ ПРИ ДОБАВЛЕНИИ НОВЫХ ЗАПИСЕЙ

Проблема 3

Добавление матча с id = 100

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 1 `SELECT * FROM matches WHERE
match_id = 100; // 'null'`

Добавление матча с id = 200

- 0 `BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;`
- 2 `SELECT * FROM matches WHERE
match_id = 200; // 'null'`

КОНКУРЕНЦИЯ ПРИ ДОБАВЛЕНИИ НОВЫХ ЗАПИСЕЙ

Проблема 3

Добавление матча с id = 100

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 1 SELECT * FROM matches WHERE
match_id = 100; // 'null'
- 3 INSERT INTO matches(match_id,
status) VALUES (100,
'active');
- 4 COMMIT;

Добавление матча с id = 200

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 2 SELECT * FROM matches WHERE
match_id = 200; // 'null'

КОНКУРЕНЦИЯ ПРИ ДОБАВЛЕНИИ НОВЫХ ЗАПИСЕЙ

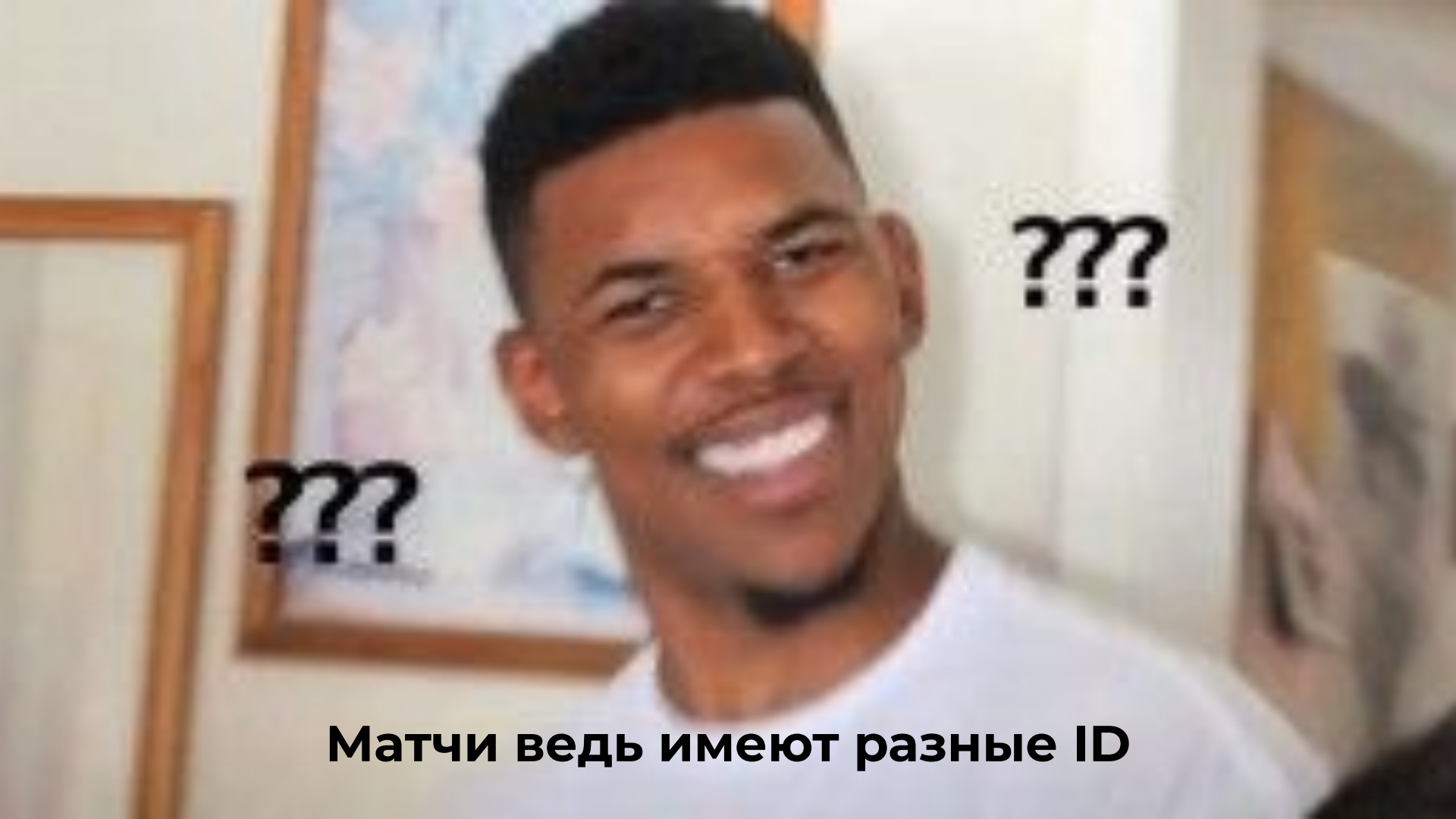
Проблема 3

Добавление матча с id = 100

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 1 SELECT * FROM matches WHERE
match_id = 100; // 'null'
- 3 INSERT INTO matches(match_id,
status) VALUES (100,
'active');
- 4 COMMIT;

Добавление матча с id = 200

- 0 BEGIN TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
- 2 SELECT * FROM matches WHERE
match_id = 200; // 'null'
- 5 INSERT INTO matches(match_id,
status) VALUES (200,
'active');
- 6 >>> SERIALIZATION ERROR



Матчи ведь имеют разные ID

КОНКУРЕНЦИЯ ПРИ ДОБАВЛЕНИИ НОВЫХ ЗАПИСЕЙ

Проблема 3

Страница индекса таблицы Matches

match_id = 1	match_id = 10	match_id = 20	match_id = 100	match_id = 200	match_id = 400	match_id = 500		
-----------------	------------------	------------------	-------------------	-------------------	-------------------	-------------------	--	--

8 KB

КОНКУРЕНЦИЯ ПРИ ДОБАВЛЕНИИ НОВЫХ ЗАПИСЕЙ

Проблема 3

Predicate lock

Страница индекса таблицы Matches

match_id = 1	match_id = 10	match_id = 20	match_id = 100	match_id = 200	match_id = 400	match_id = 500		
-----------------	------------------	------------------	-------------------	-------------------	-------------------	-------------------	--	--

8 KB

РЕШЕНИЕ

Проблема 3

Нет решения через конфигурирование PostgreSQL

Вы должны не допускать таких параллельных транзакций по бизнес-логике, либо быть готовыми к повторению транзакций

ИСПОЛЬЗОВАНИЕ НЕ ПОСЛЕДНЕЙ ВЕРСИИ PostgreSQL

Проблема 4

ИСПОЛЬЗОВАНИЕ НЕ ПОСЛЕДНЕЙ ВЕРСИИ PostgreSQL

Проблема 4

Две параллельных очень крупных транзакций, которые используют много данных из БД



РЕШЕНИЕ

По возможности
обновлять PostgreSQL, так
как Serializable улучшается
и развивается

ДИАГНОСТИКА



ЕЩЕ ОДНА НЕОЖИДАННАЯ



CONCURRENCY-ОШИБКА

ПРЕСТУПЛЕНИЕ



НА УЛИЦЕ SERIALIZABLE

ДИАГНОСТИКА. НЕОБХОДИМЫЕ ДАННЫЕ

ДИАГНОСТИКА. НЕОБХОДИМЫЕ ДАННЫЕ. ЖЕРТВА

- Жертва - транзакция, которая получила ошибку конкуренции

ДИАГНОСТИКА. НЕОБХОДИМЫЕ ДАННЫЕ. ЖЕРТВА

- Жертва - транзакция, которая получила ошибку конкуренции

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (53ms) [Parameters=[@__P_MatchId_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT m.match_id, m.status
      FROM matches AS m
      WHERE m.match_id = @__P_MatchId_0
      LIMIT 2
```

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (2ms) [Parameters=[@__P_MatchId_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT m.match_id, m.status
      FROM matches AS m
      WHERE m.match_id = @__P_MatchId_0
      LIMIT 2
```

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (2ms) [Parameters=[@__P_MatchId_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT p.match_id, p.probability_id, p.value
      FROM probabilities AS p
      WHERE p.match_id = @__P_MatchId_0
```

ДИАГНОСТИКА. НЕОБХОДИМЫЕ ДАННЫЕ. ПРЕСТУПНИК

- Преступник - параллельная транзакция, которая вызвала конкуренцию

ДИАГНОСТИКА. НЕОБХОДИМЫЕ ДАННЫЕ. ПРЕСТУПНИК

- Преступник - параллельная транзакция, которая вызвала конкуренцию
 - Попробовать найти ее также в логах приложения

ДИАГНОСТИКА. НЕОБХОДИМЫЕ ДАННЫЕ. ПРЕСТУПНИК

- Преступник - параллельная транзакция, которая вызвала конкуренцию
 - Попробовать найти ее также в логах приложения
 - Найти ее в логах Postgres'a

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Расположение логов Postgres'a:
 - В облаке, если у вас настроен их сброс в агрегатор логов (Google logs/ELK/etc)

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Расположение логов Postgres'a:
 - В облаке, если у вас настроен их сброс в агрегатор логов (Google logs/ELK/etc)
 - {PostgreSQL_path}/data/log

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Что ищем в логах Postgres'a:
 - Первую транзакцию, которая завершилась с ошибкой
 - Тайминг начала первой транзакции
 - Все транзакции, которые коммитились в период выполнения первой транзакции
 - Одна из этих транзакций будет той, которая привела к конкуренции

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Поиск второй транзакции из набора параллельных

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Поиск второй транзакции из набора параллельных
 - Создаем локальную базу, наполняем ее тестовыми данными

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Поиск второй транзакции из набора параллельных
 - Создаем локальную базу, наполняем ее тестовыми данными
 - Запускаем первую транзакцию, но не комитим

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Поиск второй транзакции из набора параллельных
 - Создаем локальную базу, наполняем ее тестовыми данными
 - Запускаем первую транзакцию, но не комитим
 - Запускаем вторую транзакцию и комитим

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Поиск второй транзакции из набора параллельных
 - Создаем локальную базу, наполняем ее тестовыми данными
 - Запускаем первую транзакцию, но не комитим
 - Запускаем вторую транзакцию и комитим
 - Комитим первую транзакцию

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Поиск второй транзакции из набора параллельных
 - Создаем локальную базу, наполняем ее тестовыми данными
 - Запускаем первую транзакцию, но не комитим
 - Запускаем вторую транзакцию и комитим
 - Комитим первую транзакцию
 - Если получаем ошибку конкурентности - мы нашли нужную пару

ДИАГНОСТИКА. ПОИСК ПРЕСТУПНИКА - ВТОРОЙ ТРАНЗАКЦИИ

- Поиск второй транзакции из набора параллельных
 - Создаем локальную базу, наполняем ее тестовыми данными
 - Запускаем первую транзакцию, но не комитим
 - Запускаем вторую транзакцию и комитим
 - Комитим первую транзакцию
 - Если получаем ошибку конкурентности - мы нашли нужную пару
 - Если не получаем ошибку - идем к следующей транзакции

ДИАГНОСТИКА. НЕОБХОДИМЫЕ ДАННЫЕ. МОТИВ ПРЕСТУПЛЕНИЯ

- Мотив преступления - информация о predicate_lock'ах в момент выполнения первой и второй транзакции

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Системная таблица `pg_locks`, которая содержит все локи (в т.ч. `predicate_lock`'и в момент выполнения транзакций)

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Системная таблица pg_locks, которая содержит все локи (в т.ч. predicate_lock'и в момент выполнения транзакций)

```
SELECT locktype, relation, page, tuple, pid  
FROM pg_locks  
WHERE mode = 'SIReadLock'
```

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Системная таблица pg_locks, которая содержит все локи (в т.ч. predicate_lock'и в момент выполнения транзакций)

```
SELECT locktype, relation, page, tuple, pid  
FROM pg_locks  
WHERE mode = 'SIReadLock'
```

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Системная таблица pg_locks, которая содержит все локи (в т.ч. predicate_lock'и в момент выполнения транзакций)

```
SELECT locktype, relation, page, tuple, pid  
FROM pg_locks  
WHERE mode = 'SIReadLock'
```

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Системная таблица pg_locks, которая содержит все локи (в т.ч. predicate_lock'и в момент выполнения транзакций)

```
SELECT locktype, relation, page, tuple, pid  
FROM pg_locks  
WHERE mode = 'SIReadLock'
```


ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Системная таблица pg_locks, которая содержит все локи (в т.ч. predicate_lock'и в момент выполнения транзакций)

```
SELECT locktype, relation, page, tuple, pid  
FROM pg_locks  
WHERE mode = 'SIReadLock'
```

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Запускаем первую транзакцию, но не комитим

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Запускаем первую транзакцию, но не комитим
- Запускаем вторую транзакцию, но не комитим

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

- Запускаем первую транзакцию, но не комитим
- Запускаем вторую транзакцию, но не комитим
- Выполняем запрос в таблицу `pg_locks` и смотрим на `predicate_lock`'и

ДИАГНОСТИКА. ОПРЕДЕЛЕНИЕ МОТИВА

	locktype	relation	page	tuple	pid
1	page	358867	1	<null>	35360
2	page	358867	1	<null>	30952
3	tuple	358869	0	55	30952
4	tuple	358869	0	53	30952
5	tuple	358869	0	54	30952
6	tuple	358862	0	33	30952
7	tuple	358862	0	33	35360
8	tuple	358869	0	56	30952
9	page	358874	1	<null>	30952
10	page	358874	1	<null>	35360

ДИАГНОСТИКА. ПОИСК РЕШЕНИЯ

- Пытаемся анализировать где произошло пересечение и почему
- Читаем статьи по Postgres'у и Serializable, чтобы понять причину создания лишних predicate_lock'ов и можно ли исправить

ВЫВОДЫ



ВЫВОДЫ. ПЛЮСЫ Serializable

- + Serializable справляется с задачей синхронизации параллельных операций
-

ВЫВОДЫ. ПЛЮСЫ Serializable

- + Serializable справляется с задачей синхронизации параллельных операций
-
- + Реализация в коде действительно оказалась не сложной

ВЫВОДЫ. НЕДОСТАТКИ Serializable

- Производительность ниже, чем у альтернативных решений
-

ВЫВОДЫ. НЕДОСТАТКИ Serializable

- Производительность ниже, чем у альтернативных решений
-

- Детектирование аномалий часто случается, где их нет
-

ВЫВОДЫ. НЕДОСТАТКИ Serializable

- Производительность ниже, чем у альтернативных решений
-
- Детектирование аномалий часто случается, где их нет
-
- Поиск причины и решения в случае ложного детектирования аномалии - больно и дорого
-

**ВЫВОДЫ.
КОГДА Serializable
МОЖЕТ БЫТЬ ПОЛЕЗЕН**

ВЫВОДЫ. КОГДА Serializable МОЖЕТ БЫТЬ ПОЛЕЗЕН

Старт-ап или MVP продукта, где не
ожидается высокая нагрузка вначале

ВЫВОДЫ. КОГДА Serializable МОЖЕТ БЫТЬ ПОЛЕЗЕН

Старт-ап или MVP продукта, где не
ожидается высокая нагрузка вначале

Система, в которой не предполагаются
высокие нагрузки по своей природе

ВЫВОДЫ. КОГДА Serializable МОЖЕТ БЫТЬ ПОЛЕЗЕН

Старт-ап или MVP продукта, где не ожидается высокая нагрузка вначале

Система, в которой не предполагаются высокие нагрузки по своей природе

Система со сложной структурой хранения данных

РЕШЕНИЕ.

АЛЬТЕРНАТИВНЫЕ СПОСОБЫ

- Использовать ручные блокировки (например, `SELECT FOR UPDATE`)
- Использовать рекомендательные блокировки (Advisory lock)

Спасибо за внимание!

Полезные ссылки

- ↪ Паттерн Outbox
- ↪ Уровни изолированности транзакций в PostgreSQL
- ↪ Описание Serializable в PostgreSQL
- ↪ Predicate-locks в PostgreSQL



Презентация

Черняев Антон

Tg: @anchernyaev