INSTITUTO FEDERAL
Paraíba
Campus Cajazeiras

PROGRAMAÇÃO P/ WEB 2

3

7. DESENVOLVENDO LÓGICA DE NEGÓCIO COM EVENT-SOURCING

PROF. DIEGO PESSOA

DIEGO.PESSOA@IFPB.EDU.BR

@DIEGOEP

CST em Análise e Desenvolvimento de Sistemas

ROTEIRO

- Usando o padrão *Event Sourcing* para desenvolver lógica de negócio
- Armazenando eventos
- Integrando sagas e lógica de negócios baseada em fornecimento de eventos
- Implementando orquestradores saga usando fornecimento de eventos

VISÃO GERAL

- Estruturar lógica de negócio como uma coleção de DDD aggregates que publicam domínios de evento é bastante útil numa arquitetura de microsserviço.

- Sagas podem ser usadas para implementar a consistência de dados em vários serviços

- Mas cada evento precisa ser modelado e disparado individualmente?

EVENT SOURCING

- *Event sourcing* é um forma de organizar lógica de negócio e persistir objetos de domínio.
- Elimina erros ao garantir que um evento irá ser publicado sempre que uma aggregate é criada ou atualizada

DESENVOLVENDO LÓGICA DE NEGÓCIO COM EVENT SOURCING

- Event sourcing é uma maneira diferente de estruturar a lógica de negócio e persistir aggregates
- A ideia é persistir aggregates como uma sequência de eventos, em que cada evento representa uma mudança de estado.
- Uma aplicação recria o estado corrente de uma aggregate através da repetição de eventos

PERSISTE UMA AGGREGATE COMO UMA SEQUÊNCIA DE EVENTOS, QUE REPRESENTAM MUDANÇAS DE ESTADO

Padrão: Event Sourcing

DESENVOLVENDO LÓGICA DE NEGÓCIO COM EVENT SOURCING

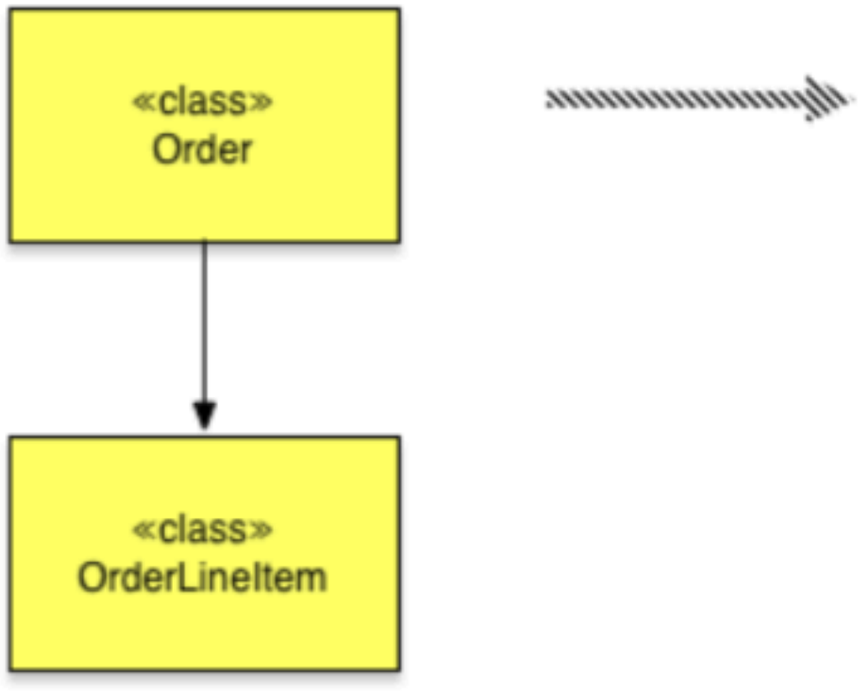
- Benefícios de usar Event Sourcing:
  - Preserva o histórico de aggregates, o que é útil para propósitos de auditoria e regulação
  - Garante a publicação de eventos de domínio, que são úteis na arquitetura de microsserviço.

DESENVOLVENDO LÓGICA DE NEGÓCIO COM EVENT SOURCING

- Desvantagens de usar Event Sourcing:
  - Curva de aprendizado, já que é uma maneira diferente de estruturar a lógica de negócio.
  - Consultar eventos armazenados geralmente requer o uso do Command Query Responsibilty Pattern (CQRS)

O PROBLEMA COM A PERSISTÊNCIA TRADICIONAL

- Tradicionalmente, tabelas são mapeadas para classes e os respectivos campos são colunas. As instâncias são as linhas das tabelas.



ORDER table

ID	CUSTOMER_ID	ORDER_TOTAL	...
1234	customer-abc	1234.56	...

ORDER_LINE_ITEM table

ID	ORDER_ID	QUANTITY	...
567	1234	2	...

O PROBLEMA COM A PERSISTÊNCIA TRADICIONAL

- A maior parte das aplicações corporativas armazenam dados desta maneira, porém há várias limitações:
  - Object-Relational impedance mismatch
  - Falta de um histórico de aggregates
  - Implementação de log e auditoria é custosa
  - Publicação de eventos é vinculada à lógica de negócios

OBJECT-RELATIONAL IMPEDANCE MISMATCH

- Descasamento conceitual entre o modelo tabular (relacional) e o modelo de objetos (mais rico).
- Ler: http://blogs.tedneward.com/post/the-vietnam-of-computer-science/
- O problema vai além das limitações de um framework ORM particular

FALTA DE HISTÓRICO DE AGGREGATES

- A persistência guarda somente um estado de uma aggregate
- Uma vez que a aggregate foi atualizada, o estado anterior é perdido
- Pode ser interessante guardar esse histórico para fins de auditoria
- Os desenvolvedores acabam tendo de desenvolver isso manualmente

IMPLEMENTAÇÃO DE LOG E AUDITORIA É CUSTOSA

- Muitas aplicações mantem log de auditoria para rastrear o que os usuário mudaram em uma aggregate
- Auditoria pode ser usado para segurança ou propósitos regulatórios
- O desafio é fazer a auditoria refletir corretamente o histórico de todas as ações realizadas no nível de negócio

PUBLICAÇÃO DE EVENTOS É VINCULADA À LÓGICA DE NEGÓCIOS

- outra limitação da persistência tradicional é que não suportam a publicação de eventos
- Eventos de domínio são úteis para sincronizadas dados e enviar notificações
- Alguns frameworks ORM proveem callbacks quando objetos de dados mudam
- Porém, não há suporte para publicação automática de mensagens como parte de uma transação

EVENT SOURCING – VISÃO GERAL

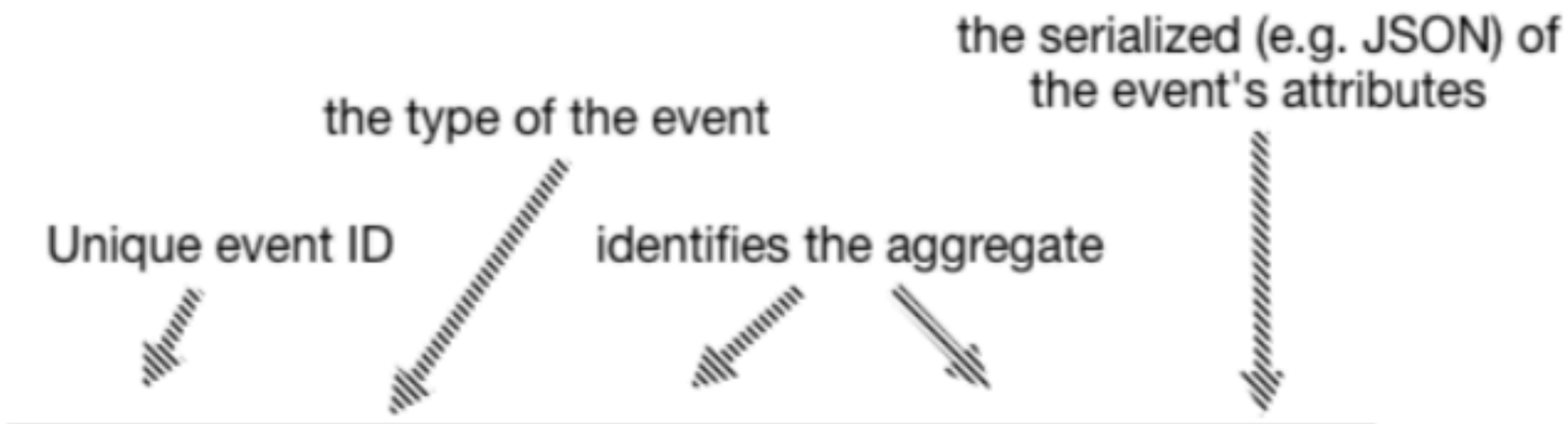
- Event sourcing é uma técnica centralizada em eventos para implementar lógica de negócio
- Uma aggregate é persistida como uma série de eventos
- Cada evento representa uma mudança de estado
- A lógica de negócio é estruturada em torno de produzir e consumir eventos

PERSISTINDO AGGREATES USANDO EVENTOS

- Considere como exemplo a aggregate "Order", ao invés de persistir cada pedido como uma linha da tabela "Order"

- Event sourcing persiste cada Order como uma ou mais linhas numa tabela EVENTOS. Cada linha é um evento de domínio, tais como Order Created, Order Approved, Order Shipped, etc.

EVENT SOURCING PERSISTE UMA AGGREGATE COMO UMA SEQUÊNCIA DE EVENTOS



event_id	event_type	entity_type	entity_id	event_data
102	Order Created	Order	101	{...}
103	Order Approved	Order	101	{...}
104	Order Shipped	Order	101	{...}
105	Order Delivered	Order	101	{...}
...	...	...	...	...

TABELA EVENTOS

- Quando uma aplicação cria ou atualiza uma aggregate, insere eventos emitidos pela aggregate na tabela eventos
- A aplicação carrega uma aggregate recuperando os seus eventos.

- Carregar uma aggregate consiste de:
  - 1. Carregar os eventos para a aggregate
  - 2. Criar uma instância genérica usando o seu construtor default
  - 3. Iterar sobre seus eventos chamando apply()

TABELA EVENTOS

- Exemplo de como framework eventuate carrega eventos:

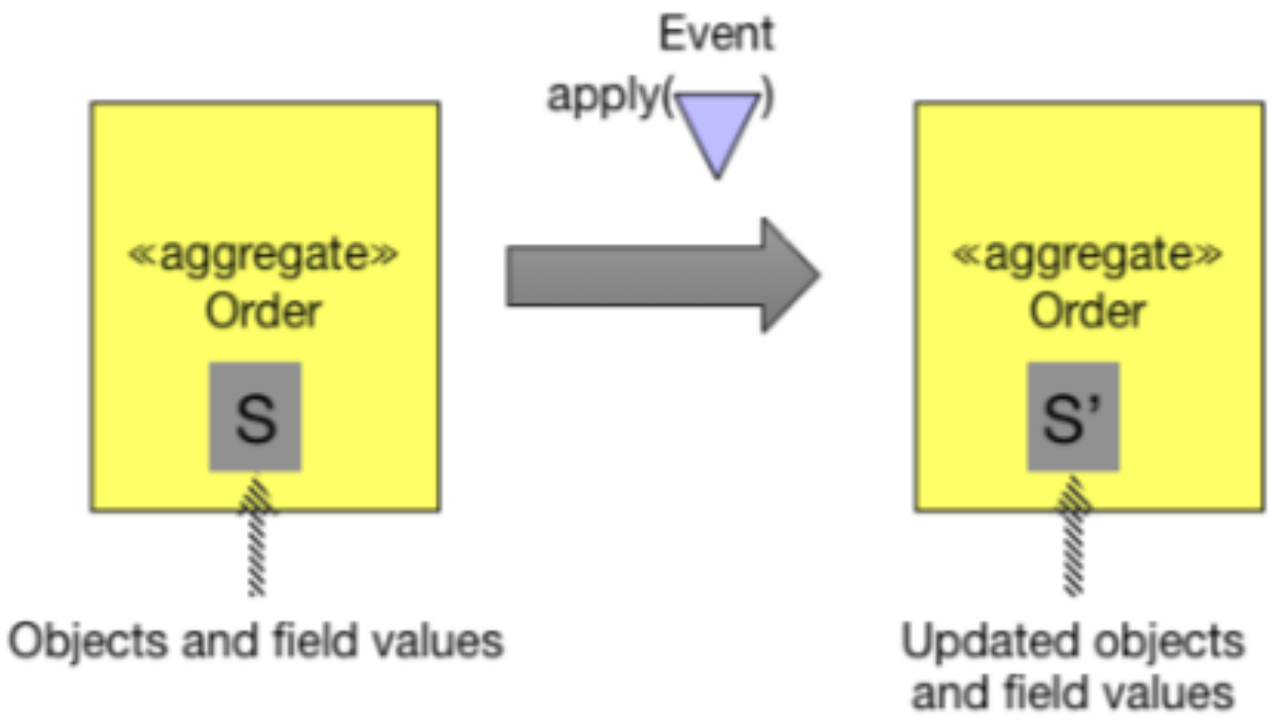
```
Class aggregateClass = ...;
Aggregate aggregate = aggregateClass.newInstance();
for (Event event : events) {
aggregate = aggregate.applyEvent(event);
}
// use aggregate...
```

TABELA EVENTOS

- Pode parecer estranho reconstruir em memória o estado de uma aggregate por carregar e reexecutar eventos
- Mas na prática alguns frameworks ORM como o Hibernate executam vários SELECT para recuperar o estado do objeto
- A diferença é que event sourcing faz a reconstrução com base em eventos

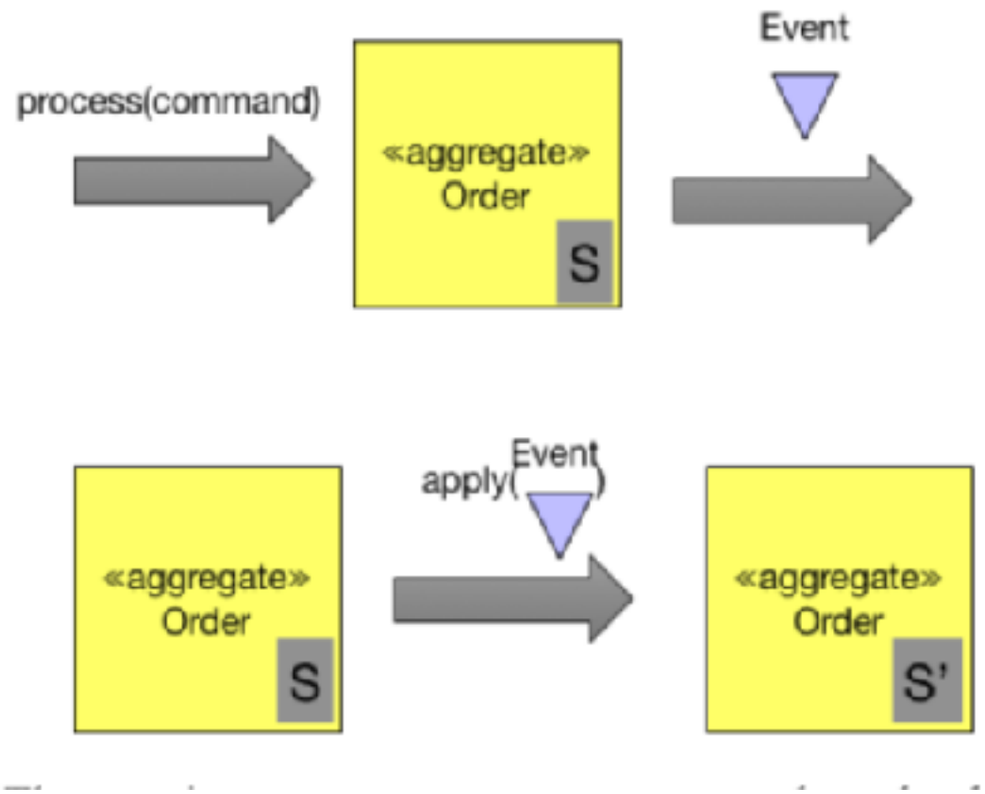
CADA EVENTO REPRESENTA MUDANÇA DE ESTADOS

- Eventos não são opcionais em event sourcing
- Cada mudança (incluindo sua criação) é representada por um evento de domíno



TODOS OS MÉTODOS DAS AGGREGATES SÃO SOBRE EVENTOS

- Aggregates são atualizadas através da aplicação de eventos (método apply()), ao invés da atribuição manual de valores (como ocorre tradicionalmente).



MÉTODOS PROCESS E APPLY

- O framework para event sourcing (Eventuate Client) define os seguintes métodos:
  - process() - para receber um comando (requisição), que contém os argumentos para a atualização e retornar uma lista de eventos
  - apply() - recebe um evento como parâmetro e retorna void.
- Uma aggreagate pode definir múltiplos eventos.

MÉTODOS PROCESS E APPLY – EXEMPLO

Returns events without updating the Order

```
public class Order {

 public List<DomainEvent> revise(OrderRevision orderRevision) {
  switch (state) {

   case AUTHORIZED:
    LineItemQuantityChange change =
          orderLineItems.lineItemQuantityChange(orderRevision);
    if (change.newOrderTotal.isGreaterThanOrEqual(orderMinimum)) {
     throw new OrderMinimumNotMetException();
    }
    this.state = REVISION_PENDING;
    return ...;

   default:
    throw new UnsupportedStateTransitionException(state);
  }
 }
```

```
public class Order {

 public List<Event> process(ReviseOrder command) {
   OrderRevision orderRevision = command.getOrderRevision();
   switch (state) {
     case AUTHORIZED:
       LineItemQuantityChange change =
                orderLineItems.lineItemQuantityChange(orderRevision);
       if (change.newOrderTotal.isGreaterThanOrEqual(orderMinimum)) {
         throw new OrderMinimumNotMetException();
       }
       return singletonList(
           new OrderRevisionProposed(
                          orderRevision, change.currentOrderTotal,
                          change.newOrderTotal));

     default:
       throw new UnsupportedStateTransitionException(state);
   }
 }
```

Applies events to update the Order

```
public class Order {

  public void apply(OrderRevisionProposed event) {
   this.state = REVISION_PENDING;
  }
```

A AGGREGATE ORDER USANDO EVENT-SOURCING

```
public class Order {

  private OrderState state;
  private Long consumerId;
  private Long restaurantId;
  private OrderLineItems orderLineItems;
  private DeliveryInformation deliveryInformation;
  private PaymentInformation paymentInformation;
  private Money orderMinimum;

  public Order() {
  }

  public List<Event> process(CreateOrderCommand command) {
    ... validate command ...
    return events(new OrderCreatedEvent(command.getOrderDetails()));
  }

  public void apply(OrderCreatedEvent event) {
    OrderDetails orderDetails = event.getOrderDetails();
    this.orderLineItems = new OrderLineItems(orderDetails.getLineItems());
    this.orderMinimum = orderDetails.getOrderMinimum();
    this.state = APPROVAL_PENDING;
  }
}
```

TRATANDO ATUALIZAÇÕES CONCORRENTES COM LOCKING OTIMISTA

- A cada atualização em aggregate, a sua versão é incrementada. Ex.:

```
UPDATE AGGREGATE_ROOT_TABLE
SET VERSION = VERSION + 1 ... WHERE
VERSION = <original version>
```

- Atualizações concorrentes em versões diferentes irão falhar.
- Pode ser usado para atualizações no armazenamento de eventos

EVENT-SOURCING E PUBLICAÇÃO DE EVENTOS

- Event sourcing persiste aggregates como eventos e reconstrui o estado corrente de uma aggregate para esses eventos
- A publicação de eventos realiza a guarda permanentemente os eventos numa tabela EVENTOS ao invés de salvar na OUTBOX.
- Opções para geração do registro na tabela evento: polling e log tailing.

USANDO POOLING PARA PUBLICAR NOVOS EVENTOS

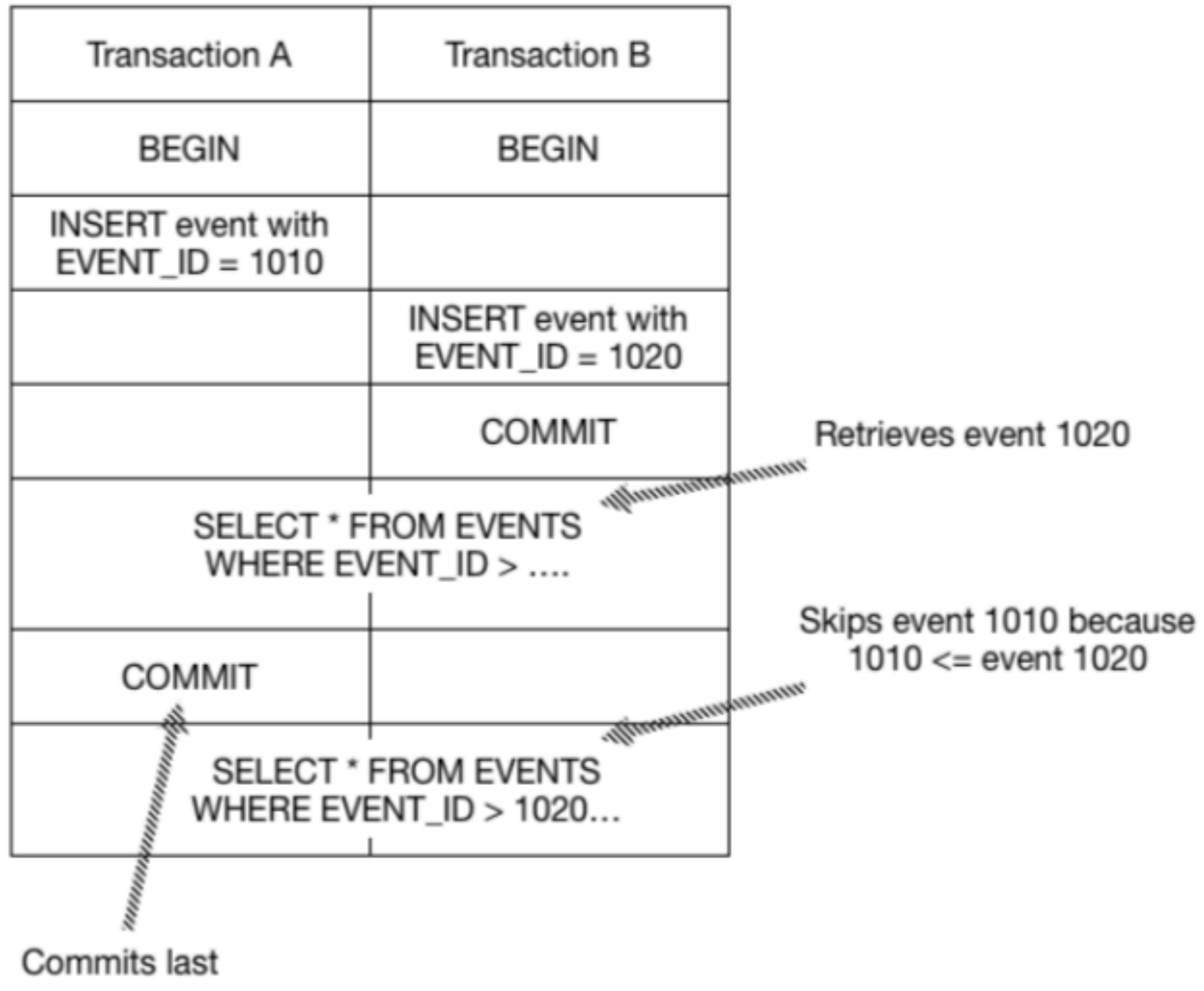
- O publicador de eventos irá fazer polling na tabela eventos para identificar novos eventos e então submeter ao message broker.

- O desafio é determinar quando os eventos são novos.

- Ex.: imagine a tabela eventos possui um ID incremental. A solução inicial seria guardar o último ID e pegar os eventos com ID maior.

USANDO POOLING PARA PUBLICAR NOVOS EVENTOS

- O problema é que as transações podem ser comitadas em ordem diferente da que o evento foi gerado
- Isso faria com que alguns eventos fossem desconsiderados



USANDO POOLING PARA PUBLICAR NOVOS EVENTOS

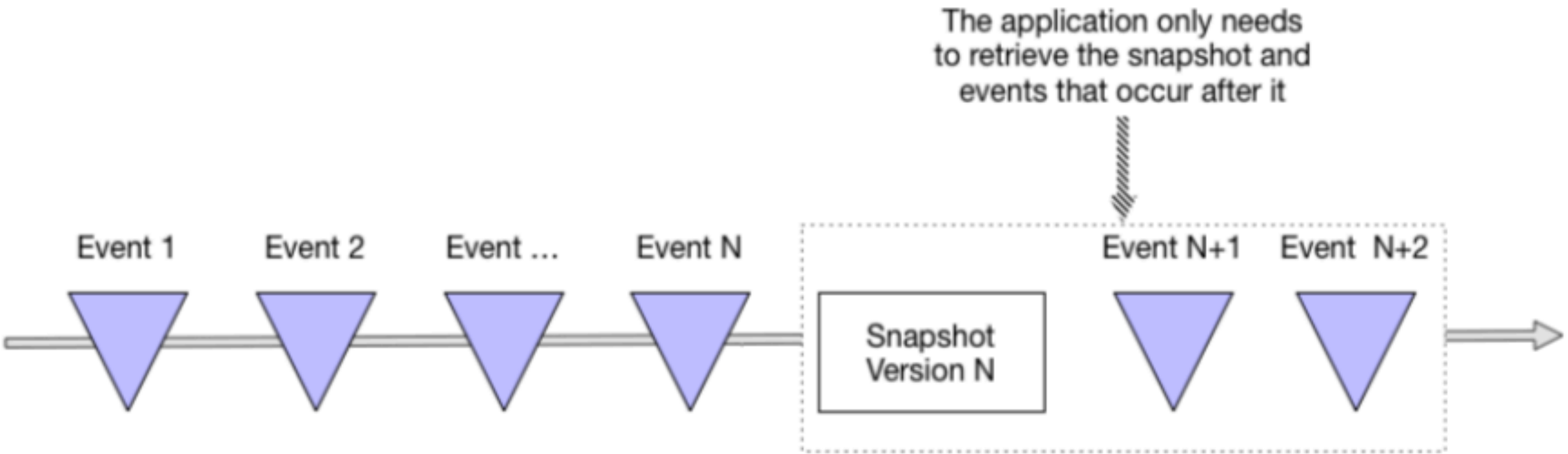
- Uma solução seria adicionar uma coluna extra à tabela eventos indicando que o evento foi publicado.
- Desta maneira, o publicador usaria o seguinte processo:
- 1. Busca evenos não publicados executando "SELECT*FROM EVENTS where PUBLISHED = 0 ORDER BY event_id ASC"
- 2. Publica as mensagens de evento no message broker
- 3. Marca o evento indicando que já foi publicado “UPDATE EVENTS SET PUBLISHED = 1 WHERE EVENT_ID in ?”

USANDO LOG TAILING PARA PUBLICAR EVENTOS

- De maneira mais sofisticada, é possível usar o log de transações à tabela eventos para poder disparar a publicação para o MessageBroker
- Possui melhor desempenho e é mais escalável
- O Framework Eventuate usa essa estratégia

USANDO SNAPSHOTS PARA MELHORAR DESEMPENHO

- Aggregates que são guardadas por muito tempo e podem possuir um grande número de eventos, podem demandar um maior esforço para reconstruir todo o histórico
- Uma solução seria persistir uma "snapshot" de um estado de uma aggregate. A aplicação então, restaura o estado carregando a snapshot mais recente.



USANDO SNAPSHOTS PARA MELHORAR DESEMPENHO – EXEMPLO

EVENTS

event_id	event_type	entity_type	entity_id	event_data
...	...	...	...	...
103	...	Customer	101	{...}
104	Credit Reserved	Customer	101	{...}
105	Address Changed	Customer	101	{...}
106	Credit Reserved	Customer	101	{...}

SNAPSHOTS

event_id	entity_type	entity_id	snapshot_data
...	...	...	...
103	Customer	101	{ name: "...", ... }
...	...	...	...
...	...	...	...

EVITANDO MENSAGENS DUPLICADAS

- Serviços geralmente consomem mensagens de outras aplicações ou serviços (ex.: evento de aggregate ou comando disparado por saga)
- Uma alternativa é gravar os IDs das mensagens já processadas numa tabela (ex.: PROCESSED_MESSAGES)
- Desta forma, caso chegue uma mensagem duplicada (com o mesmo ID de alguma existente), ela é descartada

BENEFÍCIOS DE EVENT-SOURCING

- Publica eventos de domínio de maneira confiável
- Preserva o histórico de aggregates
- Geralmente elimina o problema de impedance mismatch entre relacional e o OO
- Prover aos desenvolvedores uma máquina do tempo

DESVANTAGENS DE EVENT-SOURCING

- Modelo de programação diferente, o que demanda uma maior curva de aprendizagem
- Complexidade de aplicações baseadas em mensagens
- Evoluir eventos pode ser complicado
- Deletar dados é complicado
- Fazer buscas por eventos é desafiador

EVENT STORES

- Ferramentas que tratam o armazenamento de eventos:
  - Event Store (.net) - https://eventstore.org/
  - Lagom - https://www.lightbend.com/lagom-framework
  - Axon - http://www.axonframework.org/
  - Eventuate - http://eventuate.io/

EXEMPLO

- Usando sagas e eventos em conjunto