Replicação assíncrona em bancos de dados evolutivos

Helves Humberto Domingues
Fabio Kon
João Eduardo Ferreira

Departamento de Ciência da Computação
IME-USP

SBBD 2009 Fortaleza, Ceará

Sumário

- Introdução
- Limitações
- Contexto
- Solução proposta
- Trabalhos relacionados
- Conclusão

Introdução

- Arquitetura complexa dos sistemas de informação
- Evolução de banco de dados
- Pesquisa aplicada no projeto Borboleta
- Livro do Scott Ambler: *"Refactoring Databases: Evolutionary Database Design"*

Limitações

- Não é possível esperar que todas as aplicações sejam atualizadas para alterar o BD
- Metódos ágeis têm respondido bem aos problemas de desenvolvimento de software, mas em BD...
- ETL tradicionais são processos em lote e são "pesados" para evolução de BD
- Scott Ambler propõe replicação síncrona

Métodos tradicionais

- Processo longo e preciso:
  - análise dos requisitos
  - modelagem conceitual
  - modelagem lógica
  - modelagem física

Métodos tradicionais

- Não é capaz de atender à velocidade das mudanças de requisitos
- Custo alto para qualquer mudança

Métodos ágeis

- Ciclos rápidos de entrega e escopo reduzido:
  - ciclos de poucos meses
  - em cada ciclo, iterações de uma a quatro semanas
  - final da iteração, o cliente deve avaliar o software
  - final do ciclo, entrega do software

Métodos ágeis

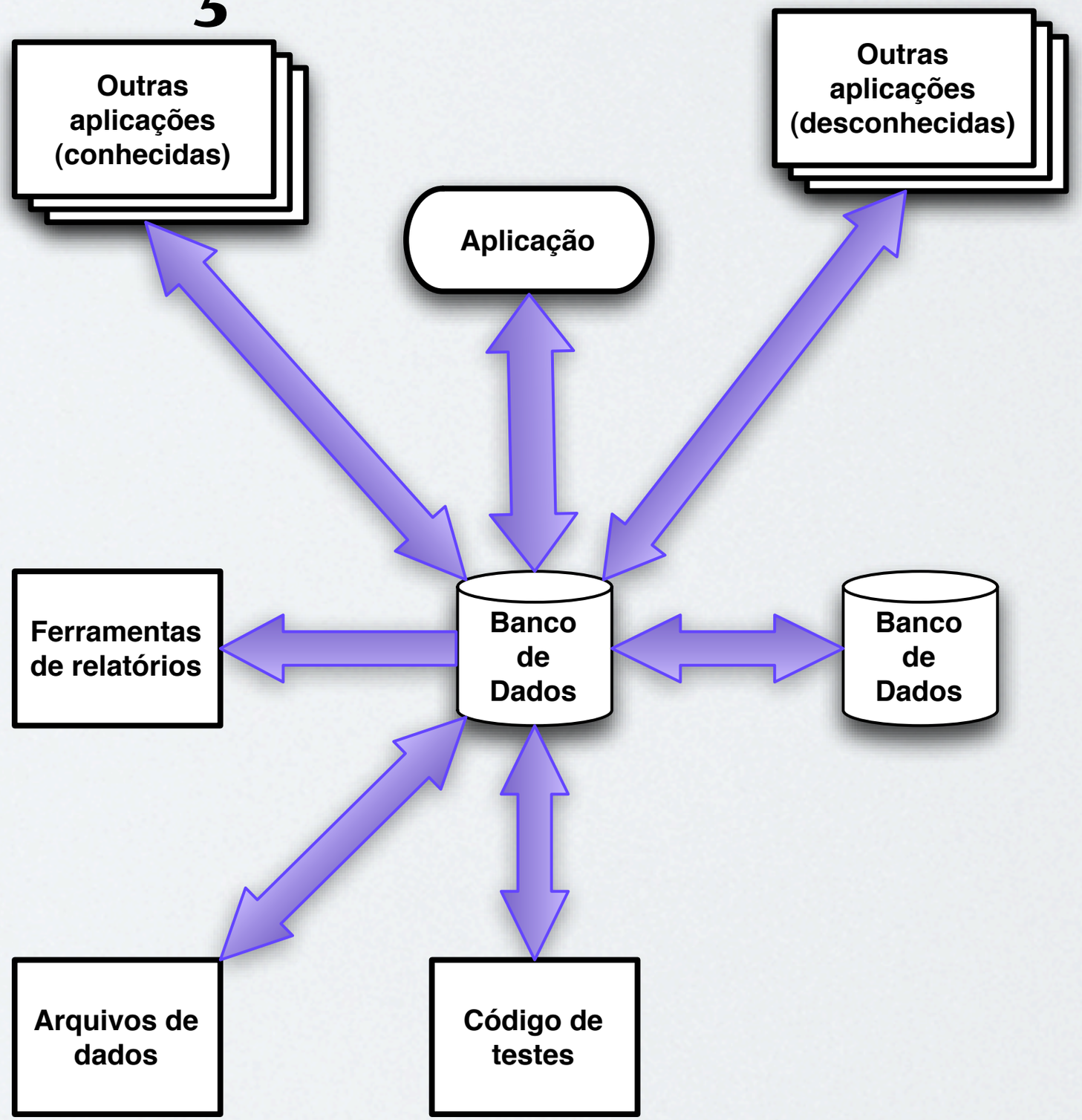
- Cada iteração tem todas as tarefas de desenvolvimento de software
- Métodos ágeis necessitam:
  - modelagem evolutiva -> BD evolutivo
  - refatoração de código -> **refatoração de BD**

Refatoração de BD

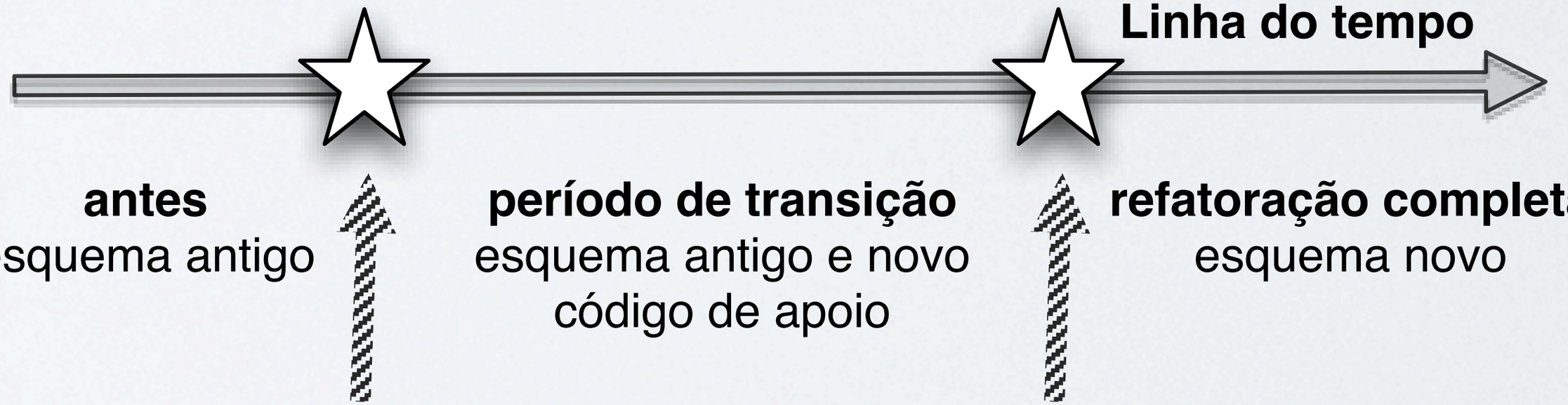
- 48 refatorações definidas por Scott Ambler
- Não acrescenta funcionalidade (“pequenas alterações”)
- Melhora o modelo
- Preserva a **semântica informacional**

Refatoração de BD

- Se preocupa com uma arquitetura complexa



Refatoração de BD



antes
esquema antigo

início
esquema novo
e código de apoio
funcionando

período de transição
esquema antigo e novo
código de apoio

refatoração completa
esquema novo

fim
esquema antigo e
código de apoio
removido

Limites da solução Ambler

- Codificação específica para cada refatoração
- Código para evitar circularidade
- Possível erro desconhecido para a aplicação
- Transação lenta, replicação síncrona de dados

Exemplo de refatoração

- Dividir tabela

Antes

Paciente
Id
Nome
Profissão
Salário
Paga Aluguel?

Durante

Paciente
Id
Nome
Profissão
Salário
Paga Aluguel?

Dados Sócio Econômicos
paciente_id
Profissão
Salário
Paga aluguel?

Período de transição
- esquema antigo obsoleto
- esquema novo disponível

Depois

Paciente
Id
Nome

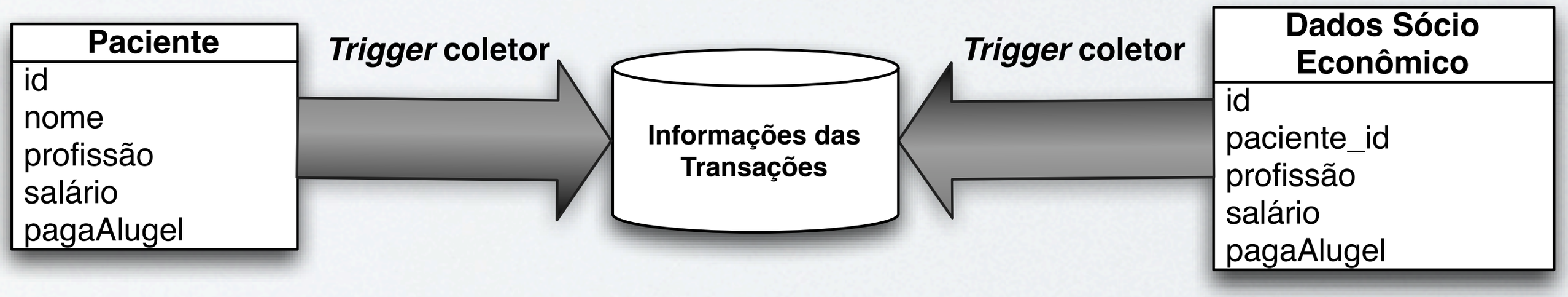
Dados Sócio Econômicos
paciente_id
Profissão
Salário
Paga aluguel?

Solução proposta

- Replicação assíncrona
- Divisão da replicação em três etapas:
  - 1 - Coleta do evento
  - 2 - Mapeamento
  - 3 - Execução da operação
- Desenvolvimento de um protótipo

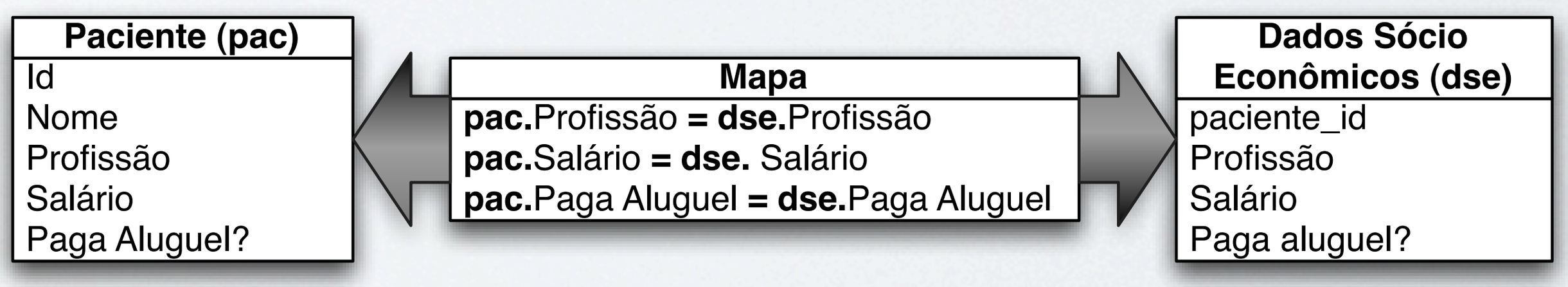
Solução proposta

- Etapa 1 - Coleta de eventos



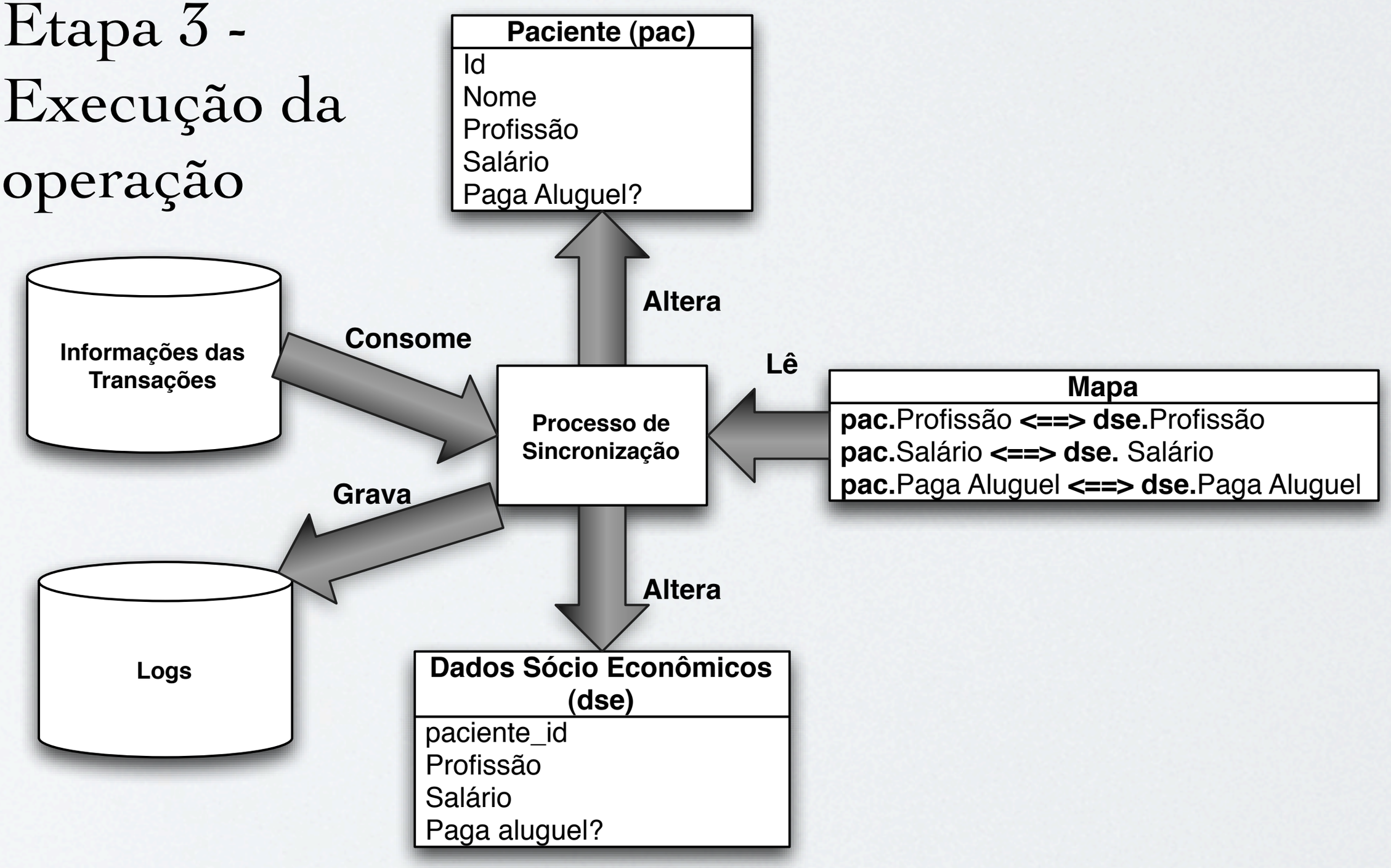
Solução proposta

- Etapa 2 - Mapeamento



Solução proposta

- Etapa 3 - Execução da operação



Avaliação

- **Limitação da solução de Ambler**
  - Codificação específica para cada refatoração
- **Solução proposta**
  - Temos três etapas para o código de apoio
  - Descrição do mapeamento (sem codificação)

Avaliação

- **Limitação da solução de Ambler**
  - Código para evitar circularidade em todos os triggers
- **Solução proposta**
  - Código de circularidade somente na etapa de execução de operação (fora da captação do evento)

Avaliação

- **Limitação da solução de Ambler**
  - Transação lenta (replicação síncrona)
- **Solução proposta**
  - A captura do evento é simples e a replicação é assíncrona

Avaliação

- **Limitação da solução de Ambler**
  - Possível erro desconhecido para a aplicação
- **Solução proposta**
  - Possível erro somente na etapa de execução da operação (geração de logs)

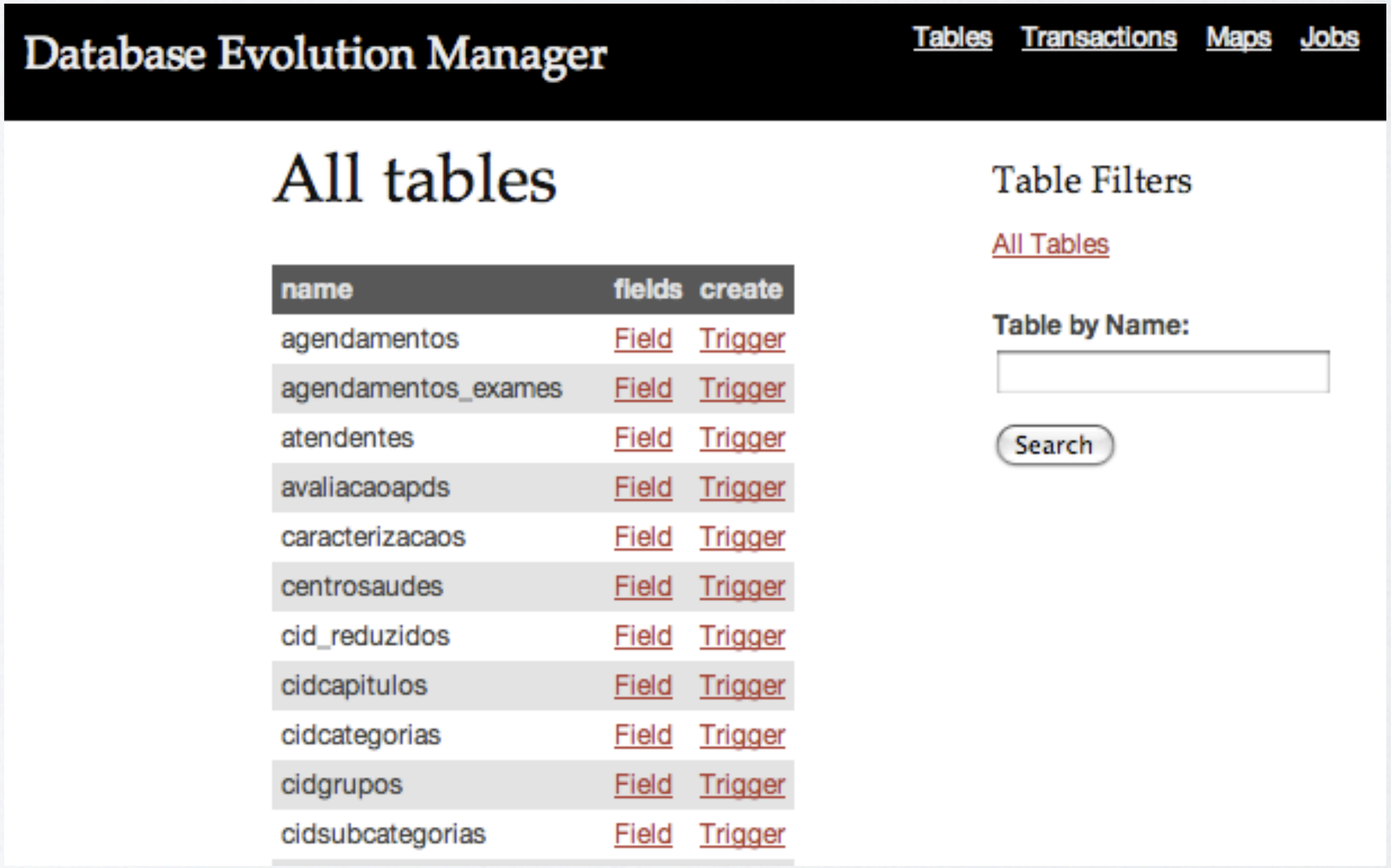
Avaliação

- **Limitação da solução proposta**
  - Os dados ficam inconsistentes até a etapa de execução da operação
- **Comentário:**
  - existem otimizações que minimizam esse limite

Protótipo

- Desenvolvido em Ruby on Rails
- Código aberto disponível do sítio do projeto
- Funcionalidades implementadas:
  - gera o código do coletor de eventos (trigger)
  - registra o mapeamento
  - executa as operações

Protótipo


Database Evolution Manager
Tables
Transactions
Maps
Jobs
All tables
name
fields
create
agendamentos
Field
Trigger
agendamentos_exames
Field
Trigger
atendentes
Field
Trigger
avaliacaoapds
Field
Trigger
caracterizacaos
Field
Trigger
centrosaudes
Field
Trigger
cid_reduzidos
Field
Trigger
cidcapitulos
Field
Trigger
cidcategorias
Field
Trigger
cidgrupos
Field
Trigger
cidsubcategorias
Field
Trigger
Table Filters
All Tables
Table by Name:
Search

Protótipo

Cria o *trigger* coletor

Database Evolution Manager

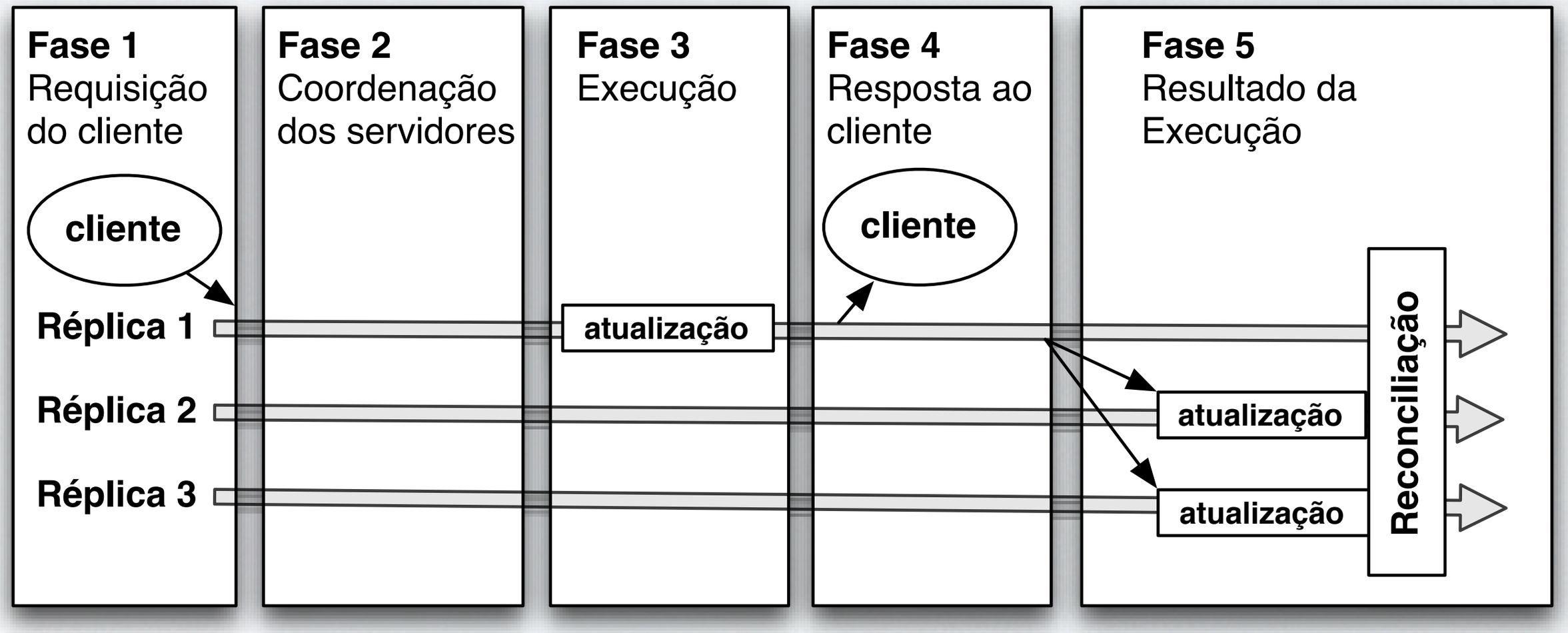
Tables Transactions Maps Jobs

Execute trigger code

```
--
-- Function and trigger to f_agendamentos_transactions() on UPDATE, INSERT, DELETE.
--
CREATE OR REPLACE FUNCTION f_agendamentos_transactions() RETURNS TRIGGER AS $f_agendamentos_transactions$
  DECLARE
  BEGIN

  IF (TG_OP = 'DELETE') THEN
--
-- INSERT DELETED ROW
--
INSERT INTO agendamentos_transactions (
status,operation,id_table,OLD_id_medico_responsavel,OLD_id_profissional_agendador,OLD_id_paciente,OLD_prazo_em_semanas,OLD_data_marcada,OLD_c
reated_at,OLD_updated_at,OLD_eh_gestante,OLD_data_registro,OLD_status_agendamento_id
)
values (
'NEW','DELETE',
OLD.id,OLD.id_medico_responsavel,OLD.id_profissional_agendador,OLD.id_paciente,OLD.prazo_em_semanas,OLD.data_marcada,OLD.created_at,OLD.updat
ed_at,OLD.eh_gestante,OLD.data_registro,OLD.status_agendamento_id
);

  ELSIF (TG_OP = 'UPDATE') THEN
--
-- INSERT UPDATED ROW
--
INSERT INTO agendamentos_transactions (
status,operation,id_table,NEW_id_medico_responsavel,NEW_id_profissional_agendador,NEW_id_paciente,NEW_prazo_em_semanas,NEW_data_marcada,NEW_
created_at,NEW_updated_at,NEW_eh_gestante,NEW_data_registro,NEW_status_agendamento_id,OLD_id_medico_responsavel,OLD_id_profissional_agendador
,OLD_id_paciente,OLD_prazo_em_semanas,OLD_data_marcada,OLD_created_at,OLD_updated_at,OLD_eh_gestante,OLD_data_registro,OLD_status_agendam
ento_id
)
values (
'NEW','UPDATE',
NEW.id,NEW.id_medico_responsavel,NEW.id_profissional_agendador,NEW.id_paciente,NEW.prazo_em_semanas,NEW.data_marcada,NEW.created_at,NEW.upd
ated_at,NEW.eh_gestante,NEW.data_registro,NEW.status_agendamento_id,OLD.id_medico_responsavel,OLD.id_profissional_agendador,OLD.id_paciente,OLD.
prazo_em_semanas,OLD.data_marcada,OLD.created_at,OLD.updated_at,OLD.eh_gestante,OLD.data_registro,OLD.status_agendamento_id
);

  ELSIF (TG_OP = 'INSERT') THEN
--
```

Trabalhos relacionados

- [Wiesmann 2000] Replicação de dados



Trabalhos relacionados

- [Windom and Ceri 2006] Banco de dados ativos - Condição/Evento/Ação - Trigger
- ETL tempo real e mais recente, CDC (Change Data Capture)
- [Curino et. al 2008] Evolução de esquemas - Reescrita automática de consultas

Conclusão

- A solução proposta é uma alternativa assíncrona para manter os esquemas atualizados.
- Não exige reescrita dinâmica de consultas
- Não exige manter todo o histórico de esquemas (período de transição)

Conclusão

- Trabalhos futuros:
  - evolução do protótipo
  - reconciliação automatizada
  - combinações de refatorações para mudanças da semântica informacional

Agradecimentos

- O Projeto Borboleta é financiado pelo Instituto Virtual Microsoft Research-FAPESP (Processo 07/54479-4)
- CCSL - Centro de Competência em Software Livre
- Laboratório de BD e Aplicações IME-USP
- Departamento de Ciência da Computação IME - USP

Dúvidas?

Helves Humberto Domingues
helves@ime.usp.br

URL do Projeto:
http://csl.ime.usp.br/borboleta/DatabaseEvolution