Evolução de Bancos de Dados em Métodos Ágeis

Minicurso SBES/SBBD'2008, Campinas

Helves Domingues, João Eduardo Ferreira e Fabio Kon
Departamento de Ciência da Computação
IME / USP

Nossa experiência

- Cursos de graduação em XP desde 2001
- Apresentações
  - SBES, SUCESU, SEPAI, SEBRAE, etc.
  - Arquivos disponíveis: www.agilcoop.org.br
- Assessorias em métodos ágeis
- Artigos científicos
- Dissertações de Mestrado

Roteiro

- Motivação
  - Problemas e possíveis direções
- Métodos ágeis
  - Princípios
  - Problemas com os métodos tradicionais
  - Alguns métodos ágeis
  - Métodos Ágeis e Bancos de Dados
- Bancos de Dados e Métodos Ágeis
- Evolução e Refatoração de Bancos de Dados

Novos ventos no mundo do Desenvolvimento de Software

- Sociedade demanda
  - grande quantidade de sistemas/aplicações
  - software complexo, sistemas distribuídos, heterogêneos
  - requisitos mutantes (todo ano, todo mês, todo dia)

- Mas, infelizmente,
  - não há gente suficiente para desenvolver tanto software com qualidade.

Problemas

- Com métodos tradicionais de desenvolvimento
  - Supõem que é possível prever o futuro
  - Pouca interação com os clientes
  - Ênfase em burocracias
    - (documentos, formulários, processos, controles rígidos, etc.)
  - Avaliação do progresso baseado na evolução da burocracia e não do código
- Com software
  - Grande quantidade de erros
  - Falta de flexibilidade

Como resolver esse impasse?

- Melhores Tecnologias
  - Padrões de Projeto (reutilização de idéias)
  - Componentes (reutilização de código)
  - Middleware (aumenta a abstração)

- Melhores Metodologias
  - Métodos Ágeis (o foco desta apresentação)
  - outras... (abordadas em ES)

O que é desenvolvimento de sofware?

- Por Alistair Cockburn:

Modelagem	(Jacobson)
Engenharia	(Meyer)
Disciplina	(Humphreys)
Poesia	(Cockburn)
Artesanato	(Knuth)
Arte	(Gabriel)

- Erro comum: olhar para software como apenas um desses itens e ignorar os demais

Métodos Ágeis de Desenvolvimento de Software

- Movimento iniciado por programadores experientes e consultores em desenvolvimento de software.
- Questionam e se opõem a uma série de mitos/ práticas adotadas em abordagens tradicionais de Engenharia de Software e Gerência de Projetos.
- Manifesto Ágil:
  - Assinado por 17 desenvolvedores em Utah em fevereiro/2001.
  - http://agilemanifesto.org

O Manifesto do *Desenvolvimento Ágil de Software*

- **Indivíduos e interações** são mais importantes que processos e ferramentas.

- **Software funcionando** é mais importante do que documentação completa e detalhada.

- **Colaboração com o cliente** é mais importante do que negociação de contratos.

- **Adaptação a mudanças** é mais importante do que seguir o plano inicial.

Princípios do Manifesto Ágil

- Objetivo: satisfazer o cliente entregando, rapidamente e com freqüência, sistemas com algum valor.
  - Entregar versões funcionais em prazos curtos.
  - Estar preparado para requisitos mutantes.
  - Pessoal de negócios e desenvolvedores juntos.
  - Troca de informações através de conversas diretas.

Mais princípios do Manifesto Ágil

- Medir o progresso através de software funcionando.
- Desenvolvimento sustentável.
- Construir projetos com indivíduos motivados.
- Simplicidade é essencial.
- De tempos em tempos, o time pensa em como se tornar mais eficiente e ajusta o seu comportamento de acordo.

Em um projeto ágil ideal…. (1/2)

- O gerente de projeto concorda em prosseguir sem que todos os requisitos estejam bem definidos.

- Os desenvolvedores concordam em prosseguir sem ter todos os requisitos documentados.

- Os membros da equipe sabem que alguém vai ajudar quando ocorrerem problemas.

Em um projeto ágil ideal….(2/2)

- Os gerentes percebem que não precisam dizer à equipe o que fazer, ou garantir o que vai ser feito.

- A equipe percebe que ninguém vai dizer o que fazer, isto faz parte do trabalho da equipe.

- Não existem mais a impressão de divisão (*testers and programmers*), todos são desenvolvedores.

Enquanto isso, no mundo tradicional...

0. Levantamento de Requisitos

1. Análise de Requisitos

2. Desenho da Arquitetura

3. Implementação

4. Testes

5. Produção / Manutenção

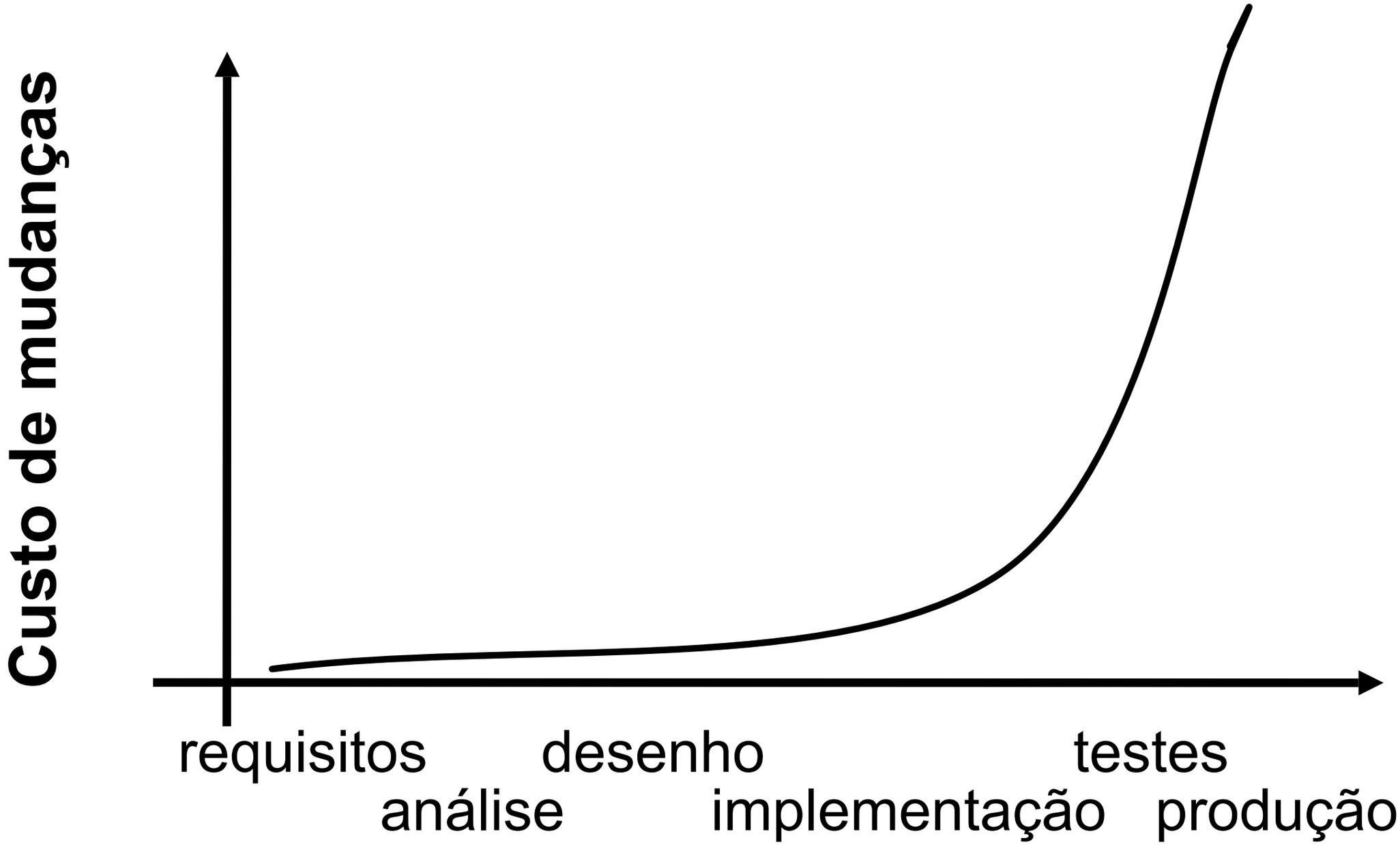
Premissas Básicas do ModeloTradicional

- É necessário fazer uma análise de requisitos profunda e detalhada antes de projetar a arquitetura do sistema.

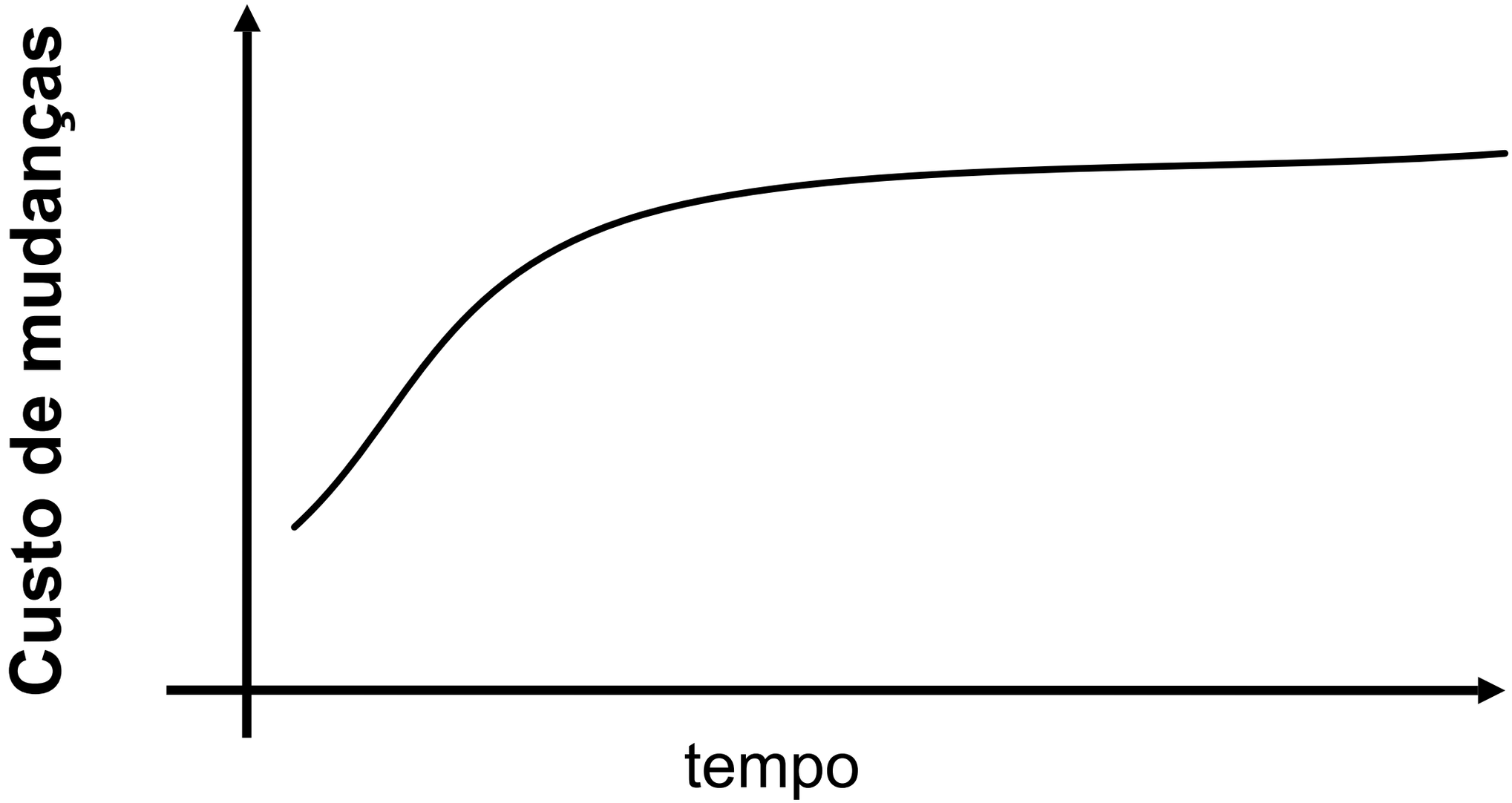
- É necessário fazer um estudo minucioso e elaborar uma descrição detalhada da arquitetura antes de começar a implementá-la.

- É necessário testar o sistema completamente antes de mandar a versão final para o cliente.

O que está por trás deste modelo?



E se a realidade hoje em dia fosse outra?



E se essa fosse a realidade?

- A atitude dos desenvolvedores de software seria completamente diferente:
  - Tomaríamos as grandes decisões o mais tarde possível.
  - Implementaríamos agora somente o que precisamos *agora.*
  - Não implementaríamos flexibilidade desnecessária (não anteciparíamos necessidades).

E essa é a nova realidade ! (pelo menos em muitos casos)

- **Orientação a Objetos**: facilita e cria oportunidades para mudanças.

- **Técnicas de Refatoração**.

- **Testes automatizados**: nos dão segurança quando fazemos mudanças.

- **Prática / cultura de mudanças**: aprendemos técnicas e adquirimos experiência em lidar com código mutante.

Principais Métodos Ágeis

- Programação eXtrema (XP)
  - O preferido dos desenvolvedores
- Scrum
  - O preferido dos gerentes
- Crystal (uma família de métodos)
- Lean Software Development
- Adaptive Software Development
- Feature Driven Development

Práticas Ágeis

- Planejamento adaptativo (cliente presente)
  - Histórias
  - Detalhamento apenas quando necessário
- Programação em pares (redundância)
- Testes Automatizados (TDD)
- Refatoração (melhoria contínua)
- Pequenos passos
  - ciclo semanal
  - design incremental
  - implantação incremental
- Integração Contínua (repositório único de código)

Refatoração (Refactoring)

- Uma [pequena] modificação no sistema que não altera o seu comportamento funcional

- mas que melhora alguma qualidade não-funcional:
  - simplicidade
  - flexibilidade
  - clareza
  - desempenho

Exemplos de Refatoração

- Mudança do nome de variáveis
- Mudanças nas interfaces dos objetos
- Pequenas mudanças arquiteturais
- Encapsular código repetido em um novo método
- Generalização de métodos
  - raizQuadrada(float x)⇒ raiz(float x, int n)

Com o código tudo vai bem

- Programadores se habituam e passam a trabalhar com mais
  - prazer
  - motivação
  - produtividade
  - qualidade

- Ferramentas boas para
  - Testes Automatizados / Cobertura
  - Refatoração
  - Coleta de Métricas / Acompanhamento
  - Planejamento e Gerenciamento do Projeto

Com o Banco de Dados a coisa complica

- Conflitos programadores vs DBAs
- Faltam ferramentas boas para
  - Testes Automatizados
  - Refatoração
  - Coleta de Métricas / Acompanhamento
  - Planejamento e Gerenciamento do Projeto

Bancos de Dados em Métodos Ágeis

- Refatorar Bancos de Dados é MUITO mais difícil do que refatorar código
  - faltam ferramentas
  - falta experiência com metodologia bem estruturada
- Quando é possível parar ou substituir o sistema completamente não é tão grave.
- Quando apenas uma aplicação simples acesso o banco não é tão grave.

- Mas em sistemas complexos reais, não é o caso.
  - É como reformar um avião em pleno vôo...

Preparando o terreno para compreensão de banco de dados evolutivos

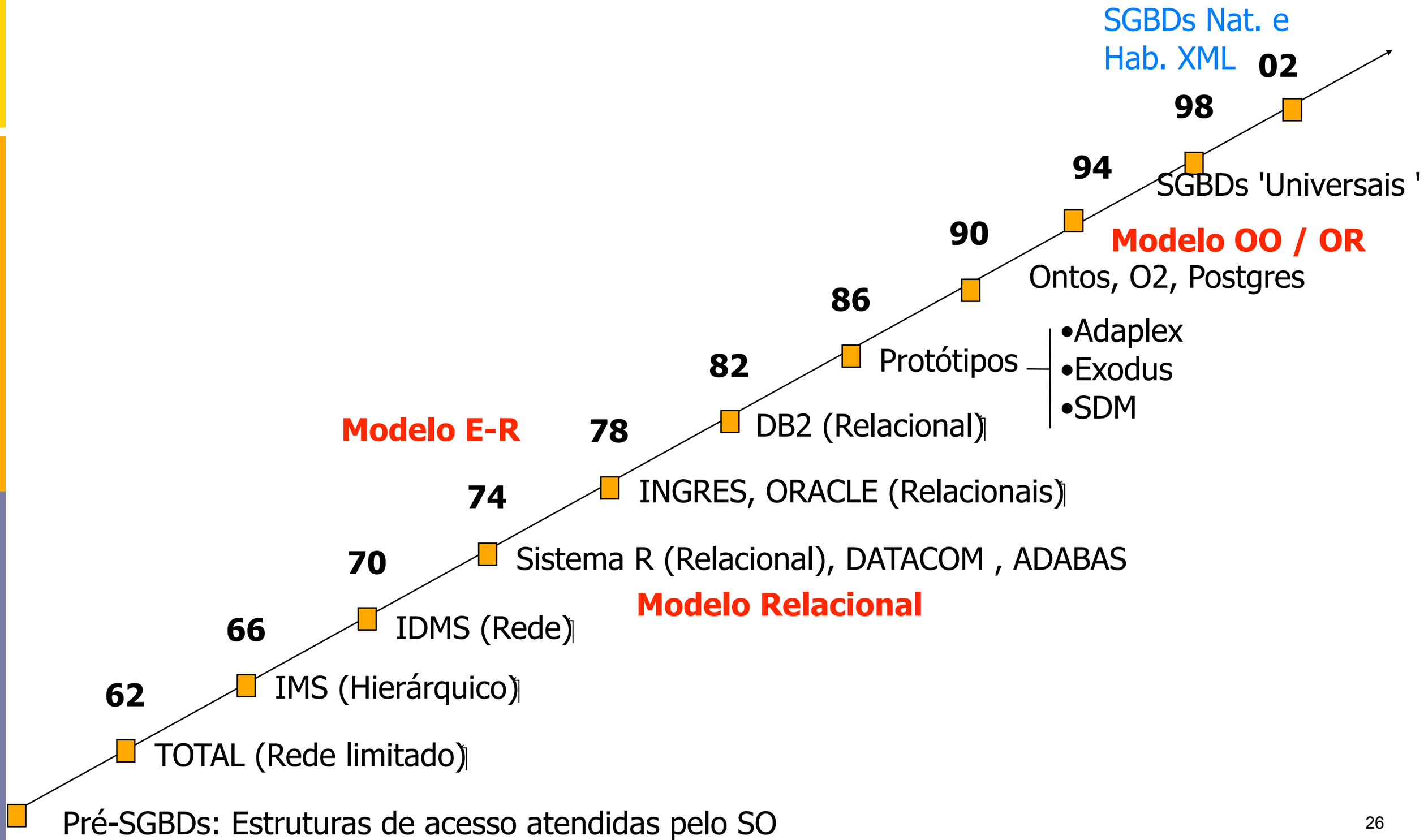
Introdução

Técnicas de persistência de dados

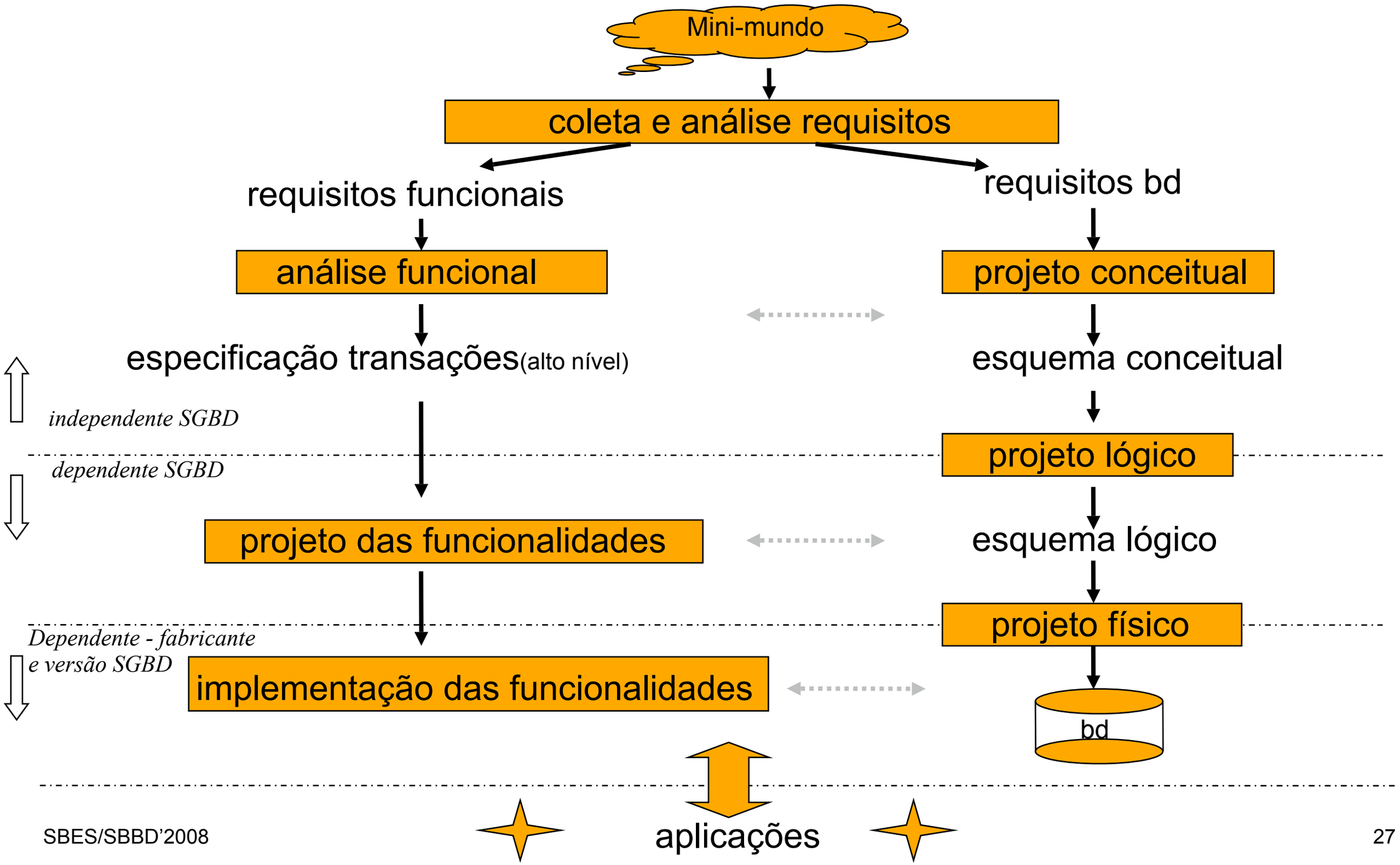
Estratégia para refatoração de BD

Prof. João Eduardo Ferreira

Evolução dos Modelos e SGBDs



Projeto Convencional



Definições

- Os objetos do Modelo de Objetos de Domínio:
  - Representam os principais estados e comportamentos da aplicação.
  - Normalmente:
    - são compartilhados por vários usuários simultaneamente.
    - são armazenados e recuperados entre as execuções da aplicação.
  - A capacidade desses objetos de sobreviverem além do tempo de execução da aplicação é chamada de **Persistência de Objetos**.

Definições

- A persistência de objetos não é exclusividade dos objetos do Modelo de Objetos de Domínio:
  - Por exemplo, um objeto da classe LogDeMensagens pode ter esta característica.
- Mas a maioria dos objetos persistentes de uma aplicação encontram-se no Modelo de Objetos de Domínio.

Repositórios para Persistência

- A persistência precisa armazenar o estado dos objetos em algum repositório para futuramente recuperá-los.
- Os repositórios podem ser:
  - Um BD relacional (mais comum).
  - Arquivos do sistema.
  - Um BD OO.

Técnicas de Persistência

Obj/Rel

SBES/SBBD'2008
Copyleft by Helves, Jef & Fabio

Técnicas de Persistência – Obj/Rel

- A maioria dos projetos de desenvolvimento de software utiliza:
  - a linguagem OO, tais como Java e C#.
  - o BD relacional para armazenar dados.
- O desenvolvimento de aplicações que usam linguagens OO e DB Relacional enfrentam o problema da **incompatibilidade conceitual** (*impedance mismatch*).
- Para superar este problema é importante conhecer:
  - O processo de mapeamento objeto-relacional.
  - Como implementar mapeamento objeto-relacional.

Técnicas de Persistência - Força Bruta

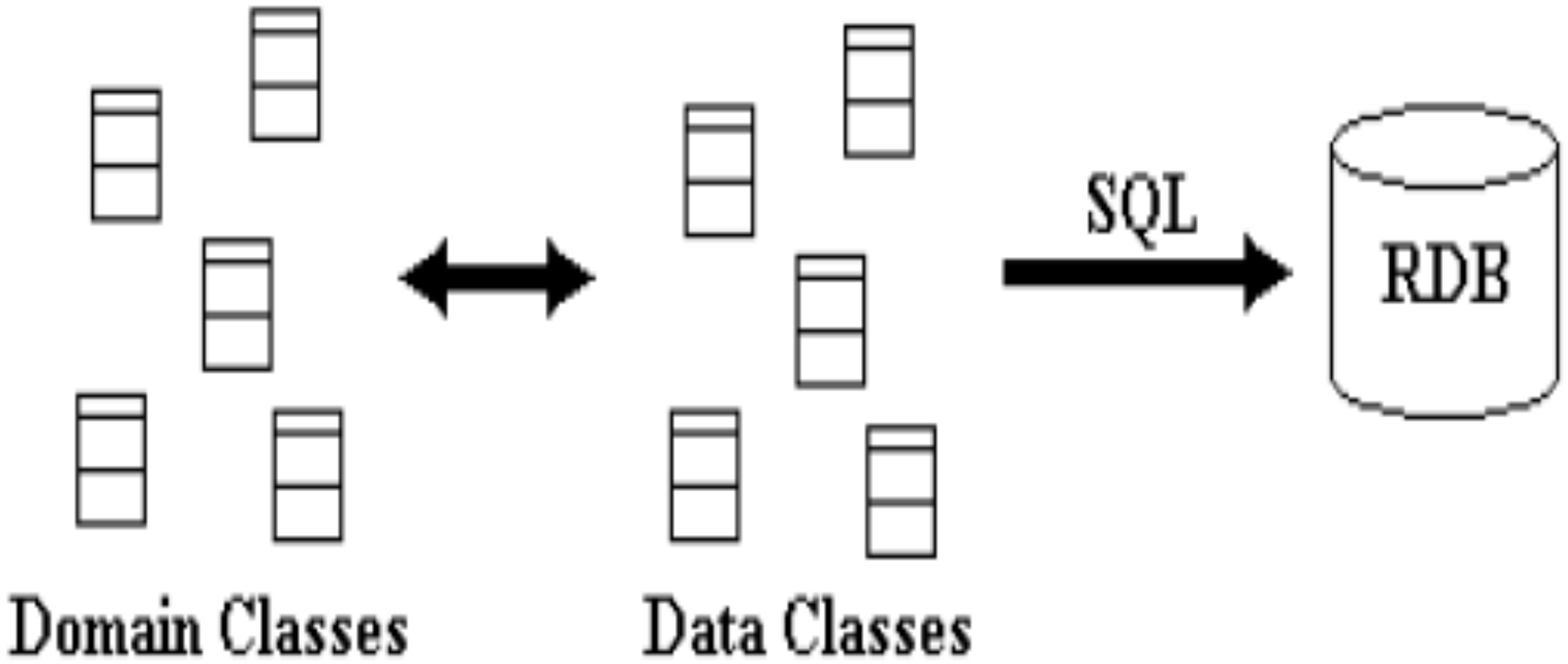
- Força Bruta (mais comum):
  - o código SQL é embutido no código-fonte das classes.
  - Vantagens:
    - permite escrever códigos muito rapidamente
    - viável para pequenas aplicações ou protótipos.
  - Desvantagens:
    - Alto acoplamento entre as classes de domínio e o esquema do banco de dados.
    - Mudanças simples no BD (por exemplo, renomear uma coluna) resulta na manutenção do código-fonte OO.

Técnicas de Persistência - Força Bruta

Técnicas de Persistência - Classes de Dados

- Classes de Dados:
  - SQL das classes de domínio são encapsuladas nas "classes de dados". Exemplos:
    - SP's no BD representam objetos (substituindo as classes de dados)
    - ActiveX Data Objects (ADO).
  - Vantagens:
    - Viável para protótipos e pequenos sistemas (40 classes de negócio).
  - Desvantagens:
    - Recompilação das classes de dados quando pequenas mudanças são feitas no BD.

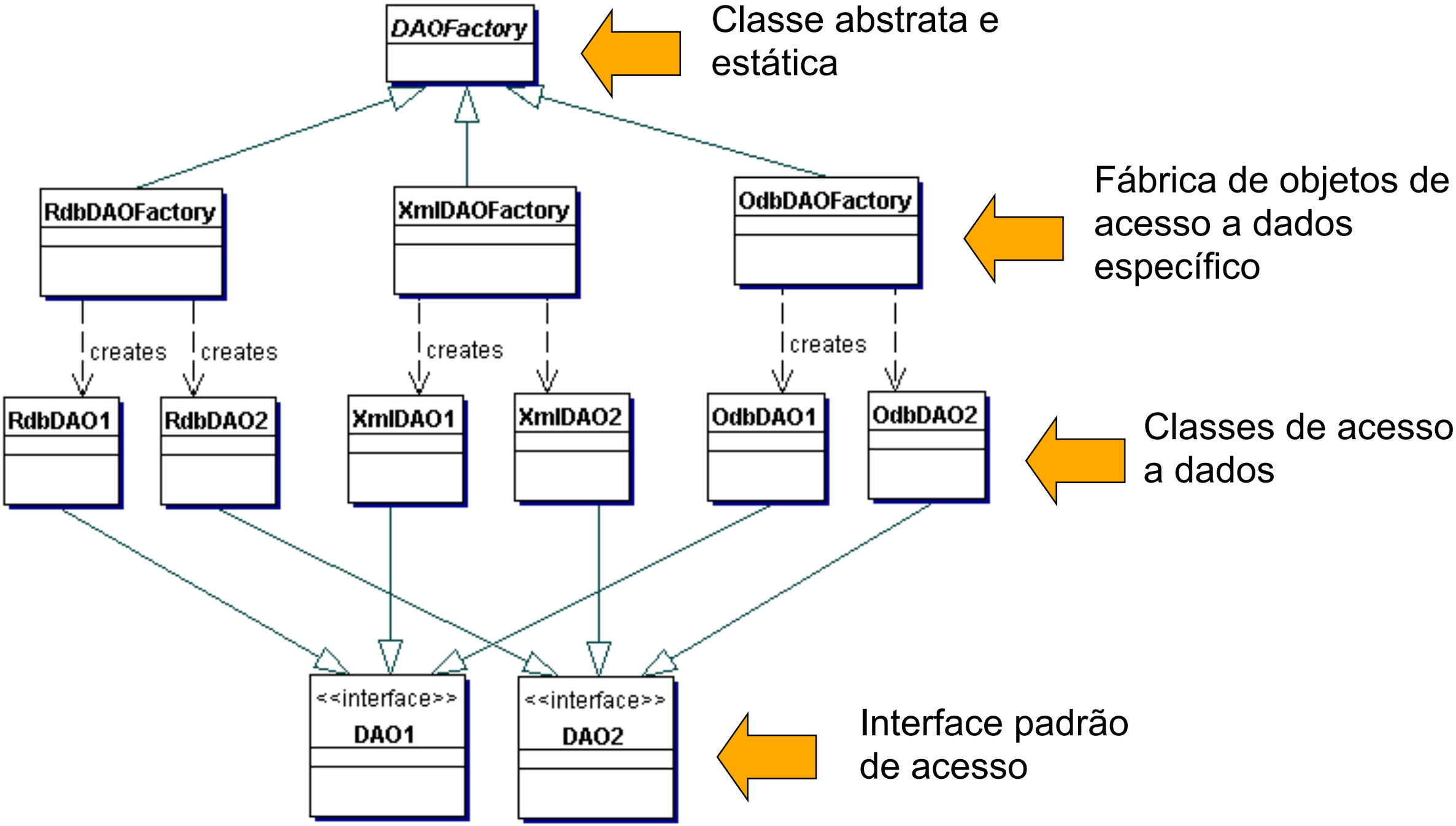
Técnicas de Persistência - Classes de Dados



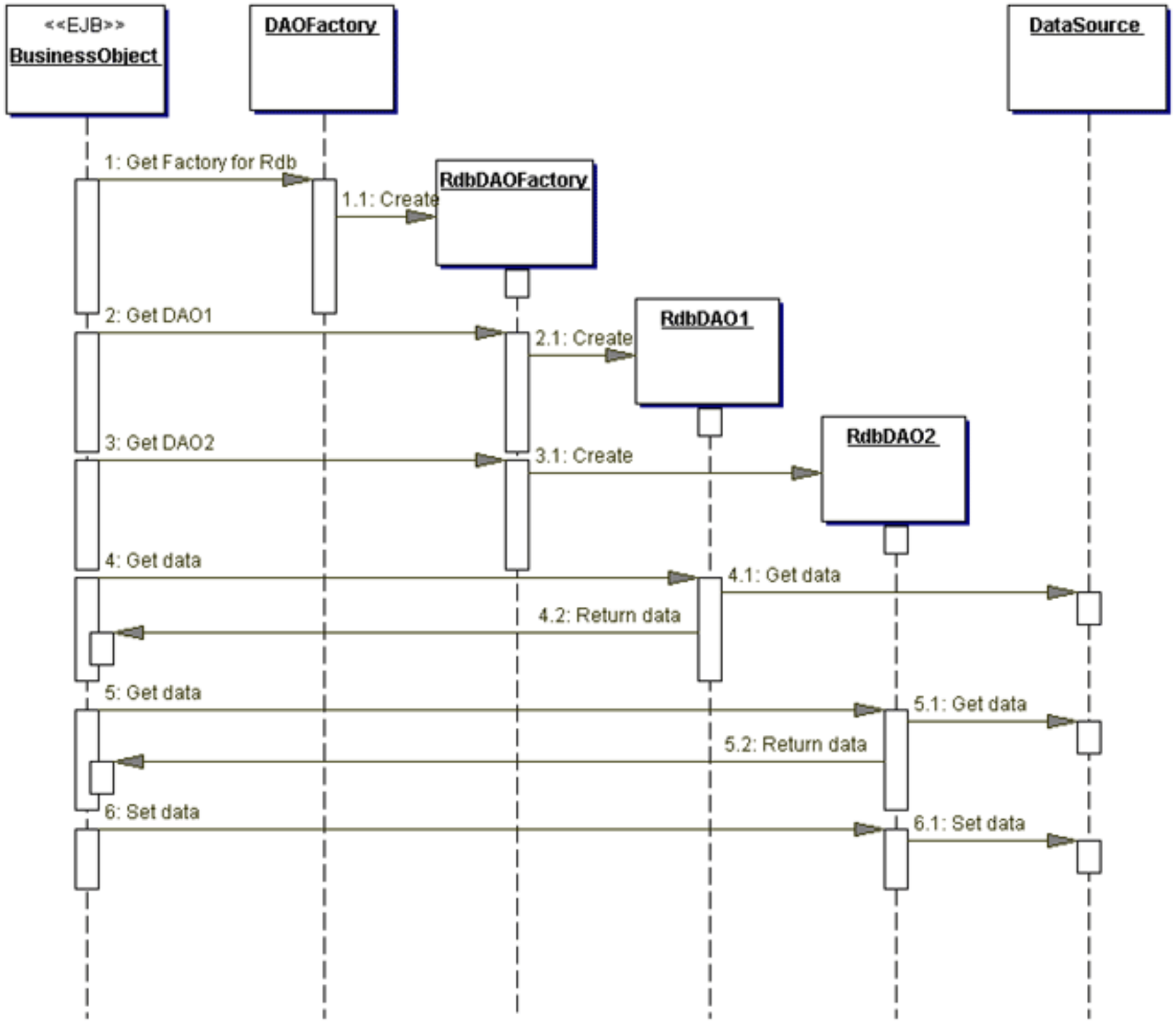
Técnicas de Persistência – Classes de Dados
O Padrão DAO

- Este padrão permite criar as classes de dados independentemente da fonte de dados ser um BD relacional.
- Para isso, encapsula os mecanismos de acesso a dados e cria uma interface de cliente genérica para acessar os dados.
- Dessa forma, o DAO permite que os mecanismos de acesso a dados mudem independentemente do código que usa o dado.

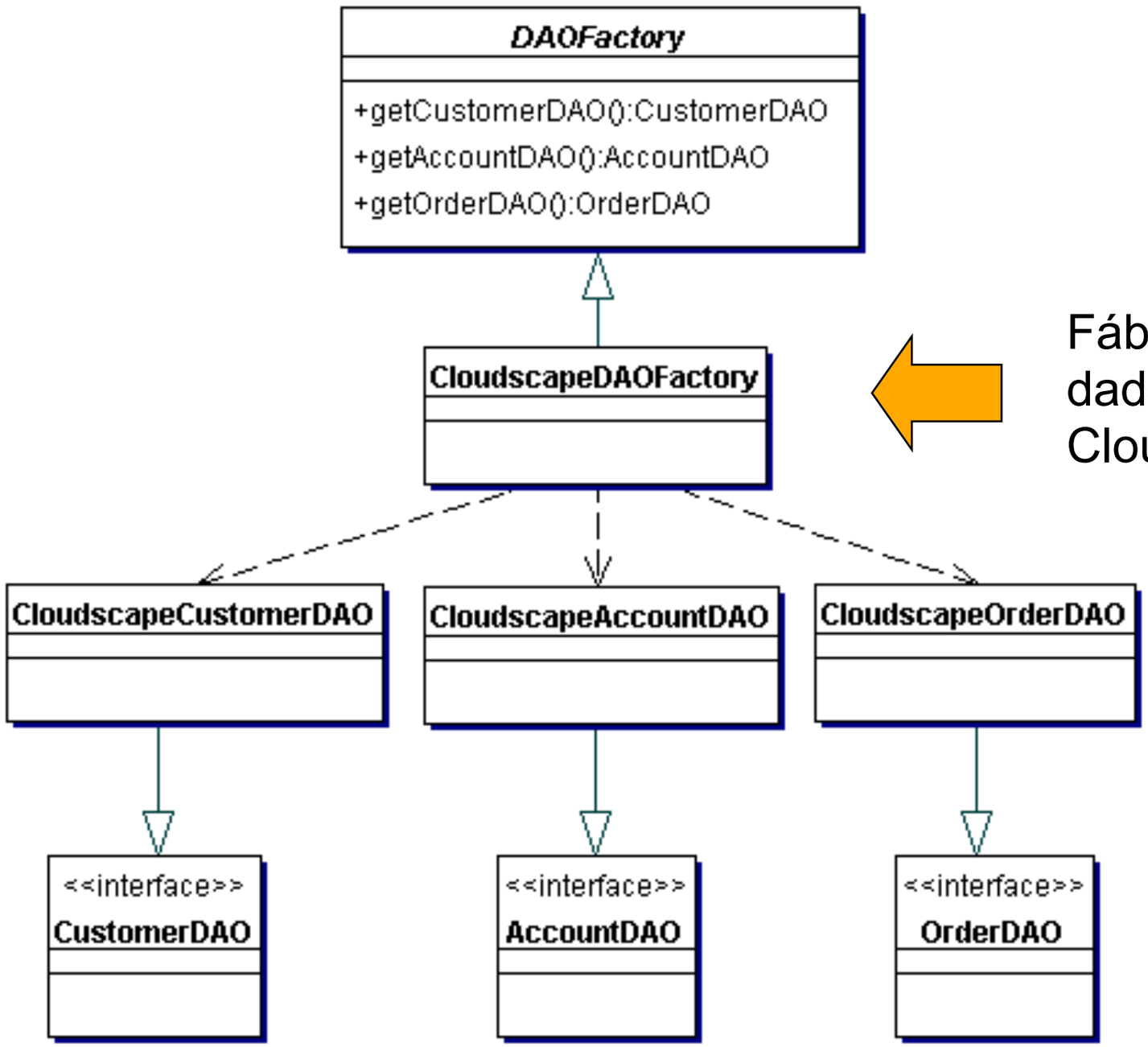
Técnicas de Persistência – Classes de Dados
O Padrão DAO



Técnicas de Persistência – Classes de Dados
O Padrão DAO



Técnicas de Persistência – Classes de Dados
O Padrão DAO



Fábrica de objetos de acesso a dados específico para o BD Cloudscape

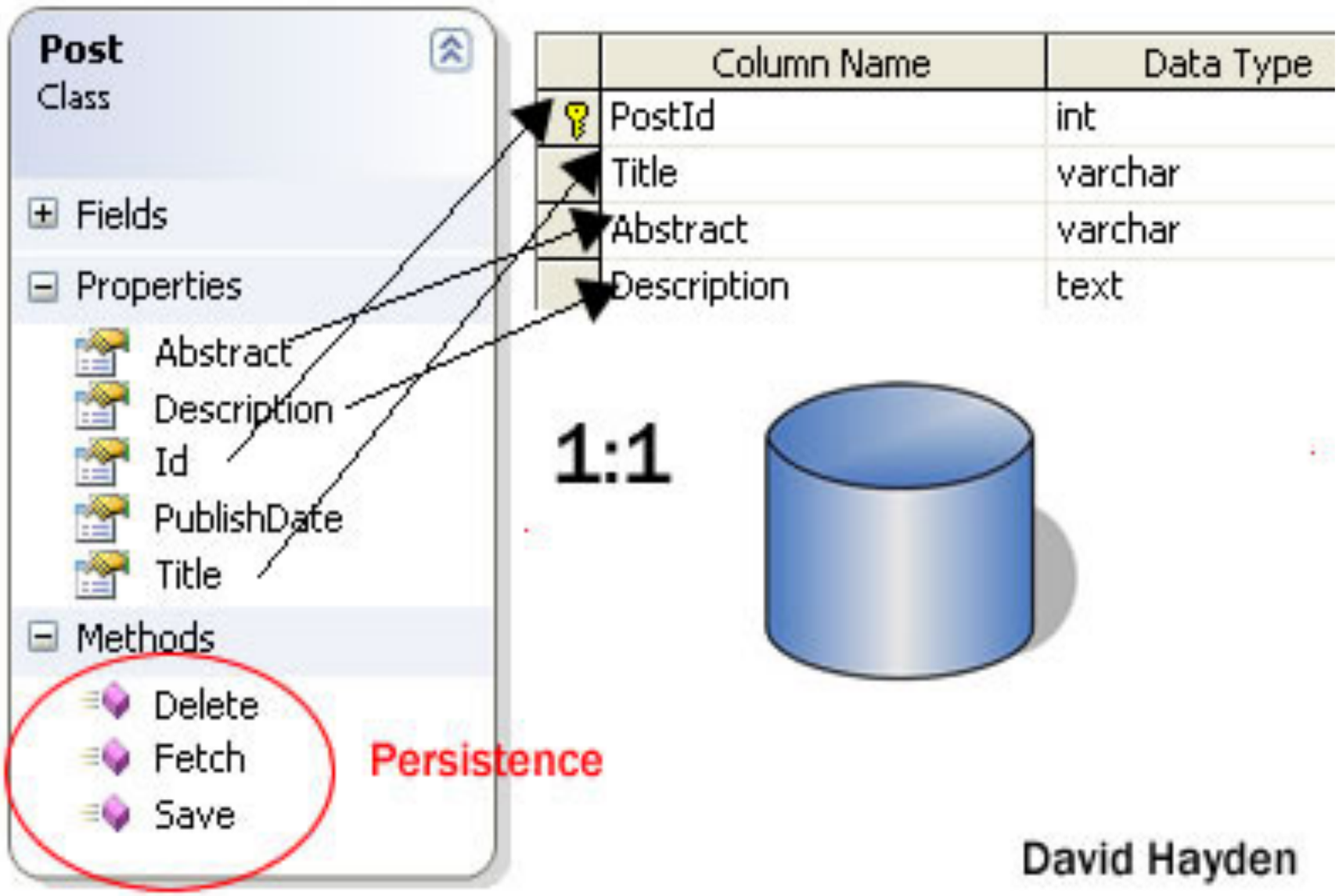
Técnicas de Persistência – Classes de Dados
O Padrão DAO

- Estratégia para desenvolvimento rápido usando o padrão DAO com BD relacionais:
  - Gerador de DAOs
    - http://www.akcess.in/download.html
    - http://www.codefutures.com/products/firestorm

Técnicas de Persistência – Classes de Dados
O Padrão Active Record

- Objeto de negócio ⇔ linha de tabela ou visão
- Encapsula o acesso ao BD
- Lógica de domínio ∈ Objetos de negócio
- Ideal quando:
  - existirem poucas regras de negócio
  - for possível manter relacionamento 1:1 entre propriedades do objeto de negócio com colunas de uma tabela
  - não houver necessidade de mapeamento adicional

Técnicas de Persistência – Classes de Dados
O Padrão Active Record



Técnicas de Persistência – Classes de Dados
O Padrão Active Record

- Com o esse padrão, focamos a persistência como uma responsabilidade ao invés de serviço.
- Operações CRUD são mapeados para métodos de instância estáticos do objeto de domínio.

Técnicas de Persistência – Classes de Dados
O Padrão Active Record

- Gerador de Active Record
  - http://subsonicproject.com
- API - jPersist
  - http://www.jwebapp.org

Técnicas de Persistência

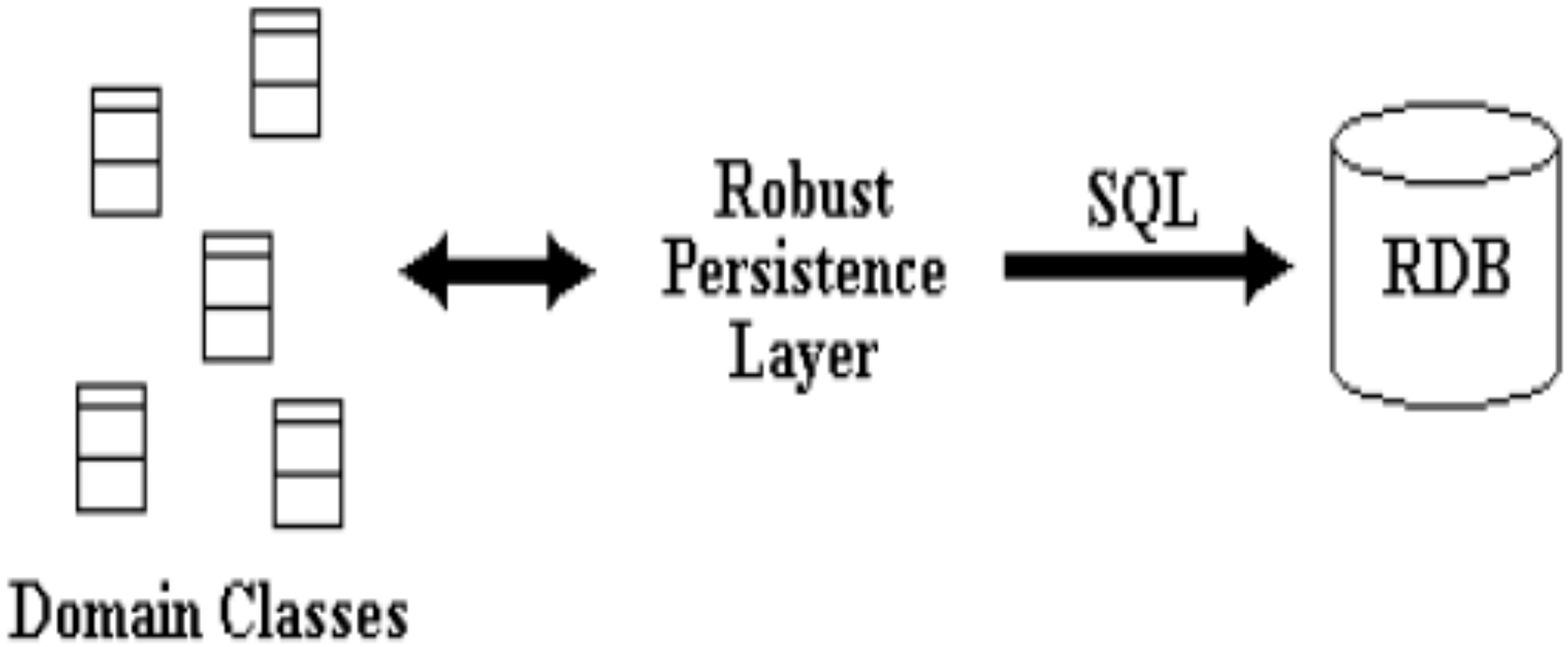
Framework

SBES/SBBD'2008
Copyleft by Helves, Jef & Fabio

Técnicas de Persistência – *Framework*

- Camada de Persistência Robusta (*framework*):
  - O *framework* mapeia objetos para bancos de dados relacionais, de maneira que as pequenas mudanças no esquema relacional não afetem o código orientado a objetos.
  - Vantagens:
    - Programadores não precisam conhecer nada a respeito do esquema do BD relacional.
    - Permite desenvolvimento de aplicações em larga escala.
  - Desvantagem:
    - Impacto no desempenho das aplicações.
    - Propicia degeneração conceitual com impacto negativo no modelo de banco de dados.

Técnicas de Persistência – *Framework*



Java Data Objects

Persistência com JDO

O que é JDO?

- O JDO (*Java Data Objects*) é um padrão para persistência transparente para objetos Java definida pela JSR-000012/JCP.
  - Fornece aos desenvolvedores uma visão de persistência e acesso ao repositório de dados centrada em objetos Java.
- Atualmente na versão 2.0.
- Permitir “plugar” drivers de diferentes fornecedores para acessar qualquer DB/repositório de dados.
- Trabalha em conjunto com *Application Servers*.

As Metas do JDO

- Persistência Transparente de Objetos
  - Reduzir para zero o número de restrições para construir classes
  - Nenhuma nova linguagem de acesso a dados
    - Java é a DDL & DML
- Disponibilidade para criar várias implementações
  - J2ME - Embedded, device-oriented
  - J2SE - Client/server
  - J2EE - Enterprise Java Beans
- Independência do repositório de dados
  - Relacional, objeto, objeto-relacional, hierárquico, sistema de arquivos, etc.

Audiência do JDO

- Desenvolvedores de aplicações Java
  - Persistência Transparente de Objetos
  - Centrado em Java, não necessitando conhecer como acessar um BD
- Desenvolvedores de aplicação EJB
  - Gerenciamento de transação e de conexões via Application Server
  - Acesso ao BD transparente arente para soluções não CMP (Session Beans & BMP)
    - Não há necessidade de usar diretamente o JDBC
    - OQL para encontrar instâncias

Comparação entre JDO, Seriação e JDBC

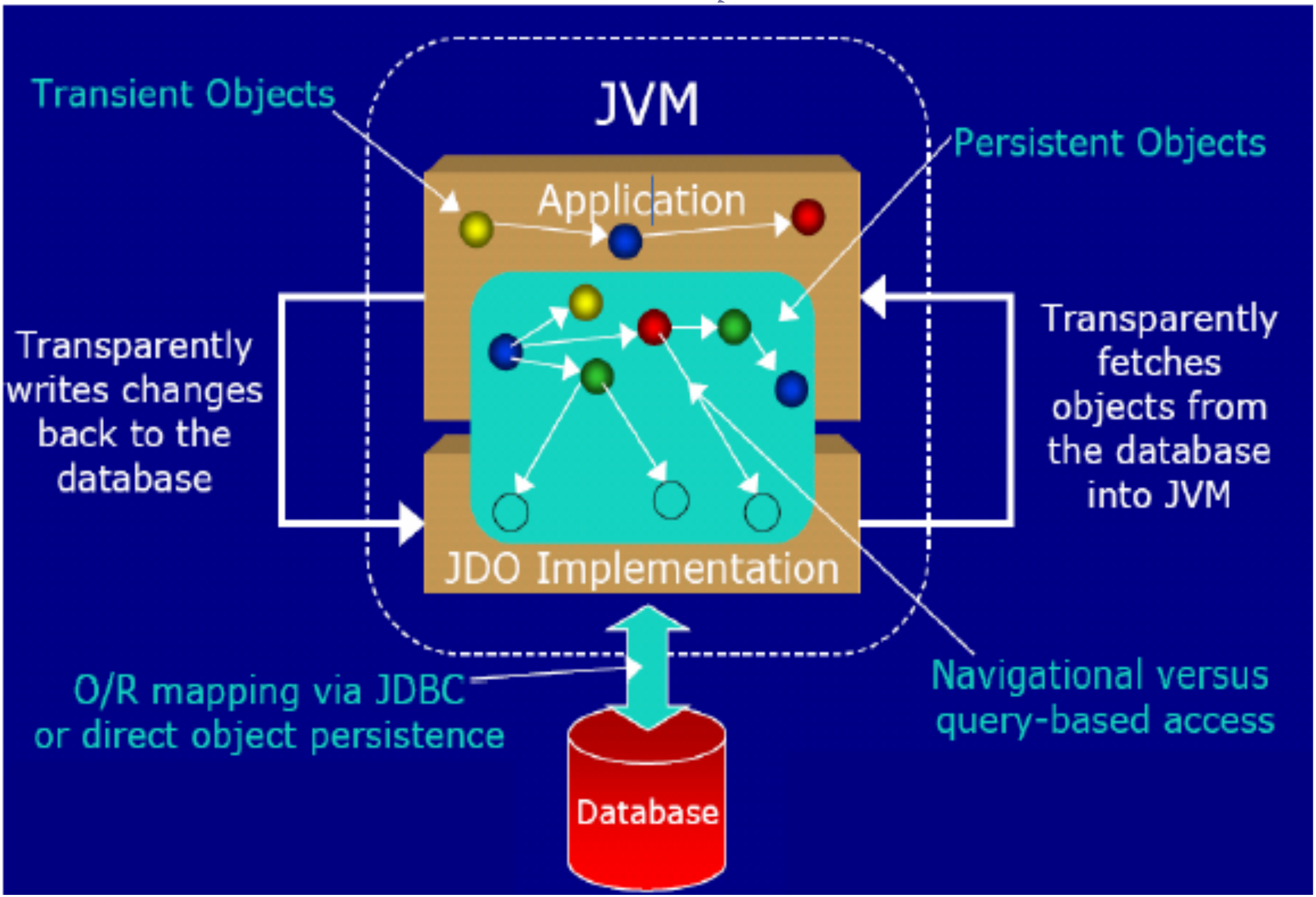
Feature	Serialization	JDBC	JDO
Data Model	Java	Relational table model	Java
Support of Java Classes	✔	✘	✔
Access granularity	Object Graph	Table cell	Object
Support of inheritance and polymorphism	✔	✘	✔
Support of references and collections	✔	✘	✔
Unique identity	✘	Primary key	Application or Datastore Identity
Automatic management of cache	✘	✘	✔
Transactions	✘	✔	✔
Concurrency	✘	✔	✔
Query Language	✘	SQL, each vendor has a different dialect (not portable)	JDOQL, standard language, Java-like syntax
Object model supported in queries	✘	✘	✔
JCA compatibility for application server integration	✘	JDBC 3.0, 4.0	✔

Fonte: Comparação entre JDO, Seriação e JDBC (JDOCentral.com)

Como o JDO Funciona?

- O JDO permite que as aplicações armazenem e recuperem “transparentemente” instâncias de qualquer classe Java de um repositório de dados.
- Para a aplicação, os “objetos persistentes” se parecem como objetos Java normais, residentes na memória.
  - Os campos dessas instâncias na realidade estão armazenadas em algum repositório de dados, persistentemente – mas sem qualquer ação explícita da aplicação.
- O JDO não precisa saber onde os métodos são executados.
  - Ele não possui meios de chamar remotamente, como no RMI ou EJB, métodos de objetos em repositórios de dados.

Ambiente de Execução JDO



Responsabilidades do Desenvolvedor

- Determinar quais objetos do modelo de domínio precisam ser persistidos:
  - Marcar essas classes como ‘capazes de persistência’ (persistence capable)
  - Usar a API JDO para persistir instâncias dessas classes

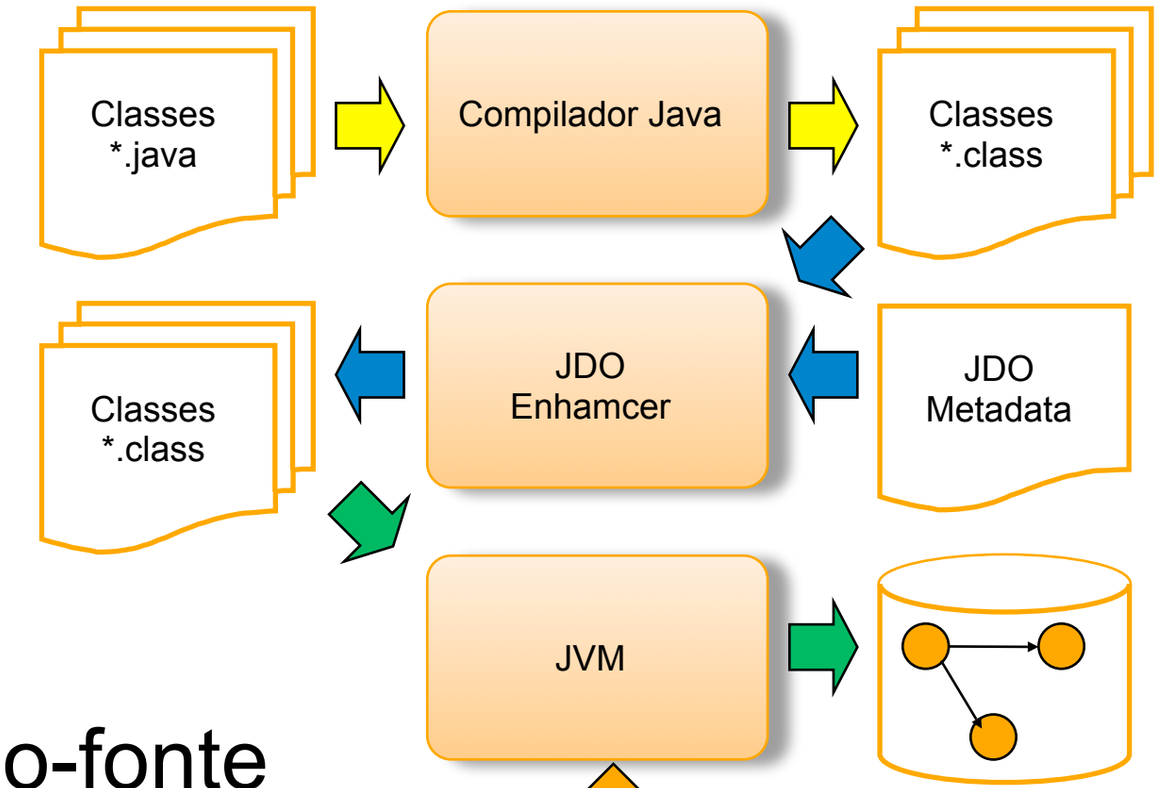
O Ciclo de Desenvolvimento JDO

- Qualquer classe que implemente **PersistenceCapable** pode ser persistida.

- Três maneiras de criar tais classes
  - Geração do código-fonte
  - Pré-processamento do código-fonte
  - *Bytecode enhancement*
    - O *Enhancement* é um passo de pós-compilação
    - Completamente transparente ao desenvolvedor

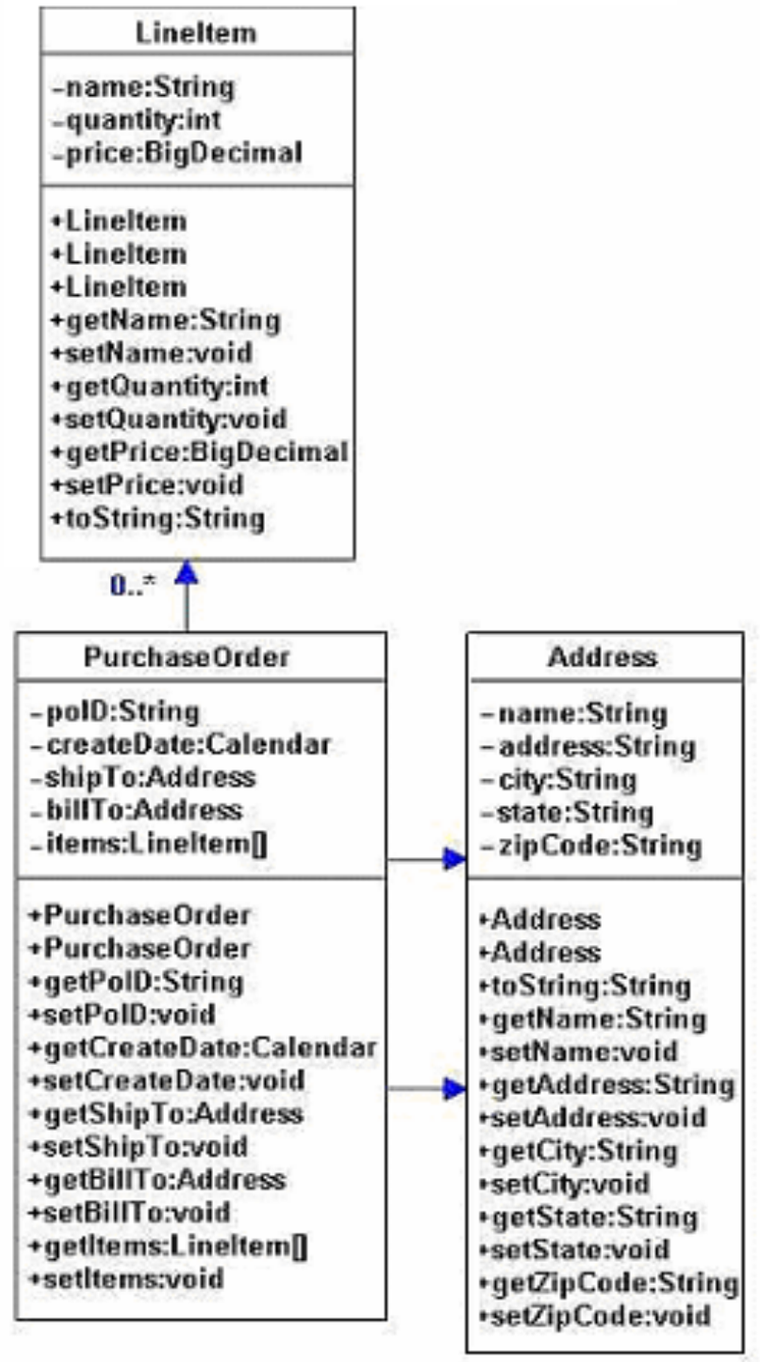
O Ciclo de Desenvolvimento JDO

- Qualquer classe que implemente **PersistenceCapable** pode ser persistida.



- Três maneiras de criar tais classes
  - Geração do código-fonte
  - Pré-processamento do código-fonte
  - *Bytecode enhancement*
    - O *Enhancement* é um passo de pós-compilação
    - Completamente transparente ao desenvolvedor

Um exemplo rápido

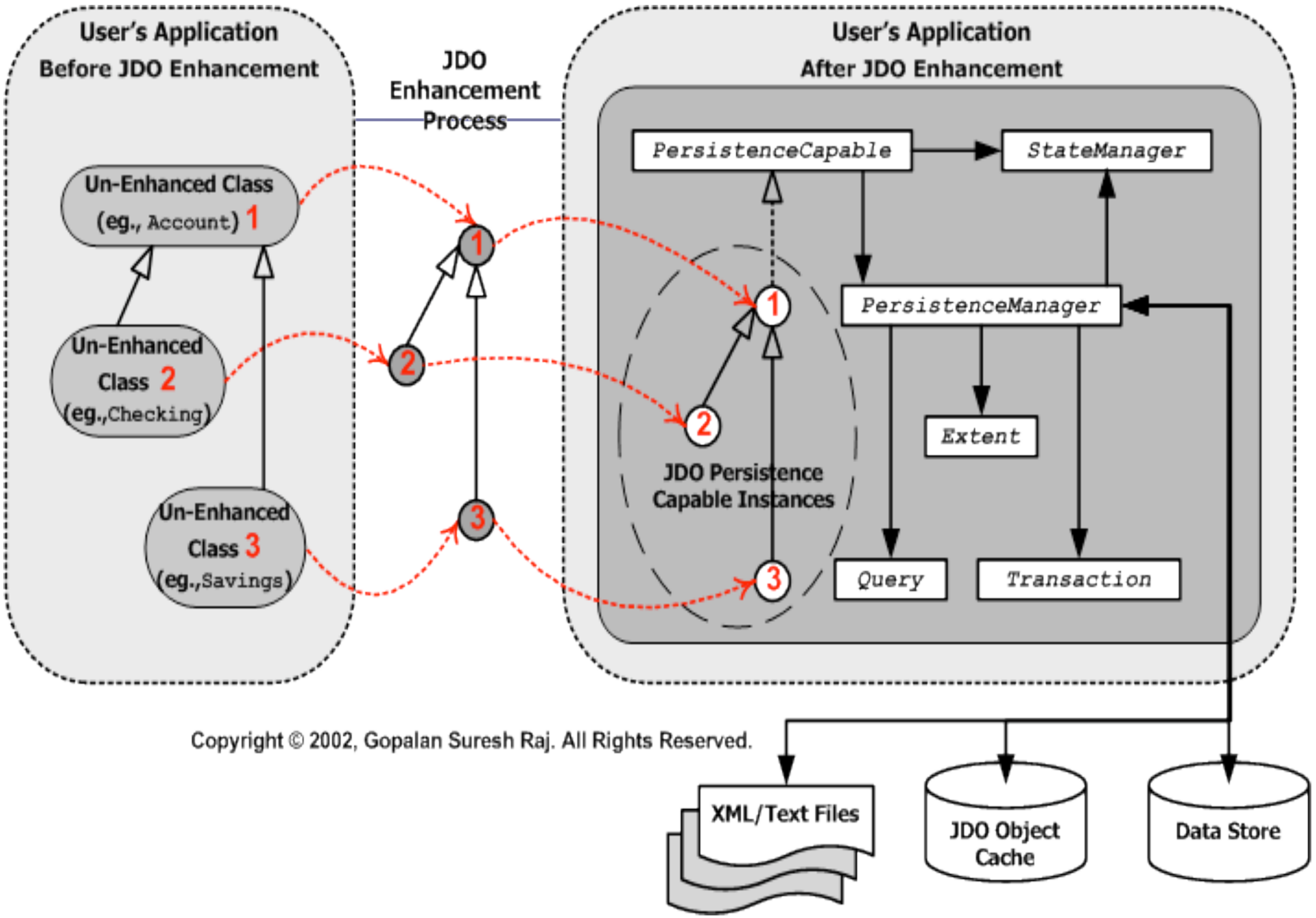



```
Properties p= new Properties();    // set some properties
PersistenceManagerFactory pmf= JDOHelper.getPersistenceManagerFactory(properties);
PersistenceManager pm =pmf.getPersistenceManager();
Transaction tx = pm.currentTransaction();
tx.begin();

Address addrs = new Address("1 Main Street“,  "Beverly Hills", "CA", "90210");

LineItem items[] = new LineItem[] {
   new LineItem("Copier Paper", new BigDecimal(10.00), 2)
};

PurchaseOrder order = new PurchaseOrder();

order.setCreateDate(Calendar.getInstance());
order.setBillTo(addrs);
order.setShipTo(addrs);
order.setItems(items);
order.setPoID("ABC-CO-19282");
pm.makePersistent(order);
tx.commit();
pm.close();
```

O que o JDO Enhancer faz?

- Lê o byte code e gera num novo byte code
  - Adiciona “ganchos” que permitem ao JDO transparentemente:
    - Recuperar de objetos
    - Rastrear mudanças nos estados de objetos
    - Escrever mudanças no repositório de dados quando confirmado
  - O Desenvolvedor não necessita fazer a busca e armazenamento explicito de objetos

Visão Geral


User's Application
Before JDO Enhancement
JDO
Enhancement
Process
User's Application
After JDO Enhancement
Un-Enhanced Class
(eg., Account) 1
Un-Enhanced
Class 2
(eg.,Checking)
Un-Enhanced
Class 3
(eg.,Savings)
PersistenceCapable
StateManager
PersistenceManager
Extent
JDO Persistence
Capable Instances
Query
Transaction
Copyright © 2002, Gopalan Suresh Raj. All Rights Reserved.
XML/Text Files
JDO Object
Cache
Data Store

Exemplos de implementação JDO

ObjectDB

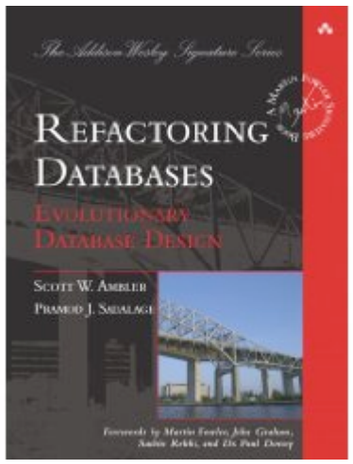
- O ObjectDB é um GBDOO que implementa o JDO.
- Recomendado para aprender o JDO, pois não existe a necessidade de pensar no mapeamento objeto-relacional.
- Site: http://www.objectdb.com
- Existe uma versão para Download free.
- Link para o tutorial.

JPOX

- É uma iniciativa de código aberto, agora responsável pela implementação de referência do JDO 2.0.
- JPOX permite realizar o mapeamento objeto-relacional compatível com a maioria dos SGBD Relacional via JDOC.
- Site: http://www.jpox.org
- Download da versão compatível com JDO 2.0.
- Plugin JPOX para Eclipse.
- Link para o tutorial.

Refatoração

- Refatoração de banco de dados



- **Refactoring Databases**
  (Evolutionary Database Design)
  - *Scott W. Ambler e Pramod J. Sadalage - 2006*

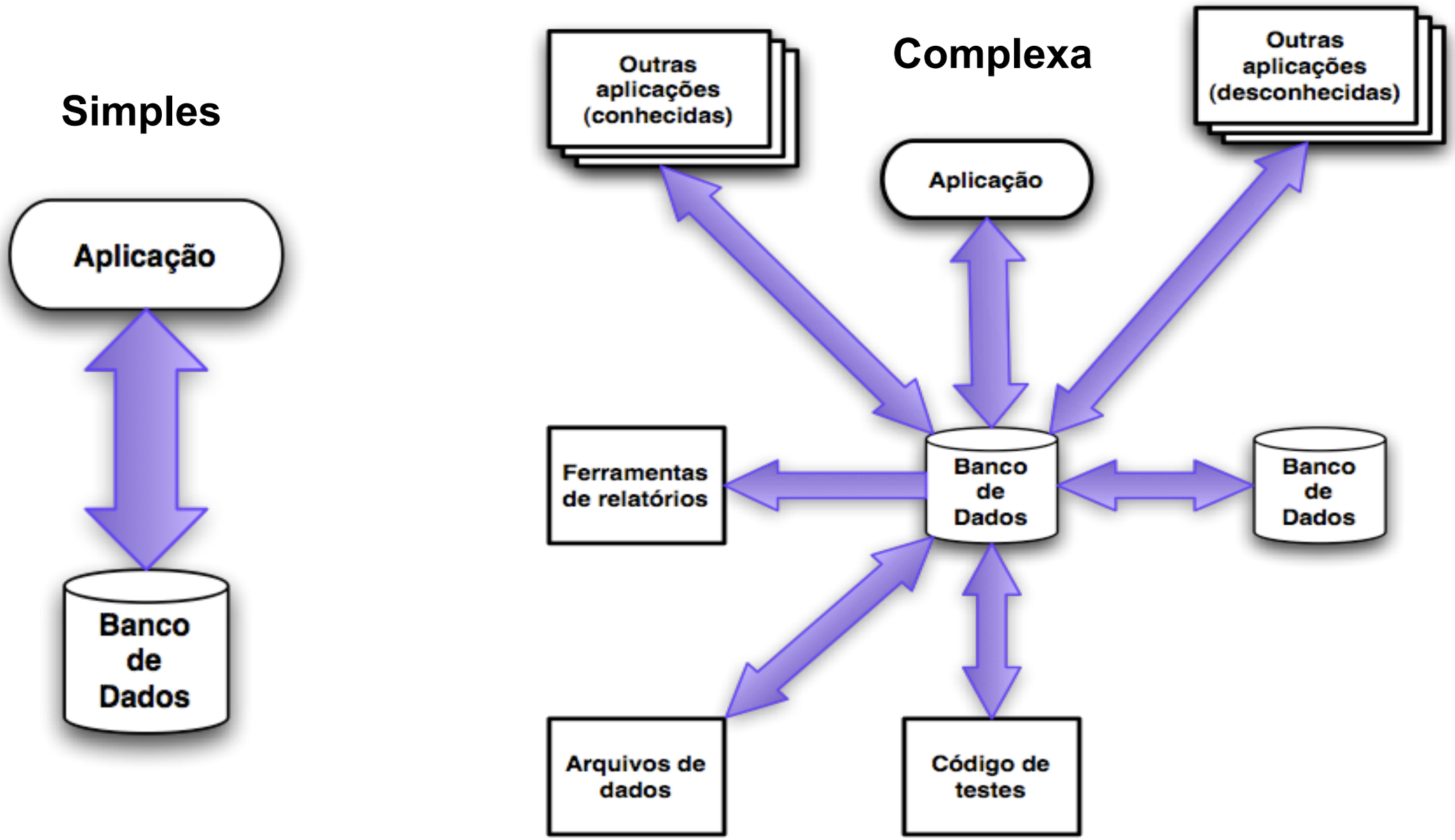
- **Recipes for Continuous Database Integration**
  (Evolutionary Database Development)
  - *Pramod J. Sadalage - 2007*

Refatoração de Bancos de Dados

- Banco de dados evolutivos:
  - Modelagem de dados evolutiva
  - Testes do banco de dados
  - Gerência de configuração (SVN, CVS, GIT etc)
  - **Refatoração do banco de dados**

Arquiteturas

- Duas categorias de arquitetura de banco de dados



Conceitos

- Refatoração de código
  - semântica comportamental
- Refatoração do banco de dados
  - semântica comportamental
  - **semântica informacional**

Conceitos

- Semântica informacional
  - Refere-se ao significado das informações do banco de dados a partir do ponto de vista dos usuários destas informações.
  - Preservar a semântica informacional implica que, se você alterar os valores dos dados armazenados em um coluna de uma tabela do banco de dados, os clientes desta informação não podem ser afetados por esta alteração.

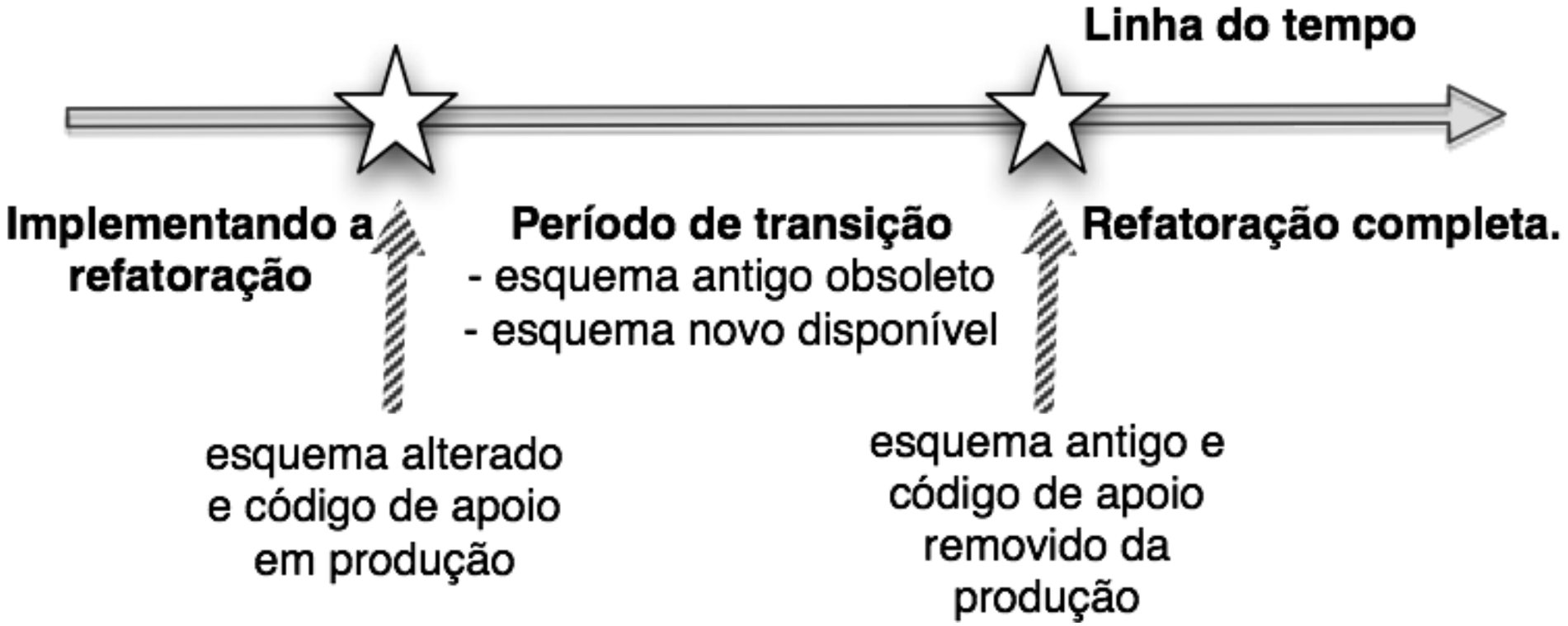
Conceitos

- Tipos de alterações no banco de dados:
  - Refatoração = pequenas
  - Transformações = médias
  - Migrações = grandes

Estratégias

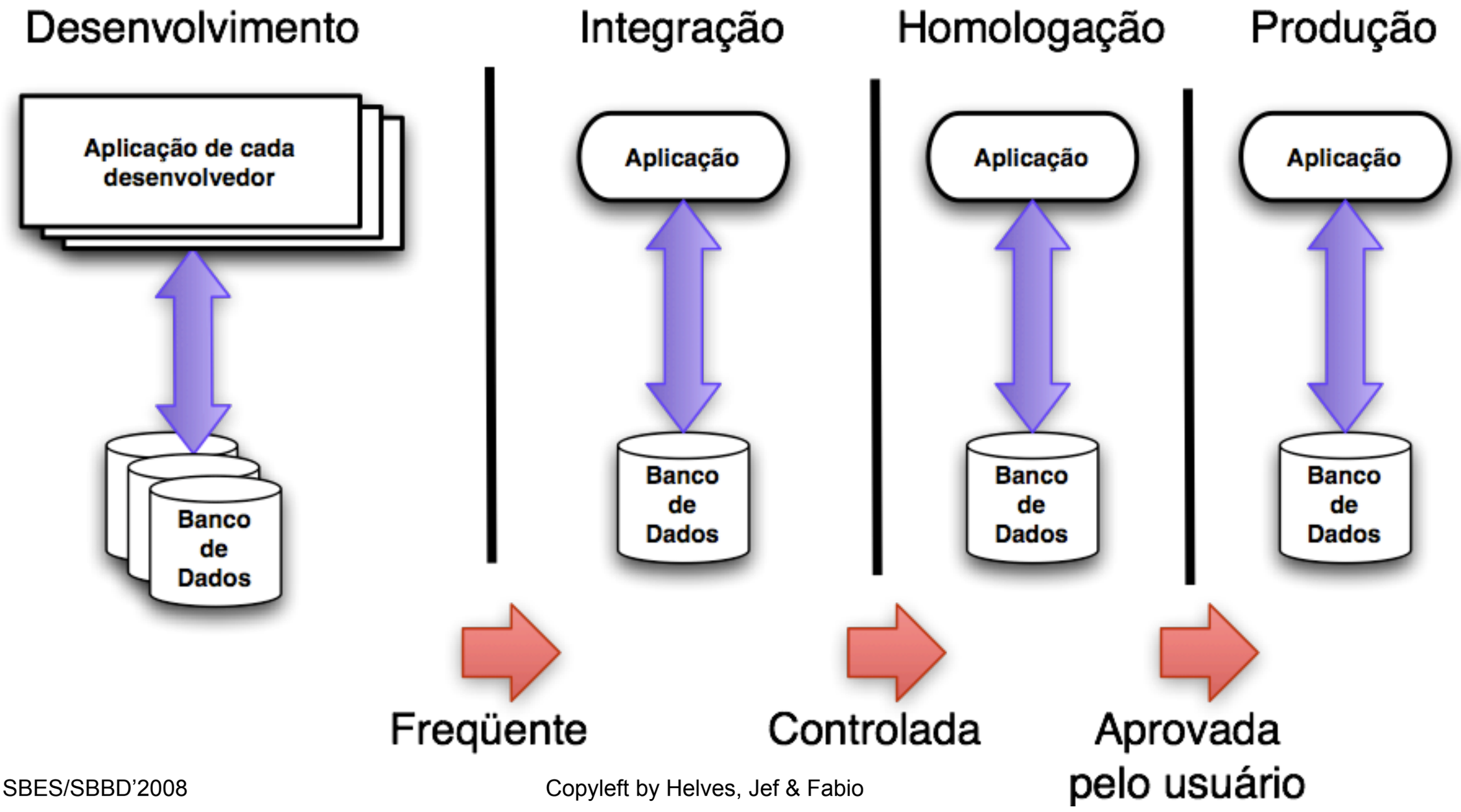
Estratégias

- Ciclo de vida de uma refatoração de banco de dados



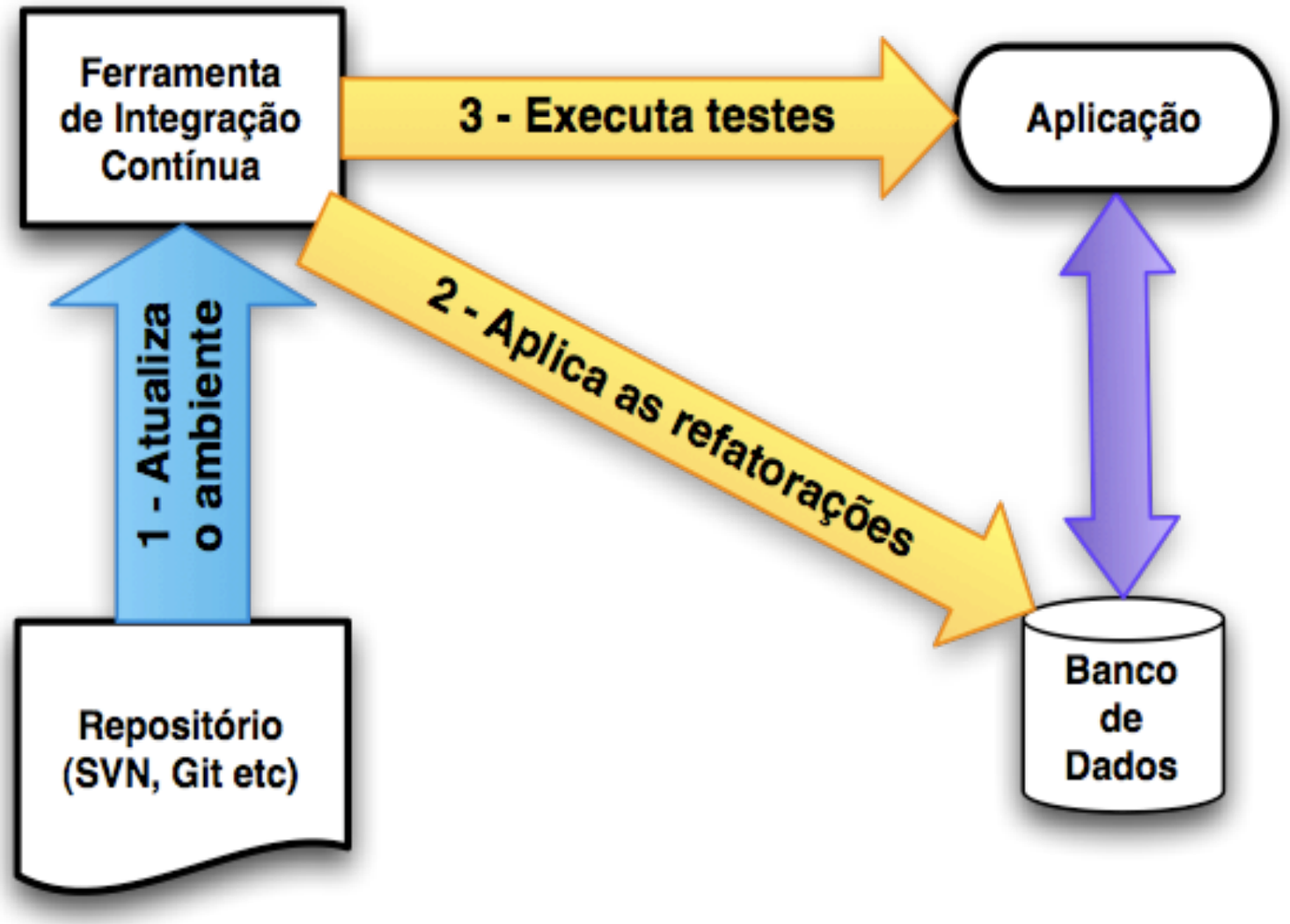
Estratégias

- Ambientes, bancos de dados e transições



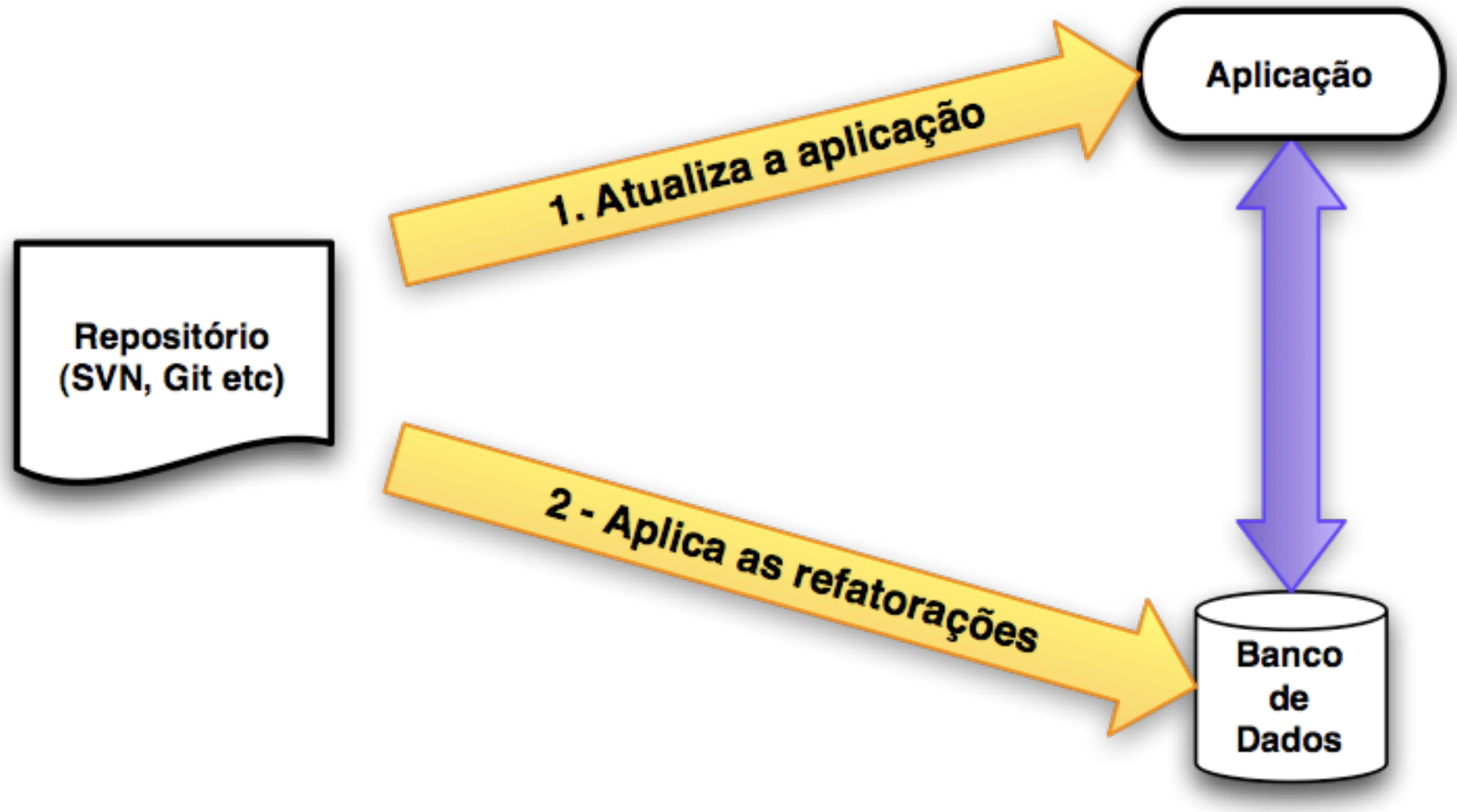
Estratégias

- Transição para o ambiente de Integração



Estratégias

- Transição para o ambiente de Homologação



Conclusão

- Modelagem de dados ágil e precisa
- Evolução das instâncias de banco de dados
- Alternativas viáveis para evolução de banco de dados.
- Quando refatorar ?
- O que fazer se não der para refatorar ?

Evolução e Refatoração de Banco de Dados

Helves Domingues (IME/USP)

Planejamento

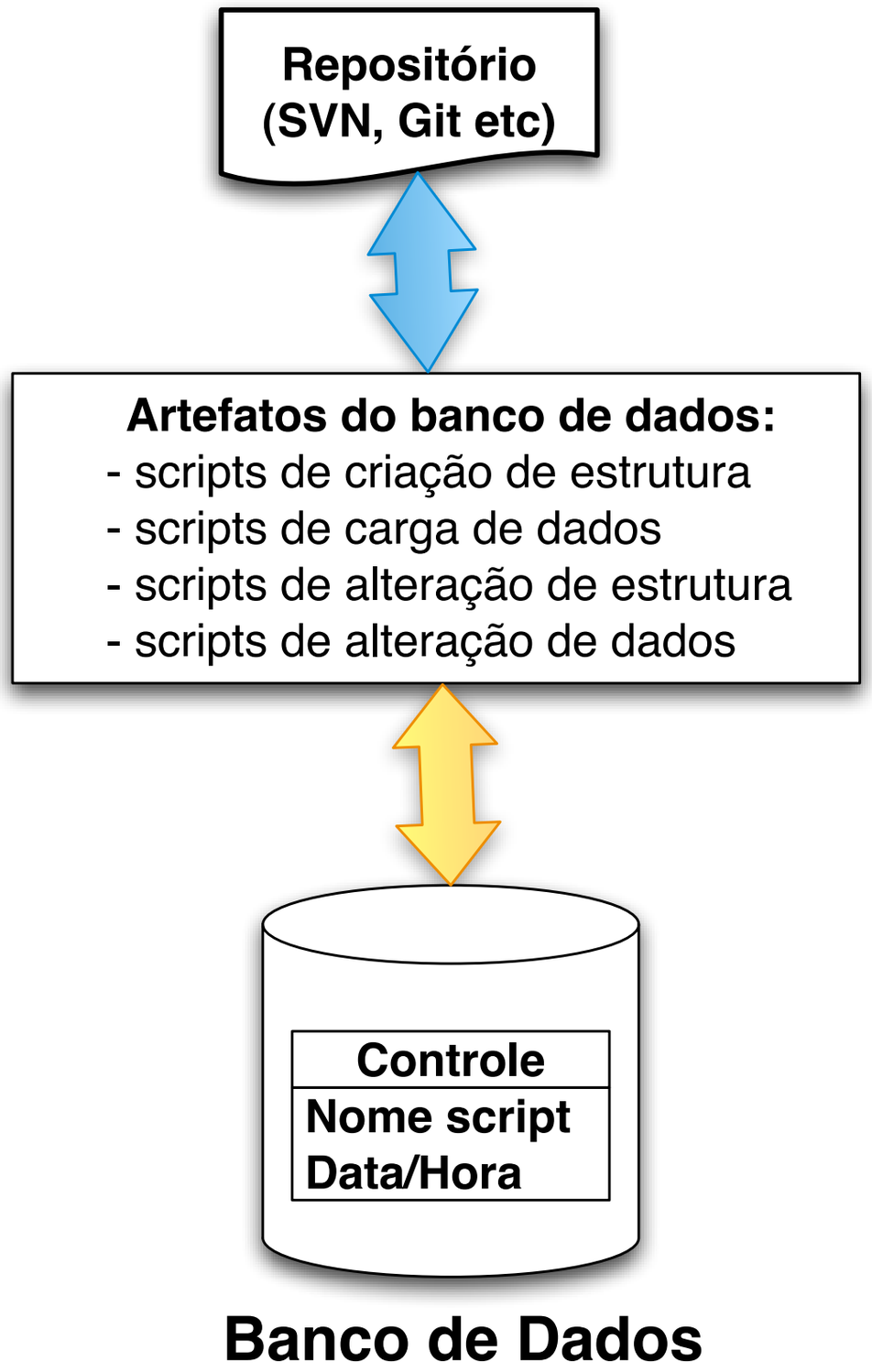
- Introdução
- Estratégias
- Melhores práticas
- Refatorações
- Refatoração Estrutural
- Refatoração de Qualidade dos Dados
- Refatoração de Integridade Referencial
- Refatoração Arquitetural
- Resumo e conclusão

Exemplos e casos de uso do Borboleta

- Projeto voltado para Assistência Domiciliar de Saúde com o uso de telefones celulares inteligentes, agora abrange também um sistema de prontuário eletrônico para Centros de Saúde.
- O Projeto Borboleta é financiado pelo Instituto Virtual FAPESP-Microsoft Research

Pré-requisitos

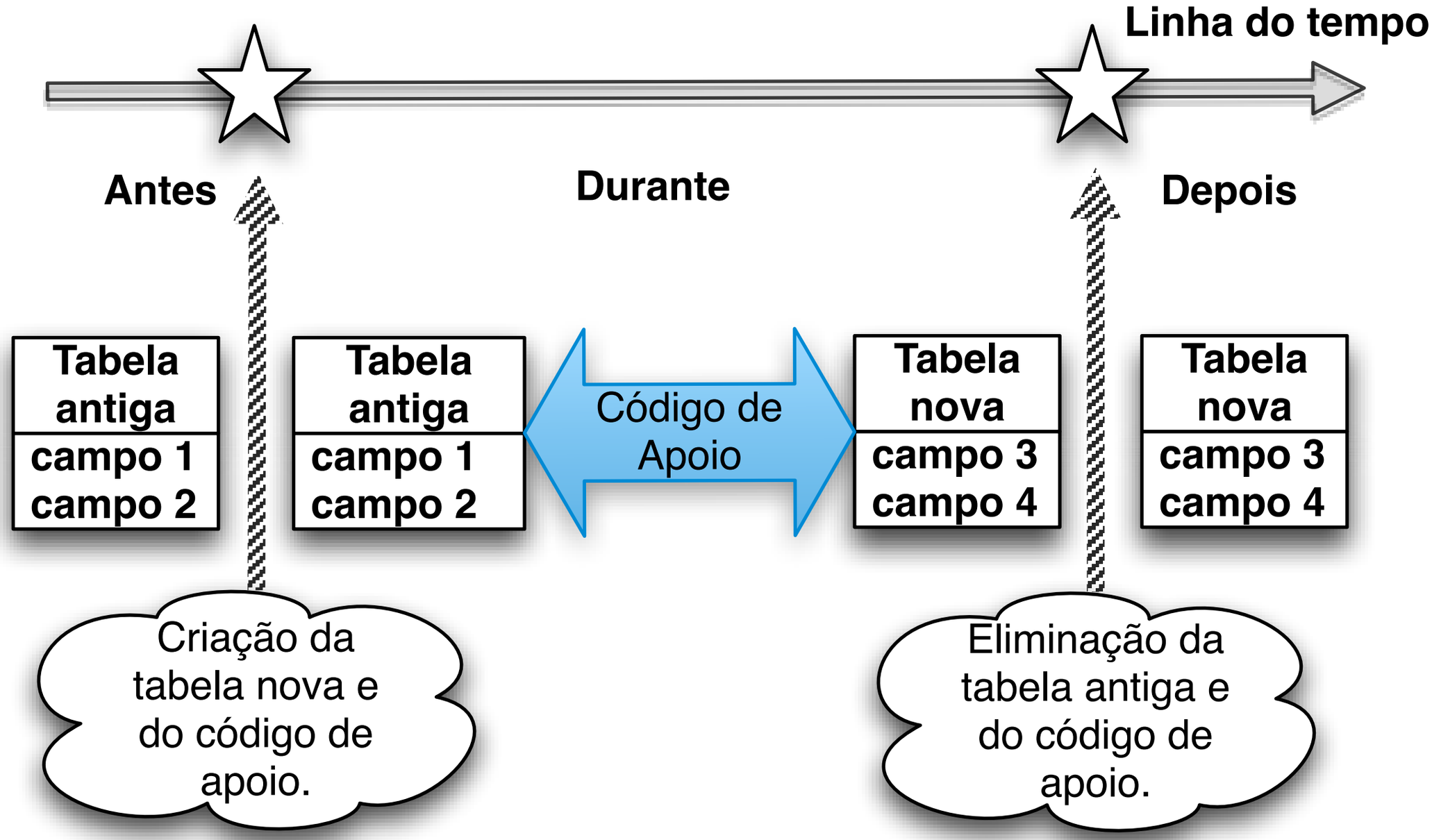
- Ambiente do Desenvolvedor:
  - Controlador de versões
  - *Scripts*
  - Banco de Dados individual



Estratégias

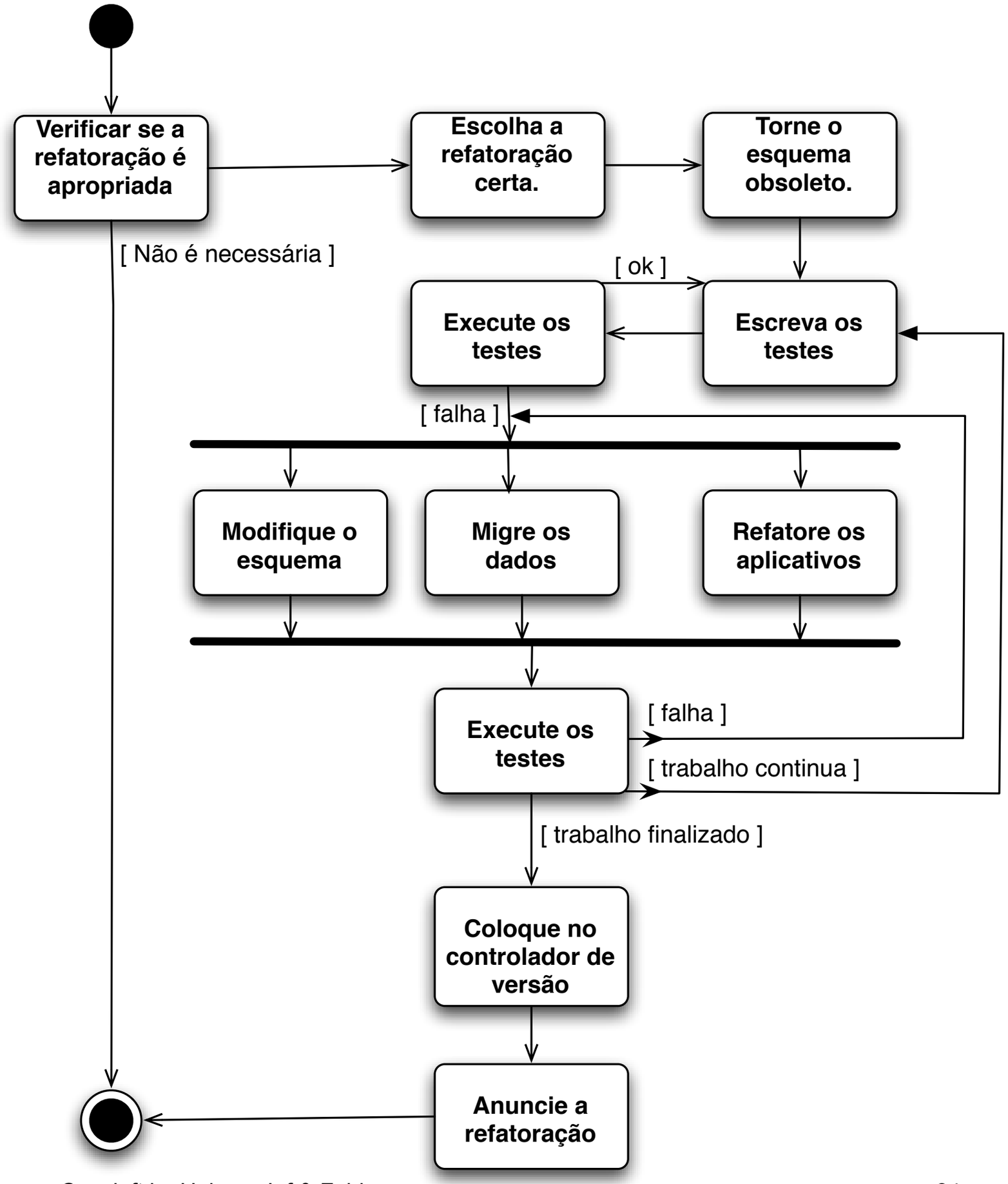
Estratégias

- Ciclo de vida de uma refatoração de banco de dados



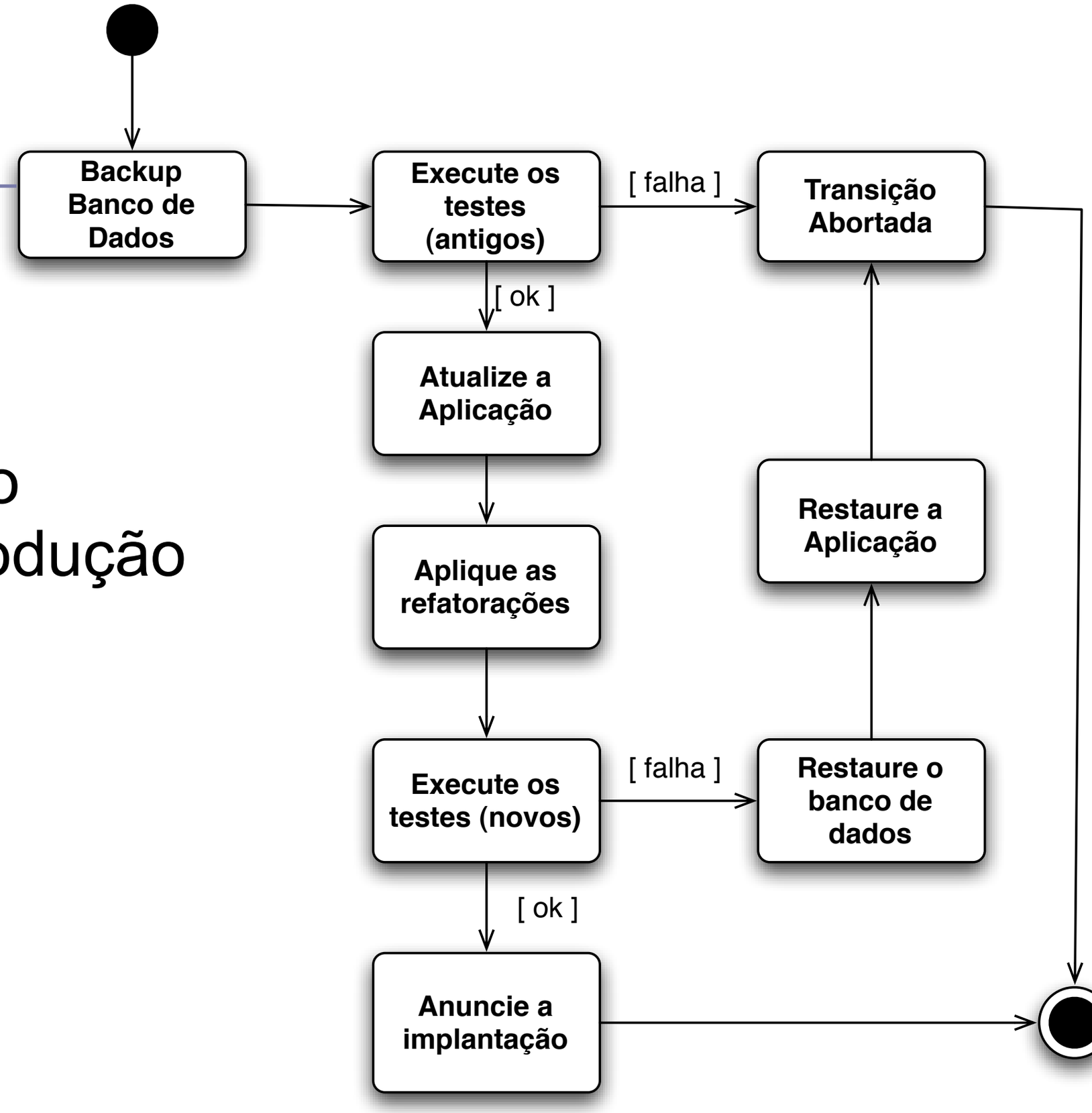
Estratégias

- Processo de refatoração
  - fluxo de trabalho



Estratégias

- Transição para o ambiente de Produção



Melhores Práticas

Melhores práticas

- Alterações pequenas são mais fáceis de aplicar
  - Implemente uma alteração grande através de várias alterações pequenas
- Identifique unicamente cada refatoração
- Simplifique o processo de negociação de alterações com outros times de desenvolvimento
- Coloque todos os códigos SQL da sua aplicação em um único lugar (não duplique código SQLs)

Melhores práticas

- Prefira *trigger* em relação a *views* e sincronização *batch*
  - *Trigger* pode trazer problemas: circularidade
- Estime e defina um período de transição correto
- Simplifique o processo de controle de alteração do banco de dados
- Use um software de integração contínua (*CruiseControl*) e alguma ferramenta para executar *scripts*: *ant*, *rake* etc

Refatorações

Refatorações

- Refatorações do livro de Scott Ambler

- Categorias de refatorações :
  - Estruturais = 17 refatorações
  - Qualidade dos dados = 13 refatorações
  - Integridade referencial = 7 refatorações
  - Arquitetura = 12 refatorações

- Outros capítulos
  - Refatoração de Métodos
  - Transformações

Refatoracões - Livro Ambler

Merge Columns

Merge two or more columns within a single table.

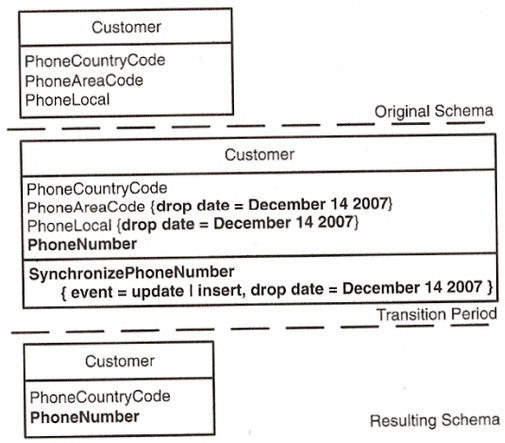


Figure 6.6 Merging columns in the Customer table.

Motivation

There are several reasons why you may want to apply *Merge Columns*:

- **An identical column.** Two or more developers may have added the columns unbeknownst to each other, a common occurrence when the developers are on different teams or when meta data describing the table schema is not available. For example, the *FeeStructure* table has 37 columns, 2 of which are called *CA_INIT* and *CheckingAccountOpeningFee*, and both of which store the initial fee levied by the bank when opening a checking account. The second column was added because nobody was sure what the *CA_INIT* column was really being used for.
- **The columns are the result of overdesign.** The original columns where introduced to ensure that the information was stored in its constituent forms, but actual usage shows that you do not need the fine details that you originally thought. For example, *Customer* table of Figure 6.6 includes

the columns *PhoneCountryCode*, *PhoneAreaCode*, and *PhoneLocal* to represent a single phone number.

- **The actual usage of the columns has become the same.** Several columns were originally added to a table, but over time the way that one or more of them are used has changed to the point where they are all being used for the same purpose. For example, the *Customer* table includes *PreferredCheckStyle* and *SelectedCheckStyle* columns (not shown in Figure 6.6). The first column was used to record which style of checks to send to the customer from next season's selection, and the second column was used to record the style which the customer previously had sent out to them. This was useful 20 years ago when it took several months to order in new checks, but now that they can be printed over night, we have started automatically storing the same value in both columns.

Potential Tradeoffs

This database refactoring can result in a loss of data precision when you merge finely detailed columns. When you merge columns that (you believe) are used for the same purpose, you run the risk that you should in fact be using them for separate things. (If so, you will discover that you need to reintroduce one or more of the original columns.) The usage of the data should determine whether the columns should be merged, something that you will need to explore with your stakeholders.

Schema Update Mechanics

To perform *Merge Columns*, you must do two things. First, you need to introduce the new column. Add the column to the table via the SQL command ADD COLUMN. In Figure 6.6, this is *Customer.PhoneNumber*. This step is optional because you may find it possible to use one of the existing columns into which to merge the data. You also need to introduce a synchronization trigger to ensure that the columns remain synchronized with one another. The trigger must be invoked by any change to the columns.

Figure 6.6 shows an example where the *Customer* table initially stores the phone number of a person in three separate columns: *PhoneCountryCode*, *PhoneAreaCode*, and *PhoneLocal*. Over time, we have discovered that few applications are interested in the country code because they are used only within North America. We have also discovered that every application uses both the

Refatorações - Livro Ambler

area code and the local phone number together. Therefore, we have decided to leave the *PhoneCountryCode* alone but to merge the *PhoneAreaCode* and *PhoneLocal* columns into *PhoneNumber,* reflecting the actual usage of the data by the application (because the application does not use *PhoneAreaCode* or *PhoneLocal* individually). We introduced the *SynchronizePhoneNumber* trigger to keep the values in the four columns synchronized.

The following SQL code depicts the DDL to introduce the *PhoneNumber* column and to eventually drop the two original columns:

```
ALTER TABLE Customer ADD PhoneNumber NUMBER(12);
        COMMENT ON Customer.PhoneNumber 'Added as the
        result of merging Customer.PhoneAreaCode and
        Customer.PhoneLocal finaldate = December 14 2007';

-On December 14 2007
ALTER TABLE Customer DROP COLUMN PhoneAreaCode;
ALTER TABLE Customer DROP COLUMN PhoneLocal;
```

Merge Columns

Data-Migration Mechanics

You must convert all the data from the original column(s) into the merge column, in this case from *Customer.PhoneAreaCode* and *Customer.PhoneLocal* into *Customer.PhoneNumber*. The following SQL code depicts the DML to initially combine the data from *PhoneAreaCode* and *PhoneLocal* into *PhoneNumber*.

```
/*One-time migration of data from Customer.PhoneAreaCode and Customer.PhoneLocal to
Customer.PhoneNumber. When both the columns are active, there is a need to have a trigger that
keeps both the columns in sync */
UPDATE Customer SET PhoneNumber = PhoneAreaCode*10000000 + PhoneLocal);
```

Access Program Update Mechanics

You need to analyze the access programs thoroughly, and then update them appropriately, during the transition period. In addition to the obvious, you need to work with *Customer.PhoneNumber* rather than the former unmerged columns. Potentially, you must remove merging code. There may be code that combines the existing columns into a data attribute similar to the merged column. This code should be refactored and potentially removed entirely.

Second, you may also need to update data-validation code to work with merged data. Some data-validation code may exist solely because the columns

have not yet been merged. For example, if a value is stored in two separate columns, you may have validation code in place that verifies that the values are the same. After the columns are merged, there may no longer be a need for this code.

The before and after code snippet shows how the *getCustomerPhoneNumber()* method changes when we merge the *Customer.PhoneAreaCode* and *Customer.PhoneLocal* columns:

```
//Before code
public String getCustomerPhoneNumber(Customer customer){
  String phoneNumber = customer.getCountryCode();
  phoneNumber.concat(phoneNumberDelimiter());
  phoneNumber.concat(customer.getPhoneAreaCode());
  phoneNumber.concat(customer.getPhoneLocal());
  return phoneNumber;
}

//After code
public String getCustomerPhoneNumber(Customer customer){
  String phoneNumber = customer.getCountryCode();
  phoneNumber.concat(phoneNumberDelimiter());
  phoneNumber.concat(customer.getPhoneNumber());
  return phoneNumber;
}
```

Merge Columns

Refatorações Estruturais

- Drop Column
- Drop Table
- Drop View
- Introduce Calculated Column
- Introduce Surrogate Key
- Merge Columns
- Merge Tables
- Move Column
- Rename Column
- Rename Table
- Rename View
- Replace LOB With Table
- Replace Column
- Replace One-To-Many With Associative Table
- Replace Surrogate Key With Natural Key
- Split Column
- Split Table

Refatorações de Qualidade dos Dados

- Add Lookup Table
- Apply Standard Code
- Apply Standard Type
- Consolidate Key Strategy
- Drop Column Constraint
- Drop Default Value
- Drop Non-Nullable
- Introduce Column Constraint
- Introduce Common Format
- Introduce Default Value
- Make Column Non-Nullable
- Move Data
- Replace Type Code With Property Flags

Refatorações de Integridade Referencial

- Add Foreign Key Constraint
- Add Trigger For Calculated Column
- Drop Foreign Key Constraint
- Introduce Cascading Delete
- Introduce Hard Delete
- Introduce Soft Delete
- Introduce Trigger For History

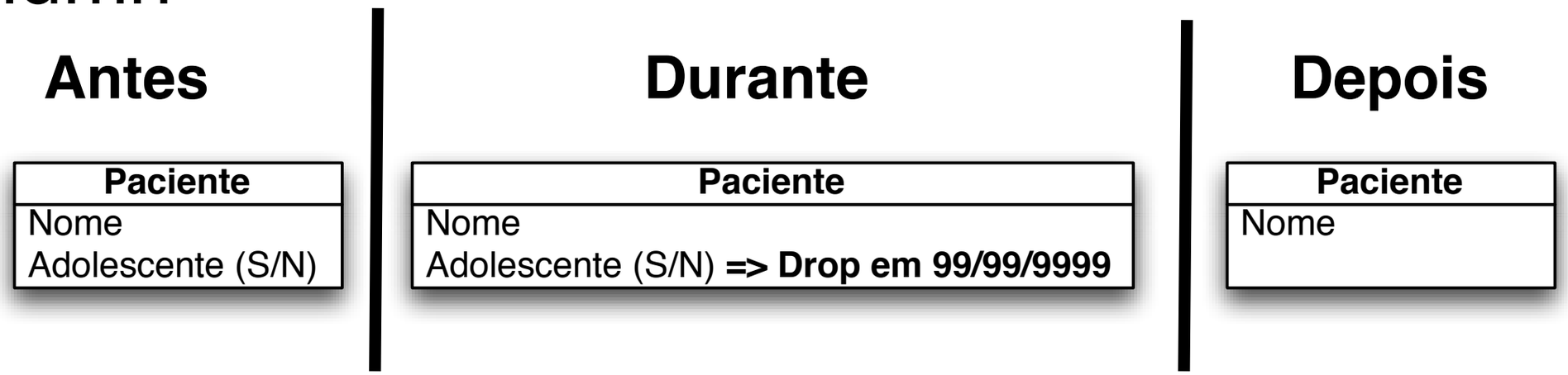
Refatorações Arquiteturais

- Add CRUD Methods
- Add Mirror Table
- Add Read Method
- Encapsulate Table With View
- Introduce Calculation Method
- Introduce Index
- Introduce Read-Only Table
- Migrate Method From Database
- Migrate Method to Database
- Replace Method(s) With View
- Replace View With Method(s)
- Use Official Data Source

Refatorações Estruturais

Refatoração estrutural 1/17

- Drop Column



Eliminar coluna

Motivação	Coluna não usada e/ou não deve ser usada. Um dos passos para a refatoração *Move Columns*.
Avaliação	Contém dados que podem ter valor. Tabelas com muitas linhas, a remoção pode ser lenta.
Esquema	Período de transição: alertar que a coluna será eliminada. Finalização: remover esta coluna de: chaves estrangeiras, visões e índices. Por último: remover a coluna.
Dados	Preservar os dados desta coluna, caso necessário, em outro esquema.
Aplicativo	Remover todas as referências a esta coluna.

Refatoração estrutural 2/17

- Drop Table

Antes

Médico
Nome
Data Nascimento
CRM

Durante

Etapa 1 - Comentário

Médico = Drop em 99/99/9999 =
Nome
Data Nascimento
CRM

Etapa 2 - Renomear a tabela

Médico_temporário
Nome
Data Nascimento
CRM

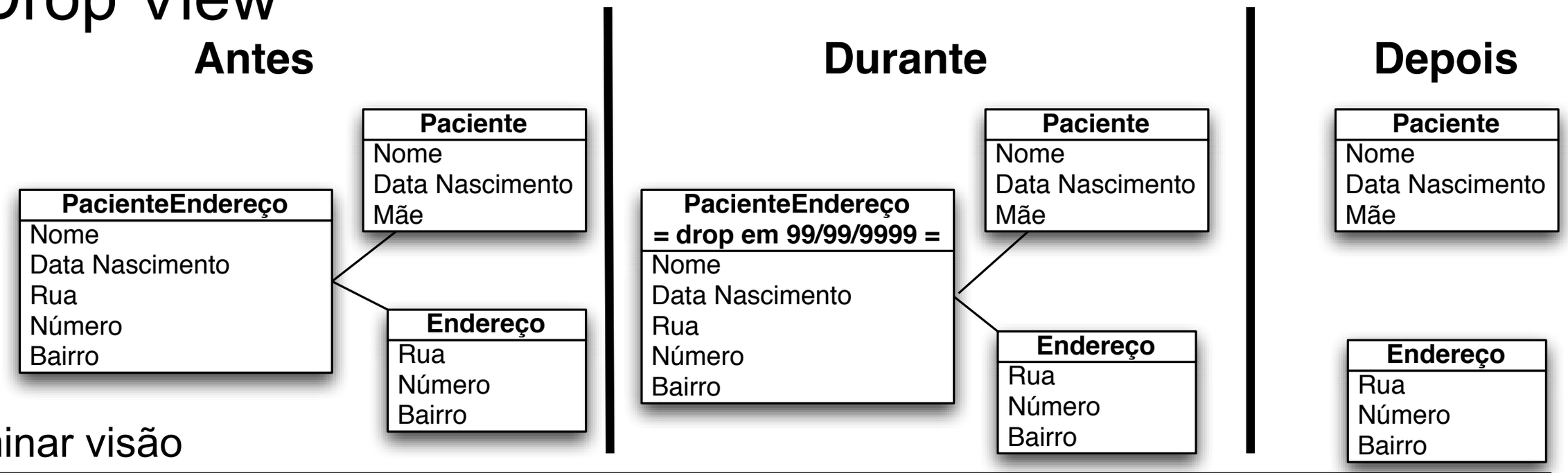
Depois

Eliminar tabela

Motivação	A tabela não é necessária e/ou não deve ser usada.
Avaliação	A tabela contém dados que podem ser importantes no futuro.
Esquema	Período de transição: alertar que a tabela será removida e/ou renomear a tabela. Finalização: remover referências à tabela em chave estrangeira (*fk constraint*) e visões. Por último, remover a tabela.
Dados	Preservar os dados desta tabela, caso necessário, em outro esquema.
Aplicativo	Remover referências no código: *querys*, *joins* e *view* que usam a tabela.

Refatoração estrutural 3/17

- Drop View

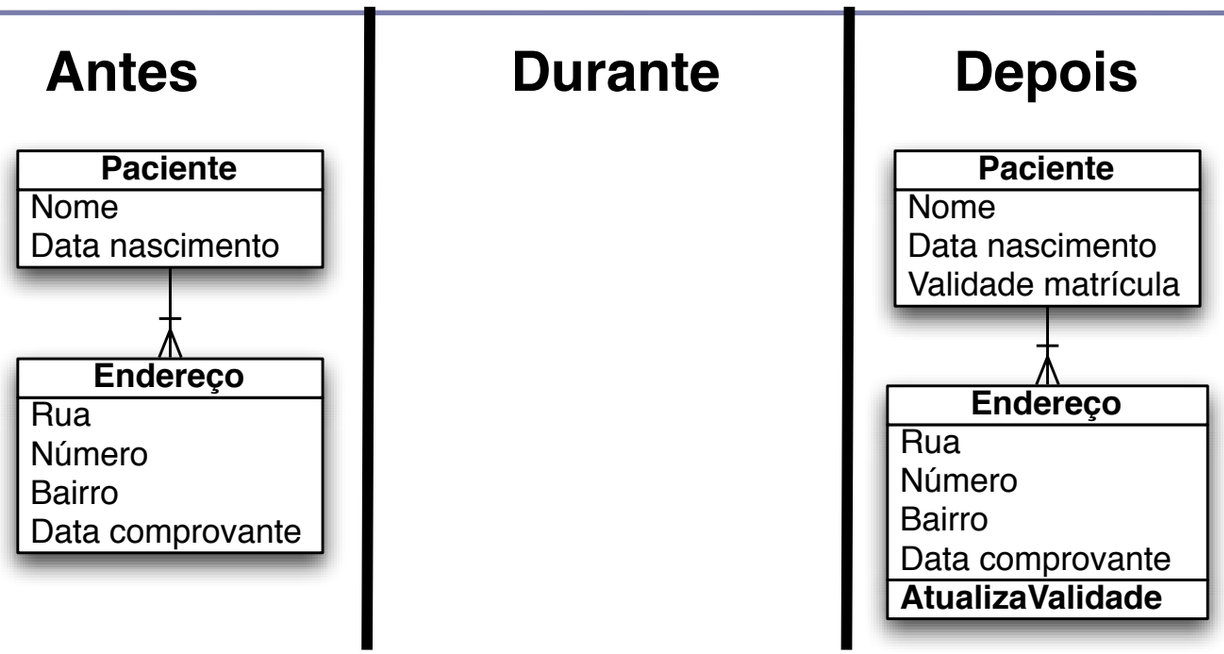


Eliminar visão

Motivação	A visão não é necessária e/ou não deve ser usada.
Avaliação	Verificar se tem algum relatório que usa a visão. Verificar se tem algum controle de acesso feito usando esta visão.
Esquema	Período de transição: alertar que a visão será removida e/ou renomear a visão. Finalização: remover referências a visão (outras visões que usam esta) e remover a visão.
Dados	Não se aplica.
Aplicativo	Remover referências no código.

Refatoração estrutural 4/17

- Introduce Calculated Column

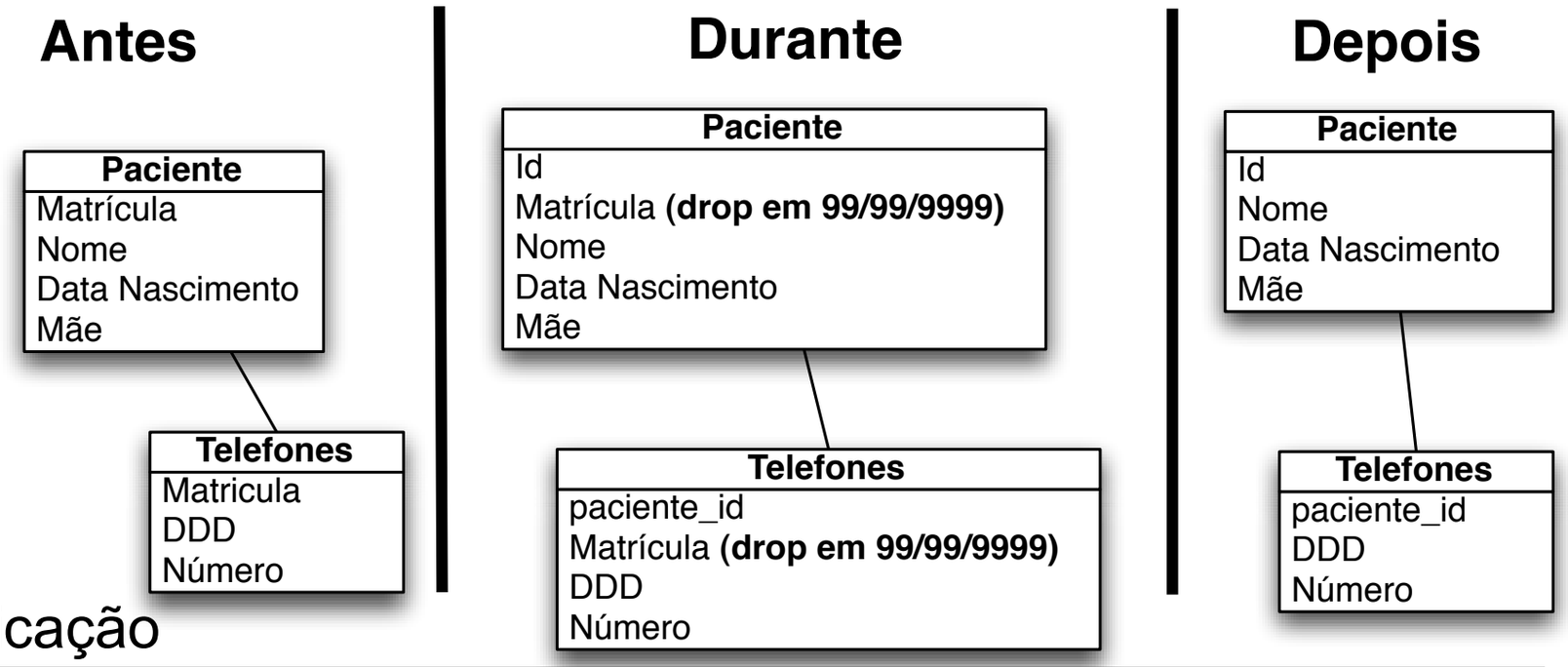


Introduzir coluna calculada

Motivação	Melhorar o desempenho. Unificar cálculo entre aplicativos diferentes.
Avaliação	O valor calculado pode ficar desatualizado por um período de tempo. Qual método a ser utilizado para atualizar o valor calculado: *batch* ou *trigger*.
Esquema	Definir as regras para calcular a nova coluna. Incluir a nova coluna na tabela. Implementar o método de atualização da coluna (caso *trigger*).
Dados	Fazer o cálculo inicial para todas as linhas já existentes.
Aplicativo	Retirar do aplicativo o código que faz o cálculo. Implementar o método de atualização da coluna (caso *batch*).

Refatoração estrutural 5/17

- Introduce Surrogate Key



Introduzir chave de identificação

Motivação	Diminuir acoplamentos, aumentar consistência, melhorar o desempenho do banco de dados.
Avaliação	Muitos preferem chaves naturais a *surrogate keys*. É muito útil ter uma estratégia única para chaves.
Esquema	Período de transição: incluir a nova coluna, adicionar um novo índice para esta coluna, alertar que a chave original não será mais utilizada e escrever *trigger* para manter a chave antiga. Finalização: Remover a coluna antiga e remover o *trigger*.
Dados	Gerar os dados para a nova chave. Acertar as referências à chave antiga.
Aplicativo	Trocar as referências às chaves antigas pela nova.

Refatoração estrutural 6/17

- Merge Columns

Antes

Telefones
paciente_id
DDD
Número

Durante

Telefones
paciente_id
DDD(**drop em 99/99/9999)**
Número(**drop em 99/99/9999)**
NúmeroCompleto

Depois

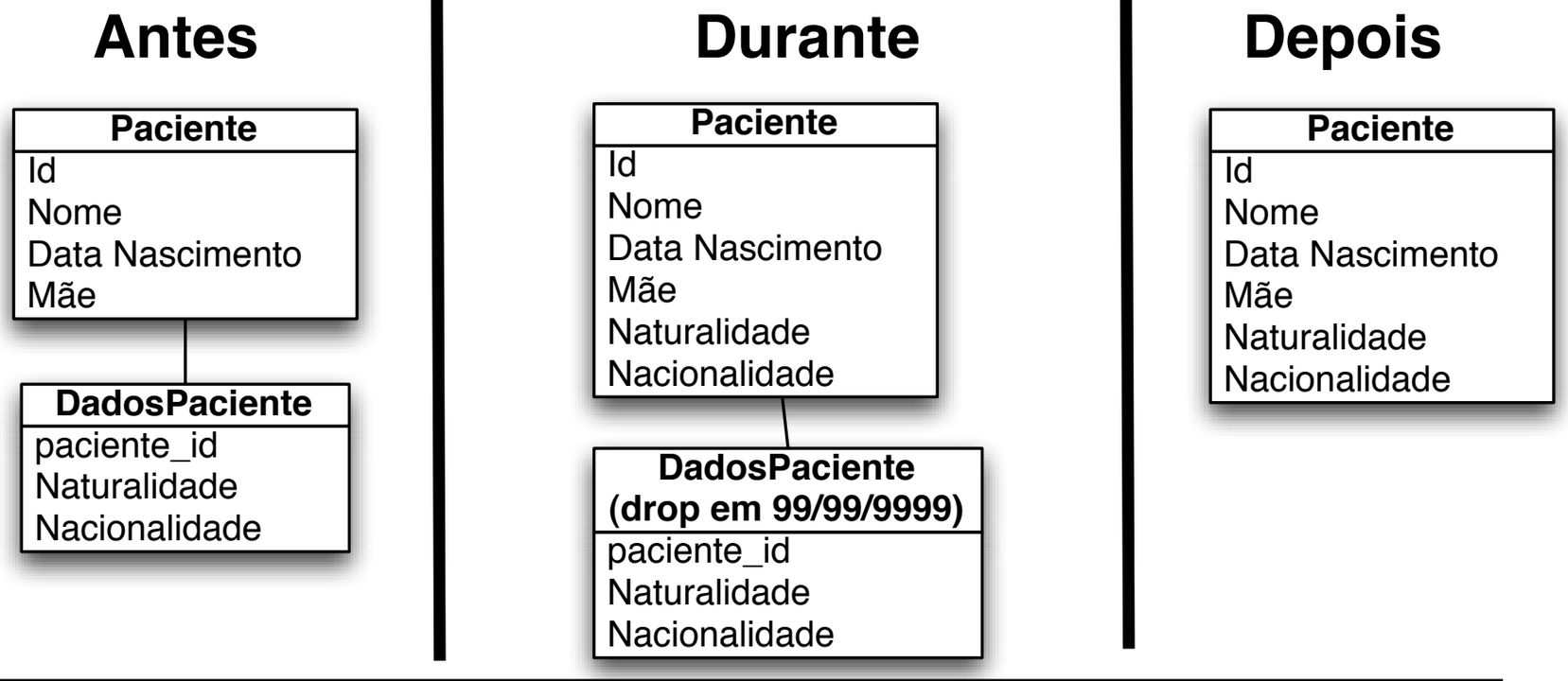
Telefones
paciente_id
NúmeroCompleto

Unir colunas

Motivação	Colunas idênticas. Separação desnecessária. O uso atual das duas colunas é o mesmo.
Avaliação	Verificar se é necessário preservar os dados para um futuro uso.
Esquema	Período de transição: incluir uma nova coluna, escrever um *trigger* que mantém esta nova coluna atualizada, sinalizar que as colunas antigas não serão utilizadas. Finalização: Remover as colunas não utilizadas e remover o *trigger*.
Dados	Gerar os dados da nova coluna. Guardar os dados das colunas quer serão removidas.
Aplicativo	Trocar as referências às colunas antigas pela nova.

Refatoração estrutural 7/17

- Merge Tables

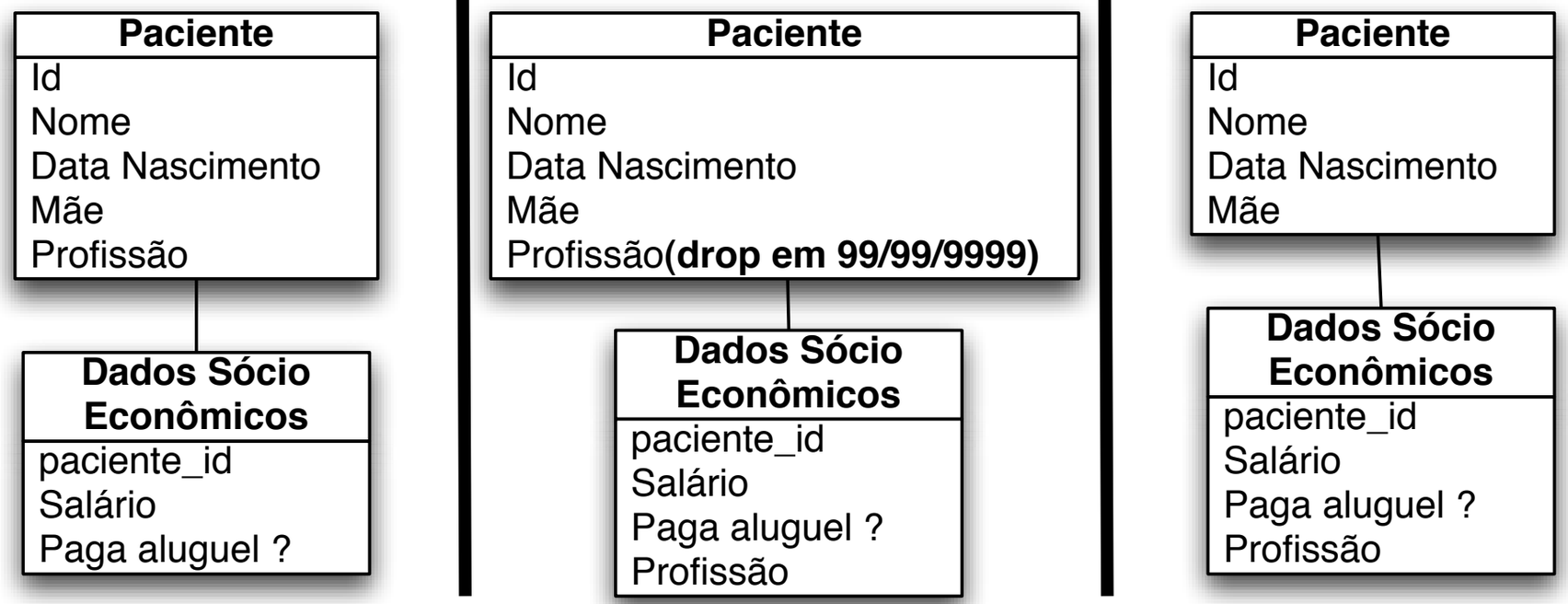


Unir tabelas

Motivação	Erro de projeto. As duas tabelas tem o mesmo uso. Desenvolvedores criaram tabelas semelhantes em projetos diferentes.
Avaliação	Avaliar a necessidade de guardar os dados. Definir qual tabela terá todos as colunas.
Esquema	Período de transição: criar as colunas na tabela final, criar os *trigger* para manter os dados sincronizados, alertar qual tabela será removida. Finalização: Remover os *triggers* e remover a tabela desnecessária.
Dados	Carregar os dados das novas colunas da tabela. Guardar a tabela que será removida.
Aplicativo	Remover todas as referências à tabela que será removida.

Refatoração estrutural 8/17

- Move Column



Mover coluna

Motivação	Normalização, desnormalização e reorganização de uma tabela.
Avaliação	Avaliar o desempenho e a consistência das informações na tabela destino.
Esquema	Período de transição: escrever os *triggers* para manter os valores na tabela antiga, criar a nova coluna na tabela destino, alertar que a coluna será removida. Finalização: remover os *triggers* e remover a coluna na tabela antiga.
Dados	A tabela que irá receber a coluna deverá ser atualizada com valores pré-existentes da coluna.
Aplicativo	Trocar as referências a esta coluna para a nova tabela.

Refatoração estrutural 9/17

- Rename Column

Antes

Paciente
Id
Nome
Data Nascimento
Mãe

Durante

Paciente
Id
Nome
Data Nascimento
Mãe **(drop em 99/99/9999)**
Nome da Mãe

Depois

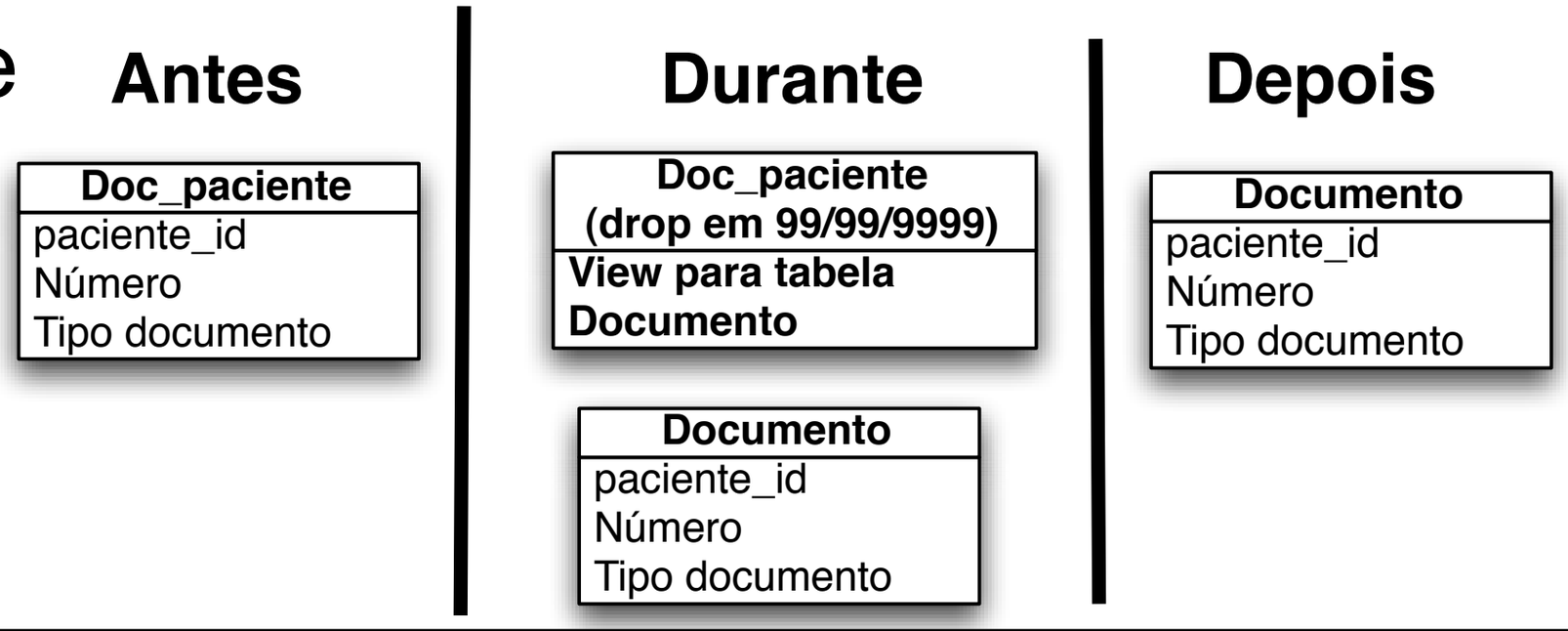
Paciente
Id
Nome
Data Nascimento
Nome da Mãe

Renomear coluna

Motivação	Melhorar o entendimento das tabelas, seguir uma convenção de nomes ou trocar de banco de dados onde o nome da coluna é palavra reservada.
Avaliação	Verificar se os benefícios compensam o trabalho a ser feito, evitando preferências de nomes com o mesmo sentido.
Esquema	Período de transição: incluir a nova coluna com o nome escolhido, escrever o *trigger* para manter o valor atualizado e alertar que a coluna antiga será eliminada. Finalização: remover a coluna antiga e remover o *trigger* criado.
Dados	Carregar os valores da nova coluna.
Aplicativo	Trocar o nome da coluna para o novo nome.

Refatoração estrutural 10/17

- Rename Table

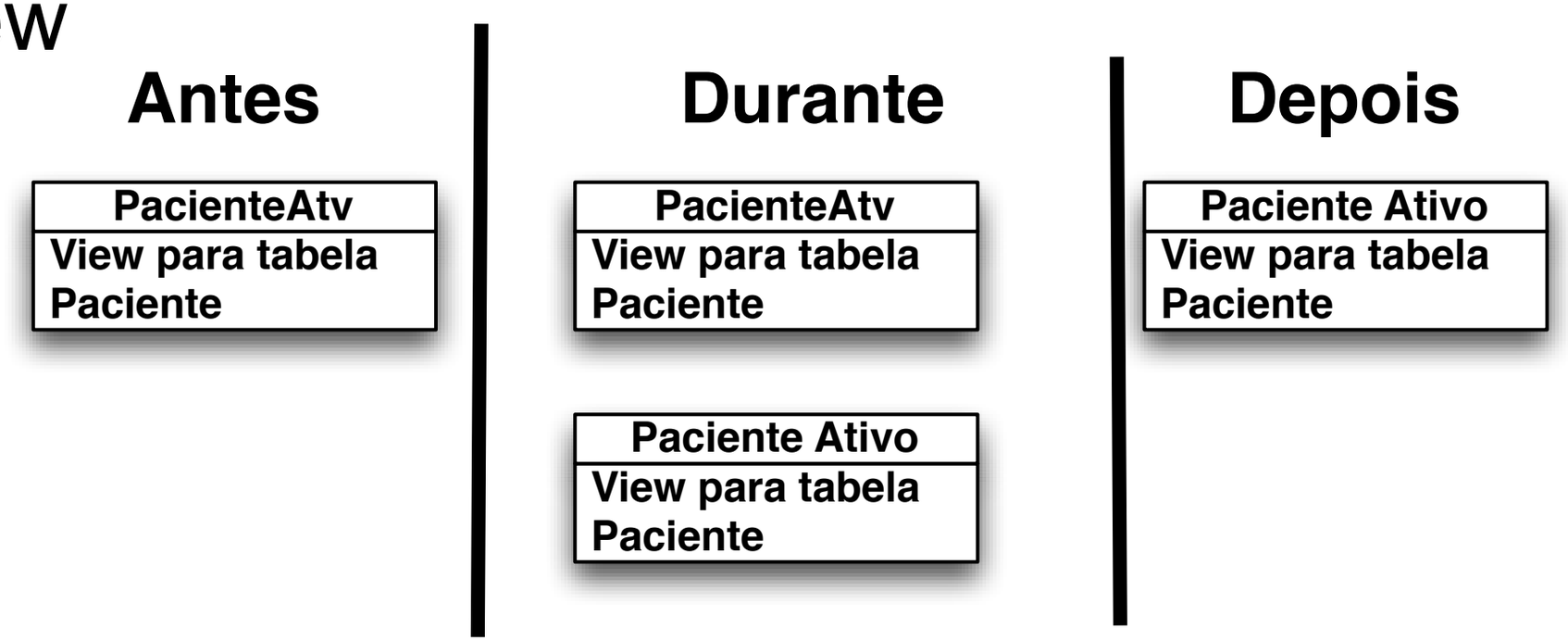


Renomear tabela

Motivação	Melhorar o entendimento das tabelas ou seguir uma convenção de nomes.
Avaliação	Avaliar a possibilidade de usar uma visão atualizável com o nome antigo no período de transição. Se não for possível, deve-se criar uma tabela nova com o novo nome.
Esquema	(Caso visão) Trocar o nome da tabela e criar uma visão com o nome antigo. (Caso nova tabela) Período de transição: criar uma tabela com o nome novo, alertar que a tabela antiga será renomeada, criar *triggers* para manter as tabelas sincronizadas. Finalização: remover os *triggers* e remover a tabela com o nome antigo.
Dados	Caso nova tabela, sincronizar os dados com a tabela com o nome antigo.
Aplicativo	Trocar as referências à tabela do nome antigo para o novo nome.

Refatoração estrutural 11/17

- Rename View



Renomear visão

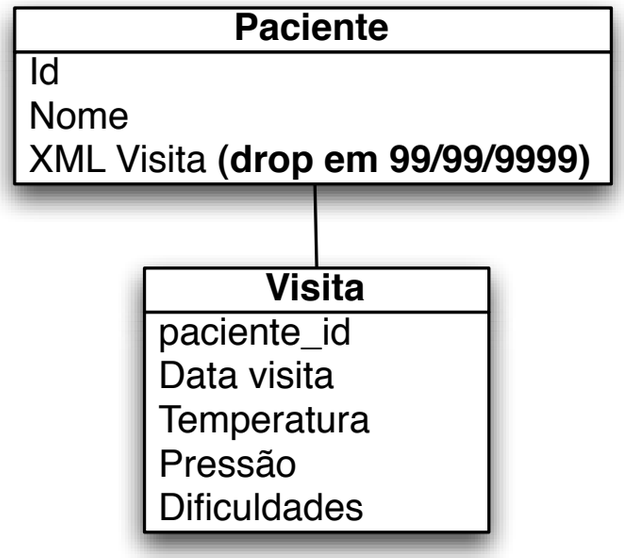
Motivação	Melhorar o entendimento das tabelas ou seguir uma convenção de nomes.
Avaliação	Verificar se os benefícios compensam o trabalho a ser feito, evitando preferências de nomes com o mesmo sentido.
Esquema	Período de transição: Criar uma nova visão com o novo nome, alertar que a visão antiga será removida. Finalização: Remover a visão antiga.
Dados	Não se aplica neste caso.
Aplicativo	Trocar as referências à visão antiga pela nova.

Refatoração estrutural 12/17

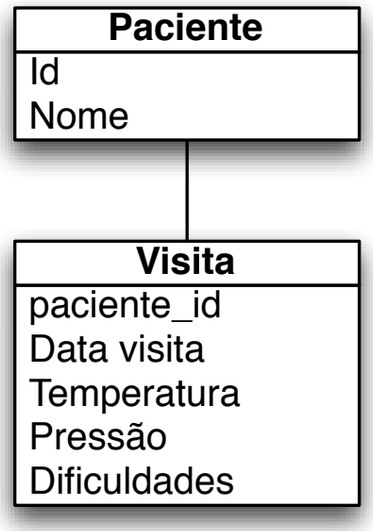
- Replace Large Object (LOB) with Table

Antes

Durante



Depois



Trocar coluna complexa por tabela

Motivação	Necessidade de tratar partes do *LOB* como elementos de dados distintos. É muito comum uma coluna *LOB* conter um arquivo XML e depois ser necessária esta refatoração.
Avaliação	Avaliar a necessidade de tratar partes desta coluna *LOB*, pois esta refatoração possivelmente exigirá maior processamento para as conversões de dados.
Esquema	Período de transição: crie uma tabela com as colunas necessárias, crie *triggers* para manter sincronizadas as tabelas e alerte que a coluna *LOB* será eliminada. Finalização: remover a coluna *LOB* e os *triggers*.
Dados	Carregar a nova tabela com os valores já existentes na coluna *LOB*.
Aplicativo	Escrever código que transforma o valor da coluna *LOB* na nova tabela e vice-versa.

Refatoração estrutural 13/17

- Replace Column

Antes

Documento
paciente_id
Número **(INT)**
Tipo documento

Durante

Documento
paciente_id
Número (**INT drop em 99/99/9999)**
Identificação (**CHAR)**
Tipo documento

Depois

Documento
paciente_id
Identificação (**CHAR)**
Tipo documento

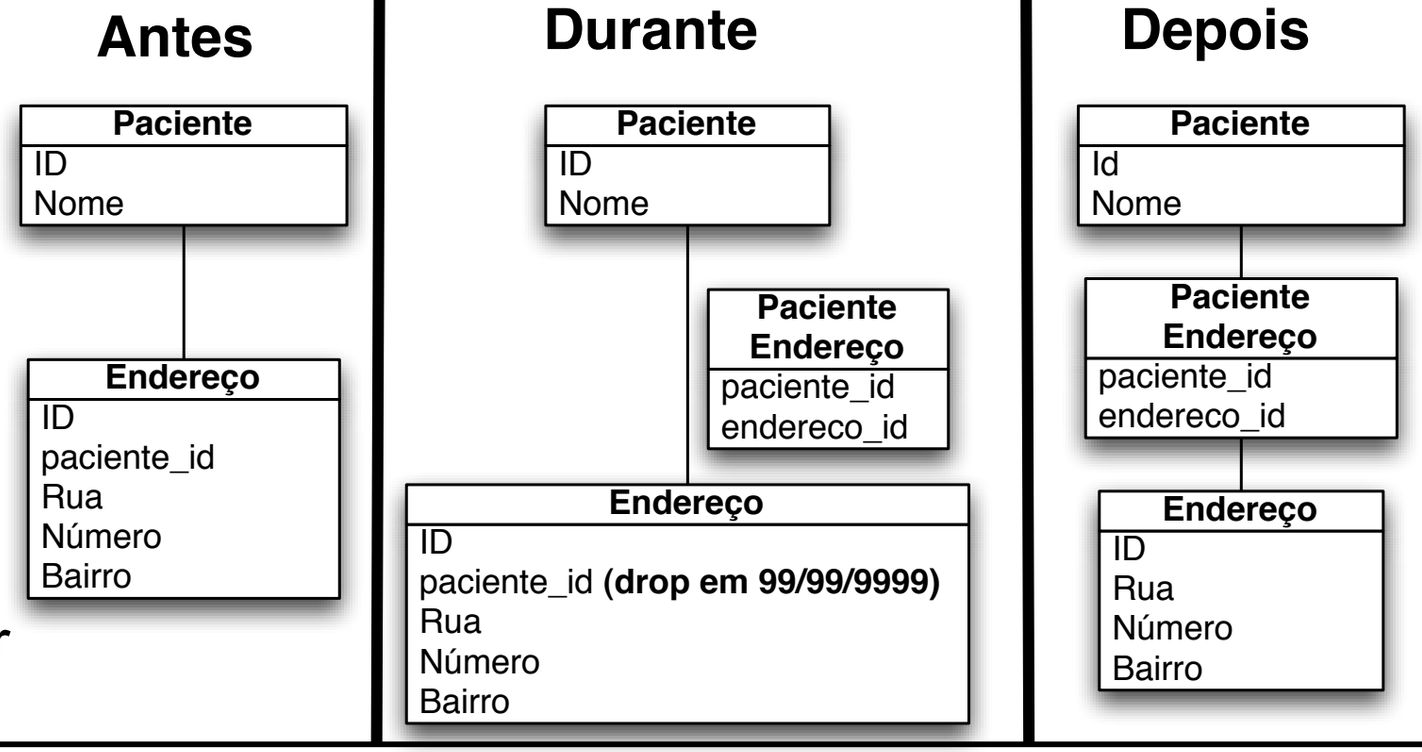
Trocar coluna

Motivação	A coluna deve trocar o tipo de dados (ex: numérico para alfanumérico), como parte de uma seqüência de refatoração.
Avaliação	Avaliar possível perda de dados caso o novo tipo de coluna for incompatível com tipo antigo. (Ex: troca de caractere para numérico).
Esquema	Período de transição: crie a nova coluna, alerte que a coluna antiga será eliminada, crie os *triggers* de sincronização. Finalização: remove os *triggers* e coluna antiga.
Dados	Carregar os valores existentes na nova coluna, fazendo as conversões possíveis.
Aplicativo	Trocar as referências à coluna antiga, verificando o tipo de dados das variáveis usadas.

Refatoração estrutural 14/17

- Replace One-to-Many with Associative Table

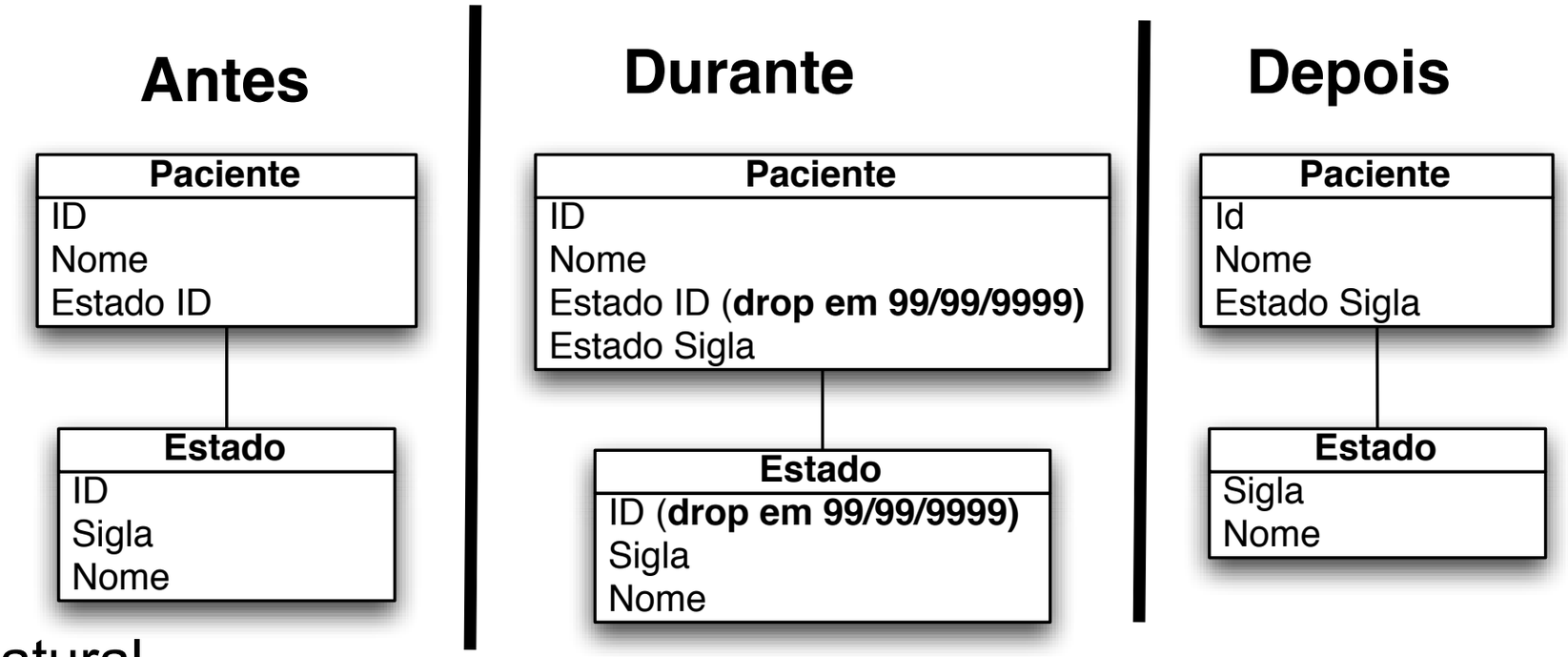
Trocar "um-para-muitos" por tabela associativa



Motivação	Transformar uma associação de “um-para-n” para “n-para-n”.
Avaliação	Verificar o desempenho dos comandos SQL que irão fazer mais trabalho para trazer os mesmos dados.
Esquema	Período de transição: crie a tabela associativa, alerte que a chave estrangeira da tabela será removida, escreva os *triggers* para manter as informações sincronizadas. Finalização: remova os *triggers* e a chave estrangeira.
Dados	Carregue os dados dos relacionamentos já existentes na tabela associativa.
Aplicativo	Troque as referências a chave estrangeira pela tabela associativa.

Refatoração estrutural 15/17

- Replace Surrogate Key With Natural Key



Trocar chave de identificação por chave natural

Motivação	Reduzir sobrecarga de uma chave desnecessária.
Avaliação	Verificar quais tabelas tem chaves estrangeiras para a tabela que será refatorada.
Esquema	Período de transição: definir qual será a nova chave principal, alertar que a coluna que é a atual chave principal será removida, acrescentar a nova chave estrangeira nas tabelas associadas e escrever o *trigger* para sincronizar os dados. Finalização: remover as colunas *surrogate key* e o *trigger*.
Dados	Carregar os valores corretos nas chaves estrangeiras para os relacionamentos já existentes.
Aplicativo	Trocar as referências das chaves antigas pela nova chave.

Refatoração estrutural 16/17

- Split Column

Antes

Paciente
ID
Nome

Durante

Paciente
ID
Nome (**drop em 99/99/9999)**
Primeiro Nome
Nome do meio
Sobrenome

Depois

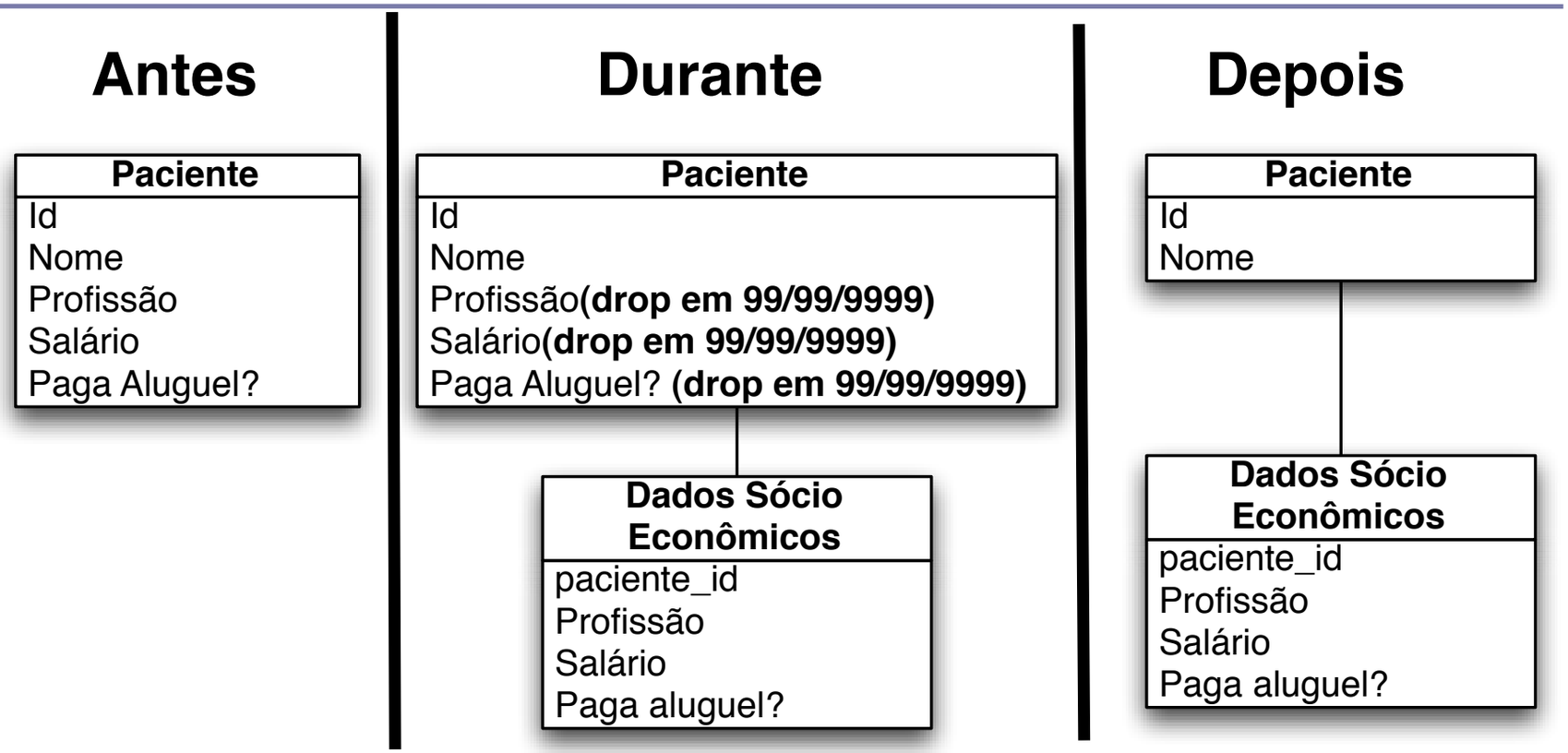
Paciente
Id
Primeiro Nome
Nome do meio
Sobrenome

Dividir coluna

Motivação	Necessidade de tratar as informações com um maior nível de detalhes.
Avaliação	Avaliar quais partes dos dados da coluna serão realmente utilizadas.
Esquema	Período de transição: adicione as novas colunas na tabela, alerte qual coluna será removida, escreva o *trigger* para manter os dados sincronizados. Finalização: remova a coluna que foi substituída pelas as outras colunas e remova o *trigger*.
Dados	Carregue as novas colunas a partir dos valores já existente na coluna que será substituída.
Aplicativo	Troque as referências à coluna completa pelas novas colunas.

Refatoração estrutural 17/17

- Split Table



Dividir tabela

Motivação	Separar informações importantes das informações acessórias ou não obrigatórias. Restringir o acesso a informações confidenciais. Efetuar normalização.
Avaliação	Avaliar corretamente quais informações (colunas) devem ser separadas da tabela principal.
Esquema	Período de transição: criar a nova tabela, escrever os *triggers* para manter esta nova tabela sincronizada e alertar quais colunas serão removidas da tabela principal. Finalização: remover as colunas da tabela principal e remover os *triggers*.
Dados	Carregar a nova tabela com os valores já existentes na tabela principal.
Aplicativo	Trocar as referências às colunas da tabela principal pela nova tabela.

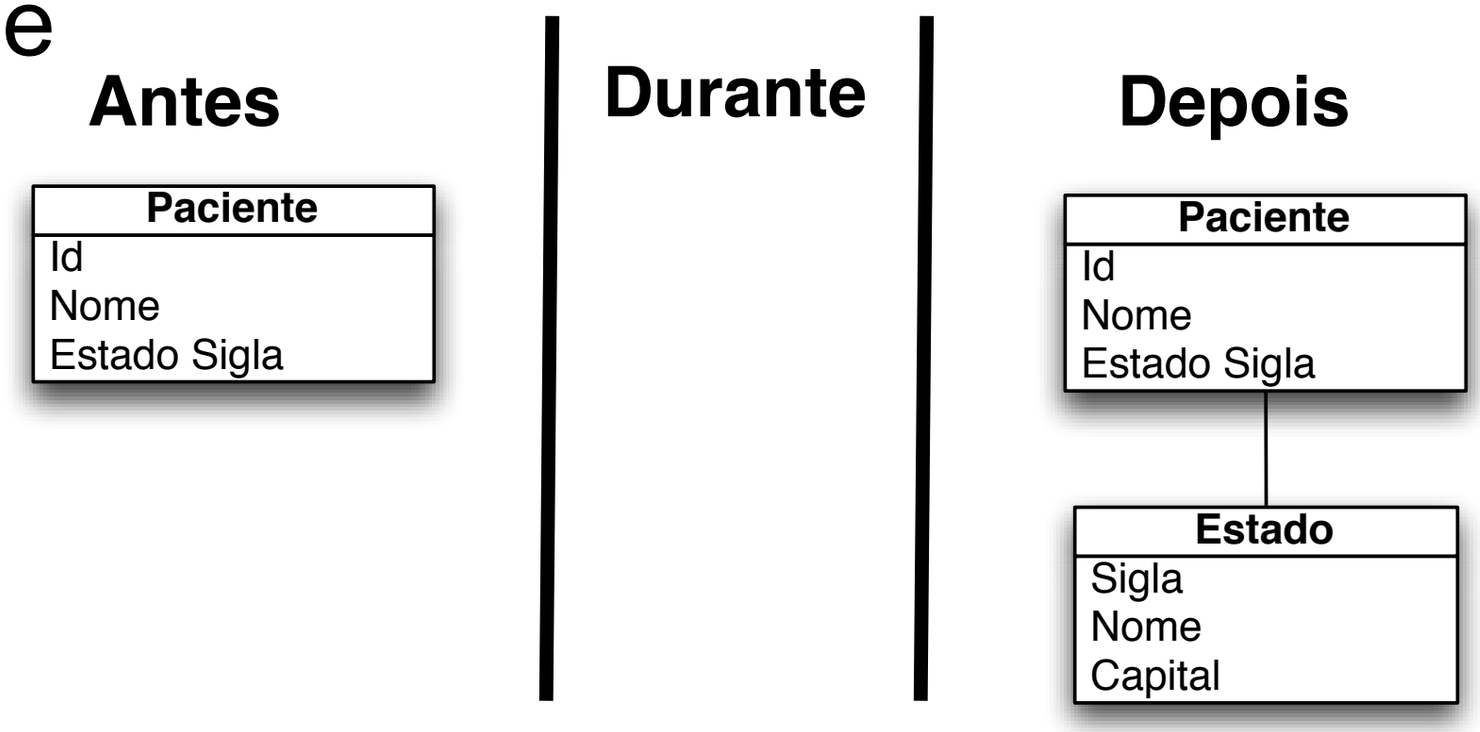
Resumo

- **Limpeza**
  - *Drop Column, Drop Table e Drop View*
- **Troca**
  - *Introduce Calculated Column, Introduce Surrogate Key*
  - *Replace LOB With Table, Replace Column, Replace 1-N With Table e Replace Surrogate Key With Natural Key*
- **Juntar ou Separar**
  - *Merge Column, Merge Table, Split Column e Split Table*
- **Organizar**
  - *Rename Column, Rename Table, Rename View e Move Column*

Refatorações de Qualidade dos Dados

Refatoração de Qualidade dos Dados 1/3

- Add Lookup Table

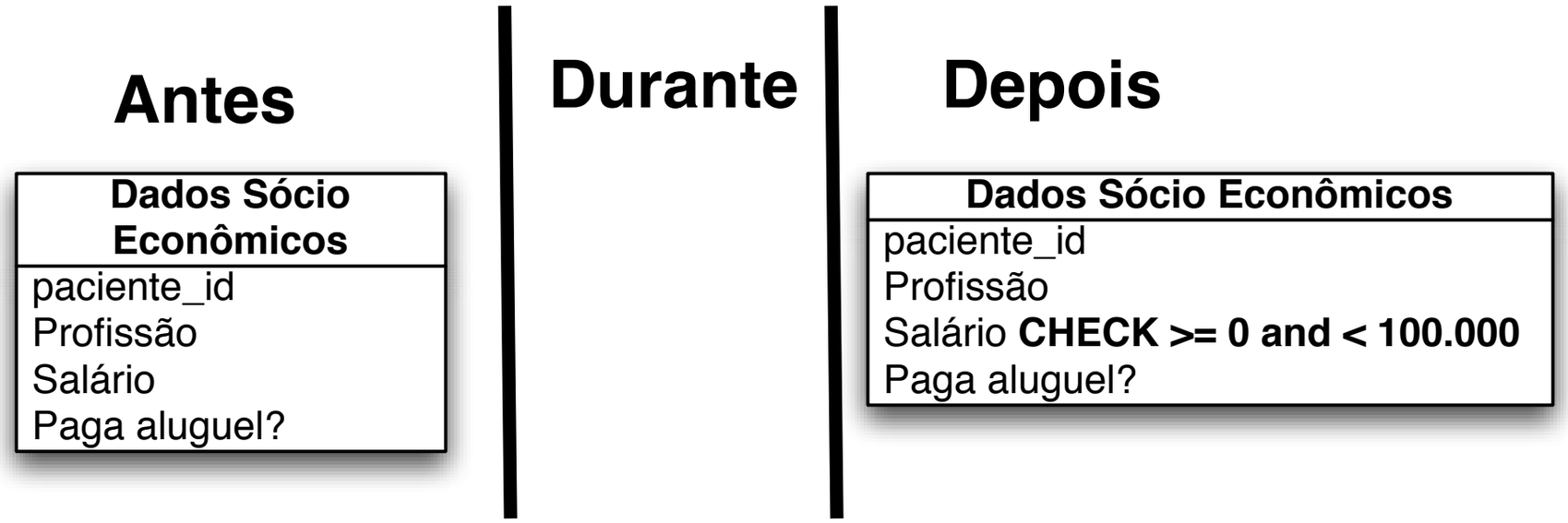


Adicionar tabela descritiva

Motivação	Introduzir integridade referencial, disponibilizar uma lista dos códigos possíveis, retirar uma validação existente no código e provê uma descrição para os códigos.
Avaliação	É necessário ter a lista completa para preencher a tabela de códigos. É preciso desenvolver aplicação para fazer (CRUD) nesta tabela.
Esquema	Período de transição: Criar a tabela de códigos e preencher com os valores possíveis Finalização: Criar a regra de restrição referencial.
Dados	Popular a tabela de códigos com os valores pré-existentes.
Aplicativo	Fazer a manutenção da tabela de códigos.

Refatoração de Qualidade dos Dados 2/3

- Introduce Column Constraint

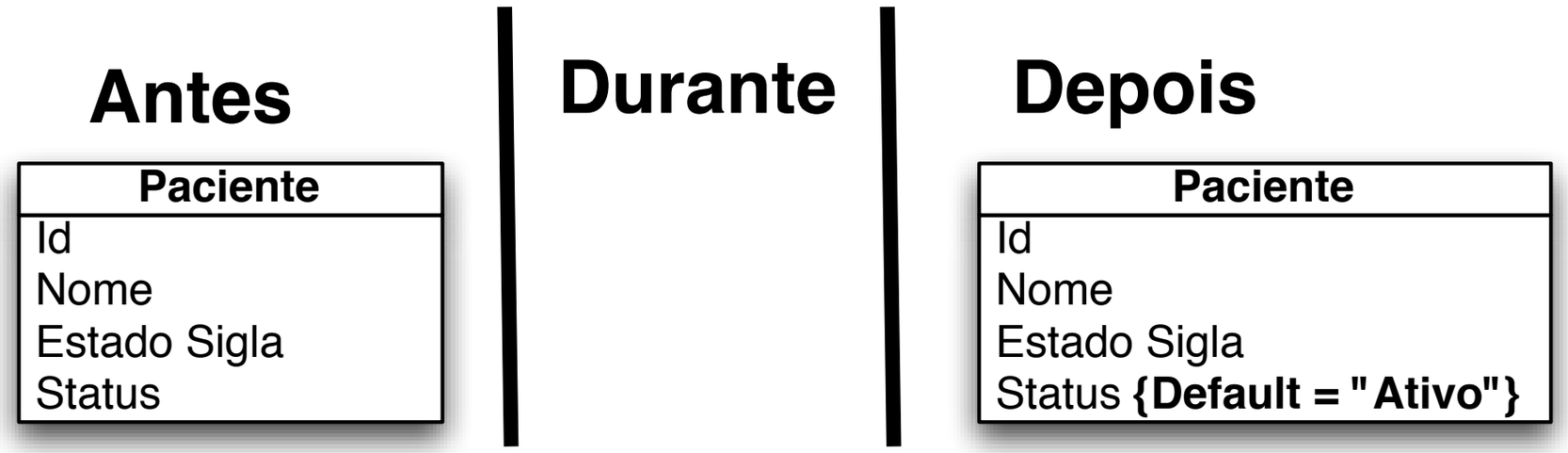


Introduzir restrição de coluna

Motivação	Implementar uma regra de validação dos dados comum a todas as aplicações do banco de dados.
Avaliação	Em ambiente heterogêneo deve avaliar se esta regra realmente atende todas as aplicações.
Esquema	Adicionar a regra de validação.
Dados	Verificar se os dados já existentes respeitam a regra de validação a ser criada.
Aplicativo	Tratar os erros provenientes da regra de validação.

Refatoração de Qualidade dos Dados 3/3

- Introduce Default Value



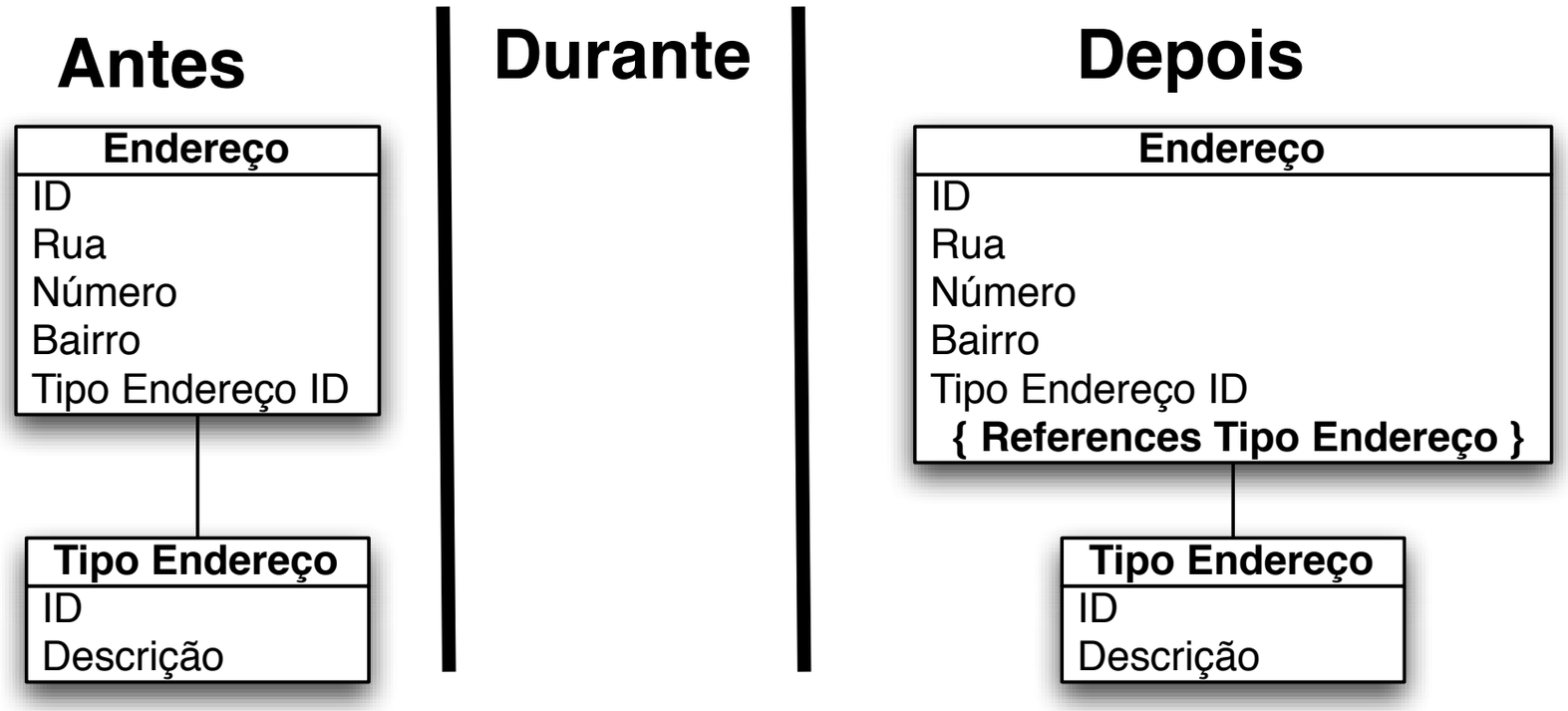
Introduzir valor padrão

Motivação	Uma nova coluna que a aplicação ainda não insere dados e garantir que uma coluna sempre tenha dados.
Avaliação	Saber definir um valor *default* correto. Verificar se a aplicação faz algum tratamento caso esta coluna tenha valor nulo.
Esquema	Alterar a tabela definindo para uma coluna o seu valor *default*
Dados	Fazer *script* para atualizar as linhas já existentes onde a coluna esteja nula.
Aplicativo	Verificar o código que faz a manutenção desta coluna alterada.

Refatorações de Integridades Referenciais

Refatoração de Integridade Referencial 1/3

- Add Foreing Key Constraint

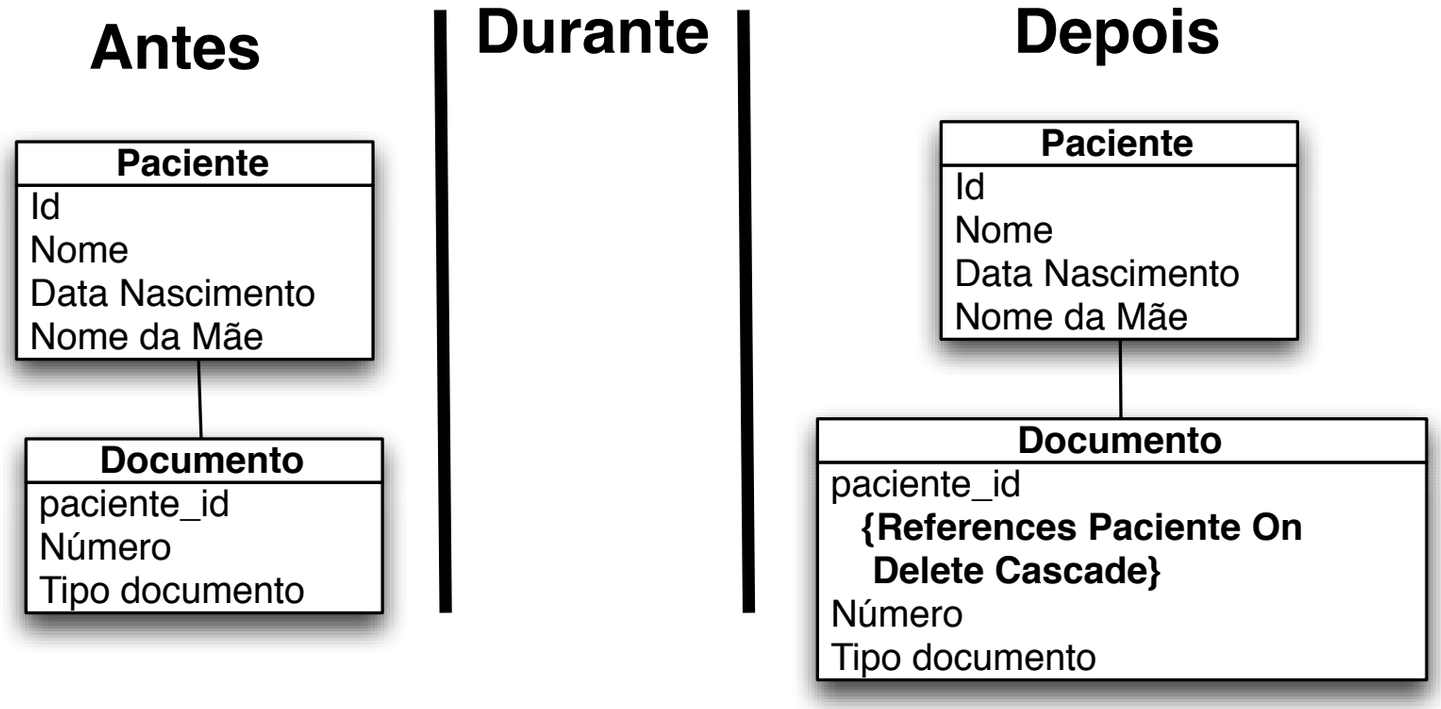


Adicionar restrição de chave estrangeira

Motivação	Garantir a integridade referencial através de regras no banco de dados.
Avaliação	Avaliar o impacto nas aplicações para os casos de inserção, atualização e exclusões na tabela com esta restrição.
Esquema	Definir a estratégia de validação: a cada comando ou somente no *commit*. Alterar a tabela adicionando a *foreing key*. Verificar se a tabela referenciada tem índice na coluna usada como chave estrangeira.
Dados	Garantir que a tabela referenciada está completa e que a tabela alterada só tem os valores válidos.
Aplicativo	Analisar a parte da aplicação que trata de regras similares a nova regra.

Refatoração de Integridade Referencial 2/3

- Introduce Cascading Delete



Introduzir exclusão em cascata

Motivação	Preservar a integridade referencial durante a exclusão de uma linha da tabela que tenha referências a outras tabelas.
Avaliação	Avaliar se a exclusão em cascata é o funcionamento desejado.
Esquema	Alterar a tabela para acrescentar a regra de exclusão em cascata.
Dados	Não se aplica.
Aplicativo	Se existir, retirar da aplicação o código que exclui as linhas associadas. Tratar possíveis erros (exceções) decorrentes a falhas destas exclusões.

Refatoração de Integridade Referencial 3/3

- Introduce Soft Delete

Antes

Paciente
Id Nome Status **{Default = "Ativo"}**

Durante

Depois

Paciente
Id Nome Status **{Default = "Ativo"}** **{ SoftDelete event=delete** **set Status = "Excluido"** **}**

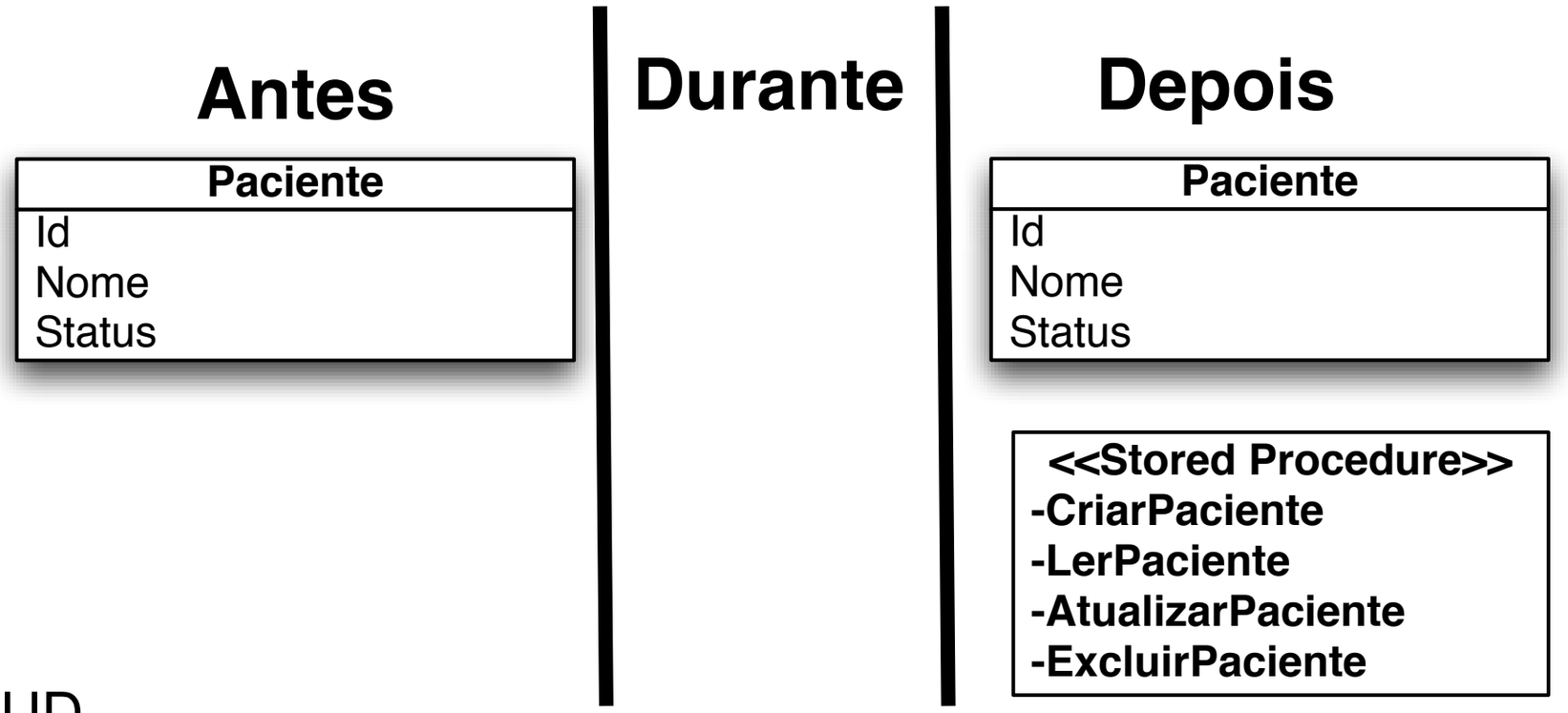
Introduzir exclusão lógica

Motivação	Preservar dados que seriam excluídos para permitir o armazenamento de um histórico.
Avaliação	Avaliar a perda de desempenho decorrente ao volume de dados armazenados na tabela.
Esquema	Período de transição: Acrescentar a coluna que indica se foi excluída ou não a linha. Finalização: Acrescentar o código que atualiza esta coluna.
Dados	A nova coluna deve ser inserida com valor negativo para todas as linhas da tabela.
Aplicativo	Acrescentar os filtros necessários para não mostrar as linhas excluídas. Acrescentar possível módulo que visualiza o histórico de exclusões.

Refatorações Arquiteturais

Refatoração Arquitetural 1/3

- Add CRUD Methods

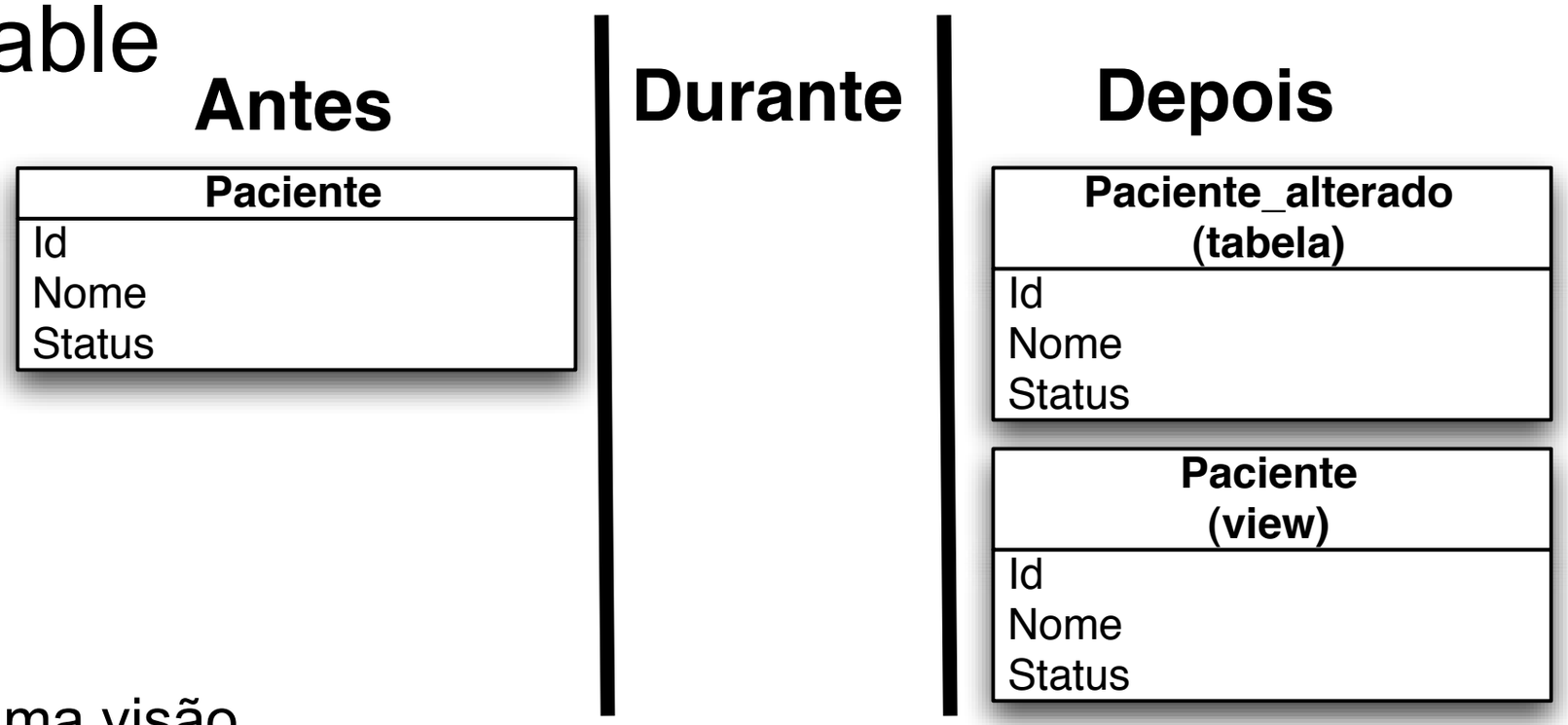


Adicionar métodos CRUD

Motivação	Encapsular o método de acesso. Diminuir o acoplamento entre a aplicação e o banco de dados. Implementar regras de negócio em ambiente heterogêneo.
Avaliação	Avaliar a diminuição de portabilidade entre banco de dados. Avaliar a quantidade código necessário para implementar esta refatoração.
Esquema	Período de transição: Criar as *stored procedures* Finalização: Retirar o acesso direto às tabelas das aplicações.
Dados	Não se aplica.
Aplicativo	Escrever as chamadas as *stored procedures* e eliminar o código semelhante.

Refatoração Arquitetural 2/3

- Encapsulate Table With View

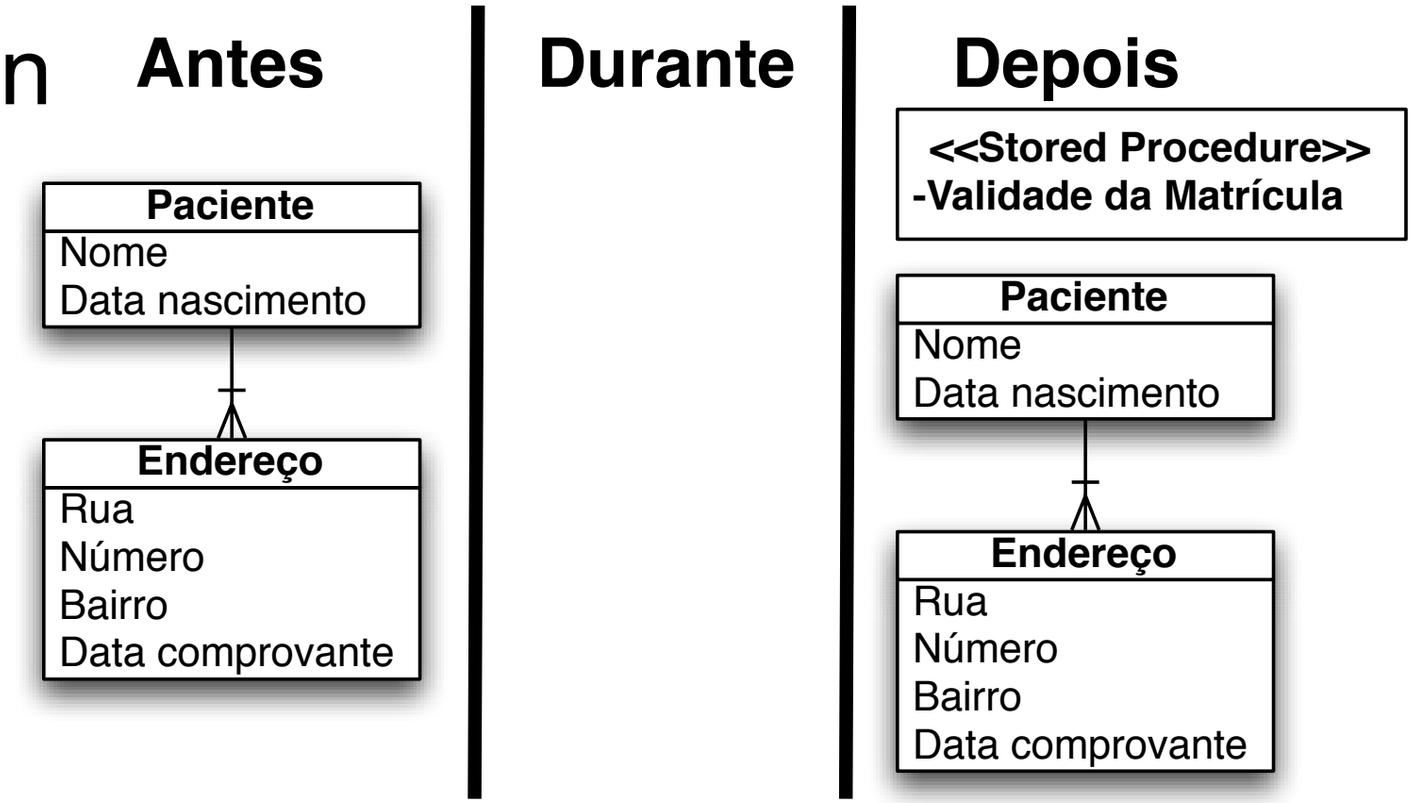


Encapsular tabela com uma visão

Motivação	Implementar uma fachada para tabelas que serão refatoradas. Implementar regras de segurança.
Avaliação	Verificar quais aplicativos não fazem somente consultas à tabela que será encapsulada. Estes aplicativos devem ser alterados antes desta refatoração.
Esquema	Trocar o nome da tabela e criar a nova visão (com o mesmo nome da tabela).
Dados	Não se aplica.
Aplicativo	Não se aplica.

Refatoração Arquitetural 3/3

- Introduce Calculation Method



Introduzir método para cálculo

Motivação	Melhorar o desempenho para cálculos que interagem com grande volume de dados. Implementar uma regra única de cálculo.
Avaliação	Avaliar o desempenho deste código no banco de dados (otimizando conforme a necessidade).
Esquema	Escrever a *stored procedure* com os parâmetros e o resultado conforme o especificado.
Dados	Não se aplica.
Aplicativo	Trocar a parte do código que fazia o cálculo pela chamada à *stored procedure*.

Resumo e Conclusão

Resumo das recomendações

- Implemente todos os pré-requisitos
- Defina uma estratégia de refatoração para o seu ambiente
- Implemente gradualmente as melhores práticas
- Estude as refatorações e consulte a documentação antes de executar as refatorações

Resumo das recomendações

- Encapsule o método de acesso ao banco de dados
- Escreva um serviço para acesso ao banco de dados
- Escreva o máximo possível de testes automatizados para o banco de dados.

Conclusão

- É possível fazer refatoração em banco de dados
- É possível (e necessário) evoluir BDs incrementalmente
- É necessário mais pesquisa para as refatorações e evoluções mais complexas.

FIM

Perguntas / Dúvidas ?

http://www.agilcoop.org.br

Minicurso SBES/SBBD'2008, Campinas

Helves Domingues, João Eduardo Ferreira e Fabio Kon

Departamento de Ciência da Computação