Banco de Dados Ágeis e Refatoração

Curso de Verão 2007 - IME/USP

www.agilcoop.org.br

Danilo Sato & João Eduardo Ferreira

AgilCoop

Pergunta:

Após colocar em produção, como fazer os banco de dados evoluirem facilmente de acordo com os novos requisitos ?

Pergunta:

Ou de forma mais específica, quem consegue mudar o nome de uma coluna do BD hoje e implantar essa alteração em produção amanhã?

Pergunta:

Qual é o principal o problema para o desenvolvimento de BD Ágeis?

Primeiro

“A HERANÇA MALDITA”

“Problema”:Persistência de objetos

- Em linguagens de programação OO:
  - **Objeto** é uma instância de uma classe.
    - Objetos têm **estados** (os valores de seus atributos).
    - Objetos têm **comportamentos** (seus métodos).
  - **Modelo de objetos** é uma coleção de todas as definições de classes de uma aplicação.
  - As classes podem representar:
    - Elementos de interfaces do usuário.
    - Recursos do sistema.
    - Eventos da aplicação.
    - Abstrações dos conceitos de negócio.
      - Nomes significativos para pessoas de negócio não-técnicas.

Definições

- Objetos que abstraem conceitos de negócio:
  - Num sistema de processamento de pedidos:
    - Cliente, Pedido e Produto.
  - Numa aplicação financeira:
    - Cliente, Conta, Crédito, Débito.
- Esses objetos modelam o domínio do negócio no qual a aplicação específica irá operar.
- Assim, eles são coletivamente chamados de **Modelo de Objetos de Domínio**.

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**.

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

- 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 BD Relacional enfrentam o problema da **incompatibilidade conceitual** (*impedance mismatch*).

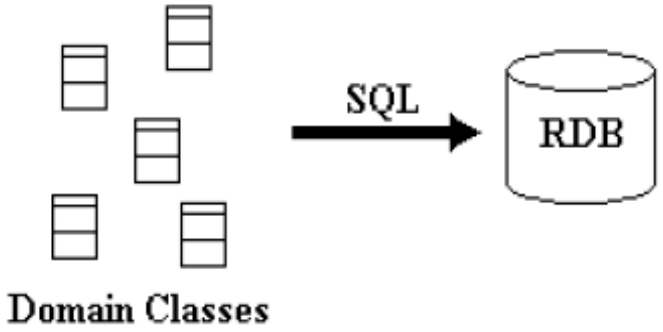
Técnicas de Persistência

- Para superar este problema é importante conhecer:
  - O processo de mapeamento objeto-relacional.
  - Como implementar mapeamento objeto-relacional.
  - E, fundamentalmente, como tornar ágil a manutenção do mapeamento!

Técnicas de Persistência - Força Bruta

- Força Bruta (mais comum):
  - o código SQL é embutido no código-fonte das classes.
  - Vantagens:
    - permite escrever código 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

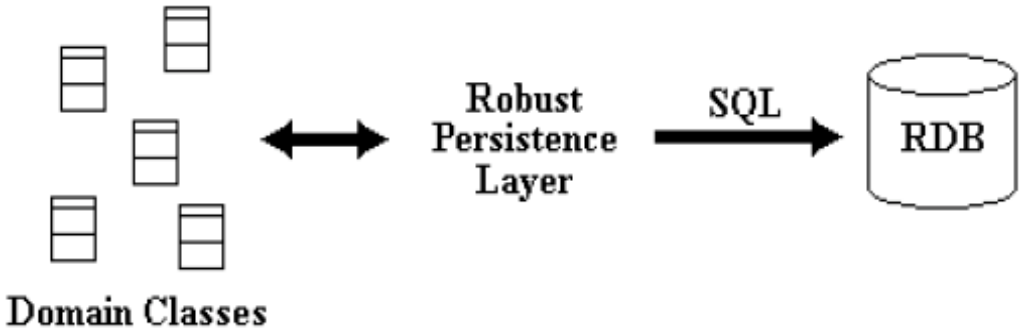
- Classes de Dados:
  - SQL das classes de domínio são encapsuladas nas "classes de dados". Exemplos:
    - SP's no BD representam objetos (substituindo as classes de dados)
    - ActiveX Data Objects (ADO).
  - Vantagens:
    - Eficiência na execução das funções de inserção e recuperação dos dados.
  - Desvantagens:
    - Recompilação das classes de dados quando pequenas mudanças são feitas no BD.
    - Recursos humanos consideráveis para implementação SQL.

Técnicas de Persistência – Classes de Dados

Técnicas de Persistência – *Framework*

- Camada de Persistência Robusta (*framework*):
  - O *framework* mapeia objetos para BDs relacionais, de maneira que as pequenas mudanças no esquema relacional não afetem o código OO.
  - Vantagens:
    - Programadores não precisam conhecer nada a respeito do esquema do BD relacional.
    - Permite que a empresa desenvolva aplicações de missão crítica e em larga escala.
  - Desvantagem:
    - Impacto no desempenho das aplicações.

Técnicas de Persistência – Framework



Características da camada de Persistência

- As camadas de persistência devem permitir que desenvolvedores de aplicação se concentrem no que eles sabem fazer melhor, desenvolver aplicações, sem ter a preocupação com o como os seus objetos serão armazenados.
- Devem, também, permitir que administradores de banco de dados façam o que eles fazem melhor, administrar bancos de dados, sem ter que se preocupar com a introdução acidental de erros nas aplicações existentes.

Características da camada de Persistência

1. Vários tipos de mecanismos de persistência.
2. Total encapsulamento dos mecanismos de persistência.
3. Transações.
4. Identificadores de objetos.
5. Cursores.
6. Proxies.
7. Múltiplas conexões.
8. *Drivers* nativos e não-nativos.

Situação Atual

- Scott Ambler (Julho 2006):
  - 95.7% consideram dados como bens corporativos (*assets*)
  - 40.3% possuem bateria de testes para BD
  - 61.9% possuem problemas com dados em produção
  - 18% sem estratégia para corrigí-los
  - 33% estratédia é não deixar ficar pior

Disparidades

- Programadores vs. DBAs ?
- Metodologias:
  - Quando projetar?
  - Práticas de Teste
- Cultural
- Generalistas vs. Especialistas

BDs Ágeis

- Novas técnicas para aproximar as comunidades:
  - Modelagem evolutiva
  - Controle de versão de artefatos do BD
  - Testes automatizados
  - *Sandboxes* individuais
  - Refatoração

Modelagem Evolutiva

- Modelo evolui com o conhecimento
- Não precisa acertar (e congelar) da primeira vez
- Alinhado com os desenvolvedores
- Integração contínua
- Mudança de paradigma
- Suporte de novas ferramentas

Modelagem Evolutiva

- Ruby on Rails
  - Scripts de migração
    - API Ruby/SQL ou SQL
    - Alterações de avanço (`up`) e retrocesso (`down`)
    - Numerados sequencialmente
  - Tabela para armazenar versão do BD
  - Tarefa `rake` para migrar a versão do BD
  - Gerenciamento de migrações em diferentes ambientes (Desenv. / Teste / Produção)

Modelagem Evolutiva

- Outras ferramentas:
  - dbdeploy (Java)
    - Scripts delta escritos em SQL
    - Tarefa ANT para executar migrações
  - MIGRATEdb (Java)
    - SQL armazenado em XML
    - Não permite retroceder versões
  - DBGhost
  - Sundog
    - Inclui IDE para refatoração

Controle de Versão

- Como armazenar o histórico da evolução do BD?
- Como desfazer uma mudança?
- Desenvolvedores já utilizam ferramentas:
  - CVS
  - Subversion
- Toda alteração no BD deve ser armazenada!

Controle de Versão

- Artefatos:
  - Scripts de criação (DDL)
  - Scripts de migração de dados (delta)
  - Modelos de dados
  - Arquivos de mapeamento O/R
  - Dados de referência
  - *Stored Procedures*
  - *Triggers*

Controle de Versão

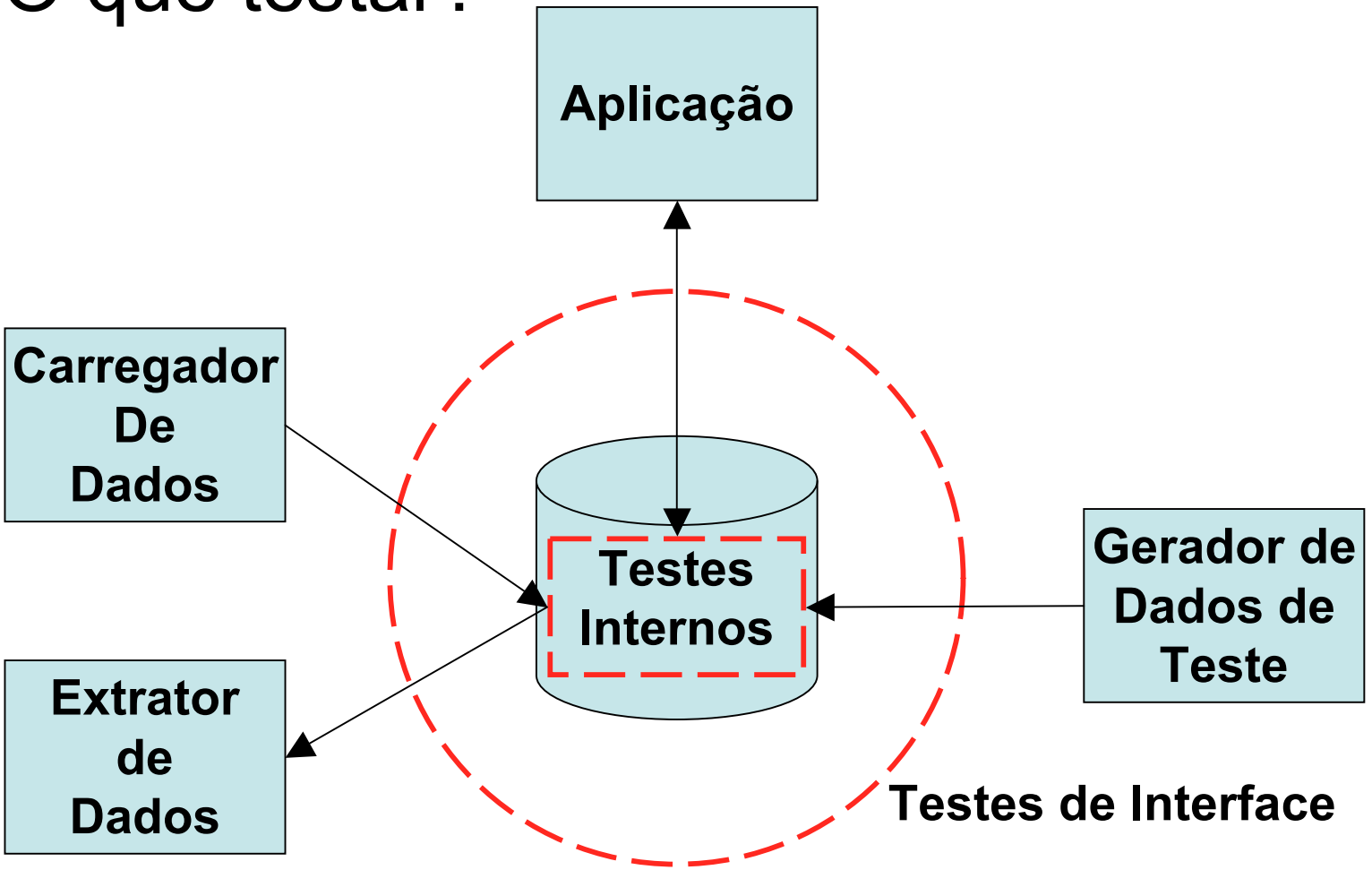
- Artefatos:
  - *Views*
  - Restrições de integridade
  - Índices
  - Sequências
  - Dados de teste
  - Scripts para gerar dados de teste
  - Scripts de teste

Testes Automatizados

- Como garantir a qualidade do BD?
- Como verificar que uma alteração não quebrou funcionalidades existentes?
- Bateria de testes de regressão
- Desenvolvedores já utilizam ferramentas:
  - XUnit, Selenium, FIT, …

Testes Automatizados

- O que testar?



Testes Automatizados

- Testes de interface:
  - Testam corretude dos dados que entram e saem do BD
  - Estilo “caixa-preta”
  - Simulam a interação das diversas aplicações externas com o BD
  - Exemplos:
    - Validar dados antes de persistir
    - Validar dados retornados
    - Testar mapeamentos O/R

Testes Automatizados

- Testes internos:
  - Testam integridade interna do BD
  - Estilo “caixa-branca”
  - Falta ferramentas
  - Testes de unidade para:
    - *Stored Procedures*
    - *Triggers*
    - Restrições de Integridade
    - *Views*
  - Testes de qualidade dos dados

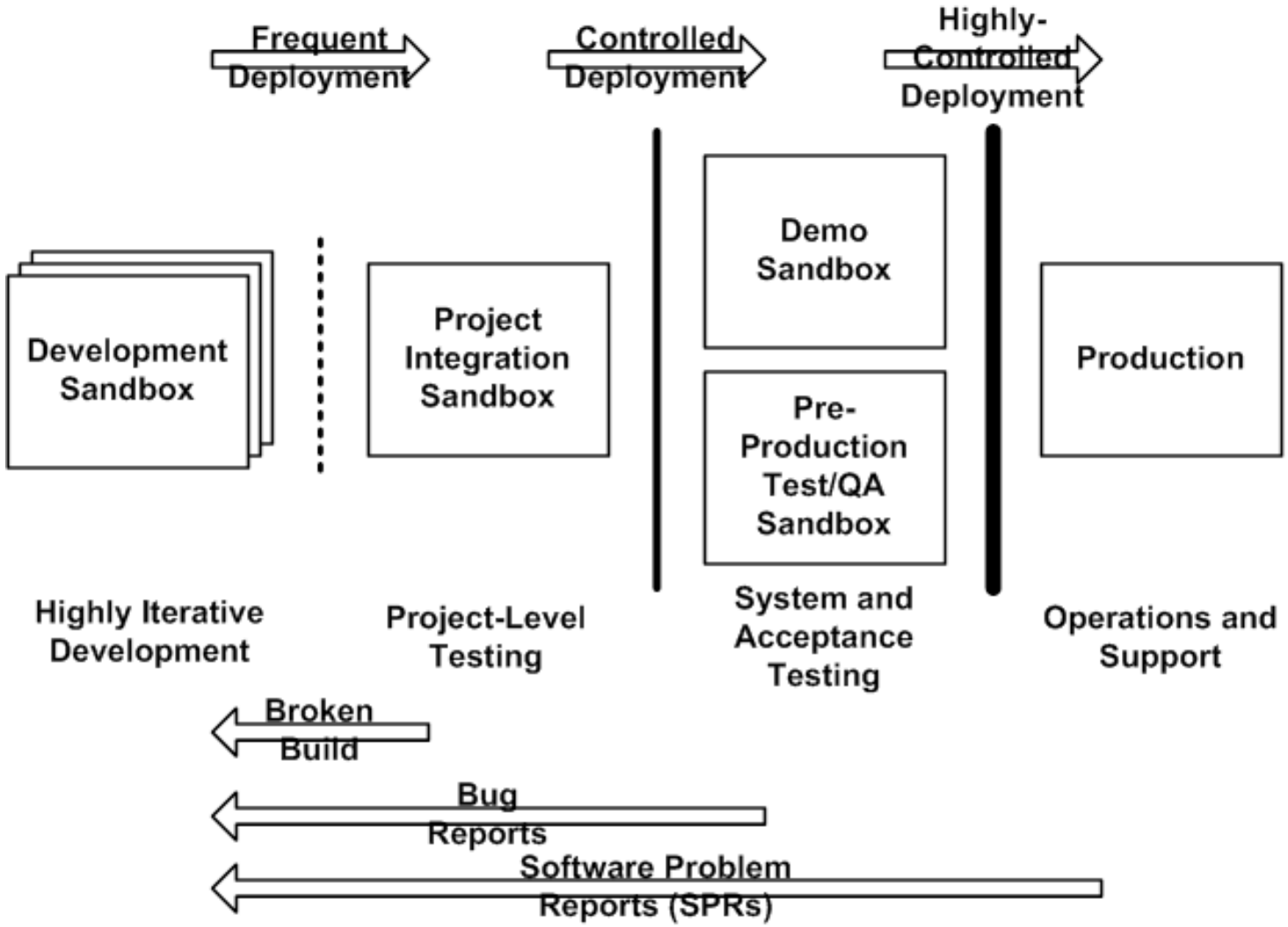
Testes Automatizados

- Como escrever testes?
  - Criar dados e cenários
  - Executar teste
  - Validar resultados
- TDD
- Poucas ferramentas ainda:
  - DBUnit, DBM Data Generator, NDbUnit, OUnit
  - Quest Unit Tester, TSQLUnit

Sandboxes Individuais

- Ambientes independentes para testar alterações
- Cada *sandbox* possui uma cópia do BD inteiro
- Diversos níveis de isolamento
- Diminuem o risco de um erro impactar muita gente
- Deve ser fácil criar um novo *sandbox*

Sandboxes Individuais



Refatoração de BD

"Pequena alteração no BD para melhorar o design, preservando o comportamento e a semântica dos dados"

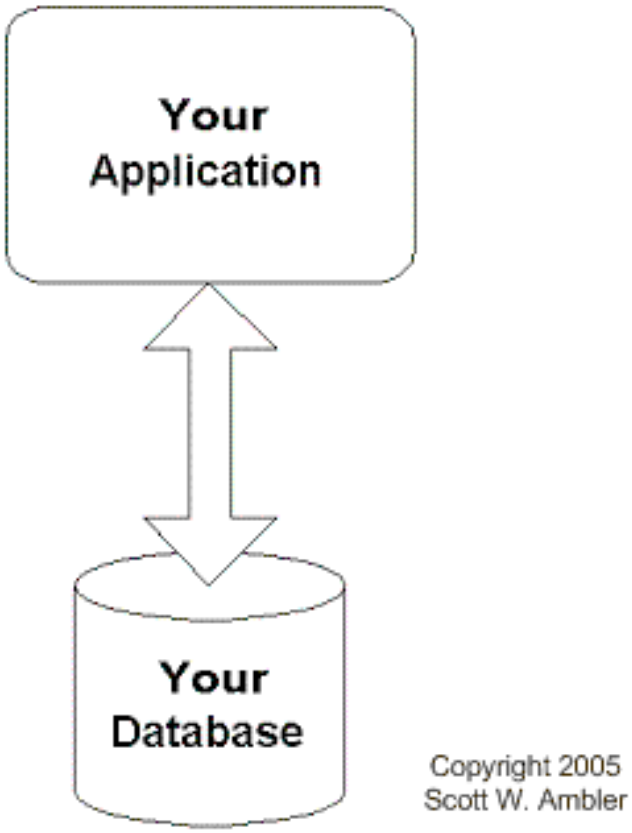
- Refatorar o BD é mais difícil que código:
  - Estrutura do BD
  - Dados
  - Código dos sistemas que acessam o BD
- Dificuldade varia com acoplamento

Refatoração de BD

- Dificuldade varia com acoplamento:
  - Diversas aplicações
  - Scripts de extração de dados
  - Scripts de importação de dados
  - Frameworks de persistência
  - Documentação

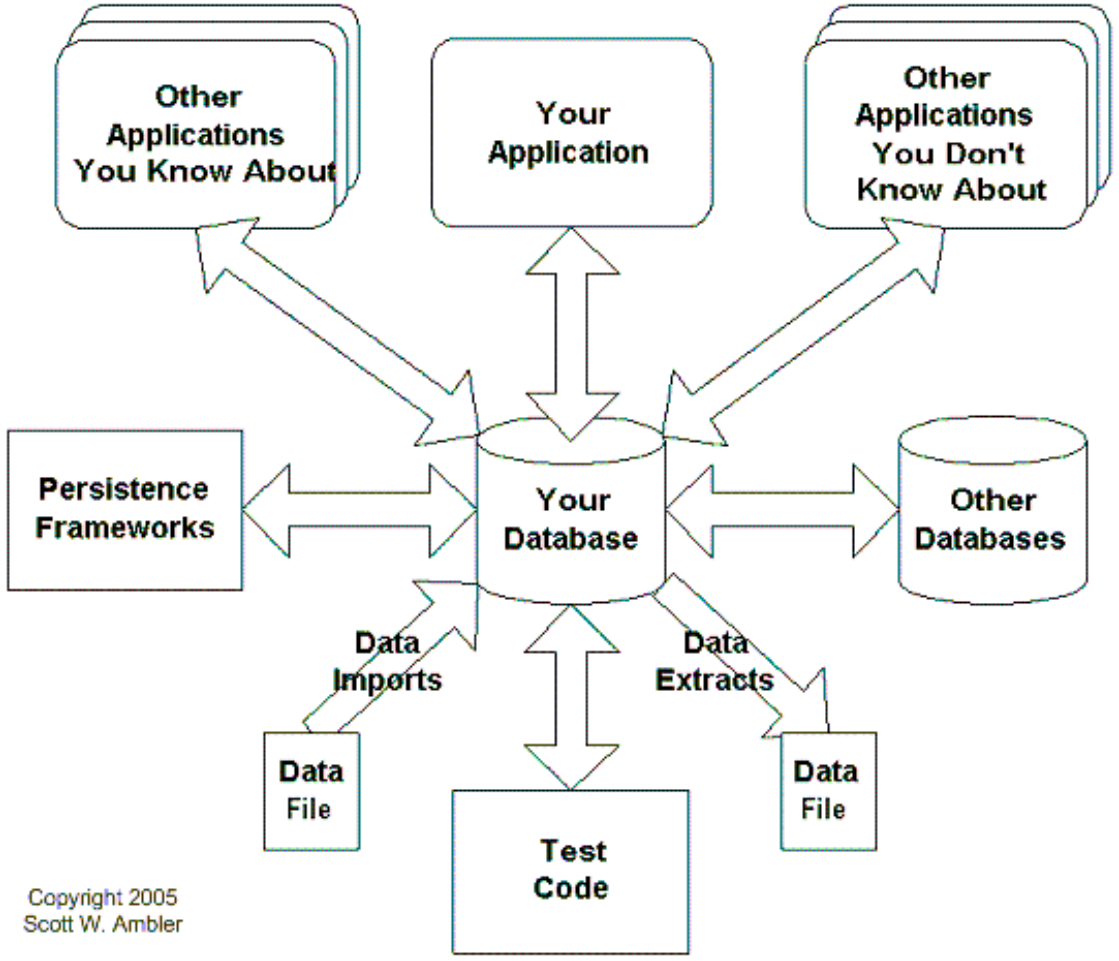
Refatoração de BD

- Uma aplicação acessando o BD



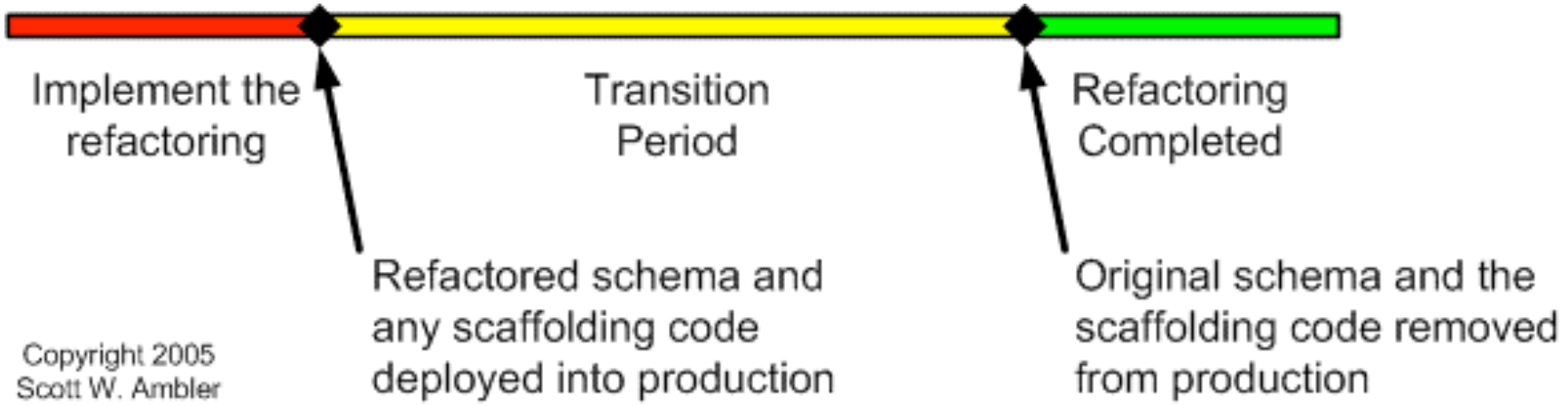
Refatoração de BD

- Várias aplicações acessando o BD



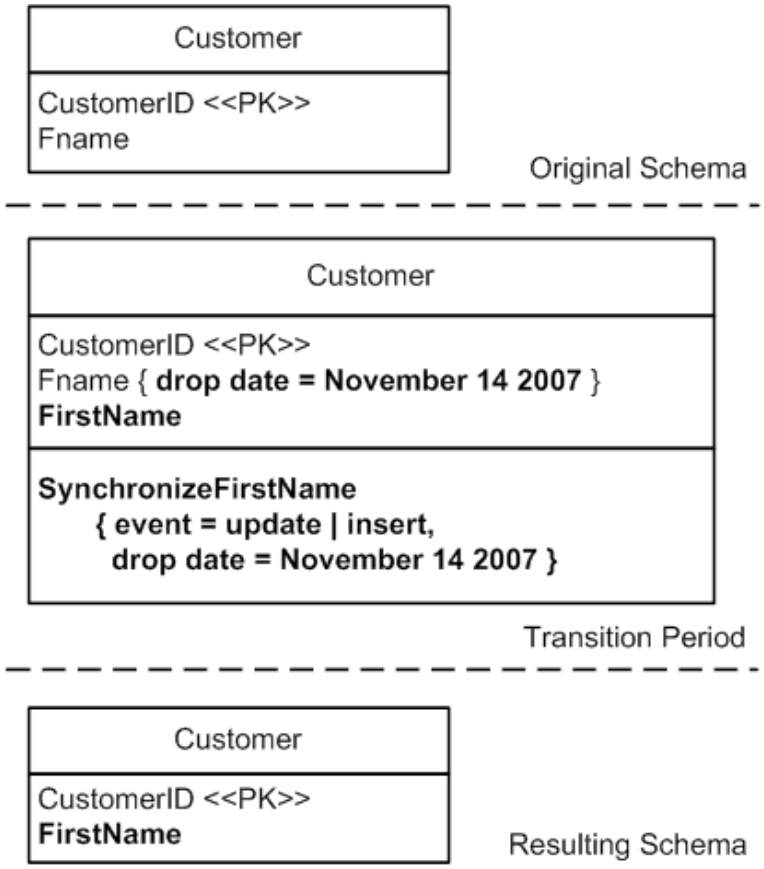
Refatoração de BD

- Uma refatoração de BD pode exigir um período de transição:



Refatoração de BD

- Exemplo: Renomear Coluna



Refatoração de BD

- O que NÃO é uma refatoração:
  - Alterar estrutura para estender o modelo
  - Fazer alterações grandes de uma vez
- Uma Transformação pode ou não mudar o comportamento
- Podem ser um passo da Refatoração
- Exemplo: Introduzir Coluna

Refatoração de BD

- Categorias de Refatoração:
  - Estruturais
  - Qualidade de dados
  - Integridade referencial
  - Arquiteturais
  - Método
  - Transformações (não são Refatorações)

Refatoração de BD

- “Mal cheiros” são sintomas para refatorar
  - Colunas multi-uso
  - Tabelas multi-uso
  - Dados redundantes
  - Tabelas com muitas colunas
  - Tabelas com muitas linhas
  - Colunas “espertas”
  - Resistência a mudanças

Processo de Refatoração de BD

1. Verificar se a refatoração é apropriada
   - A refatoração faz sentido?
   - A mudança é necessária agora?
   - O esforço recompensa?
2. Escolher a refatoração mais apropriada
   - Os dados podem estar em outro lugar
3. Depreciar o esquema original do BD
   - Passo opcional
   - Analisar a necessidade e duração da transição

Processo de Refatoração de BD

4. Testar antes, durante e depois
   - Esquema do BD
   - Migração de dados
   - Aplicações que usam o BD
5. Executar as alterações:
   1. Modifique o esquema do BD
   2. Faça as migrações necessárias
   3. Refatore as aplicações externas

Processo de Refatoração de BD

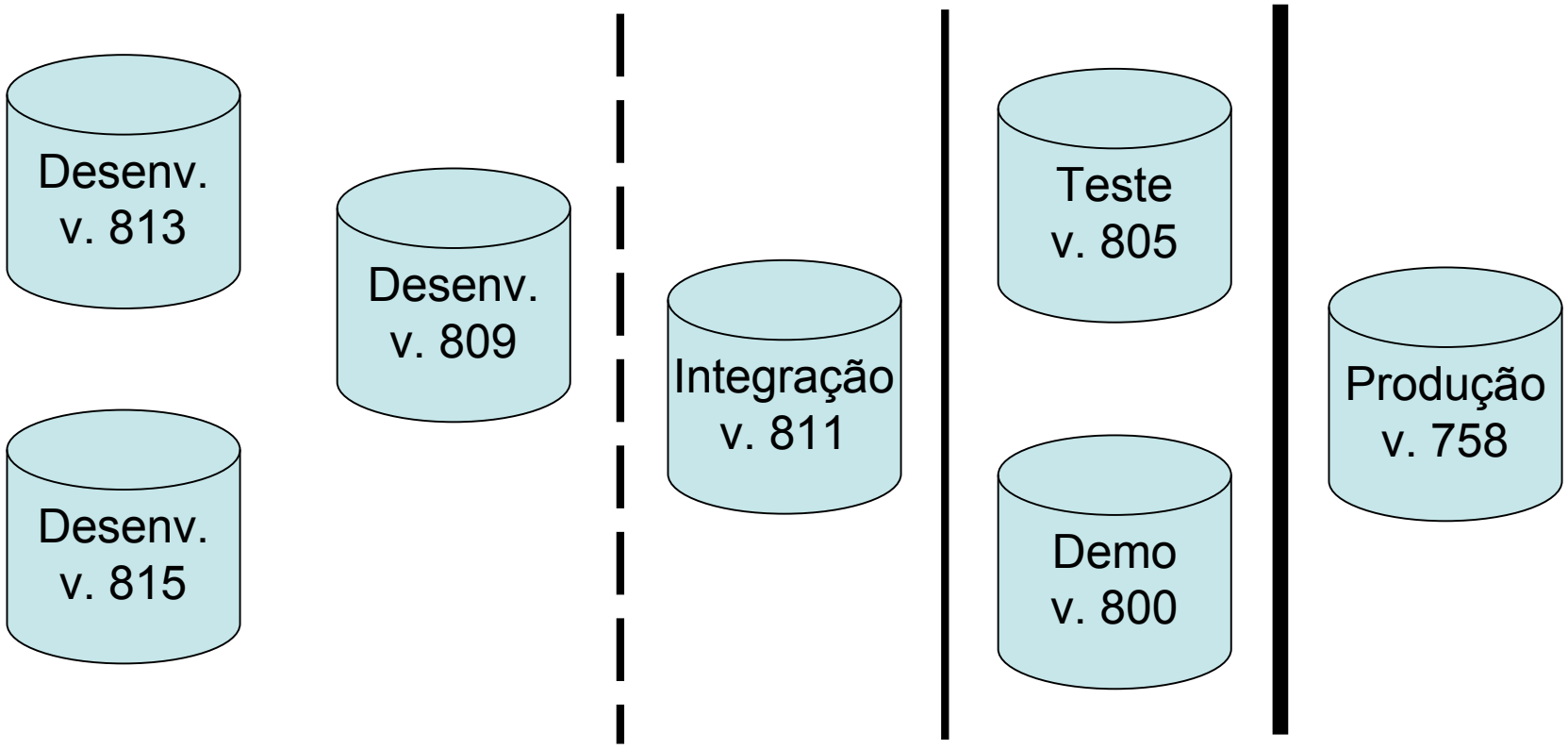
6. Executar os testes de regressão
7. Armazenar as alterações no repositório
8. Anunciar a refatoração
   - Atualizar documentação
   - "*Release Notes*" é uma estratégia simples:
     - Associa o número dos scripts da refatoração com a alteração realizada

Antes de Implantar

- Refatoração é executada no *sandbox* do desenvolvedor
- Precisa ser implantada nos outros *sandboxes* de integração
- Processo simples:
  - Fazer o *build* de toda a aplicação
  - Executar os scripts das refatorações
  - Executar os testes de regressão

Antes de Implantar

- É interessante poder executar um conjunto de scripts de uma só vez



Antes de Implantar

- É preciso agendar janelas para implantação
- Geralmente em períodos de baixa atividade no sistema

Processo de Implantação

1. Fazer um *backup* do BD
2. Executar os testes de regressão
   - Antes, é preciso garantir que tudo está funcionando
   - Se falhar, pode ser melhor abortar
3. Implantar as alterações nas aplicações
4. Implantar as alterações no BD
5. Executar os testes de regressão

Processo de Implantação

6. Desfazer, caso necessário
   - Falhas sérias nos testes de regressão
   - Utilize os *backups* do passo 1
   - Desfaça as alterações nas aplicações
7. Anunciar a implantação

- A refatoração não está completa até a remoção do esquema depreciado

Dicas e Estratégias

- Mudanças pequenas são mais simples
- Utilize identificadores para refatorações:

Estratégia	Vantagens	Desvantagens
Número do *Build*	Simples / FIFO Versão do BD alinhada com a aplicação	Números “saltados” Difícil de gerenciar se houverem muitas aplicações
Data/Timestamp	Simples / FIFO	Estranho associar o nome do script Mapear c/ Aplicação
ID Único	Estratégias existentes para gerar o ID (GUID)	Não definem a ordem Mapear c/ Aplicação

Dicas e Estratégias

- Implante uma alteração grande como várias pequenas:
  - Separar Tabelas:
    - Introduzir Nova Tabela
    - Mover Coluna (diversas vezes)
      - Introduzir Nova Coluna
      - Mover Dados
    - Introduzir Índice

Dicas e Estratégias

- Utilize uma tabela de configuração no BD:
  - Armazena a versão atual do BD
- Escolha um período de transição suficiente:
  - Trabalhe para diminuir o tempo ao máximo
  - Cenário ideal: Implantação Contínua (XP2E)

Dicas e Estratégias

- Sincronização:

Estratégia	Vantagens	Desvantagens
Triggers	Atualização em tempo real	Performance Ciclos (*deadlock*) Duplicação
Views	Atualização em tempo real Não precisa mover dados	Nem todos SGBDs dão suporte Complexidade adicional (adicionar e remover)
Atualizações em *Batch*	Desempenho	Duplicação, versão de dados e integridade

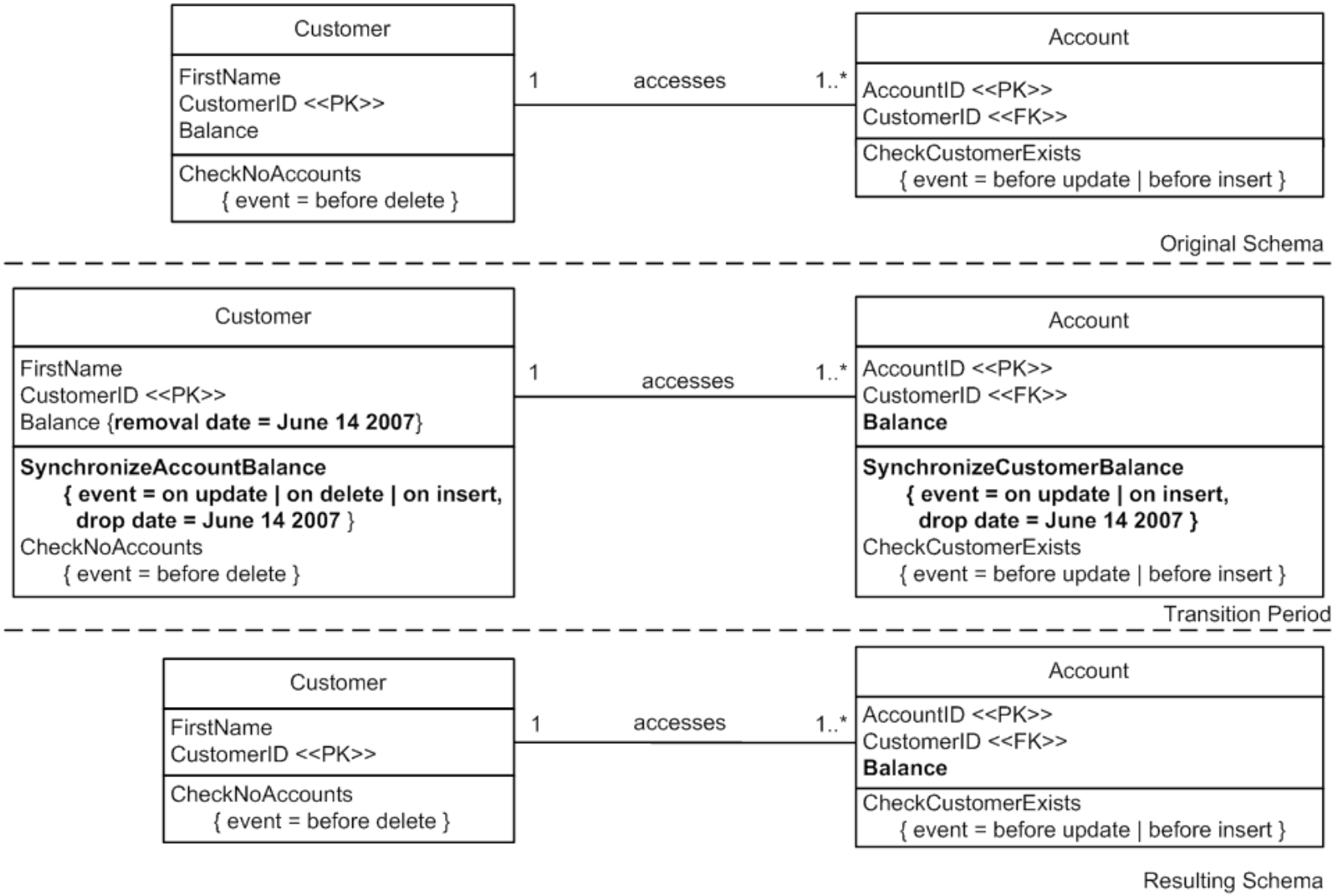
Dicas e Estratégias

- Encapsule o acesso ao BD:
  - Reduz o acoplamento
- Simplifique o processo de criação do BD em um novo ambiente
- Evite duplicação de código SQL
- Evite acessar colunas pela posição
- Prefira acesso pelo nome

Refatoração Estrutural

- Cuidados:
  - Evite ciclos em triggers
  - *Triggers* podem ser afetadas
  - *Views* podem ser afetadas
  - *Stored Procedures* podem ser afetadas
  - Outras tabelas podem ser afetadas
  - Defina o período de transição

Exemplo: Mover Coluna



Refatoração de Qualidade de Dados

- Cuidados:
  - Restrições de integridade podem ser afetadas
  - *Views* podem ser afetadas
  - *Stored Procedures* podem ser afetadas
  - Estratégia para atualização de dados:
    - Travar todos os dados que serão atualizados
    - Travar parte dos dados que serão atualizados

Refatoração de Qualidade de Dados

- Exemplo: Introduzir Formato Padrão

Before

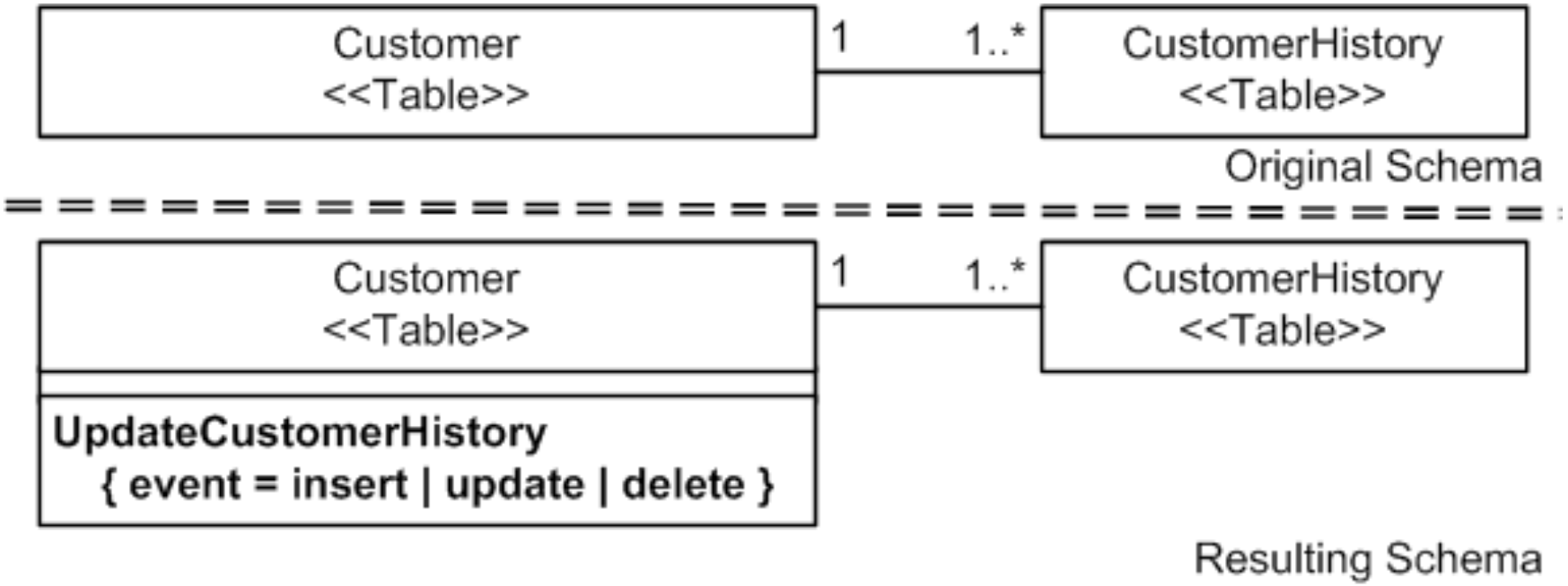
PhoneNumber
(416) 967-1111
9055551212
415.555.1234
(416) 555 1234
+1 905 234-5678
4166546543

After

PhoneNumber
4169671111
9055551212
4155551234
4165551234
9052345678
4166546543

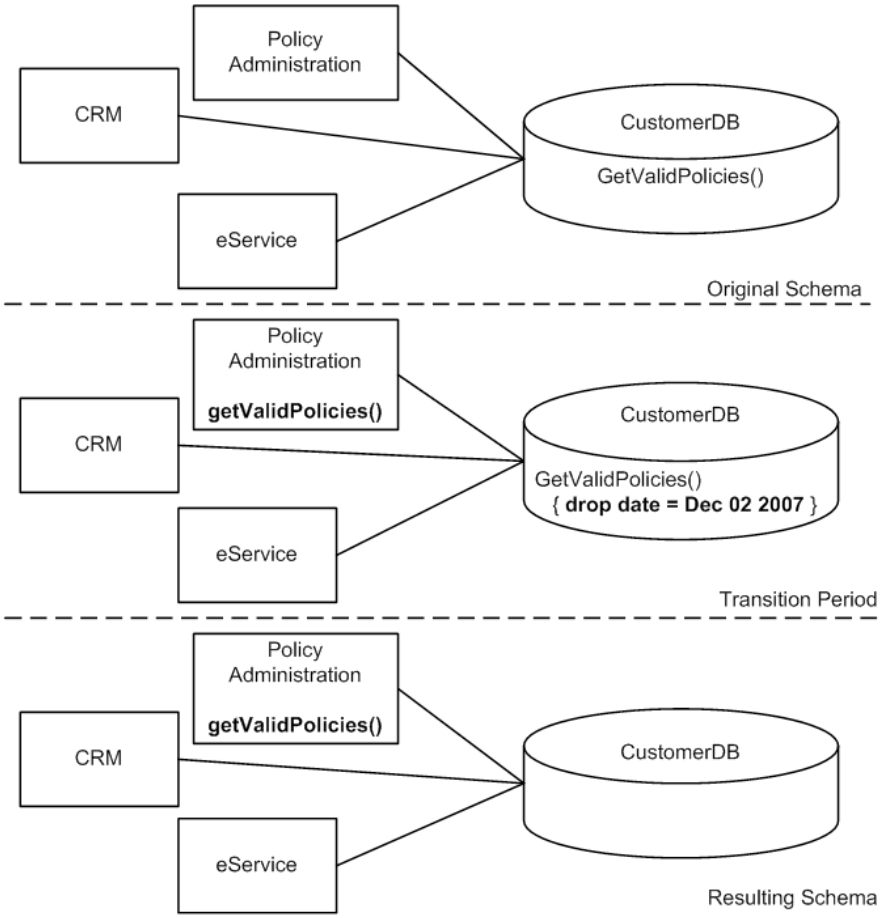
Refatoração de Integridade Referencial

- Exemplo: Introduzir Trigger para Histórico



Refatoração Arquitetural

- Exemplo: Migrar Método do BD

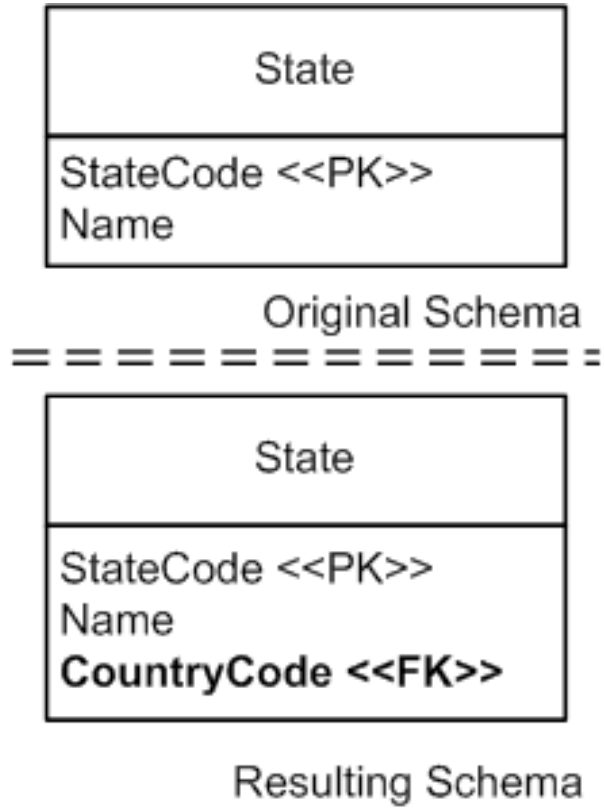


Refatoração de Métodos

- Exemplo: Parametrizar Método

Transformação

- Exemplo: Introduzir Nova Coluna



Conclusões

- É preciso quebrar barreiras culturais.
- Aproximar programadores e DBAs.
- É possível trabalhar de forma evolutiva com banco de dados.
- Ainda existem poucas ferramentas para a migração de dados.
- A Refatoração não implica em mudanças drásticas nos modelos de dados.

Epitáfio

- O sucesso de BD Ágéis está fortemente ligado a facilidade de evolução do modelo de dados.
- A evolução do modelo de dados implica em processos de migração de dados.
- A facilidade da migração dos dados depende do nível de abstração do modelo de dados.

Referências

- Livros:
  - Scott Ambler e Pramod Sadalage, “Refactoring Databases: Evolutionary Database Design”, Addison-Wesley Professional, 2006
  - Scott Ambler, “Agile Database Techniques”, Wiley Publishing, 2003
- Online:
  - http://www.agiledata.org/essays/databaseRefactoring.html
  - http://www.agiledata.org/essays/agileDataModeling.html
  - http://www.databaserefactoring.com/