



PUCPR  
GRUPO MARISTA

---

Otimização de consultas e *refactoring*

# Otimização de consultas

Operações que envolvem filtros de registro são sujeitas a problemas de desempenho.

Além das técnicas de *refactoring*, que serão vistas adiante, as principais causas de problemas de desempenho são:

- Falta de índices.
- Comparação entre colunas com tipos de dado diferentes.
- Uso de funções para comparação.

# Falta de índices

Índices são como atalhos no emaranhado de dados, apontando caminhos curtos para localizar o registro buscado.

- Quando não há índice, é realizada a busca em todos os registros da tabela ou TABLESCAN, que é o pior tipo de busca possível.

O problema mais comum, relacionado a provavelmente 80% dos casos, é a falta de índice em FKs e PKs.

- Não é obrigatória a criação de índice; então, mesmo FKs e PKs precisam de uma declaração explícita para tanto.
  - Alguns SGBDs criam automaticamente, mas não é regra.

Colunas usadas na cláusula *where* ou em *joins* que não possuem índice.

- Os desenvolvedores da aplicação acabam usando as tabelas para geração de relatórios de uma forma não prevista originalmente e os problemas só são descobertos quando o volume de dados aumenta, em produção.

# Comparação entre colunas com tipos de dados diferentes

Tipos de dado diferentes em filtros provocam buscas em TABLESCAN, mesmo que as colunas envolvidas tenham índices.

- Por exemplo:
  - Na tabela ALUNO, o CPF é do tipo VARCHAR(11).
  - Na tabela MENSALIDADE, o CPF é do tipo DECIMAL(11,0).
  - A comparação WHERE ALUNO.CPF = MENSALIDADE.CPF será obrigatoriamente em TABLESCAN.
  - Com números de registro pequenos (< 1.000), a diferença é mínima.

# Uso de funções para comparação

O uso de funções na cláusula WHERE impede o emprego de índices, pois se está comparando o resultado da função e não o valor da coluna.

- Algumas versões de SGBD bem recentes permitem que sejam criados índices utilizando funções, ou seja, o resultado da função na coluna é indexado.

A consulta deve ser reescrita para não utilizar funções.

- Se realmente necessário, aplicam-se outros filtros antes para reduzir o número de registros a ser testados pela função.
- *Selects* aninhados podem ser uma alternativa.

# Notas

Estes três pontos devem ser verificados quando um comando apresenta problemas de desempenho:

- Existe índice em todas as PKs e FKs?
- As colunas comparadas são do mesmo tipo?
- Existem funções na comparação?

Embora pareçam triviais, eles resultam em ganhos maiores que 99%, reduzindo o tempo de execução do comando de horas para segundos.

# *Database Refactoring*

## Conceito

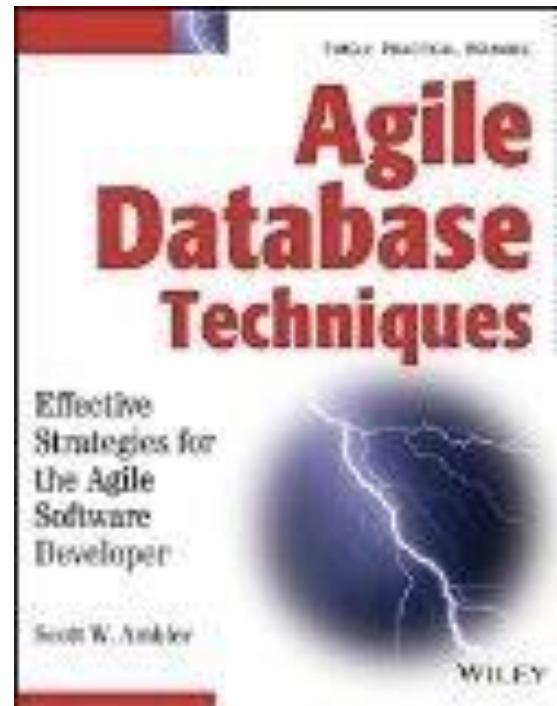
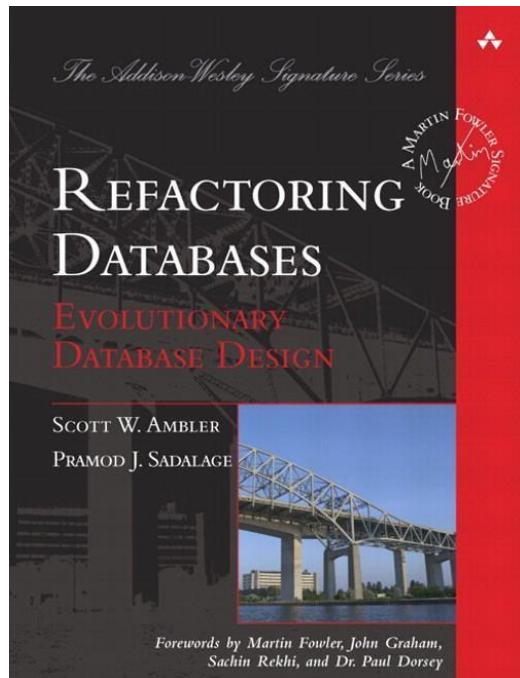
- Simples mudanças no banco de dados que melhoram o desempenho, mantendo a mesma semântica comportamental e informacional.

## Razões para refatorar

- Corrigir de forma segura bancos de dados legados existentes.
- Suportar evolução dos sistemas, mantendo a atualização tecnológica e de negócio.

# *Database Refactoring*

Literatura específica para *refactoring* de banco de dados



<http://agiledata.org/essays/databaseRefactoring.html>

<https://martinfowler.com/books/refactoringDatabases.html>

# Refatoração de banco de dados

## *Refactoring*

- Alteração no esquema do banco de dados que melhora o projeto, sem alterar a semântica funcional e não funcional.

<https://martinfowler.com/books/refactoringDatabases.html>

<http://agiledata.org/essays/databaseRefactoring.html>

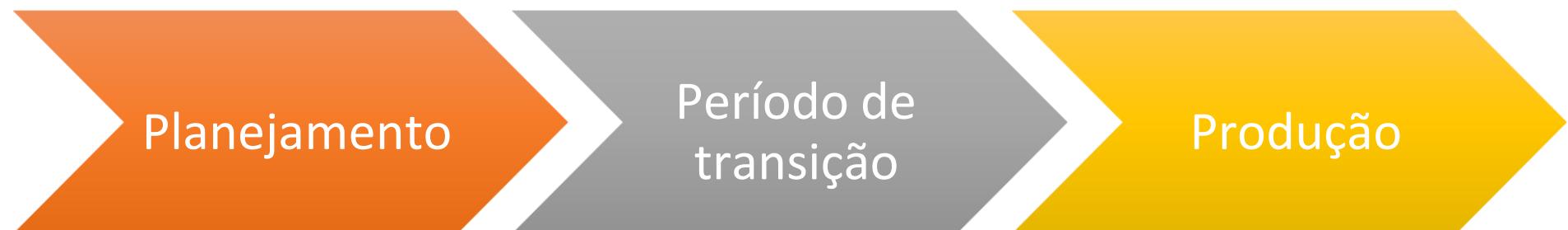
## Por que “refazer”?

- Para modificar de modo seguro bancos de dados legados.
- Para suportar desenvolvimento evolucionário.

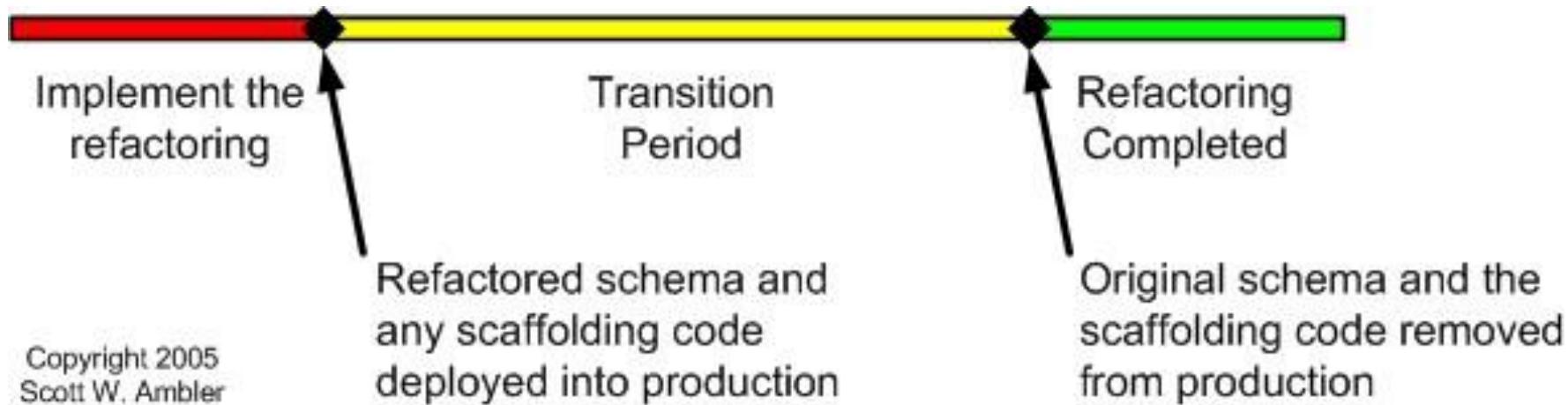
## Transformar

- Mudança na semântica do esquema do banco, adicionando elementos ou alterando atributos/tabelas existentes.
  - Não é *refactoring*!!!

# Processo de *refactoring*



# Período de transição



# Por que *database refactoring* é difícil?

## Acoplamento do banco de dados com:

Código-fonte da aplicação.

Código-fonte de outras aplicações.

Carga de dados (*script*).

Programas de extração de dados (XML, CSV).

*Frameworks* de persistência.

Codificação interna (*stored procedures, functions*).

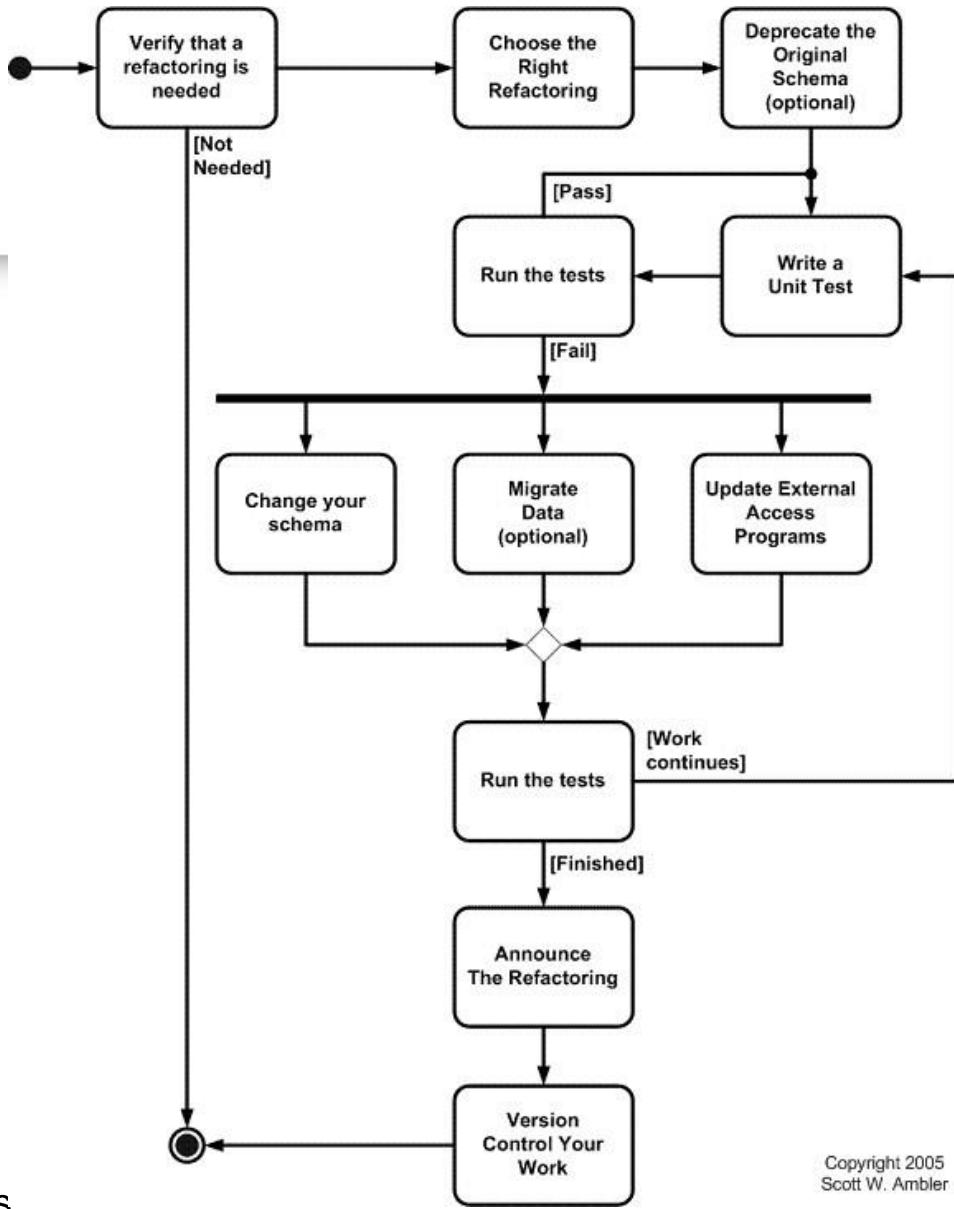
*Script* de migração de dados.

Código de teste.

Documentação (*update only when it hurts*).

# Processo de database refactoring

1. Descrever o cenário atual, justificando a necessidade de refatoração.
2. Escolher *refactoring* para aplicar.
3. Clonar esquema atual.
4. Identificar programas externos afetados.
5. Definir plano de teste.
6. Modificar esquema.
7. Executar a mudança.
8. Migrar dados (se necessário).
9. Alterar programas externos.
10. Testar.
11. Notificar equipe (se necessário).
12. Versionar artefatos (esquema, código, programas etc.).



# Passos do processo de *refactoring* – exemplo

1. Descrever o cenário atual, justificando a necessidade de refatoração.

Exemplo:

- As aplicações utilizam o CPF da tabela PENSIONISTA e, por ser campo numérico, os números não são armazenados, acusando CPF inválido.
- O CPF da tabela PESSOA permite caracteres além do permitido e possui dados inconsistentes. Existem números de CPF inválidos, com e sem formatação, e até *e-mail*.
- Necessidade: o atributo CPF das tabelas PESSOA e PENSIONISTA deve ser alterado para tipo caracter tamanho 11 e somente CPF válidos devem ser mantidos nas duas tabelas.

# Passos do processo de *refactoring* – exemplo

## 2. Escolher *refactoring* para aplicar.

Tipos de *refactoring* necessários:

- Estrutural.
- Qualidade de dados.

## 3. Clonar esquema atual.

### Pensionista

PensioCodnr: NUMBER(8)
ServidorMatriculanr: NUMBER(8) (FK)
PensioNomevc: VARCHAR2(60)
PensioDataNascimentodt: DATE
PensioNumCPFnr: NUMBER(11)
PensioIndSexoch: CHAR(1)
PensioEstCivilCodnr: NUMBER(1)
PensioNumFonenr: NUMBER(12)
BancoCodnr: NUMBER(3) (FK)
PensioNumAgenciavc: VARCHAR2(6)
PensioNumContavc: VARCHAR2(13)
PensioContaOperacaonr: NUMBER(3)
PensioLogradourov: VARCHAR2(40)
PensioBairrovc: VARCHAR2(25)
PensioMunicipiovc: VARCHAR2(30)
PensioUFch: CHAR(2)
PensioNumCEPnr: NUMBER(8)
OcorrenciaGruponr: NUMBER(2) (FK)
OcorrenciaNumnr: NUMBER(3) (FK)
PensioDataIniciodt: DATE
PensioDataFimdt: DATE
GrauParenteCodnr: NUMBER(8) (FK)
PensioNaturesaPensaoNr: NUMBER(1)

### PESSOA

PESSCODNR: NUMBER(8)
PESSDIGNR: NUMBER(1)
PESSNOMEVC: VARCHAR2(50)
PESSEXEOCH: CHAR(1)
PESSNASCDT: DATE
ESTCIVCODNR: NUMBER(2)
PAISCODNR: NUMBER(3)
PESSPAIVC: VARCHAR2(50)
PESSMAEVC: VARCHAR2(50)
PESSENDEREVC: VARCHAR2(100)
PESSCOMPLEMVC: VARCHAR2(20)
PESSBAIRROVC: VARCHAR2(50)
PESSCIDADEVC: VARCHAR2(60)
UFCODENDCH: CHAR(2)
PESSFONEVC: VARCHAR2(20)
PESSCEPVC: VARCHAR2(10)
PESSATIVINR: NUMBER(5)
PESSRGNVC: VARCHAR2(20)
PESSRGDT: DATE
PESSRGSIGLAVC: VARCHAR2(11)
UFCODROCH: CHAR(2)
PESSCPFNVC: VARCHAR2(19)
PESSRECNVC: VARCHAR2(13)
PESSRESDT: DATE
PESSRESCATVC: CHAR(2)
PESSRESREGVC: CHAR(2)

# Passos do processo de *refactoring* – exemplo

## 4. Identificar programas externos afetados.

Listar programas.



# Passos do processo de *refactoring* – exemplo

## 5. Definir plano de teste.

- Selecionar CPFs inválidos das duas tabelas, executando função ValidaCPF().
- Remover caracteres não numéricos e espaços da tabela PESSOA.
- Atualizar CPFs inválidos para *null* depois do esquema alterado.
- Validar CPFs nas duas tabelas, usando a função validaCPF().
- Sucesso: somente CPFs válidos ou nulos nas duas tabelas.

# Passos do processo de *refactoring* – exemplo

6. Modificar esquema.
7. Executar a mudança.
8. Migrar dados (se necessário).

- Fase de transição
  - Criar outro campo nas tabelas PESSOA e PENSIONISTA.
  - Fazer a atualização do campo antigo para o campo novo somente dos CPFs válidos, por meio da função ValidaCPF().
  - Excluir atributo antigo das duas tabelas.

## Passos do processo de *refactoring* – exemplo

9. Alterar programas externos.
10. Testar.
11. Notificar equipe (se necessário).
12. Versionar artefatos (esquema, código, programas etc.).

# Tipos de *refactoring* em banco de dados

## Estrutural

- Mudança na estrutura do esquema do banco de dados.

## Qualidade de dados

- Mudança que melhora a consistência e o uso dos valores armazenados.

## Integridade referencial

- Mudança que garante que a tupla referenciada exista em outra tabela e/ou que a tupla desnecessária seja removida apropriadamente.

## Arquitetural

- Mudança que melhora de maneira geral a forma como os programas externos interagem com o banco de dados.

## Desempenho

- Mudança que melhora a qualidade dos dados de procedimentos, funções e gatilhos armazenados.

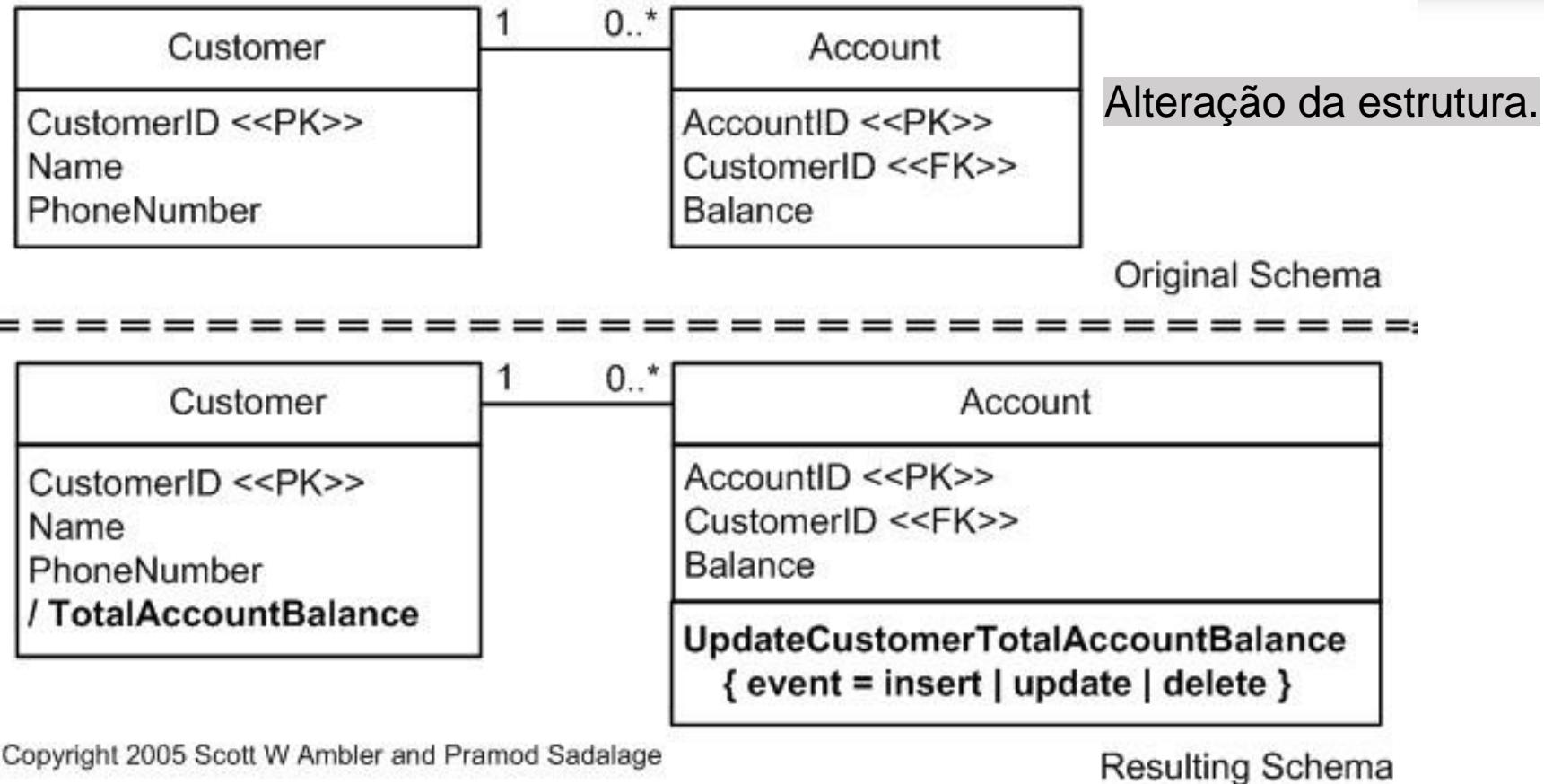
# Refatoração estrutural

Modificações no esquema do banco de dados:

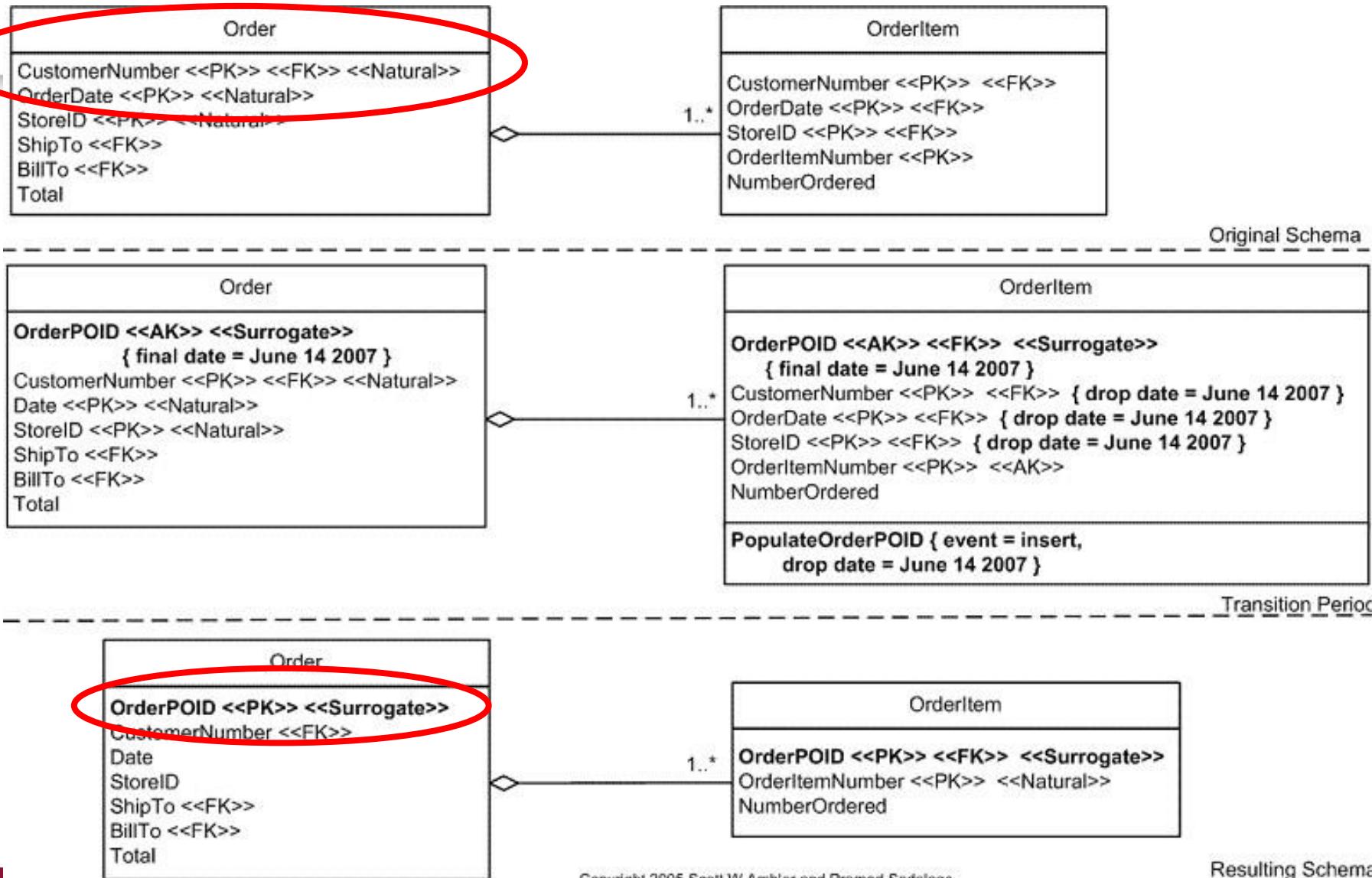
- Remover coluna, tabela ou *view*.
- Incluir coluna calculada, chave artificial.
- Mesclar colunas, tabelas.
- Mover ou renomear colunas, tabelas ou *views*.
- Substituir LOB por tabela.
- Substituir colunas.
- Substituir relacionamento 1:n por tabela associativa.
- Substituir chave artificial por chave natural.
- Dividir coluna ou tabela.

<http://www.agiledata.org/essays/databaseRefactoringCatalogStructural.html>

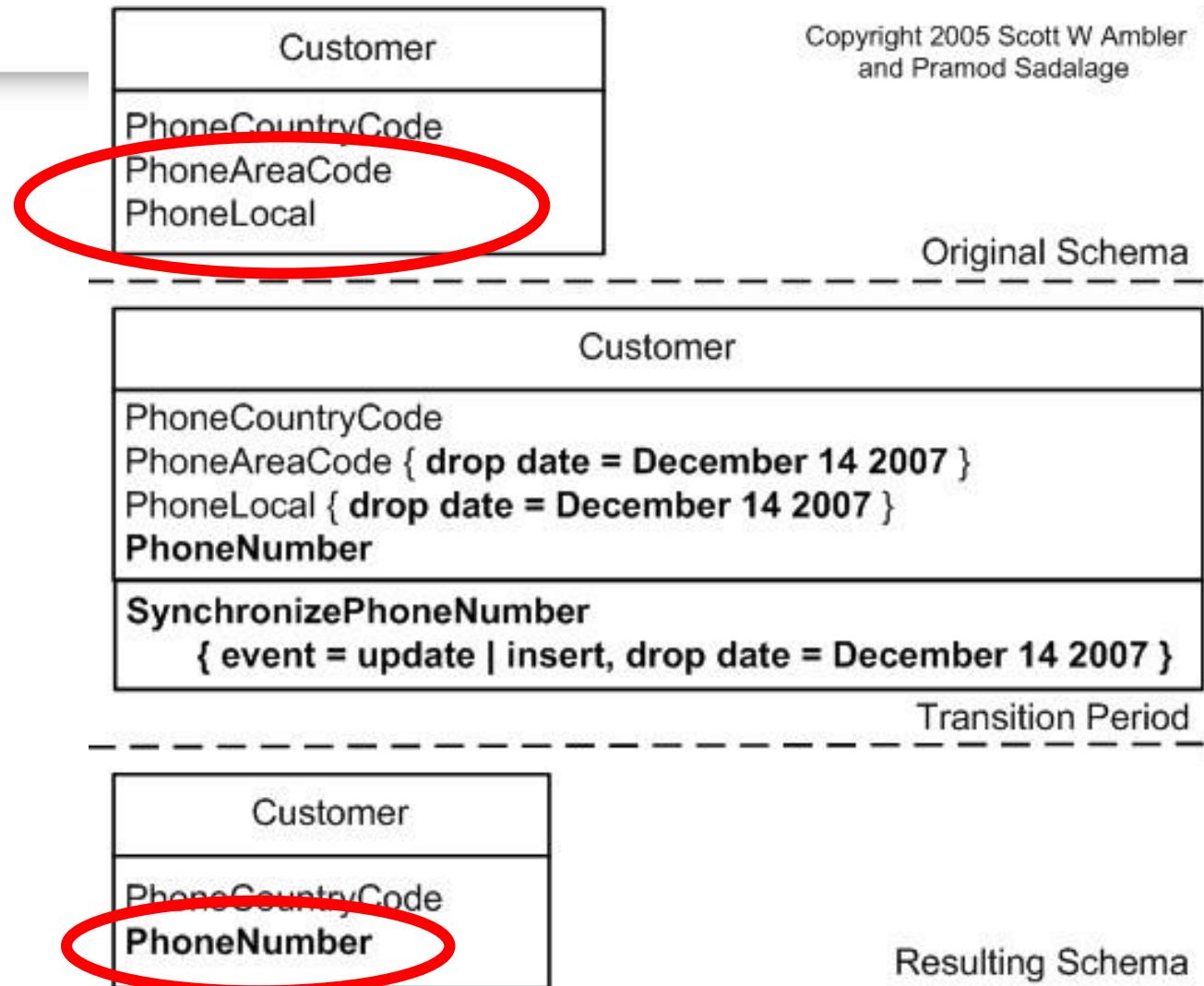
# Incluir coluna calculada



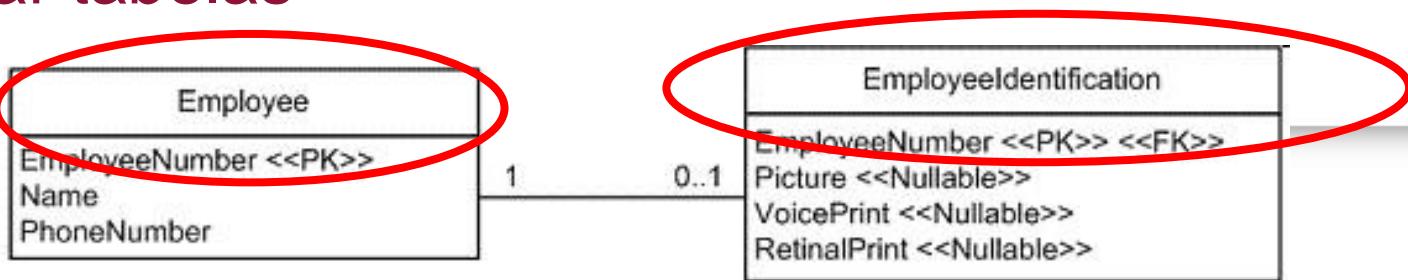
# Incluir chave artificial



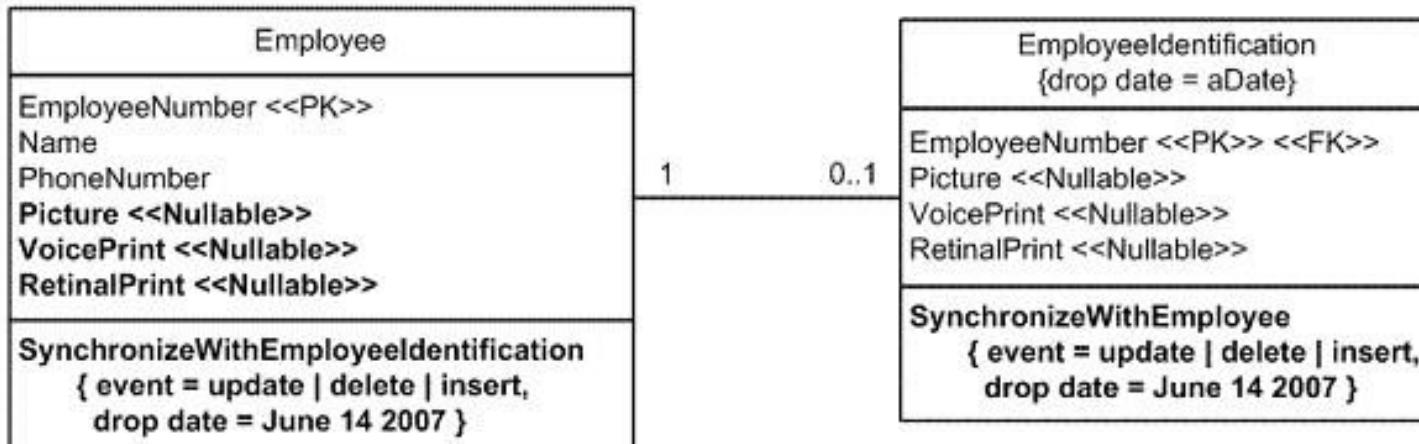
# Mesclar colunas



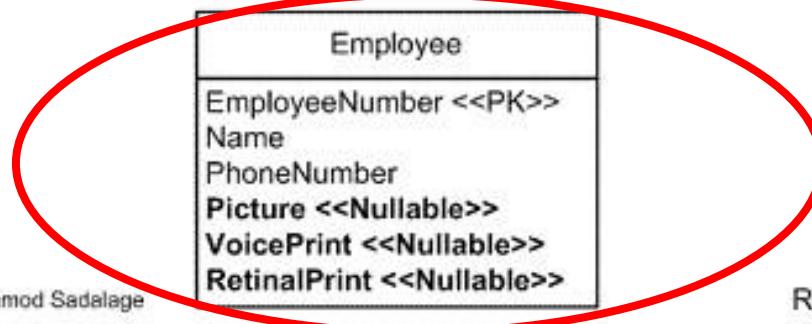
# Mesclar tabelas



Original Schema



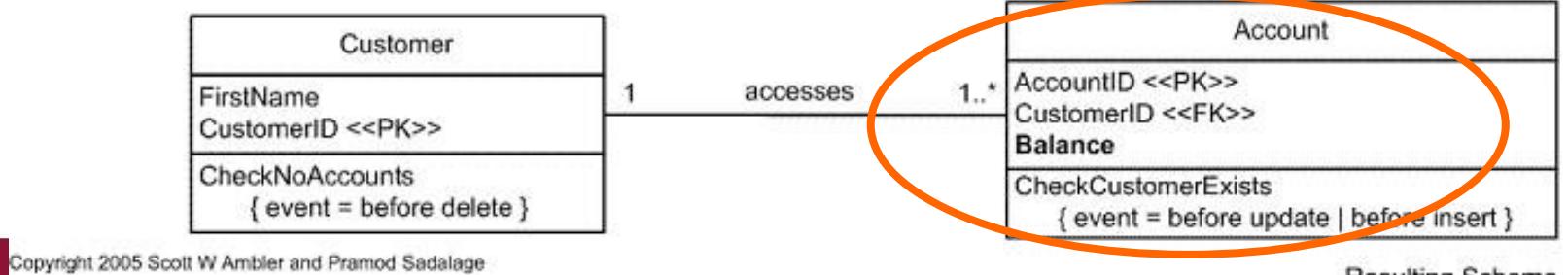
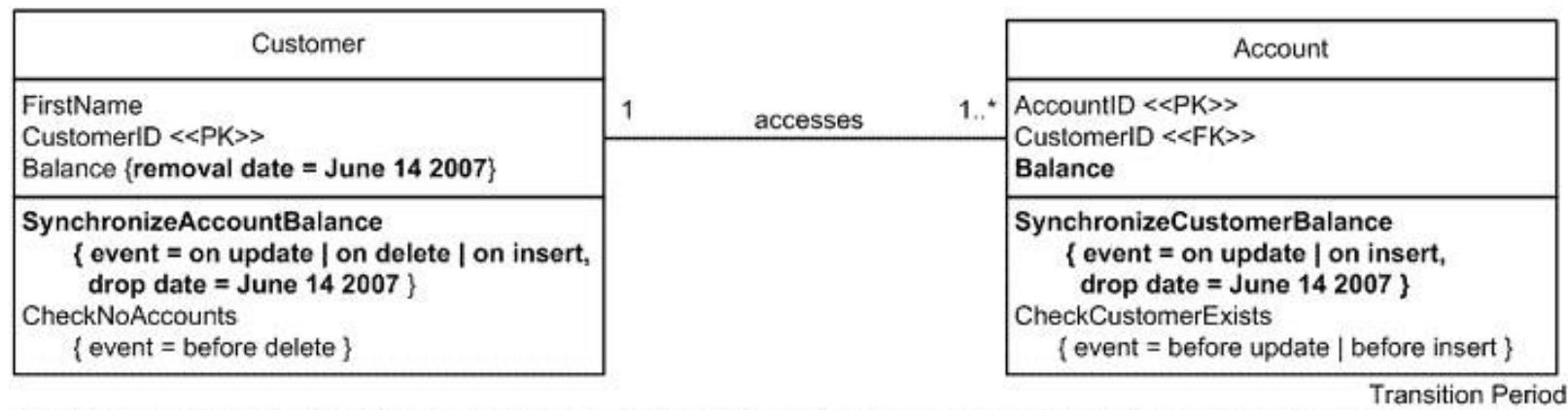
Transition Period



Resulting Schema

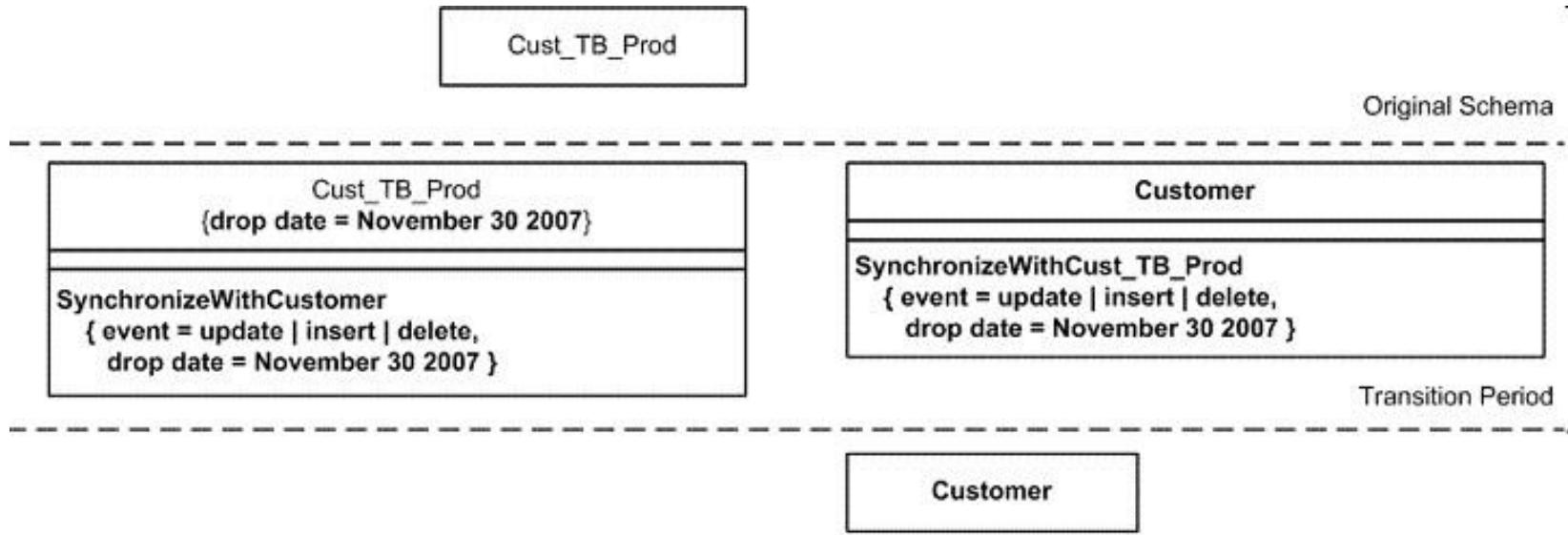
# Mover coluna

Migrar a coluna de uma tabela, com seus respectivos dados, para outra.



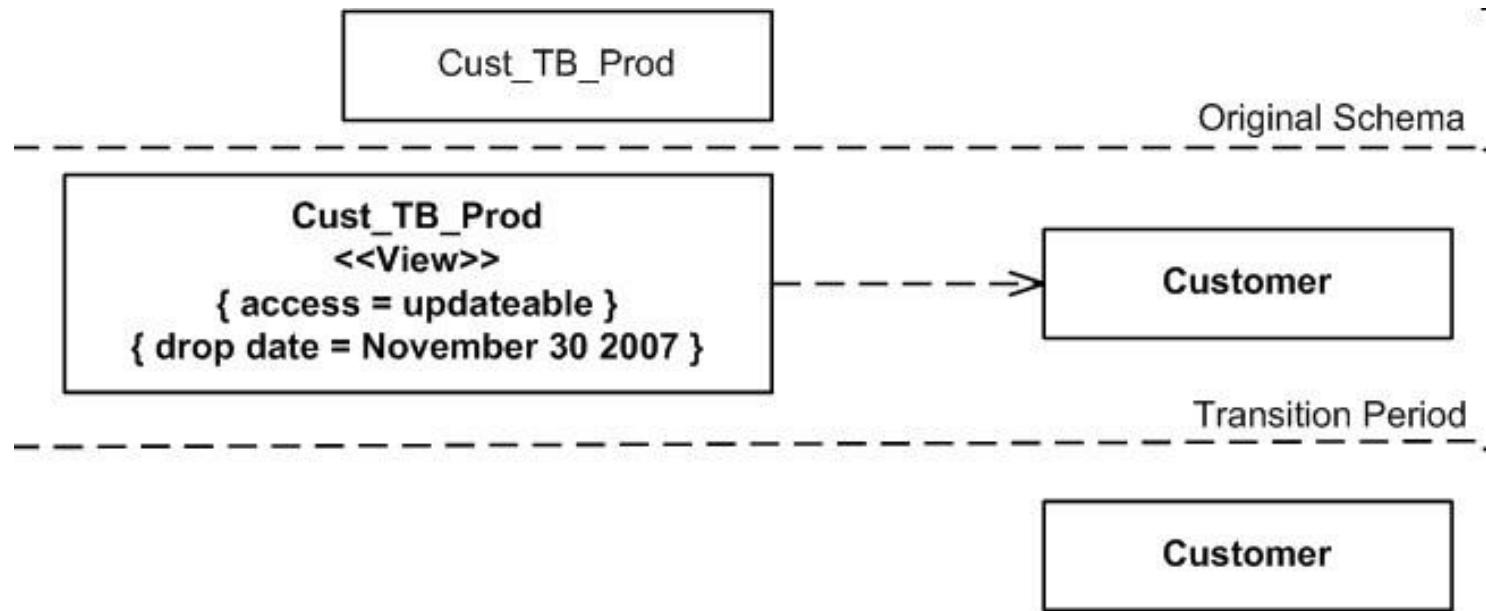
# Renomear tabela

Alterar diretamente o nome da tabela.



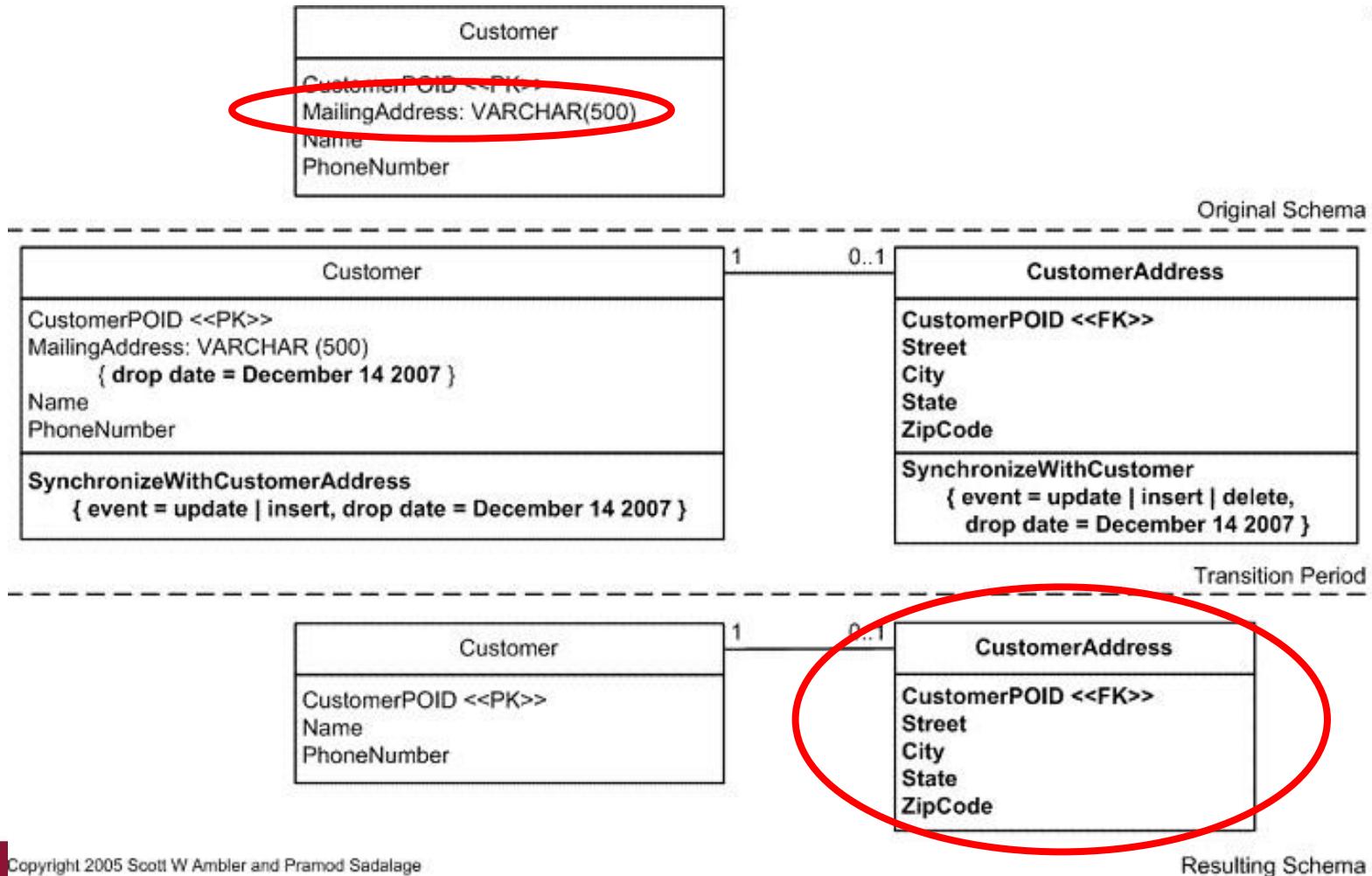
# Renomear tabela usando view

Criar uma *view*, em vez de alterar o nome da tabela.



# Substituir LOB por tabela

Substituir um *large object* (LOB) que contém dados estruturados por uma tabela nova ou a mesma tabela.



# Dividir colunas

Customer
CustomerID
Name
PhoneNumber

Original Schema

Customer
CustomerID
Name { drop date = June 14 2007 }
FirstName
MiddleName
LastName
PhoneNumber
<b>SynchronizeCustomerName</b> { event = update   insert, drop date = June 14 2007 }

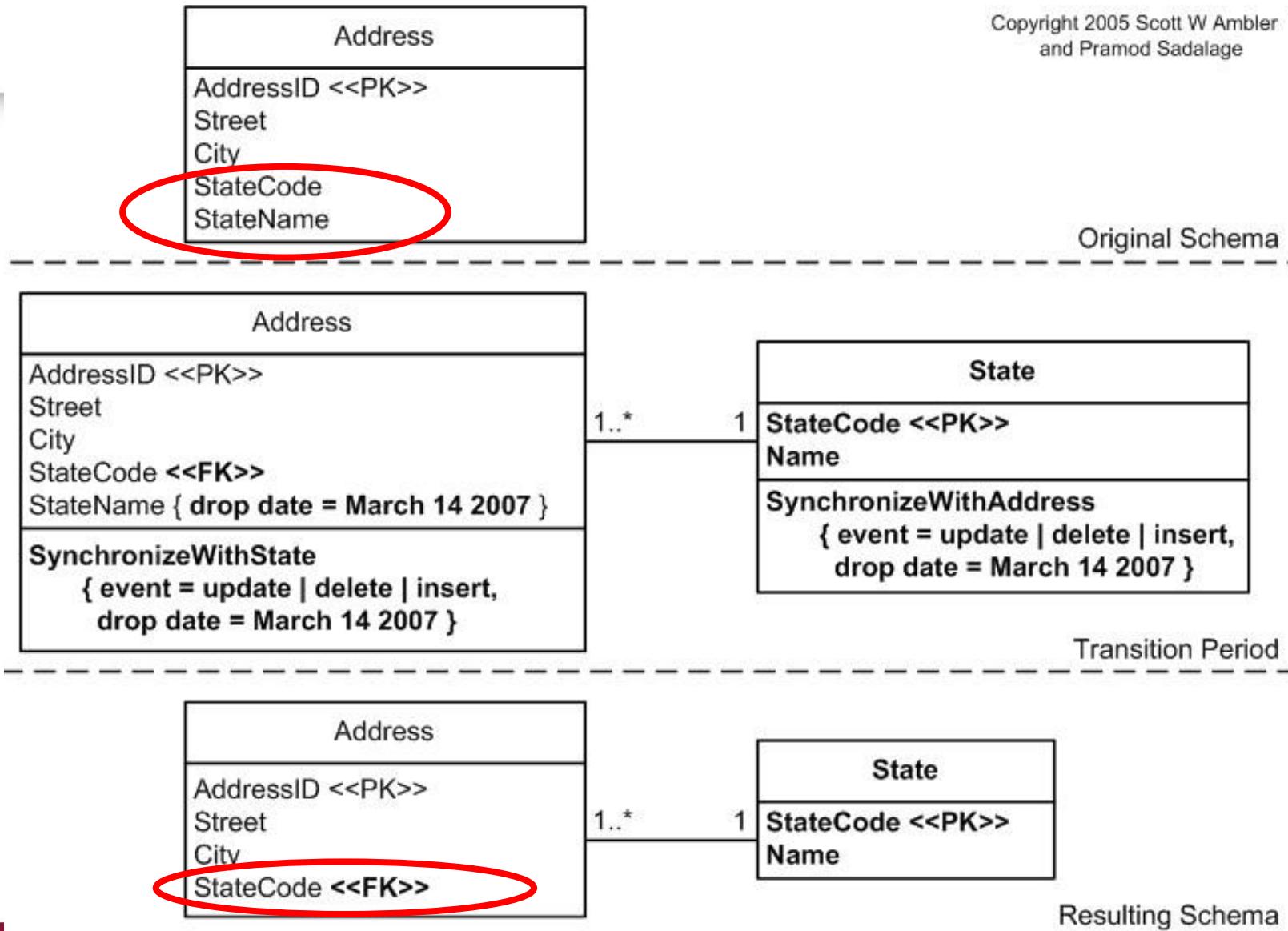
Transition Period

Customer
CustomerID
<b>FirstName</b>
<b>MiddleName</b>
<b>LastName</b>
PhoneNumber

Resulting Schema

# Dividir tabelas

Copyright 2005 Scott W Ambler  
and Pramod Sadalage



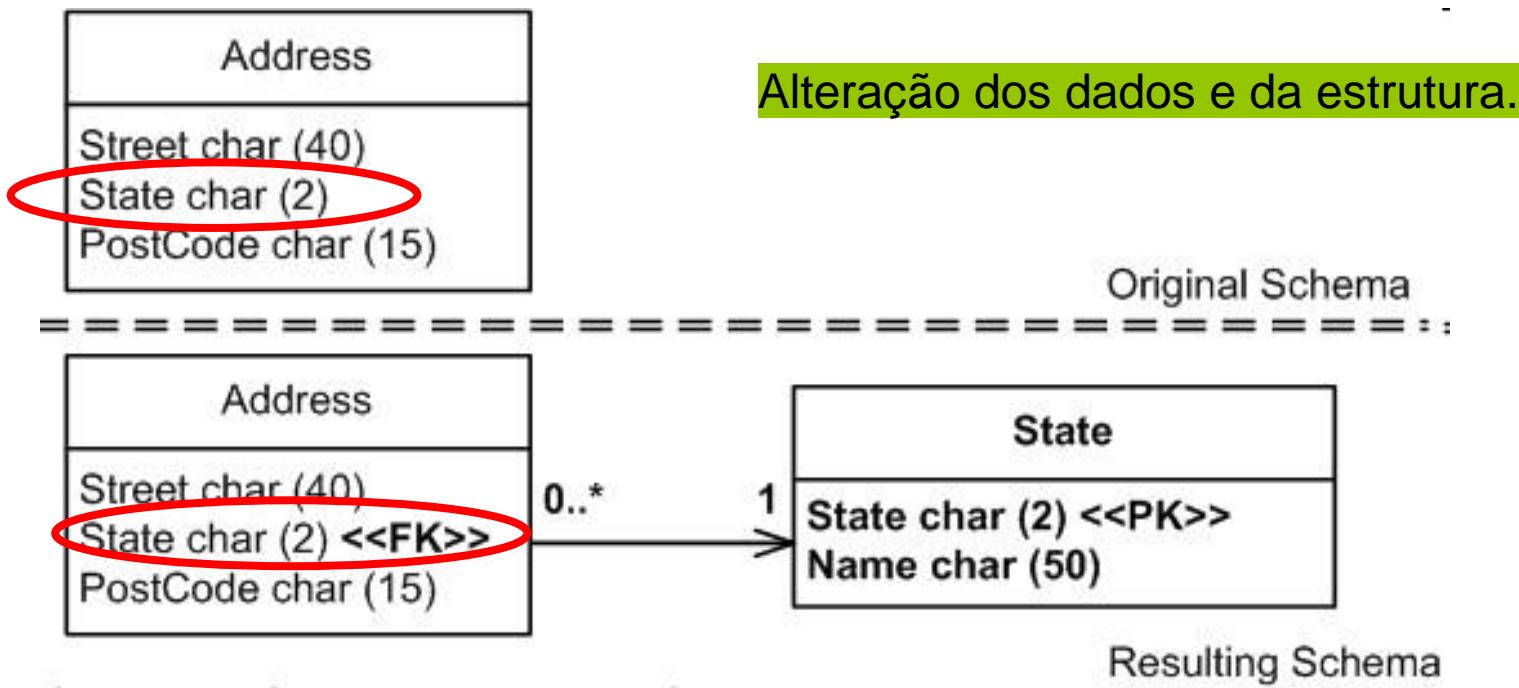
# Refatoração de qualidade de dados

Melhorias nos dados e na estrutura, visando a garantir a consistência e o uso dos valores armazenados:

- Adicionar tabelas auxiliares (*look up table*).
- Padronizar códigos e tipos.
- Consolidar estratégia de geração de chave.
- Remover/incluir restrição de coluna, valor *default* ou não nulo.
- Incluir formato comum.
- Mover dados.
- Substituir tipo código por *flag*.

<http://www.agiledata.org/essays/databaseRefactoringCatalogDataQuality.html>

# Adicionar tabelas auxiliares (look up table)



Copyright 2005 Scott W Ambler and Pramod Sadalage

# Padronizar códigos de dados

Alteração dos dados.

Before

Address			
Street	City	State	CountryCode
123 Main St.	Borington	ON	CAN
456 Elm St.	Hickton	CA	USA
4321 Oak Lane	New York	NY	US

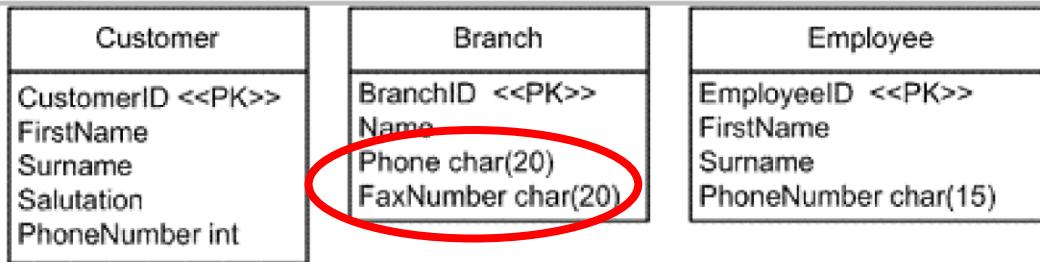
After

Address			
Street	City	State	CountryCode
123 Main St.	Borington	ON	CA
456 Elm St.	Hickton	CA	US
4321 Oak Lane	New York	NY	US

Country	
CountryCode	Name
CAN	Canada
USA	United States

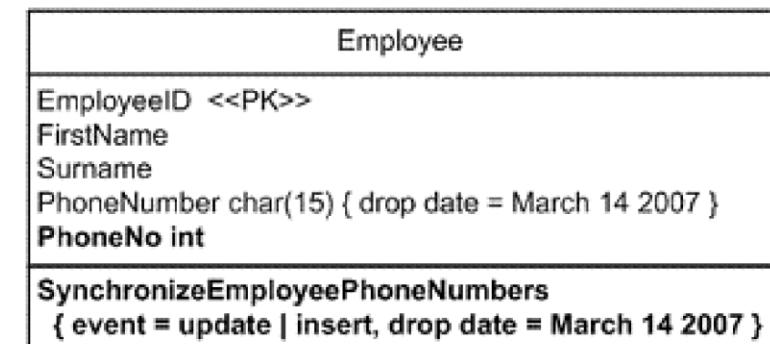
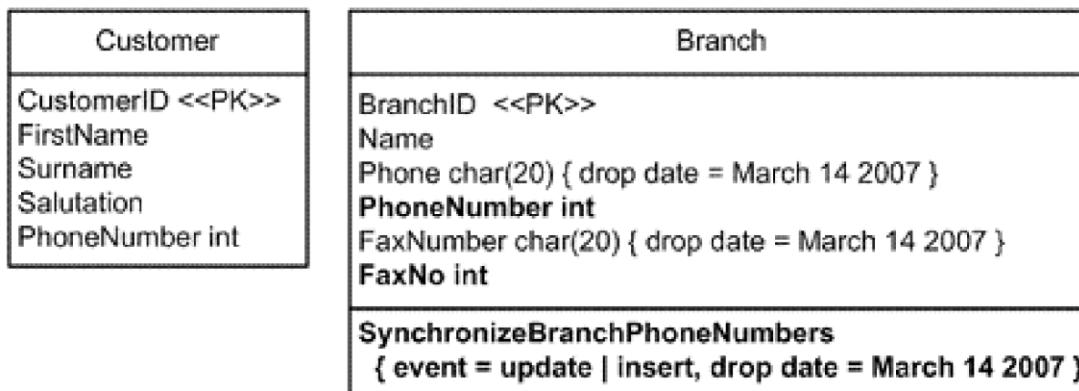
Country	
CountryCode	Name
CA	Canada
US	United States

# Padronizar tipos de dado

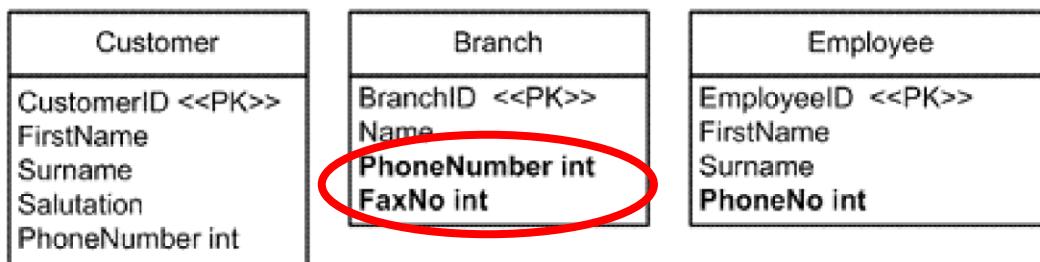


Alteração dos dados e da estrutura.

Original Schema



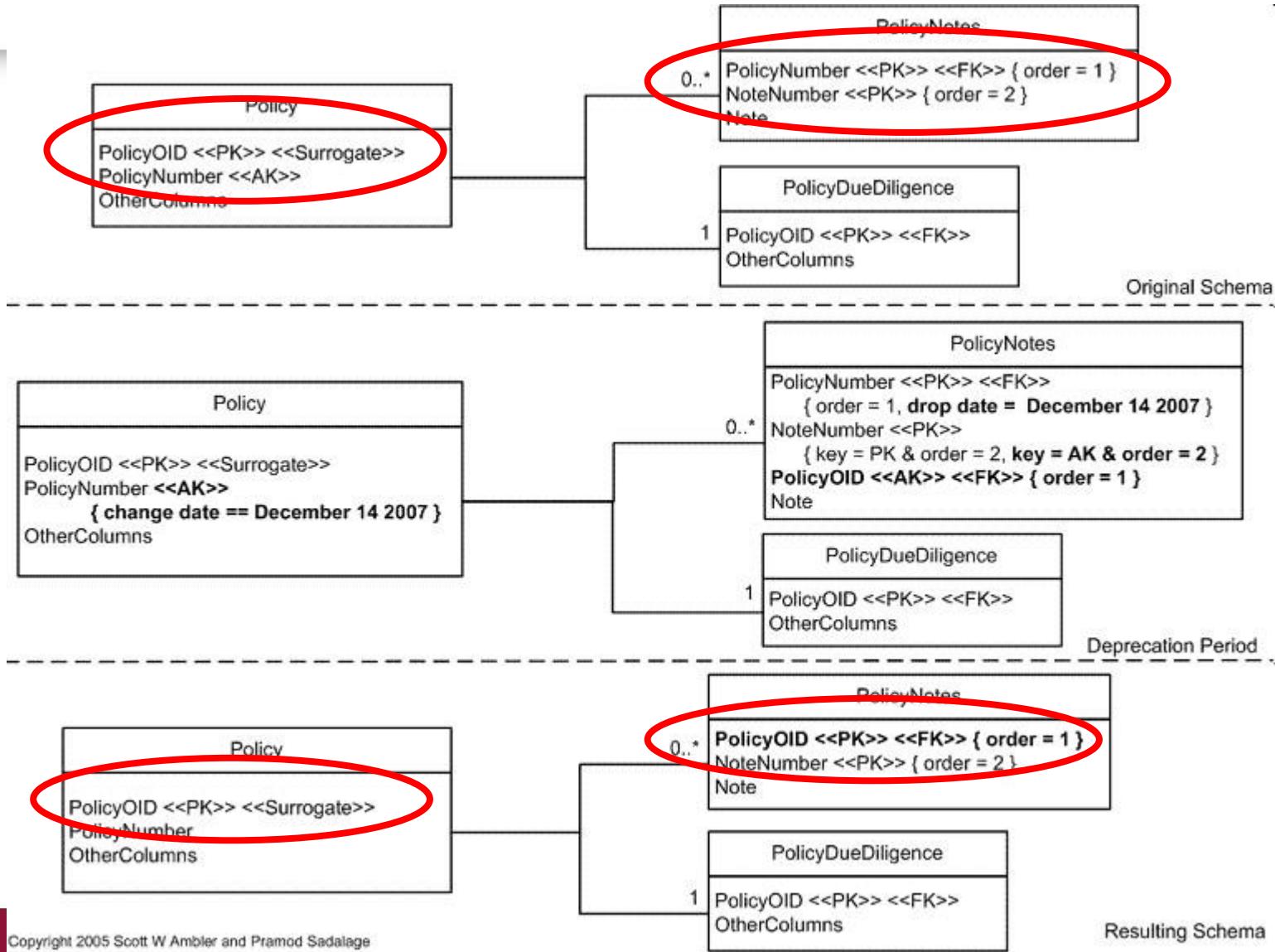
Transition Period



Garantir que o tipo da coluna seja consistente com o tipo de outras colunas similares no banco de dados.

Resulting Schema

# Consolidar estratégias de geração de chave



## Incluir formato comum

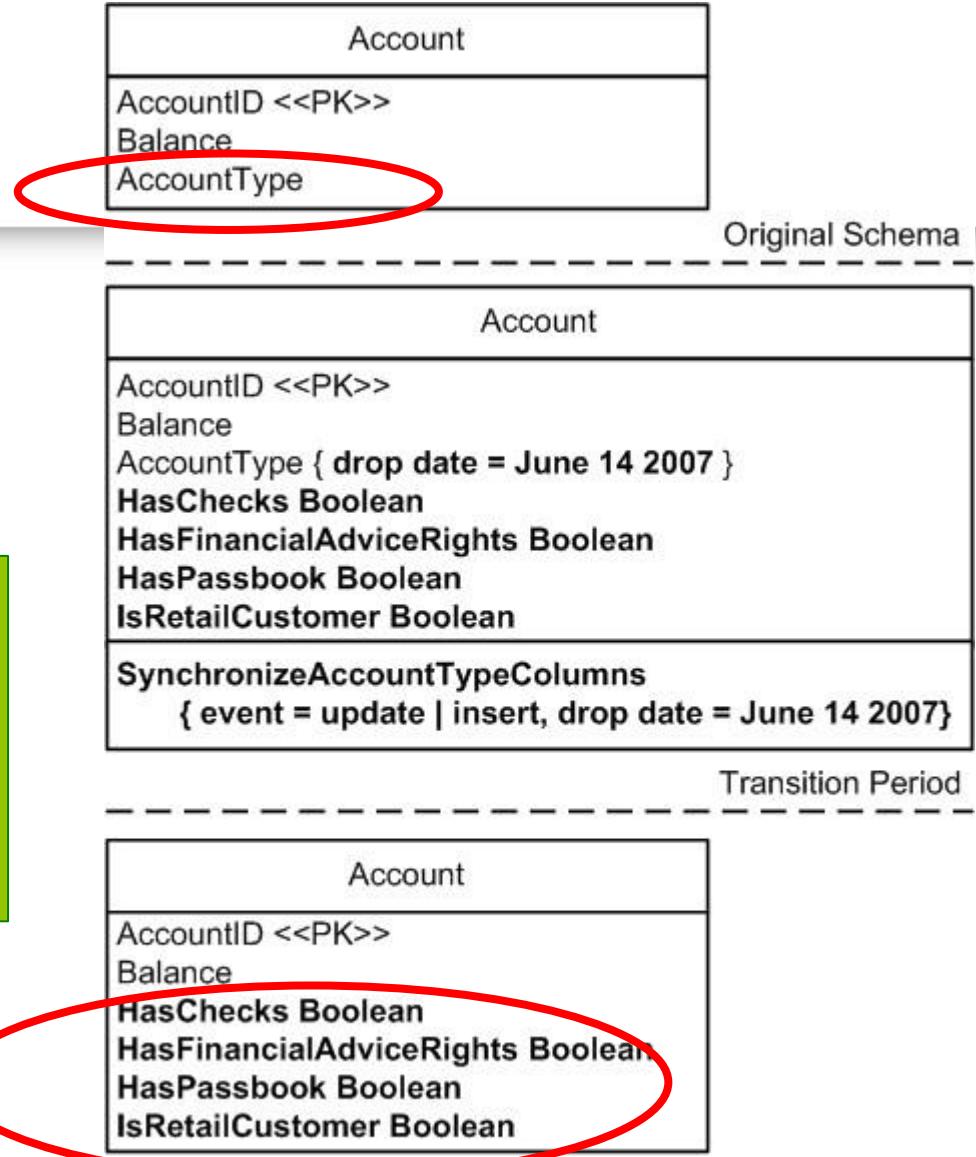
Alteração dos dados, por meio de programa específico para remover caracteres não numéricos.

Before	After
PhoneNumber	PhoneNumber
(416) 967-1111	<b>4169671111</b>
9055551212	<b>9055551212</b>
415.555.1234	<b>4155551234</b>
(416) 555 1234	<b>4165551234</b>
+1 905 234-5678	<b>9052345678</b>
4166546543	<b>4166546543</b>

Copyright 2005 Scott W Ambler and Pramod Sadalage

# Substituir tipo código por *flag*

Substituir um código com propriedade individual (tipo *flag*), geralmente implementado como colunas booleanas na mesma tabela.



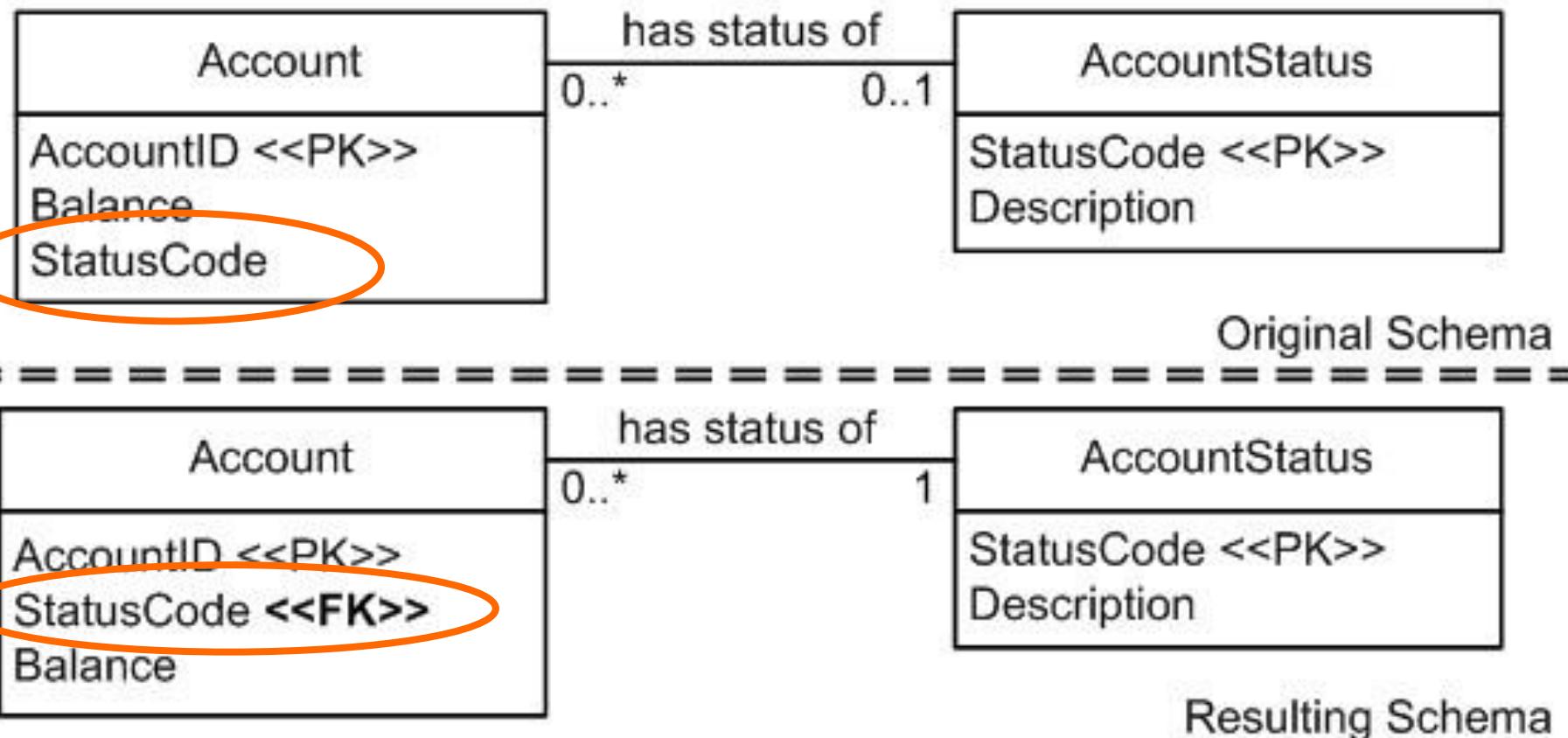
# Refatoração de integridade referencial

Mudanças que garantem que a tupla referenciada exista em outra tabela e/ou que a tupla desnecessária seja adequadamente removida:

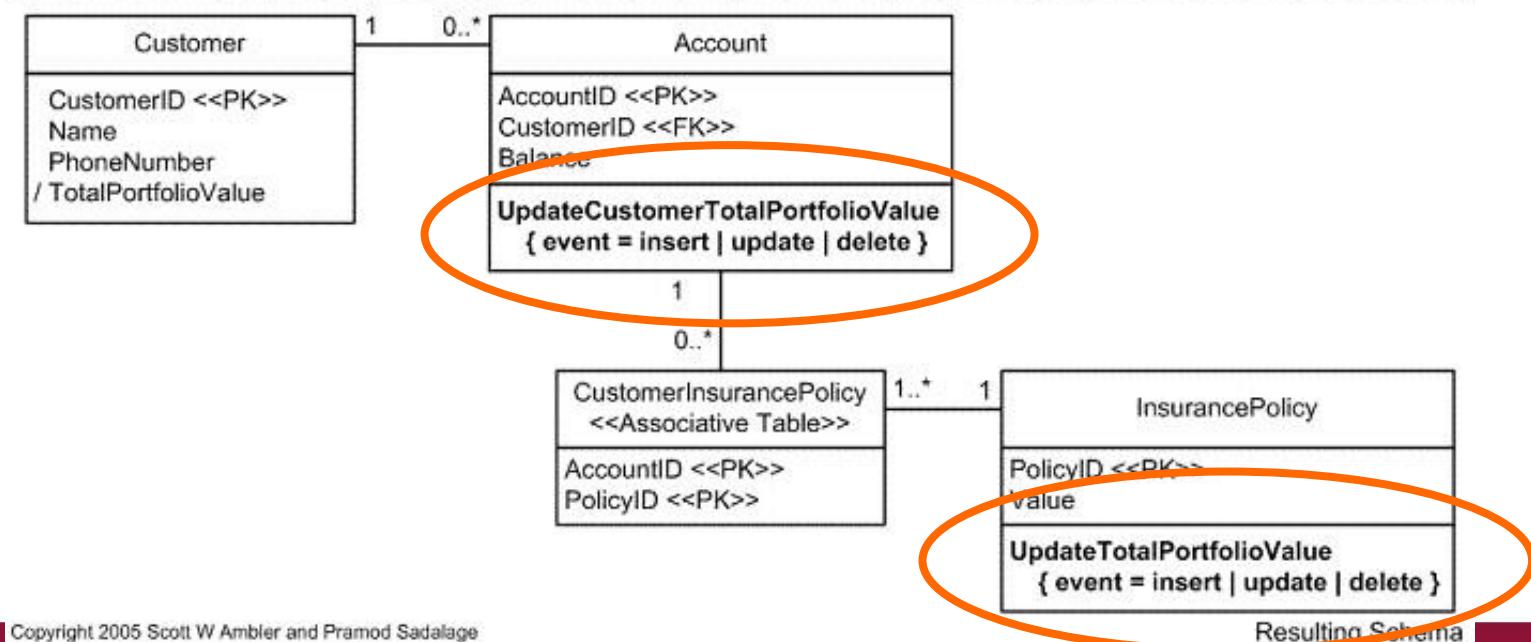
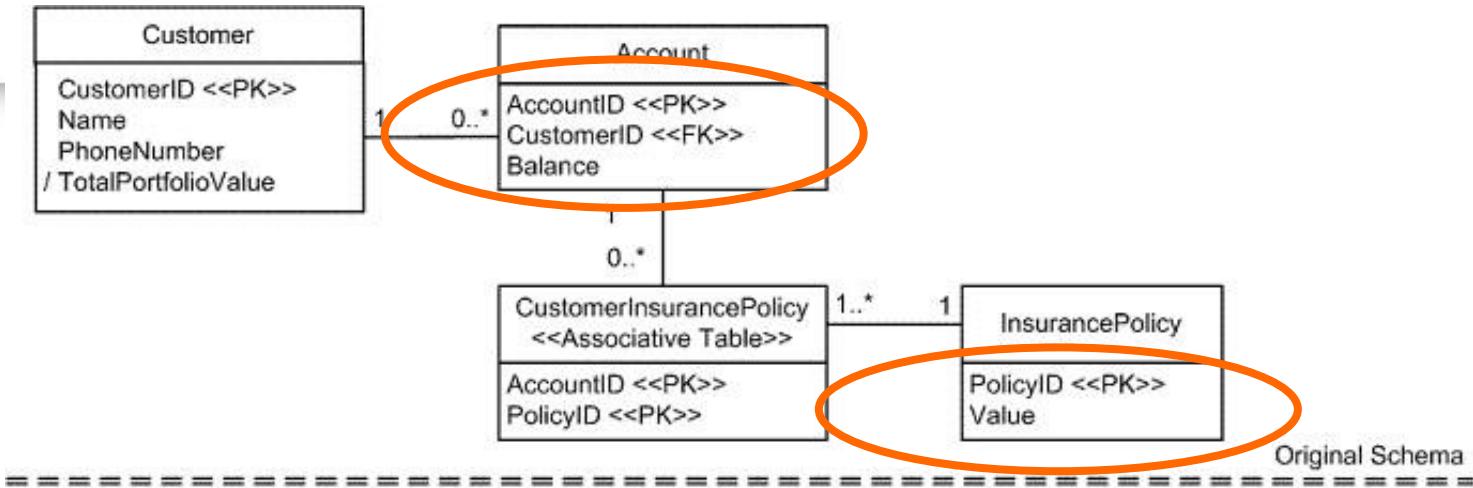
- Incluir restrição de chave estrangeira.
- Incluir *trigger* para coluna calculada.
- Remover restrição de chave estrangeira.
- Incluir exclusão em cascata, *hard/soft delete*.
- Incluir histórico gerado por *trigger*.

<http://www.agiledata.org/essays/databaseRefactoringReferentialIntegrity.html>

## Incluir restrição de chave estrangeira

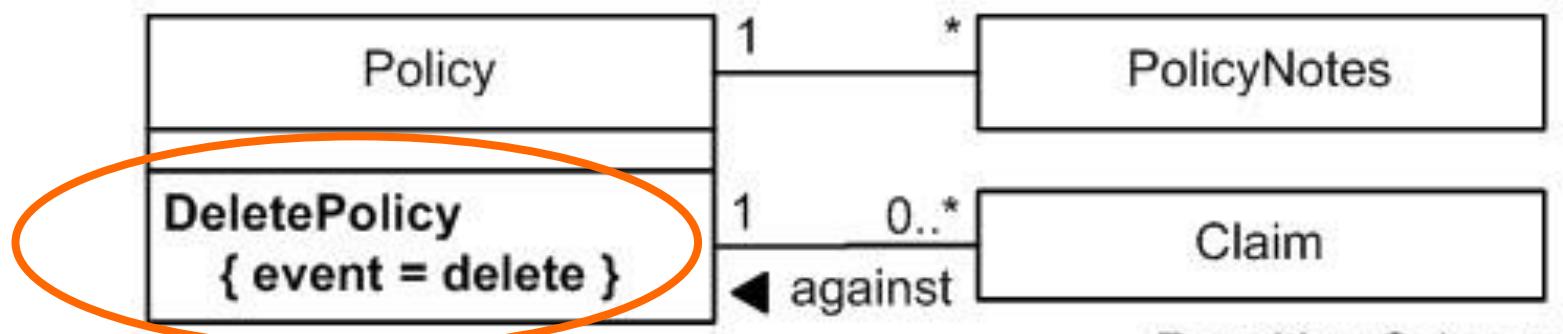
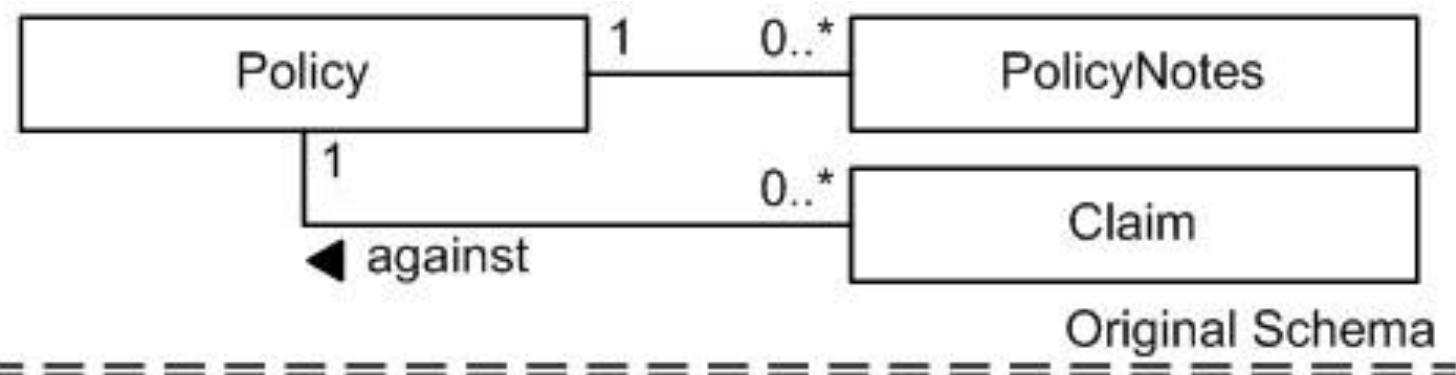


# Incluir trigger para coluna calculada

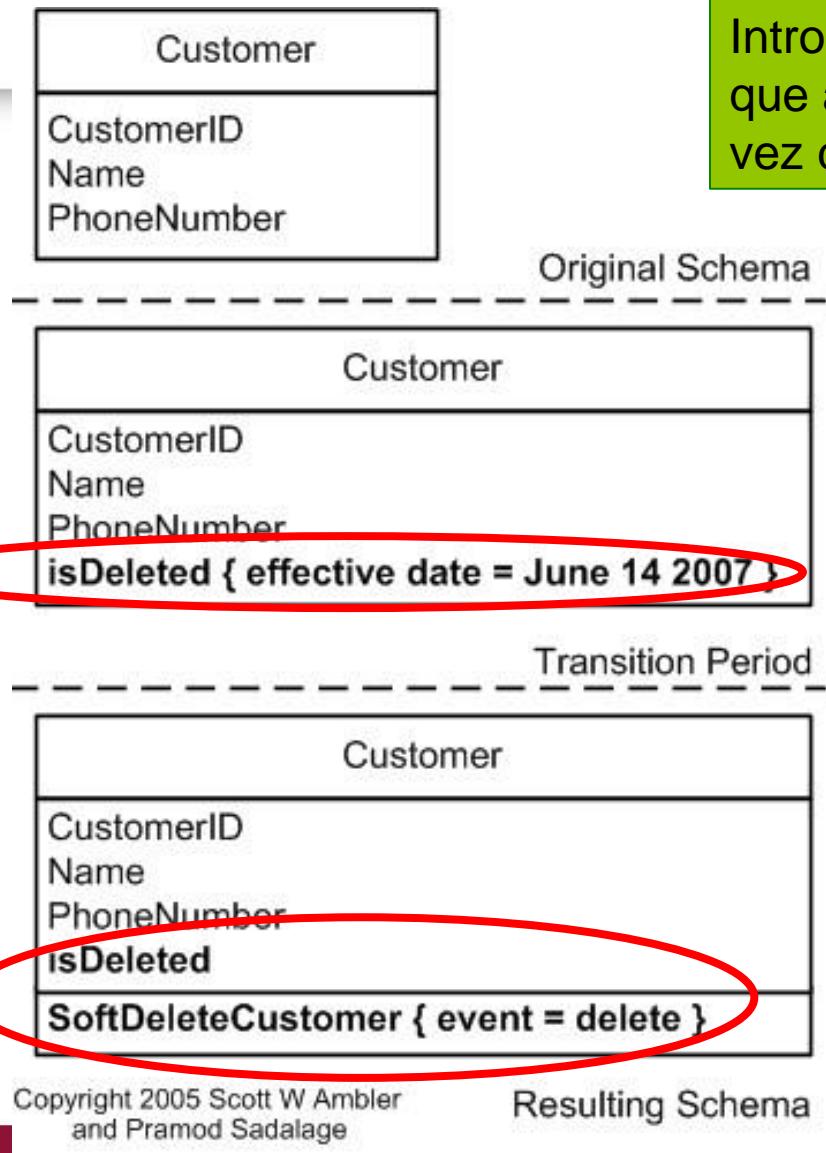


# Incluir exclusão em cascata

Garantir que o banco de dados exclua automaticamente os registros filhos apropriados quando um registro-pai é excluído.



# Incluir exclusão soft delete



## Incluir exclusão *hard delete*

Customer	
CustomerID	
Name	
PhoneNumber	
isDeleted	
SoftDeleteCustomer { event = delete }	

Remove uma coluna existente quando indicado que a tupla foi excluída.

Original Schema

Customer	
CustomerID	
Name	
PhoneNumber	
isDeleted { drop date = June 14 2007 }	
SoftDeleteCustomer { event = delete, drop date = June 14 2007 }	

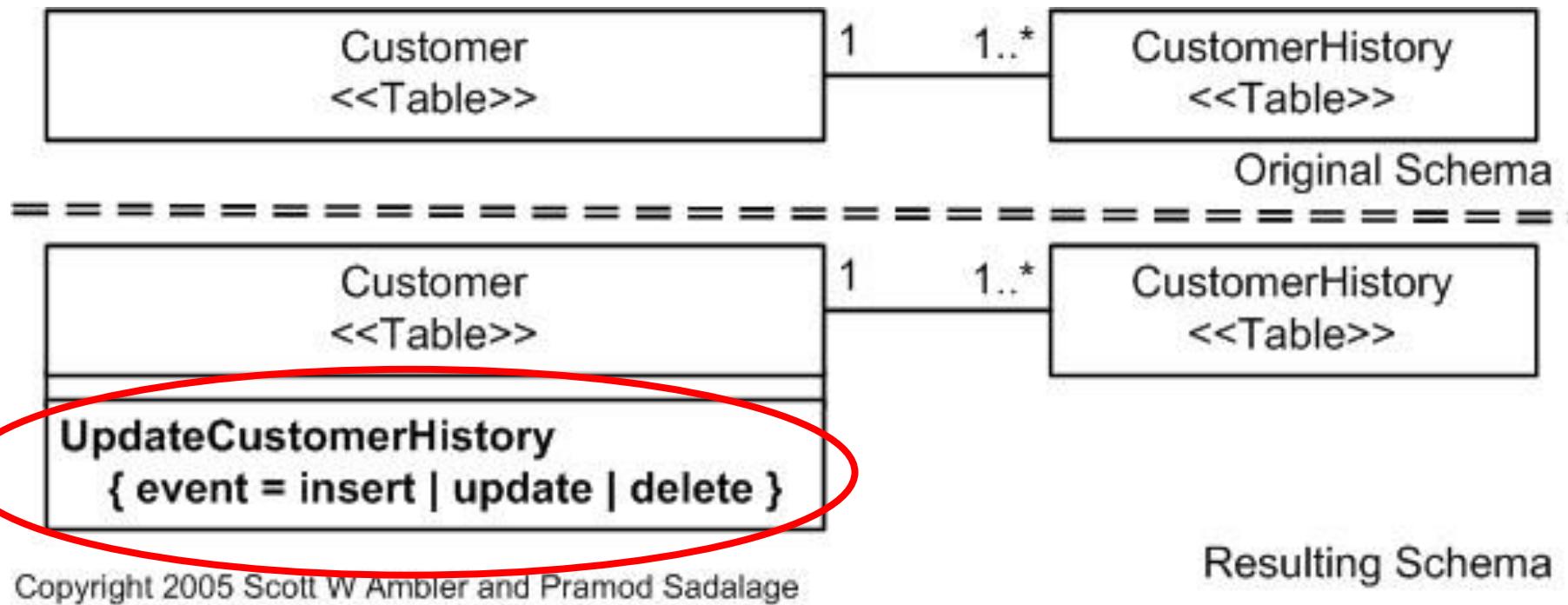
Transition Period

Customer	
CustomerID	
Name	
PhoneNumber	

Resulting Schema



# Incluir histórico gerado por trigger



Copyright 2005 Scott W Ambler and Pramod Sadalage

Resulting Schema

# Refatoração arquitetural

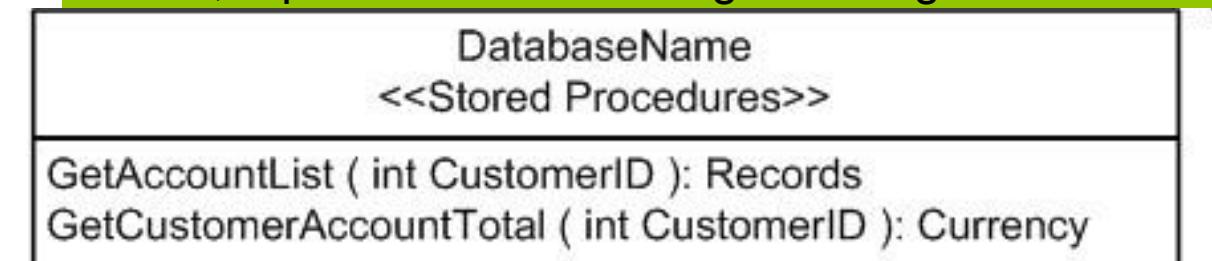
Mudanças que garantem melhorias de maneira geral na forma como programas externos interagem com o banco de dados:

- Adicionar métodos CRUD (*creation, retrieval, update, deletion*).
- Adicionar tabelas-espelho.
- Adicionar métodos de leitura de dados.
- Encapsular tabelas usando *views*.
- Introduzir cálculo por *stored procedure*.
- Introduzir índice/tabelas somente leitura.
- Migrar métodos do/para o banco de dados.

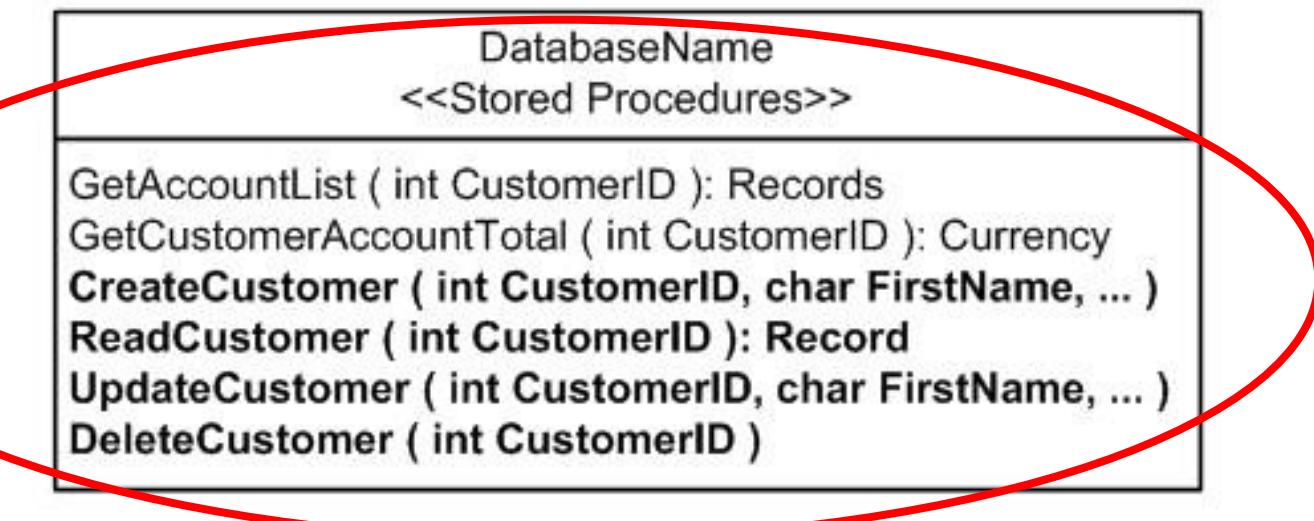
- Substituir método por *view*. • Substituir *view* por método.  
<http://www.agiledata.org/essays/databaseRefactoringCatalogArchitectural.html>

# Adicionar métodos CRUD

Introduzir *stored procedures* para implementar criação, recuperação, atualização e exclusão de dados, representando uma regra de negócio.

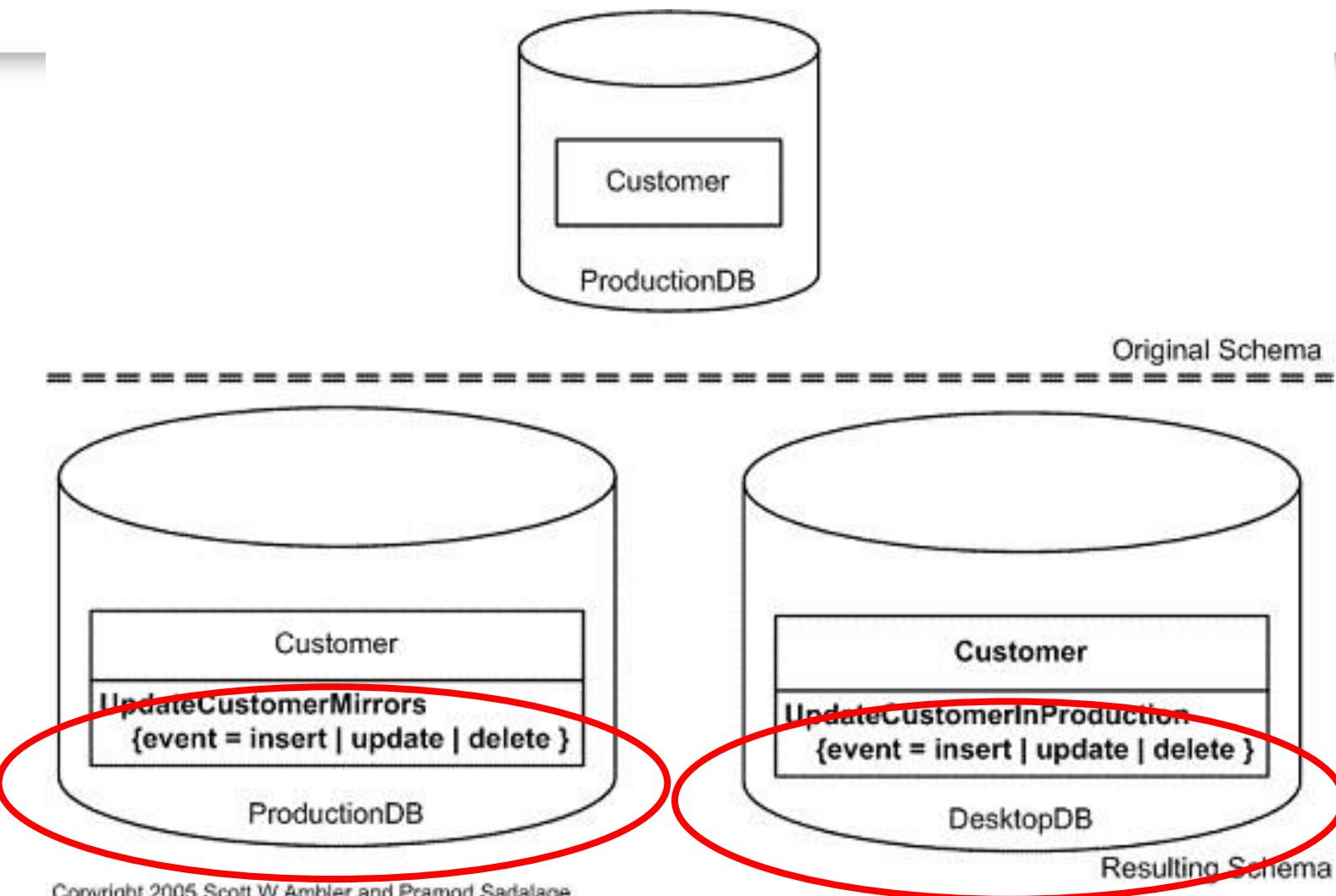


Original Schema



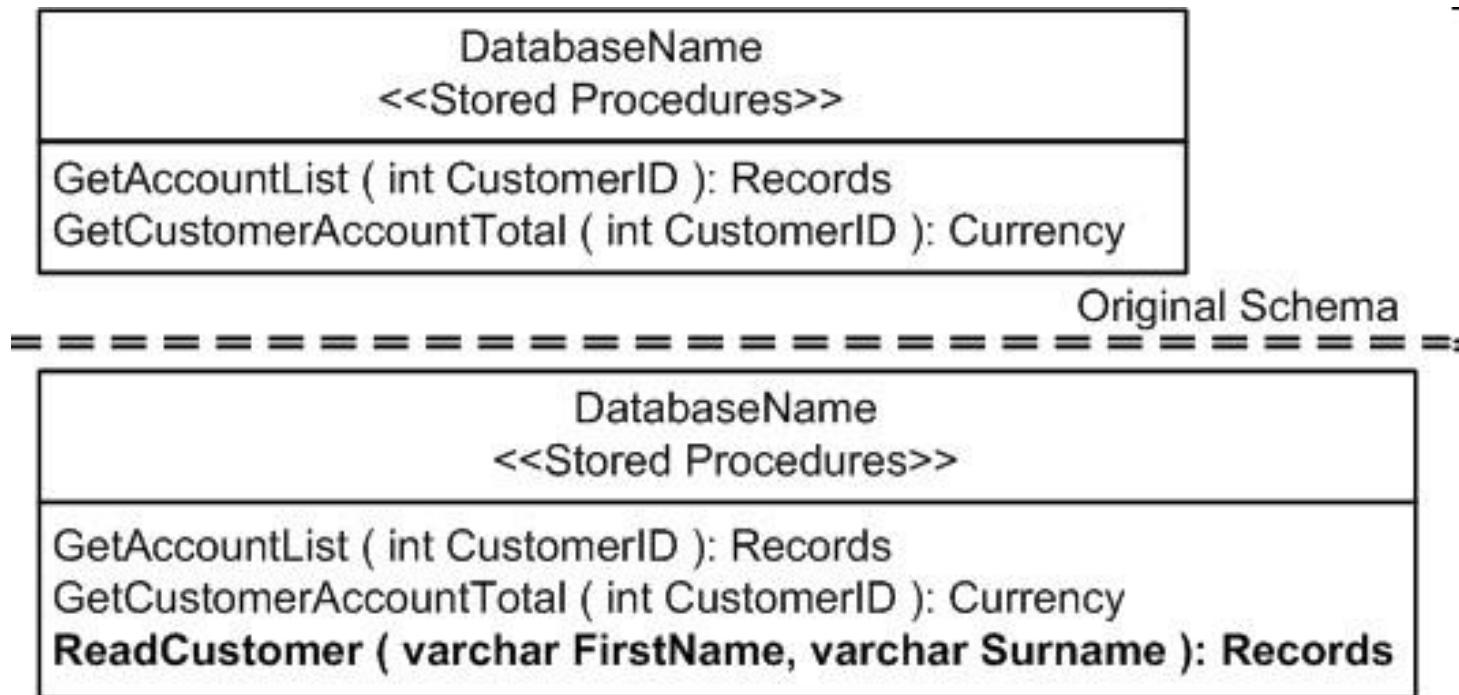
Resulting Schema

# Adicionar tabelas-espelho

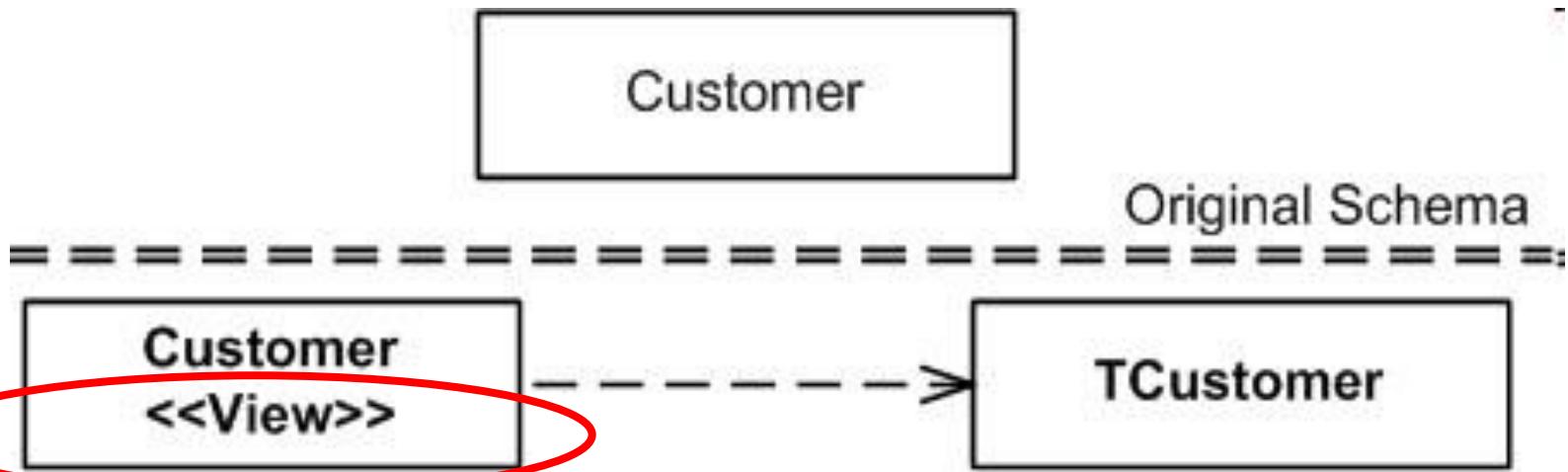


# Adicionar métodos de leitura de dados

Introduzir método para implementar a recuperação de dados que representam uma ou mais entidades de negócio armazenadas no banco de dados.

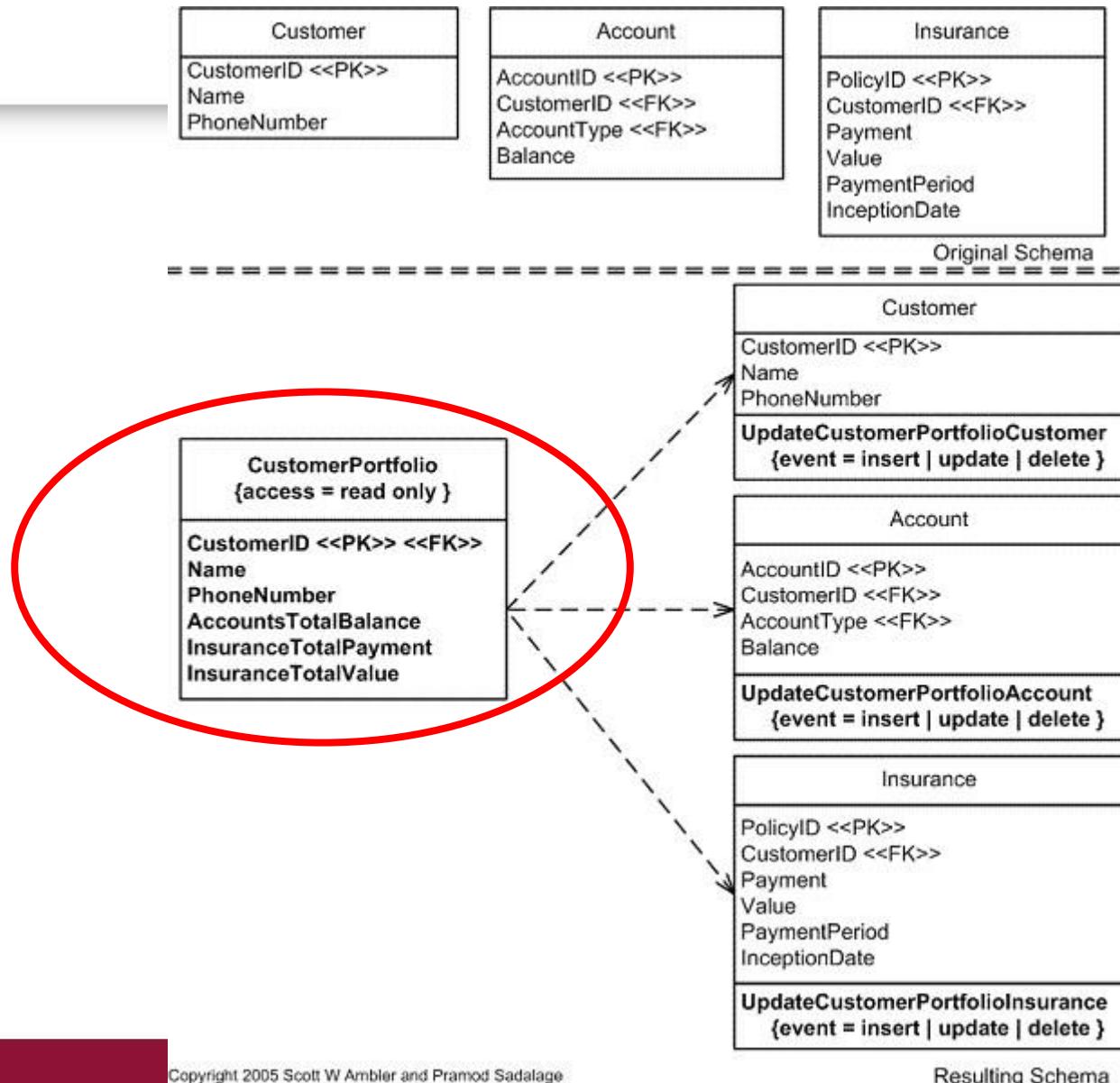


# Encapsular tabelas usando views



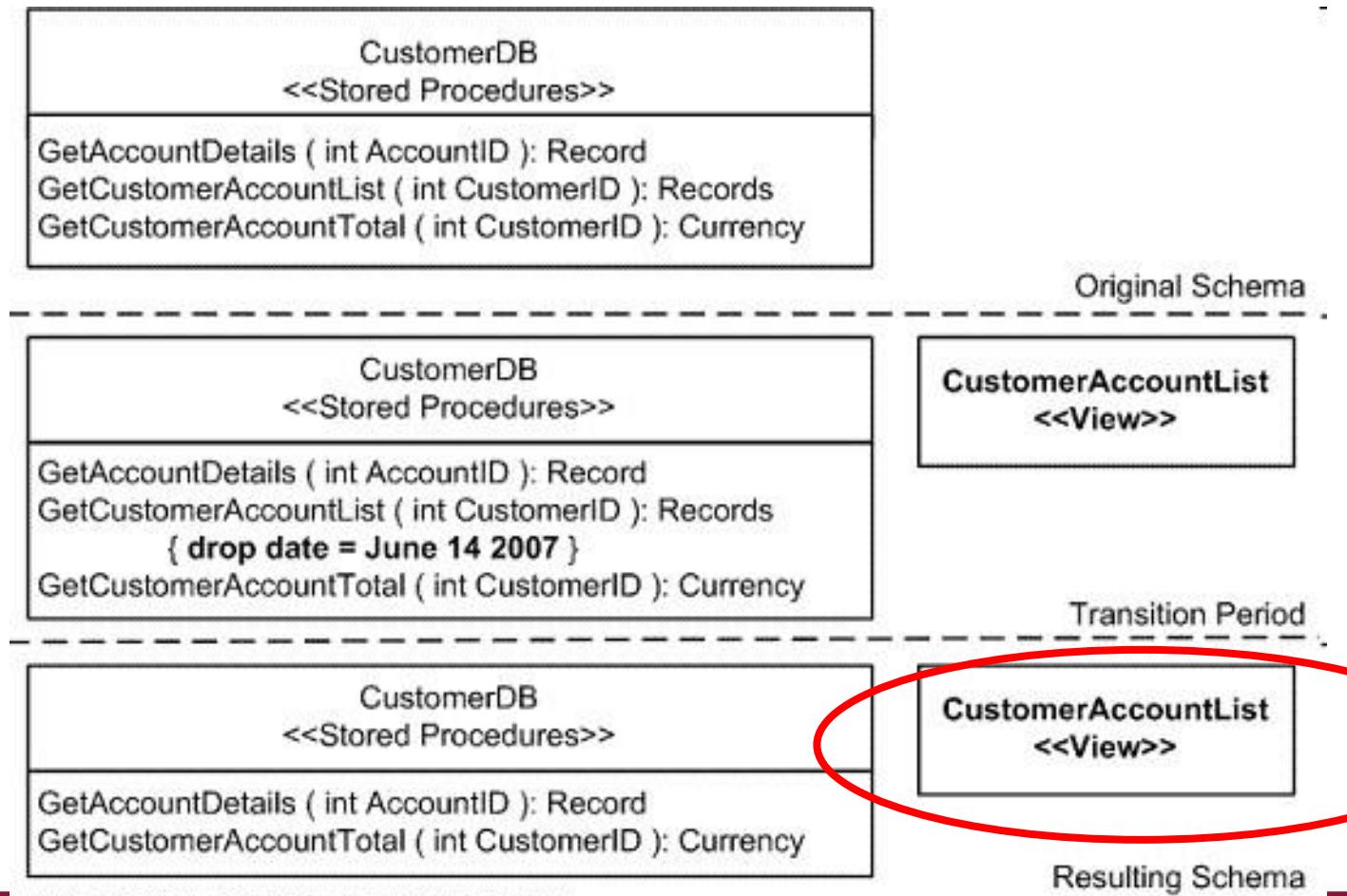
Copyright 2005 Scott W Ambler and Pramod Sadalage

# Introduzir tabelas somente leitura

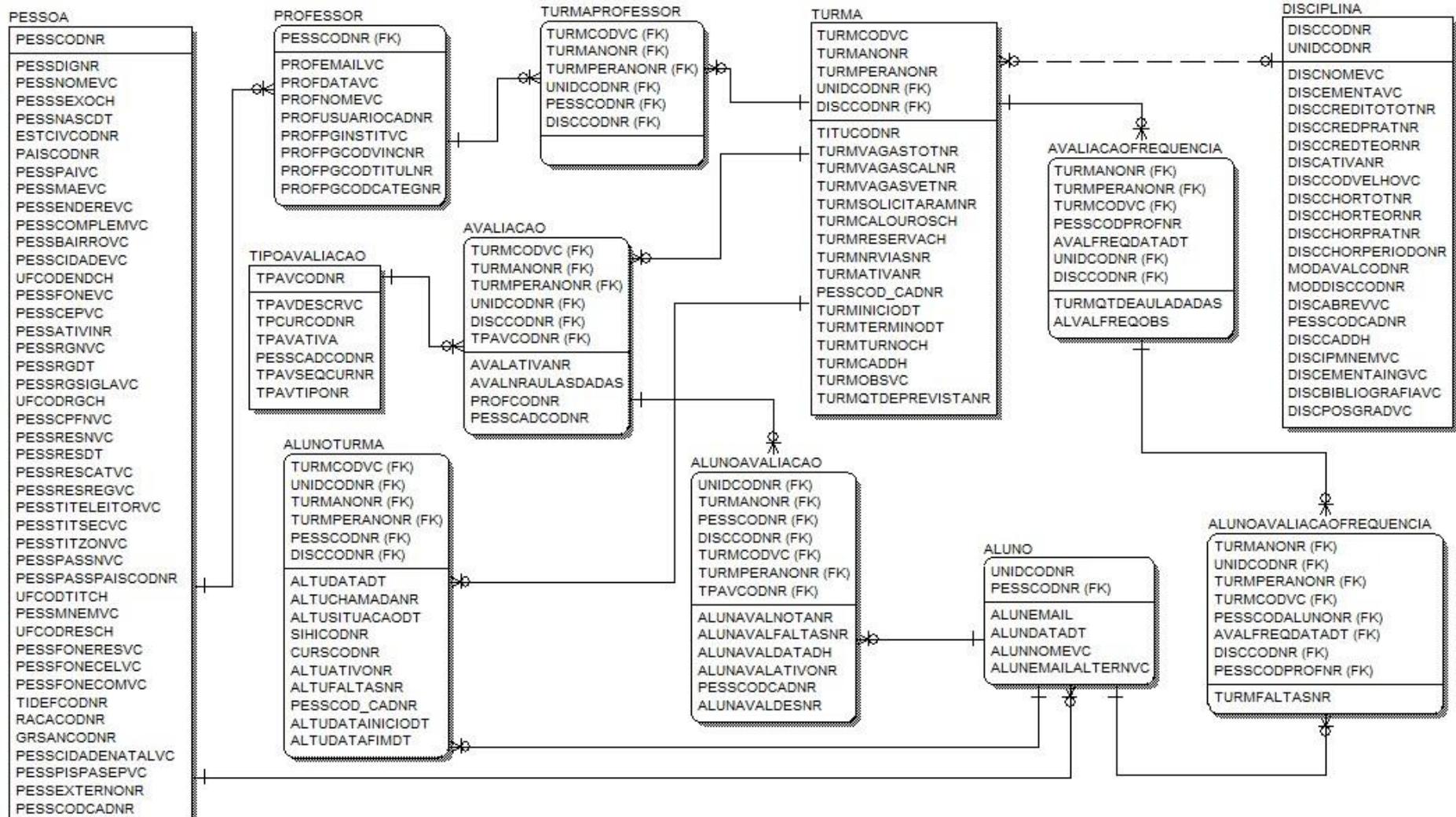


# Substituir métodos por views

Criar uma view baseada em um ou mais métodos existentes no banco de dados (*stored procedures*, *stored functions* ou *triggers*).



# Exemplo de análise



# Problemas identificados

## Tabela TURMA

- Existem turmas que não têm disciplinas correspondentes. Criar uma disciplina “especial” e relacionar esses registros.
- Acrescentar média da turma, a partir das notas de avaliação.

## Tabela ALUNOAVALIAÇÃO

- Manter *log* de quem alterou a nota do aluno.

## Tabela ALUNOTURMA

- Habilitar FK para tabela CURSO.

## Tabela ALUNOAVALIAFREQUENCIA

- Incluir professor que lançou a frequência.

## Tabelas ALUNO e PROFESSOR

- Incluir endereço para correspondência (derivado da tabela PESSOA).



# PUCPR

---

GRUPO MARISTA

Professor convidado  
**Mauro Augusto Borchrdt**

© 2017 – PUCPR. Todos os direitos reservados.  
Nenhum texto pode ser reproduzido sem prévia autorização.