PostgreSQL – Projeto, Administração e Programação

Projeto (Arquitetura)

2.1) Conceitos
2.2) Modelo Entidade Relacionamento
2.3) Modelo Objeto Relacional
2.4) Normalização
2.5) Integridade Referencial
2.6) Planejamento e Otimização

Iniciar com os conceitos básicos: dado, informação e conhecimento.

Dado – representação simbólica, quantificada ou quantificável. Um texto, um número e uma foto são exemplos de dados. O computador somente trabalha com dados, nada de informações e muito menos de conhecimento.

Informação – são mensagens sob forma de dados, que são recebidas e compreendidas. Caso não seja compreendida elas são apenas dados.

Portanto dados são entes meramente *sintáticos* (estruturas).
As informações contém *semântica*, ou seja, contém um significado próprio. A informação é algo objetivo e subjetivo.

Conhecimento – enquanto a informação é um conhecimento teórico, o conhecimento é algo prático.

Só podemos transmitir dados, jamais informações e conhecimento. Os dados podem ser transmitidos, ao serem interpretados tornam-se informações, mas somente ao serem experimentados transformam-se em conhecimento.

Fonte: Livro Bancos de Dados de Valdemar W. Setzer e Flávio Soares Corrêa da Silva.

2.1) Conceitos

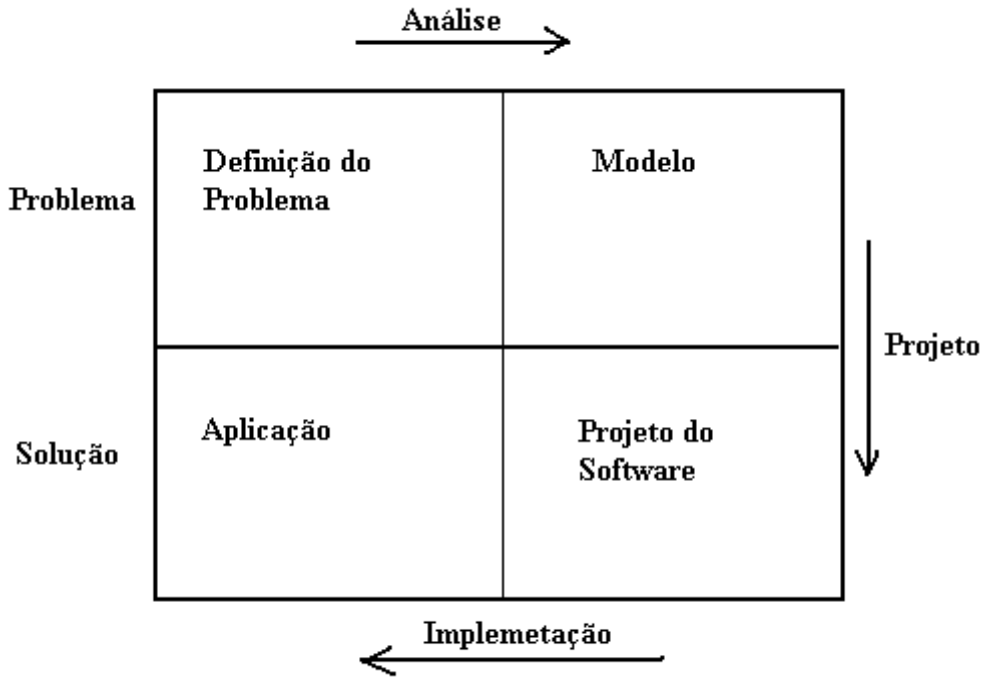
Termo	Conceito
Modelo Relacional	A maioria dos SGBDs existentes são baseados no modelo relacional.
SGBD	É um sistema de computador cuja função principal é a de manter bancos de dados e fornecer informações os mesmos quando solicitados.
DBA	DataBase Administrator (Administrador de Bancos de Dados), é quem trabalha criando os bancos de dados, implementando controle técnico e administrativo dos dados e implementando a segurança.
Entidades	Ou classes, podem ser concretas ou abstratas e representam pessoas, lugares ou coisas (objetos). Entidade no projeto representa Tabela na implementação.
Atributos	Ou propriedades. As entidades são compostas por atributos. Atributos no projeto correspondem a campos na implementação.
Relacionamentos	Associação entre as entidades.
Projeto	Modelo, lógico.
Implementação	Criação e administração, físico.
Modelo de dados	É uma definição lógica e abstrata de um banco de dados.
Implementação	É a aplicação física (prática) do modelo de dados em um computador.
Aplicação cliente	Também chamada de front-end do banco de dados. Roda sobre um SGBD.
DDL	Conjunto de instruções para definir ou declarar objetos do banco de dados.
DML	Conjunto de instruções para manipular ou processar os objetos do banco de dados.
Dicionário de dados	É um conjunto de tabelas do banco de dados que contém informações sobre os objetos do banco, dados sobre dados (metadados).
Bancos distribuídos	Banco que é logicamente centralizado e fisicamente distribuído. A distribuição pode ser em vários sites. Os sites podem estar em vários SO diferentes, redes e máquinas diferentes.
Projeto lógico	É a fase de identificação das entidades de interesse para a empresa e de identificação das informações a serem armazenadas nas entidades. Deve acontecer antes do projeto físico (implementação).
CAST	Valores podem ser convertidos de um tipo para outro através do operador CAST ou através de coerção implícita. O PostgreSQL em sua versão 8.3 está passando a usar bem mais as coerções explícitas.
Projeto de banco	Projetar bancos tem mais de arte que de ciência. Um projeto

	adequado previne perca da integridade dos dados e permite consultas adequadas e eficientes. Qualquer projeto requer uma noção exata do que o banco deverá armazenar. Temos que colher o máximo de informações junto ao cliente e ir além percebendo necessidades que ele próprio não percebe.
Casos de uso	São textos com formatação livre, que descrevem operações do ponto de vista de um eventual usuário interagindo com o sistema. Uma boa forma de capturar as finalidades do banco é construindo casos de uso.
Modelagem dos dados	Exige que reflitamos cuidadosamente sobre os diferentes conjuntos ou classes (entidades) necessárias para solucionar o problema.
Dados	Representação simbólica (abstrata).
Informação	Dados com significado. Mensagem compreendida é informação, caso contrário são apenas dados. Computadores armazenam apenas dados e não informação.
MER	É um modelo que não pode ser representado num computador, existindo apenas na mente de uma pessoa. Pode ser representado como texto ou em forma de diagrama gráfico (DER).
IDs	Caso um registro tenha todos os valores dos campos semelhantes aos valores de outro registro, com exceção do ID, então isso mostra que o ID não é adequado para chave primária da tabela.
Chave Primária	Formada por um campo ou mais que identifica todos os registros de uma tabela de forma única.
Chave Candidata	É a chave onde nenhum subconjunto de campos é também uma chave.
Superchave	É a chave formada por um conjunto de campos além do necessário para identificar todos os registros de forma única, ou seja, contém a chave primária mais um ou mais campos.
Chave alternativa	Quando uma tabela contém uma chave primária, qualquer outra chave será uma chave alternativa.

Projeto do Banco de Dados – Quando um atributo de uma entidade armazena valores duplicados, isso indica que devemos criar uma nova entidade para asmazenar os valores deste atributo e então relacionas as entidades através desse atributo.

CPF no Brasil não é uma boa escolha para **chave primária** em algumas entidades, pois membros de uma famílias podem utilizar o mesmo CPF.

Diagrama de Zelzowitz



Este é um diagrama usado para o desenvolvimento de software mas também podemos aplicá-lo ao projeto de bancos de dados.

Análise – partindo do mundo real, define o problema e gera o modelo
Projeto – partindo do modelo gera o projeto do software (banco)
Implementação – partindo do projeto do software (banco) cria a aplicação.

Partimos da situação real, que deve ser **analisada** gerando um modelo que será transformado em **projeto** e então se **implementará** no computador.

Classes – objetos – atributos (propriedades e métodos)
Tabelas – registros – campos

Uma boa estratégia é portar um conjunto de perguntas a serem respondidas:

1) Qual o principal objetivo do sistema?
2) Que dados são necessários para satisfazer este objetivo
3) Que informações já existem: algum sistema de computador anterior, fichas, formulários, etc.
4) Que entidades foram identificadas?
5) Quais as informações de cada entidade?
6) Qual o rascunho do modelo?
7) Que saídas são esperadas?

Ao final elaborar um primeiro diagrama de classes e vários casos de uso, que relatem no formato texto as interações do sistema.

Anotar o que pode dar errado em cada faze do sistema.

História

As três primeiras formas normais foram definidas por **Ted Codd**.

O Modelo de Entidades e Relacionamentos foi introduzido por Peter Chen.

O MR (Modelo Relacional) foi definido em 1970 por E. F. Codd.

Modelo Relacional Normalizado

- Cada tabela tem um nome distinto
- Cada coluna tem um nome distinto
- Não existem dois registros iguais
- A ordem dos registros e dos campos é irrelevante

SQL – É uma linguagem de declaração e não de programação e atualmente é a linguagem padrão dos SGBDRs e SGBDRO.

Referências:

- Bancos de Dados – Aprenda o que são, Melhore seu conhecimento, Construa os seus
  - De Valdemar W. Seltzer e Flávio Soarea Corrêa da Silva
- Beginning Database Design
  - De Clare Churcher
- A Introduction to Database Systems
  - De C. J. Date

2.2) O Modelo Relacional

- Todos os dados são representados como tabelas
  - Os resultados de qualquer consulta é apenas mais uma tabela!
- Tabelas são formadas por linhas e colunas
- Linhas e colunas são (oficialmente) desordenadas (isto é, a ordem com que as linhas e colunas são referenciadas não importa).
  - Linhnas são ordenadas apenas sob solicitação. Caso contrário, a sua ordem é arbitrária e pode mudar para um banco de dados dinâmico
  - A ordem das colunas é determinada por cada consulta
- Cada tabela possui uma **chave primária,** um identificador único constituído por uma ou mais colunas
- A maioria das chaves primárias é uma coluna apenas (por exemplo, TOWN_ID)
- Uma tabela é ligada (conectada) à outra incluindo-se a vave primária da outra tabela. Esta coluna incluída é chamada uma **chave externa**
- Chaves primárias e chaves externas são os conceitos mais importantes no projeto de banco de dados. Gaste o tempo necessário para entender o que são!

Qualidades de um Bom Projeto de Banco de Dados

- Reflete a estrutura real do problema
- Pode representar todo os dados esperados ao longo do tempo

- Evita armazenamento redundade de items de dados
- Fornece acesso eficiente aos dados
- Suporta a manutenção da integridade dos dados ao longo do tempo Limpa, consistente e fácil de entender
- *Nota: Estes objetivos são algumas vêzes contraditórios!*

Introdução à Modelagem Entidades - Relacionamento

- Modelagem E-R: Um método para projetar banco de dados
- Uma versão simplificada é apresentada aqui
- Representa o dado por **entidades** que possuem **atributos**.
- Uma entidade é uma classe de objetos ou conceitos identificáveis distintos
- Entidades possuem **relações** umas com as outras
- O resultado o processo é um banco de dados **normalizado** que facilita o acesso e evita dados duplicados

*Nota: Muito do projeto formal de banco de dados é focado em **normalizar** o banco de dados e assegurar que o projeto adere ao nível de normalização (isto é, primeira forma normal, segunda forma normal, etc.). Este nível de formalidade está além desta discussão,*
mas deve-se saber que tais formalizações exitem.

Processo de Modelagem E-R
- Identifique as entidades que o seu banco de dados deve representar
- Determine as relações de **cardinalidade** entre as entidades e classifique-as como
    - **Um-para-Um** (por exemplo, um imóvel tem um endereço)
    - **Um-para-muitos** (por exemplo, um imóvel pode ser envolvido em muitos incêndios)
    - **Muitos-para-muitosy** (por exemplo, venda de imóveis: um imóvel pode ser vendido para muitos proprietários, e um proprietário individual pode vender muitos imóveis)
- Desenhe o diagrama entidade-relação
- Determine os atributos de cada entidade
- Defina a chave primária (única) de cada entidade

Do modelo E-R para o Projeto de Banco de Dados
- Entidades com relações **um-para-um** deve ser fundidas em uma única entidade
- Cada entidade restante é modelada por uma tabela com uma chave primária e atributos, alguns dos quais podem ser chaves externas
- Relações **Um-para-muitos** são modeladas por um atributo de chave externa na tabela represntando a entidade do lado "muitos" da relação
- Relações **Muitos-para-muitos** entre duas entidades são modeladaspor uma terceira tabela que possui chaves externas que referem-se às entidades. Estas chaves externas devem ser incluidas na chave primária da tabela da relação, se apropriado
- Ferramentas disponíveis comercialmente podem autimatizar o processo de conversão de um modelo E-R para um esquema de banco de dados

Fonte: http://www.universia.com.br/mit/11/11208/pdf/lecture5-2.pdf (de Thomas H. Grayson)

2.3) Modelo Objeto Relacional

A maioria dos sistemas de gerência de bancos de dados (SGBD) utlizados hoje em dia são baseados no modelo relacional, definido por E.F. Codd. Devido a grande demanda de dados complexos que devem ser armazenados (textos, imagens, som, etc), está havendo uma grande migração para sistemas que suportam esses tipos de dados. Entre as opções de sistemas, estão os bancos de dados orientados a objetos e os objeto-relacionais. Os bancos de dados objeto-relacionais estão situados entre os relacionais e os orientados a objetos. Com este resumo, queremos mostrar algumas características do modelo objecto-relacional.

Principais características do SGBD Objeto-Relacional

Devido a falta de padronização do modelo objeto-relacional, cada fabricante definiu as características do seu SGBD. Analizaremos algumas das principais características dos SGBDs objeto-relacional e daremos uma ênfase no SGBD Oracle 8.x, devido a sua grande popularidade como ORDBMS.

Uma das principais características é permitir que o usuário defina tipos adicionais de dados – especificando a estrutura e a forma de operá-lo – e use estes tipos no modelo relacional. Desta forma, os dados são armazenados em sua forma natural [SAPM 98].

Além dos tipos base já existentes, é permitido ao usuário criar novos tipos e trabalhar com dados complexos como imagens, som e vídeo, por exemplo.

Em última análise, o tipo objeto-relacional está evoluindo mais do que o orientado a objeto, tornando-se cada vez mais um novo produto, que não pode ser comparado com o orientado a objeto, pois este já é um produto maduro [MS 97].

Fonte:
http://paginas.terra.com.br/informatica/arruda/Downloads/Artigos/artigo04/index.htm

2.4) Normalização

Normalização de Tabelas

Normalizar bancos tem o objetivo de tornar o banco mais eficiente, reduzindo as redundâncias, evitando retrabalho em eventuais alterações futuras do modelo.

Uma regra muito importante advinda da normalização ao criar tabelas é atentar para que cada tabela contenha informações sobre um único assunto, de um único tipo.

Tabelas não devem conter campos nulos.

Tabelas não devem conter campos com valores repetidos.

Todas as tabelas de um banco devem estar interligadas por relacionamentos.

Geralmente a aplicação das três primeiras formas normais atende à maioria dos projetos e dificilmente precisamos recorrer à quarta e/ou quinta (somente quando estritamente necessário).

Normalização total não é obrigatório no MER mas fortemente recomendado.

1a Forma Normal
Uma entidade encontra-se nesta forma quando nenhum dos atributos se repete.

Exemplo: evitar cadastrar um produto duas vezes. Quando houver repetição devemos criar uma outra entidade com os campos repetidos e relacionar as tabelas por esse campo.

Exemplo:

clientes
cod nome uf
1 – joão – CE
2 – pedro – CE
3 – manoel – MA

Solução: Criar uma tabela para o campo UF e relacionar ambas.

estados
1 – CE
2 - MA

clientes
1 – joão – 1
2 – pedro – 1
3 – manoel - 2

Outro Exemplo:
Alunos: matricula, nome, data_nasc, serie, pai, mae

Se a escola tem vários filhos de um mesmo casal haverá repetição do nome dos pais. Estão para atender à primeira regra, criamos outra tabela com os nomes dos pais e a matrícula do aluno.

2ª Forma Normal
Quando a chave primária é composta por mais de um campo.

Todos os campos que não fazem parte da chave devem depender da chave (de todos os seus campos).

Caso algum campo dependa somente de parte da chave, então devemos colocar este campo em outra tabela.

Exemplo:
TabelaAlunos

Chave (matricula, codigo_curso)

avaliacao descricao_curso

Neste caso o campo descricao_curso depende apenas do codigo_curso, ou seja, tendo o código do curso conseguimos sua descrição. Então esta tabela não está na 2ª Forma Normal.

Solução:

Dividir a tabela em duas (alunos e cursos) e incorporar à chave existente:

TabelaAlunos
Chave (matricula, codigo_curso)

avaliacao
TabelaCursos
codigo_curso
descricao_curso

3ª Forma Normal
Quando um campo não é dependente diretamente da chave primária ou de parte dela, mas de outro campo da tabela que não pertence à chave primária. Quando isso ocorre esta tabela não está na terceira forma normal e a solução é dividir a tabela.

Um atributo comun não deve depender de outro atributo comun, mas somente da PK.

Exemplo:
Campo tipo total, que depende de deus fatores: quantidade e preco.

Solução:
Não devemos armazenar campos calculados na tabela, pois além de gerar inconsistência

são desnecessários, já que podemos consegui-lo com uma operação.

4a.Forma Normal
Uma tabela está na 4FN, se e somente se, estiver na 3FN e não existirem dependências multivaloradas.

Exemplo: Dados sobre livros

Relação não normalizada: Livros(nrol, (autor), título, (assunto), editora, cid_edit, ano_public)

1FN: Livros(nrol, autor, assunto, título, editora, cid_edit, ano_public)
2FN: Livros(nrol,título, editora, cid-edit, ano_public)
AutAssLiv(nrol, autor, assunto)
3FN: Livros(nrol, título, editora, ano_public)
Editoras(editora, cid-edit)
AutAssLiv(nrol, autor, assunto)

* Redundância para representar todas as informações
* Evitar todas as combinações: representação não-uniforme (repete alguns elementos ou posições nulas)

Passagem à 4FN:

* Geração de novas tabelas, eliminando Dependências Multivaloradas
* Análise de Dependências Multivaloradas entre atributos:
* autor, assunto - Dependência multivalorada de nrol

5a. Forma Normal

Está ligada à noção de dependência de junção Se uma relação é decomposta em várias relações e a reconstrução não é possível pela junção das outras relações, dizemos que existe uma dependência de junção.
Existem tabelas na 4FN que não podem ser divididas em duas relações sem que se altere os dados originais.
Exemplo: Seja as relações R1(CodEmp, CodPrj) e R2(CodEmp, Papel) a decomposição da relação ProjetoRecuro(CodEmp, CodPrj, Papel).

Dica:
Em caso de dúvida, o modelo ideal é o menos engessado, mesmo que paguem o preço de alguma redundância.

- Ao aplicar a 3a. FN ainda persistirem redundâncias, então divide-se a tabela em duas.

Exemplos de Relacionamentos tipo N para N

Autores – Músicas
Alunos – Disciplinas
Produtos – Fornecedores
Produtos – Pedidos

Fases na criação de um banco de dados (aplicativo):

- Análise
- Projeto
- Implementação
- Testes
- Homologação
- Administração

Análise

Nesta fase "enfrentamos" o mundo real para analisar e diagnosticar o problema para então passar um esboço do modelo para a fase de projeto.

Nessa fase procedemos ao levantamento, gerenciamento e validação de informações importantes para o modelo.

Análise é a fase que tem contato com o mundo real, coletando informações junto ao cliente (usuário). Depois da coleta realiza a modelagem das entidades e relacionamentos e das normalizações. Ao final deve gerar um diagrama das entidades e relacionamentos DER.

Projeto - definição das tabelas, índices, chaves (PK e FK), relacionamentos, views, etc.

Melhorando a Normalização de Tabelas

A tabela de CEPs

```
create table cep_full_index
(
        cep char(8),
        tipo char(72),
        logradouro char(70),
        bairro char(72),
        municipio char(60),
        uf char(2)
);
```

Este é um exemplo de tabela não normalizada.
O que podemos melhorar? Praticamente todos os campos se repetem muito, com exceção do cep,
que é a chave primária.

Então deveriamos criar outras tabelas para cada um dos campos que se repetem.
Idealmente devemos normalizar as tabelas, pelo menos até a terceira forma normal,
no momento da criação do banco, na fase de projeto do mesmo, mas para os casos em que
já encontramos a tabela em uso e com muitos registros, como é o caso desta, veja como se pode proceder.
Vamos mostrar como proceder para normalizar o campo uf, criando uma tabela externa

com as ufs brasileiras importando da tabela atual:

Como Exemplo criaremos a Tabela ufs

```
-- Criar a tabela de ufs
create table ufs as select distinct(uf) as uf from cep_full_index;
select * from ufs

-- Adicionar o campo codigo na tabela ufs
alter table ufs add column codigo serial

-- Atualizar o campo uf na tabela cep_full_index para números de 1 a 27, ainda como
CHAR(2)
update cep_full_index set uf=
        case
         when uf ='AC' then '1'
         when uf ='AL' then '2'
         when uf ='AM' then '3'
         when uf ='AP' then '4'
         when uf ='BA' then '5'
         when uf ='CE' then '6'
         when uf ='DF' then '7'
         when uf ='ES' then '8'
         when uf ='GO' then '9'
         when uf ='MA' then '10'
         when uf ='MG' then '11'
         when uf ='MS' then '12'
         when uf ='MT' then '13'
         when uf ='PA' then '14'
         when uf ='PB' then '15'
         when uf ='PE' then '16'
         when uf ='PI' then '17'
         when uf ='PR' then '18'
         when uf ='RJ' then '19'
         when uf ='RN' then '20'
         when uf ='RO' then '21'
         when uf ='RR' then '22'
         when uf ='RS' then '23'
         when uf ='SC' then '24'
         when uf ='SE' then '25'
         when uf ='SP' then '26'
         when uf ='TO' then '27'
        end;
-- Alterando o campo com CAST, de CHAR(2) para INT
alter table cep_full_index alter column uf type int using cast(uf as int)
select distinct(uf) from cep_full_index

select uf from cep_full_index where cep='60420440'

-- Adicionar o relacionamento na tabela de CEP
```

```
alter table ufs add constraint ufs_pk primary key (codigo)
alter table cep_full_index add constraint cep_uf_fk foreign key (uf) references ufs(codigo)

-- Testar novamente a mesma consulta
select uf from cep_full_index where cep='60420440'
```

Fases do Projeto do Banco de Dados:
• Modelagem Conceitual
• Projeto Lógico

Modelo Conceitual - Define apenas quais os dados que aparecerão no banco de dados, sem se importar com a implementação do banco. Para essa fase o que mais se utiliza é o DER (Diagrama Entidade-Relacionamento).

Modelo Lógico - Define quais as tabelas e os campos que formarão as tabelas, como também os campos-chave, mas ainda não se preocupa com detalhes como o tipo de dados dos campos, tamanho, etc.

Não devemos misturar as tarefas de cada uma das fases.

Implementação - criação do banco de acordo com o projeto e também criação dos usuários, grupos e privilégios.

Análise e projeto são fases lógicas e implementação é física.

Comparando Construção de bancos de dados com edifícios:

Análise - reconhecimento do terreno e elaboração de croquis (esboço da planta).

Projeto - Elaboração das plantas de acordo com o esboço: planta baixa, de elevação, detalhes, de situação, etc.

Implementação - Construção do edifício.

Administração - manutenção do edifício.

2.5) Integridade Referencial

Tradução livre do documentação "CBT Integrity Referential":
http://techdocs.postgresql.org/college/002_referentialintegrity/.

Integridade Referencial (relacionamento) é onde uma informação em uma tabela se refere à informações em outra tabela e o banco de dados reforça a integridade.

Tabela1 ----------> Tabela2
Onde é Utilizado?

Onde pelo menos em uma tabela precisa se referir para informações em outra tabela e ambas precisam ter seus dados sincronizados.

Exemplo: uma tabela com uma lista de clientes e outra tabela com uma lista dos pedidos efetuados por eles.
Com integridade referencial devidamente implantada nestas tabelas, o banco irá garantir que você nunca irá cadastrar um pedido na tabela pedidos de um cliente que não exista na tabela clientes.

O banco pode ser instruído para automaticamente atualizar ou excluir entradas nas tabelas quando necessário.

Primary Key (Chave Primária) - é o campo de uma tabela criado para que as outras tabelas relacionadas se refiram a ela por este campo. Impede mais de um registro com valores iguais. É a combinação interna de UNIQUE e NOT NULL.

Qualquer campo em outra tabela do banco pode se referir ao campo chave primária, desde que tenham o mesmo tipo de dados e tamanho da chave primária.

Exemplo:
clientes (codigo INTEGER, nome_cliente VARCHAR(60))

codigo nome_cliente
1 PostgreSQL inc.
2 RedHat inc.

pedidos (relaciona-se à Clientes pelo campo cod_cliente)
cod_pedido cod_cliente descricao

Caso tentemos cadastrar um pedido com cod_cliente 2 ele será aceito.
Mas caso tentemos cadastrar um pedido com cod_cliente 3 ele será recusado pelo banco.

Criando uma Chave Primária
Deve ser criada quando da criação da tabela, para garantir valores exclusivos no campo.

```
CREATE TABLE clientes(cod_cliente BIGINT, nome_cliente VARCHAR(60)
PRIMARY KEY (cod_cliente));
```

Criando uma Chave Estrangeira (Foreign Keys)
É o campo de uma tabela que se relaciona (associa) com o campo Primary Key de outra.
O campo pedidos.cod_cliente refere-se ao campo clientes.codigo, então pedidos.cod_cliente é uma chave estrangeira, que é o campo que liga esta tabela a uma outra.

```
CREATE TABLE pedidos(
cod_pedido BIGINT,
cod_cliente BIGINT REFERENCES clientes,
descricao VARCHAR(60)
);
```

Outro exemplo:

FOREIGN KEY (campoa, campob)
REFERENCES tabela1 (campoa, campob)
ON UPDATE CASCADE
ON DELETE CASCADE);

Cuidado com exclusão em cascata. Somente utilize com certeza do que faz.
Dica: Caso desejemos fazer o relacionamento com um campo que não seja a chave primária, devemos passar este campo entre parênteses após o nome da tabela e o mesmo deve obrigatoriamente ser UNIQUE.
...
cod_cliente BIGINT REFERENCES clientes(nomecampo),
...

Parâmetros Opcionais:

ON UPDATE parametro e ON DELETE parametro.

ON UPDATE paramentros:

NO ACTION (RESTRICT) - quando o campo chave primária está para ser atualizado a atualização é abortada caso um registro em uma tabela referenciada tenha um valor mais antigo. Este parâmetro é o default quando esta cláusula não recebe nenhum parâmetro.

Exemplo: ERRO Ao tentar usar "UPDATE clientes SET codigo = 5 WHERE codigo = 2. Ele vai tentar atualizar o código para 5 mas como em pedidos existem registros do cliente 2 haverá o erro.

CASCADE **(Em Cascata)** - Quando o campo da chave primária é atualizado, registros na tabela referenciada são atualizados.

Exemplo: Funciona: Ao tentar usar "UPDATE clientes SET codigo = 5 WHERE codigo = 2. Ele vai tentar atualizar o código para 5 e vai atualizar esta chave também na tabela pedidos.

SET NULL (atribuir NULL) - Quando um registro na chave primária é atualizado, todos os campos dos registros referenciados a este são setados para NULL.

Exemplo: UPDATE clientes SET codigo = 9 WHERE codigo = 5;
Na clientes o codigo vai para 5 e em pedidos, todos os campos cod_cliente com valor 5 serão setados para NULL.

SET DEFAULT (assumir o Default) - Quando um registro na chave primária é atualizado, todos os campos nos registros relacionados são setados para seu valor DEFAULT.

Exemplo: se o valor default do codigo de clientes é 999, então

UPDATE clientes SET codigo = 10 WHERE codigo = 2. Após esta consulta o campo código com valor 2 em clientes vai para 999 e também todos os campos cod_cliente em pedidos.

ON DELETE parametros:

NO ACTION (RESTRICT) - Quando um campo de chave primária está para ser deletado, a exclusão será abortada caso o valor de um registro na tabela referenciada seja mais velho. Este parâmetro é o default quando esta cláusula não recebe nenhum parâmetro.

Exemplo: ERRO em DELETE FROM clientes WHERE codigo = 2. Não funcionará caso o cod_cliente em pedidos contenha um valor mais antigo que codigo em clientes.

CASCADE - Quando um registro com a chave primária é excluído, todos os registros relacionados com aquela chave são excluídos.

SET NULL - Quando um registro com a chave primária é excluído, os respectivos campos na tabela relacionada são setados para NULL.

SET DEFAULT - Quando um registro com a chave primária é excluído, os campos respectivos da tabela relacionada são setados para seu valor DEFAULT.

Excluindo Tabelas Relacionadas

Para excluir tabelas relacionadas, antes devemos excluir a tabela com chave estrangeira. Tudo isso está na documentação sobre CREATE TABLE:
http://www.postgresql.org/docs/8.0/interactive/sql-createtable.html

ALTER TABLE
http://www.postgresql.org/docs/8.0/interactive/sql-altertable.html

Chave Primária Composta (dois campos)

```
CREATE TABLE tabela (
codigo INTEGER,
data DATE,
nome VARCHAR(40),
PRIMARY KEY (codigo, data)
);
```

Integridade Referencial

Luiz Paulo de O. Santos

Os bancos de dados relacionais disponibilizam facilidades aos desenvolvedores que os tornam mais produtivos, mais confiáveis, rápidos e consistentes no acesso às informações em tabelas.

Uma dessas facilidades é o recurso de Integridade Referencial (IR).

Quem desenvolveu aplicativos DOS em Clipper deve lembrar bem de como era complicada e lenta a instrução SET RELATION, que “criava” um relacionamento lógico entre tabelas.

No mundo relacional, a integridade referencial feita através de chaves cuida para que os dados estejam sempre consistentes, evitando, por exemplo, o aparecimento de registros órfãos.

Vejamos os tipos de chave disponíveis:

Chave Primária

Utilizada para identificar um único registro em uma tabela. A chave primária impede que tenhamos mais de um registro com valores iguais nos campos definidos por ela, ou seja, se tornarmos um campo chave primária, não poderemos ter mais de um registro com o mesmo valor nessa coluna.

Por exemplo: Em um cadastro de pessoas, seria interessante tornar o campo CPF como chave primária, pois não queremos ter uma mesma pessoa cadastrada duas vezes. NOME é um exemplo de campo que não deve ser utilizado como chave primária, pois podemos ter diversos homônimos (pessoas diferentes com mesmo nome). Além dessa verificação, a definição de uma chave primária cria automaticamente um índice para os campos envolvidos, o que pode gerar ganho de performance para determinados tipos de consulta nessa tabela.

Supondo que temos a seguinte tabela em um banco de dados:

CODIGO NOME CIDADE SALARIO
A0001 José da Silva Americana R$ 2.450,00
A0010 Bárbara Alves Piracicaba R$ 8.310,00
B1201 Manoela Tavares São Paulo R$ 1.890,00

Na tabela acima, a chave primária é a coluna CODIGO, logo, em situação alguma podemos incluir outro código A0001, A0010 ou B1201. O banco irá recusar toda entrada duplicada.

Chave Estrangeira

É um tipo de chave criada basicamente para definir um relacionamento entre registros de tabelas diferente, chamado também de relacionamento pai-filho, ou mestre-detalhe, ou mesmo mestre-escravo. No caso, para cada ocorrência do registro da tabela mestre (pai), poderá existir nenhum, um ou mais de um registro relacionados na tabela detalhe (filha), definidos pela chave estrangeira. Diferente da chave primária, a chave estrangeira aceita duplicação de conteúdo nas colunas envolvidas. Assim como na chave primária, a definição de uma chave estrangeira acarreta na criação de um índice composto pelos campos definidos pela chave, podendo haver ganho de perfomance em determinadas pesquisas, devido à existência do índice.

Cuidado! - *Alguns SGBDs permitem o uso da cláusula UNIQUE no momento da definição de uma chave estrangeira, o que impedirá duplicidade na chave, fazendo com que somente relacionamentos de 1 para 1 sejam possíveis. Seguindo as formas normais (teoria da normalização de tabelas), num caso como esse deveríamos aglutinar essas duas tabelas em uma única tabela.*

Supondo que temos a seguinte tabela de vendas, abaixo:

NUMVENDA CODPESSOA DATA VALOR
1 A0001 01/09/2005 R$ 120,00
2 A0001 02/09/2005 R$ 30,00
3 A0010 10/09/2005 R$ 210,00
4 B0010 20/09/2005 R$ 230,00

Observe que a chave primária da tabela acima é a coluna NUMVENDA, definada por um tipo auto-incremento, onde os valores são gerados automaticamente conforme registros forem sendo inseridos. A coluna em vermelho é uma chave estrangeira, onde claramente podemos observar duplicidade, ou “n” ocorrências da chave no decorrer da tabela. No exemplo, a chave estrangeira estaria ligando o campo CODPESSOA da tabela de vendas ao campo CODIGO de uma outra tabela de PESSOAS, através de uma relação de 1 (PESSOAS) para “n” (VENDAS).

No caso de alterações na tabela “pai”, as tabelas relacionadas sofrerão algum tipo de ação para garantir a consistência do relacionamento.

As ações sempre partem da tabela “pai” para a(s) tabela(s) filha(s), sendo elas:

1. Nenhuma ação.
Nessa situação, quando a tabela “pai” (1) sofrer alterações ou exclusões, nenhuma ação automática será realizada em relação às tabelas “filhas”. Se você não fizer um tratamento manual para tornar os dados consistentes, um erro será retornado, pois estaria gerando registros órfãos.

2. Cascade
Nesse caso, sempre que a tabela pai sofre exclusões ou alterações, essas mudanças são repassadas à(s) tabela(s) filha(s). Se o campo relacionado da tabela “pai” foi alterado, automaticamente o campo relacionado nas tabelas filhas será alterado para o mesmo valor, sem a necessidade de implementação de nenhum código adicional. Se algum registro da tabela “pai” for excluído, os registros relacionados a ele nas tabelas “filhas” também serão removidos.

3. Atribuir NULL
Caso a tabela “pai” sofra alterações ou exclusões, os campos relacionados da chave estrangeira nas tabelas filhas assumirão o valor NULL, ficando “jogados” na tabela, sem estarem relacionados com qualquer registro “pai”. Vale lembrar que NULL não é valor, e sim um estado indefinido. Veja mais a frente o item 5 de “Dicas de otimização de chaves”.

4. Assumir o default
Nesse caso, com a alteração ou exclusão dos registros “pai”, os campos relacionados nas tabelas filhas assumirão um valor default.

Observação - *Tanto as chaves primárias como as estrangeiras baseiam-se em índices, ou seja, sua implementação torna-se impossível sem o uso destes. Índices são estruturas organizadas de forma crescente ou decrescente, e que permitem encontrar registros em uma tabela rapidamente, sem precisar percorrê-los um a um. Além disso, os índices podem ajudar na ordenação do resultado das pesquisas (selects), poupando trabalho para o SGBD.*

A maioria dos SGBDs atuais possui um *engine* de pré-processamento de instruções SQL (otimizador), que analisa o código SQL e otimiza sua execução, verificando se é interessante ou não utilizar determinados índices para a realização da consulta. Em algumas situações específicas, o otimizador determina que uma varredura seqüencial dos registros é mais rápida do que a utilização de um índice. Alguns SGBDs permitem que o desenvolvedor pré-determine o “plano” de consulta utilizado para uma determina pesquisa, pulando assim a etapa de otimização.

Dicas de otimização de chaves
É notório e do conhecimento de qualquer desenvolvedor que um projeto mal-feito geralmente gera re-trabalho, e no pior dos casos, precisa ser refeito completamente. Logo, deve-se pensar no que fazer (e como fazer), antes de começar a implementar um sistema. Abaixo citarei algumas dicas que podem acelerar o desenvolvimento de um projeto:

1 Use chaves com campos pequenos
Principalmente na ***chave primária***, que tem uma importância muito grande nos bancos de dados relacionais. Quanto menor for o tamanho do campo, mais rápido será o acesso à chave.

2 Evite campos com tamanhos variáveis
Campos de tamanho fixo são mais indicados para serem chaves primárias. Na maioria dos bancos de dados, o ganho de performance no uso de campos de tamanho fixo é significativo e sensivelmente notado pelos usuários.

3 Evitar excesso de índices
A indexação acelera as buscas, porém, o excesso de índices definidos em tabelas que sofrem muitas inserções (como tabelas de *logs* ou de históricos diversos), exclusões ou mesmo alterações do conteúdo de campos que são chaves, poderá gerar perda de performance, pois a estrutura de todos os índices será atualizada a cada alteração sofrida na tabela. Portanto, não saia por aí criando índices a torto e a direito. Estude caso a caso onde um índice é aconselhável ou não.

4 Evitar, sempre que possível, uso de chaves compostas
Sempre que empregamos chaves compostas (dois ou mais campos), o banco necessita de mais tempo de CPU para processar essas informações, principalmente se os campos forem de conteúdo variável, como proposto no ítem 2. Logo, apesar de ser um recurso bastante interessante e prático para o desenvolvedor, deve ser usado com certo cuidado, principalmente em tabelas onde o conteúdo dos campos que compõe a chave sofre alterações

constantemente, e necessitam ser atualizados em *cascade*.

5 Atribuir NOT NULL para campos que serão empregados como chaves
Deve-se atribuir NOT NULL para todos os campos que serão empregados como chave. Isso é obrigatório no caso das chaves primárias.

No caso das chaves estrangeiras, tome cuidado principalmente se tiver utilizando a ação mencionada no item 4 mais acima (Atribuir NULL).

Definindo chaves em SQL

As instruções para criação de chaves primárias e secundárias são básicas em todos os bancos de dados, sofrendo pequenas mudanças de sintaxe de banco para banco. A seguir citaremos como proceder para sua criação, utilizando uma sintaxe tradicional:

Chaves Primárias:
As chaves primárias são criadas com a instrução:

```
PRIMARY KEY (campo, [...])
```

Exemplo:

```
CREATE TABLE TAB01 (
col_1 integer not null,
col_2 integer not null,
col_3 varchar(2000),
col_4 varchar(500),
PRIMARY KEY (col_1, col_2)
);
```

A chave primária pode ser montada no instante da criação da tabela (CREATE TABLE) ou incluída depois, com a tabela já existente, através do comando ALTER TABLE.

Chaves estrangeiras
As chaves estrangeiras são criadas com a instrução:

```
FOREIGN KEY (campo, [...])
REFERENCES <tabela> (campo, [...])
```

Exemplo:

```
CREATE TABLE TAB02 (
col_a integer,
col_b integer,
col_c varchar(1000),
FOREIGN KEY (col_a, col_b)
REFERENCES TAB01 (col_1, col_2)
);
```

A chave estrangeira pode ser montada no instante da criação da tabela (CREATE TABLE), ou incluída depois com a tabela já existente através do ALTER TABLE.

Conclusão

Espero que esse artigo tenha deixado claro para o leitor a importância da integridade referencial, bem como da sua correta implementação.

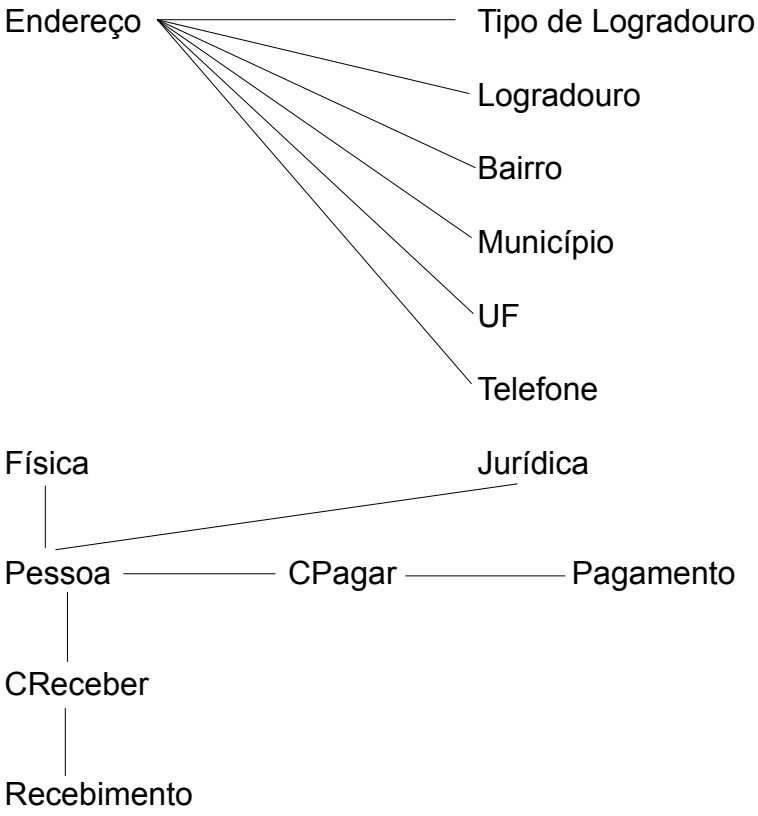
Um complemento ao assunto abordado nesse artigo seriam as formas de normalização, mas deixaremos isso para uma outra oportunidade.

Autor - Luiz Paulo de Oliveira Santos

http://www.metodosistemas.com.br/db/DBFreeMagazine_008.pdf

Alguns Modelos de Sistemas

Sistema de Contas a Pagar e Receber (Entidades e Relacionamentos)

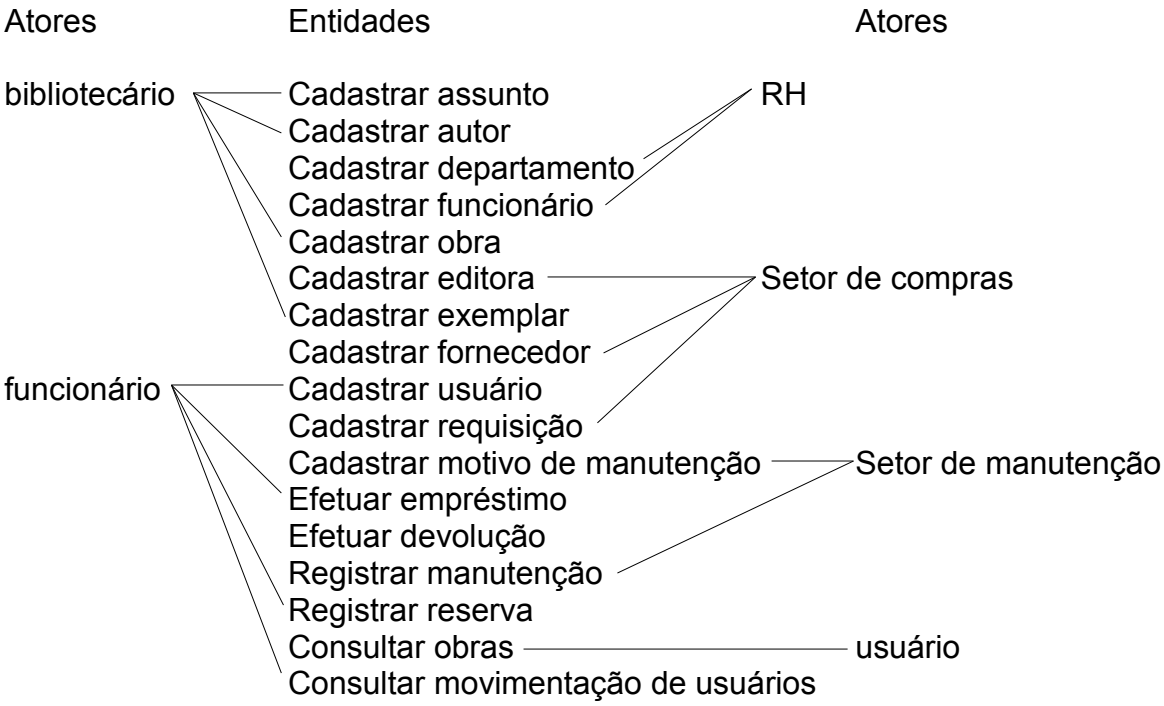


Sistema Acadêmico

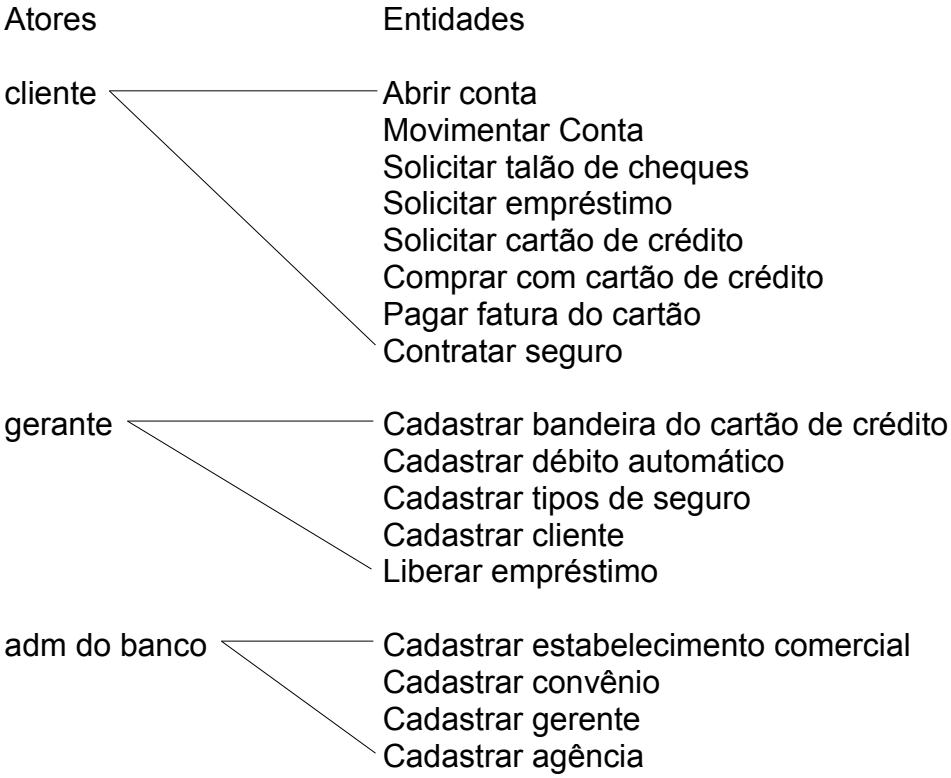
Cadastro do curso
Cadastro de disciplina
Cadastro de aluno
Cadastro de professor

Abrir turma
Matrícular aluno
Emitir diário
Lançar avaliação
Emitir histórico
Consultar avaliação
Alocar professor

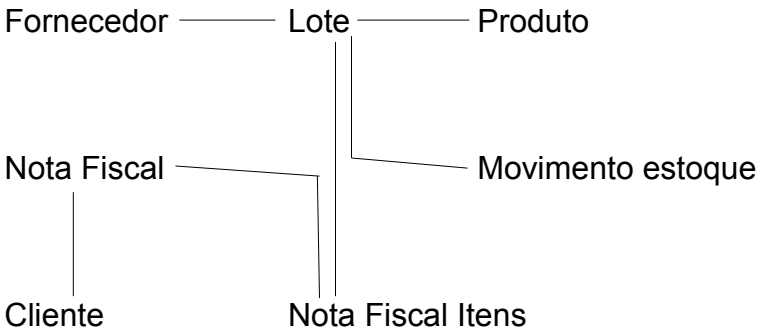
Sistema de Biblioteca



Sistema Bancário



Sistema Comercial



Fonte destes modelos: Revista SQL Magazine 53

Um bom artigo sobre projeto de software:

Programação (I) - Planejamento e Otimização, de Edvaldo Silva de Almeida Júnior no Viva o Linux: http://www.vivaolinux.com.br/artigos/verArtigo.php?codigo=8057

Tabelas são mais conceituais e menos físicas que arquivos.

Simplificando:
- DDL e DCL é para o DBA
- DML é para o programador

SGBDs

SGBDR – SGBD Relacional - tabelas são formadas por linhas e colunas (bidimensional)
SGBDOR - Combina os modelos OO e Relacional
SGBDOO - Modelo OO puro

Características de SGBDs:

- Controle de redundância
- Compartilhamento de dados
- Controle de acesso
- Esquematização (relacionamentos armazenados no banco)
- Backups

Objetivo do SGBD - armazenar objetos de forma a tornar ágil e segura a manipulação das informações.

MER - Modelo Entidades e Relacionamentos - é o modelo composto de entidades e relacionamentos

DER - Diagrama Entidades e Relacionamentos - é o resultado da fase inicial do projeto

(análise)

Exemplos de Entidades:

- Físicas ou jurídicas - pessoas, funcionários, clientes, vendedores, fornecedores, empresas, filiais
- Documentos - ordens de compra, ordens de serviço, pedidos, notas fiscais
- Tabelas - CEPs, UF, cargos, etc

Tupla (termo para a fase de projeto)
Registro (termo para a fase de implementação)

Fase de Análise

Ao definir uma entidade pergunte:
- Há informação relevante sobre esta entidade para a empresa?

Faça um diagrama da entidade com alguns atributos

Relacionamentos

A B
1 ---------- N

Perguntar se A tem 1 ou + B
Perguntar se B tem somente 1 A

Relacionamentos N - N devem ser divididos em dois 1 para N:

1 ---------- N e N ---------- 1

Referência: Livro Instant SQL Programming - Joe Celko

Independência de Banco de Dados

Publicado Fevereiro 5, 2008 Engenharia de software , Todos 0 Comentários
Tags: arquitetura, banco de dados, camada, DAO, engenharia, hibernate, independência, negócio, padrão de projeto, persistência, procedures, software

Um requisito não-funcional importante na engenharia de software é a escolha do repositório de dados da aplicação. A princípio, isso não constitui um grande problema. Afinal, durante a construção do sistema, é sulficiente se adaptar a infra-extrutura já existente no estabelecimento que irá usar o sistema. Se não houver um parque de informática definido, é comum escolher um banco de dados gratuito que normalmente atende a maioria das situações ou, mesmo, comprar licenças de uso de algum banco de dados comercial.

O problema começa na manutenção do sistema. A empresa decide, por exemplo, mudar o banco de dados padrão. Isso pode acontecer por vários motivos: contenção de custos, limitações do banco de dados atual, mudança da plataforma, etc. Aí pode ser um transtorno para os desenvolvedores: migrar os dados e modificar o sistema para se adaptar à nova situação. Sem falar nas mudanças e

reinstalações que poderão ocorrer nas máquinas dos usuários. E se forem muitos usuários?! E se for pra ontem?! E se o pessoal pra fazer essas mudanças for limitado?! Veja que o desgaste gerado por esta “mudança” pode crescer de maneira exponencial. Eu já passei por isso. Posso garantir que os problemas envolvidos podem exigir várias noites de pesquisas e aprendizado instantâneo para poder continuar merecendo a confiança dos seus superiores. Fui obrigado a mudar duas vezes de plataforma de desenvolvimento em menos de dois anos. Hoje cheguei a conclusão que só existem duas plataformas que se adaptam bem a quase todos os cenários no mundo da micro-informática: .NET ou Java.

Pensando nisso decidi postar algumas considerações visando atingir o tão sonhado grau de **indepêndencia de banco de dados**.

Vamos analisar a seguinte situação:

Fig 01: Acesso direto a base

Note o seguite:

1. A aplicação desktop faz acesso direto a base de dados. Todas a regras de negócio e persistência de dados são feitas diretamente pela aplicação. Essa arquitetura é muito utilizada pelos programadores Delphi, VB e similares.
2. Numa aplicação que utiliza servidor WWW, o browser faz requisições ao servidor que por sua vez acessa a base diretamente, da mesma forma que uma aplicação desktop. Os sites no servidor web contém regras de negócio e são responsáveis pelo acesso aos dados. Isso é comum para programadores ASP, JSP, PHP, ColdFusion e similares.

Uma maneira de atingir a indepêndencia de banco neste contexto é utilizar somente instruções SQL ANSI para persistir dados. Mesmo assim isso possui alguns incovenientes. Numa aplicação desktop que está instalada em várias máquinas, qualquer mudança na regra de negócio implicará em reinstalação em todas as máquinas que fazem uso dessa aplicação. Se forem muitos usuários? Se os usuários não sabem instalar sistema? Certamente o help desk vai ter trabalho dobrado pra implantar a nova versão. Utilizando servidor WWW isso ameniza e muito o problema. Basta atualizar o servidor e tudo bem. Todos os usuários terão a nova versão no seu browser. Mas a depender de como essas páginas foram criadas os programadores terão que atualizar a regra ou as configurações de banco em cada página do site. Sem falar que nem sempre é possível desenvolver com instruções SQL ANSI apenas. Uma função de banco, uma VIEW, procedure ou trigger precisa ser reescrita no momento que o banco for trocado.

Quando eu utilizava o Delphi ou o PHP optei pela seguinte solução: Minhas aplicações serviam apenas como GUI. As regras de negócio e a persistência de dados era responsabilidade das procedures de banco. Usava também usuários de banco, functions, roles e views. Triggers não utilizava por serem um tipo de procedures.

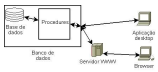


Fig 02: Acesso a base por procedure

Quando tinha que mudar alguma regra, apenas fazia nas procedures. Mudanças na GUI eram mínimas. Assim não precisava reinstalar as aplicações Delphi toda vez que mudasse alguma regra da aplicação. No PHP, da mesma forma. Só mudava a página se houvesse extrema necessidade. Era ótimo a princípio. Mas… E a independência de banco??? Assim a situação só piora o problema. Ganha de um jeito, na manutenção da regra e help desk, mas perde na independencia de banco, ou seja, teria que reescrever todas procedures caso mudasse o banco. Conclusão: abandonei completamente o uso das procedures, views, roles, functions, usuários de banco e tudo mais que dependa diretamente do banco de dados. Minhas aplicações tiveram que ser reescritas completamente em outra arquitetura.

Uma solução padrão para o problema da independência de banco é utilizar um padrão de projeto

chamado DAO. Isso mesmo, PADRÃO DE PROJETO. Padrão DAO faz parte de um conjunto de padrões que resolvem problemas recorrentes. Mas isso é outra história…

Fig 03: Acesso a base por DAO

Cria-se um meio, que normalmente é uma classe da orientação a objetos, por onde a aplicação fará todas as chamadas ao banco, de forma que mudar o banco significaria apenas mudar alguns atributos dessa classe. Ou seja, a classe DAO iria se moldando de acordo com algumas informações passadas no seu construtor. É possivel que alguns métodos precisem ser adaptados por causa do SQL. Mas utilizar SQL ANSI diminue bastante esta necessidade. Eu comecei a fazer isso também. O problema maior é que construir uma classe dessas não pode ser um trabalho solitário. A programação da classe cresce muito. Afinal, para que fique boa, todas as possibilidades do SQL tem que estarem previstas: inserts, updates, deletes e selects e para todos os bancos de dados envolvidos. Se não, pode-se construir uma classe DAO com menos recursos e ir aprimorando a medida que necessidades surjam.

Uma solução que eu considero muito boa é construir uma classe DAO que faça uso de algum framework de persistência.

Fig 04: Acesso a base por DAO com framework de persistência

A responsabilidade de persistir os dados fica para o framework que normalmente já vem preparado para utilizar vários bancos de dados, enquanto a classe DAO da aplicação apenas herda essas características. Eu aconselho a utilização de um framework muito bom, tanto para o Java quanto para .NET: o Hibernate.

O Hibernate consegue ser independente de banco de dados devido a algumas características principais:

1. através de mapeamento de entidades do SGBD como classes da linguagem e via arquivos XML;
2. ampliando a linguagem SQL para HQL (Hibernate Query Language). Esta linguagem manipula objetos e coleções que, depois, o próprio framework converte para SQL que o SGBD entende. Essa conversão é feita utilizando os arquivos XML de mapeamento, configurações de dialeto do banco e drivers de comunicação. Também pode-se utilizar outras formas de montagem do SQL, por exemplo, utilizando métodos de classes. Esses métodos recebem parâmetros que montam o SQL também utilizando os arquivos XML de mapeamento, configurações de dialeto do banco e drivers de comunicação;

Parece complicado mas não é. Inclusive tudo isso é muito produtivo. Desenvolver aplicações sobre essa ótica aumenta bastante a produtividade. Linhas de cada tabela do banco tornam-se objetos e as colunas, propriedades desses objetos. O resultado dos selects do SQL é visto como coleções de objetos e os DML's são vistos como metódos das classes de regra de negócio. É responsabilidade do framework converter tudo isso para o SQL que o banco entenda, e essa conversão deve ser de acordo com o dialeto do banco escolhido e definido em algum arquivo de configuração.

Dessa forma mudar de SGBD significa migrar os dados de um banco pra outro, mudar a configuração de dialeto e alterar, se for o caso, os arquivos XML de mapeamento das entidades. Posso garantir que poucas modificações são necessárias para migrar um sistema do SQL Server para Oracle, se voce utilizar um framework de persistência como Hibernate durante o desenvolvimento. Se a migração for, por exemplo, do Firebird para Oracle ou vice-versa as modificações se resumem a uma ou duas linhas no arquivo de configuração, além, claro, da migração dos dados.Bom, é isso. Espero que este pequeno artigo seja útil e sirva como roteiro pra quem deseja programar com independência de banco.
Link do original:http://reginaldojr.wordpress.com/tag/arquitetura/