

rojeto de

Banco de Dados

Parte IV: Modelo Físico

Vinicius Lourenco

Durante as edições anteriores você aprendeu a construir um modelo lógico (MER), através dos requisitos apurados na análise com o cliente. Vimos a importância de iniciar um projeto de banco de dados com este modelo, nos abstraindo de questões tecnológicas e com a atenção voltada exclusivamente ao domínio do problema.

Neste artigo iniciaremos o modelo físico, que é a transformação do modelo lógico em um formato que possa ser materializado em um banco de dados relacional. Ele apresenta uma preocupação de mais *baixo nível*, ou seja, a partir de agora são analisadas questões relacionadas à performance, ambiente físico, limitações do modelo relacional, entre outras, sem perder o foco na manutenção da integridade das regras de negócio. Algumas literaturas denominam essa transformação de **mapeamento E/R - relacional**.

O modelo físico pode ser utilizado como documentação do banco de dados, sendo o ideal criá-lo antes de iniciar a implementação, assunto que veremos nas próximas edições. No entanto, alguns profissionais pulam essa parte, ou realizam o modelo e a implementação em paralelo. O primeiro caso não é recomendável, pois o modelo físico pode revelar problemas na análise e apresenta uma boa oportunidade para corrigir erros antes da implementação, o que acaba sempre saindo mais barato. O segundo caso pode ser viável se uma situação semelhante já tiver sido modelada em outro projeto e um mesmo profissional fique responsável pelas duas atividades, evitando inconsistências entre o banco de dados e a documentação. Como em todo projeto, é importante que as alterações futuras no banco de dados sejam refletidas na documentação, para evitar sua obsolescência.

Veremos a seguir alguns passos para transformação do modelo lógico em físico. Ao construir vários projetos você ganha experiência e percebe que algumas etapas podem ser omitidas, otimizando o tempo de modelagem.

Iniciando o Modelo Físico

A partir de agora alguns elementos mudam de nome, no entanto, o conceito permanece o mesmo: entidades e atributos serão chamados de tabelas e campos, respectivamente. As tabelas vão identificar as entidades e alguns relacionamentos. O atributo identificador se torna chave primária da tabela. Conceitos como agregação, especialização e generalização não são suportados pelo modelo relacional e não existem no modelo físico.

O primeiro passo é a padronização na descrição dos elementos. O modelo lógico apresenta alguns atributos (como *Preço Aluguel* e *Valor Multa*) e entidades (como *Pessoa Física* e *Pessoa Jurídica*) cujos nomes possuem “espaço” em sua composição. Alguns bancos de dados relacionais possuem regras para nomenclatura de objetos, como o não uso de espaços e acentuação. Por estas limitações, deve haver uma convenção para nomes - no modelo, os espaços foram substituídos por “_” e os acentos foram eliminados (**figura 1**).

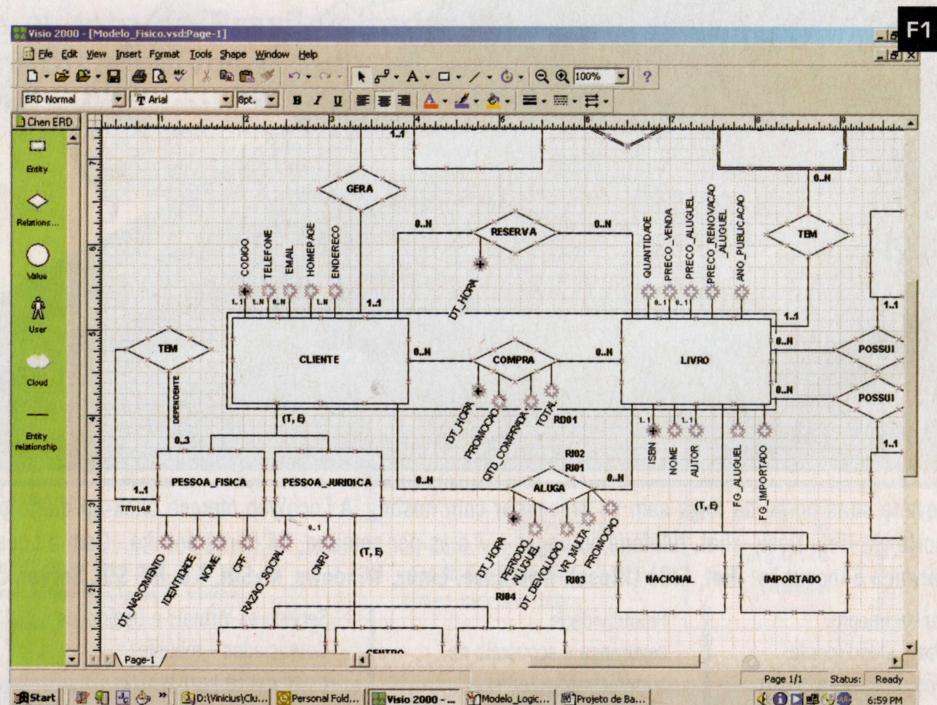


Figura 1 - Modelo de Entidade-Relacionamento parcial para Book.net

Chaves Primárias

Por padrão, os atributos identificadores são mapeados como chaves primárias, como podemos ver nas tabelas *Livro*, *Cliente* e *Nota Fiscal*. No entanto, por questões de performance, isso nem sempre vai ocorrer. Por exemplo, as entidades *Editora* e *Assunto* receberam um novo campo, chamado *Código*, para representar a chave primária, substituindo os atributos identificadores *Nome* e *Assunto*. Na próxima edição veremos maiores considerações sobre a definição de chaves. Veja o modelo atualizado na figura 2.

Chaves Estrangeiras

Existem alguns elementos que são característicos do modelo físico. Um deles é a chave estrangeira (*foreign key*), recurso utilizado para ligar tabelas em um relacionamento. A idéia é criar um campo na tabela filho que recebe o conteúdo da chave primária da tabela pai, garantindo um valor comum nas duas tabelas e possibilitando ao banco de dados realizar o relacionamento. Por exemplo, na relação entre *Editora* (pai) e *Livro* (filho), este último receberá um novo campo, *Cod_Editora*, que será chave estrangeira. O mesmo acontece no relacionamento com *Assunto*, onde a tabela *Livro* receberá o campo *Cod_Assunto* (figura 2). Para entender na prática, observe o conteúdo da tabela *Editora* abaixo:

Código	Nome
1	Neoficio Editora
2	Editora Relativa

Para relacionar um livro com uma determinada editora, utilizamos a chave estrangeira conforme a tabela *Livro* abaixo:

ISBN	Nome	Cod_Editora
1234	Kit Delphi 6 Completo	2
5678	Delphi & Flash	2
9012	Revista ClubeDelphi	1

Relacionamentos

Em todo relacionamento 1:N, a chave primária do lado 1 é exportada para o lado N, gerando uma chave estrangeira na tabela filha.

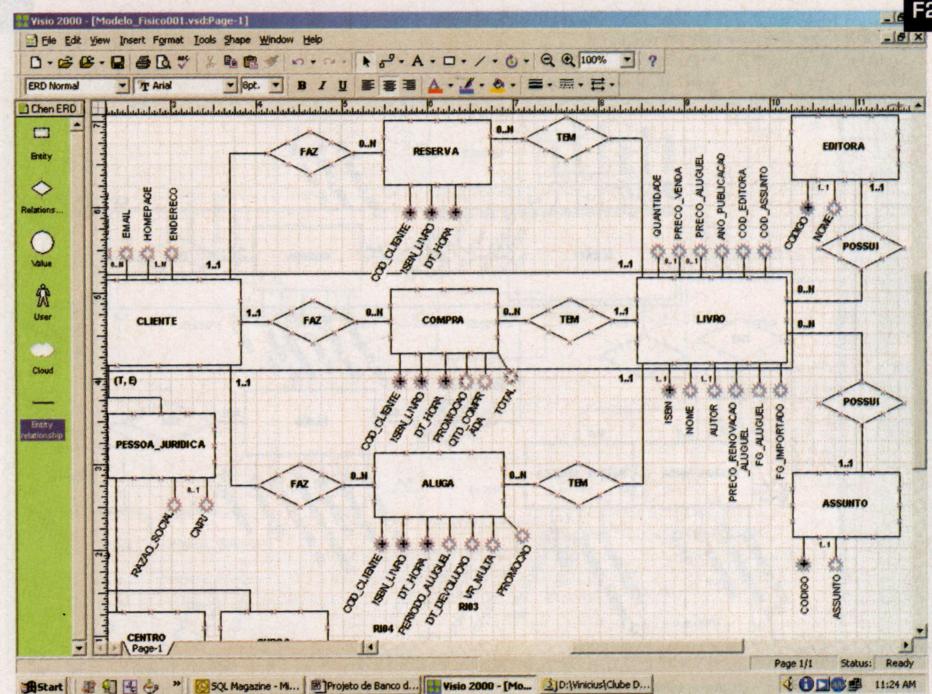


Figura 2 - Modelo de Entidade-Relacionamento parcial para Book.net

Como vimos, a tabela *Livro* ganhou duas chaves estrangeiras: *Cod_Editora* e *Cod_Assunto*. A tabela *Item_Nota_Fiscal* possui um relacionamento 1:N com *Livro* e recebe a chave estrangeira *ISBN_Livro*.

Em relacionamentos 1:1 escolhe-se uma das tabelas para exportar a chave primária. Neste modelo, selecionaremos a tabela que possui menos campos na chave (*Nota Fiscal*), causando assim maior performance na junção dos elementos. No relacionamento entre *Compra* e *Nota Fiscal*, a primeira receberá o campo *Numero_Nota_Fiscal* como chave estrangeira. Se fosse o contrário, *Nota_Fiscal* receberia três novos campos, correspondentes a chave primária de *Compra*, composta por *Cod_Cliente*, *ISBN_Livro* e *Dt_Hora*.

Outra regra diz respeito à relações N:N. Nesse caso, cria-se uma tabela correspondente ao relacionamento e transpõe-se para ela as chaves primárias das entidades envolvidas. No modelo, as entidades *RESERVA*, *ALUGA* e *COMPRA* serão transformadas em três novas tabelas. Na figura 2 observa-se que essas tabelas herdaram as chaves primárias de *Cliente* e *Livro*. Vemos também que cada chave primária possui o atributo *Data_Hora*, existente inicialmente como identificador do relacionamento que deu origem às tabelas em questão.

No entanto, a criação da tabela *Compra*

gera um obstáculo. Lembre-se que no modelo físico estamos no domínio da aplicação e começamos a encontrar problemas inerentes ao modelo relacional para armazenamento dos dados. Observe que a tabela *Compra*, neste momento, precisa guardar cada item comprado em um registro, sendo necessário vários registros para representar uma compra contendo vários livros. Dessa forma, se torna impossível manter o relacionamento 1:1 modelado logicamente com a entidade *Nota Fiscal*. Para resolver um obstáculo do mundo físico, esse relacionamento será convertido para 1:N, onde uma nota fiscal será representada por uma ou várias ocorrências da tabela *Compra*. No entanto, essa ainda não é a solução final - quando estudarmos a **normalização**, veremos novas técnicas que transformarão ainda mais as características destas tabelas.

Especialização/Generalização

Como vimos na parte II, quando modelamos uma especialização novas entidades são criadas. Contudo, essas entidades podem ou não dar origem a tabelas no modelo físico. Por exemplo, observe que as entidades *Pessoa Física* e *Pessoa Jurídica*, especializações de *Cliente*, possuem comportamentos particulares: uma *pessoa física* contém um auto relacionamento, pois existem titulares e dependentes. Uma *pessoa jurídica* pode ser

Projeto de Banco de Dados

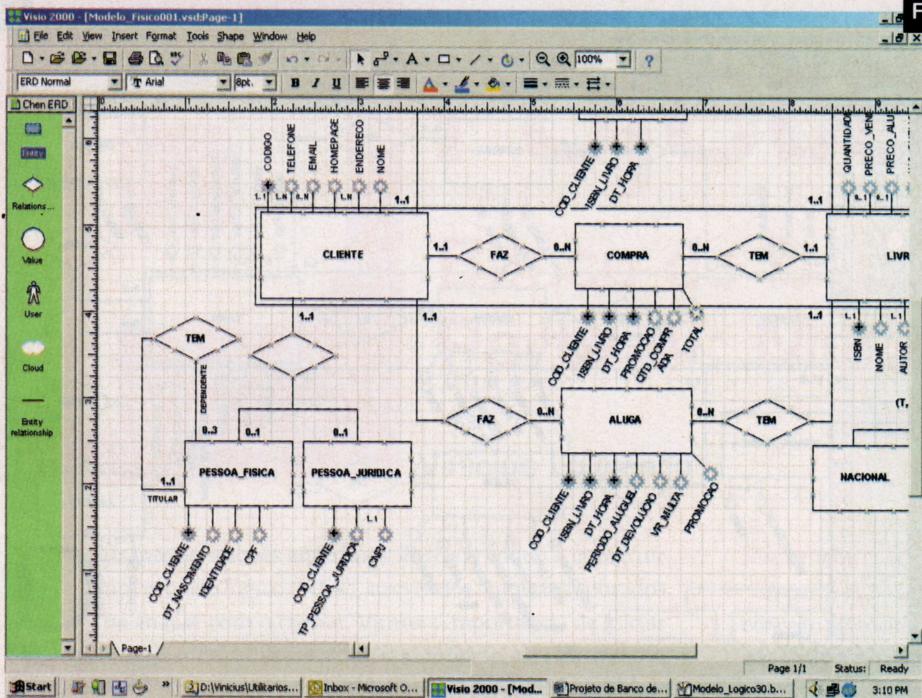


Figura 3 - Modelo de Entidade-Relacionamento parcial para Book.net

uma universidade, um centro educacional ou um curso. Além disso, elas possuem atributos próprios, característicos da regra de negócio.

Por terem comportamento e atributos diferentes, as entidades *Pessoa Física* e *Pessoa Jurídica* se tornarão tabelas no modelo físico.

Como um banco de dados relacional não possui o conceito de especialização, uma tabela não pode herdar atributos, características ou comportamentos de outra tabela (que neste caso seria a tabela *Cliente*). Podemos então modelar de duas formas:

1- A tabela *Cliente* (generalização) deixa de existir e todos os seus atributos são repetidos nas tabelas *Pessoa_Física* e *Pessoa_Juridica*;

2- A tabela *Cliente* continua a existir, possuindo dois relacionamentos 1:1 um com a tabela *Pessoa_Física* e outro com a tabela *Pessoa_Juridica*. Essa solução é bastante utilizada pelos analistas, desenvolvedores e administradores de banco de dados.

Seguiremos a segunda opção. Observe na figura 3 como ficou modelada a especialização da tabela *Cliente*.

Na especialização da entidade *Pessoa Jurídica*, podemos observar que as subentidades *Universidade*, *Centro Educacional* e

Curso existem apenas para representar tipos de cliente da livraria. Elas não possuem atributos, características ou comportamentos diferentes entre si, não fazendo sentido se tornarem tabelas. Criaremos então um novo campo (*Tp_Pessoa_Juridica*) na tabela *Pessoa_Juridica*, para indicar seu tipo (figura 3).

E se criássemos uma tabela auxiliar para os tipos de pessoa jurídica? Observe que o tipo de especialização desta entidade é total e exclusiva. Como foi mostrado na parte II do artigo, o tipo total significa que todas as possíveis subentidades estão modeladas, ou seja, não existe outro tipo de pessoa jurídica no negócio da livraria. Nesse caso não há a necessidade de criarmos uma tabela somente para conter três registros. Se o tipo de especialização não fosse total, outros tipos de pessoas jurídicas poderiam existir e neste caso a melhor solução seria criar a tabela auxiliar.

Observações:

• Podemos notar que o relacionamento entre a tabela *Cliente* e as tabelas *Pessoa_Física* e *Pessoa_Juridica* não possui nome, pois não foi encontrado um que expressasse uma idéia clara sobre a relação (figura 3). Na parte I, vimos que a descrição é importante para a legibilidade do modelo, mas não é obrigatória.

• A cardinalidade do campo *CNPJ* mudou para 1:1, indicando sua obrigatoriedade na tabela *Pessoa_Juridica*. No modelo lógico a cardinalidade era 0:1, pois as tabelas eram especializações e o cliente poderia ser físico ou jurídico.

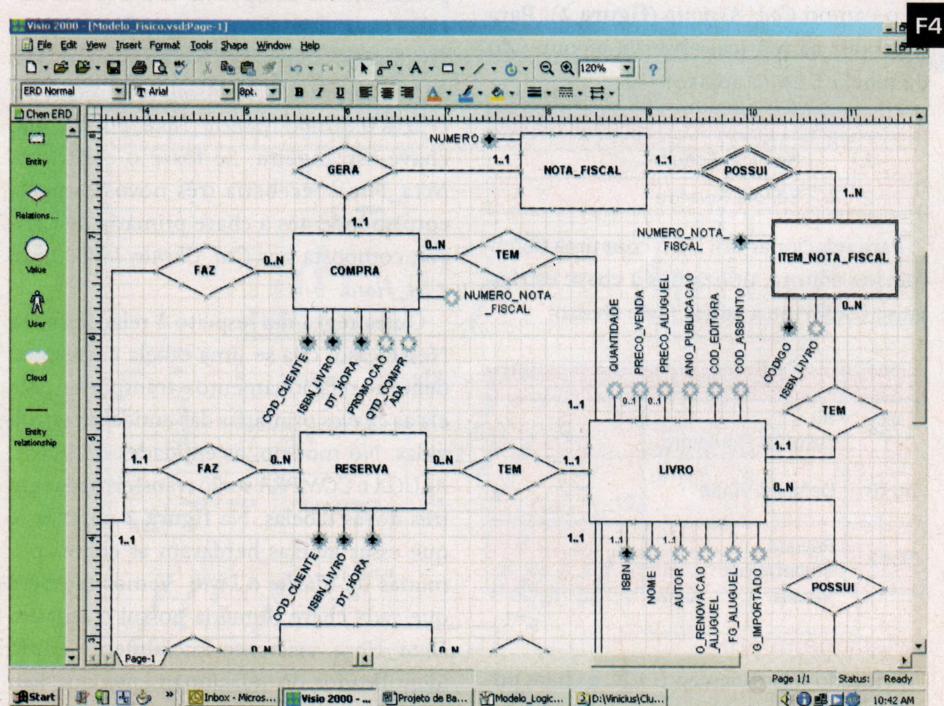


Figura 4 - Modelo de Entidade-Relacionamento parcial para Book.net

- O campo *Nome* foi inserido na tabela *Cliente*, com o intuito de armazenar tanto o nome de pessoas físicas quanto a razão social. Isso foi feito em função da performance, pois uma consulta não precisará unir várias tabelas para recuperar o nome ou a razão social do cliente. Lembre-se que em telas de busca esse campo é muito utilizado.

- Na especialização de *Livro*, as subentidades *Nacional* e *Importado* apenas indicam a origem do livro. Assim como em *Pessoa_Jurídica*, um campo substituirá a especialização. Na verdade, esse campo já existe desde o modelo lógico e foi apenas renomeado para *Fg_Importado* (**figura 2**).

Agregação

A agregação não existe no modelo físico. Como vimos na edição anterior, só existe agregação para relacionamentos com cardinalidade N:N, os quais se tornaram tabelas no modelo físico. Sendo assim, não é mais necessário existir a agregação para que haja um relacionamento entre *Compra* e *Nota_Fiscal* (**figura 4**).

A agregação indica que para existir o relacionamento entre as entidades *Nota Fiscal* e *Compra* é necessário que haja, no mínimo, uma compra efetuada. No modelo físico deve-se manter essa regra.

Entidades Fracas

Foi visto que as entidades fracas dependem das entidades fortes para existir. No modelo físico, a chave primária de uma tabela fraca contém sempre uma cópia da chave primária da tabela forte. Por exemplo, a tabela *Item_Nota_Fiscal* será identificada pelos campos *Numero_Nota_Fiscal*, que é chave primária da tabela *Nota_Fiscal*, e *Código*, que individualiza cada item da nota (**figura 4**).

Observe na **figura 4** que a tabela *Item_Nota_Fiscal* e seu relacionamento com *Nota_Fiscal* possuem a mesma representação visual do modelo lógico, indicando que ela é uma tabela fraca. Isso significa que alguns recursos visuais do modelo lógico podem ser reutilizados para deixar o modelo físico mais legível.

Restrição de Integridade / Regra de Derivação

Observe nas figuras 1 à 4 que algumas

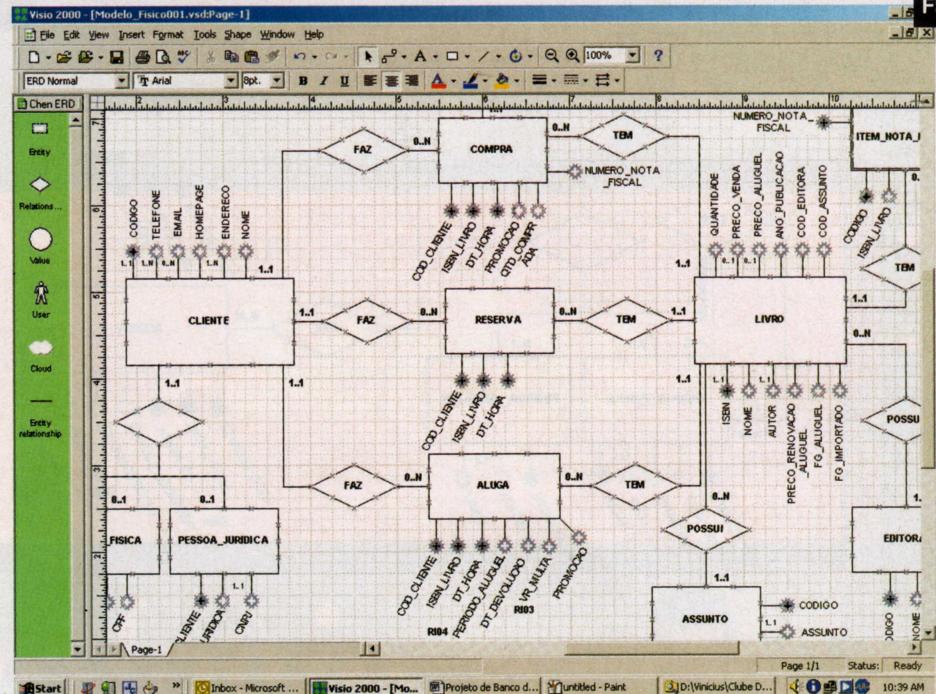


Figura 5 - Modelo de Entidade-Relacionamento parcial para Book.net

restrições de integridade (RIs) foram retiradas. O motivo é que os relacionamentos que possuem RIs no modelo lógico foram transformados em tabelas. Logo, como uma RI não é representada ao lado de uma tabela, somente as restrições referentes aos atributos permaneceram no modelo. Vale lembrar que todas as RIs, representadas graficamente ou não, devem ser especificadas no dicionário de dados. Veja o modelo atualizado na **figura 5**.

Normalmente, no mapeamento das regras de derivação (RDs), os campos calculados são eliminados do modelo físico e passam a constar somente no dicionário de dados. Em alguns casos, por motivos de performance, esse campo é criado fisicamente na tabela. Essa é a análise principal sobre um campo calculado: se ele será ou não transformado em um campo físico.

No modelo, somente o atributo *Total do relacionamento Compra* possui RD. Já que esse campo apenas mostra o total de uma determinada compra e não tem grande utilização, vamos deixá-lo como calculado e retirá-lo do diagrama.

NOTA: Em alguns servidores de banco de dados é possível criar um campo físico e dizer que seu valor será calculado, indicando a fórmula correspondente.

Auto relacionamento

Em todo auto relacionamento, a tabela receberá um novo campo como chave estrangeira, o qual terá um relacionamento com a chave primária da própria tabela. Na **figura 6**, a tabela *Pessoa_Fisica* tem um novo campo chamado *Cod_Cliente_Titular*. Quando esse campo for nulo, o cliente será titular; quando estiver preenchido, o cliente será um dependente. Nesse caso, o campo receberá o valor de algum outro registro da própria tabela.

Outras convenções para o modelo físico

Note que as convenções utilizadas neste artigo não são únicas. Existem diversos padrões para a elaboração de modelos físicos, como os três tipos abaixo:

Modelo Físico sem campos – É utilizado por programadores que desejam representar somente os relacionamentos entre as tabelas. Uma vantagem é que há uma economia de espaço na visualização de modelos extensos.

Modelo Físico com campos sem seus tipos – É o nosso caso. Geralmente utilizado quando estamos iniciando a construção do modelo físico, esse modelo deixa a definição de tipos para a fase de implementação.

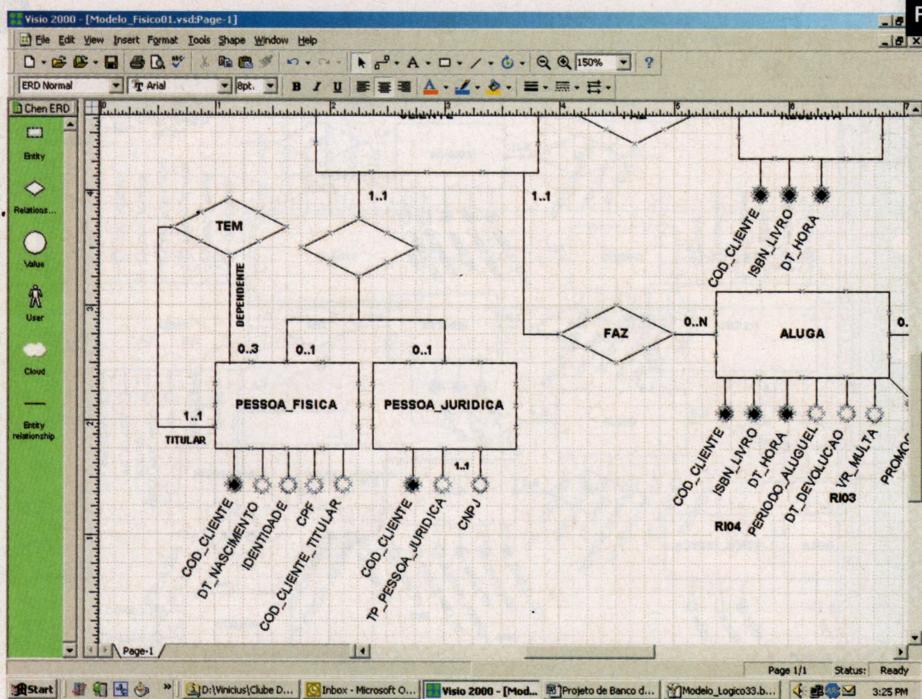


Figura 6 - Modelo de Entidade-Relacionamento parcial para Book.ne

Modelo Físico com campos e seus tipos

– É o modo mais completo de visualização, visto que ficam explícitos os tipos de dados a serem utilizados, os campos e os relacionamentos entre as tabelas.

Conclusão

É importante ter em mente que não existe o melhor modelo ou a melhor implementação para um banco de dados. Muitas soluções podem ser levantadas para um mesmo

problema e todas podem estar corretas sob diferentes pontos de vista. Assegure-se de, antes de iniciar a implementação, questionar o modelo mais de uma vez, identificando a maior quantidade de cenários possíveis. Se a equipe chegou até o final do modelo físico e ainda não detectou nenhum erro, é muito provável que o problema do cliente não tenha sido analisado corretamente.

Na próxima edição daremos continuidade ao modelo físico, discutindo a normalização e considerações sobre performance. Até lá. ■

Vinicius Lourenço de Sousa é analista de sistemas e desenvolvedor Delphi/Java em projetos Web e Off-Line, utilizando os bancos de dados InterBase, Oracle 9i e DB2/AS400, na DBA Engenharia de Sistemas. É Pós-Graduado em Análise, Projetos e Gerência de Sistemas pela PUC-RJ e possui certificação BrainBench em Delphi e RDBMS. Pode ser contatado no e-mail vsouza@dba.com.br

Pensando Java?

www.javamagazine.com.br
info@javamagazine.com.br



rojeto de

Banco de Dados Parte V: Modelo Físico

Vinicius Lourenço de Sousa

Continuando a construção do projeto de banco de dados da livraria Book.NET, veremos mais alguns recursos do modelo físico: normalização, chaves primárias e desnormalização.

Normalização

Normalizar significa organizar as tabelas de forma que informações redundantes não sejam armazenadas. A redundância gera uma série de problemas para o banco de dados – veja o artigo “Normalização - Conceitos e Técnicas”, publicado nesta edição, para maiores detalhes.

A normalização é composta por várias fases, onde cada uma é chamada de forma normal (FN). As tabelas, para serem consideradas bem projetadas, devem respeitar a definição das FN's. Neste artigo serão estudadas e aplicadas as três primeiras formas normais, já que as demais FN's são muito específicas e não são utilizadas na maioria das aplicações comerciais.

A seguir, daremos início a normalização aplicando a primeira forma normal nas tabelas do nosso projeto.

Nota: Muitos conceitos utilizados aqui são explicados no artigo “Normalização - Conceitos e Técnicas”, publicado nesta edição.

Primeira Forma Normal (1FN)

A regra da 1FN é simples: cada campo de uma tabela deve ser atômico, ou seja, indivisível. A 1FN geralmente é violada de três formas em um projeto de banco de dados:

1) **Campos compostos:** por exemplo, o campo *Endereço* da tabela *Cliente*. Nesse caso, obter a 1FN significa separar esse campo em partes indivisíveis. Veja o resultado da divisão na **figura 1**.

Definir se um campo é composto ou atômico não é uma tarefa simples. Normalmente, essa decisão é tomada com base na regra de negócio da aplicação. Por exemplo: o campo *Nome* deve ser dividido em partes menores (primeiro nome, sobrenome etc.)? A resposta é “depende” – em algumas aplicações (como venda de passagens) a separação do nome é importante; em outras, não.

2) **Campos repetidos:** imagine uma tabela de notas com campos *Nota1*, *Nota2*, *Nota3* etc. Uma estrutura como essa tem problemas pois não é suficientemente flexível, desperdiça espaço e é ineficiente para realizar alguns tipos de busca. Se você armazenar uma única nota, estará desperdiçando espaço com colunas vazias. Se precisar armazenar quatro notas, terá que criar colunas extras.

Cód.	Nome	Telefone	Email	Homepage	Rua	Nº	Bairro	Cidade	Cep	UF	Complemento	Tipo
1	João	21234568	bd@sql.com.br	www.sqlmagazine.com.br	José Lemos	154	Centro	Rio de Janeiro	25514000	RJ	Casa	Residencial
2	Maria	32457145			Parque Ramos	352	Lagoa	Rio de Janeiro	32456213	RJ	Apartamento	Residencial
3	Aline	37841256			Mário Lagos	521	Centro	Rio de Janeiro	52364145	RJ	Casa	Residencial

Tabela com campos repetidos

Co_aluno	Nome	Nota1	Data 1	Nota 2	Data 2
1	José	6	15/12/2002	8	15/01/2003
2	Maria	7.5	16/12/2002	5	16/01/2003

Tabela na 1FN

Co_aluno	Nome
1	José
2	Maria

Tabela Auxiliar

Co_aluno	Nota	Data
1	6	15/12/2002
1	8	15/01/2003
2	7.5	16/12/2002
2	5	16/01/2003

Cod_Cliente	Codigo	Rua	Numero	Bairro	Cidade	Cep	Estado	Complemento	Tipo
1	1	José Lemos	154	Centro	Rio de Janeiro	25514000	RJ	Casa	Residencial
1	2	Mário Lagos	546	Centro	Rio de Janeiro	25578213	RJ	Apartamento	Comercial
2	1	Parque Ramos	352	Lagoa	Rio de Janeiro	32456213	RJ	Apartamento	Residencial

F3

Uma solução eficiente é criar uma tabela auxiliar e relacioná-la com a tabela principal através de uma chave estrangeira. Observe a figura 2.

3) Ocorrência de tabelas aninhadas ou atributos multivalorados: ambos possuem múltiplos valores para um mesmo atributo. A entidade *Cliente* possui quatro atributos multivalorados: *telefone*, *email*, *homepage* e *endereço* (observe no modelo que a cardinalidade destes atributos é 1:N).

A solução é criar uma tabela auxiliar, semelhante ao que é feito com atributos repetidos. Como exemplo, veja a tabela auxiliar *Endereço* na figura 3.

O mesmo procedimento foi feito para os campos *email*, *homepage* e *telefone*. O modelo atualizado após a aplicação da 1FN pode ser visualizado na figura 4.

Nota: Observe que as chaves primárias de Endereço, Telefone, Email e Homepage possuem o campo *Cod_cliente*, indicando que as tabelas são fracas em relação a Cliente.

Segunda Forma Normal (2FN)

Uma tabela está em 2FN quando está em 1FN e cada campo que não faz parte da chave primária depende de todos os campos da chave primária. Uma tabela que não se encontra na segunda forma normal contém **dependências funcionais parciais**.

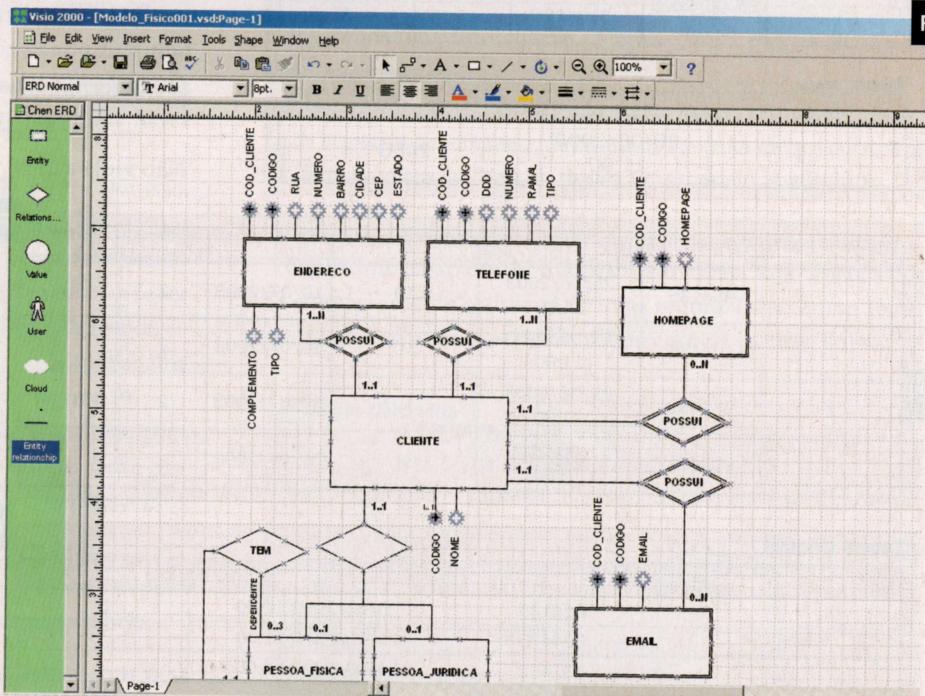
Veja a tabela *Item_Nota_Fiscal* abaixo:

Numero_Nota_Fiscal	Código	Isbn_Livro
1	1	154
1	2	546
2	1	512

Essa tabela atende a 2FN, pois para obtermos o valor do campo *ISBN_livro* precisamos saber o valor dos campos *Numero nota fiscal* e *Código* (que são a chave primária).

A tabela *Compra* não está na segunda forma normal. Veja na figura 5 um exemplo de instância de *Compra* (como vimos na edição anterior, numa compra de vários livros essa tabela armazena a informação de cada item comprado em um registro diferente).

O campo que desobedece a 2FN é *Numero nota fiscal* – ele não necessita de toda a chave primária para ser identificado. Observe que apenas com o valor dos



F4

campos *Cod_cliente* e *Dt_Hora* já conseguimos obter a nota fiscal correspondente:

Cod_Cliente	Dt_Hora	Numero_Nota_Fiscal
1	25/06/2003	= 2456

Para normalizar essa tabela, devemos:

- 1) Mover os campos com dependência funcional parcial – nesse caso, *Numero_nota_fiscal* – e colocá-los em uma nova tabela;
- 2) A chave primária da tabela criada será formada pelo(s) campo(s) que era(m) determinante(s), na tabela original, dos campos movidos;
- 3) Repetir os passos até eliminar a dependência parcial.

Veja na figura 6 que a tabela *Compra* foi decomposta em duas tabelas e que a dependência funcional parcial deixou de existir. Para melhorar a legibilidade, vamos chamar a tabela nova de *Compra* e a tabela original de *Itens_Compra* (que era o que ela realmente representava).

Cod_Cliente	Isbn_Livro	Dt_Hora	Promocao	Qtd_Comprada	Numero_Nota_Fiscal
1	854135	25/06/2003 11:00	10	1	2456
1	624475	25/06/2003 11:00		2	2456
1	631482	25/06/2003 11:00		1	2456

Uma compra de três livros

F5

P rojeto de Banco de Dados

Tabela original

Cod_Cliente	Dt_Hora	ISBN_Livro	Promoção	Qtd_Comprada
1	25/06/2003 11:00	854135	10	1
1	25/06/2003 11:00	624475		2
1	25/06/2003 11:00	631482		1

Tabela nova

Cod_Cliente	Dt_Hora	Numero_Nota_Fiscal
1	25/06/2003 11:00	2456

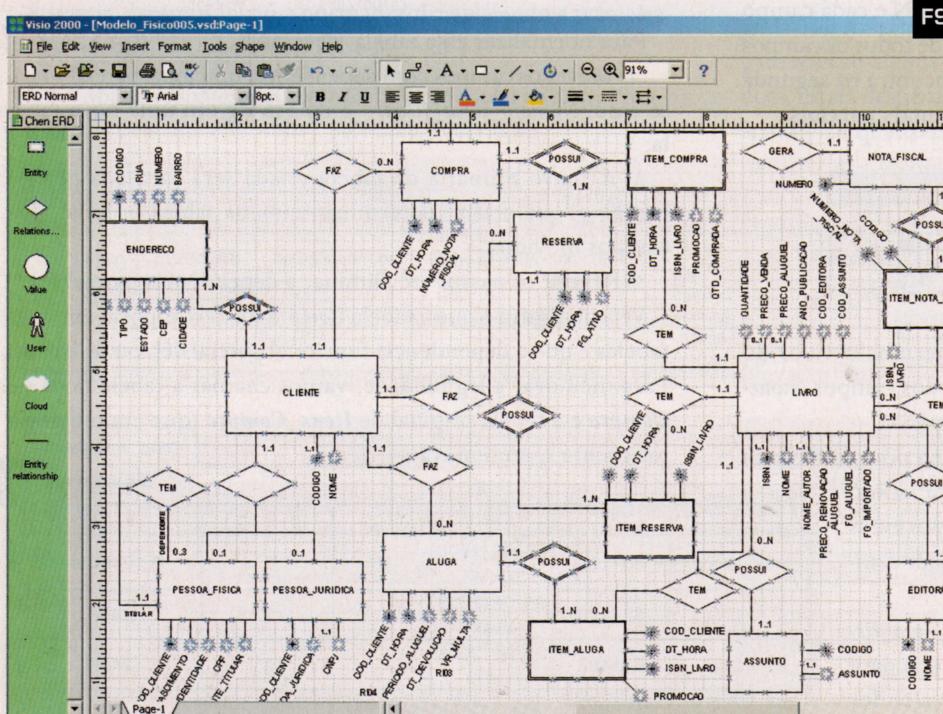
Cód_Cliente	ISBN_Livro	Dt_Hora	Periodo_Aluguel	Dt_Devolucao	Vr_Multa	Promoção
1	854135	25/06/2003 11:00	10	10/07/2003	2	5
1	624475	25/06/2003 11:00	10	10/07/2003	2	
1	631482	25/06/2003 11:00	10	10/07/2003	2	
2	952131	30/06/2003 10:00	8	02/07/2003		

Tabela Original

Cód_Cliente	ISBN_Livro	Dt_Hora	Promoção
1	854135	25/06/2003 11:00	5
1	624475	25/06/2003 11:00	
1	631482	25/06/2003 11:00	
2	952131	30/06/2003 10:00	

Tabela Nova

Cód_Cliente	Dt_Hora	Periodo_Aluguel	Dt_Devolucao	Vr_Multa
1	25/06/2003 11:00	10	10/07/2003	2
2	30/06/2003 10:00	8	02/07/2003	



F6

Dessa forma, a nova tabela (*Compra*) terá um relacionamento 1:1 com *Nota_Fiscal* e um relacionamento 1:N com a tabela original (*Itens_Compra*).

A tabela *Aluguel* também viola a 2FN. Os campos dessa tabela são determinados da seguinte forma:

Cod_cliente, ISBN_livro e Dt_hora: Compõem a chave primária;

Período_Aluguel, Dt_devolução e Vr_Multa: Violam a 2FN pois são determinados somente por *Cod_cliente* e *Dt_hora*;

Promoção: determinado por *Cod_cliente, ISBN_livro* e *Dt_hora*. Não viola a 2FN.

F7

Através da especificação levantada com o cliente na parte I desta série, o aluguel foi modelado de forma simples: mesmo que o cliente leve mais de um livro, ele é obrigado a definir um período de locação igual para todos os livros e tem que devolvê-los na mesma data (o sistema não poderá dar baixa no aluguel se algum livro estiver faltando). Se o cliente necessitar de um período de aluguel diferente para cada livro, o atendente terá que cadastrar aluguéis diferentes. Se o cliente entregar os livros depois do período definido, um valor de multa fixo será aplicado para cada livro.

F8

De acordo com essa modelagem, os campos *Período_Aluguel*, *Dt_devolução* e *Vr_multa* não precisam do campo *ISBN_Livro* para que sejam identificados. Veja um exemplo de instância da tabela *Aluguel* na figura 7.

Devemos seguir o mesmo roteiro visto na normalização da tabela *Compra* para remover a dependência funcional parcial de *Aluguel*. Veja o resultado na figura 8.

A princípio, a tabela *Reserva* não viola a 2FN pois todos os seus campos pertencem a chave primária. No entanto, temos uma errata na modelagem dessa entidade: não foi criado um atributo para indicar se a reserva está ou não em aberto. Por simplificação, a aplicação só permite uma reserva em aberto por vez para cada livro (não modelamos o conceito de “fila” de reservas).

Esse novo campo será chamado de *Fg_ativo* e terá o valor ‘fechado’ quando: i) o cliente desistir da reserva; ii) o cliente alugar a fita.

Após a criação do campo a tabela *Reserva* passa a violar a 2FN, pois *Fg_ativo* é

determinado apenas por *Cod_cliente* e *Dt_hora*. O processo de normalização é o mesmo realizado nas tabelas *Compra* e *Aluguel*. O modelo completo na 2FN pode ser visualizado na figura 9.

Terceira Forma Normal (3FN)

A 3FN procura eliminar campos de uma tabela que não dependam somente da chave primária. Em outras palavras, uma tabela estará em 3FN se já estiver em 2FN e se nenhum campo não chave depender de outro campo não chave. Outra forma de interpretar essa forma normal é dizer que não pode haver dependências funcionais transitivas.

No modelo da livraria nenhuma tabela viola a 3FN. Por isso, veremos a aplicação dessa regra em um exemplo fictício: um sistema de controle acadêmico que gerencia a sala de trabalho dos coordenadores. Nele temos que: i) um coordenador pode ser responsável por várias disciplinas e uma disciplina é de responsabilidade de um único coordenador; ii) um coordenador trabalha em apenas uma sala. Veja a tabela *Disciplina* da figura 10.

O campo *sala* possui dependência funcional transitiva, pois pode ser obtido através de *Codigo_Disciplina* (que é chave primária) ou de *Coordenador*. Para normalizar essa tabela devemos seguir o roteiro:

1) Eliminar a transitividade. Os campos que causam a dependência transitiva devem ser movidos para uma nova tabela;

2) Na tabela criada, definir uma chave primária com o(s) campo(s) que era(m) determinante(s) direto(s), na tabela original, dos campos movidos.

Veja o resultado na figura 11. Note que o campo determinante (*Coordenador*) permanece na tabela original como chave estrangeira. A tabela *Disciplina* agora está na terceira forma normal.

Desnormalização

Desnormalizar significa desfazer a normalização de uma ou mais tabelas. Normalmente o projetista utiliza esse recurso para ganhar performance nas consultas, pois a desnormalização diminui a quantidade de *joins* necessários para se obter uma informação.

O custo desse recurso pode ser alto, pois a estrutura de armazenamento das tabelas volta a ficar desordenada. Na média a desnormalização cria mais desvantagens do que vantagens e deve ser feita de forma criteriosa mesmo por projetistas experientes.

Como exemplo, vamos desnormalizar as tabelas *Telefone*, *Homepage* e *Email*. Já que elas têm praticamente o mesmo objetivo (armazenar um meio de comunicação) iremos juntá-las em uma única tabela, com o intuito de aumentar a performance na busca

Código_disciplina	Coordenador	Sala	F10
MT001	João	120	
PT001	Luiz	100	
GF005	Antônio	202	
IG005	Luiz	100	

pelas informações do cliente. A nova tabela será chamada de *Meio_Comunicacao*.

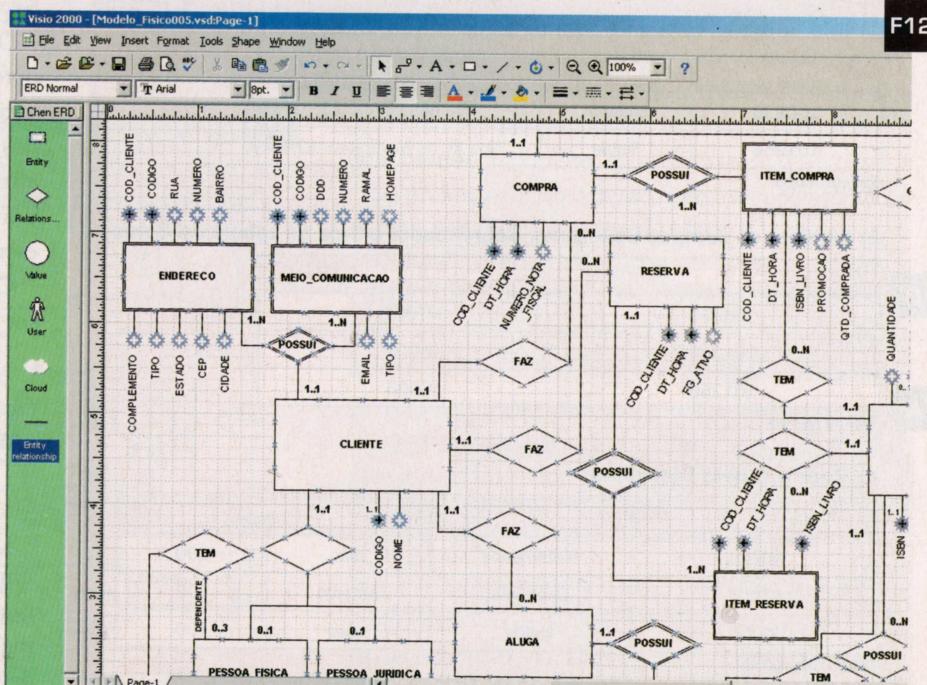
Observe que geralmente um cliente terá uma quantidade pequena de registros nessa tabela, diminuindo assim os efeitos colaterais da desnормalização. Veja na figura 12 o modelo atualizado.

Melhorando o modelo físico

O próximo passo é fazer o acabamento do modelo físico, a fim de otimizá-lo e de retirar as últimas redundâncias (nem todas as repetições podem ser eliminadas com as formas normais).

Tabela Disciplina		F11
Código_disciplina	Coordenador	
MT001	João	
PT001	Luiz	
GF005	Antônio	
IG005	Luiz	

Tabela Coordenador		
Coordenador	Sala	
João	120	
Antônio	202	
Luiz	100	



Projeto de Banco de Dados

Nesse momento não existem muitas regras a seguir e o que vale é a experiência do projetista. Veja as melhorias que podem ser feitas no modelo da livraria:

Tabelas Repetidas

Atualmente existem algumas tabelas que fazem praticamente a mesma coisa: *Item_Compra*/*Item_Nota_Fiscal* e *Compra*/*Nota_Fiscal*. De acordo com a regra de negócio da livraria, podemos unir essas tabelas sem maiores transtornos. Assim, teremos apenas um relacionamento: *Compra*/*Item_Compra* (o número da nota fiscal será armazenado na tabela *Compra*).

Mais um exemplo de 2FN

Para esclarecer ainda mais a 2FN, temos a tabela *Curso* abaixo:

Matricula	Cod_Disciplina	Nome	Endereco	Descricao	Coord	Sala
100936	IG005	Eduardo	Rua das Flores, 23	Inglês	Luis	100
101256	GF005	Felipe	Rua XV de Novembro, 12	Geografia	Antonio	202
101256	IG005	Felipe	Rua XV de Novembro, 12	Inglês	Luis	100

Note que apenas com o campo *Matricula* podemos obter as informações do aluno (nome e endereço), caracterizando a dependência funcional parcial. Observe também que apenas com o campo *Cod_Disciplina* podemos obter a descrição da disciplina, o nome e a sala do coordenador responsável. Neste caso, devemos fazer o seguinte:

1) Eliminar os campos Nome e Endereco da tabela de curso;

Matricula	Cod_Disciplina	Descricao	Coord	Sala
100936	MT001	Matemática	João	100
101256	GF005	Geografia	Antonio	202
101256	IG005	Inglês	Luis	100

Tabela Curso

2) Criar a tabela de aluno, com os campos Matricula, Nome e Endereco;

Matricula	Nome	Endereco
100936	Eduardo	Rua das Flores, 23
101256	Felipe	Rua XV de Novembro, 12

Tabela Aluno

3) Eliminar os campos Descricao, Coordenador e Sala da tabela de Curso

Matricula	Cod_Disciplina
100936	MT001
101256	GF005
101256	IG005

Tabela Curso

4) Criar a tabela Disciplina

Cod_Disciplina	Descricao	Coord	Sala
MT001	Matemática	João	100
GF005	Geografia	Antonio	202
IG005	Inglês	Luis	100

Tabela Disciplina

Observe que não estamos desnormalizando, pois as tabelas *Nota_Fiscal* e *Item_Nota_Fiscal* não foram geradas a partir de uma normalização.

Histórico

Atualmente, a informação do preço de venda/aluguel é armazenada apenas na tabela *Livro*. Dessa forma, o sistema não mantém um histórico e cria uma inconsistência virtual: se o preço de um livro for alterado, todas as compras/notas fiscais que o referenciam serão modificadas.

Como solução criaremos um campo nas tabelas *Item_Compra* e *Item_Aluguel* para duplicar a informação do preço e manter o histórico. Quando o usuário iniciar uma venda/aluguel, esse campo deverá ser inicializado com o valor do preço de venda/aluguel da tabela *Livro*. Essa regra deve ser registrada no dicionário de dados.

Dessa forma o usuário também poderá fornecer descontos durante uma venda, apesar de isso não ter sido solicitado na especificação de requisitos.

Nota: A tabela *Livro* possui dois campos referentes ao aluguel: *Preco_aluguel* e *Preco_renovacao_aluguel*. No entanto, a tabela *Item_Aluguel* receberá apenas um campo, chamado *Preco_Aluguel*. Quando o usuário cadastrar uma renovação o sistema criará um novo aluguel, inicializando o preço de cada livro com o valor de *Preco_Renovacao_Aluguel*. Esse comportamento deve ser cadastrado no dicionário de dados. O sistema não emite relatórios sobre aluguéis novos x renovações, por isso, nenhum controle adicional foi inserido.

Geração de tabelas auxiliares

O campo *Autor*, da tabela *Livro*, tem grande potencial de repetição, já que ao longo do tempo vários livros do mesmo autor podem ser cadastrados. Como medida pró-ativa, criaremos uma tabela auxiliar para armazenar os autores - essa tabela terá um relacionamento 1:N com *Livro* (um autor pode escrever vários livros).

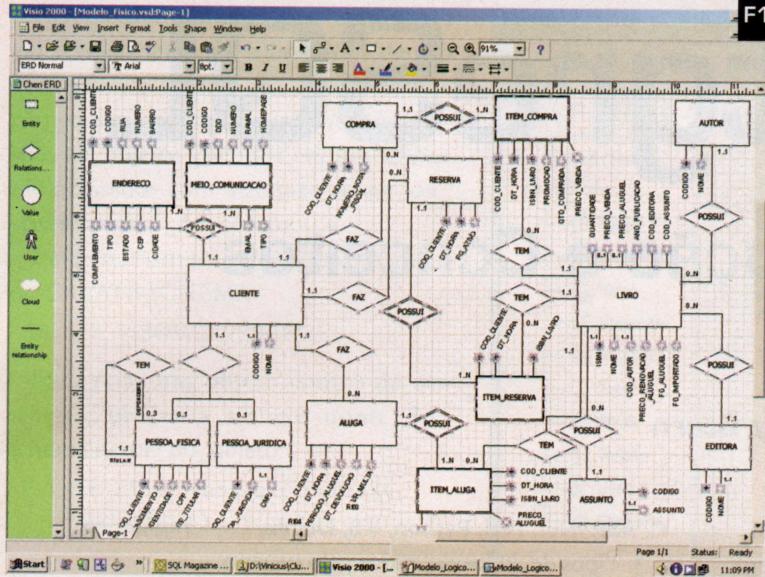
O modelo com as três alterações propostas pode ser visualizado na figura 13.

Nota: Por motivos de simplificação, desde a parte I não foi considerada a possibilidade de um livro ser escrito por mais de um autor.

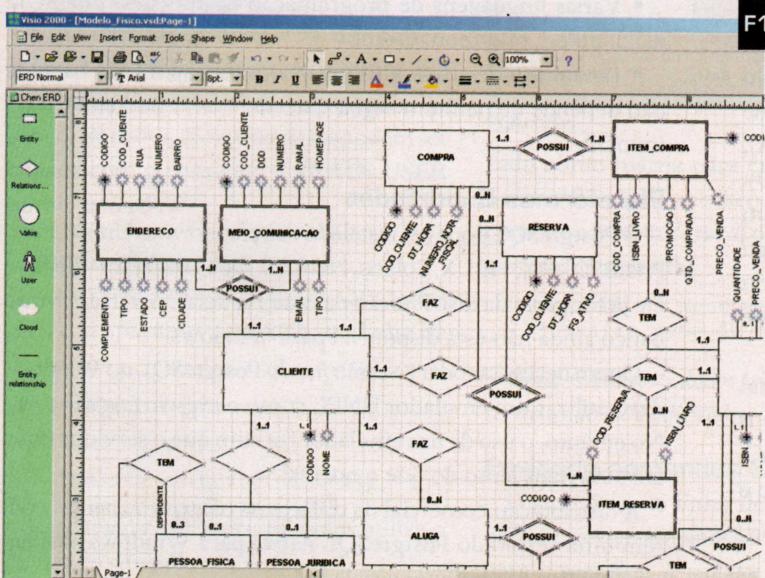
Chaves Primárias

Qualquer conjunto de campos que possa identificar um registro é referido como **chave candidata**. De todas as chaves candidatas possíveis, devemos escolher a melhor e definí-la como **chave primária**. Essa escolha deve ser baseada em dois fatores:

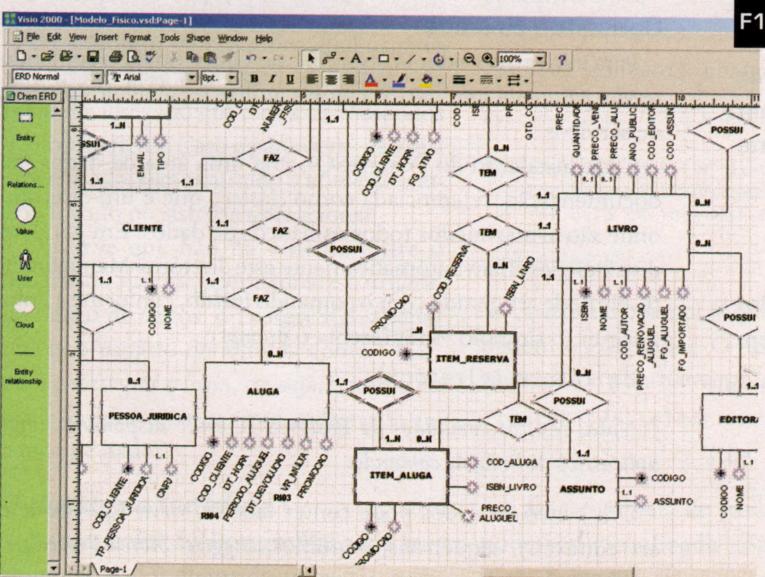
F13



F14



F15



Imutabilidade: A chave primária ideal é aquela que nunca tem seu valor alterado. De acordo com as regras de integridade, se modificarmos o valor de uma chave primária teremos que atualizar todos as chaves estrangeiras que a referenciam, criando mais um processo no sistema.

Minimização: Quanto menor o número de campos na chave primária, mais rápida serão as operações sobre a tabela.

Atualmente, a maioria dos projetistas utiliza uma chave artificial auto-incremento, conhecida como chave substituta (*surrogate key*), para compor a chave primária. Esse tipo de chave, por não fazer parte da modelagem natural do banco de dados, oferece máxima imutabilidade e minimização – ela nunca precisará ser modificada e sempre será composta por apenas um campo.

Todas as chaves primárias foram minimizadas através desse recurso. O modelo físico completo pode ser visualizado nas **figuras 14 e 15**.

Nota: Observe que as tabelas *Item_Compra*, *Item_Aluguel* e *Item_reserva* ganharam os campos *Cod_compra*, *Cod_aluguel* e *Cod_reserva*, respectivamente. Esses campos são a nova chave estrangeira do relacionamento com *Compra*, *Aluguel* e *Reserva*.

Conclusão

Chegamos ao final do modelo físico. É claro que o artigo, por mais prático que tenha tentado ser, não reflete com exatidão todos os passos que podem ocorrer na modelagem real de um banco de dados. Mesmo para aplicações simples, um banco costuma ser mais complexo e ter uma quantidade bem maior de entidades.

Esse foi o pontapé inicial – ao praticar tudo o que vimos, você descobrirá que a beleza teórica demonstrada aqui nem sempre poderá ser seguida; afinal, vivemos num mundo imperfeito.

Na próxima edição daremos início à implementação do banco de dados. Espero você lá! ■

www.sqlmagazine.com.br/sql6/down.html

Vinicius Lourenço de Sousa

(vsouza@dba.com.br) é analista de sistemas e desenvolvedor Delphi/Java, utilizando os bancos de dados Interbase, Oracle 9i e DB2/AS400, na DBA Engenharia de Sistemas. É Pós-Graduado em Análise, Projetos e Gerência de Sistemas pela PUC-RJ e possui certificação BrainBench em Delphi e RDBMS.

D

A