Universidade Federal do Piauí
Centro de Educação Aberta e a Distância

FUNDAMENTOS DE BANCO DE DADOS

Flávio Ferry de Oliveira Moreira

Ministério da Educação - MEC
Universidade Aberta do Brasil - UAB
Universidade Federal do Piauí - UFPI
Universidade Aberta do Piauí - UAPI
Centro de Educação Aberta e a Distância - CEAD

Fundamentos de Banco de Dados

Flávio Ferry de Oliveira Moreira

UFPI
2013

M838f Moreira, Flávio Ferry de Oliveira.
Fundamentos de banco de dados / Flávio Ferry de Oliveira Moreira. – Teresina : EDUFPI/CEAD, 2013.
102p.

ISBN

1. Banco de Dados. 2. Banco de Dados - Modelagem. 3. Educação a Distância. I. Título.

CDD 005.74

Apresentação

Este material didático é destinado aos alunos que chegaram ao Módulo IX, do curso de Sistemas de Informação modalidade à Distância, da Universidade Federal do Piauí, consorciada da Universidade Aberta do Piauí. Aqui serão abordados os conceitos e princípios fundamentais da área de Banco de Dados (BD).

No contexto mercadológico, a área da Informática é muito importante para todos os setores, visto que, na atualidade, os recursos tecnológicos se instalam em todas as áreas do conhecimento.

O presente material divide-se em quatro unidades, assim constituídas:

Unidade 1 - Introdução, visão geral, objetivos, vantagens e desvantagens, requisitos funcionais, modelos de BD, componentes de SBD (Sistemas de Banco de dados) e linguagens de Banco de Dados.

Unidade 2 - Modelagem e projeto de Bancos de Dados.

Unidade 3 - Modelo relacional, álgebra relacional, teoria das dependências, normalização e processamento de consultas.

Unidade 4 - Linguagem SQL e uso do MySQL e demais softwares de apoio para o aprendizado desta disciplina.

Sumário

UNIDADE 1

Conceitos e Estrutura de Banco de Dados

Resumindo

Este material destina-se a iniciar os alunos na área de Banco de Dados, promovendo esclarecimentos sobre esta poderosa forma de armazenar os valiosos itens de informação das empresas.

1 CONCEITOS E ESTRUTURA DE BANCO DE DADOS

Histórico

Nas décadas de 60 e 70 do século passado, a computação ganhou grande importância na vida das pessoas, com o processamento e armazenando de dados e informações de importância cada vez maior para todos. Percebendo esta realidade, os profissionais da área de Computação fizeram pesquisas sobre como armazenar tais dados, de forma que estes ficassem em locais seguros e de acesso bem organizado. Desta forma, houve uma evolução dos primitivos arquivos tipados para armazenagem de dados até chegar aos Bancos de Dados, os quais são constituídos de suas estruturas internas e seus métodos de acesso.

Ao longo dos anos, a evolução desta área foi instituída com técnicas cada vez mais avançadas, de modo a gerenciar o acesso simultâneo de diversos usuários aos dados, sem corrompê-los, além de promover a segurança destes e garantir a confiabilidade do Sistema de Banco de Dados como um todo.

A busca pela evolução das linguagens de programação, em nível de Banco de Dados prosseguiu, entretanto, alguns detalhes de padronização impedem que os Sistemas de Gerenciamento de Banco de Dados da atualidade tenham implementações totalmente compatíveis com as linguagens orientadas a objeto.

Evolução

Os primeiros Sistemas de Bancos de Dados foram baseados na estrutura de árvore. Seus algoritmos de organização e acesso são bem conhecidos por todos nós. A complexidade do uso deste tipo de estrutura

de dados é bastante conhecida. Basta lembrar-se da rigidez de acesso que uma árvore tem. Ao remover um nó corre-se o risco de fazer a remoção de vários dados ligados a ele. Para evitar este tipo de problema deve-se usar algoritmos de reordenação, de forma que o dado a ser removido fique em uma folha da árvore. Portanto, o uso desta estrutura para Banco de Dados mostrou-se ineficiente e ineficaz, ainda naquela época, em razão do seu baixo poder de processamento e pouca memória.

Em seguida, foram lançados os Sistemas de Bancos de Dados em Rede. Estes resolveram em parte o problema de estruturação em forma de árvores, pois eram baseados em grafos não direcionados. O grande detalhe deste modelo de dados é que grafos comuns trazem consigo uma série de problemas relacionados a problemas NP, caixeiro viajante, dentre outros. A deleção dos nós foi resolvida, todavia outros problemas mais complexos foram aparecendo, fato que tornou este modelo de Banco de Dados inviável.

Alguns outros modelos foram propostos sem sucesso ou aceitação, até que surgiu a ideia de usar base matemática para a armazenagem de dados. Surge, então, a ideia de se usar tabela para armazenar dados e controlar este armazenamento através do uso de teorias da álgebra, desta forma tendo instituído o modelo de dados Relacional, que teve grande sucesso e aceitação, sendo o mais usado no mundo inteiro até hoje, em razão de sua forte padronização. Este modelo foi proposto pelo matemático E. F. Codd, na década de 1970. Naquela época as linguagens de programação dominantes eram todas imperativas, o que levou este Banco de Dados a ter sua implementação baseada neste paradigma de Linguagem de Programação.

Na década de 1990, o paradigma de programação Orientada a Objetos ganhou muita força, cuja categoria de linguagens já vinha há mais de 10 anos, sendo proposta e aprimorada. Com isso ganhou toda a popularidade que tem hoje, sem ter ganhado força junto às entidades e pessoas que padronizam Banco de Dados. Todos concordam que Bancos de Dados Orientados a Objetos sejam úteis e necessários, porém, pouco se consegue fazer para chegar a um novo padrão especificado, para ser implementado por quem tiver interesse em comercializar o novo modelo de Banco de Dados.

Objetivos Fundamentais dos Sistemas de Bancos de Dados

Um Sistema de Banco de Dados deve fazer parte da retaguarda de um programa aplicativo desenvolvido para suprir as necessidades de uma

organização. Sob tal circunstância, imagine que seja necessário controlar o estoque de uma empresa na área de comércio ou que seja necessário informatizar uma clínica médica ou uma escola. Estes são casos típicos de Sistemas de Informação que serão desenvolvidos para serem usados em rede por vários usuários ao mesmo tempo.

Desenvolver um Sistema destes com um meio de armazenamento qualquer, que não seja um Sistema de Gerenciamento de Banco de Dados, é puro suicídio. O caos se instala facilmente na empresa e os Profissionais de Informática ficam desacreditados por lá.

A solução é implementar as regras de negócio em uma linguagem de programação, seguindo normas e regras da Interface Humano-Computador e da Engenharia de *Software*. Todavia, na hora de guardar os dados é importante não se deixar seduzir pelas facilidades de tabelas de pacotes de uso pessoal, tais como: *Paradox, Access, Dbase* e outros componentes dos *Offices* da vida. Estas tabelinhas de brinquedo são muito boas para treinar programação, mas na vida real convém usar um SGBD, para não ficar mal visto pelos seus clientes.

Vejamos as características fundamentais dos Sistemas de Bancos de Dados:

- **Reduzir ou evitar a Inconsistência de Dados** – os dados mantidos de forma consistente são fontes de confiabilidade para todos que usam o Banco de Dados ao longo de toda a sua vida útil;
- **Facilitar o acesso aos dados** – Se o usuário tiver o direito de acessar um dado, que este acesso seja possibilitado da forma mais fácil possível, pois desta forma ganha-se produtividade e tempo;
- **Evitar o Isolamento de Dados** – Dados isolados são uma praga a ser combatida. Uma tabela isolada em um esquema de Banco de Dados tende a ficar apenas ocupando espaço no disco, sem, com isso, ajudar em nada no bom funcionamento do Sistema. Uma exceção deve ser feita às tabelas auxiliares;
- **Combater as Anomalias de Acesso Concorrente** – Eis a grande dificuldade dos aplicativos pessoais de Banco de Dados (*Access, Paradox*, etc), os quais são concebidos apenas para o usuário brincar de cadastro. Quando são usados em rede, esses aplicativos ficam sem saber o que fazer com o acesso concorrente, e acabam por danificar os dados dos usuários. Um bom sistema de Banco de Dados deve controlar este acesso concorrente de maneira prática, eficiente e transparente para o usuário;

- **Garantir a Segurança** – Nem todo usuário de Banco de Dados deve ter acesso total à tudo que está armazenado. Uma boa política de segurança deve ser garantida pelo SGBD, visando preservar os dados e interesses da organização;
- **Evitar Problemas com Integridade** – Este conceito tem muita semelhança com aquele da inconsistência visto anteriormente. A consistência está relacionada com a maneira correta de se armazenar um dado. A integridade, além de ser a maneira correta de armazenar relaciona-se também com o fato deste dado ter um significado para o sistema. Integridade guarda preocupação com o significado do dado e se faz a referência correta.

Estes são os pontos fundamentais que um Sistema de Banco de Dados deve ter para começar a ser confiável, a fim de se tornar um SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS – SGBD.

Abstração de dados

O Conceito de Abstração de Dados é fundamental para o bom entendimento do funcionamento de um Sistema de Banco de Dados. Ele divide o Sistema em três camadas, a fim de facilitar o trabalho de todos os profissionais da área, sejam aqueles que desenvolvem SGBDs ou aqueles que desenvolvem aplicativos comerciais usando o SGBD.

O Primeiro nível é o FISICO – Sendo o nível mais baixo, define-se aqui o modo como os dados são fisicamente armazenados, que tipo de estrutura de dados se usa para implementá-los, bem como o tipo de algoritmo de segurança deve ser usado;

O Segundo nível é o CONCEITUAL – Neste nível são definidos quais dados são armazenados no Sistema, e de que forma eles estão relacionados. Este é o nível usado pelos Administradores de Bancos de Dados ou pelos Desenvolvedores de Sistemas de Informação;

O Terceiro e último nível é o das VISÕES – Sendo o mais alto nível de abstração, aqui os desenvolvedores mostram o que um Banco de Dados deve apresentar aos usuários. Trata-se da visão que alguns funcionários terão do sistema de informação, tal como o funcionário do caixa vai ter acesso à tela de vendas e faturamento, o vendedor vai ter acesso à tela e ao banco de realização de pedidos e romaneios, e assim sucessivamente.

A figura a seguir ilustra esta organização de um Sistema de Banco de Dados.

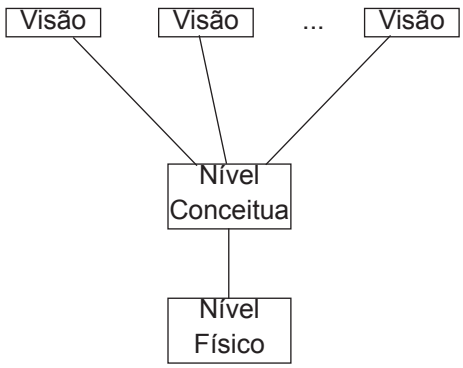


Figura 1 – Níveis de Abstração de Dados

Instâncias e Esquemas

Estes dois conceitos são muito polêmicos na área de Banco de Dados, pois muitos autores fazem inversões de seus significados. Serão apresentados aqui os significados de Instâncias e Esquemas aceitos pela maioria dos autores e também por aqueles mais respeitados. Talvez o grande problema desta inversão esteja em traduções mal feitas de textos escritos originalmente em Espanhol, já que neste idioma os significados destas palavras são oficialmente invertidos em relação ao Português.

ESQUEMA – Também conhecido como BANCO DE DADOS – refere-se ao Banco de Dados vazio recém-instalado no computador ou ao Banco de Dados apenas com o nível Conceitual definido com apenas tabelas e estruturas criadas sem dados inseridos ainda.

INSTÂNCIAS – Também conhecidas como BASE DE DADOS, referem-se aos dados que foram inseridos, estando a partir de então sob a responsabilidade do Banco de Dados. Esta é a parte que vale outro na organização. É o tesouro a ser guardado no cofre com segredos e sete chaves.

Os Sistemas Gerenciadores de Bancos de Dados são vendidos em lojas ou disponibilizados em sítios da *Internet*, podendo ser baixados de maneira livre. As Bases de Dados, por ser propriedade de alguma organização, não devem ser vendidas nem disponibilizadas em sítios aleatoriamente, pois nelas existem informações confidenciais e de propriedade de alguém ou algum projeto de trabalho de uma empresa.

Independência de Dados

Este é um conceito fundamental em Banco de Dados. A Independência de Dados deve ser mantida em dois níveis pelo Sistema de Banco de Dados: o Físico e o Lógico.

Independência Física de Dados – É a capacidade de se fazer modificações no nível físico dos dados, sem haver a necessidade de reescrever os programas aplicativos que dependem destes dados. Neste conceito é comum acontecer mudanças de versão do SGBD com novas implementações para tipos de dados já conhecidos; porém, os tipos antigos são mantidos em respeito a este importante conceito. Por exemplo: ao se lançar uma nova versão de um SGBD relacional de nome Xis, o tipo *String* é melhorado e dá origem ao novo tipo *Varchar*. Os dois tipos de dados passam a conviver pacificamente no SGBD, a partir daquela versão, em respeito ao conceito de independência física de dados.

Independência Lógica de Dados – Trata-se da capacidade de se modificar o esquema conceitual dos dados, sem precisar recompilar as aplicações que não dependam da modificação feita. Neste caso, muitas das vezes se torna necessário fazer acréscimos ou modificações de alguns campos em uma tabela do Sistema de Informação. Porém, aqueles módulos que não dependam diretamente desta alteração, não devem precisar também ser recompilados.

Linguagens de Banco de Dados

Todo Sistema de Gerenciamento de Banco de Dados deve implementar obrigatoriamente duas linguagens: uma para definir estruturas e métodos de armazenamento e outra para manipular os dados armazenados.

DDL – *Data Definition Language* – Linguagem de Definição de Dados – Esta linguagem é usada para definir estruturas dentro do Sistema de Banco de Dados. São os populares comandos de criação.

DML – *Data Manipulation Language* – Linguagem de Manipulação de Dados – Esta é a linguagem que realiza as inserções, buscas, deleções e demais operações sobre os dados armazenados pelo Banco de Dados. É a linguagem que manipula diretamente as Instâncias do SGBD.

Existem dois tipos de DML: a **procedural**, que conta com linha de comando para o usuário digitar comandos de manipulação e verificar seus resultados; **não-procedural**, que conta com uma interface que facilita a interação do usuário com o sistema, sem haver a necessidade de saber de memorização dos comandos da linguagem.

Sistema de Gerenciamento de Banco de Dados

O tão falado Sistema de Gerenciamento de Banco de Dados deve respeitar e implementar cada um destes conceitos vistos até agora, e ainda garantir mais algumas tarefas muito importantes para o seu bom funcionamento, sendo elas:

- I**nteração com o Gerenciador de Arquivos** – A intimidade com o Sistema Operacional é tão grande, que chega ao ponto do SGBD gravar arquivos no disco sem passar pelo controle do Sistema Operacional. Isto é importante para algumas implementações de segurança em Banco de Dados;
- **Cumprimento de Integridade** – Este conceito é bem mais complexo do que se possa imaginar. O SGBD deve respeitá-lo amplamente;
- **Cumprimento de Segurança** – Se as instâncias valem ouro para a organização, a segurança do Banco de Dados é fundamental para este ouro ser bem guardado. Desta forma, todo SGBD é obrigado a gerenciar usuários como se fosse um Sistema Operacional, à parte do existente na máquina;
- **Cópias de reserva (*BACKUP*) e restauração** – Todo dado valioso deve ter sua cópia em mídia de armazenamento em massa removível, tais como: uma fita, um DVD ou um *Blu-ray*. O Banco de Dados, por sua vez, deve garantir meios para se fazer cópias destes dados, periodicamente, além de restaurações sempre que necessário;
- **Controle de Concorrênci**a – Aqui vale um reforço para esta importante funcionalidade que se espera de um Sistema de Gerenciamento de Banco de Dados. É fundamental ter suporte para o acesso em ambiente concorrente, com todo o controle necessário, a fim de evitar perdas e danos em dados armazenados.

Recursos Humanos

Aqui serão definidas as pessoas que trabalham ou usam Banco de Dados em suas atividades, com a classificação e a definição de cada grupo. Classificar as pessoas que estão envolvidas com Banco de Dados é uma etapa muito importante para as definições de segurança, em nível de usuários e permissões de acesso.

Administrador de Banco de Dados

Este profissional é bastante conhecido no Brasil pela sigla em Inglês: DBA – *DataBase Administrator*. É o usuário mais poderoso do Banco de Dados da Empresa. Ele tem acesso a todas as informações que quiser, fato que pode justificar até uma remuneração diferenciada para compensar as responsabilidades que lhe são atribuídas. Além do mais, tem que ser um profissional muito comprometido com a ética, para evitar problemas dentro e fora da organização.

Dentre as atribuições do DBA pode-se destacar:

- **Definição de Esquemas** – é a atribuição responsável pelo uso da DDL no Banco de Dados, criando estruturas necessárias para fazer os devidos armazenamentos;
- **Definição de Estruturas de Armazenamento e métodos de acesso** – às vezes, são necessárias algumas estruturas além daquelas já definidas via DDL;
- **Modificação de Esquema e de Organização Física** – executa alterações nos esquemas de armazenamento e reorganiza as estruturas necessárias para garantir o bom funcionamento das aplicações;
- **Definição e Gerenciamento da Política de Usuários** – concede ou revoga direitos de acesso aos usuários de acordo com a estratégia da empresa;
- **Especificação de Restrições de Integridade** – estas regras automáticas e internas do Banco de Dados são criadas pelo DBA.

Demais usuários de Banco de Dados

A seguir, apresentaremos a lista dos demais usuários do Banco de Dados que devem ser gerenciados pelo DBA.

Tabela 1 – Classificação dos usuários de Banco de Dados

Classificação	Comentário
Programadores de Aplicativos	São os profissionais que usam muita DML, fazem requisições ao DBA para criação de estruturas e, muitas das vezes, possuem uma área do Banco de Dados destinada aos testes de seus programas.
Usuários de Alto Nível	Não são necessariamente programadores, pois sabem usar a DML para realizar suas tarefas sem, para isto, precisar criar aplicativos. São potencialmente perigosos, pois dominam a DML e suas permissões de acesso devem ser cuidadosamente definidas, a fim de evitar acessos indevidos aos dados não permitidos a estes.
Usuários Especializados	Estes usuários normalmente dominam alguma linguagem de programação específica para determinada área. Seus programas são estratégicos para algum departamento da empresa, mas não para todos, o que não justifica deixar a implementação destes a cargo dos programadores. Um exemplo comum deste tipo de usuário são os estatísticos da empresa que fazem estudos baseados em dados armazenados. Nesta equipe de usuário, normalmente é encontrado também um profissional de Informática para fazer o suporte de Banco de Dados.
Usuários Ingênuos	Esta é a categoria dominante em quantidade, pois a imensa maioria das pessoas que vão trabalhar usando um Banco de Dados não entende muito de Informática. Isto pode abranger desde os membros da diretoria da empresa até os funcionários mais humildes. São pessoas que desempenham seus papéis dominando muito pouco da tecnologia que usam. Na prática, é necessário criar algumas subclasses de usuários dentro desta categoria.

Exercícios

1. Diferencie um SBD de um SGBD.
2. Por que programas pessoais de Banco de Dados como *Access, Paradox, Dbase* e outros não podem ser considerados SGBDs?
3. Conceitue Independência de Dados em seus dois aspectos.
4. Fale da evolução dos modelos de Banco de Dados.

5. Por que os modelos baseados em árvores e em grafos não tiveram muito sucesso?
6. Defina DDL e DML.
7. Fale dos três níveis de abstração de dados.
8. Quais as atribuições do DBA?
9. Fale de cada uma das categorias de usuários de Banco de Dados.
10. Discuta sobre a expressão que diz que: “O SGBD é o verdadeiro cofre da empresa. É nele que está armazenado o verdadeiro ouro da organização”.

UNIDADE 2

Modelo Entidade-Relacionamento

Resumindo

Neste capítulo são abordados os aspectos de projeto de um Banco de Dados Relacional, através do Modelo Entidade-Relacionamentos. Para que um Banco de Dados Relacional seja criado corretamente, é fundamental saber fazer projetos lógicos e físicos. Nesta unidade são estudados os projetos lógicos. Os projetos físicos fluem naturalmente após os estudos das duas unidades seguintes, principalmente após a teoria de Normalização de tabelas e comandos de SQL.

2 MODELO ENTIDADE-RELACIONAMENTO

Conceitos

Estudo de Caso (Descrição de Minimundo) - Texto que descreve detalhes levantados a respeito da situação e dos procedimentos a serem modelados para Banco de Dados. O analista deve fazer entrevistas com as pessoas interessadas em encomendar o sistema para identificar que tipo de dado é relevante para a programação e para o Banco de Dados.

Observação:

A partir deste ponto os nomes que serão mapeados para o Banco de Dados serão grafados sem acentuação e sem cedilha. Esta prática evita problemas de implementação e execução.

Entidades (Conjunto de Entidades) - É um elemento da vida real, na situação a ser modelada para o Banco de Dados, que pode ser distinguível dos demais, estando sempre representadas no plural. No Estudo de Caso, normalmente, as entidades têm ligações com os substantivos mais relevantes do texto.

Relacionamentos (Conjunto de Relacionamentos) - São as ligações que devem existir entre as entidades. Uma vez que foram destacados os substantivos que levarão às entidades, os relacionamentos são os verbos que estão ligados a estes substantivos.

Ex.: (...) Os Alunos podem fazer apenas um curso por vez. (...)

Tem-se neste trecho duas entidades: ALUNOS e CURSOS. O relacionamento entre estas entidades é PODEM FAZER.

Atributos - São qualificações pertencentes às entidades. Quando se faz necessário um relacionamento ter atributos, este será elevado à categoria de entidade associativa, conforme será visto posteriormente. No texto, os

atributos são qualificadores como adjetivos, advérbios e outros.

Ex.: (...) Cada aluno é identificado por sua matricula, além de nome, endereco, telefone. (...)

Cada valor grifado é um atributo.

Chave - Atributo especial que tem a função de identificar o elemento dentro do conjunto. Uma chave pode ter diversas classificações:

- **Chave Primária (eleita)** - Atributo principal de identificação. É ÚNICA para cada elemento da entidade. Sua existência é obrigatória para qualquer entidade.
- **Chave Candidata** - Atributo que tem a mesma funcionalidade da chave primária, porém, neste momento, não está exercendo a função de chave primária, pois está sendo usado como atributo comum.
- **Chave Estrangeira** - Atributo que é chave primária em outra entidade, sendo importado para garantir a ligação entre as duas entidades.
- **Superchave** - Conjunto formado pela análise conjunta das chaves Primária e Candidata.
- **Chave Composta** - Quando a chave é formada por mais de um atributo ao mesmo tempo.

Exemplos:

Imagine a entidade ALUNOS.

O campo matrícula é a identificação de cada aluno, portanto, a **chave primária**. No Brasil existe um documento de numeração única usado pela Receita Federal, o CPF (CNPF), antigo CIC. O CPF seria, neste exemplo, uma **chave candidata** do aluno. Para aplicação de alguns conceitos de Bancos de Dados que serão vistos mais adiante, a **superchave** desta tabela seria matrícula e CPF.

No relacionamento existente entre ALUNO e CURSO, a chave primária de CURSOS seria copiada para dentro da entidade ALUNOS, com a função de **chave estrangeira**.

ATENÇÃO PARA AS NOTAÇÕES!

Na lista dos atributos de uma entidade, a chave primária ganha um asterisco ao lado de seu nome: matrícula*. Sem o asterisco, pode aparecer apenas sublinhada: matrícula.

Uma chave estrangeira aparece com a sigla CE (ou FK) acima de seu nome. Exemplo: *codCursoCE*.

Restrições de Mapeamento (Cardinalidades)

As ligações entre entidades promovidas pelos relacionamentos precisam ser quantificadas. Para tanto, existem as cardinalidades, que podem se dividir em:

- Um-para-um – um elemento de uma entidade está ligado a apenas um elemento da outra entidade.
- Um-para-muitos – um elemento de uma entidade está ligado a diversos elementos da outra entidade.
- Muitos-para-muitos – vários elementos de uma entidade estão ligados a um grupo de elementos da outra entendidade.
- Zero associado a um ou muitos – quando a ligação entre uma entidade e outra é opcional. Neste caso diz-se que a entidade ligada ao ZERO ou um, ou ao ZERO ou muitos, é conhecida como ENTIDADE FRACA.

Notações para os diagramas

As notações dos Diagramas de Entidades e Relacionamentos (D E-R) são muito diversificadas, pois na década de 80 não havia padronizações de Engenharia de *Software*, circunstância em que permitia que cada autor criasse a sua própria notação.

Serão expostas neste ponto do material as duas principais e mais aceitas notações: a de Peter Chen (criador do D E-R), e a de James Martin (conhecida como notação da Engenharia de Informação ou pé-de-galinha – pé-de-corvo em inglês).

Apresentando a notação para cada elemento:

ENTIDADES – Em todos os casos são representadas por um retângulo com o nome destas no seu interior.

Figura 2 – Uma entidade

ENTIDADES FRACAS – Semelhantes às entidades normais, aqui se faz também um retângulo, mas com os cantos arredondados e contorno tracejado.

Figura 2.1 – Uma entidade fraca

ATRIBUTOS – São pequenos balões ligados às entidades.

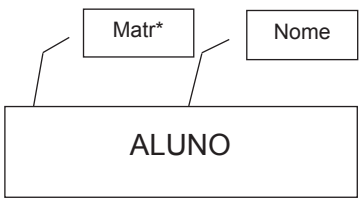
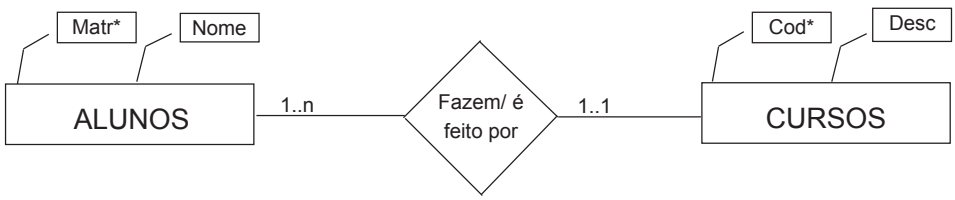


Figura 2.2 – Uma entidade com atributos

RELACIONAMENTOS – Aqui começam as diferenças. Serão exibidos relacionamentos nos dois formatos citados.

Vejamos:

Peter Chen



James Martin

Figura 2.3 – As duas principais notações de relacionamentos

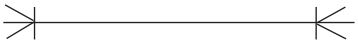
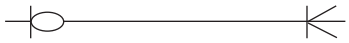
Observe que, ambas as notações, são importantes enfatizar o verbo do relacionamento e o sentido de sua leitura. Ademais, têm-se diferenças nas

notações de cardinalidades que são próprias de cada notação.

CARDINALIDADES – Serão listadas aqui as diferenças de notação de cardinalidades entre as notações vistas.

Observe a tabela a seguir e perceba essas diferenças.

TABELA 2 – DIFERENÇAS DE CARDINALIDADES

Tipo	**James Martin**	**Peter Chen**
Um-para-um		1..1 1..1
Um-para-muitos		1..1 1..m
Muitos-para-muitos		m..n m..n
Quantidades fixas	NÃO TEM	1..5 2..3
Zero ou um-para-muitos		0..1 1..m
Um-para-zero, um ou muitos		1..1 0..m

Combinações de cardinalidades dentro de uma mesma notação são permitidas.

Atenção: Recomendado nunca misturar notações distintas em um mesmo Diagrama Entidade-Relacionamentos.

Relacionamentos Especiais

Alguns tipos de relacionamentos não se encaixam no padrão visto até agora. Dentre estes se destacam a GENERELIZAÇÃO e a AGREGAÇÃO.

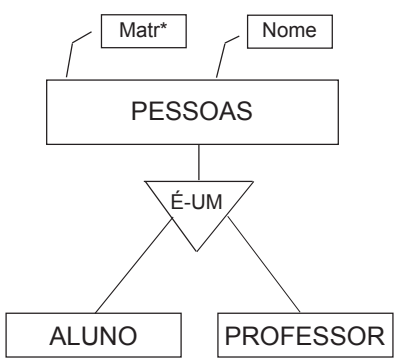


Figura 2.4 – Notação de generalização/especialização

Generalização

Também conhecido como especialização, este tipo de relacionamento é muito importante quando uma entidade contém o geral que deve se tornar específica, ao ser usada a primitiva “é-um” ou “é-uma”. Segue uma ilustração que melhora a percepção deste conceito.

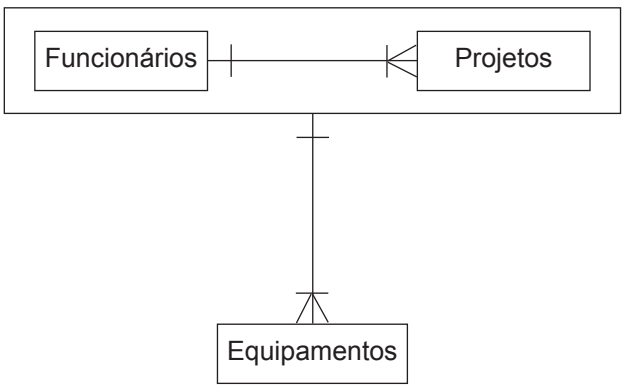


Figura 2.5 – Notação de agregação

Neste caso, a entidade PESSOAS contém os dados que são comuns a alunos e professores, sendo que os dados específicos de cada um destes se encontram nas respectivas especializações.

Agregação

Uma limitação do modelo Entidade-Relacionamentos diz respeito à quantidade de elementos envolvidos em cada relacionamento. Em cada ponta de um relacionamento deve haver apenas uma entidade. Sendo assim, deve-se relacionar sempre duas entidades em cada relacionamento.

Em alguns estudos de casos podem ocorrer limitações como: “um funcionário que trabalha em um determinado projeto é responsável por um dado equipamento”. Neste caso fica clara a necessidade de fazer um relacionamento ternário (que não é permitido) entre FUNCIONÁRIOS, PROJETOS e EQUIPAMENTOS. Para resolver este problema existe a agregação.

Resolvendo este caso, faz-se a agregação entre funcionário e projeto ligando estes dois aos equipamentos relacionados, conforme ilustra a figura a seguir.

Desta forma, percebe-se que é possível adaptar um relacionamento ternário ao M E-R através do recurso da agregação, pois o relacionamento

que está agregado passa a ser considerado pelo modelo, como sendo uma grande entidade em relação à terceira entidade envolvida.

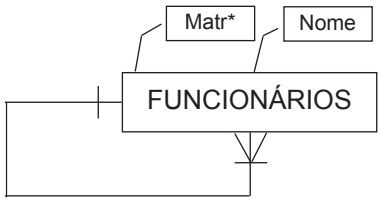


Figura 2.6 - notação de autorrelacionamento

Autorrelacionamento

Em muitos casos, é necessário relacionar uma entidade consigo mesma. Estes são os conhecidos autorrelacionamentos. Um exemplo prático pode ser visto a seguir, onde, no caso, dentre todos os funcionários, um deles é o chefe. Assim, o relacionamento deve sair da entidade e voltar para a mesma, seja qual for a notação usada.

Um Exemplo Resolvido

Estudo de caso:

Uma empresa tem vários funcionários que são lotados em departamentos. Um funcionário está em apenas um departamento de cada vez. Ao funcionário é dada a opção de ter dependentes. Cada departamento é responsável por um projeto da empresa. Os dados mais importantes de funcionário são: matrícula, nome, CPF, endereço e telefone. Os dependentes devem ter nome e código. Cada departamento tem código, sigla e nome completo. Os projetos possuem código, data de criação e prazo de execução. Pede-se que seja feito o diagrama de entidades-relacionamentos deste caso.

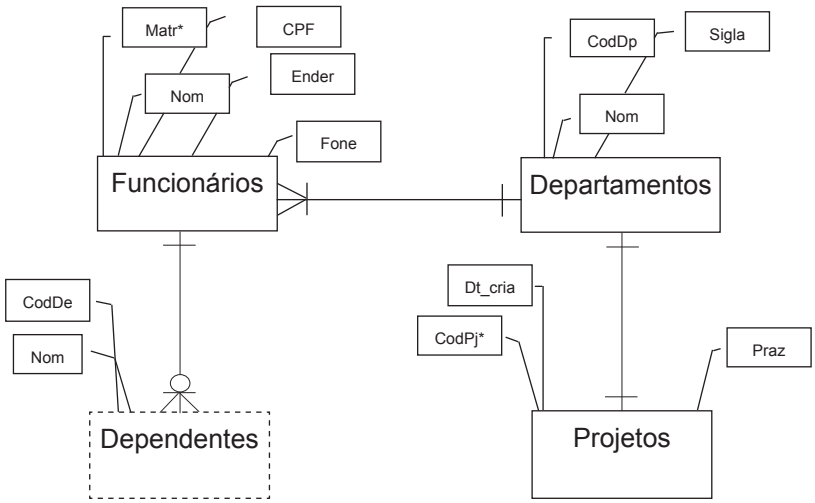


Figura 2.7 – Solução do exemplo proposto

Exercícios

1. Conceitue os elementos básicos do modelo entidade-relacionamentos, mostrando as notações de cada um deles.
2. O que se entende por cardinalidade?
3. Em que situações são usadas as entidades fracas?
4. Quando as agregações são necessárias?
5. Uma notação de generalização deve ser usada em qual caso
6. Faça o Diagrama Entidade-Relacionamentos do estudo de caso a seguir: A empresa de organização de festas infantis XPTO está precisando de um sistema para gerenciar suas rotinas de trabalho. Deve-se fazer um cadastro de clientes contendo (CPF, nome, endereço, telefone e observações). Os temas de festa também devem ser gerenciados pelo sistema com os dados: código, descrição, quantidade de alegorias, dimensão da mesa, dimensão do painel e preço da diária. O contrato do evento relaciona o cliente com o tema da festa, e conta ainda, com os seguintes dados: código, data de contratação, preço total, forma de pagamento, sinal, parcelas, data da festa, data de devolução e status.
7. Uma empresa faz o gerenciamento de seu pessoal. Todos são cadastrados no sistema. Cada funcionário pode ter dependentes, como também pode optar por um plano de saúde ofertado por uma entidade conveniada com a empresa. Todo funcionário está alocado em um

departamento onde trabalha com vários colegas. Existe apenas um chefe para todos. Cada departamento é dividido em equipes que são responsáveis por um ou mais projetos da empresa. Modele um diagrama entidade-relacionamento para este caso.

8. Em uma clínica são atendidas diversas especialidades médicas. Os convênios são classificados em dois grupos: SUS e demais convênios (que incorpora o particular). Quando o paciente solicita determinada especialidade, é marcada uma consulta ligando este ao médico que vai atendê-lo. Quando o paciente não sabe qual especialidade precisa, é enviado a uma consulta de enfermagem, para ser então marcada a consulta com o especialista. Tanto a consulta de enfermagem quanto à do médico pode gerar ou não requisição de exames. Apresente o Diagrama de Entidades e Relacionamentos deste caso.
9. A empresa de transportes 'Corisco' precisa de um sistema de acompanhamento de cargas e entregas. É função deste, gerenciar o cadastro de veículos que trafegam com um ou dois motoristas, dependendo da distância marcada na rota. Cada veículo possui uma capacidade de carga diferente. As entregas são feitas para pedidos de clientes cadastrados. Os produtos podem ser classificados nas categorias perecíveis e não perecíveis, e isto influencia no tipo de veículo usado no transporte. Apresente o Diagrama de Entidades e Relacionamentos deste caso.
10. Apresente o Diagrama de Entidades e Relacionamentos deste caso: um hotel precisa de um sistema de controle. Os hóspedes telefonam e fazem reservas fornecendo o CPF, nome completo, endereço de residência, cidade, estado, telefone fixo, celular, data de chegada e data prevista de partida. São reservados quartos dentro de uma das categorias disponíveis, com seus respectivos preços de diárias. Cada quarto pertence a uma categoria. Nas reservas que se concretizam ficam agregadas também as despesas de restaurante e de frigobar, detalhando os produtos e pratos consumidos.

UNIDADE 3

Modelo Relacional

Resumindo

Este modelo de banco de Dados foi escolhido para ser estudado nesta etapa do material por que é o modelo dominante no mundo há mais de 25 anos. O Banco de Dados Orientado a Objetos ainda não se firmou como um modelo amplamente aceito no mundo, nem sequer foi padronizado para tal, apesar de as linguagens de programação terem evoluído do paradigma imperativo para as linguagens orientadas a objetos. O Modelo relacional de Dados surgiu da ideia de melhorar as estruturas de suporte aos dados existentes nos Bancos de Dados da época dos modelos: Hierárquico e em Rede. A ideia fundamental foi escapar dos grafos e árvores e seus algoritmos complexos e alguns até sem solução.

3 MODELO RELACIONAL

Origens e Conceitos

O Modelo Relacional foi criado por Codd em 1970, com forte embasamento matemático. Ideias baseadas na álgebra e na teoria dos conjuntos, têm por finalidade representar os dados como uma coleção de relações, onde cada relação é representada por uma tabela. Quando uma relação é pensada como uma **tabela** de valores, cada linha nesta tabela representa uma coleção de dados relacionados, conhecidos por **registro**.

Já para cada coluna da tabela dá-se o nome de **atributo**, com o mesmo significado de atributo visto no capítulo anterior. Estes valores podem ser interpretados como fatos descrevendo uma instância de uma entidade ou de um relacionamento. Os nomes da tabela e de suas colunas são utilizados para facilitar a interpretação dos valores armazenados. Todos os valores em uma coluna são necessariamente do mesmo tipo.

Na terminologia do modelo relacional, cada tabela é chamada de **relação**; uma linha de uma tabela é chamada de **tupla**; o nome de cada coluna é chamado de **atributo**, e o tipo de dado que descreve cada coluna é chamado de **domínio**.

O conceito de **chave** é válido aqui também e tem as mesmas características e classificações já conhecidas.

As linguagens DDL e DML deste modelo são fundidas em uma só, que é conhecida atualmente pelo nome de **SQL *(Structured Query Language)***. Isto porque no início faltava definir as linguagens deste modelo, quando a IBM lançou seu SGBD no mercado, o **DB/2**, ainda hoje muito popular e usado. Este produto da "*big blue*" veio com linguagens DDL e DML inovadoras e unidas numa só, a **SeQueL**. Daí houve uma grande revolução dentre aqueles que pesquisavam na área de Banco de Dados, no final dos anos 70 do século XX.

A linguagem criada pela IBM passou a ser adotada como o padrão para todo e qualquer Banco de Dados Relacional. Apenas o nome foi trocado pela hoje tão popular sigla de SQL, isto, pelo fato de **sequel** parecer palavra associada a coisas ruins em alguns idiomas, inclusive em português, pois se assemelha a **sequela** que significa coisa ruim.

O que tem de tão bom na linguagem SQL é que a teoria de Banco de Dados Relacionais define a álgebra relacional e suas operações, como sendo a forma de se fazer criação de estruturas e manipulação de dados dentro deste modelo. A linguagem criada pela IBM simplesmente implementa em nível de programação, cada operação da referida álgebra. Desta forma, a teoria foi totalmente respeitada, nascendo um novo padrão mundial em computação.

Finalmente, garantir o sucesso total do modelo só falta definir mecanismos para garantir a confiabilidade dos dados, preservando sua integridade e mantendo o seu real sentido. O embasamento teórico desta característica tão importante fica por conta da **Teoria das Dependências**, também conhecida por **Normalização**.

Normalizar um Banco de Dados significa aplicar um conjunto de regras às tabelas e aos seus atributos, de maneira que estes fiquem divididos dentro da forma que mais preserve seus significados e valores, evitando a redundância e o isolamento dos dados, garantindo o acesso facilitado àqueles que tiverem o devido direito de tê-lo.

Logo adiante, neste mesmo capítulo, serão detalhadas as chamadas **CINCO FORMAS NORMAIS**, que são as regras para aplicar a normalização em um Banco de Dados. Neste ponto, faz-se necessário, primeiramente, um estudo de álgebra relacional e SQL básica.

Álgebra Relacional

Esta é uma linguagem de consulta procedural que será detalhada em suas operações fundamentais e avançadas a partir deste ponto. Já existem diversas evoluções de versão, tanto da álgebra relacional, quanto da SQL. A seguir, serão vistas as operações mais populares desta dupla.

a) Operação Seleção (σ)

Esta operação permite selecionar em uma tabela registros que

atendam a um determinado critério. Seu símbolo é a letra sigma minúscula. A sua sintaxe é mostrada logo abaixo:

σ (campoX = valor) (nomeTabela)

Observe que o critério aparece subscrito ao símbolo da operação; entretanto, o nome da tabela deve aparecer no mesmo nível da operação. Para a formulação do critério pode-se usar qualquer campo da tabela, juntamente com qualquer símbolo de comparação válido, tal como: =, <>, < , >, <=, >= , além do valor a ser comparado ao campo condizente com o domínio do campo, ou seja, para um campo inteiro deve-se fazer comparações com valores inteiros.

Tomando a tabela a seguir como exemplo:

Nome da tabela

Alunos

Tabela 1: Alunos

Matr*	Nome	Endere	Fone	CodCurso^{CE}
123	Flávio José	Rua Pedro, 2290	5859-6930	01
205	Aline Maria	Rua Pereira, 224	7452-8520	01
330	José Luiz	Rua Pedro, 450	3358-9665	02
440	Clarita Silva	Rua Riachuelo, 177	6598-8754	05

Fazendo a operação de seleção para o nome de aluno igual a 'Aline Maria' obtém-se:

σ (alunos.nome = 'Aline Maria') (Alunos)

205	Aline Maria	Rua Pereira, 224	7452-8520	01

Observe que todo o registro é retornado pela operação. Esta mesma operação em SQL seria escrita assim:

```
SELECT * FROM alunos
WHERE
    Alunos.nome = 'Aline Maria'
```

Algumas considerações a respeito desta cláusula SQL:

1. Os termos destacados em maiúsculas são apenas para chamar atenção para as palavras reservadas da linguagem, pois esta não é *'case-sensitive'*;

2. Tanto em álgebra relacional, quanto em SQL, ao citar um campo de tabela, é opcional colocar antes o nome da tabela e usar o ponto para referenciar o campo. Esta é uma boa e elegante prática de programação, pois ajuda na leitura e compreensão do algoritmo;
3. Especificamente para campos *string* e assemelhados, a SQL possui a possibilidade de usar o critério de comparação LIKE associado ao caractere coringa %. Deste modo, poder-se-ia fazer seleção de todos os alunos cuja rua tivesse, por exemplo, a *string* 'Pedro' na sua denominação. Segue o exemplo:

```
SELECT * FROM alunos
WHERE
    Alunos.endere LIKE '%Pedro%'
```

Isto retorna registros cujo endereço contenha 'Pedro' em qualquer posição:

123	Flávio José	Rua Pedro, 2290	5859-6930	01
330	José Luiz	Rua Pedro, 450	3358-9665	02

b) Operação Projeção (π)

Esta operação simbolizada pela letra grega pi minúscula e lista apenas alguns campos da tabela. É útil quando, em alguns resultados, torna-se importante omitir certos valores em cada registro da tabela. A sintaxe da operação é a seguinte:

π (campo_1, ..., campo_n)(nomeTabela)

Como exemplo, utilize a tabela anteriormente descrita para fazer a projeção dos campos de matrícula e de nome dos alunos.

π (matr, nome)(Alunos)

Matr*	**Nome**
123	Flávio José
205	Aline Maria
330	José Luiz
440	Clarita Silva

Em SQL, a cláusula para obter este mesmo resultado seria:

SELECT Matr, Nome *FROM* alunos

Observa-se que:

1. O comando *SELECT* não deve ser traduzido como seleção, pois serve para implementar várias operações da álgebra relacional;
2. Para implementar a projeção, faz-se a substituição do tradicional coringa * após o *select* pela lista de campos a serem projetados;
3. Por questões de elegância, o nome da tabela pode também ser usado antes de cada referência aos seus campos.

c) Operação Produto Cartesiano (T1 x T2)

Nesta operação que envolve duas tabelas acontece a inevitável volta ao passado na vida de todos. Volta-se aos tempos da escolinha, onde a professora, tia para alguns, desenhava dois diagramas de Venn cheios de elementos e dizia que o Produto Cartesiano dos dois conjuntos significava apenas ligar cada elemento do primeiro conjunto a TODOS os elementos do segundo conjunto.

Além disso, certamente não havia utilidade para esta operação de conjuntos em sua vida. Portanto, aprenda agora e ensine depois para a sua ex-professora a grande utilidade desta operação.

No universo relacional é correto que cada tabela tenha seus devidos dados e que fiquem relacionadas entre si, direta ou indiretamente, através de suas chaves primárias e estrangeiras. Às vezes, faz-se necessário ligar temporariamente duas tabelas para satisfazer uma consulta, sendo necessário para isto utilizar o produto cartesiano. Veja abaixo, um exemplo de duas tabelas relacionadas e seu produto cartesiano:

Tabela 2 - Alunos

Matr*	Nome	Endere	Fone	CodCurso^{CE}
123	Flávio José	Rua Pedro, 2290	5859-6930	01
205	Aline Maria	Rua Pereira, 224	7452-8520	01
330	José Luiz	Rua Pedro, 450	3358-9665	02
440	Clarita Silva	Rua Riachuelo, 177	6598-8754	05

Tabela 3 - Cursos

Codigo*	**Descricao**	**cargaHoraria**
01	Informática	120
02	Veterinária	220
05	Agronomia	190

Onde Alunos.codCurso é referente a Cursos.codigo.

A sintaxe do produto cartesiano é:

Tabela_1 x Tabela_2

Sendo assim, para realizar o cruzamento das informações de alunos e cursos, seria: **Alunos x Cursos**, obtendo-se ligações do tipo:

123	Flávio José	Rua Pedro, 2290	5859-6930	01	01	Informática	120

Porém, esta operação conta com um grande problema: as ligações indevidas que originam as chamadas tuplas-fantasmas, ou seja, como existem três cursos, o aluno 'Flávio José' e todos os demais apareceriam ligados aos seus cursos originais, e também aos outros cursos dos quais não fazem parte, desse modo, originando informações inverídicas, conforme demonstrado na tabela a seguir:

Tabela 4

123	Flávio José	Rua Pedro, 2290	5859-6930	01	02	**Veterinária**	220
123	Flávio José	Rua Pedro, 2290	5859-6930	01	05	**Agronomia**	190

Para resolver este tipo de problema usa-se uma operação de seleção conjugada ao produto cartesiano. Esta seleção é conhecida como **seleção de filtro**, pois fica responsável em mostrar apenas a verdade. A operação seria refeita da seguinte maneira:

$\sigma_{(Alunos.codCurso = Cursos.codigo)}$(Alunos x Cursos)

E o resultado seria o correto:

Tabela 5

Matr*	Nome	Endere	Fone	CodCursoCE	Codigo	Descricao	cargaHoraria
123	Flávio José	Rua Pedro, 2290	5859-6930	01	01	Informática	120
205	Aline Maria	Rua Pereira, 224	7452-8520	01	01	Informática	120
330	José Luiz	Rua Pedro, 450	3358-9665	02	02	Veterinária	220
440	Clarita Silva	Rua Riachuelo, 177	6598-8754	05	05	Agronomia	190

Esta combinação de operações é tão necessária que se define como uma operação avançada da álgebra relacional, tendo exatamente este significado: a JUNÇÃO NATURAL, simbolizada por: ⋈

Então, a expressão anterior poderia ser substituída pela junção natural:

Alunos ⋈ **Cursos**

obtendo-se assim o mesmo resultado.

Em SQL a expressão para realizar esta junção natural é:

SELECT * *FROM* alunos, cursos
WHERE
Alunos.codCurso = Cursos.codigo

Conceito: União-Compatível

Diz-se que duas tabelas são união-compatíveis quando ambas têm a mesma quantidade de campos com a mesma sequência de domínios, ou seja, para uma tabela de três campos, sendo: um inteiro, um *varchar* (*string*) e um lógico. A outra tabela terá obrigatoriamente: um inteiro, um varchar e um lógico para ser união-compatível. Algumas operações da álgebra relacional são definidas sobre este conceito, tais como: união, diferença e intersecção.

d) União de Tabelas (T1 U T2)

Esta operação é definida entre duas tabelas união-compatíveis que consiste em juntar os conteúdos de duas tabelas em uma só.

A Sintaxe desta operação é:

Tabela1 U Tabela2

Segue um exemplo deste caso.

Clientes

Cod*	Nome	Fone
01	Flávio	9966-6363
02	Aline	9494-4477

Funcionários

Matr*	NomeF	FoneF
120	Ana	8855-2121
121	José	8101-8585

A operação Clientes U Funcionarios resulta em:

Cod*	Nome	Fone
01	Flávio	9966-6363
02	Aline	9494-4477
120	Ana	8855-2121
121	José	8101-8585

Em SQL usa-se a palavra reservada UNION, para unir duas consultas e obter resultado para esta operação.

e) Diferença de Tabelas (T1 – T2)

Desta vez, a operação também é definida entre duas tabelas união-compatíveis. A semelhança fica ligada à diferença de conjuntos, pois no resultado ficam os dados que estão em uma tabela, mas não estão na outra. Segue um exemplo desta operação:

Clientes-Poupança

Cod*	Nome	Fone
01	Flávio	9966-6363
02	Aline	9494-4477

Clientes-ContaCorrente

Matr*	NomeF	FoneF
120	Ana	8855-2121
121	José	8101-8585
01	Flávio	9966-6363

Fazendo a operação *Clientes-Poupança - Clientes-ContaCorrente*, obtém-se:

02	Aline	9494-4477

Ou seja, o cliente que tem conta-poupança, mas não tem conta-corrente.

f) Intersecção de Tabelas (T1 ∩ T2)

Esta operação é definida entre tabelas união-compatíveis, retorna algo semelhante à intersecção de conjuntos; ou seja, tudo aquilo que é comum às duas tabelas.

Retomando o exemplo da operação anterior, pode-se investigar o cliente que tem conta-poupança e conta-corrente. Para tanto, usa-se a operação *Clientes-Poupança* ∩ *Clientes-ContaCorrente*, obtendo-se o seguinte resultado:

01	Flávio	9966-6363

Normalização – Teoria das Dependências

Este é o tópico que encerra o capítulo de “Modelo Relacional”, onde os conceitos iniciais situam o leitor neste universo; a álgebra relacional mostra como operar com valores nestas tabelas, e a normalização finaliza com as regras que devem ser obedecidas para que os dados sejam armazenados e manipulados da maneira correta, de acordo com a teoria dos criadores do modelo.

Desse modo, vale lembrar que tudo começa com um bom projeto, o qual deve ser elaborado com base no capítulo anterior que mostrou o “Modelo Entidade-Relacionamento”.

Para que se possa fazer um Banco de Dados Relacional eficiente, deve-se aplicar cada uma das Cinco Formas Normais, a saber:

Primeira Forma Normal (1NF)

Esta regra consiste em verificar se todos os atributos que compõem a tabela são atômicos, ou seja, são atributos indivisíveis.

Caso algum atributo não-atômico seja encontrado, deve-se fazer sua decomposição em atributos menores e indivisíveis.

Existe uma exceção para esta forma normal, que é o atributo endereço, devido a uma polêmica existente entre diversos autores de Banco de Dados, onde alguns consideram que endereço é uma grande *string*, e outros, porém, entendem que endereço seja não-atômico e divisível em seus componentes, como: tipo de logradouro, nome do logradouro, número e complemento.

Neste material, as duas apresentações deste campo serão consideradas pertinentes, de acordo com a primeira forma normal.

Segunda Forma Normal (2FN)

Diz-se que uma tabela está na segunda forma normal se estiver na primeira forma normal e TODOS os atributos dependerem da chave por completo. Explicando melhor, se a chave da tabela for composta, não devem existir atributos que dependam apenas de uma parte desta chave e mesmo que esta não seja composta, é importante que todos os atributos presentes sejam dependentes desta.

Segue um exemplo:

Itens-Pedidos

codPed*	codProd*	quant	pre_total	qt_min	data_ped
10	25	2	123,00	58	07-04-2009

Observe que a chave é composta pelos atributos codPed e codProd que são na realidade duas chaves estrangeiras, onde codPed aponta para a tabela Pedidos, e codProd para a tabela Produtos.

Os atributos quant e pre_total, por sua vez, têm dependência total da chave composta. Porém, o atributo qt_min significa a quantidade mínima que este produto deve ser mantido em estoque, tem dependência de Código de Produto e não de Código de Pedidos, assim como data_ped tem dependência em relação a Código de Pedidos, e não em relação a Código de Produtos.

Neste caso, para aplicar a segunda forma normal deve-se levar qt_min para a tabela dos Produtos e data_ped deve pertencer à tabela dos Pedidos. Concluindo, espera-se que a nova tabela Itens_Pedidos tenha o seguinte aspecto:

codPed*	codProd*	quant	pre_total
10	25	2	123,00

Terceira Forma Normal (3FN)

Esta forma normal introduz o conceito de Dependência Transitiva que consiste em um atributo depender de outro atributo, e este depender diretamente da chave primária.

Para aplicar a 3FN, a tabela deverá estar na 2FN e as dependências transitivas devem ser corrigidas.

Alternativamente, os pesquisadores Boyce e Codd fizeram uma redefinição da Terceira Forma Normal, ao afirmarem que todo atributo deve ser uma consequência direta da superchave da tabela. Por isso, esta forma normal também é chamada de **Forma Normal Boyce-Codd**.

Segue um exemplo de tabela que precisa da aplicação da Terceira Forma Normal:

Alunos

Matr*	Nome	Cd_curso	Desc_curso	Carga_hor
01	Flávio José	120	Aplicações Financeiras	200
02	Aline Maria	125	Engenharia de *Software*	72

Observe que Desc_curso e carga_hor são atributos que dependem de cd_curso, e este, por sua vez, depende da chave primária matr.

Para solucionar este problema faz-se:

1. Os atributos que dependem transitivamente da chave primária são levados para outra tabela juntamente com aquele que os determina;
2. O atributo que determinou os demais fica na tabela original na condição de chave estrangeira.

Sendo assim, a tabela do exemplo ficaria normalizada da seguinte forma:

Alunos

Matr*	Nome	Cd_cursoCE
01	Flávio José	120
02	Aline Maria	125

Cursos

Cd_curso	Desc_curso	Carga_hor
120	Aplicações Financeiras	200
125	Engenharia de *Software*	72

Onde Alunos.cd_curso refere-se a Cursos.cd_curso. Desta forma, as tabelas estão normalizadas e respeitando a 3FN.

IMPORTANTE!
Mais de 95% dos casos da vida real têm sua normalização finalizada na Terceira Forma Normal.

Quarta Forma Normal (4FN)

Apesar de a maioria dos sistemas se normalizarem até a 3FN, esta forma também é muito importante devido ao tratamento de mais um tipo de dependência, a Dependência Multi-Valorada, que consiste em uma tabela que tenha ao menos três atributos e cuja chave seja composta por três atributos, que serão referenciados por A, B e C.

Neste caso, o atributo A relaciona um conjunto de valores do atributo B e o atributo A relaciona um conjunto de valores do atributo C, cujos conjuntos de valores de B e de C não têm necessariamente a mesma quantidade de elementos.

Neste caso, podemos dizer que A multidetermina B e A multidetermina C.

Em simbologia matemática: A ➔ ➔ B e A ➔ ➔ C consequentemente A ➔ ➔ B|C

Como exemplo, suponha que uma empresa tenha entre seus funcionários, alguns com diversas especializações e fluência em um grupo de idiomas, sabendo que existem tabelas com funcionários, especialidades e idiomas e outra tabela que relaciona as três, a qual vai nos interessar no exemplo a seguir:

Fun-esp-lin

Matrícula*	cdEspecialid*	cdIdioma*
120	02	01
120	03	03
120	03	04
121	01	01
121	06	02

Veja que o funcionário está determinando quantas especialidades e quantos idiomas estão a ele relacionados. Para resolver este problema de multivaloração deve-se fazer a divisão desta tabela em duas, de modo que fiquem agrupados os atributos (A,B) e (A,C). Portanto, aplicar a Quarta Forma Normal neste caso seria fazer:

Fun-esp

Matrícula*	cdEspecialid*
120	02
120	03
120	03
121	01
121	06

Fun-lin

Matrícula*	cdIdioma*
120	01
120	03
120	04
121	01
121	02

Assim, resolve-se o problema da Dependência Multi-Valorada encontrado neste caso.

Quinta Forma Normal (5FN)

Algumas vezes, quando a 4FN é aplicada, existem regras a serem respeitadas (restrições do caso) que obrigam a fazer o teste da junção para validar se dados incompatíveis não serão associados após a decomposição imposta pela 4FN.

Se existirem tais regras e a junção resultar em associações indevidas de valores, fica caracterizada a Dependência de Junção, que é um caso especial de multivaloração, originando-se a partir daí a Quinta Forma Normal.

Para resolver uma Dependência de Junção, deve-se em primeiro lugar desfazer a 4FN retornando a tabela à sua condição original quando foi aplicada a 3FN, devendo-se, em seguida, decompor a tabela em três.

Observe que este é o único caso onde uma tabela será decomposta em três e não em duas. A maneira correta de fazer a decomposição é: (A,B), (A,C) e (B,C).

Deixando mais clara a aplicação desta forma normal, veja a seguir um exemplo de um controle de vôos.

Imagine que os pilotos conduzam aviões e estes façam determinadas rotas. Porém, devido a restrições em alguns aeroportos, aviões de porte maior não podem fazer operações de pouso e decolagem nestes.

Segue a tabela na 3FN:

Voos

matPiloto*CE	**refAviao***CE	**Trecho***CE
01	737-300	THE-REC
01	A320	REC-RIO
02	A330	RIO-BSB
02	FKK-100	BSB-THE

Imagine que o aeroporto em restrição seja o de Teresina, que, em razão de ter sua pista reduzida e muitas residências nas proximidades, não é recomendável que aviões do porte dos A320 e A330 pousem neste aeroporto; porém, os pilotos que conduzem estas aeronaves também levam aviões de porte menor com destino a Teresina.

Aplicando a quarta forma normal neste caso, é arriscado o sistema selecionar um piloto com um *AirBus* rumo a Teresina, o que seria um desrespeito à restrição do caso.

Para solucionar este problema aplica-se a Quinta Forma Normal, deixando as tabelas da seguinte forma:

Vôo1

matPiloto*CE	**refAviao***CE
01	737-300
01	A320
02	A330
02	FKK-100

Vôo2

matPiloto*CE	**Trecho***CE
01	THE-REC
01	REC-RIO
02	RIO-BSB
02	BSB-THE

Vôo3

refAviao*CE	**Trecho***CE
737-300	THE-REC
A320	REC-RIO
A330	RIO-BSB
FKK-100	BSB-THE

Desta forma, resolve-se 100% dos problemas que envolvem Bancos de Dados Relacionais. A 5FN sempre é realizável, porém, nem sempre é necessária.

Exercício

1. Discuta por que os Bancos de Dados Relacionais se tornaram um padrão respeitado mundialmente.
2. Como o uso do Modelo Entidades-Relacionamentos colabora com um bom projeto de Banco de Dados Relacional?
3. Qual a ligação entre álgebra relacional e SQL?
4. Qual a importância da normalização das tabelas?
5. Observe a tabela:

ALUNOS

Matr*	Nome	Endere	Fone
1	José	Rua 12,12	3251-1414
2	Pedro	Av. 95, 189	3366-3635
3	Ana	Av Rosas, 10	3458-9587

6. Apresente os predicados da álgebra relacional que se pede:
   a) Selecione o Aluno de matrícula 2.
   b) Projete o nome e o telefone de cada aluno.
   c) Mostre apenas o nome e o endereço de José.
7. Num cadastro de pessoas que contenha: CPF*, Nome, Endereço, Bairro, CEP, Cidade e Estado, suponham que seja necessário fazer uma atualização do campo CEP, sendo um valor único para todos que morem no bairro 'Ininga'. Como você tentaria fazer um predicado SQL para tal operação?
8. O comando *SELECT * FROM* CLIENTES, PEDIDOS *WHERE* CLIENTES. CODCLI= PEDIDOS.CODICLI implementa qual operação da álgebra relacional?
9. Nas duas tabelas relacionadas abaixo, como fazer para executar uma Junção Natural em SQL?
   Alunos (matri*, nome, endere, fone, cod_cursoCE)
   Cursos (codigo*, descri, cargahor)
   Alunos.cod_curso refere-se a Cursos.codigo
10. Mostre como seriam representadas as operações de junção natural e de produto cartesiano referentes ao comando de SQL usado na questão anterior.
11. Um plano de saúde funciona na modalidade cartão de descontos para consultas. Os clientes são cadastrados fornecendo: CPF, nome, endereço,

data de nascimento, telefone, local de trabalho, endereço de trabalho, telefone do trabalho e celular. Cada cliente é ligado a um contrato que tem: número, a data de contratação, data de validade, dia do vencimento e valor. O cliente pode cadastrar até quatro dependentes, bastando fornecer o nome e a data de nascimento de cada um destes. Para cada indivíduo é fornecida uma carteirinha a ser apresentada em consultórios médicos conveniados no ato das consultas. O sistema também mantém ligado aos contratos, o cadastro de médicos disponíveis para os clientes e dependentes. Os médicos têm: CPF, nome, especialidade, endereço do consultório e horário de atendimento.

Apresente o esquema de Banco de Dados já normalizado para este caso, indicando até que forma normal foi aplicada.

12. Estudo de Caso:

Uma empresa de recursos humanos precisa fazer um sistema de gerenciamento de currículos e vagas. As empresas clientes entram em contato e informam a quantidade de vagas que dispõem, inclusive, com a qualificação requerida para o candidato. Os candidatos preenchem uma ficha com dados pessoais e entregam *Curriculum Vitae* com suas qualificações. Os funcionários da empresa fazem os devidos cruzamentos de informações para selecionar aqueles que serão chamados para a entrevista, e provavelmente para as vagas ofertadas. O sistema deve acompanhar todo este processo.

Faça o esquema de Banco de Dados para esta aplicação e mostre-o normalizado.

13. Observe a tabela a seguir.

Matr*	Nome	Endere	Fone	Curso	Turma	CargaHor	Turno
1	José	Rua 12,12	3251-1414	Ciências	T02	120	M
2	Pedro	Av. 95, 189	3366-3635	Física	T01	180	T
3	Ana	Av Rosas, 10	3458-9587	Tec. Veter	T01	125	M

Apresente a normalização deste banco de dados, e informe até que forma normal foi usada. Justifique o uso de cada uma das formas normais nesta tabela.

14. Imagine um caso onde seja necessário contratar funcionários que tenham diversas especializações e dominem alguns idiomas além de Português. Sabendo que o funcionário é identificado por sua matrícula numérica, como fazer para criar corretamente a tabela que vai associar

o funcionário com suas especialidades e os idiomas que este domina? Lembre-se da Teoria das Dependências.

15. Fale dos problemas que podem ocorrer com um Banco de Dados Relacional não-normalizado.

UNIDADE 4

Práticas com os *Softwares* de Apoio

Resumindo

Nesta unidade será tratada a parte prática da disciplina, onde serão abordados programas para fazer modelagem de estudos de casos, uso da linguagem SQL dentro do SGBD MySQL, os quais todos os programas aqui utilizados são do tipo freeware ou open source. Também estão todos entre os mais populares do mercado de trabalho atual.

4 PRÁTICAS COM OS SOFTWARES DE APOIO

Considerações Iniciais

Os programas a serem abordados podem ser baixados na *Internet*, a partir dos sítios de seus produtores ou de alguns outros sítios que distribuem programas livres. A localização dos endereços é extremamente simples, através do sistema *Google* (), bastando apenas digitar o nome do programa no campo de busca.

Estes arquivos também serão disponibilizados pelo sítio do ensino à distância da Universidade Aberta do Piauí/Plataforma *Moodle*.

Eis a lista dos programas a serem instalados para o devido acompanhamento desta unidade:

1. **MySQL *Workbench* 5.0 OSS** – Este programa serve para fazer modelagem conceitual e física de Bancos de Dados Relacionais, onde seu SGBD alvo é o MySQL. Neste programa se faz Diagramas Entidades-Relacionamentos e gera-se os scripts SQL para execução no SGBD MySQL. Alternativamente a este programa pode-se trabalhar com o ***BDDesigner* Fork 1.4**;
2. **MySQL 5** – Este SGBD deve ser instalado em sua máquina, recomendando instalação típica. Na configuração, enquanto seus conhecimentos de BD não evoluírem mais um pouco, recomenda-se que esta seja feita na base do avançar em todas as telas. Porém, é importante definir uma senha para o root, que é o usuário principal deste SGBD, pois deixá-lo com senha vazia é abrir uma importante brecha na segurança de sua máquina quando entrar na *Internet.*

   Use uma senha fácil de memorizar e de lembrar, pois precisaremos dela em nossas práticas;
3. **DB *Manager*** – Esta ferramenta serve de interface gráfica para

executarmos comandos no MySQL de forma mais cômoda e didática.

Usando o MySQL Workbench 5.0 OSS

Considerando que este programa já foi instalado, execute-o a partir de seu atalho criado no *menu* Iniciar, grupo MySQL.

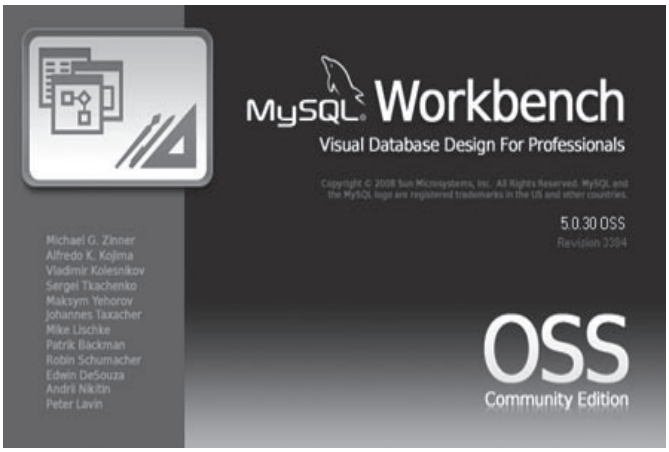


Figura 4 – Tela de boas vindas do programa

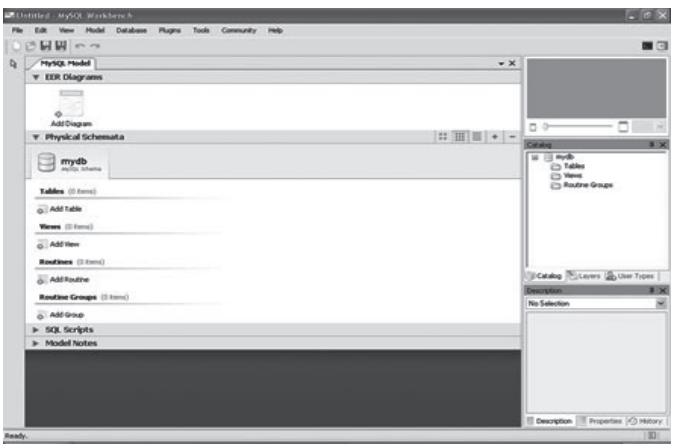


Figura 4.1 – Tela inicial do *Workbench 5.0*

Para começar a trabalhar, clique na opção “Add Diagram” na aba EER Diagram. O trabalho seguirá na tela mostrada abaixo:

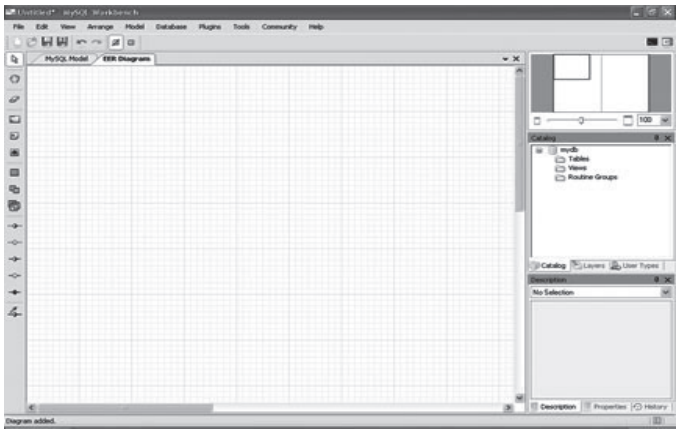


Figura 4.2 – Tela de construção do diagrama E-R

Figura 4.3 – Barra de Ferramentas do diagrama E-R

Observe a figura 4.3, pois aqui estão todas as ferramentas que serão usadas no diagrama de exemplo. Permita a liberdade deste autor de rotacionar à figura das ferramentas que originalmente aparecia na vertical.

Contando de cima para baixo, usaremos inicialmente a sétima ferramenta (*place a new table*). Clique nesta ferramenta e na área de trabalho.

Veja a figura a seguir:

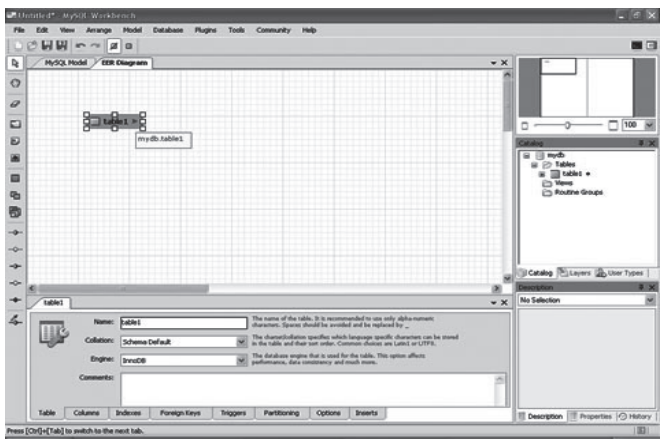


Figura 4.4 – Definindo as propriedades da tabela (entidade)

Agora, usaremos a parte inferior da tela definindo nome para a tabela, como sendo “Alunos”, sem as aspas.

Na aba *colmuns*, definiremos os campos da tabela (atributos), criando:

- matricula, tipo INT e marque Primary Key;
- nome, tipo *varchar*(60) e
- email, tipo *varchar*(45).

Sua tabela deverá ficar com o aspecto a seguir:

Figura 4.5 – Formato da tabela após definições

Agora repetiremos o procedimento criando uma tabela chamada cursos, com os seguintes atributos:

- codigo, tipo INT, *Primary Key*;
- descricao, tipo *varchar*(45) e
- cargaHoraria, tipo INT.

O grupo de botões que fica logo abaixo da tabela de ferramentas serve para definir os relacionamentos. Na legenda que aparece ao se posicionar o mouse, existem informações vitais sobre o relacionamento: sua cardinalidade **1:1, 1:n** ou **m:n**, e se seu tipo é identificador ou não-identificador.

Aqui fica um esclarecimento importante: relacionamento identificador é aquele cuja chave estrangeira chega para fazer parte da chave primária, ou seja, os relacionamentos mais comuns são os não-identificadores, aonde a chave estrangeira não chega para fazer parte da chave primária.

Um ponto relevante deste programa é a aplicação automática de parte da normalização, o qual ao se usar um relacionamento **m:n** entre duas tabelas, a entidade (tabela) associativa entre as duas é criada automaticamente.

Para ligar as duas tabelas criadas, usa-se um relacionamento não-**identificador, 1:n**, clicando-se primeiro em alunos e depois em cursos. Assim, o aluno poderá fazer um curso e o curso será feito por vários alunos. Em caso

de erro, basta clicar no relacionamento errado com a ajuda da ferramenta borracha.

A figura a seguir mostra como deve ficar a ligação:

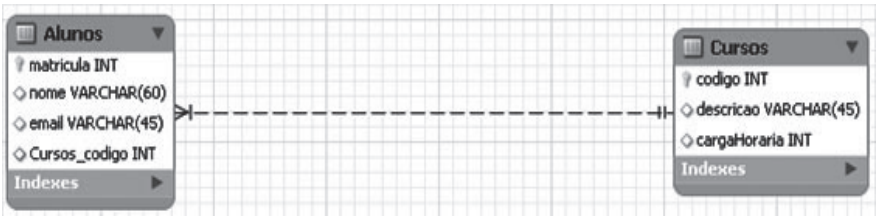


Figura 4.6 – Relacionamento do exemplo

Usando os menus é possível fazer diversas operações neste programa, tais como salvar o diagrama e exportar o SQL. Quando a opção tiver a sigla SE ao lado, ela estará disponível apenas na versão paga do programa. Infelizmente imprimir é uma destas opções, daí o semelhante DB*Designer Fork* ser também recomendado nesta fase de modelagem. Seu uso é parecido com o programa atual e sem restrições ou limitações.

Salve o seu diagrama, depois acesse novamente o menu ***File***, opção ***Export → Forward Engineer SQL Create Script***... Em ***output file*** informe o nome de seu arquivo, podendo-se usar o botão **browse** para indicar uma pasta para conter o arquivo. Vamos trabalhar com "**C:\saida.sql**" sem as aspas. Clique em avançar e deixe marcada a opção *MySQL table*. Finalize clicando em ***Finish***. Confira o arquivo no disco **C:** de seu computador, ele pode ser aberto no bloco de notas ou em algum editor equivalente, que é usado, por exemplo, para escrever programas em sua linguagem favorita.

Usando o DB *Manager*

Agora o *script* precisa ser adicionado ao seu servidor de Banco de Dados. Para tanto, no menu Iniciar, procure o grupo DBTools *software* e clique no atalho do DB *Manager*. Será exibida a tela a seguir:

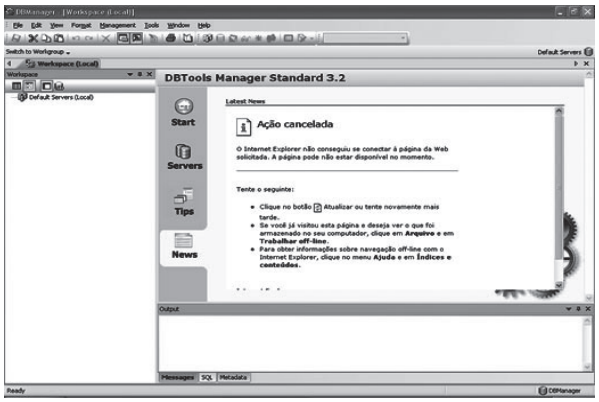


Figura 4.7 – Tela inicial do DBManager

A primeira providência a ser tomada neste programa é configurar a conexão com o Servidor de BD. Para tanto, usa-se o menu *Tools* – opção *Server* – opção *Server Manager*.

A seguinte tela é mostrada:

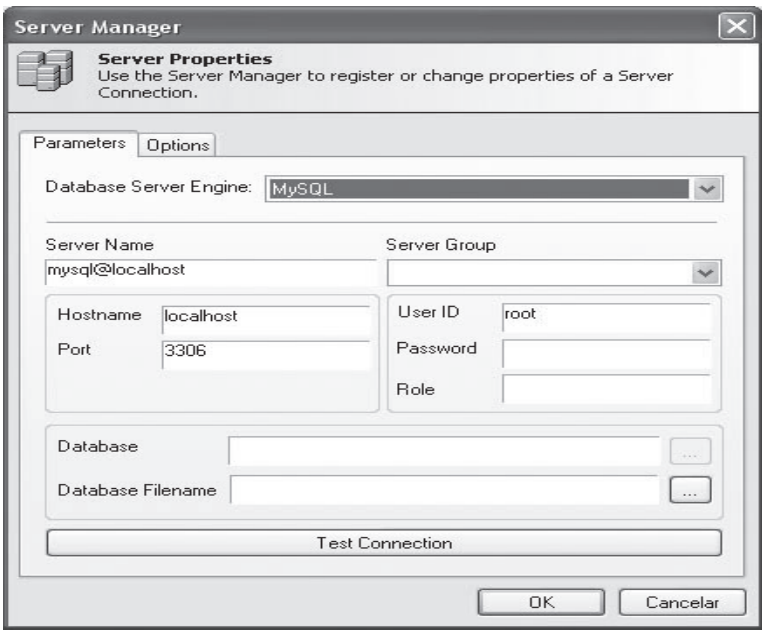


Figura 4.8 – Configurando a conexão

Se seu MySQL é local, ou seja, foi instalado como "*developer*", as opções devem ser mantidas como estão, lembrando apenas de informar a senha do usuário root, que foi aquela que você definiu. Clique no botão "*Test Connection*" e observe se a conexão foi um sucesso com a respectiva versão do MySQL. Caso contrário, algo deverá estar errado. Verifique então, as instalações e as opções usadas.

Se o servidor estiver em outra máquina na rede, pergunte ao administrador quais são os parâmetros do MySQL, o nome do host e a porta onde este funciona.

Figura 4.9 – Conexão realizada com sucesso

Clique em OK e vamos prosseguir com a prática.

Procure na tela principal o sinal + que fica ao lado da expressão "***Default Servers***". Expanda os demais sinais de + até chegar ao **mysql@ localhost**. Clique em ***databases*** e, ao lado, use a opção ***create a database***. Informe o nome ***mydb*** e dê OK na tela.

Neste passo, vamos importar o *script* criado com o programa anterior. Clique com o botão direito em ***mydb*** na lista da esquerda da tela. Escolha ***new query***. Na tela que abrir use a opção "***open from disk***" e indique seu arquivo de SQL. Seu *script* deverá aparecer elegantemente na tela. Aqui você pode adicionar novos comandos de SQL ou alterar o *script* criado pelo programa anterior ou finalmente clicar no botão **EXECUTE**.

O resultado com sucesso aparece da seguinte forma:

```
Executing Query. Wait ...
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
```

```
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
Query 'saida' Statistics (Time: 00:00:01, Records: 0, Fields: 0)
Affected Rows: 0
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
Query 'saida' Statistics (Time: 00:00:00, Records: 0, Fields: 0)
Affected Rows: 0
```

Isto quer dizer que está tudo bem até agora.

Veja a seguir a estrutura do workspace com as tabelas criadas.

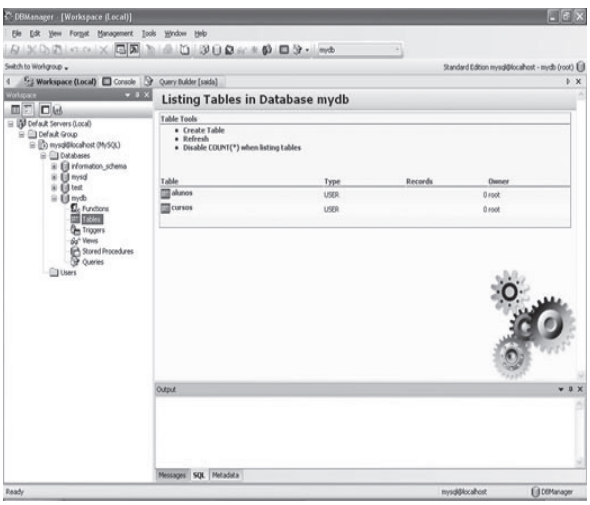


Figura 4.10 – Tabelas criadas com sucesso

Agora vamos povoar estas tabelas. *Use new query* em mydb novamente e digite o seguinte *script*:

Insert into cursos *values* (01,'computação',120);
Insert into cursos *values* (02,'vendas',200);
Insert into cursos *values* (03,'secretariado básico',200);
Insert into alunos *values* (101,'José Silva','js@uapi.br',01);
Insert into alunos *values* (102,'Ana Braga','ab@globo.com',02);
Insert into alunos *values* (103,'Larissa','soletrando@pe180.com.br',03);

Não se esqueça de clicar no botão EXECUTE. Limpe o *script* e teste os conteúdos com o famoso:

SELECT * *FROM* Alunos; (clique em EXECUTE).

Depois com:

SELECT * *FROM* Cursos; (clique em EXECUTE).

Pronto! O pontapé inicial em sua vida de Profissional de Tecnologias da Informação (TI) foi dado. Agora que já sabemos usar Banco de Dados, é só praticar.

O restante deste capítulo vai se dedicar a algumas dicas sobre MySQL, onde, para isto, recomendamos aprofundamentos através de documentação do MySQL e apostilas disponíveis na *Internet*.

Tópicos sobre MySQL

Para melhor entendimento deste tópico recomendamos uma consulta ao apêndice A, existente no final deste capítulo.

Trata-se de uma apostila muito bem conceituada no sítio do apostilando, escrita por autores de muita experiência no uso e suporte ao MySQL.

Exercícios

1. Fazer todos os estudos de caso das unidades 2 e 3 nos programas estudados nesta unidade.

APÊNDICE - A

MySQL – Tutorial Básico

Este texto foi adaptado a partir do tutorial escrito por Rafael V. Aroca - rafael@linuxqos.cjb.net, disponível em http://www.apostilando.com.

Esta seção destina-se ao estudo do SGBD MySQL, desenvolvido pela TcX *DataKonsultAB*. O programa está disponível para *download* em sua versão original em www.mysql.com, em inglês, e também em www.mysql.com.br para sua versão brasileira, onde se encontram também, projetos de tradução e documentação do MySQL, em português.

Este SGBD, além de oferecer vários recursos não existentes em outros, tem a vantagem de ser totalmente gratuito para uso, tanto comercial, quanto privado, em conformidade com a licença pública GPL.

As principais metas da equipe de desenvolvimento do MySQL é construir um servidor rápido e robusto.

Os recursos acima mencionados incluem:

- Capacidade de lidar com um número ilimitado de usuários;
- Capacidade de manipular mais de cinquenta milhões (50.000.000) de registros;
- Execução muito rápida de comandos, provavelmente o mais rápido do mercado;
- Sistema de segurança simples e funcional.

São usuários do MySQL:

- *Silicon Graphics* (www.sgi.com)
- *Siemens* (www.siemens.com)
- *Yahoo* (www.yahoo.com)
- IFX *Networks* (www.ifx.com.br)
- Dezenas de Web *hosting* e Provedores, devido ao enorme sucesso que o MySQL vem fazendo.

Algumas bases de dados chegam a mais de 200GB. A lista completa pode ser consultada em www.mysql.com.

Operação Básica

Supondo que o MySQL esteja devidamente instalado e funcionando, seguem-se os comandos mais importantes para seu uso diário. Estes comandos podem ser usados via MySQL *Command Line Client* ou via um

programa de interface gráfica para MySQL, como o DB *Manager.*

Começando pelo *login* em linha de comando:

---

mysql -u flavio -p supersenha

Sintaxe:
mysql -u (usuário) -p (senha)
ou
mysql -h servidor -u (usuário) --password=(senha)

Após verificação de senha aparecerá algo como:

Cliente MySQL - NBS

Bem vindo ao monitor do MySQL. Use ; após os comandos ou \g.

Sua *id* é 49, e você está conectado a um servidor versão 3.21.23-beta-log

Digite '*help*' para ajuda.

mysql>

Uma vez no *prompt* do MySQL, podemos começar a utilizar os comandos do MySQL e manipular os dados e o servidor. Contudo, antes de modificar a base de dados, nós devemos escolher qual desejamos usar, da seguinte forma:

---

mysql>use mydb;

Base de dados alterada.

Troque mydb pelo nome da base de dados desejada. Você obterá uma mensagem confirmando a alteração da base de dados. Isto significa que você está conectado a ela.

Repare que todo comando será seguido de ponto e vírgula, pois como em C, a maioria dos comandos do MySQL são sucedidos por ponto e vírgula.

Antes de fazer qualquer coisa, seria interessante você consultar a ajuda, que listará os comandos disponíveis neste momento da execução do MySQL. Isto deve ser feito através do comando *help*.

```
---------------------------------------------------------------
mysql>help
...
---------------------------------------------------------------
```

Muitas destas funções não serão usadas a princípio, mas algumas merecem destaque e atenção especial neste momento, sendo elas: *status, connect, clear, e qui*t, que serão usadas com mais frequência e por isso serão facilmente lembradas.

Tabelas e campos

Neste ponto, fica claro toda a estrutura de um SGBD relacional, onde são definidas as Bases de Dados que vão conter Tabelas com seus Registros divididos em Atributos, e seus respectivos Domínios.

Observando tudo isto de fora, podemos formar o seguinte esquema:

Base de dados ➔ Tabela ➔ Registro ➔ Coluna (domínio)
Banco de dados ➔ Tabela ➔ Linha ➔ Campo

Os campos podem ser de diferentes tipos e tamanhos, permitindo ao programador criar tabelas que satisfaçam o escopo do projeto. A decisão de quais campos usarem e quais não usar é muito importante, pois influi drasticamente na *performance* da base de dados que estamos desenvolvendo, sendo portanto, um conhecimento sólido destes conceitos.

Tais conceitos são parte fundamental de uma base de dados. É no atributo campos que as informações ficam armazenadas. Para um melhor aproveitamento da base de dados, antes de utilizar, deve-se definir os campos que se deseja usar e especificar o que cada um pode conter.

Neste momento, a modelagem mostra-se vital para o bom

desempenho do projeto. Tabelas mal projetadas, não-normalizadas e com tipos mal escolhidos podem arruinar todo o sistema de informação.

Tipos de campos no MySQL

O MySQL oferece os mais comuns campos. Alguns deles estão aqui listados:

CHAR(M)

Os campos CHAR são usados para caracteres alfanuméricos, como endereços e nomes. Seu tamanho é fixo e instaurado ao ser criado. Um campo do tipo CHAR pode ter de 1 a 255 caracteres.

Exemplo:
endereco_comercial CHAR(10);

Define um campo chamado 'endereco_comercial', que pode conter até dez letras.

Observe que não há acentos no nome do campo, pois muitos servidores não acentuam e sua tabela teria difícil acesso.

VARCHAR(M)

Sua estrutura e funcionamento são idênticos ao campo anterior, salvo que no tipo *CHAR*, o tamanho definido é fixo e mesmo não usado, aquele espaço em disco é alocado.

Já o campo *VARCHAR*, aloca apenas o espaço necessário para gravação. Contudo, vale lembrar que trocamos espaço por velocidade, pois este campo é 50% mais lento que o anterior.

Exemplo:
endereco_comercial *VARCHAR*(10);

Define um campo chamado endereco_comercial que pode conter até dez letras. Se você preencher apenas duas, o campo não ocupará todos dez bytes, mas apenas dois.

INT(M) [*Unsigned*]

O campo INT, como o próprio número diz, suporta o conjunto dos números inteiros numa variação de -2147483648 a 2147483647. O parâmetro *Unsigned* pode ser passado, excluindo os números negativos, proporcionando um intervalo de 0 até 4294967295.

Exemplos:

carros_estocados INT;

Inteiro válido: -245
Inteiro válido: 245
Inteiro inválido: 3000000000

carros_estocados INT unsigned;

Inteiro válido: 245
Inteiro inválido: -245
Inteiro inválido: 3000000000

***FLOAT*[(M,D)]**

Os pontos flutuantes (*FLOAT*) representam pequenos números decimais, e são usados para representar números com maior precisão.

Exemplo:
voltagem_cadeira_eletrica *FLOAT*(4,2);

Float válido: 324.50

DATE

Campo usado para armazenar informações referentes à data. A forma padrão é 'AAAA-MM-DD', onde AAAA corresponde ao ano, MM ao mês, e DD ao dia. Ele pode variar de 0000-00-00 a 9999-12-31.

O MySQL possui um conjunto poderoso de comandos de formatação e manipulação de datas. Consulte documentação adicional, se houver interesse.

Exemplo:
data_de_nascimento DATE;

Data válida: 1999-12-06
Data inválida: 1999-06-12

TEXT/BLOB

Os campos texto e *blob* são usados para guardar grandes quantidades de caracteres, já que, podendo conter de 0 a 65535 *bytes*, são úteis para armazenar documentos completos, como este que você está lendo.

A única diferença entre os campos *BLOB* e *TEXT*, está no fato de um campo *TEXT* não ser sensível a letras maiúsculas e minúsculas quando uma comparação é realizada e os BLOBs sim.

Exemplo:

relatório *BLOB*;
BLOB válido: 'Minha terra tem palmeiras onde canta o...'

relatório *TEXT*;
TEXT válido: 'A que saudades que eu sinto...'

SET

Um campo interessante que permite que o usuário faça uma escolha dado determinado número de opções.
Cada campo pode conter até 64 opções.

Exemplo:
vicio *SET*("cafe", "cigarro") NOT NULL;

Neste exemplo o campo pode conter apenas os seguintes itens:

""
"cafe"

"cigarro"
"cafe,cigarro"

ENUM

Este campo tem funcionamento semelhante ao *SET*, com a diferença que apenas um valor pode ser escolhido.

Exemplo:

virtude *ENUM*("programar", "amar") *NOT NULL*;
Neste exemplo este campo pode conter os seguintes valores:
""
"programar"
"amar"

Registros

São conjuntos de campos relacionados. Um registro, portanto, pode ter a seguinte estrutura:

nome *CHAR*(15);
email *CHAR*(25);
telefone INT;

Neste exemplo, o registro contém três campos, podendo armazenar o *e-mail* e o telefone de uma determinada pessoa. Observe que o campo nome foi definido como *CHAR*, portanto, poderá conter qualquer tipo de caractere.

Já o campo telefone, definido como INT, poderá apenas conter números, pois foi configurado como INT.

Tabelas

Um conjunto de registros forma uma tabela. As tabelas, portanto, armazenam grande quantidade de dados.

Como no exemplo anterior, poderíamos ter centenas de nomes diferentes cadastrados em nossa tabela de pessoas, pois cada conjunto de dados corresponde a um registro.

Antes de usar uma base de dados ou dar qualquer comando, nós precisamos de uma tabela, pelo menos para armazenar os dados. Isto pode ser feito usando o comando *CREATE TABLE*, que recebe como parâmetro o nome da tabela que desejamos usar.

```
---------------------------------------------------------------
mysql> CREATE TABLE teste(
>codigo INT,
>nome CHAR(15),
>email CHAR(25),
>telefone INT);
---------------------------------------------------------------
```

Notas:

- Não é possível criar duas tabelas com o mesmo nome;
- Cada conjunto de dados, quando visto em uma tabela, é também chamado linha;
- Você acaba de criar sua primeira tabela!

Características das linhas:

- O nome de uma coluna pode ter até 64 letras;
- O nome de uma coluna pode começar com um número;
- O nome de uma coluna não pode ser composto de números apenas.

Opções de uma tabela:

As seguintes opções podem ser adicionadas a qualquer campo de uma tabela, concatenando recursos adicionais a estes campos.

- Chave primária:

Define o campo como chave primária da tabela. Para se definir uma chave primária, basta adicionar '*PRIMARY KEY*' à definição do campo que se deseja a não-duplicidade.

Exemplo:

Matricula INT *PRIMARY KEY*;

Esta declaração faz com que não seja permitido o cadastro de dois registros com matrículas iguais na tabela.

- Auto-incremento:

Este recurso atribui ao campo de um novo registro, o valor do campo gravado no registro anterior somado mais um.

Exemplo:

codigo INT *AUTO_INCREMENT*;

Soma automaticamente um a cada registro neste campo, começando a partir de um em diante.

Comandos relativos às tabelas:

Pode-se executar comandos para saber as condições que as tabelas se encontram e também manipulá-las.

Seguem os comandos fundamentais:

- Mostrar tabelas

Função:

Lista todas as tabelas existentes no banco de dados atual.

Comando:

mysql>show tables;

- Mostrar colunas

Função:

Mostra as informações referentes à estrutura, ou seja, as colunas das tabelas desejadas.

Comando:

mysql>show columns from teste;

Pode manipular à vontade a tabela criada para os testes, pois o aprendizado depende de sua disposição em fazer, desfazer e refazer comandos e tarefas, sem medo de danificar a base de dados, que nem é tão frágil assim.

Manipulando a base de dados

Neste ponto aprende-se as operações básicas de DML: Incluir, Apagar, Alterar e Pesquisar dados.

Inserindo registros

Para se adicionar dados a uma tabela, usamos o comando INSERT, que diz por si só sua função, como o exemplo que segue:

```
---------------------------------------------------------------------
mysql>INSERT INTO teste VALUES
mysql>(NULL, 'Ernesto', 'ernesto@nbsnet.com.br',
mysql>2742729);
---------------------------------------------------------------------
```

Observações:

- Apóstrofes foram colocadas na entrada de campos tipo *CHAR*. Todos os campos que contém texto, ou seja, *CHAR*, *VARCHAR*, *BLOB*, *TEXT*, etc. têm de ficar entre apóstrofes, ou, do contrário, um erro ocorrerá;

- Para campos do tipo número não se usa apóstrofes.

- A entrada *NULL* em um campo do tipo auto-incremento, permite que o MySQL providencie o conteúdo deste campo de forma automática. No caso do primeiro campo, o valor será 1, no segundo 2 e no terceiro 3 e assim consecutivamente.

- Se possuíssemos um campo *DATE*, a entrada *NULL* faria com que o valor gravado no registro se tornasse a data atual.

Nota 1:

É importante lembrar-se sempre de passar para o camando INSERT, um número de parâmetros igual ao número de campos na tabela que está

recebendo os dados. Caso contrário, você obterá uma mensagem de erro.

O mesmo erro também ocorre ao tentar inserir mais parâmetros do que o número de campos suportado pela tabela.

Nota 2:

Uma das grandes vantagens do MySQL é a capacidade de converter campos com facilidade e sem problemas entre eles, ou seja, o sistema converte automaticamente entre números, caracteres e datas, sem causar danos.

Pesquisando registros

As pesquisas no MySQL são feitas através do comando *SELECT*. Este comando pode realizar desde uma simples e trivial pesquisa até uma pesquisa extremamente complexa.

Exemplos:

Ação:
mysql>*SELECT* * *FROM* teste;

Resultado:
Lista todos os registros da tabela teste.

Ação:
mysql>*SELECT* * *FROM* teste
mysql>*WHERE* (nome = "Ernesto");

Resultado:
Lista todos os registros da tabela teste que contém "Ernesto" no campo nome.

Nota:
Consulte maiores detalhes e estude mais itens relacionados a pesquisas SQL. Você pode economizar centenas de linhas de códigos em

seus programas usando recursos de pesquisa.

Apagando registros

Quando um registro não nos é mais útil, podemos apagá-lo, a fim de que a tabela não contenha dados obsoletos.

Isto pode ser facilmente feito usando o comando *DELETE*, que tem o funcionamento semelhante ao comando INSERT.

Ação:

mysql>*DELETE FROM* teste

mysql>*WHERE* (telefone = 2744747);

Resultado:

Apaga da tabela teste todos os registros que têm o conteúdo "2744747" no campo telefone.

Alterando registros

Esta operação é vital, pois se um dado muda no universo da tabela, ele deve ser alterado também na Base de Dados. Para tal, existe o camando *UPDATE*.

Observe os exemplos:

Ação:

mysql>*UPDATE* teste *SET* nome = 'Nonato'

mysql>*WHERE* nome = 'Ernesto';

Resultado:

Procura na tabela um registro que contenha no campo nome o conteúdo 'Ernesto', definido pelo comando *WHERE*. Encontrado o registro, ele é substituído pelo nome definido no comando *SET*, que é 'Nonato'.

Comandos avançados

A partir deste ponto, mostraremos alguns comandos avançados do MySQL:

Operadores lógicos:

O MySQL suporta todas operações lógicas, estando abaixo listadas as mais conhecidas:

***AND* (&&)**

Este operador implementa o E lógico entre duas cláusulas. Observe o exemplo:

```
-----------------------------------------------------
mysql>SELECT * FROM teste WHERE
mysql>(nome = "Aline") AND
mysql>(telefone = 2728918);
-----------------------------------------------------
```

Esta pesquisa mostrará todos os registros que contém no campo nome e conteúdo "Aline"; E (*AND*) no campo telefone, o conteúdo 2728988.

***OR* (||)**

Este operador implementa o OU lógico entre duas cláusulas.

```
-----------------------------------------------------
mysql>SELECT * FROM teste WHERE
mysql>(nome = "Aline") OR
mysql>(telefone = 2728918);
-----------------------------------------------------
```

A pesquisa realizada através deste operador fará com que todos os resultados que contenham o conteúdo "Aline" no campo nome OU 2728918 sejam exibidos na tela.

NOT (!)

O operador lógico *NOT*, que significa NÃO, realiza uma pesquisa, excluindo valores determinados do resultado.

```
--------------------------------------------------
mysql>SELECT * FROM teste WHERE
mysql>(nome != "Aline");
--------------------------------------------------
```

Esta pesquisa listará todos os registros da base de dados teste, NÃO (*NOT*) mostrando aqueles que possuem "Aline" como conteúdo do campo nome.

ORDER BY

O operador lógico *ORDER BY*, ou ORDENAR POR, simplesmente lista os registros, colocando-os em ordem de acordo com o campo solicitado.

```
--------------------------------------------------
mysql>SELECT * FROM teste ORDER BY nome;
--------------------------------------------------
```

O resultado desta busca mostra todos os registros de teste em ordem alfabética de nome.

FINALIZANDO

Mais detalhes sobre o MySQL podem ser obtidos na sua documentação (apostila) e no sítio oficial do programa.

APÊNDICE - B

Uso prático de SQL padrão

Neste apêndice, são listados usos da linguagem SQL propostos pelos professores Jorge Surian e Luiz Nicochelli, em sua apostila de SQL disponível no sítio do apostilando.

1) Seleção de todos os campos (ou colunas) da tabela de Departamentos.

Resp:

```
SELECT * FROM DEPT;
```

O exemplo utiliza o coringa "*" para selecionar as colunas na ordem em que foram criadas. A instrução *Select*, como se pode observar, seleciona um grupo de registros de uma ou mais tabelas. A instrução *From*, por sua vez, nos indica a necessidade de pesquisarmos tais dados apenas na tabela Dept.

Where como base das Restrição de tuplas.

A cláusula *"where"* corresponde à operadora restrição da álgebra relacional. Contém a condição que as tuplas devem obedecer a fim de serem listadas. Ela pode comparar valores em colunas, literais, expressões aritméticas ou funções.

A seguir, apresentamos operadores lógicos e complementares a serem utilizados nas expressões apresentadas em *"where"*.

Operadores lógicos

operador	significado
=	igual a
>	maior que
>=	maior que ou igual a
<	menor que
<=	menor que ou igual a

Exemplos:

```
SELECT EMPNOME, EMPSERV
   FROM EMP
   WHERE DEPNUME > 10;
```

```
SELECT  EMPNOME, EMPSERV
    FROM  EMP
    WHERE EMPSERV = 'GERENTE';
```

O conjunto de caracteres ou datas devem estar entre apóstrofes (‘) na cláusula *"where"*.

2) Selecione todos os departamentos cujo orçamento mensal seja maior que 100000. Apresente o Nome de tal departamento e seu respectivo orçamento anual, cujo resultado será obtido multiplicando-se o orçamento mensal por 12.

Resp:

Neste problema precisamos de uma expressão, que é a combinação de um ou mais valores, operadores ou funções que resultarão em um valor. Esta expressão poderá conter nomes de colunas, valores numéricos, constantes e operadores aritméticos.

```
SELECT  DEPNOME,  DEPORCA * 12
    FROM  DEPT
    WHERE DEPORCA > 100000;
```

3) Apresente a instrução anterior, porém, ao invés dos "feios" DepNome e DepOrca, apresente os Títulos Departamento e Orçamento.

Resp:

Neste exemplo, deveremos denominar colunas utilizando apelidos. Os nomes das colunas mostradas por uma consulta, são geralmente os nomes existentes no Dicionário de Dados, mas geralmente estão armazenados na forma do mais puro "informatiquês", onde "todo mundo" sabe que CliCodi significa Código do Cliente.

É possível (e provável) que o usuário desconheça estes símbolos; portanto, devemos apresentá-los dando apelidos às colunas "contaminadas" pelo informatiquês, que apesar de fundamental para os analistas, somente são vistos como enigmas para os usuários.

SELECT DEPNOME "DEPARTAMENTO", DEPORCA * 12 "ORCAMENTO ANUAL"
FROM DEPT
WHERE DEPORCA > 100000;

4) Apresente todos os salários existentes na empresa, porém, omita eventuais duplicidades.

Resp:

A cláusula *Distinct* elimina duplicidades, significando que somente relações distintas serão apresentadas como resultado de uma pesquisa.

SELECT DISTINCT EMPSERV
FROM EMP;

5) Apresente todos os dados dos empregados, considerando sua existência física diferente de sua existência lógica, ou seja, devidamente inicializado.

Resp:

Desejamos um tratamento diferenciado para valores nulos. Qualquer coluna de uma tupla que não contenha informações é denominada de nula, portanto, informação não existente.

Isto não é o mesmo que "zero", pois zero é um número como outro qualquer, enquanto que um valor nulo utiliza um "byte" de armazenagem interna, sendo tratado de forma diferenciada pelo SQL.

SELECT EMPNOME, EMPSALA + EMPCOMI
FROM EMP;

SELECT EMPNOME, NVL(EMPSALA,0) + NVL(EMPCOMI,0)
FROM EMP;

6) Apresente os nomes e funções da cada funcionário contidos na tabela empresa, porém classificados alfabeticamente (A..Z) e depois alfabeticamente invertido (Z..A).

A função "NVL" é utilizada para converter valores nulos em zeros.

Resp:

A cláusula *Order By* modificará a ordem de apresentação do resultado da pesquisa (ascendente ou descendente).

```
SELECT  EMPNOME, EMPSERV
    FROM  EMP
    ORDER  BY  EMPNOME;

SELECT  EMPNOME, EMPSERV
    FROM  EMP
    ORDER  BY  EMPPNOME  DESC;
```

Nota:

Também é possível fazer com que o resultado da pesquisa venha classificado por várias colunas. Sem a cláusula "order by", as linhas serão exibidas na sequência que o SGBD determinar.

7) Selecione os Nomes dos Departamentos que estejam na fábrica.

Resp:

```
SELECT DEPNOME
    FROM DEPT
    WHERE DEPLOCA = "SAO PAULO";
```

O exemplo exigiu uma restrição (São Paulo) que nos obrigou a utilizar da instrução *"Where"*. Alguns analistas costumam afirmar em tom jocoso que SQL não passa de:

"Selecione algo De algum lugar Onde se verificam tais relações"

Acreditamos que esta brincadeira pode ser útil ao estudante, na medida em que facilita sua compreensão dos objetivos elementares do SQL.

Demais Operadores

Operador	Significado
between ... and ...	entre dois valores (inclusive)
in (....)	lista de valores
like	com um padrão de caracteres
is null	é um valor nulo

Exemplos:

```
SELECT  EMPNOME, EMPSALA
    FROM  EMP
   WHERE  EMPSALA BETWEEN  500  AND  1000;

SELECT  EMPNOME,  DEPNUME
    FROM  EMP
    WHERE  DEPNUME  IN  (10,30);

SELECT  EMPNOME, EMPSERV
    FROM  EMP
    WHERE   EMPNOME  LIKE  'F%';

SELECT EMPNOME, EMPSERV
    FROM  EMP
    WHERE  EMPCOMI  IS  NULL;
```

O símbolo "%" pode ser usado para construir a pesquisa ("%" = qualquer sequência de nenhum até vários caracteres).

Operadores Negativos

Operador	Significado
<>	diferente
not nome_coluna =	diferente da coluna
not nome_coluna >	não maior que
not between	não entre dois valores informados
not in	não existente numa dada lista de valores

not like	diferente do padrao de caracteres informado
is not null	não é um valor nulo

8) Selecione os Empregados cujos salários sejam menores que 1000 ou maiores que 3500.

Resp:
Necessitaremos aqui a utilização de expressões negativas.
A seguir, apresentamos operadores negativos.

```
SELECT EMPNOME, EMPSALA
    FROM EMP
    WHERE EMPSALA NOT BETWEEN 1000 AND 3500;
```

9) Apresente todos os funcionários com salários entre 200 e 700 e que sejam Vendedores.

Resp:
Necessitaremos de consultas com condições múltiplas.
Operadores "*AND*" (E) e "*OR*" (OU).

```
SELECT EMPNOME, EMPSALA, EMPSERV
    FROM  EMP
    WHERE  EMPSALA  BETWEEN  700 AND 2000
              AND  EMPSERV =  'VENDEDOR';
```

10) Apresente todos os funcionários com salários entre 200 e 700 ou que sejam Vendedores.

Resp:

```
SELECT EMPNOME, EMPSALA, EMPSERV
    FROM  EMP
    WHERE EMPSALA  BETWEEN  700 AND 2000
              OR  EMPSERV =  'VENDEDOR';
```

11) Apresente todos os funcionários com salários entre 200 e 700 e que sejam Vendedores ou Balconistas.

Resp:

```
SELECT EMPNOME, EMPSALA, EMPSERV
    FROM  EMP
    WHERE  EMPSALA  BETWEEN  700 AND 2000
            AND ( EMPSERV =  'BALCONISTA' OR  EMPSERV =
'VENDEDOR' );
```

Funções de Caracteres

Lower - força caracteres maiúsculos aparecerem em minúsculos.

Upper - força caracteres minúsculos aparecerem em maiúsculos.

***Concat*(x,y)** - concatena a *string* "x" com a string "y".

***Substring*(x,y,str)** - extrai um *substring* da *string* "str", começando em "x", e termina em "y".

***To_Char*(num)** - converte um valor numérico para uma *string* de caracteres.

***To_Date*(char,fmt)** - converte uma *string* caractere em uma data.

^Q - converte data para o formato apresentado.

12) Apresente o nome de todos os empregados em letras minúsculas.

Resp:

```
SELECT LOWER( EMPNOME )
    FROM EMP;
```

13) Apresente o nome de todos os empregados (somente as 10 primeiras letras).

Resp:

```
SELECT  SUBSTRING (1,10,EMPNOME)
    FROM EMP;
```

14) Apresente o nome de todos os empregados admitidos em 01/01/80.

Resp:

```
SELECT  *
   FROM  EMP
   WHERE  EMPADMI =  ^Q"DD-AAA-YYYY"("01-JAN-1980");
```

ou

```
SELECT  * FROM  EMP
   WHERE  EMPADMI =  ^Q("01-JAN-1980");
```

Funções Agregadas (ou de Agrupamento)

função	retorno
avg(n)	média do valor n, ignorando nulos
count(expr)	vezes que o número da expr avalia para algo não nulo
max(expr)	maior valor da expr
min(expr)	menor valor da expr
sum(n)	soma dos valores de n, ignorando nulos

15) Apresente a Média, o Maior, o Menor e também a Somatória dos Salários pagos aos empregados.

Resp:

```
SELECT  AVG(EMPSALA)  FROM  EMP;

SELECT  MIN(EMPSALA)  FROM  EMP;

SELECT  MAX(EMPSALA)  FROM EMP;

SELECT SUM(EMPSALA) FROM  EMP;
```

Agrupamentos

As funções de grupo operam sobre grupos de tuplas (linhas).

Retornam resultados baseados em grupos de tuplas, em vez de resultados de funções por tupla individual. A cláusula *"group by"* do comando *"select"* é utilizada para dividir tuplas em grupos menores.

A cláusula "GROUP BY" pode ser usada para dividir as tuplas de uma tabela em grupos menores. As funções de grupo devolvem uma informação sumarizada para cada grupo.

Qualquer coluna ou expressão na lista de seleção, que não for uma função agregada, deverá constar da cláusula *"group by"*. Portanto, é errado tentar impor uma "restrição" do tipo agregada na cláusula *Where*.

16) Apresente a média de salários pagos por departamento.

Resp:

```
SELECT DUPNUME, AVG(EMPSALA)
    FROM EMP
    GROUP BY DEPNUME;
```

Having

A cláusula *"HAVING"* pode ser utilizada para especificar quais grupos deverão ser exibidos, portanto, restringindo-os.

17) Retome o problema anterior, porém apresente resposta apenas para departamentos com mais de 10 empregados.

Resp:

```
SELECT  DEPNUME, AVG(EMPSALA)
    FROM  EMP
    GROUP BY DEPNUME
    HAVING COUNT(*)  >  3;
```

Obs.: A cláusula *"group by"* deve ser colocada antes da *"having"*, pois os grupos são formados e as funções de grupos são calculadas antes de se resolver a cláusula *"having"*.

A cláusula *"where"* não pode ser utilizada para restringir grupos que deverão ser exibidos.

Exemplificando ERRO típico - Restringindo Média Maior que 1000:

```
SELECT DEPNUME, AVG(EMPSALA)
    FROM EMP
    WHERE  AVG(SALARIO) > 1000
    GROUP BY DEPNUME;
```

(Esta seleção está ERRADA!)

```
SELECT  DEPNUME, AVG(EMPSALA)
    FROM EMP
    GROUP BY  DEPNUME
    HAVING AVG(EMPSALA) > 1000;
```

(Seleção Adequada)

Sequência no comando *"Select"*:

SELECT	coluna(s)
FROM	tabela(s)
WHERE	condição(ões) da(s) tupla(s)
GROUP BY	condição(ões) do(s) grupo(s) de tupla(s)
HAVING	condição(ões) do(s) grupo(s) de tupla(s)
ORDER BY	coluna(s);

A "SQL" fará a seguinte avaliação:

a) WHERE, para estabelecer tuplas individuais candidatas (não pode conter funções de grupo).

b) GROUP BY, para fixar grupos.

c) HAVING, para selecionar grupos para exibição.

Equi-Junção (Junção por igualdade)

O relacionamento existente entre tabelas é chamado de equi-junção, pois os valores de colunas das duas tabelas são iguais. A Equi-junção é possível apenas quando tivermos definido de forma adequada, a chave estrangeira de uma tabela e sua referência à chave primária da tabela precedente.

Apesar de em alguns casos, admitir-se a equi-junção de tabelas, sem a correspondência Chave Primária - Chave Estrangeira, recomendamos ao estudante não utilizar este tipo de construção, pois certamente em nenhum momento nos exemplos propostos em nossa disciplina ou nas disciplinas de Análise e Projeto de Sistemas serão necessárias tais junções.

18) Listar Nomes de Empregados, Cargos e Nome do Departamento onde o empregado trabalha.

Resp:

Observemos que dois dos três dados solicitados estão na Tabela Emp, enquanto o outro dado está na Tabela Dept. Deveremos então acessar os dados restringindo convenientemente as relações existentes entre as tabelas. De fato, sabemos que DEPNUME é chave primária da tabela de Departamentos, e também é chave estrangeira da Tabela de Empregados. Portanto, este campo será o responsável pela equi-junção.

```
SELECT A.EMPNOME, A.EMPSERV, B.DEPNOME
FROM EMP A, DEPT B
    WHERE A.DEPNUME = B.DEPNUME;
```

Obs.:

Note que as tabelas quando contêm colunas com o mesmo nome, usa-se um apelido "alias" para substituir o nome da tabela associado à coluna. Imagine que alguém tivesse definido NOME para ser o Nome do Empregado na Tabela de Empregados, e também NOME para ser o Nome do Departamento na Tabela de Departamentos. Tudo funcionaria de forma adequada, pois o “alias” se encarregaria de evitar que uma ambiguidade fosse verificada.

Embora SQL resolva de forma muito elegante o problema da nomenclatura idêntica para campos de tabelas, recomendamos que o estudante evite tal forma de nomear os campos. O SQL nunca confundirá

um A.NOME com um B.NOME, porém, podemos afirmar o mesmo de nós mesmos?

19) Liste os Códigos do Cada Funcionário, seus Nomes, seus Cargos e o nome do Gerente ao qual este se relaciona.

Resp:

Precisamos criar um autorrelacionamento, ou seja, juntar uma tabela a ela própria. É possível juntarmos uma tabela a ela mesma com a utilização de apelidos, permitindo juntar tuplas da tabela a outra tuplas da mesma tabela.

```
SELECT A.EMPNUME, A.EMPNOME, A.EMPSERV, B.EMPNOME
    FROM  EMP A, EMP B
    WHERE  A.EMPGERE  = B.EMPNUME;
```

As SubConsultas

Uma subconsulta é um comando *"select"* que é aninhado dentro de outro "*select*" e que devolve resultados intermediários.

20) Relacione todos os nomes de funcionários e seus respectivos cargos, desde que o orçamento do departamento seja igual a 300000.

Resp:

```
SELECT  EMPNOME, EMPSERV
    FROM EMP A
    WHERE  300000 IN ( SELECT  DEPORCA
          FROM  DEPT
          WHERE DEPT.DEPNUME = A.DEPNUME
```

Nota:

Observe que a cláusula *IN* torna-se verdadeira quando o atributo indicado está presente no conjunto obtido através da subconsulta.

21) Relacione todos os departamentos que possuem empregados com remuneração maior que 3500.

Resp:

```
SELECT DEPNOME
    FROM DEPT A
    WHERE EXISTS (SELECT *
                  FROM EMP
                      WHERE EMPSALA > 3500 AND EMP.
DEPNUME = A.DEPNUME');
```

Nota:

Observe que a cláusula *EXISTS* indica se o resultado de uma pesquisa contém ou não tuplas. Observe também que poderemos verificar a não-existência (*NOT EXISTS*), caso esta alternativa seja mais conveniente.

Uniões

Podemos eventualmente unir duas linhas de consultas, simplesmente utilizando a palavra reservada *UNION*.

22) Liste todos os empregados que tenham códigos > 10 ou Funcionários que trabalhem em departamentos com código maior que 10.

Resp:

Poderíamos resolver esta pesquisa com um único Select, porém, ao fato de estarmos trabalhando em nosso exemplo com apenas duas tabelas, não conseguimos criar um exemplo muito adequado para utilização deste recurso.

```
(Select *
    From Emp
    Where EmpNume > 10)
Union
(Select *
    From Emp
    Where DepNume > 10);
```

Inserções, Alterações e Exclusões

Uma linguagem direcionada à extração de informações de um conjunto de dados, em tese não deveria incorporar comandos de manipulação dos dados. Devemos observar, contudo, que a mera existência de uma linguagem padronizada para acesso aos dados, "convidava" os desenvolvedores a aderirem a uma linguagem "padrão" de manipulação de tabelas.

Naturalmente cada desenvolvedor coloca "um algo mais" em seu SQL (SQL PLUS, SQL *, ISQL e toda sorte de nomenclaturas), por um lado desvirtuando os objetivos da linguagem (padronização absoluta), mas em contrapartida otimiza os acessos ao seu banco de dados, e por maior que sejam estas mudanças, jamais são tão importantes a ponto de impedir que um programador versado em SQL tenha grandes dificuldades em se adaptar ao padrão de determinada implementação.

De fato, as diferenças entre o SQL da *Sybase, Oracle, Microsoft* são muito menores que as existentes entre o C, o *BASIC* e o Pascal, os quais são chamados de linguagens "irmãs", pois todas se originam conceitualmente no *FORTRAN*.

Podemos observar que todas as três linguagens mencionadas possuem estruturas de controle tipo "para" *(for)*, "enquanto" *(while)* e repita (*do..while, repeat..until*). Todas trabalham com blocos de instrução, todas têm regras semelhantes para declaração de variáveis, e todas usam comandos de tomada de decisão baseados em instruções do tipo "se" ou "caso". Porém, apesar de tantas semelhanças (*sic*), é praticamente impossível que um programador excelente em uma linguagem consiga rapidamente ser excelente em outra linguagem do grupo.

Poderíamos arriscar a dizer que um excelente programador C que utilize a implementação da *Symantech*, terá que passar por um breve período de adaptação para adaptar-se ao C da *Microsoft.*

O que ocorreria então se este programador tiver que adaptar-se ao Delphi (Pascal) da Borland?

De forma alguma, o mesmo ocorrerá com o especialista em SQL ao ter que migrar do Banco de Dados X para o Banco de Dados Y. Naturalmente, existirá a necessidade de aprendizado, mas este programador poderá ir adaptando-se aos poucos sem precisar ser treinado novamente, o que é um aspecto extremamente vantajoso para as empresas.

Inserir (*Insert*)

INSERT INTO <tabela> [<campos>] [*VALUES* <valores>]

Ex:

INSERT INTO DEPT;

Possibilita a inserção de registros de forma interativa.

I*NSERT INTO DEPT* (DEPNUME,DEPNOME,DEPLOCA) *VALUES* (70,"PRODUCAO","RIO DE JANEIRO");

Possibilita a inserção de registros em tabelas sem digitação dos dados.

Atualizar (Update)

UPDATE <tabela> *SET* <campo> = <expressão> [*WHERE* <condição>];

Ex:

*UPDATE EMP SET EMPSALA = EMPSALA** 1.2 *WHERE EMPSALA*< 1000;

Excluir (*Delete*)

DELETE FROM <tabela> [*WHERE* <condição>];

Ex:

DELETE FROM emp *WHERE EMPSALA* > 5000;

Webliografia

http://disciplinas.dcc.ufba.br/svn/MATA60/tarefa1/historico/

historico.pdf?revision=21

http://www.ime.usp.br/~andrers/aulas/bd2005-1/

http://www.cos.ufrj.br/~marta/BdRel.pdf

http://www.webng.com/pmds/Arquivos/HistoricoBD.pdf

http://www.sqlmagazine.com.br/Colunistas/RicardoRezende/03_ConceitosBD_P2.asp

http://www.devmedia.com.br/articles/viewcomp.asp?comp=5935

http://www.apostilando.com/

http://www.scribd.com/doc/512604/Modelo-EntidadeRelacionamento

http://www.inf.puc-rio.br/~casanova/INF1731-BD/modulo02.pdf

http://chasqueweb.ufrgs.br/~paul.fisher/apostilas/basdad/bd_mod_er.htm

http://www.ime.usp.br/~andrers/aulas/bd2005-1/aula6.html

http://www2.dc.ufscar.br/~carvalho/GUBD/aulas/36.html

http://www.ic.unicamp.br/~geovane/mo410-091/Ch02-MER_pt.pdf

http://www.shammas.eng.br/acad/materiais/mer.pdf

http://www.juliobattisti.com.br/artigos/office/modelorelacional_p1.asp

http://www.dca.fee.unicamp.br/cursos/PooJava/javadb/bdrel.htm

http://www.dpi.inpe.br/spring/teoria/consulta/consulta.html

http://pt.tech-faq.com/relational-database.shtm

http://www.baixaki.com.br/download/mysql-workbench.htm

http://forum.imasters.uol.com.br/index.php?showtopic=248701

http://www.webtuga.com/ola-mysql-workbench-adeus-dbdesigner4/

http://www.tipandtrick.net/2008/mysql-workbench-visual-database-deign-tool-free-download/pt/

http://www.mysqlbrasil.com.br/

http://dev.mysql.com/doc/refman/4.1/pt/index.html

http://br-linux.org/2009/sun-lanca-o-mysql-54/

http://www.criarweb.com/manuais/mysql/

http://www.dbtools.com.br/

Referências

ALVES, William Pereira. **Fundamentos de Bancos de Dados**. São Paulo: Érica, 2004. 382p.

COUGO, Paulo. **Modelagem Conceitual e Projeto de Banco de Dados**. 1ª edição. 4ª tiragem. São Paulo: Ed. CAMPUS, 2000.

ELMASRI, Ramez & NAVATHE, Shamkant B. **Fundamentos de Sistemas de Banco de Dados**. 4ª Edição. Cidade: Ed. Pearson/Addinson Weslley, 2006.

GUIMARÃES, Célio C. **Fundamentos de Bancos de Dados**. Campinas-SP: Ed. Unicamp, 2003.

KORTH, Henry et all. **Sistemas de Banco de Dados**. 5ª edição. São Paulo: Ed. CAMPUS, 2006.

MACHADO, Felipe Nery Rodrigues. **Banco de dados**: projeto e implementação. São Paulo: Érica, 2004. 398p.

www.uapi.ufpi.br