ESCOLA
Alcides Maya

Banco de Dados

1 INTRODUÇÃO

Existem hoje gigantescas bases de dados gerenciando nossas vidas. De fato, sabemos que nossa conta bancária faz parte de uma coleção imensa de contas bancárias de nosso banco. Nosso Título Eleitoral ou nosso Cadastro de Pessoa Física, certamente estão armazenados em Bancos de Dados colossais. Sabemos também que quando sacamos dinheiro no caixa eletrônico de nosso banco, nosso saldo e as movimentações existentes em nossa conta bancária já estão à nossa disposição.

Nestas situações, sabemos que existe uma necessidade em se realizar o armazenamento de uma série de informações que não se encontram efetivamente isoladas umas das outras, ou seja, existe uma ampla gama de dados que se referem a relacionamentos existentes entre as informações a serem manipuladas.

Estes Bancos de Dados, além de manterem todo este volume de dados organizado, também devem permitir atualizações, inclusões e exclusões do volume de dados, sem nunca perder a consistência. E não podemos esquecer que na maioria das vezes estaremos lidando com acessos concorrentes a várias informações de nosso banco de dados, algumas vezes até com mais de um acesso à mesma informação!

Então, o fato de se montar uma mala direta em um computador com um drive já faz de nós um autor de um Banco de Dados? É claro que não! Um Banco de Dados é, antes de mais nada, uma coleção logicamente coerente e organizada de dados com determinada significação intrínseca.

Um Banco de Dados contém os dados dispostos numa ordem pré-determinada em função de um projeto de sistema, sempre para um propósito muito bem definido. Um Banco de Dados representará sempre aspectos do Mundo Real. Assim sendo uma Base de Dados (ou Banco de Dados, ou ainda BD) é uma fonte de onde poderemos extrair uma vasta gama de informações derivadas, que possui um nível de interação com eventos como o Mundo Real que representa.

A forma mais comum de interação Usuário e Banco de Dados, dá-se através de sistemas específicos que, por sua vez, acessam o volume de informações através de uma linguagem apropriada.

No processamento tradicional de arquivos, a estrutura dos dados está incorporada ao programa de acesso. Desta forma, qualquer alteração na estrutura de arquivos implica na alteração no código fonte de todos os programas. Já na abordagem banco de dados, a estrutura é alterada apenas no catálogo, não alterando os programas.

2 CONCEITOS GERAIS

2.1 Banco de Dados

Um banco de dados pode ser definido como um conjunto de dados devidamente relacionados. Porém, o significado do termo banco de dados é mais restrito que simplesmente a definição dada acima. Um banco de dados possui as seguintes propriedades:

- um banco de dados é uma coleção lógica coerente de dados com um significado inerente; uma disposição desordenada dos dados não pode ser referenciada como um banco de dados;
- um banco de dados é projetado, construído e populado com dados para um propósito específico; um banco de dados possui um conjunto pré definido de usuários e aplicações;
- um banco de dados representa algum aspecto do mundo real, o qual é chamado de mini-mundo; qualquer alteração efetuada no mini-mundo é automaticamente refletida no banco de dados.

Um banco de dados representa o arquivo físico, armazenado em dispositivos periféricos, onde estão armazenados os dados de diversos sistemas, para consulta e atualização pelo usuário.

2.1.1 Vantagens de um Banco de Dados

Redução ou eliminação de redundâncias: os dados, que eventualmente são comuns a mais de um sistema, são armazenados em um único lugar e são compartilhados por todos os sistemas, permitindo o acesso a uma única informação.

- Eliminação de inconsistências: através do armazenamento da informação em um único local e, sendo compartilhada a vários sistemas, os usuários estarão utilizando uma informação confiável. A inconsistência ocorre quando um mesmo campo tem valores diferentes em sistemas diferentes. Isto ocorre porque uma informação foi atualizada em um sistema e não foi feito o mesmo em outro. Quando o dado é armazenado em um único local e compartilhado pelos sistemas, este problema não ocorre.
- Compartilhamento dos dados: permite a utilização simultânea e segura de um dado, por mais de uma aplicação ou usuário, independente da operação que esteja sendo realizada. Deve ser observado apenas o processo de atualização concorrente, para não gerar erros de processamento (atualizar simultaneamente o mesmo campo do mesmo registro). Os aplicativos são por natureza multiusuário.
- Restrições de segurança: define para cada usuário o nível de acesso a ele concedido (ex.: leitura, leitura e gravação ou sem acesso) ao arquivo e/ou campo. Este recurso impede que pessoas não autorizadas utilizem ou atualizem uma determinada informação.
- Padronização dos dados: permite que os campos armazenados na base de dados sejam padronizados segundo um determinado formato de armazenamento. Ex. Para o campo “Sexo” somente será permitido armazenamento dos conteúdos “M” ou “F”.
- Manutenção de integridade: exige que o conteúdo dos dados armazenados no banco de dados possua apenas valores coerentes ao objetivo do campo, não permitindo que valores absurdos sejam cadastrados. Exemplo: Um funcionário que faça no mês 500 horas extras, ou um aluno que tenha nascido no ano de 1860.
- Evitar necessidades conflitantes: representa a capacidade que o administrador do banco de dados deve ter para solucionar “prioridades sempre altas” de todos os sistemas, tendo ele que avaliar a real necessidade de cada sistema para a empresa para priorizar a sua implantação.
- Independência dos dados: representa a forma física de armazenamento dos dados no banco de dados e a recuperação das informações pelos programas de aplicação. Esta recuperação deverá ser totalmente independente da maneira com que os dados estão fisicamente armazenados. Ex.: Quando um programa retira ou inclui dados, o sistema gerenciador compacta-os para que haja um menor consumo de espaço no disco. Este conhecimento do formato de armazenamento do campo é totalmente transparente para o usuário.

2.2 SGDB (Sistema Gerenciador de Bancos de Dados)

É um software que permite a definição de estruturas para o armazenamento de informações e o fornecimento de mecanismos para manipulá-las. Um SGBD permite aos usuários criarem e manipularem bancos de dados de um propósito geral.

Este software deve fornecer a interface entre os dados armazenados em um banco de dados e os programas aplicativos.

São tarefas de um SGBD:

- interação com o sistema de arquivos do sistema operacional,
- cumprimento da integridade,
- cumprimento da segurança,
- cópias de segurança (“backup”) e recuperação,
- controle de concorrência.

Um SGBD deve manter não somente os dados, mas também a forma como os mesmos são armazenados, guardando uma descrição completa do banco de dados. Estas informações são armazenadas no catálogo do SGBD, o qual contém informações como a estrutura de cada arquivo, o tipo e o formato de armazenamento de cada tipo de dado, restrições, etc.

A informação armazenada no catálogo é chamada de *metadado*. Um metadado é uma informação sobre uma informação, ou seja, uma informação que descreve uma informação primária.

No processamento tradicional de arquivos, o programa que irá manipular os dados deve conter este tipo de informação, ficando limitado a manipular as informações que o mesmo conhece.

O conjunto formado por um banco de dados somado às aplicações que manipulam o mesmo é chamado de “Sistema de Banco de Dados”.

2.3 Sistema de Bancos de Dados

Consiste em uma coleção de dados interrelacionados e uma coleção de programas para prover o acesso a esses dados. O objetivo principal de um sistema de banco de dados é prover um ambiente que seja adequado e eficiente para uso na recuperação e no armazenamento de informações.

Os objetivos de um Sistema de Banco de Dados envolve:

- isolar os usuários dos detalhes mais internos do banco de dados (abstração de dados),
- prover independência de dados às aplicações (estrutura física de armazenamento e estratégia de acesso).

As vantagens de um Sistema de Banco de Dados são:

- rapidez na manipulação e no acesso à informação,
- redução do esforço humano (desenvolvimento e utilização),
- disponibilização da informação no tempo necessário,
- controle integrado de informações distribuídas fisicamente,
- redução de redundância e de inconsistência de informações,
- compartilhamento de dados,
- aplicação automática de restrições de segurança,
- redução de problemas de integridade.

3 ARQUITETURA DE UM BANCO DE DADOS

3.1 Introdução

Uma base de dados nada mais é do que estruturas complexas de dados. Estes dados são gravados em forma de registros em tabelas. Parece simples, certo? Façamos então uma analogia para que este conceito se torne ainda mais simples:

Imagine um arquivo de fichas, numa empresa onde há várias caixas, cada uma contendo os dados dos funcionários de um certo setor. Cada caixa possui várias fichas, que são os cadastros dos funcionários. Cada ficha contém os dados de apenas um funcionário. Indo mais longe, podemos concluir que cada ficha contém diversas informações sobre o funcionário em questão.

Portanto, cada caixa será uma tabela, contendo diversas fichas – os registros, e cada ficha possuirá várias informações sobre o funcionário – os campos.

Como foi dito, há várias caixas, uma para cada departamento, a soma de todas as caixas forma a base de dados.

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

Base de dados < Tabela < Registro < Coluna (tipo de dado)

Banco de dados < Tabela < Linha < Campo

As duas linhas acima mostram os termos normalmente usados para o que acabamos de aprender. Os campos podem ser de diferentes tipos e tamanhos, permitindo ao programador criar tabelas que satisfaçam ao escopo do projeto. A decisão de quais campos usar e quais não usar é muito importante, pois influi drasticamente na performance da base de dados que estamos desenvolvendo. Portanto, é de bom grado um conhecimento sólido destes conceitos.

A etapa de montagem das tabelas é, senão a mais importante, uma das principais etapas da montagem de uma base de dados, pois um bom projeto pode facilitar muito o trabalho de programação.

3.1.1 Campos

Como já sabemos, os campos são a parte fundamental de uma base de dados. É nos campos que as informações ficam armazenadas. Antes de utilizar, devemos definir os campos que desejamos usar, e especificar o que cada um pode conter.

3.1.2 Registros

Um conjunto de campos relacionados, forma o que chamamos de registro (também conhecido como linha). Um registro, portanto, pode ter a seguinte estrutura:

nome CHAR(15);

email CHAR(25);

telefone INT;

Neste exemplo, nosso registro contém três campos, podendo armazenar o email e o telefone de uma determinada pessoa. Nele, cada conjunto dos dados “nome”, “email” e “telefone” corresponde a um registro. Observe que o campo nome foi definido como CHAR, portanto poderá conter qualquer tipo de caracter. Contudo, o campo telefone, definido como INT, poderá apenas conter números, pois foi configurado como INT (veremos os tipos de dados que cada campo deverá ter mais adiante).

3.1.3 Tabelas

Um conjunto de registros, forma uma tabela. As tabelas podem armazenar grandes quantidades de dados. Como no exemplo anterior, poderíamos ter centenas de nomes diferentes cadastrados em nossa tabela de pessoas.

Antes de utilizar uma base de dados, precisamos de uma tabela pelo menos, para armazenar os dados. Não é possível criar duas tabelas com o mesmo nome.

3.1.4 Chave primária

Usada para que não seja permitido que o usuário consiga cadastrar dois registros com identificações iguais. Isto é claramente útil, quando não é desejado que seja digitado um segundo registro igual ao primeiro por engano. Para se definir uma chave primária, basta adicionar 'PRIMARY KEY' à definição do campo que se deseja a não duplicidade.

Exemplo:

nome CHAR(15) PRIMARY KEY;

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

3.2 Níveis de abstração (a arquitetura três esquemas)

Uma das principais características da abordagem banco de dados, é que a mesma fornece alguns níveis de abstração de dados omitindo ao usuário final, detalhes de como estes dados são armazenados.

O principal objetivo dos níveis de abstração (Figura 3) é separar as aplicações do usuário do banco de dados físico. Os níveis podem ser definidos como:

- nível interno: descreve a estrutura de armazenamento físico do banco de dados; utiliza um modelo de dados e descreve detalhadamente os dados armazenados e os caminhos de acesso ao banco de dados.
- nível conceitual: descreve a estrutura do banco de dados como um todo; é uma descrição global do banco de dados, que não fornece detalhes do modo como os dados estão fisicamente armazenados.
- nível externo: ou esquema de visão, descreve as visões do banco de dados para um grupo de usuários; cada visão descreve quais porções do banco de dados um grupo de usuários terá acesso.

3.3 Visões

Como um conjunto de informações pode ser utilizada por um conjunto diferenciado de usuários, é importante que estes usuários possam ter "visões" diferentes da base de dados. Uma "visão" é definida como um subconjunto de uma base de dados, formando deste modo, um conjunto "virtual" de informações.

- VISÃO INTERNA - É aquela vista pelo responsável pela manutenção e desenvolvimento do SGBD. Existe a preocupação com a forma de recuperação e manipulação dos dados dentro do Banco de Dados.
- VISÃO CONCEITUAL - É aquela vista pelo analista de desenvolvimento e pelo administrador das bases de dados. Existe a preocupação na definição de normas e procedimentos para a manipulação dos dados, para garantir a sua segurança e confiabilidade, o desenvolvimento de sistemas e programas aplicativos e a definição no banco de dados de novos arquivos e campos. Na visão conceitual, existem 2 (duas) linguagens de operação que são:
  - Linguagem de definição dos dados (DDL) - Linguagem que define as aplicações, arquivos e campos que irão compor o banco de dados (comandos de criação e atualização da estrutura dos campos dos arquivos).
  - Linguagem de manipulação dos dados (DML) - Linguagem que define os comandos de manipulação e operação dos dados (comandos de consulta e atualização dos dados dos arquivos).
- VISÃO EXTERNA - É aquela vista pelo usuário que opera os sistemas aplicativos, através de interfaces desenvolvidas pelo analista (programas), buscando o atendimento de suas necessidades.

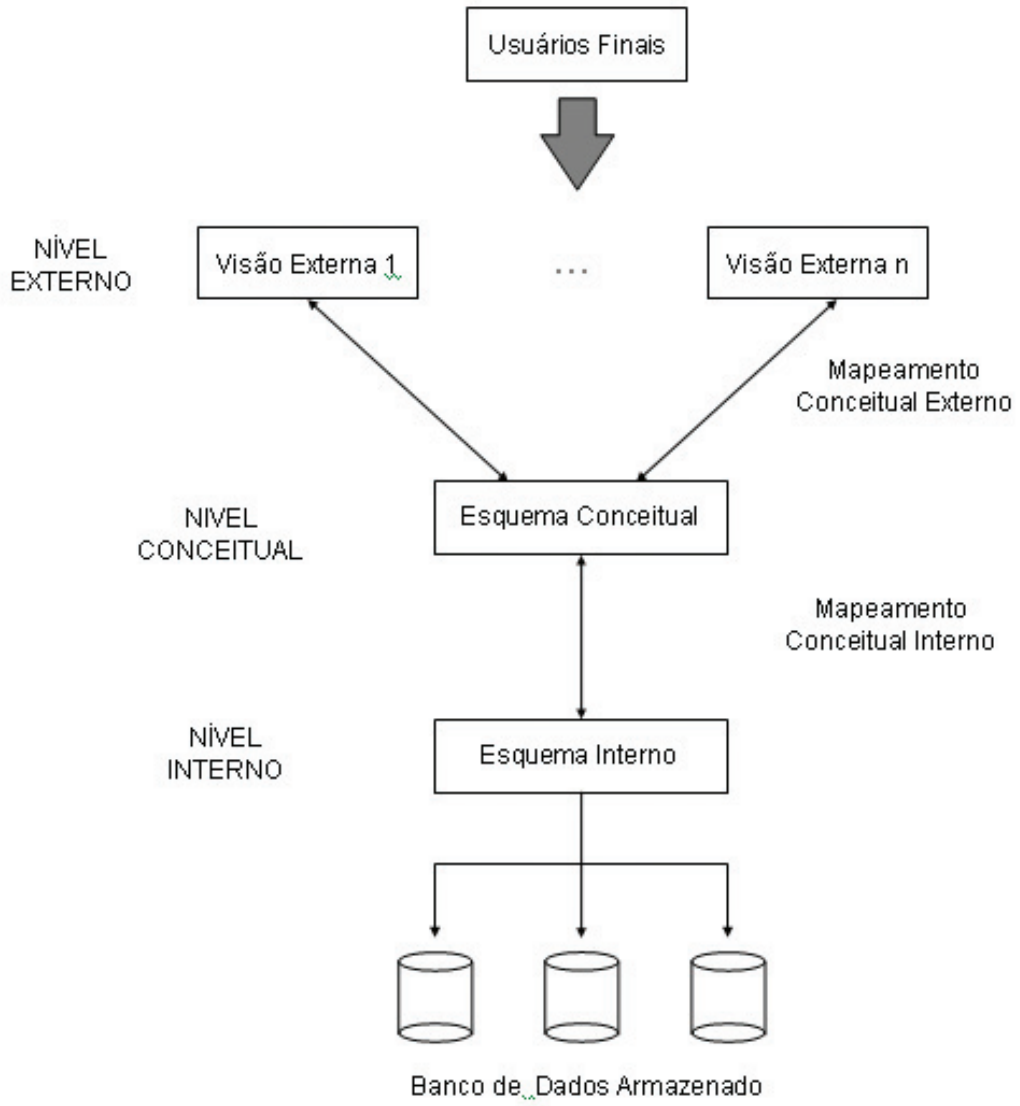


Figura: Arquitetura três esquemas

UTILIZAÇÃO DAS APLICAÇÕES DESENVOLVIDAS	VISÃO EXTERNA
DESENVOLVIMENTO DE APLICAÇÕES UTILIZANDO RECURSOS DO S.G.B.D.	VISÃO CONCEITUAL
DESENVOLVIMENTO DO S.G.B.D	VISÃO INTERNA

3.4 Independência de dados

A "independência de dados" pode ser definida como a capacidade de modificar uma definição de esquema em um nível sem afetar as definições de esquema em um nível superior. Existem dois tipos de independência de dados:

- independência de dados lógica: é a capacidade de se modificar o esquema conceitual sem ter que modificar o esquema externo ou os programas de aplicação.

Ex.: Você decide adicionar uma nova coluna da tabela Estudante do esquema conceitual. Todos os programas continuarão funcionando. Porém, se uma coluna for removida, os programas que fazem uso desta coluna não funcionarão mais.

- independência de dados física: é a capacidade de se modificar o esquema interno sem ter que alterar o esquema conceitual, o esquema externo ou as aplicações do usuário.

Ex.: Você decide modificar o layout do registro físico da tabela Estudante. O esquema conceitual permanece inalterado e todos os programas de aplicações continuarão funcionando.

3.5 Modelos de Dados

Os bancos de dados são diferenciados por muitas características. A mais útil e usada é o modelo de programação.

Um "modelo de dados" é um conjunto de conceitos que podem ser utilizados para descrever a estrutura "lógica" e "física" de um banco de dados. Por "estrutura" podemos compreender o tipo dos dados, os relacionamentos e as restrições que podem recair sobre os dados.

É o conjunto de ferramentas conceituais para a descrição dos dados, dos relacionamentos entre os mesmos e das restrições de consistência e de integridade.

Existem dois tipos de modelos de dados: aqueles baseados em objetos e aqueles baseados em registros.

Por sua vez, nos modelos baseados em objetos podemos encontrar pelo menos dois tipos: o modelo entidade-relacionamento e o modelo orientado a objetos. Já nos modelos baseados em registros, podemos encontrar, por exemplo, o modelo de redes, o modelo hierárquico e o modelo relacional.

Vale também lembrar que este tópico foi abordado na disciplina de Análise de Sistemas. Esta se incumbiu de apresentar as regras e normas para a elaboração destes tipos de modelos. Agora que estamos voltados à construção prática dos bancos de dados, e não de sua construção teórica, apenas citaremos os aspectos básicos da construção teórica, de forma a facilitar o entendimento sobre a relação que existe entre Análise de Sistemas e Banco de Dados.

Por isso, nosso objetivo neste momento será darmos uma relembrada no modelo Entidade-Relacionamento, que tem a responsabilidade de enfocar a elaboração e a construção "teóricas" de um banco de dados, e no modelo relacional, que é responsável por colocar na prática essa elaboração e construção. Então, vamos a eles.

3.5.1 Modelo Entidade-Relacionamento

É um modelo baseado na percepção do mundo real, que consiste em um conjunto de objetos básicos chamados entidades e nos relacionamentos entre esses objetos.

Seu objetivo é mapear o mundo real do sistema em um modelo gráfico que irá representar os objetos básicos e os relacionamentos existentes entre eles.

Uma das ferramentas mais utilizadas para se chegar a este objetivo é uma ferramenta teórica chamada Diagrama Entidade-Relacionamento (DER), que veremos a seguir:

3.5.2 Diagrama Entidade-Relacionamento (DER)

A estrutura lógica geral de um banco de dados pode ser expressa graficamente por um Diagrama Entidade-Relacionamento.

Componentes do Diagrama

• Entidade - Identifica o objeto de interesse do sistema e tem "vida" própria, ou seja, é a representação abstrata de um objeto ou fato do mundo real sobre o qual desejamos guardar informações.

Exemplo: Clientes, Fornecedores, Alunos, Funcionários, Departamentos, etc.

Não são entidades:

- Objeto ou fato com apenas 1 característica;
- Operações do sistema;
- Saídas do sistema;
- Pessoas que realizam trabalhos (usuários do sistema);
- Cargos de direção.

Representação Gráfica: Uma entidade é representada por um retângulo com o seu nome dentro.

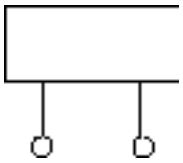
• Atributo – é um elemento de dado que contém informação que descreve uma entidade.

Exemplo: Nome do aluno, Número da turma, Endereço do fornecedor, Sexo do funcionário, etc.

• Atributo Monovalorado: assume um único valor para cada elemento (ocorrência) da entidade. Ex.: nome do funcionário

• Atributo Composto: assume um ou mais sub-atributos. Ex.: endereço do fornecedor

Representação Gráfica: o atributo é representado por uma elipse ou pelo seu nome junto à entidade correspondente.



• Domínio do atributo – é o conjunto de valores permitidos para um atributo.

Exemplo:

Conjunto de valores do atributo Sexo do funcionário: M ou F;

Conjunto de valores do atributo Nome aluno: 40 caracteres alfanuméricos

Conjunto de valores do atributo salário: inteiro maior que 5000

• Relacionamento - representa a associação entre os elementos de duas entidades.

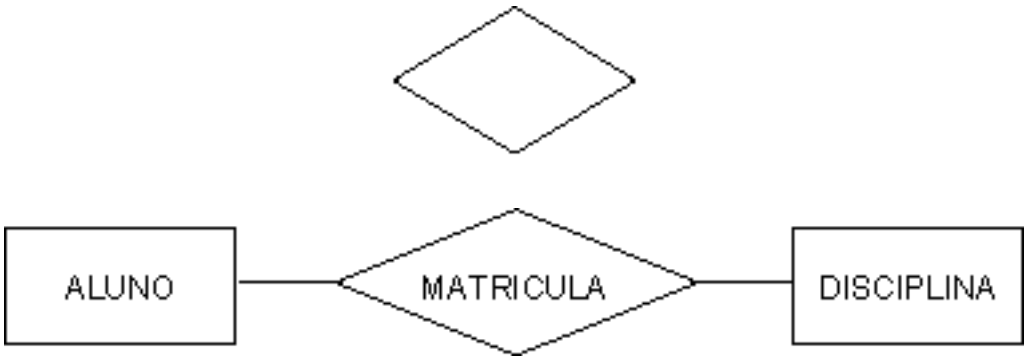
Exemplo:

O João está matriculado na disciplina de Banco de Dados

onde:

- João - Elemento do conjunto de valores do atributo Nome do aluno da entidade Aluno;
- Banco de Dados - Elemento do conjunto de valores do atributo Nome da disciplina da entidade Disciplina;
- matriculado - Ligação (relacionamento) existente entre um aluno e uma disciplina.

Representação Gráfica: o relacionamento é representado por um losango com seu nome no interior, geralmente sendo este um verbo.



Um determinado relacionamento também pode possuir atributos, também conhecido como relacionamento valorado. Esta situação ocorre apenas em relacionamentos (N : M).

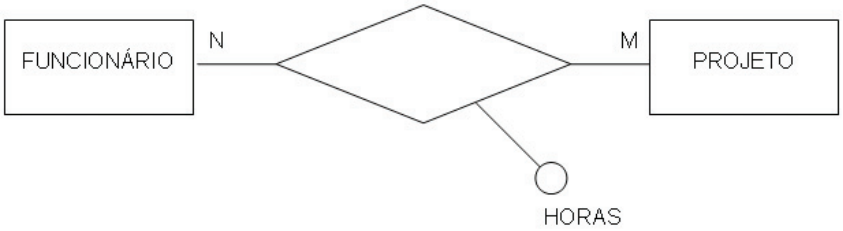
Ex. Pedro trabalha no projeto Alfa 30 horas.

onde:

- Pedro – é um elemento do conjunto de valores do atributo “Nome do funcionário” da entidade “Funcionário”.

- Alfa - é um elemento do conjunto de valores do atributo “Nome do Projeto” da entidade “Projeto”.

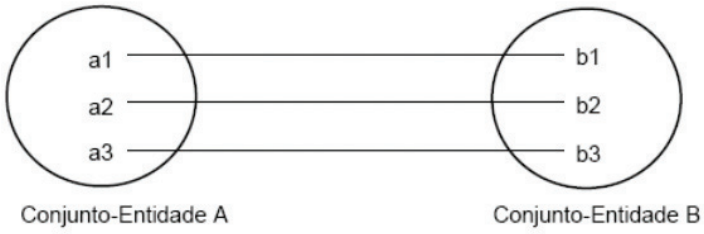
- trabalha – é a ligação existente entre um funcionário e um projeto. Neste caso, este funcionário trabalha 30 horas neste projeto, porém este mesmo funcionário poderá trabalhar outro número de horas em outro projeto, assim como outro funcionário trabalha outro número de horas no mesmo projeto Alfa. Podemos concluir que 30 horas é o atributo que pertence ao Pedro no projeto Alfa, portanto, é um atributo do relacionamento “trabalha”.



• Cardinalidade - Representa a frequência com que o relacionamento existe.

Tipos: (1:1), (1:N) ou (N:1), (N:M)

- Relacionamento 1:1 – uma entidade em A está associada no máximo a uma entidade em B e uma entidade em B está associada no máximo a uma entidade em A.



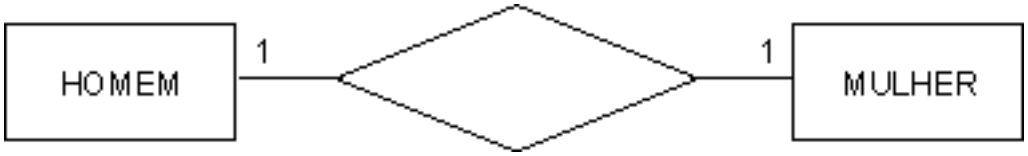
Ex.: João é casado com Maria.

onde:

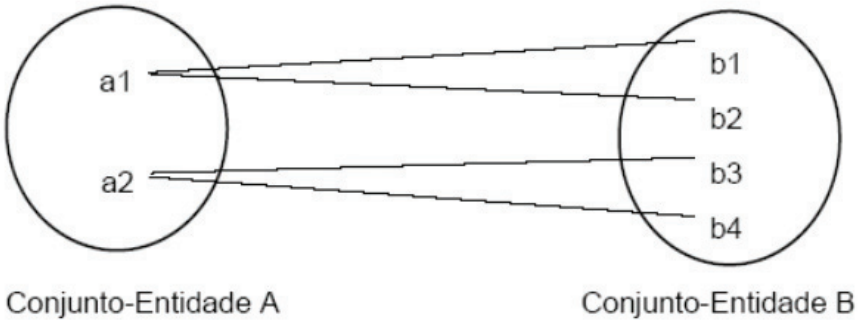
- João - é um elemento do conjunto de valores do atributo “Nome” da entidade “Homem”.

- Maria - é um elemento do conjunto de valores do atributo “Nome” da entidade “Mulher”.

- casado - é a ligação entre um homem e uma mulher, sendo que um homem pode ser casado com uma e apenas uma mulher, assim como uma mulher pode ser casada com um e apenas um homem.



- Relacionamento 1:N ou N:1 – uma entidade em A está associada a qualquer número de entidades em B, enquanto uma entidade em B está associada no máximo a uma entidade em A.



Ex.: Pedro trabalha no Departamento Pessoal.

onde:

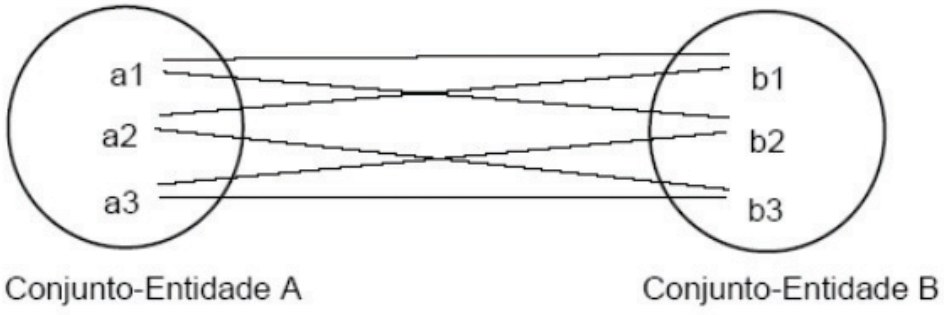
- Pedro - é um elemento do conjunto de valores do atributo "Nome" da entidade "Funcionário".

- Depart. Pessoal - é o elemento do conjunto de valores do atributo "Nome do departamento" da entidade "Departamento".

- trabalha - é a ligação entre um Funcionário e um Departamento, onde um funcionário pode trabalhar em um e somente um departamento e um departamento pode ter vários funcionários.

OBS.: A chave estrangeira aparece na direção "muitos".

- Relacionamento N : M – Uma entidade em A está associada a qualquer número de entidades em B, e uma entidade em B está associada a qualquer número de entidades em A.



Ex.: Antonio está matriculado na disciplina Banco de Dados.

onde:

- Antonio – é um elemento do conjunto de valores do atributo "Nome" da entidade "Aluno".

- Banco de Dados – é um elemento do conjunto de valores do atributo "Nome da Disciplina" da entidade "Disciplina".

- matriculado – é a ligação existente entre um aluno e uma disciplina, onde um aluno pode estar matriculado em várias

disciplinas e cada disciplina pode ter vários alunos matriculados.

OBS.: Este tipo de relacionamento requer uma entidade extra para ser representado.

• Chaves – é um conjunto de um ou mais atributos que, tomados coletivamente, permitem-nos identificar unicamente uma ocorrência (registro) dentro de uma entidade.

Integridade de entidade: nenhum atributo que participe da chave de uma entidade deve aceitar valores nulos.

Aspectos relevantes:

• A questão fundamental do projeto de chaves é reduzir ao máximo os efeitos de redundância.

• A alteração dos valores de campos da chave primária ou a remoção de uma ocorrência em uma entidade pode ocasionar problemas de integridade referencial.

• **Restrições de Integridade**

Um Banco de Dados armazena informação sobre parte de um mundo real que chamamos de mini-mundo. Certas regras, denominadas de Restrições de Integridade, governam este mini-mundo. As restrições de Integridade especificam os estados do BD que são considerados consistentes. O estado de um Banco de Dados é dito consistente ou válido se todas as restrições de integridade forem satisfeitas.

3.5.3 Modelo Relacional

Os bancos de dados relacionais são constituídos não de tabelas, mas de três componentes: uma coleção de estruturas de dados, a saber relações; um conjunto de operadores, a saber a álgebra e o cálculo relacionais; e um conjunto de restrições de integridade que definem os conjuntos de estados e mudanças de estado consistentes do banco de dados.

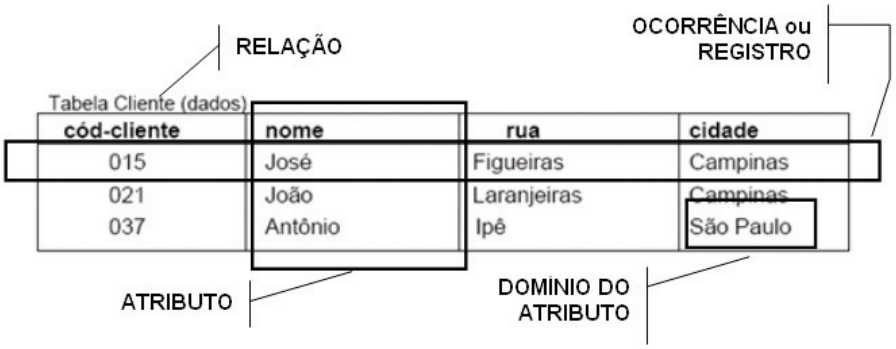
O modelo relacional foi criado por Codd em 1970 e tem por finalidade representar os dados como uma coleção de relações, onde cada relação é representada por uma tabela (entidade), ou, falando de uma forma mais direta, um arquivo. Porém, um arquivo é mais restrito que uma tabela. Toda tabela pode ser considerada um arquivo, porém, nem todo arquivo pode ser considerado uma tabela.

Quando uma relação é pensada como uma tabela de valores, cada linha nesta tabela representa uma coleção de dados relacionados. Estes valores podem ser interpretados como fatos descrevendo uma instância (ocorrência) de uma entidade ou de um relacionamento. O nome da tabela e das colunas desta tabela são utilizados para facilitar a interpretação dos valores armazenados em cada linha da tabela. 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 (ou ocorrência);
- o nome de cada coluna é chamado de atributo;
- o tipo de dado que descreve cada coluna é chamado de domínio.

Vamos enxergar tudo isso:



Os nomes das tabelas e dos campos são de fundamental importância para nossa compreensão entre o que estamos armazenando, onde estamos armazenando e qual a relação existente entre os dados armazenados.

Um domínio é um conjunto de valores atômicos, sendo que por atômico podemos entender que cada valor do domínio é indivisível. Durante a especificação do domínio é importante destacar o tipo, o tamanho e a faixa do atributo que está sendo especificado. Por exemplo, Rio de Janeiro, Paraná e Pará são estados válidos para o Brasil, enquanto que Corrientes não é um estado válido (pertence a Argentina e não ao Brasil). Outras observações são:

• As relações não podem ser duplicadas (não podem existir dois estados do Pará, no conjunto de estados brasileiros, por exemplo);

• A ordem de entrada de dados no Banco de Dados não deverá ter qualquer importância para as relações, no que concerne ao seu tratamento;

• Os atributos deverão ser atômicos, isto é, não são passíveis de novas divisões.

Este é um exemplo de informações em um banco de dados:

Tabela Cliente (dados)

cód-cliente	nome	rua	cidade
015	José	Figueiras	Campinas
021	João	Laranjeiras	Campinas
037	Antônio	Ipê	São Paulo

Tabela Conta (dados)

nro-conta	saldo
900	55
556	1.000
647	5.366
801	10.533

Tabela Cliente-Conta (relacionamento)

cód-cliente	nro-conta
015	900
021	556
021	647
037	647
037	801

3.6 Esquemas e Instâncias

Em qualquer modelo de dados utilizado, é importante distingüir a “descrição” do banco de dados do “banco de dados” por si próprio. Saberemos a diferença entendendo o que é esquema e o que é instância em um banco de dados.

3.6.1 Esquema

O esquema de uma relação é a descrição da estrutura dos dados e as restrições que precisam ser obedecidas em um banco de dados. Ele nada mais é do que os campos (colunas) existentes em uma tabela. O esquema raramente muda com o tempo.

3.6.2 Instância

A instância de uma relação consiste no conjunto de valores que cada atributo recebe em um determinado instante. Portanto, os dados armazenados no banco de dados, são formados pelas instâncias das relações. É também chamada de "estado" ou "ocorrência" do banco de dados. A instância muda freqüentemente pois ela é alterada toda vez que uma alteração é feita no banco de dados.

O SGBD é responsável por garantir que toda instância do banco de dados satisfaça ao esquema do banco de dados, respeitando sua estrutura e suas restrições. O esquema de um banco de dados também pode ser chamado de "intensão" de um banco de dados e a instância de "extensão" de um banco de dados.

3.7 Normalização

O processo de normalização pode ser visto como o processo no qual são eliminados esquemas de relações (tabelas) não satisfatórios, decompondo-os, através da separação de seus atributos em esquemas de relações menos complexas mas que satisfaçam as propriedades desejadas.

O processo de normalização como foi proposto inicialmente por Codd conduz um esquema de relação através de um bateria de testes para certificar se o mesmo está na 1ª, 2ª e 3ª Formas Normais. Estas três Formas Normais são baseadas em dependências funcionais dos atributos do esquema de relação. Na prática, os procedimentos de normalização consideram-se geralmente satisfatórios se as tabelas atingirem a terceira Forma Normal (3FN).

Vamos a elas.

1ª FORMA NORMAL

Uma tabela encontra-se na primeira forma normal (1FN) se todos os seus atributos estiverem definidos em domínios que contenham apenas valores atômicos (indivisíveis). Dito por outras palavras: os domínios devem ser formados por valores elementares e não por conjuntos de valores. Leve em consideração o esquema a seguir:

Imagine-se que é criada uma tabela destinada a registrar a informação sobre os alunos e sobre as disciplinas em que esses alunos estão matriculados.

Aluno (IdAluno, Nome, Endereço, Disciplinas)

Tabela: Aluno

IdAluno	Nome	Endereço	Disciplinas
A1	João	Santo Ângelo	Matemática, Economia, Direito
A2	Ana	São Luiz Gonzaga	Matemática, Física
A3	Pedro	Entre-Ijuis	Matemática, Economia, Física
A4	Felipe	Santo Ângelo	Matemática

Esta tabela não obedece à primeira forma normal (1FN), uma vez que o atributo Disciplinas admite conjuntos de valores.

Pode-se resolver este problema repetindo valores nas linhas da tabela, de forma que, em cada linha, o atributo Disciplinas contenha um único valor.

Os valores da tabela seriam os seguintes:

Tabela: Aluno

IdAluno	Nome	Endereço	Disciplinas
A1	João	Santo Ângelo	Matemática
A1	João	Santo Ângelo	Economia
A1	João	Santo Ângelo	Direito
A2	Ana	São Luiz Gonzaga	Matemática
A2	Ana	São Luiz Gonzaga	Física
A3	Pedro	Entre-Ijuis	Matemática
A3	Pedro	Entre-Ijuis	Economia
A3	Pedro	Entre-Ijuis	Física
A4	Felipe	Santo Ângelo	Matemática

Esta tabela encontra-se agora na primeira forma normal (1FN), pois todos os atributos contêm valores elementares. No entanto, apresenta uma evidente redundância de informação, que se reflete na repetição dos identificadores, dos nomes e dos endereços dos alunos.

Como se não bastasse, pode-se apontar ainda:

Problemas de atualização

Se o endereço de um aluno for alterado, essa alteração precisaria ser feita em várias linhas da tabela, sob pena de gerar um estado de inconsistência da base de dados.

Problemas de inserção

- Com esta estrutura não é possível registrar informação relativa a um aluno que, por qualquer razão, não esteja matriculado a nenhuma disciplina. Imagine-se, por exemplo, o caso de um aluno "externo" que precise apenas efetuar exames, mas que não está matriculado a nenhuma disciplina.
- Problemas de eliminação

Observe-se na tabela anterior o caso do aluno Felipe, que está matriculado apenas a uma disciplina. Se essa matrícula for anulada, perde-se, não apenas, a informação referente à matrícula, mas também a informação relativa aos restantes atributos do aluno.

Os problemas referidos podem ser eliminados criando:

- uma tabela para registrar apenas os dados dos alunos
- uma tabela para registrar apenas os dados relativos às disciplinas
- uma tabela para representar as matrículas

Logo, o esquema dessas tabela poderia ser o seguinte:

Aluno (IdAluno, Nome, Endereço,…)

Disciplina (IdDisciplina, CH,…)

Matrícula (IdAluno, IdDisciplina, Data,…)

Esta decomposição da tabela permite eliminar os problemas anteriormente referidos, relacionados com a atualização, a inserção e a eliminação de dados nas tabelas, pois a primeira forma normal (1FN) tem como principal objetivo remover grupos repetitivos de dados.

2ª FORMA NORMAL

Para se compreender a representação da segunda forma normal (2FM), é necessário antes de mais apresentar o conceito de dependência ou dependência funcional entre atributos.

Dependência funcional

Considere-se a tabela A com os atributos x, y, z e w

A (x, y, z, w)

O atributo z é funcionalmente dependente do atributo x se, para um dado valor x, o valor de z é sempre o mesmo. Ou seja:

conhecido x, sabe-se automaticamente qual é o valor de z.

Existe assim uma dependência funcional entre o atributo x e o atributo z. O atributo x designa-se por determinante; o atributo z designa-se por dependente.

Consideremos a seguinte tabela:

Aluno (IdAluno, Nome, Endereço, IdDiscilina, CHDisciplina)

Como se pode notar, o atributo Nome depende funcionalmente do atributo IdAluno, visto que o identificador de um aluno determina o nome desse aluno. O mesmo se pode observar relativamente ao atributo CHDisciplina. Este atributo depende funcionalmente do atributo IdDisciplina.

Conclusão: esta tabela não está na segunda forma normal (2FN). A transformação feita em um exemplo anterior conduziu ao seguinte conjunto de tabelas:

Aluno (IdAluno, Nome, Endereço,…)

Disciplina (IdDisciplina, CH,…)

Matrícula (IdAluno, IdDisciplina, Data,…)

Pode-se verificar que todas estas tabelas estão na segunda forma normal (2FN). Isso é evidente em relação às duas primeiras, Aluno e Disciplina.

3ª FORMA NORMAL

Uma tabela encontra-se na terceira forma normal (3FN) se:

- estiver na segunda forma normal e
- nenhum dos atributos que não fazem parte da chave for funcionalmente dependente de qualquer combinação dos restantes.

Para que uma tabela esteja na terceira forma normal (3FN) é necessário que todos os atributos que não pertencem à chave sejam mutuamente independentes.

Considere-se uma tabela destinada a registar informação sobre jogos de futebol e árbitros, com a seguinte estrutura:

Jogo (IdJogo, Estádio, Nome-árbitro, Categoria-árbitro)

Nesta tabela verifica-se o seguinte:

- Todos os atributos são funcionalmente dependentes da chave.
- Existe uma situação de transitividade: o atributo categoria-árbitro depende funcionalmente do atributo Nome-árbitro. Os atributos Nome-árbitro e Categoria-árbitro não são independentes.

A tabela não se encontra, por isso, na terceira forma normal. Esta situação provoca as já referidas dificuldades ao nível da atualização, inserção e eliminação de dados:

- Problemas de atualização

Se o árbitro mudar de categoria, essa alteração deve ser efetuada em todas as ocorrências desse valor. Se existisse uma tabela própria para registar os dados referentes aos árbitros, a alteração da categoria seria feita num único registrado.

- Problemas de inserção

Se um árbitro ainda não tiver arbitrado nenhum jogo, mas já estiver inscrito na respectiva associação com determinada categoria, ele não pode ser registrado nesta tabela (mas poderia sê-lo se existisse uma tabela independente para árbitros).

- Problemas de eliminação

Admita-se por hipótese que determinado árbitro arbitrou um só jogo. Se for eliminado o registro relativo a esse jogo, elimina-se também a informação relativa ao árbitro.

A normalização transforma esta tabela no seguinte conjunto de tabelas, todas elas na 3FN:

Jogo (IdJogo, Estádio)

Árbitro (IdÁrbitro, Nome-árbitro, Categoria-árbitro)

Jogo-Árbitro (Idjogo, IdÁrbitro)

4 LINGUAGEM SQL

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas a sua manipulação. SQL (Structured Query Language – Linguagem Estruturada para Pesquisas) foi originalmente criada pela IBM, mas por causa da criação de "dialetos" para ela, feitos por muitos desenvolvedores, foi adotado um padrão para a linguagem pelo ANSI (American National Standards Institute), em 1986 e pela ISSO, em 1987. Assim, ela ficou conhecida como uma linguagem de computador para acessar e manipular bancos de dados.

SQL, embora padronizado pelo ANSI e pela ISO, tem muitas variações e extensões por causa dos diferentes fabricantes de sistemas gerenciadores de banco de dados. Normalmente a linguagem pode ser aportada de plataforma para plataforma sem mudanças estruturais principais.

Declarações SQL são usadas para recuperar e atualizar dados em um banco de dados. SQL trabalha com programas de banco de dados como DB2, Oracle, Sybase, MySQL, MS SQL Server, etc.

A Linguagem SQL tem como grandes virtudes e características:

- sua capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro.
- sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens independentes das tabelas e organização lógica dos dados.
- a capacidade que dispomos de cancelar uma série de atualizações ou de as gravarmos, depois de iniciarmos uma seqüência de atualizações. Os comandos "Commit" e "Rollback" são responsáveis por estas facilidades.
- SQL representa um conjunto de comandos responsáveis pela:
- Definição dos dados das tabelas, chamada de DDL (Data Definition Language);
- A DDL é um conjunto de comandos e atualização dos dados em um Sistema Gerenciador de Banco de Dados (SGBD). Como exemplo de comandos da classe DDL temos os comandos Create, Alter e Drop.
- Manipulação dos mesmos dados, chamada de DML (Data Manipulation Language).
- Os comandos da série DML são destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML temos os comandos Select, Insert, Update e Delete.

4.1 Linguagem de manipulação de dados (DML)

É utilizada para especificar consultas e atualizações em um BD. Os comandos DML podem ser utilizados diretamente ou serem embutidos (embedded) em uma linguagem de programação.

As aplicações fazem seus acessos ao pré-compilador DML da linguagem hospedeira, que os envia ao Compilador DML (Data Manipulation Language) onde são gerados os códigos de acesso ao BD.

Uma vez que o esquema esteja compilado e o banco de dados esteja populado, usa-se a DML (Data Manipulation Language - Linguagem de Manipulação de Dados) para fazer a manipulação dos dados. Esta linguagem é responsável pela:

- recuperação da informação armazenada
- inserção de novas informações
- exclusão de informações
- modificação de dados armazenados

A DML permite ao usuário acessar ou manipular os dados, vendo-os da forma como são definidos no nível de abstração mais alto do modelo de dados utilizado.

A parte de uma DML que envolve recuperação de informação é chamada linguagem de consulta. Uma consulta ("query") é um comando que requisita uma recuperação de informação.

4.1.1 Comandos DML (Data Manipulation Language)

São os comandos responsáveis pela consulta e atualização dos dados armazenados em um banco de dados.

4.1.2 SELECT

Objetivo:

Selecionar um conjunto de registros em uma ou mais tabelas que atenda a uma determinada condição definida pelo comando.

Sintaxe:

```
SELECT ALL FROM <nome-tabela>  [, <nome-tabela>]
    DISTINCT
  WHERE <condição>
  GROUP BY <nome-coluna>
  HAVING <condição>
  ORDER BY <nome-campo>  ASC
                         DESC
```

onde:

- nome-tabela - representa o nome da(s) tabela(s) que contém as colunas que serão selecionadas ou que serão utilizadas para a execução da consulta.
- condição - representa a condição para a seleção dos registros. Esta seleção poderá resultar em um ou vários registros.
- nome-coluna - representa a(s) coluna(s) cujos resultados são grupados para atender à consulta.
- ALL - opção default. Mostra todos os valores obtidos na seleção.
- DISTINCT - opção que mostra os valores obtidos na seleção eliminando as duplicidades.
- WHERE - especifica o critério de seleção dos registros nas tabelas especificadas.
- GROUP BY - especifica o(s) campo(s) que será(ão) agrupados para atender a consulta.
- HAVING - especifica uma condição para seleção de um grupo de dados. Esta opção só é utilizada combinada com a opção GROUP BY.
- ORDER BY - esta opção, quando utilizada apresenta o resultado da consulta ordenado de forma crescente ou decrescente pelos campos definidos.

ESTUDO DE CASOS DO COMANDO SELECT

Caso 1: Selecionar todos os campos (colunas) da tabela "departamento".

```
SELECT * FROM dept;
```

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

4.1.3 Cláusula WHERE

A cláusula "WHERE" corresponde ao operador restrição, da Álgebra Relacional. Ela 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, é mostrada uma lista de operadores lógicos e complementares que podem ser utilizados nas expressões apresentadas em "WHERE".

4.1.4 Operadores lógicos

Operador	Significado
=	Igual a
>	Maior que
>=	Maior ou igual

<	Menor que
<=	Menor ou igual

Exemplos:

```
SELECT  empnome, empserv
    FROM  emp
    WHERE  depnume > 10;

SELECT  empnome, empserv
    FROM  emp
    WHERE empserv = 'gerente';
```

O conjunto de caracteres ou datas deve estar entre apóstrofes (') na cláusula "WHERE".

Caso 2: Selecionar todos os departamentos cujo orçamento mensal seja maior que 100000. Apresente o nome de tal departamento e seu orçamento anual, que será obtido multiplicando-se o orçamento mensal por 12.

Resposta: 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;
```

Caso 3: Apresentar a instrução anterior, porém, ao invés dos nomes de campos DepNome e DepOrca, trocar para Departamento e Orçamento.

Resposta: Neste exemplo, deveremos denominar colunas por apelidos. Os nomes das colunas, mostradas por uma consulta, são geralmente os nomes existentes no Dicionário de Dados, porém 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 os apresentar 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;
```

4.1.5 Cláusula DISTINCT

Caso 4: Apresentar todos os salários existentes na empresa, porém omitir eventuais duplicidades.

Resposta: A cláusula "DISTINCT" elimina duplicidades, significando que somente relações distintas serão apresentadas como resultado de uma pesquisa.

```
SELECT DISTINCT empserv
    FROM emp;
```

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

Resposta: Deseja-se um tratamento diferenciado para valores nulos. Qualquer coluna de uma tupla que não contiver 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 e são tratados de forma diferenciada pelo SQL.

```
SELECT  empnome, empsala + empcomi
   FROM emp;

SELECT  empnome, NVL(empsala,0) + NVL(empcomi,0)
   FROM emp;
```

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

4.1.6 Cláusula ORDER BY

Caso 6: Apresentar os nomes e funções da cada funcionário contidos na tabela Empresa, porém, classificados alfabeticamente (A..Z) e depois alfabeticamente invertido (Z..A).

Resposta: A cláusula "ORDER BY" modificará a ordem de apresentação do resultado da pesquisa (ascendente ou descendente).

```
SELECT  empnome, empserv
   FROM  emp
   ORDER  BY  empnome [DESC] ;
```

Caso 7: Selecionar os nomes dos Departamentos que estejam na fábrica.

Resposta:

```
SELECT depnome
   FROM dept
   WHERE deploca = "Sao Paulo";
```

O exemplo exigiu uma restrição (São Paulo) que nos obrigou a utilizar a instrução "WHERE".

4.1.7 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 seqüência de nenhum até vários caracteres).

4.1.8 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

Caso 8: Selecionar os empregados cujos salários sejam menores que 1000 ou maiores que 3500.

Resposta: Será necessário aqui a utilização de expressão negativa.

```
  SELECT empnome,  empsala
    FROM  emp
    WHERE  empsala  NOT  BETWEEN  1000  AND  3500;
```

Caso 9: Apresentar todos os funcionários com salários entre 200 e 700 e que sejam vendedores.

Resposta: Será necessário consultas com condições múltiplas. Os operadores "AND" (E) e "OR" (OU) fazem isto.

```
 SELECT empnome, empsala, empserv
    FROM  emp
    WHERE  empsala  BETWEEN  700  AND  2000
           AND  empserv =  'vendedor';
```

Caso 10: Apresentar todos os funcionários com salários entre 200 e 700 ou que sejam vendedores.

Resposta:

```
  SELECT empnome, empsala, empserv
    FROM  emp
    WHERE empsala  BETWEEN  700  AND  2000
           OR  empserv =  'vendedor';
```

Caso 11: Apresentar todos os funcionários com salários entre 200 e 700 e que sejam vendedores ou balconistas.

Resposta:

```
  SELECT empnome, empsala, empserv
    FROM  emp
    WHERE  empsala  BETWEEN  700  AND  2000
```

```
AND ( empserv =  'balconista' OR  empserv =  'vendedor' );
```

4.1.9 Cláusula HAVING

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

Caso 12: Retomar o problema anterior, porém apresentar resposta apenas para departamentos com mais de 10 empregados.

Resposta:

```
SELECT  depnume, AVG(empsala)
    FROM  emp
    GROUP BY depnume
    HAVING COUNT(*)  >  3;
```

Obs.: A claúsula "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.

Aqui, vamos exemplificar um erro típico, no exemplo de restringir a média maior que 1000:

Seleção errada	Seleção adequada
SELECT depnume, AVG(empsala) FROM emp WHERE AVG(SALARIO) > 1000 GROUP BY depnume;	SELECT depnume, AVG(empsala) FROM emp GROUP BY depnume HAVING AVG(empsala) > 1000;

4.1.10 Seqüência 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 linguagem 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.

4.1.11 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 se tiver definido de forma adequada a chave estrangeira de uma tabela e sua referência à chave primária da tabela precedente.

Caso 13: Listar nomes de empregados, cargos e nome do departamento onde o empregado trabalha.

Resposta: Pode-se observar que dois dos três dados solicitados estão na tabela Emp, enquanto o outro dado está na tabela Dept. Deve-se então acessar os dados restringindo convenientemente as relações existentes entre as tabelas. De fato sabe-se 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.: Pode-se notar que, quando as tabelas contém colunas com o mesmo nome, usa-se um apelido: o “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 ambigüidade fosse verificada. Embora SQL resolva de forma muito elegante o problema da nomenclatura idêntica para campos de tabelas, ainda assim recomenda-se evitar 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?

Caso 14: Listar os códigos de cada funcionário, seus nomes, seus cargos e o nome do gerente ao qual este se relaciona.

Resposta: Precisa-se criar um auto-relacionamento, ou seja, juntar uma tabela a ela própria. É possível juntar uma tabela a ela mesma com a utilização de apelidos, permitindo juntar ocorrências da tabela a outras ocorrências da mesma tabela.

```
SELECT  A.empnume, A.empnome, A.empserv, B.empnome
   FROM  emp A, emp B
   WHERE  A.empgere  = B.empnume;
```

4.1.12 As Sub-Consultas

Uma sub-consulta é uma instrução “SELECT” aninhada dentro de uma “SELECT”, “INSERT”, “DELETE” ou “UPDATE”, ou dentro de uma outra sub-consulta, e que devolve resultados intermediários.

Sintaxe

Você pode usar três formas de sintaxe para criar uma subconsulta:

• comparação [ANY | ALL | SOME] (instruçãosql)

• expressão [NOT] IN (instruçãosql)

• [NOT] EXISTS (instruçãosql)

Uma subconsulta possui as partes abaixo:

Parte	**Descrição**
Comparação	Uma expressão e um operador de comparação que compara a expressão com o resultado da subconsulta.
Expressão	Uma expressão para a qual o resultado definido da subconsulta é procurado.
Instrução SQL	Uma instrução SELECT de acordo com as mesmas regras e formato de qualquer outra instrução SELECT. Ela deve estar entre parênteses.

Comentários

• Você pode usar uma subconsulta ao invés de usar uma expressão na lista de campo de uma instrução SELECT ou em uma cláusula WHERE ou HAVING. Em uma subconsulta, você usa uma instrução SELECT para fornecer um conjunto de um ou mais valores específicos para avaliar as expressões das cláusulas WHERE ou HAVING.

• Use o predicado ANY ou SOME, que são sinônimos, para recuperar registros na consulta principal que satisfaçam a comparação com quaisquer registros recuperados na subconsulta. O exemplo abaixo retorna todos os produtos cujo preço unitário é maior que o preço de qualquer produto vendido com um desconto de 25 por cento ou mais:

```
SELECT *
  FROM produto
  WHERE preçounit > ANY (SELECT preçounit
                         FROM pedido_detalhes
                         WHERE desconto >= .25);
```

• Use o predicado ALL para recuperar apenas os registros na consulta principal que satisfaçam a comparação com todos os registros recuperados na subconsulta. Se você mudasse ANY para ALL no exemplo acima, a consulta retornaria apenas os

produtos cujo preço unitário fosse maior que o de todos os produtos vendidos com um desconto de 25 por cento ou mais. Isto é muito mais restritivo.

• Use o predicado IN para recuperar apenas os registros na consulta principal para os quais alguns registros na subconsulta contêm um valor igual. O exemplo abaixo retorna todos os produtos com um desconto de 25 por cento ou mais:

```
SELECT *
  FROM produto
  WHERE produtoID IN
    (SELECT produtoID FROM pedido_detalhes WHERE desconto >= .25);
```

• De maneira contrária, você pode usar NOT IN para recuperar apenas os registros na consulta principal para os quais não existam registros com valores iguais na subconsulta. Utilize o predicado EXISTS (com a palavra reservada NOT opcionalmente) em comparações TRUE/FALSE para determinar se a subconsulta retorna algum registro.

• Você também pode usar "aliases" de nomes de tabelas em uma subconsulta para fazer referência a tabelas listadas em uma cláusula FROM fora da subconsulta. O exemplo abaixo retorna os nomes dos funcionários cujos salários sejam iguais ou superiores à média de salários de todos os funcionários na mesma função. Para a tabela Funcionários é dado o alias "T1":

```
SELECT sobrenome, nome, título, salário
  FROM funcionário AS T1
  WHERE salário >= (SELECT AVG(salário)
                    FROM funcionário
                    WHERE T1. T1.título = funcionário.título)
  ORDER BY title;
```

No exemplo acima, a palavra reservada AS é opcional. Algumas subconsultas são aceitas em consultas de tabela cruzada especialmente como predicados (as da cláusula WHERE). Subconsultas como saída (as da lista SELECT) não são aceitas em tabelas de referência cruzada.

Caso 15: Listar o nome, o título e o salário de todos os representantes de vendas cujos salários sejam superiores aos de todos os gerentes e diretores.

Resposta:

```
SELECT sobrenome, nome, título, salário
 FROM funcionário
 WHERE título LIKE "*repr vendas*" AND salário > ALL
   (SELECT salário
        FROM funcionário
        WHERE (título LIKE "*gerente*")
        OR (título LIKE "*diretor*"));
```

Caso 16: listar o nome e o preço unitário de todos os produtos cujo preço unitário seja igual ao do Licor de Cacau.

Resposta:

```
SELECT nome_produto, preçounit
 FROM produto
 WHERE preçounit = (SELECT preçounit
                    FROM [produto]
                    WHERE nome_produto = "Licor de Cacau");
```

Caso 17: listar a empresa e o contato de cada empresa de todos os clientes que fizeram pedidos no segundo trimestre de 1995.

Resposta:

```
SELECT nome_contato, nome_empresa, contato_título, fone
 FROM cliente
 WHERE clienteID IN   (SELECT clienteID
```

```
                    FROM pedido
                    WHERE data_pedido BETWEEN #1/04/95# AND
                                              #1/07/95#);
```

Caso 18: listar os funcionários cujo salário seja maior que a média dos salários de todos os funcionários.

Resposta:

```
SELECT sobrenome, nome, título, salário
 FROM funcionário T1
 WHERE salário >=   (SELECT AVG (salário)
                    FROM funcionário
                    WHERE funcionário.título = T1.título)
                    ORDER BY título;
```

Caso 19: selecionar o nome de todos os funcionários que tenham registrado pelo menos um pedido.

Resposta:

```
SELECT nome, sobrenome
 FROM funcionário AS E
 WHERE EXISTS   (SELECT *
                    FROM pedido AS O
                    WHERE O.funcionárioID = E.funcionárioID);
```

Caso 20: Alterar o campo "Efetuado" do arquivo de serviços para 2 caso o mesmo tenha parecer técnico da entidade "encaminhanhamento" diferente de nulo.

Resposta:

```
UPDATE servico SET efetuado = 2
    WHERE numero_servico = ANY
     (SELECT servico.numero_servico
            FROM servico
                INNER JOIN encaminhamento
                ON (servico.numero_servico = encaminhamento.numero_servico)
                AND (servico. ano_servico = encaminhamento.ano_servico)
            WHERE (((servico.efetuado) Is Null) AND
                ((encaminhamento.parecer_tecnico) Is Not Null))
            GROUP BY servico.numero_servico
            ORDER BY servico.numero_servico);
```

Caso 21: Relacionar todos os nomes de funcionários e seus respectivos cargos, desde que o orçamento do departamento seja igual a 300000.

Resposta:

```
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 sub-consulta.

Caso 22: Relacionar todos os departamentos que possuem empregados com remuneração maior que 3500.

Resposta:

```
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 verficar a não existência (NOT EXISTS) caso esta alternativa seja mais conveniente.

4.1.13 Uniões

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

Caso 23: Listar todos os empregados que tenham código maior que 10 ou funcionários que trabalhem em departamentos com código maior que 10.

Resposta:

```
(SELECT *
   FROM  emp
   WHERE empnume > 10)
UNION
(SELECT *
   FROM emp
   WHERE depnume > 10);
```

4.1.14 Comando INSERT INTO

Objetivo:

Incluir um novo registro em uma tabela do banco de dados.

Sintaxe:

```
INSERT INTO  <nome-tabela> [(<nome-coluna>, [<nome-coluna>])]
  VALUES (<relação dos valores a serem incluídos>);
```

onde:

− nome-tabela - representa o nome da tabela onde será incluído o registro.

− nome-coluna - representa o nome da(s) coluna(s) que terá(ão) conteúdo no momento da operação de inclusão.

Obs.: Este comando pode ser executado de duas maneiras:

- Quando todos os campos da tabela tiverem conteúdo - Neste caso, não é necessário especificar as colunas, entretanto a relação dos valores a serem incluídos deverão obedecer a mesma seqüência da definição da tabela.
- Quando apenas parte dos campos da tabela tiverem conteúdo - Neste caso, devem ser especificadas todas as colunas que terão conteúdo e os valores relacionados deverão obedecer esta seqüência. Para os campos que não têm conteúdo especificado, será preenchido o valor NULL.

Exemplos:

Possibilita a inserção de registros de forma interativa.

```
INSERT INTO dept;
```

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

```
INSERT INTO dept (depnume,depnome,deploca)
 VALUES (70,"producao","Rio de Janeiro");
```

4.1.15 Comando UPDATE

Objetivo:

Atualizar os dados de um ou um grupo de registros em uma tabela do banco de dados.

Sintaxe:

```
UPDATE <nome-tabela>
  SET <nome-coluna> = <novo conteúdo para o campo>
        [<nome-coluna> = <novo conteúdo para o campo>]
        WHERE <condição>
```

onde:

• nome-tabela - representa o nome da tabela cujo conteúdo será alterado.

• nome-coluna - representa o nome da(s) coluna(s) que terá(ão) seus conteúdos alterados com o novo valor especificado.

• condição - representa a condição para a seleção dos registros que serão atualizados. Esta seleção poderá resultar em um ou vários registros. Neste caso a alteração irá ocorrer em todos os registros selecionados.

Exemplo:

```
UPDATE emp
 SET empsala = empsala* 1.2
 WHERE empsala< 1000;
```

UPDATE é especialmente útil quando se quer alterar muitos registros ou quando os registros que se quer alterar estão em várias tabelas. Pode-se alterar vários campos ao mesmo tempo. O exemplo abaixo aumenta o "valor do pedido" em 10 por cento e o "valor do frete" em 3 por cento para embarques do Reino Unido:

```
UPDATE pedido
 SET valor_pedido = valor_pedido * 1.1, frete = frete * 1.03
 WHERE país_embarque = 'RU';
```

UPDATE não gera um conjunto de resultados. Se você quiser saber quais resultados serão alterados, examine primeiro os resultados da consulta seleção que use os mesmos critérios e então execute a consulta atualização.

Exemplos de instrução UPDATE

Esse exemplo muda os valores no campo "relatório_para" para 5 para todos os registros de funcionários que atualmente têm valores de "relatório_para" de 2.

```
UPDATE funcionário
 SET relatório_para = 5
 WHERE relatório_para = 2;
```

Esse exemplo aumenta o preço unitário de todos os produtos não suspensos do fornecedor 8 em 10 porcento.

```
UPDATE produto
 SET preçounit = preçounit * 1.1
 WHERE fornecedorID = 8 AND suspenso = No;
```

Esse exemplo reduz o preço unitário de todos os produtos não suspensos fornecidos pela Tokyo Traders em 5 porcento. As tabelas "produto" e "fornecedor" têm uma relação um para vários.

```
UPDATE fornecedor INNER JOIN produto
 ON fornecedor.fornecedorID = produto.fornecedorID
 SET preçounit = preçounit * .95
 WHERE nome_empresa = 'Tokyo Traders' AND Suspenso = No;
```

4.1.16 Comando DELETE

Objetivo:

Deletar um ou um grupo de registros em uma tabela do Banco de Dados.

Sintaxe:

```
DELETE FROM  <nome-tabela>
  WHERE <condição>
```

onde:

− nome-tabela - representa o nome da tabela cujos registros serão deletados.

− condição - representa a condição para a deleção dos registros. Esta seleção poderá resultar em um ou vários registros. Neste caso a operação irá ocorrer em todos os registros selecionados.

Exemplo:

```
DELETE  FROM emp
 WHERE empsala > 5000;
```

DELETE é especialmente útil quando você quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados, você pode usar o método EXECUTE com uma instrução DROP.

Entretanto, se você eliminar a tabela, a estrutura é perdida. Por outro lado, quando você usa DELETE, apenas os dados são excluídos. A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos.

Você pode usar DELETE para remover registros de tabelas que estão em uma relação um para vários com outras tabelas. Operações de exclusão em cascata fazem com que os registros das tabelas que estão no lado "vários" da relação sejam excluídos quando os registros correspondentes do lado "um" da relação são excluídos na consulta. Por exemplo, nas relações entre as tabelas "cliente" e "pedido", a tabela "cliente" está do lado "um" e a tabela "pedido" está no lado "vários" da relação. Excluir um registro em "cliente" faz com que os registros correspondentes em "pedido" sejam excluídos se a opção de exclusão em cascata for especificada.

Uma consulta de exclusão exclui registros inteiros e não apenas dados em campos específicos. Se você quiser excluir valores de um campo específico, crie uma consulta atualização que mude os valores para NULL.

Importante: Após remover os registros usando uma consulta exclusão, você não poderá desfazer a operação. Se quiser saber quais arquivos foram excluídos, primeiro examine os resultados de uma consulta seleção que use o mesmo critério, e então execute a consulta exclusão. Mantenha os backups de seus dados. Se você excluir os registros errados, poderá recuperá-los a partir dos seus backups.

Exemplos de instrução DELETE:

Esse exemplo exclui todos os registros de funcionários cujo título seja "Estagiário". Quando a cláusula FROM inclui apenas uma tabela, não é necessário indicar o nome da tabela na instrução DELETE.

```
DELETE * FROM funcionário
 WHERE título = 'Estagiário';
```

Esse exemplo exclui todos os registros de funcionários cujo título seja "Estagiário" e que também tenham um registro na tabela "folha_pagamento". As tabelas "funcionário" e "folha_pagamento" têm uma relação um para um.

```
DELETE funcionário.* FROM funcionário
 INNER JOIN folha_pagamento
   ON funcionário.funcionárioID = folha_pagamento.funcionárioID
```

WHERE funcionário.título = ‘Estagiário’;

4.2 Linguagem de definição de dados (DDL)

É a linguagem utilizada pelo DBA e pelos projetistas do BD para especificar o esquema conceitual. Em alguns SGBDs, a DDL é também utilizada para definir os esquemas interno e externo (as visões). O SGBD possui um compilador DDL que permite a execução das declarações para identificar as descrições dos esquemas e para armazená-las no catálogo do SGBD.

4.2.1 Comandos DDL (Data Definition Language)

É o conjunto de comandos responsáveis pela criação, alteração e deleção da estrutura das tabelas, índices e visões de um sistema.

4.2.2 Comando CREATE

Objetivo:

Criar a estrutura de uma tabela (arquivo) definindo as colunas (campos) e as chaves primárias e estrangeiras existentes, ou a criação do próprio banco de dados.

Sintaxe:

CREATE DATABASE < nome_db >;

onde:

nome_db - indica o nome do Banco de Dados a ser criado.

Sintaxe:

```
CREATE TABLE <nome-tabela>
(<nome-coluna> , <tipo-do-dado>     [NOT NULL]
                                    [NOT NULL WITH DEFAULT] )
PRIMARY KEY    (nome-coluna-chave)
FOREIGN KEY    (nome-coluna-chave-estrangeira) REFERENCES
               (nome-tabela-pai) ON DELETE [RESTRICT]
                                           [CASCADE]
                                           [SET NULL]
```

onde:

- nome-tabela - representa o nome da tabela que será criada.
- nome-coluna - representa o nome da coluna que será criada. A definição das colunas de uma tabela é feita relacionando-as uma após a outra.
- tipo-do-dado - cláusula que define o tipo e tamanho dos campos definidos para a tabela. Os tipos de dados mais comuns serão definidos mais à frente.
- NOT NULL - Exige o preenchimento do campo, ou seja, no momento da inclusão é obrigatório que possua um conteúdo.
- NOT NULL WITH DEFAULT - Preenche o campo com valores pré-definidos, de acordo com o tipo do campo, caso não seja especificado o seu conteúdo no momento da inclusão do registro. Os valores pré-definidos são:
  - Campos numéricos - Valor zero.
  - Campos alfanuméricos - Caracter branco.
  - Campo formato Date - Data corrente.
  - Campo formato Time - Horário no momento da operação.
- PRIMARY KEY (nome-coluna-chave) - Definir para o banco de dados a coluna que será a chave primária da tabela. Caso ela tenha mais de um coluna como chave, elas deverão ser relacionadas entre os parênteses.
- FOREIGN KEY (nome-coluna-chave-estrangeira) REFERENCES (nome-tabela-pai) - Definir para o banco de dados as colunas que são chaves estrangeiras, ou seja, os campos que são chaves primárias de outras tabelas. Na opção REFERENCES deve ser especificado a tabela na qual a coluna é a chave primária.

• ON DELETE - Esta opção especifica os procedimentos que devem ser feitos pelo SGBD quando houver uma exclusão de um registro na tabela pai quando existe um registro correspondente nas tabelas filhas. As opções disponíveis são:

• RESTRICT - Opção default. Esta opção não permite a exclusão na tabela pai de um registro cuja chave primária exista em alguma tabela filha.

• CASCADE - Esta opção realiza a exclusão em todas as tabelas filhas que possua o valor da chave que será excluída na tabela pai.

• SET NULL - Esta opção atribui o valor NULO nas colunas das tabelas filhas que contenha o valor da chave que será excluída na tabela pai.

Tipos de dados mais comuns:

Numéricos:

• Smallint - Armazena valores numéricos, em dois bytes binários, compreendidos entre o intervalo -32768 a +32767.

• Integer - Armazena valores numéricos, em quatro bytes binários, compreendidos entre o intervalo -2147483648 a +2147483647

• Decimal(n,m) - Armazena valores numéricos com no máximo 15 dígitos. Nesta opção deve ser definida a quantidade de dígitos inteiros (n) e casas decimais (m) existentes no campo.

Alfanuméricos:

• Varchar (n) - Definir um campo alfanumérico de até n caracteres, onde n deve ser menor ou igual a 254 caracteres.

• Char (n) - Definir um campo alfanumérico de n caracteres, onde n deve ser menor ou igual a 254 caracteres.

• Long Varchar - Definir um campo alfanuméricos de comprimento maior que 254 caracteres.

• Campo Date - Definir um campo que irá armazenar datas.

• Campo Time - Definir um campo que irá armazenamento de horário.

4.2.3 Comando ALTER TABLE

Objetivo:

Alterar a estrutura de uma tabela (arquivo) acrescentando, alterando e retirando nomes, formatos das colunas e a integridade referencial definidas em uma determinada tabela.

Sintaxe:

```
ALTER TABLE <nome-tabela>
  DROP <nome-coluna>
  ADD <nome-coluna> <tipo-do-dado>  [NOT NULL]
                                    [NOT NULL WITH DEFAULT]
  RENAME <nome-coluna> <novo-nome-coluna>
  RENAME TABLE <novo-nome-tabela>
  MODIFY <nome-coluna> <tipo-do-dado> [NULL]
                                    [NOT NULL]
                                    [NOT NULL WITH DEFAULT]
  ADD PRIMARY KEY <nome-coluna>
  DROP PRIMARY KEY <nome-coluna>
  ADD FOREIGN KEY (nome-coluna-chave-estrangeira) REFERENCES
                  (nome-tabela-pai) ON DELETE [RESTRICT]
                                              [CASCADE]
                                              [SET NULL]
  DROP FOREIGN KEY (nome-coluna-chave-estrangeira) REFERENCES
          (nome-tabela-pai)
```

onde:

- nome-tabela - Representa o nome da tabela que será atualizada.
- nome-coluna - Representa o nome da coluna que será criada.
- tipo-do-dado - Cláusula que define o tipo e tamanho dos campos definidos para a tabela.
- DROP <nome-coluna> - Realiza a retirada da coluna especificada na estrutura da tabela.

• ADD <nome-coluna> <tipo-do-dado> - Realiza a inclusão da coluna especificada na estrutura da tabela. Na coluna correspondente a este campo nos registros já existentes será preenchido o valor NULL (Nulo). As definições NOT NULL e NOT NULL WITH DEFAULT são semelhantes à do comando CREATE TABLE.

• RENAME <nome-coluna> <novo-nome-coluna> - Realiza a troca do nome da coluna especificada.

• RENAME TABLE <novo-nome-tabela> - Realiza a troca do nome da tabela especificada.

• MODIFY <nome-coluna> <tipo-do-dado> - Permite a alteração na característica da coluna especificada.

Opções:

Além das existentes na opção ADD (NOT NULL e NOT NULL WITH DEFAULT), temos a opção NULL que altera a característica do campo passando a permitir o preenchimento com o valor Nulo.

• ADD PRIMARY KEY <nome-coluna> - Esta opção é utilizada quando é acrescido um novo campo como chave primária da tabela.

• DROP PRIMARY KEY <nome-coluna> - Esta opção é utilizada quando é retirado um campo como chave primária da tabela.

• ADD FOREIGN KEY <nome-coluna> - Esta opção é utilizada quando é acrescido um novo campo sendo ele uma chave estrangeira.

• DROP FOREIGN KEY <nome-coluna> - Esta opção é utilizada quando é retirado uma chave estrangeira da estrutura da tabela.

4.2.4 Comando DROP TABLE

Objetivo:

Deletar a estrutura e os dados existentes em uma tabela. Após a execução deste comando estarão deletados todos dados, estrutura e índices de acessos que estejam a ela associados.

Sintaxe:

DROP TABLE <nome-tabela>

onde:

• nome-tabela - representa o nome da tabela que será deletada.

4.2.5 Comando CREATE INDEX

Objetivo:

Criar uma estrutura de índice de acesso para uma determinada coluna em uma tabela. Um índice de acesso permite um acesso mais rápido aos dados em uma operação de seleção. Os índices podem ser criados a partir de um ou mais campos de uma tabela.

Sintaxe:

CREATE [UNIQUE] INDEX <nome-índice>

ON <nome-tabela> (<nome-coluna> [ASC], [<nome-coluna> [ASC]])
[DESC] [DESC]

onde:

− nome-índice - representa o nome da estrutura de índice que será criada.

− nome-tabela - representa o nome da tabela que contem a coluna na qual será criada o índice de acesso.

− nome-coluna - representa o nome da coluna para a qual será criado o índice.

− Opção ASC/DESC - representa a criação do índice ordenado em ordem crescente (ASC) ou decrescente (DESC).

4.2.6 Comando DROP INDEX

Objetivo:

Deletar uma estrutura de índice de acesso para uma determinada coluna em uma tabela.

Sintaxe:

DROP INDEX <nome-índice>

onde:

- nome-índice - Representa o nome da estrutura de índice que será deletada.

4.2.7 Uso de Índices

Quando fazemos consultas em uma tabela, estamos selecionando registros com determinadas propriedades. Dentro do conceito de Álgebra Relacional, estamos fazendo uma simples operação de determinar um subconjunto de um conjunto. A forma trivial de realizar esta operação é avaliar cada um dos elementos do conjunto para determinar se ele possui ou não as propriedades desejadas. Ou seja, avaliar, um a um, todos os seus registros.

Em tabelas grandes, a operação descrita acima pode ser muito custosa. Imaginemos que se deseje relacionar todas as apólices vendidas para um determinado cliente, para saber seu histórico. Se for necessário varrer toda a tabela de apólices para responder esta questão o processo certamente levará muito tempo.

A forma de resolver este problema é o uso de índices. Índices são criados para localizar registros mais rápida e eficientemente, pois eles possibilitam ao banco de dados o acesso direto às informações desejadas. É possível criar um índice sobre uma ou mais colunas de uma tabela e a cada índice é dado um nome.

Importante: Atualizar uma tabela que contenha um ou mais índices leva mais tempo do que atualizar uma tabela que não os tenha. Isto é porque os índices também precisam ser atualizados. Então, é uma boa idéia criar índices sobre dados de tabelas que sejam freqüentemente usadas para uma pesquisa.

Fisicamente, a tabela não está organizada em nenhuma ordem. Os registros são colocados na tabela na ordem cronológica de inserção. Deleções ainda causam mudanças nesta ordem.

Um registro no índice é composto pelo conjunto de valores dos campos que compõem o índice e pelo endereço fisico do registro na tabela original. Ao escrever uma consulta SQL, não informamos especificamente qual índice será usado pela consulta. Esta decisão é tomada pelo banco de dados. Cabe a nós apenas escrever a consulta de forma que o uso do índice seja possível. É preciso que nossa consulta disponibilize o índice.

Para resolver uma consulta, o banco de dados fará o acesso ao índice, onde irá recuperar o endereço fisico, na tabela, dos registros candidatos a compor o resultado. Com este endereço, ele verifica cada registro quanto às outras condições. Os que satisfizerem as outras condições comporão o resultado.

4.2.8 Campos nulos não entram

Toda vez que um registro possuir valores nulos para todos os campos que compõem um índice, este registro não será colocado no índice. Isto causa problemas de performance em sistemas mau projetados.

Suponha que a modelagem de dados de um sistema de contas a pagar tenha resultado em um projeto onde existe uma tabela de contas a pagar. Esta tabela contém, entre outros, dois campos: data de vencimento e data de pagamento. A primeira é a data em que o título deve ser pago. A segunda é a data em que o título foi efetivamente pago.

Suponha ainda que a dinâmica do sistema determine que todo título, ao ser inserido no sistema, tenha valor nulo para o campo data de vencimento. Quando o pagamento vier a ser efetuado, o campo será atualizado. É bastante possível que seja construída uma consulta para recuperar todos os títulos não pagos.

Neste caso, não será possível o uso de índices, pois estamos procurando campos com valor nulo. Se tivermos, nesta tabela, 200.000 títulos, dos quais 500 são não pagos, esta consulta terá desempenho bastante aquém do possível. Uma solução melhor, seria inicializar o campo data de vencimento com o valor 01/01/2000, significando conta não paga.

Como última consideração sobre consultas em uma tabela, vale lembrar que quando fazemos uma consulta a uma tabela bastante pequena, não compensa usar índices. O trabalho envolvido em acessar o índice, pegar o endereço e, somente depois, acessar a tabela, é maior que o esforço de ler a tabela inteira.

4.2.9 Visões

Em SQL, uma visão é uma tabela virtual baseada em um grupo de resultados de uma declaração SELECT.

Uma visão contém linhas e colunas, exatamente como uma tabela real. Os campos em uma visão são campos de uma ou mais tabelas reais de um banco de dados. Você pode adicionar funções SQL e declarações WHERE e JOIN em uma visão e apresentar o dado como se este tivesse vindo de uma única tabela.

Importante: O projeto e a estrutura de um banco de dados não são afetados por funções ou por declarações WHERE ou JOIN em uma visão.

Sintaxe:

```
CREATE VIEW nome_visao AS
 SELECT nome_coluna(s)
 FROM nome_tabela
 WHERE condicao
```

Nota: O banco de dados não armazena os dados de uma visão! A base de dados recria os dados utilizando a declaração SELECT dentro da visão, toda vez que o usuário pesquisar essa visão.

Uma visão pode ser usada dentro de uma query ou dentro de uma outra visão. Adicionando funções, JOINS, etc. a uma visão, ela permite a você apresentar exatamente o dado que você deseja.

Imaginemos, dentro de um banco de dados, uma visão chamada “Lista de produtos” que mostra todos os produtos ativos (produtos que não são interrompidos) de uma tabela de produtos. A visão é criada da seguinte maneira:

```
CREATE VIEW lista_produtos AS
 SELECT ID_produto, nome_produto
 FROM produto
 WHERE interrompido=No;
```

Podemos criar a query para a visão acima da seguinte maneira:

```
SELECT * FROM lista_produtos
```

Outra visão do mesmo banco de dados seleciona todos os produtos da tabela “produto” que tem um preço unitário que é maior que a média dos preços unitários:

```
CREATE VIEW  produtos_media_ preco_acima AS
 SELECT  nome_produto, preco_unit
 FROM  produto
 WHERE preco_unit > (SELECT AVG (preco_unit)
                     FROM produto)
```

Podemos criar a query para a visão acima da seguinte maneira:

```
SELECT * FROM lista_produtos
```

5 MYSQL

5.1 Introdução ao programa

O programa MySQL é um servidor robusto de bancos de dados SQL (Structured Query Language - Linguagem Estruturada para Pesquisas) muito rápido, multi-tarefa e multiusuário.

MySQL é uma marca registrada da MySQL AB. O programa MySQL é de Licença Dupla. Os usuários podem escolher entre usar o programa MySQL como um produto Open Source/Free Software, sob os termos da GNU General Public License ou podem comprar uma licença comercial padrão da MySQL AB.

O site web do MySQL (http://www.mysql.com/) dispõe das últimas informações sobre o programa MySQL, assim como de qualquer informação que se precise sobre o produto e o software.

O MySQL, um sistema de gerenciamento de banco de dados. Um banco de dados relacional armazena dados em tabelas separadas em vez de colocar todos os dados em um só local. Isso proporciona velocidade e flexibilidade.

Como uma das características técnicas do MySQL, podemos dizer que o Programa de Banco de Dados MySQL é um sistema cliente/servidor que consiste de um servidor SQL multi-tarefa que suporta acessos diferentes, diversos programas clientes e bibliotecas, ferramentas administrativas e diversas interfaces de programação (API's).

O Servidor MySQL também é concedido como uma biblioteca multi-tarefa que se pode ligar a alguma aplicação para chegar a um produto mais rápido, menor e mais fácilmente gerenciável.

O programa "libmysqld" permite que o MySQL Server seja adequado para uma grande área de aplicações, devido ao Servidor Embutido MySQL. Usando-se a biblioteca do servidor MySQL embutido, o MySQL Server pode ser colocado em vários aplicativos e dispositivos eletrônicos, onde o usuário final não tem conhecimento de possuir um banco de dados básico.

O servidor MySQL embutido serve para uso em aplicações de Internet, quiosques públicos, e responsável por unidades de combinaçãao hardware/ software, e assim por diante.

5.2 Conectando e desconectando do Servidor

Para conectar ao servidor, normalmente vai se precisar fornecer um nome de usuário quando o mysql for chamado e, na maioria dos casos, uma senha. Se o servidor executa em uma máquina diferente de onde se está, também vai se precisar especificar um nome de máquina.

Uma vez que se saiba quais os parâmetros corretos, já se pode conectar da seguinte forma:

```
shell> mysql -h servidor -u usuario -p
Enter password: ********
```

Os asteriscos (********) representam a senha. Se isto funcionar, vão aparecer algumas informações iniciais, seguidas de um prompt mysql>.

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 4.0.14-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

O prompt diz que o mysql está pronto para que se comece a digitar os comandos.

Também pode-se desconectar a qualquer momento, digitando QUIT (ou \q) no prompt mysql>:

```
mysql> QUIT
Bye
```

No Unix, também pode-se desconectar pressionando Control-D.

A maioria dos exemplos nas seções seguintes assumem que já se esteja conectado ao servidor. Isto é indicado pelo prompt mysql>.

5.3 Fazendo Consultas

Tenha certeza que você está conectado ao servidor, como discutido na seção anterior. Neste momento, vamos saber um pouco sobre como fazer consultas. Esta seção descreve os princípios básicos da entrada de comandos. Utilizando diversas consultas pode-se tentar familiarizar com o funcionamento do MySQL.

Aqui está um comando simples que solicita ao servidor seu número de versão e a data atual. Digite-o como visto abaixo seguindo o prompt mysql> e digite a tecla RETURN:

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

```
+--------------+--------------+
| version() | CURRENT_DATE |
+--------------+--------------+
| 3.22.20a-log | 1999-03-19 |
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

Esta consulta ilustra várias coisas sobre o MySQL:

- Um comando normalmente consiste de uma instrução SQL seguida por um ponto e vírgula. (Existem algumas exceções onde um ponto e vírgula podem ser omitidos. QUIT, mencionado anteriormente, é um deles.)
- Quando você emite um comando, o MySQL o envia para o servidor para execução e mostra os resultados. Depois imprime outro prompt mysql> para indicar que está pronto para outro comando.
- O MySQL mostra a saída da consulta em forma tabular (linhas e colunas). A primeira linha contém rótulos para as colunas. As linhas seguintes são o resultado da consulta. Normalmente, rótulos de colunas são os nomes das colunas que você busca das tabelas do banco de dados. Se você está recuperando o valor de uma expressão no lugar de uma coluna de tabela, o MySQL rotula a coluna usando a própria expressão.
- O MySQL mostra quantas linhas foram retornadas e quanto tempo a consulta levou para executar, o que lhe dá uma vaga idéia da performance do servidor. Estes valores são imprecisos porque eles representam tempo de relógio (Não tempo de CPU ou de máquina), e porque eles são afetados pelos fatores como a carga do servidor e latência de rede. (Para resumir, a linha “rows in set” não é mostrada nos exemplos seguintes.)

5.4 Criação e Utilização de um Banco de Dados

Agora que você já sabe como entrar com os comandos, é hora de acessar um banco de dados. Utilize a instrução SHOW para saber quais bancos de dados existem atualmente no servidor:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql |
| test |
| tmp |
+----------+
```

Agora, tente acessar o banco de dados test:

```
mysql> USE test
Database changed
```

Perceba que o USE, como o QUIT, não necessitam de um ponto e vírgula. (Você pode terminar tais declarações com um ponto e vírgula se gostar; isto não importa) A instrução USE é especial em outra maneira, também: Ela deve ser usada em uma única linha.

Para criar seu banco de dados digite o seguinte:

```
mysql> CREATE DATABASE nome_bd;
```

No Unix, nomes de bancos de dados são caso sensitivo (ao contrário das palavras chave SQL), portanto você deve sempre fazer referência ao seu banco de dados como nome_bd e não Nome_BD, NOME_BD ou outra variação qualquer. Isto também é verdade para nomes de tabelas. (No Windows, esta restrição não se aplica, entretanto, você deve referenciar os bancos de dados e tabelas usando o mesmo caso em toda a parte da consulta.)

Criar um banco de dados não o seleciona para o uso; você deve fazer isso de forma explícita.

Para fazer do banco de dados que você quer mexer ser o atual, use o comando:

```
mysql> USE nome_bd
Database changed
```

Seu banco de dados necessita ser criado somente uma única vez, mas você deve selecioná-lo para o uso cada vez que você iniciar uma seção MySQL.

Criar o banco de dados é a parte fácil, mas neste ponto ele está vazio, como o SHOW TABLES mostrará:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

A parte mais difícil é decidir qual a estrutura que seu banco de dados deve ter: quais tabelas você precisará e que colunas estarão em cada uma delas.

Você irá precisar de uma tabela para guardar um registro para cada um de seus dados.

Utilize a senteça CREATE TABLE para especificar o layout de sua tabela:

```
mysql> CREATE TABLE     nome_tabela (nome_campo1 TIPO_DADO(),
                        nome_campo2 TIPO_DADO(), etc);
```

Agora que você criou uma tabela, a instrução SHOW TABLES deve produzir alguma saída:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in nome_bd |
+---------------------+
| nome_tabela |
+---------------------+
```

Para verificar se sua tabela foi criada da forma que você esperava, utilize a instrução DESCRIBE:

```
mysql> DESCRIBE nome_tabela;
+---------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name | varchar(20) | YES | | NULL | |
| owner | varchar(20) | YES | | NULL | |
| species | varchar(20) | YES | | NULL | |
| sex | char(1) | YES | | NULL | |
| birth | date | YES | | NULL | |
| death | date | YES | | NULL | |
+---------+-------------+------+-----+---------+-------+
```

Você pode usar DESCRIBE a qualquer momento, por exemplo, se você esquecer os nomes das colunas na sua tabela ou de que tipos elas são.

Depois de criar sua tabela, você precisará preenchê-la. As instruções LOAD DATA e INSERT são úteis para isto.

Suponha que seu registro possa ser descrito como é abaixo: (Observe que o MySQL espera datas no formato AAAA-MM-DD; isto pode ser diferente do que você está acostumado.)

name owner species sex birth death
Fluffy Harold gato f 1993-02-04
Claws Gwen gato m 1994-03-17
Buffy Harold cachorro f 1989-05-13
Fang Benny cachorro m 1990-08-27
Bowser Diane cachorro m 1979-08-31 1995-07-29
Chirpy Gwen pássaro f 1998-09-11
Whistler Gwen pássaro 1997-12-09
Slim Benny cobra m 1996-04-29

Quando você desejar adicionar novos registros um a um, a instrução INSERT é usada. Na sua forma mais simples, você fornece valores para cada coluna, na ordem em que as colunas foram listadas na instrução CREATE TABLE.

Um registro pode ser adicionado utilizando uma instrução INSERT desta forma:

```
mysql> INSERT INTO nome_tabela
    VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

Perceba que os valores de string e datas são especificados aqui como strings com aspas. Com o INSERT você também pode inserir NULL diretamente para representar um valor em falta. Não pode ser usado \N como você fez com LOAD DATA.

A instrução SELECT é usada para recuperar informações de uma tabela. A forma geral da instrução é:

```
SELECT o_que_mostrar
  FROM nome_tabela
  WHERE condicoes_para_satisfazer;
```

A forma mais simples do SELECT recuperar tudo de uma tabela é:

```
mysql> SELECT * FROM nome_tabela;
+----------+--------+---------+------+------------+------------+
| name | owner | species | sex | birth | death |
+----------+--------+---------+------+------------+------------+
| Fluffy | Harold | cat | f | 1993-02-04 | NULL |
| Claws | Gwen | cat | m | 1994-03-17 | NULL |
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
| Fang | Benny | dog | m | 1990-08-27 | NULL |
| Bowser | Diane | dog | m | 1979-08-31 | 1995-07-29 |
| Chirpy | Gwen | bird | f | 1998-09-11 | NULL |
| Whistler | Gwen | bird | NULL | 1997-12-09 | NULL |
| Slim | Benny | snake | m | 1996-04-29 | NULL |
| Puffball | Diane | hamster | f | 1999-03-30 | NULL |
+----------+--------+---------+------+------------+------------+
```

Esta forma do SELECT é útil se você deseja ver sua tabela inteira como agora, depois de você acabar de carregá-la com os dados iniciais.

A forma de se corrigir algo que foi digitado de forma errada é através de uma instrução UPDATE:

```
mysql> UPDATE nome_tabela
    SET birth = "1989-08-31"
    WHERE name = "Bowser";
```

O UPDATE altera apenas o registro em questão e não exige que você recarregue a tabela.

Bem, vimos neste capítulo os comandos básicos para se criar e manipular um banco de dados. A tarefa, a partir de agora, será testar, dentro do MySQL, todos os comandos SQL vistos no capítulo anterior, não esquecendo que a gama de possibilidades, tanto da linguagem SQL como do MySQL, é muito maior do que o que foi visto aqui. Bom trabalho!

6 BIBLIOGRAFIA

Database Management Systems, Ramakrishnan, R.; McGraw-Hill, 1998.

Fundamentals of Database Systems; Ramez Elmasri, Shamkant Navathe; The Benjamin CummingsPublishing Company; 1989;

MySQL. **Manual de referência do MySQL.** Disponível por WWW em http://dev. mysql.com/doc/mysql/pt/index.html (2005).

MySQL. **MySQL: The world's most popular open source database.** Disponível por WWW em http://www.mysql.com/ (2005).

MySQL Brasil. **Soluções em gestão comercial baseadas em software livre.** Disponível por WWW em http://www.eacnet.com.br/index.pl/mysqlbrasil (2005).

Projeto de Banco de Dados, Heuser, C. A.; Sagra Luzatto, 1998.

Sistema de Banco de Dados; Henry F. Korth, Abraham Silberschatz; Makro Books; 1995;

Sistemas de Bancos de Dados, Korth, H.F. e Silberschatz, A.; Makron Books, 2ª edição revisada, 1994.

SQLMagazine.com. The top SQL SERVER resources on the net. Disponível por WWW em http://www.sqlmagazine.com/ (2005).

W3schools. Online Web Tutorials. Disponível por WWW em http://www. w3schools.com/ (2005).

Wikipedia. A enciclopédia livre. Disponível por WWW em http://pt.wikipedia. org/wiki/ (2005).