Colégio Salesiano de Lins - “Dom Henrique Mourão”

Habilitação Profissional de Técnico em Desenvolvimento de Sistemas para Web

Disciplina: Linguagem de Programação III

Parte I

Introdução a Banco de Dados e Structured Query Language (SQL)

Prof.: **Anderson Pazin**

SUMÁRIO

1. Conceitos de Banco de dados

1.1. Introdução

Hoje em dia o termo banco de dados é bastante popular em diversas áreas de atuação. Com o aumento da utilização de computadores na manipulação de dados que envolvem diversas aplicações, os bancos de dados estão sendo desenvolvidos e aplicados nas diferentes áreas que envolvem o comércio, a indústria e a pesquisa acadêmica entre outras.

Por exemplo, uma conta bancária faz parte de uma coleção imensa de contas bancárias de nosso banco, o Título Eleitoral ou o Cadastro de Pessoa Física (CPF), certamente estão armazenados em Bancos de Dados de grande porte. Quando um dinheiro é sacado no Caixa Eletrônico do banco, o saldo e as movimentações existentes em nessa conta bancária já estão à disposição do cliente.

Mas o que vem a ser um Banco de Dados? Um ***Banco de Dados*** (ou *Base de Dado*s) é uma coleção de dados relacionados, organizados e armazenados visando facilitar a manipulação desses dados, permitindo realizar alterações, inserções, remoções e consultas. Os tipos de "coleções de dados" são ilimitados, ou seja, quaisquer aplicações do mundo real que possam ser representadas através de dados computáveis, podem ser armazenadas em um banco de dados. Exemplos de coleções são: dados de um banco financeiro, dados de controle de uma universidade, dados de controle de estoque de empresas, dados sobre os genes humanos (projeto Genoma), dados sobre metereologia, etc.

A manipulação desses dados armazenados é feita por um conjunto de programas computadorizados denominado ***Sistema Gerenciador de Bancos de Dados (SGBDs).*** Um SGBD tem uma gama de funções pré-implementadas que gerenciam as operações de inserção, remoção, atualização e consulta dos dados armazenados.

Os *SGBDs* e os *Bancos de Dados* juntos formam um ambiente denominado ***Sistema de Banco de Dados (SBD).*** Pode-se definir esse sistema como um ambiente cujo objetivo global é registrar e manter informação. Um SBD busca oferecer:

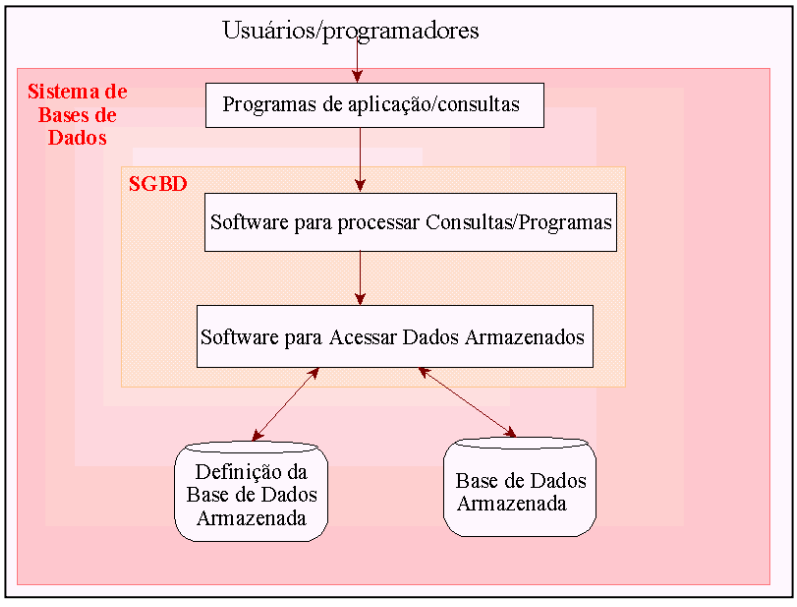
- ✓ **Rapidez:** consultas on-line para informação;
- ✓ **Disponibilidade total:** toda a informação contida no interior da base está disponível o tempo todo;
- ✓ **Flexibilidade:** questões não tratadas tornam-se tratáveis, ou seja, mudanças são relativamente fáceis de se implementar.
- ✓ **Integridade:** a duplicação de dados é reduzida, e políticas de atualização podem ser padronizadas, resultando em consistência de dados.
- ✓ Como um todo, fazem parte de um SBD:
- ✓ **Dados:** valores fisicamente registrados no banco de dados;
- ✓ **Hardware:** memória secundária, unidades de controle, canais de comunicação, etc.
- ✓ **Software:** SGBD.
- ✓ **Usuários:** todos os usuários que estão envolvidos na definição e utilização de um banco de dados. Esses usuários podem ser divididos em três classes:
    - *programadores de aplicações:* responsáveis pela escrita de programas de aplicação que utilizem o banco de dados;
    - *usuários finais:* utilizam uma linguagem de consulta fornecida como parte integrante do sistema, ou podem chamar uma aplicação escrita pelo programador sob a forma de um programa (efetua operações de recuperação, criação, eliminação ou modificação);
    - *DBA:* administrador do banco de dados, ou seja, o responsável pelo controle do "bom funcionamento" do banco de dados.

1.2. Componentes de um Banco de Dados

Um Banco de Dados é composto pelas seguintes partes:

- **Gerenciador de Acesso ao Disco:** O SGBD utiliza o Sistema Operacional para acessar os dados armazenados em disco, controlando o acesso concorrente às tabelas do Banco de Dados. O Gerenciador controla todas as pesquisas (*queries*) solicitadas pelos usuários no modo interativo, os acessos do compilador DML, os acessos feitos pelo Processador do Banco de Dados ao Dicionário de Dados e também aos próprios dados.
- O **Compilador DDL** (Data Definition Language) processa as definições do esquema do Banco de Dados, acessando quando necessário o Dicionário de Dados do Banco de Dados.
- O **Dicionário de Dados** contém o esquema do Banco de Dados, suas tabelas, índices, forma de acesso e relacionamentos existentes.
- O **Processador do Banco de Dados** manipula requisições à própria Base de Dados em tempo de execução. É o responsável pelas atualizações e integridade da Base de Dados.
- O **Processador de Pesquisas** (*queries*) dos usuários analisa as solicitações, e se estas forem consistentes, aciona o Processador do Banco de Dados para acesso efetivo aos dados.

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 Banco de Dados.



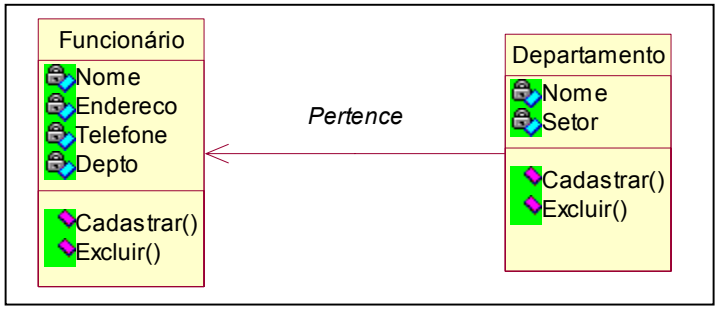
1.3. Tipos de Banco de Dados

Atualmente existem diversos Banco de Dados no mercado, tais como Oracle, SQL Server, MySQL, Sybase, Jasmine, ZIM, DB2, PostGree etc. Cada um tem suas características próprias o que os tornam vantajosos ou não para determinados tipos de aplicação. Mas esses bancos de dados são classificados segundo a forma que os dados podem ser armazenados e posteriormente acessados. Dentre os principais tipos de classificações banco de dados temos:

Banco de dados Orientado a Objetos.

Representam os dados como coleções que obedecem propriedades. São modelos geralmente conceituais dispondo de pouquíssimas aplicações reais. Cada objeto tem características próprias (atributos) com ações próprias (métodos)

Neste Modelo, por exemplo, não seria interessante a existência de uma tabela de funcionários e dentro dela alguma referência para cada registro, de forma a podermos saber onde (em que departamento) o funcionário está trabalhando. Um conjunto de regras disponibilizaria em separado os funcionários da fábrica que, no entanto estariam agrupados aos demais, para o sistema de folha de pagamento.



Banco de Dados Relacional

O Modelo de Dados relacional representa os dados contidos em um Banco de Dados através de relações. Estas relações contêm informações sobre as entidades representadas e seus relacionamentos. O Modelo Relacional é claramente baseado no conceito de matrizes, onde as chamadas linhas (das matrizes) seriam os registros e as colunas (das matrizes) seriam os campos. 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.

Cada linha de nossa relação será chamada de *TUPLA* e cada coluna de nossa relação será chamada de *ATRIBUTO*. O conjunto de valores que um determinado atributo pode assumir é intitulado de *DOMÍNIO*.

O domínio consiste de um grupo de valores a partir dos quais um ou mais atributos retiram seus valores reais. Assim sendo 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).

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.

Chamaremos de Chave Primária ao Atributo que definir um registro, dentre uma coleção de registros. Chamaremos de Chave Composta, aquela chave que contém mais de um atributo (Por exemplo um cadastro ordenado alfabeticamente por Estado, Cidade e Nome do Cliente, necessitaria de uma chave composta que contivesse estes três atributos). Chamaremos de Chave Estrangeira, aquela chave que permitir a ligação lógica entre uma tabela (onde ela se encontra) com outra na qual ele é chave primária.

Exemplo:

Cidade	Estado
@ CidCodi	@ EstCodi
CidNome	EstNome
# EstCodi	

CidCodi e EstCodi, são chaves primárias respectivamente das tabelas Cidade e Estado, enquanto EstCodi é chave estrangeira na tabela de cidades. É precisamente por este campo (atributo, ou coluna), que será estabelecida a relação entre as tabelas Cidade-->Estados.

Resumindo

Resumidamente podemos dizer que um Banco de dados relacional:

- ✓ Contém uma quantidade qualquer de tabelas;
- ✓ Os dados existentes em uma tabela nunca deveriam aparecer em outra;
- ✓ Um Banco de Dados projetado corretamente contém todos os vínculos necessários para permitir que registros sejam relacionados entre tabelas diferentes;
- ✓ Tabelas podem ser relacionadas;
- ✓ A integridade referencial deve ser mantida (Integridade Referencial é a existência de um valor ou atributo relacionado em um banco de dados dependendo de um outro valor ou atributo.);
- ✓ Em geral as entidades do DED serão as tabelas de um banco de dados relacional;
- ✓ Chave Primária (@) é um campo ou conjunto de campos que identifica um registro ou uma tupla;
- ✓ Chave Estrangeira (#) é o campo que serve para relacionar as tabelas.

2. NORMALIZAÇÃO DE DADOS

Consiste em definir o formato lógico adequado para as estruturas de dados identificados no projeto lógico do sistema, com o objetivo de minimizar o espaço utilizado pelos dados e garantir a integridade e confiabilidade das informações.

A normalização é feita, através da análise dos dados que compõem as estruturas utilizando o conceito chamado "Formas Normais (FN)". As FN são conjuntos de restrições nos quais os dados devem satisfazê-las. Por exemplo, pode-se dizer que a estrutura está na primeira forma normal (1FN), se os dados que a compõem satisfizerem as restrições definidas para esta etapa.

A normalização completa dos dados é feita, seguindo as restrições das três formas normais existentes, sendo que a passagem de uma FN para outra é feita tendo como base o resultado obtido na etapa anterior, ou seja, na FN anterior.

Para realizar a normalização dos dados, é primordial que seja definido um campo chave para a estrutura, campo este que permitirá identificar os demais campos da estrutura. Formas Normais existentes:

2.1. Primeira Forma Normal (1FN)

Consiste em retirar da estrutura o elemento repetitivo, ou seja, aqueles dados que podem compor uma estrutura de vetor. Podemos afirma que uma estrutura está normalizada na 1FN, se não possuir elementos repetitivos. Exemplo:

Arquivo de Notas Fiscais:
(Num. NF, Série, Data emissão, Cod. do Cliente, Nome do cliente, Endereço do cliente, CGC do cliente, Relação das mercadorias vendidas (onde para cada mercadoria temos: Código da Mercadoria, Descrição da Mercadoria, Quantidade vendida, Preço de venda e Total da venda desta mercadoria) e Total Geral da Nota)

Analisando a estrutura acima, observamos que existem várias mercadorias em uma única Nota Fiscal sendo, portanto elementos repetitivos que deverão ser retirados.

Estrutura na primeira forma normal (1FN):

Arquivo de Notas Fiscais:
(Num. NF, Série, Data emissão, Código do Cliente, Nome do cliente, Endereço do cliente, CGC do cliente e Total Geral da Nota)
Arquivo de Vendas:
(Num. NF, Código da Mercadoria, Descrição da Mercadoria, Quantidade vendida, Preço de venda e Total da venda desta mercadoria)

Obs. Os campos sublinhados identificam as chaves das estruturas.

Como resultado desta etapa ocorre um desdobramento dos dados em duas estruturas, a saber:

- ✓ Primeira estrutura (Arquivo de Notas Fiscais): Dados que compõem a estrutura original, excluindo os elementos repetitivos.
- ✓ Segundo estrutura (Arquivo de Vendas): Dados que compõem os elementos repetitivos da estrutura original, tendo como chave o campo chave da estrutura original (Num. NF) e o campo chave da estrutura de repetição (Código da Mercadoria).

2.2. Segunda Forma Normal (2FN)

Consiste em retirar das estruturas que possuem chaves compostas (campo chave sendo formado por mais de um campo), o elemento que são funcionalmente dependentes de parte da chave. Podemos

afirmar que uma estrutura está na 2FN, se ela estiver na 1FN e não possuir campos que são funcionalmente dependentes de parte da chave. Exemplo:

Estrutura na primeira forma normal (1FN):
Arquivo de Notas Fiscais:
(Num. NF, Série, Data emissão, Código do Cliente, Nome do cliente, Endereço do cliente, CGC do cliente e Total Geral da Nota)
Arquivo de Vendas:
(Num. NF, Código da Mercadoria, Descrição da Mercadoria, Quantidade vendida, Preço de venda e Total da venda desta mercadoria)

Estrutura na segunda forma normal (2FN):
Arquivo de Notas Fiscais:
(Num. NF, Série, Data emissão, Código do Cliente, Nome do cliente, Endereço do cliente, CGC do cliente e Total Geral da Nota)
Arquivo de Vendas:
(Num. NF, Código da Mercadoria, Quantidade vendida e Total da venda desta mercadoria)
Arquivo de Mercadorias:
(Código da Mercadoria, Descrição da Mercadoria, Preço de venda).

Como resultado desta etapa, houve um desdobramento do arquivo de Vendas (o arquivo de Notas Fiscais, não foi alterado, por não possuir chave composta) em duas estruturas a saber:

- ✓ Primeira estrutura (Arquivo de Vendas): Contém os elementos originais, sendo excluídos os dados que são dependentes apenas do campo Código da Mercadoria.
- ✓ Segundo estrutura (Arquivo de Mercadorias): Contém os elementos que são identificados apenas pelo Código da Mercadoria, ou seja, independentemente da Nota Fiscal, a descrição e o preço de venda serão constantes.

2.3. Terceira Forma Normal (3FN)

Consiste em retirar das estruturas os campos que são funcionalmente dependentes de outros campos que não são chaves. Podemos afirmar que uma estrutura está na 3FN, se ela estiver na 2FN e não possuir campos dependentes de outros campos não chaves. Exemplo:

Estrutura na segunda forma normal (2FN):
Arquivo de Notas Fiscais:
(Num. NF, Série, Data emissão, Código do Cliente, Nome do cliente, Endereço do cliente, CGC do cliente e Total Geral da Nota)
Arquivo de Vendas
(Num. NF, Código da Mercadoria, Quantidade vendida e Total da venda desta mercadoria)
Arquivo de Mercadorias:
(Código da Mercadoria, Descrição da Mercadoria, Preço de venda)

Estrutura na terceira forma normal (3FN):
Arquivo de Notas Fiscais:
(Num. NF, Série, Data emissão, Código do Cliente e Total Geral da Nota)
Arquivo de Vendas:
(Num. NF, Código da Mercadoria, Quantidade vendida e Total da venda desta mercadoria)
Arquivo de Mercadorias:
(Código da Mercadoria, Descrição da Mercadoria, Preço de venda)
Arquivo de Clientes:

(Código do Cliente, Nome do cliente, Endereço do cliente e CGC do cliente)

Como resultado desta etapa, houve um desdobramento do arquivo de Notas Fiscais, por ser o único que possuía campos que não eram dependentes da chave principal (Num. NF), uma vez que independente da Nota Fiscal, o Nome, Endereço e CGC do cliente são inalterados. Este procedimento permite evitar inconsistência nos dados dos arquivos e economizar espaço por eliminar o armazenamento freqüente e repetidas vezes destes dados. A cada nota fiscal comprada pelo cliente, haverá o armazenamento destes dados e poderá ocorrer divergência entre eles. As estruturas alteradas foram pelos motivos, a saber:

- ✓ Primeira estrutura (Arquivo de Notas Fiscais): Contém os elementos originais, sendo excluídos os dados que são dependentes apenas do campo Código do Cliente (informações referentes ao cliente).
- ✓ Segundo estrutura (Arquivo de Clientes): Contém os elementos que são identificados apenas pelo Código do Cliente, ou seja, independente da Nota Fiscal, o Nome, Endereço e CGC dos clientes serão constantes.

Após a normalização, as estruturas dos dados estão projetadas para eliminar as inconsistências e redundâncias dos dados, eliminando desta forma qualquer problema de atualização e operacionalização do sistema. A versão final dos dados poderá sofrer alguma alteração, para atender as necessidades específicas do sistema, a critério do analista de desenvolvimento durante o projeto físico do sistema.

3. LINGUAGEM SQL

A linguagem SQL (Structured Query Language) representa um conjunto de comandos responsáveis pela definição das tabelas, comandos e atualização dos dados em um S.G.B.D. Os comandos existentes nesta linguagem são subdivididos em dois grupos:

Comandos DDL (Data Definition Language) - Conjunto de comandos responsáveis pela criação, alteração e remoção da estrutura das tabelas e índices de um sistema.

Comandos DML (Data Manipulation Language) - Conjunto de comandos responsáveis pela consulta e atualização dos dados armazenados em um banco de dados.

3.1. Comandos DDL

Comando: `Create Table`

O comando CREATE TABLE permite criar e definir a estrutura de uma tabela(arquivo) definido as colunas (campos) e as chaves primárias e estrangeiras existentes. Sua sintaxe é:

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

Obs: os campos entre colchete [] são opcionais.

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 - Define o tipo e tamanho dos campos definidos para a tabela. Os tipos de dados mais comuns são:

1) Numéricos: (NUMBER)

- ✓ *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.

2) 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.

3) *Date* - Definir um campo que irá armazenar datas.

4) *Time* - Definir um campo que irá armazenamento de horário.

`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 pre-definidos, de acordo com o tipo do campo, caso não seja especificado o seu conteúdo no momento da inclusão do registro. Os valores pre-definidos são:

e.1) Campos numéricos - Valor zero.
e.2) Campos alfanuméricos - Caracter branco.
e.3) Campo formato Date - Data corrente.
e.4) Campo formato Time - Horário no momento da operação.

`CONSTRAINT nome PRIMARY KEY (`*`nome-coluna-chave`*`)` - Defini 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.

`CONSTRAINT nome FOREIGN KEY (`*`nome-coluna-chave-estrangeira`*`)` REFERENCES *nome-tabela-pai (nome-campo-pai)* - Defini 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:

h.1) `RESTRICT` - Opção *default*. Esta opção não permite a exclusão na tabela pai de um registro cuja chave primária exista em alguma tabela filha.

h.2) `CASCADE` - Esta opção realiza a exclusão em todas as tabelas filhas que possua o valor da chave que será excluída na tabela pai.

h.3) `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.

Observe alguns exemplos do comando CREATE TABLE.

Exemplo 1: Criando a tabela de professores

```
CREATE TABLE Professores (
      codigo           number(5),
      nome             varchar(30),
      rg               varchar(10),
      telefone         char(10),
      endereco         varchar(40),
      data_nascimento  date,
      formacao         varchar(40),
CONSTRAINT PK_PROF PRIMARY KEY (codigo));
```

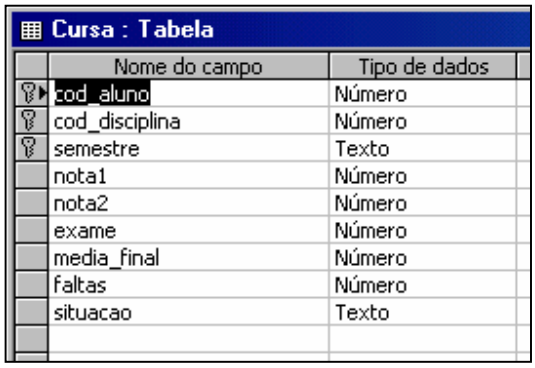
Professores : Tabela

Nome do campo	Tipo de dados
codigo	Número
nome	Texto
rg	Texto
endereco	Texto
telefone	Número
data_nascimento	Data/Hora
formacao	Texto

Exemplo 2: Criando a tabela de disciplina cursada por um aluno

```
CREATE TABLE Cursa(
      cod_aluno         number(5),
      cod_disciplina    number(5),
      semestre          varchar(5),
      nota1             number(2,2),
      nota2             number(2,2),
      exame             number(2,2),
      media_final       number(2,2),
      faltas            number(2),
      situacao          char(1),

CONSTRAINT PK_CURSA PRIMARY KEY
     (cod_aluno,cod_disciplina,semestre),
CONSTRAINT FK_ALUNO FOREIGN KEY (cod_aluno)
           REFERENCES Alunos (codigo),
CONSTRAINT FK_DISCIPLINA FOREIGN KEY (cod_disciplina)
           REFERENCES Disciplina(codigo));
```



Comando: `Alter Table`

O comando ALTER TABLE permite alterar a estrutura de uma tabela(arquivo) acrescentando, alterando, retirando e alterando nomes, formatos das colunas e a integridade referencial definidas em uma determinada tabela. A sintaxe correta para esse comando é:

```
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 CONSTRAINT nome PRIMARY KEY nome-coluna
      DROP PRIMARY KEY nome-coluna
      ADD CONSTRAINT nome FOREIGN KEY (nome-coluna-chave-estrangeira)
            REFERENCES  nome-tabela-pai (nome-campo-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.

`ADD CONSTRAINT nome 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 CONSTRAINT nome 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 retirada uma chave estrangeira da estrutura da tabela.

Comando: `Drop Table`

O comando `DROP TABLE` permite excluir 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. A sintaxe correta para esse comando é:

onde:

```
DROP TABLE nome-tabela
```

a) nome-tabela - Representa o nome da tabela que será deletada.

Comando: Create Index

O comando CREATE INDEX permite 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. A sintaxe para a execução desse comando é:

```
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 que será criada.

Opção ASC/DESC - Representa a criação do índice ordenada crescentemente (ASC) ou decrescentemente (DESC).

Comando: Drop Index

Este comando exclui uma estrutura de índice de acesso para uma determinada coluna em uma tabela. A sintaxe correta para esse comando é:

```
DROP INDEX nome-índice
```

onde:
a) nome-índice - Representa o nome da estrutura de índice que será deletada.

Atenção: Os comados DML desta apostila foram elaborados com o **auxílio on-Line** do banco **MS-ACCESS**, O **SQL** para este banco não é totalmente compatível com o **SQL Padrão ANSI**, que é o oficial na maioria dos bancos de dados, então algumas cláusulas podem não funcionar em outros bancos.

3.2. Comandos DML

Instrução INSERT INTO

Este comando permite adicionar um ou vários registros a uma tabela do Banco de Dados. A sintaxe para a execução é:

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

onde:

nome-tabela - Representa o nome da tabela onde será incluída o registro.

nome-coluna - Representa o nome da(s) coluna(s) terão conteúdo no momento da operação de inclusão.

Relação dos valores:- Representa os valores a serem incluídos na tabela.

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

Quando todos os campos da tabela terão 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. Por exemplo:

```
INSERT INTO Alunos
VALUES (2, "Andre", "303569871", "R. Jaboticaba 37",
27/03/1978, "14-5223778");
```

Quando apenas parte dos campos da tabela terão 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 tem conteúdo especificado será preenchido o valor NULL, como mostra o exemplo abaixo:

```
INSERT INTO Alunos (ra, nome, rg, endereco)
VALUES (1, "Anderson", "2722589-1", "R. Vendaval, 75”);
```

Obs: O Access permite usar INSERT INTO para acrescentar um conjunto de registros de uma outra tabela ou consulta utilizando a cláusula SELECT ... FROM, como mostrado acima na sintaxe da consulta acréscimo de múltiplos registros. Nesse caso, a cláusula SELECT especifica os campos a acrescentar à tabela destino especificada. Veja no exemplo abaixo, a instrução SELECT cria uma tabela de consulta onde serão inseridos os valores passados por parâmetro.

```
INSERT INTO Alunos ( ra, nome, rg, endereco, data_nascimento,
telefone )SELECT ra, nome, rg, endereco, data_nascimento, fone;
```

Instrução UPDATE

Objetivo:

Esse comando permite atualizar os dados de um ou um grupo de registros em uma tabela do Banco de Dados. Sua sintaxe é:

```
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) terão seus conteúdos alterados com o novo valor especificado.

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

Comentários

UPDATE é especialmente útil quando se quer alterar muitos registros ou quando os registros que se quer alterar estão em várias tabelas. É possível alterar vários campos ao mesmo tempo. O exemplo abaixo altera o professor da disciplina de S.O. no período noturno, atribuindo a disciplina um novo Professor.

```
UPDATE Leciona SET cod_pro = 3 WHERE cod_dis = 2 OR cod_dis = 7;
```

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.

Instrução Delete

Essa instrução permite remover (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. Este seleção poderá resultar em um ou vários registros. Neste caso a operação irá ocorrer em todos os registros selecionados.

Comentários

DELETE é especialmente útil quando se quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados.

Pode-se usar DELETE para remover registros de tabelas que estão em uma relação um por 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 Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz com que os registros correspondentes em Pedidos 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 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, não será possível 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.

Exemplo de instrução DELETE

Esse exemplo exclui todos os registros de Professores cujo código seja 1.

```
DELETE * FROM Professores WHERE codigo=1;
```

Se fosse necessário passar um parâmetro para o delete a estrutura seria assim

```
DELETE * FROM Professores
WHERE Professores.nome=[nome professor];
```

Instrução Select

A instrução SELECT é a mais poderosa da linguagem SQL. Ela permite selecionar um conjunto de registros em uma ou mais tabelas que atenda a uma determinada condição definida pelo comando. Sua sintaxe é:

```
SELECT ALL FROM nome-tabela [AS APELIDO] [,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 contem as colunas que serão selecionadas ou que serão utilizadas para a execução da consulta.

Apelido: Os nomes que serão usados como títulos de colunas em vez dos nomes originais das colunas na tabela.

condição - Representa a condição para a seleção dos registros. Este 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 serão grupados 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.

Comentários

Para executar esta operação, o programa principal de banco de dados procura a tabela ou tabelas especificadas, extrai as colunas escolhidas, seleciona as linhas que satisfazem o critério e classifica ou agrupa as linhas resultantes na ordem especificada.

A instrução SELECT **não muda** os dados no banco de dados. SELECT é normalmente a primeira palavra em uma instrução SQL. A maior parte das instruções SQL são instruções SELECT. A sintaxe mínima da instrução SELECT é:

```
SELECT <campos> FROM <tabela>
```

Você pode usar um asterisco (*) para selecionar todos os campos na tabela. O exemplo abaixo seleciona todos os campos na tabela Alunos:

```
SELECT * FROM Alunos;
```

A Clausula WHERE

Freqüentemente é necessário localizar registros em um banco de dados que satisfaçam certos critérios de seleção. O SQL utiliza a clausula WHERE para especificar os critérios de seleção para a consulta. O formato mais simples para uma consulta com critérios é:

```
SELECT * FROM nome-tabela WHERE condição;
```

```
SELECT * FROM Alunos WHERE ano_nascimento < 1984;
```

A condição da cláusula WHERE pode conter os operadores <, >, <=, >=, <> e LIKE. O operador LIKE é utilizado para coincidência de padrão, o que permite a busca de valores semelhantes ao digitado utilizando os caracteres curinga (* e ?). O asterisco indica qualquer número de caracteres em seqüência na posição do asterisco dentro do padrão. Já a interrogação indica um único caracter na posição. Por exemplo, em uma tabela que temos os alunos Anderson, Andre e Manoel.

```
SELECT * FROM Alunos WHERE nome LIKE "?n*";
```

No SELECT acima o retorno será Anderson e Andre, uma vez que buscamos apenas os valores iniciados por qualquer letra, tendo a segunda letra um "n" e finalizado por qualquer seqüência

```
SELECT * FROM Alunos WHERE nome LIKE "*n*";
```

Nesse SELECT procuramos qualquer aluno que tenha a letra "n" em seu nome, no caso Anderson , Andre e Manoel.

A clausula ODER BY

ORDER BY é opcional. Entretanto, se você quiser exibir seus dados na ordem classificada, você deve utilizar ORDER BY. O padrão ordem de classificação é ascendente (A a Z, 0 a 9). Os dois exemplos abaixo classificam os nomes dos funcionários pelo sobrenome.

```
SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome;

SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome ASC;
```

Para classificar em ordem descendente (Z a A, 9 a 0), adicione a palavra reservada DESC ao final de cada campo que você quiser classificar em ordem descendente. O exemplo abaixo seleciona salários e os classifica em ordem descendente

```
SELECT Sobrenome, Salário FROM Funcionários ORDER BY Salário DESC,
Sobrenome;
```

ORDER BY é normalmente o último item em uma instrução SQL. Você pode incluir campos adicionais na cláusula ORDER BY. Os registros são classificados primeiro pelo primeiro campo listado depois de ORDER BY. Os registros que tiverem valores iguais naquele campo são classificados pelo valor no segundo campo listado e assim por diante.

A Clausula INNER JOIN

O INNER JOIN mescla os registros de duas tabelas testando a correspondência com valores em um campo que é comum para as duas tabelas. O formato mais simples de uma clausula INNER JOIN é:

```
SELECT * FROM tabela1 INNER JOIN tabela2
ON Tabela1.campo = Tabela2.campo;
```

A parte ON da cláusula INNER JOIN especifica os campos de cada tabela que devem ser comparados para determinar quais registros serão selecionados. No exemplo abaixo, o campo Departamento está nas tabelas Funcionários e Supervisores. A instrução SQL seleciona Departamento da tabela Funcionários e NomeSupv da tabela Supervisores:

```
SELECT Funcionários.Departamento, Supervisores.NomeSupv
FROM Funcionários INNER JOIN Supervisores
ON Funcionários.Departamento = Supervisores.Departamento;
```

A mesma instrução SELECT poderia ser feita sem a clausula INNER JOIN, porém a performance da consulta seria prejudicada.

```
SELECT Funcionários.Departamento, Supervisores.NomeSupv
FROM Funcionários, Supervisores
WHERE Funcionários.Departamento = Supervisores.Departamento;
```

A Clausula BETWEEN

A clausula BETWEEN permite fazer uma seleção de valores entre um determinado intervalo. Por exemplo deseja-se saber todos os alunos que estão matriculados entre os semestres 2001A ate o semestre 2002B

```
SELECT * FROM Cursa
WHERE Cursa.semestre BETWEEN "2001A" AND "2002B";
```

A Clausula GROUP BY

Os dados resultantes de uma seleção podem ser agrupados de acordo com um critério específico. Este procedimento é realizado usando a cláusula GROUP BY. Apenas uma linha do grupo é apresentada. Como condição de agrupamento somente poderão aparecer itens do SELECT ou funções do grupo. Por exemplo, deseja-se saber o total de créditos cursados pelos alunos. Para resolver esse problema usamos um função SUM (explicada mais adiante) e a clausula GROUP BY.

```
SELECT cod_alu, SUM(Disciplinas.credito) AS [Total de Aula]
FROM Disciplinas INNER JOIN Cursa
ON Cursa.cod_dis= Disciplinas.codigo
GROUP BY cod_alu;
```

Nesse exemplo a quantidade de créditos está na tabela disciplinas, para cada aluno soma-se a quantidade de créditos das disciplinas que ele cursa.

Algumas funções utilizadas no comando SELECT.

a) **COUNT**(*) (nome-campo)

Retorna a quantidade de registros existentes no campo especificado. Quando a opção * é utilizada o resultado é a quantidade de registros existentes. Quando é referenciado o nome de um campo retorna a quantidade de valores existentes na coluna. Por exemplo, deseja-se saber quantos alunos estão matriculados na disciplina de Analise de Sistemas I (código da disciplina igual a 1).

```
SELECT COUNT(cod_alu) AS Total FROM cursa WHERE cod_dis=1
```

Outro exemplo deseja-se saber quantas disciplinas são cursadas pelos alunos

```
SELECT cod_alu, count(*) AS [Total Disciplinas]
FROM Cursa
GROUP BY cod_alu;
```

b) **SUM** (nome-campo)

Retorna a soma dos valores existentes no campo especificado. Por exemplo deseja-se saber quantos créditos (aulas por semana) o aluno Anderson (código igual a 1) cursa, para que se possa calcular a sua mensalidade.

```
SELECT SUM(Disciplinas.credito) AS [Total de Aula]FROM
Disciplinas INNER JOIN Cursa ON
Cursa.cod_dis = Disciplinas.codigo
WHERE Cursa.cod_alu=1;
```

Outro exemplo, deseja-se calcular a mensalidade que os alunos pagam sabendo que o valor do crédito é R$7,00.

```
SELECT cod_alu, SUM(Disciplinas.credito * 7.0)
                              AS [Total de Aula]
FROM Disciplinas INNER JOIN Cursa
```

```
ON Cursa.cod_dis= Disciplinas.codigo
GROUP BY cod_alu;
```

c) **AVG** (nome-campo)

Retorna a média dos valores existentes no campo especificado. Por exemplo, deseja-se saber quantas aulas em média um aluno assiste por dia.

```
SELECT AVG(Disciplinas.credito) AS [Media de Aulas] FROM
Disciplinas INNER JOIN Cursa ON
Disciplinas.codigo=Cursa.cod_dis
```

d) **MAX** (nome-campo)

Retorna o maior valor existente no campo especificado. Por exemplo, deseja-se saber qual aluno teve a nota mais alta na primeira prova da disciplina da Analise de Sistemas I (código da disciplina igual a 1) no semestre 2001A.

```
SELECT MAX(nota1) AS [Maior Nota], Alunos.nome AS nome
FROM Cursa INNER JOIN Alunos ON Alunos.RA=Cursa.cod_alu
WHERE Cursa.cod_dis=1 AND semestre="2001A"
GROUP BY Cursa.cod_Alu, Alunos.nome;
```

e) **MIN** (nome-campo)

Retorna o menor valor existente no campo especificado. Por exemplo, deseja-se saber qual aluno teve a nota mais baixa na primeira prova da disciplina da Analise de Sistemas I (código da disciplina igual a 1) no semestre de 2001A.

```
SELECT MIN(nota1) AS [Menor Nota], Alunos.nome AS nome
FROM Cursa INNER JOIN Alunos ON Alunos.RA=Cursa.cod_alu
WHERE Cursa.cod_dis=1 AND semestre="2001A"
GROUP BY Cursa.cod_Alu, Alunos.nome;
```

4. Bibliografia

FERNANDES, Lúcia; **ORACLE 9i para Desenvolvedores – Curso Completo.** Rio de Janeiro: Axcel Books, 2002.

DEITEL, H. M.; DEITEL, P. J.; **Java Como Programar**. 3 ed. São Paulo: Bookman, 2000.

FANDERUFF, Damaris; **Oracle 8i utilizando SQL* Plus e PL/SQL.** São Paulo:Makron Books, 2000.

HELP On-Line Microsoft Access 97.

Apostilas Internet.