Apostila de Banco de Dados e SQL

Introdução

Devido a carência de literatura destinada ao ensino de Banco de Dados e SQL para estudantes, elaboramos a presente apostila, que não possue o intento de esgotar tão abrangente volume de informações, servindo tão somente para estabelecer um mínimo de conhecimentos destinados a introduzir o estudante no mundo dos Gerenciadores de Banco de dados e da Linguagem SQL.

Banco de Dados

Todos nós sabemos existirem 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 relacioamentos 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 tabelas de nosso banco de dados, algumas vezes com mais de um acesso ao mesmo registro de uma mesma tabela!

O fato de montarmos uma Mala Direta em um micro PC-XT com um drive já faz de nós um autor de um Banco de Dados?

Claro que não! Um Banco de Dados é antes de mais nada uma coleção logicamente coerente de dados com determinada significação intrínseca. Em outras palavras um arquivo contendo uma série de dados de um cliente, um arquivo com dados aleatoriamente gerados e dois arquivos padrão dbf (dBase) que tem uma relação definida entre ambos, não pode ser considerada uma Base de Dados Real.

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 geralmente através da linguagem SQL.

Os Administradores de Banco de Dados (DBA) são responsáveis pelo controle ao acesso aos dados e pela coordenação da utilização do BD. Já os projetistas de Banco de Dados (DBP) são analistas que identificam os dados a serem armazenados em um Banco de Dados e pela forma como estes serão representados.

Os Analistas e Programadores de Desenvolvimento, criam sistemas que acessam os dados da forma necessária ao Usuário Final, que é aquele que interage diretamente com o Banco de Dados.

SGBD x GA

Um SGBD - Sistema de Gerenciamento de Banco de Dados é uma coleção de programas que permitem ao usuário definir, construir e manipular Bases de Dados para as mais diversas finalidades.

Um conceito que deverá ficar bastante claro inicialmente é o que envolve a separação clara entre os Gerenciadores de Base de Dados dos Gerenciadores de Arquivo.

Sistemas baseados em "Banco de Dados" baseados em Btrieve e dBase (Fox e Clipper), podem no máximo simular as características típicas de um ambiente de Banco de Dados. As linguagens Delphi (utiliza opcionalmente o padrão dBase) e o VB (que utiliza o Access), recomendam a utilização de Banco de Dados reais, porém utilizam àqueles "Banco de Dados" que possuem algumas características de Bancos de Dados, mas possuem características típicas de Gerenciadores de Arquivo.

Vamos definir algumas regras básicas e claras para um sistema de manipulação de dados ser considerado um SGBD. Fica implícito que se ao menos uma das características abaixo não estiver presente no nosso "candidato" a SGBD, este poderá ser um GA (Gerenciador de Arquivo) de altíssima qualidade, "quase" um SGBD, mas não um SGBD.

Regra 1: Auto-Contenção- Um SGBD não contém apenas os dados em si, mas armazena completamente toda a descrição dos dados, seus relacionamentos e formas de acesso. Normalmente esta regra é chamada de Meta-Base de Dados. Em um GA, em algum momento ao menos, os programas aplicativos declaram estruturas (algo que ocorre tipicamente em C, COBOL e BASIC), ou geram os relacionamentos entre os arquivos (típicos do ambiente xBase). Por exemplo, quando você é obrigado a definir a forma do registro em seu programa, você não está lidando com um SGBD.

Regra 2: Independência dos Dados- Quando as aplicações estiverem realmente imunes a mudanças na estrutura de armazenamento ou na estratégia de acesso aos dados, podemos dizer que esta regra foi atingida. Portanto, nenhuma definição dos dados deverá estar contida nos programas da aplicação. Quando você resolve criar uma nova forma de acesso, um novo índice, *se precisar alterar o código de seu aplicativo*, você não está lidando com um SGBD.

Regra 3: Abstração dos Dados- Em um SGBD real é fornecida ao usuário somente uma representação conceitual dos dados, o que não inclui maiores detalhes sobre sua forma de armazenamento real. O chamado Modelo de Dados é um tipo de abstração utilizada para fornecer esta representação conceitual. Neste modelo, um esquema das tabelas, seus relacionamentos e suas chaves de acesso são exibidas ao usuário, porém nada é afirmado sobre a criação dos índices, ou como serão mantidos, ou qual a relação existente entre as tabelas que deverá ser mantida íntegra. Assim se você desejar inserir um pedido em um cliente inexistente e esta entrada não for automaticamente rejeitada, você não está lidando com um SGBD.

Regra 4: Visões- Um SGBD deve permitir que cada usuário visualize os dados de forma diferente daquela existente previamente no Banco de Dados. Uma visão consiste de um subconjunto de dados do Banco de Dados, necessariamente derivados dos existentes no Banco de Dados, porém estes não deverão estar explicitamente armazenados. Portanto, toda vez que você é obrigado a replicar uma estrutura, para fins de acesso de forma diferenciada por outros aplicativos, você não está lidando com um SGBD.

Regra 5: Transações- Um SGBD deve gerenciar completamente a integridade referencial definida em seu esquema, sem precisar em tempo algum, do auxílio do programa aplicativo. Desta forma exige-se que o banco de dados tenha ao menos uma instrução que permita a gravação de uma série modificações simultâneas e uma instrução capaz de cancelar um série modificações. Por exemplo, imaginemos que estejamos cadastrando um pedido para um cliente, que este deseje reservar 5 itens de nosso estoque, que estão disponíveis e portanto são reservados, porém existe um bloqueio financeiro (duplicatas em atraso) que impede a venda. A transação deverá ser desfeita com apenas uma instrução ao Banco de Dados, sem qualquer modificações suplementares nos dados. Caso você se obrigue a corrigir as reservas, através de acessos complentares, você não está lidando com um SGBD.

Regra 6: Acesso Automático- Em um GA uma situação típica é o chamado Dead-Lock, o abraço mortal. Esta situação indesejável pode ocorrer toda vez que um usuário travou um registro em uma tabela e seu próximo passo será travar um resgistro em uma tabela relacionada à primeira, porém se este registro estiver previamente travado por outro usuário, o primeiro usuário ficará paralisado, pois, estará esperando o segundo usuário liberar o registro em uso, para que então possa travá-lo e prosseguir sua tarefa. Se por hipótese o segundo usuário necessitar travar o registro travado pelo primeiro usuário (!), afirmamos que ocorreu um abraço mortal, pois cada usuário travou um registro e precisa travar um outro, justamente o registro anteriormente travado pelo outro! Imaginemos um caso onde o responsável pelos pedidos acabou de travar o Registro Item de Pedido, e, necessita travar um registro no Cadastro de Produtos, para indicar uma nova reserva. Se concomitantemente estiver sendo realizada uma tarefa de atualização de pendências na Tabela de Itens, e para tanto, previamente este segundo usuário travou a Tabela de Produtos, temos a ocorrência do abraço mortal. Se a responsabilidade de evitar esta ocorrência for responsabilidade da aplicação, você não está lidando com um SGBD.

Conclusão: Um SGBD deve obedecer INTEGRALMENTE as seis regras acima. Em caso contrário estaremos diante de um GA ou de um "quase" SGBD.

Considerações Finais

Atualmente, existe uma tendência de mercado em se dizer que qualquer problema será resolvido, caso a empresa adquira um Banco de Dados. Naturalmente, em um ambiente com acesso constante ao Banco de Dados (acesso concorrente, obviamente), onde a segurança seja de vital importância e que o desempenho da aplicação escrita estiver comprometendo a empresa, considerando-se logicamente uma aplicação bem escrita, sem dúvida a aquisição de um Banco de Dados *poderá ser* o primeiro passo na solução do problema.

Analogamente ao que ocorreu com o aparecimento das primeiras linguagens de programação voltadas ao Windows, onde estas foram apresentadas como capazes de alavancar os negócios da empresa, e no geral causaram mais frustação do que solução, a aquisição do Banco de Dados, pode gerar o mesmo tipo de problema.

É fundamental que a empresa candidata a utilizar um Banco de Dados, normatize-se totalmente, pois soluções “quebra-galho”, típicas do ambiente que dispõe de um Gerenciador de Arquivo, tendem a ser impossíveis em um ambiente estruturado sobre o Banco de Dados. Portanto, sob pena de se realizar um grande investimento, e não se colher fruto algum, é muito conveniente, que a empresa *antes* de adquirir um Banco de Dados, passe por um processo de adaptação, preferencialmente contando com pessoal especializado, geralmente consultores, que *não* tenham qualquer ligação com fabricantes de Bancos de Dados.

Características Gerais de um SGBD

Os SGBD tem sete características operacionais elementares sempre observadas, que passaremos a listarr:

Característica 1: Controle de Redundâncias- A redundância consiste no armazenamento de uma mesma informação em locais diferentes, provocando inconsistências. Em um Banco de Dados as informações só se encontram armazenadas em um único local, não existindo duplicação descontrolada dos dados. Quando existem replicações dos dados, estas são decorrentes do processo de armazenagem típica do ambiente Cliente-Servidor, totalmente sob controle do Banco de Dados.

Característica 2: Compartilhamento dos Dados- O SGBD deve incluir software de controle de concorrência ao acesso dos dados, garantindo em qualquer tipo de situação a escrita/leitura de dados sem erros.

Característica 3: Controle de Acesso- O SGDB deve dispor de recursos que possibilitem selecionar a autoridade de cada usuário. Assim um usuário poderá realizar qualquer tipo de acesso, outros poderão ler alguns dados e atualizar outros e outros ainda poderão somente acessar um conjunto restrito de dados para escrita e leitura.

Característica 4: Interfaceamento- Um Banco de Dados deverá disponibilizar formas de acesso gráfico, em linguagem natural, em SQL ou ainda via menus de acesso, não sendo uma "caixa-preta" somente sendo passível de ser acessada por aplicações.

Característica 5: Esquematização- Um Banco de Dados deverá fornecer mecanismos que possibilitem a compreensão do relacionamento existentes entre as tabelas e de sua eventual manutenção.

Característica 6: Controle de Integridade-Um Banco de Dados deverá impedir que aplicações ou acessos pelas interfaces possam comprometer a integridade dos dados.

Característica 7: Backups- O SGBD deverá apresentar facilidade para recuperar falhas de hardware e software, através da existência de arquivos de "pré-imagem" ou de outros recursos automáticos, exigindo minimamente a intervenção de pessoal técnico.

Existe a possibilidade de encontramos Bancos de Dados que não satisfaçam completamente todas as características acima, o que não o inválida como Banco de Dados. Na prática podemos encontrar situações onde a primeira característica não seja importante, pois podemos ter o Banco de Dados baseado totalmente em um único servidor, e as redundâncias podem ser aceitas em algumas situações sob controle da aplicação (algo não muito recomendado, mas passível de aceitação, em situações onde a existência do nome do cliente em um arquivo contendo duplicatas emitidas, possibilita o acesso a apenas uma tabela sem relacionamentos, e sabe-se de antemão que uma duplicata depois de emitida, não pode ter seu cliente alterado).

A segunda característica (Compartilhamento dos Dados) pode ser desconsiderada principalmente em ambiente de desenvolvimento, ou ainda em aplicações remotas.

O Controle de Acesso pode ser descartado em pequenas empresas, sendo que o aplicativo em questão, mais o software de rede, podem facilmente se imcumbir desta característica, no caso de pequenas empresas, com reduzido número de pessoas na área operacional.

O Interfaceamento e a Esquematização, são características sempre disponíveis, o que varia neste caso é qualidade destes compoenentes, que vai desde o sofrível até o estado da arte. É muito conveniente que esta característica seja muito boa em um Banco de Dados, onde estiverem em atuação mais de um Administrador de Banco de Dados e tivermos um número relativamente alto de sistemas desenvolvidos ou em desenvolvimento neste ambiente.

De fato, quanto maior o número de pessoas envolvidas no desenvolvimento de aplicações e gerenciamento do Banco de Dados, mais importante tornam-se estas duas características, pois cada novo sistema desenvolvido precisará sempre estar adequado ao Banco de Dados da Empresa e aderente aos padrões de acesso utilizados nos sistemas concorrentes.

As interfaces ISQL e WinSQL devem deixar muito claro ao estudante como uma interface pobre (no caso a existente no ISQL) perde muito, quando comparada a uma interface mais recursiva. A esquematização existente no Banco de Dados é muito melhor do que aquela mantida em alguma pasta, em algum arquivo do CPD, que sempre está “um pouquinho” desatualizada.

O Controle de Integridade, é outra característica sempre presente nos Bancos de Dados, mas existem diferenças quando da implementação desta característica. Assim, é comum encontrarmos Bancos de Dados que suportam determinado acesso, enquanto outros não dispõe de recurso equivalente.

O Backup em tempo de execução, é outra característica sempre disponível, porém temos aplicações que invariavelmente são comprometidas por falhas de hardware, e outras, que o mesmo tipo de falha não causa perda alguma de dados ou de integridade. Novamente, cada Banco de Dados tem esta característica melhor ou pior implementada, cabendo ao Administrador de Banco de Dados escolher aquele que lhe oferecer mais segurança.

Devemos ressaltar ainda, que podemos ter um Banco de Dados Modelo A, que respeite integralmente as regras básicas e disponha de todas as características apresentadas, enquanto um Modelo B que apesar de respeitar as regras básicas, não suporte uma ou outra característica desejável, mas tenha um desempenho excelente, enquanto o Modelo A seja apenas razoável no quesito desempenho, nos levará seguramente a escolher o Modelo B como sendo o ganhador para nossa instalação!

Isto ocorre pois, na prática, todo usuário deseja um tempo de resposta muito pequeno. O chamado “prazo de entrega” muito comum em Bancos de Dados operando nos limites de sua capacidade, ou nos casos onde o hardware está muito desatualizado, é fonte de inúmeros problemas para o pessoal de informática. Neste caso é melhor abrirmos mão de uma Interface Amigável, de um Gerenciamente Automático de Backups ou ainda de outras características que não julgarmos fundamentais, para nos livrarmos do problema típico de ambiente extremamente comprometido, por má performance do Banco de Dados.

A escolha do Banco de Dados da empresa, portanto é uma decisão muito delicada, na medida em que está irá acarretar troca de aplicativos e troca de hardware. Os investimentos diretamente aplicados no Banco de Dados, costumam ser infinitamente menores do que aqueles a serem aplicados na empresa, visando sua perfeita adeqüação ao novo SGBD. Esta decisão, sempre que possível, deve ser tomada por especialistas em Banco de Dados, com profundos conhecimentos de Análise de Sistemas, de Banco de Dados e de Software de Gerenciamento de Bases de Dados, de forma a evitar que a empresa escolha um Banco de Dados inadequado aos seus propósitos, e que pouco tempo depois, seja obrigada a perder todos investimento realizado em Software e Hardware.

Arquitetura de um SGBD

Estrutura

Podemos dizer que o Banco de Dados tem um Nível Interno, onde é descrita a estrutura de armazenamento físico dos dados, um Nível Intermediário, onde temos a descrição lógica dos dados e um Nível Externo onde são descritas as visões para grupos de usuários.

Não podemos deixar de lembrar ainda que o Banco de Dados garante a Independência Lógica e Física dos Dados, portanto podemos alterar o esquema conceitual dos dados, sem alterar as visões dos usuários ou mesmo alterar o esquema interno, sem contudo alterar seu esquema conceitual.

Modelos de Dados

O Modelo de Dados é basicamente um conjunto de conceitos utilizados para descrever um Banco de Dados. Não existe uma única forma de representação deste modelo, porém qualquer forma que permita a correta compreensão das estruturas de dados compreendidas no Banco de Dados, pode ser considerada adequada. Vamos descrever sucintamente este modelo, pois estes serão objetos de outras disciplinas:

Modelo Orientado ao Registro: São modelos que representam esquematicamente as estruturas das tabelas de forma bastante próxima a existente fisicamente. Basicamente são apresentados os registros de cada tabela (inclusive seus campos) e seus relacionamentos elementares. O Modelo Relacional, o Modelo de Rede e o Hierárquico são exemplos deste tipo de representação.

Modelo Semântico: São modelos onde existe uma representação explícita das entidades e relacionamentos. O Modelo Entidade-Relacionamento e o Funcional, são exemplos deste tipo de abordagem.

Modelo Orientado ao Objeto: São modelos que procuram representar as informações através dos concietos típicos da Programação Orientada ao Objeto, utilizando o conceito de Classes que irão conter os objetos. Citamos os Modelos O2 e o de Representação de Objetos como exemplos típicos desta abordagem.

O conceito de instância, sempre muito presente, poderia ser definido como sendo o conjunto de dados que definem claramente um Banco de Dados em deteminado instante. Devemos entender então o Banco de Dados como sendo não apenas um conjunto de dados digitados, mas também todo o esquema e regras armazenada e controladas pelo SGBD.

Em outras palavras, podemos dizer que os SGBD, vieram para eliminar todo o trabalho que anteriormente um programador de aplicação realizava controlando o acesso, integridade e redundância dos dados.

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.

Tipos de SGBD

Introdução

Podemos citar como tipos principais os Bancos de Dados Relacionais, os Hierárquicos, os de Rede, os Semânticos, os Orientados a Objetos e os Universais.

Os Bancos de Dados alvo de nosso estudo serão os Relacionais, sendo que os demais tipos serão apenas citados superficialmente, por não serem parte integrante de nosso curso.

Esquema de Organização dos Dados

Em Linguagem C os estudantes tomaram (ou irão tomar contato) com os ponteiros de registro, que aqui representaremos como sendo números de acesso ao registro. Visando diferenciar o número do registro físico do número do registro apontado pelo ponteiro, usaremos o símbolo (#) para indicar o número do registro físico, enquanto o símbolo (*) será utilizado pelo para indicar o endereço indicado pelo ponteiro, a semelhança da representação usual dos programadores da Linguagem C.

Vamos supor o arquivo abaixo ordenado alfabeticamente (físico):

#1- Amarelo *2
#2- Azul *3
#3- Branco *4
#4- Preto *5
#5- Verde *6
#6- Vermelho --

Supondo desejarmos incluir a cor Laranja, seríamos obrigado a re-escrever todo o arquivo de modo a Laranja ocupar o registro 4. Vamos antes de fazer uma nova ordenação, analisar a solução abaixo:

#1- Amarelo *2
#2- Azul *3
#3- Branco *7
#4- Preto *5
#5- Verde *6
#6- Vermelho --
#7- Laranja *4

Observe que o registro #3 (Branco) passou a apontar para o registro *7, que contém o novo dados (Laranja). O novo dado passa a apontar para o registro previamente apontado pelo registro que agora o aponta. Parece, e é confuso, mas se você analisar o esquema abaixo perceberá que apesar do palavreado confuso, facilmente qualquer um de nós percebe a maneira adequada de inserir novos registros.

Algo --> Apontado

Algo --> Novo --> Apontado

#1 Algo --> *2
#2 Apontado não aponta (é o último física e logicamentex)

#1 Algo --> *3

#2 Apontado não aponta (é o último logicamente)
#3 Novo --> *2 (é o último fisicamente)

A chamada perda de ponteiros, fenômeno dos mais temidos pelos profissionais de sistema, nada mais é que a perda de referência lógica entre registros de uma tabela.

Existem diversas técnicas de acesso como as chamadas Btree+ (Arvore Binária Balanceada), Hashing, Sequencial Ordenado, Hashing Dinâmico, Hashing Extensível e Hashing Linear, próprios para um curso específico de Banco de Dados, que não chegaremos a analisar em nosso curso.

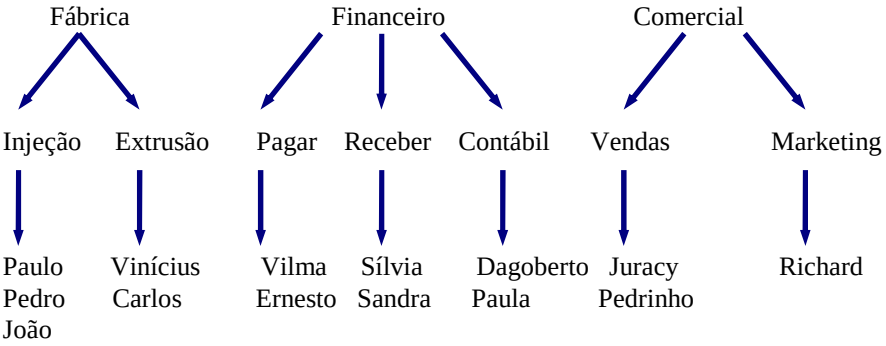
Sabemos que em linguagem C foi (ou será) apresentada a técnica de balanceamento de estruturas, que mostrou (ou mostrará) como um Banco de Dados é arranjado internamente.

Exercício: Represente esquematicamente o arranjo alfabético dos seguintes times de futebol: Fluminense, Flamengo, Vasco, Botafogo, Madureira, América e Olária. Suponha ainda que o Bangu queira participara do campeonato, como ficaria a nova ordem da tabela?

Banco de Dados Hierárquicos

Seguem o estilo de um organograma empresarial (Diretoria-Divisão-Seção-Setor) ou de biblioteca (Exata-Matemática-Algebra Linear-Vetores). Este modelo é capaz de representar este tipo de organização de forma direta, mas apresenta inconvenientes quando esta situação não aparece claramente com relações de hierarquia.

O Exemplo a seguir (Folha de Pagamento) deve servir para esclarecer melhor o estilo deste modelo



Sabemos que Paulo é "filho" da Injeção que por sua vez é "filha" da Fábrica.

Banco de Dados em Redes

Neste modelos os dados são dispostos em registros, previamente classificados em classes que descrevem a estrutura de determinado tipo de registro. Os registros são descritos em relações de conjuntos onde são estabelecidas as ligações lógicas entre eles.

O esquema abaixo representa este tipo de Ligação

Fábrica
#1 Nome Local ... Apontada Aponta_Início Aponta_Final
Injeção
#7 Nome Máquina ... Apontada (*1) Aponta_I(*15) Aponta_F(*18)
#15 Paulo 28 (Idade) ... (*7) (*17)
#18 João 25 ... (*17) (*7)

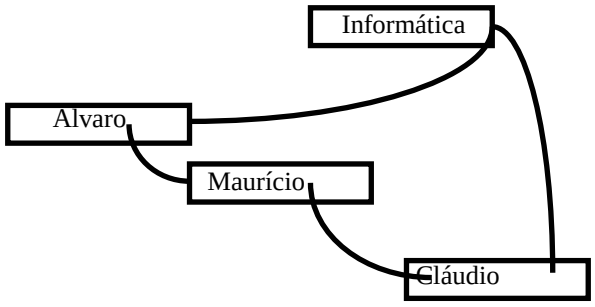
Um confusão habitualmente verificada, diz respeito a confusão que existe entre o conceito do Modelo de Redes e o existente na matemática. No modelo de Redes temos sempre um elemento distitivo, o registro base e a partir dele são dispostos os demais registros. Temos sempre *tipos de conjunto*, que dispõe de três elementos, a saber: nome, tipo de registro pai e tipo de registro filho. Supondo um Registro contido no Arquivo de Disciplinas ministradas na Íbero, este seria um registro pai, na medida em que conteria a referência aos seus registros filhos (os alunos cursando aquela disciplina).

As restrições impostas pelo Modelo de Redes podem ser descritas como de ordem de Entrada e de Existência. Em relação as restrições de entrada citamos a obrigatoriedade de cada novo registro estar conectado (ou apontado, como preferem os programadores C) ao conjunto indicado. Em relação a restrições de Existência

podemos dizer que um componente de um tipo de registro pode existir de forma independente de outros desde que esteja conectado a algum outro registro fazendo parte de algum conjunto, ou sendo base de um novo conjunto. A identificação de um conjunto pode ser verificada através do esquema de ligação entre o registro pai e o registro filho, assim sendo, cada instância de conjunto apresenta um elemento de distinção, o tal registro pai, e os registros filhos devidamente ordenados, e portanto passíveis de serem acessados pelos seus elementos.

Exemplo: Disciplina Tópicos Avançados e seus Alunos

Registro de Disciplinas



Registro de Alunos

O exemplo anterior representa uma instância de connjunto, no caso Disciplinas (Tópicos Avançados) e seus alunos (no caso Álvaro, Amorim e Cláudio).

Banco de Dados Orientados ao Objeto

Representam os dados como coleções que obedecem propriedades. São modelos geralmente conceituais dispondo de pouquíssimas aplicações reais. Neste Modelo 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 alocado. 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 Universal

Usa fortemente o conceito dos bancos de dados relacionais (ainda a serem vistos), no que concerne ao tratamento da informação dita caracter e muito do Modelo Orientado ao Objeto, no tocante ao tratamento de Imagens e Sons. É um dos assuntos top do momento, e será alvo de pesquisas na disciplina Tópicos Avançados - Atualidades, não sendo objeto imediato de nossa matéria.

Banco de Dados Relacional

O Modelo de Dados relacional representa os dados contidos em um Banco de Dados através de relações. Estas relações contém 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 passíveis de serem assumidos por um atribruto, será intitulado de *DOMÍNIO*.

Estes tópicos serão estudados cuidadosamente na disciplina Análise de Sistemas, que se incumbirá de apresentar cuidadosamente regras e normas para elaboração destes modelos.

Em nosso curso, voltado à construção prática dos Bancos de Dados, e não de sua construção teóricas, apenas citaremos os aspectos básicos da construção teórica, de forma a facilitar ao estudante o relacionamento que existe entre Análise de Sistemas e Banco de Dados (uma das sub-disciplinas de Tópicos Avançados).

O domínio consiste de um grupo de valores atômicos 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).

O esquema de uma relação, nada mais são que os campos (colunas) existentes em uma tabela. Já a instância da relação consiste no conjunto de valores que cada atributo assume em um determinado instante. Portanto, os dados armazenados no Banco de Dados, são formados pelas instâncias das relações.

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 íveis de novas divisões.

Chamaremos de Chave Primária ao Atributo que definir um resgistro, dentre uma coleção de registros. Chave Secundária (Terceária, etc), serão chaves que possibilitarão pesquisas ou ordenações alternativas, ou seja, diferentes da ordem criada a partir da chave primária ou da ordenação natural (física) da tabela. Chamaremos de Chave Composta, aquela chave que contém mais de um atributo (Por exemplo um cadastro ordenado alfabéticamente 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 (E)
```

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-->Estado.

Forma Normal

A disciplina Análise de Sistemas abordará detalhadamente esta importante metodologia para definição das tabelas que irão compor a base de dados, que aqui apenas citaremos.

Primeira Forma Normal: Uma relação se encontra na primeira forma normal se todos os domínios de atributos possuem apenas valores atômicos (simples e indivisíveis), e que os valores de cada atributo na tupla seja um valor simples. Assim sendo todos os atributos compostos devem ser divididos em atributos atômicos.

Segunda Forma Normal: Uma relação se encontra na segunda forma normal quando estiver na primeira forma normal e todos os atributos que não participam da chave primária são dependentes desta. Assim devemos verificar se todos os atributos são dependentes da chave primária e retirar-se da relação todos os atributos de um grupo não dependente que dará origem a uma nova relação, que conterá esse atributo como não chave. Desta maneira, na segunda forma normal evita inconsistências devido a duplicidades.

Terceira Forma Normal: Uma relação estará na terceira forma normal, quando estiver na primeira forma norma e todos os atributos que não participam da chave primária são dependentes desta porém não transitivos. Assim devemos verificar se existe um atributo que não depende diretamente da chave, retirá-lo criando uma nova relação que conterá esse grupo de atributos, e defina com a chave, os atributos dos quais esse grupo depende diretamente.

O processo de normalização deve ser aplicado em uma relação por vez, pois durante o processo de normalização vamos obtendo quebras, e por conseguinte, novas relações. No momento em que o sistema estiver satisfatório, do ponto de vista do analista, este processo iterativo é interrompido. De fato existem literaturas indicando quarta, quinta formas normais, que não nos parece tão importante, nem mesmo academicamente.

A normalização para formas apoiadas em dependências funcionais evita inconsistências, usando para isso a própria construção da Base. Se a mesma consistência for passível de ser garantida pelo aplicativo, a normalização pode ser evitada com ganhos reais no desempenho das pesquisas. No caso da consistência não ser importante, também podemos não normalizar totalmente uma Base de Dados.

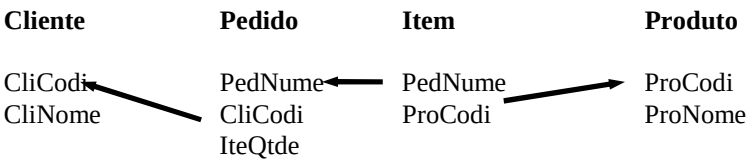
Exemplo: Normalizar os seguintes atributos:

Nº do Pedido, Nome do Cliente, Nome dos Produtos, Quantidades

Nº do Pedido, Código do Cliente, Nome dos Produtos, Quantidades
Código do Cliente, Nome do Cliente

Nº do Pedido, Código do Cliente, Código dos Produtos, Quantidades
Código do Cliente, Nome do Cliente
Código do Produto, Nome do Produto

Nº do Pedido, Código do Cliente
Código do Cliente, Nome do Cliente
Código do Produto, Nome do Produto
Nº do Pedido, Código do Produto, Quantidade



O esquema apresentado anteriormente poderia ser inferido diretamente, usando metodologia tipicamente apresentada em Organização e Método. Se soubermos, por hipótese, que um profissional habilitado desenhou o pedido da empresa, e que esta o está utilizando com sucesso, poderíamos basear nosso modelo de dados neste formulário. Devemos notar que muitos Analistas de Sistemas não adotam estes procedimentos, por preferirem os métodos convencionais para elaboração do Modelo de Dados.

Considerando qualquer formulário de pedidos podemos notar que o **Número do Pedido** geralmente tem destaque e *sempre é único*, ou seja encontramos nossa *chave primária* da **Tabela de Pedidos**, como sabemos que um cliente pode fazer mais de uma compra, achamos nossa **Tabela de Clientes,** que pode ter um **Código**, portanto achamos sua *chave primária*, que por conseguinte será a *chave estrangeira* da **Tabela de Pedidos.**

Um ponto delicado, diz respeito aos itens do pedido, que formam geralmente um espaço destacado dentro do formulário de pedidos. Geralmente, e este é um dos casos, estas áreas em separado dos formulários darão origem a tabelas filhas, como é o caso típico das duplicatas em notas fiscais, ou dos dependentes na ficha de funcionários. Portanto achamos nossa **Tabela de Itens** que será ligada à Tabela de Pedidos através do **Número do Pedido,** que é ao mesmo tempo *chave primária e chave estrangeira* para a Tabela de Itens.

Finalmente podemos perceber, que da mesma forma como os clientes se repetem em relação a Tabela de Pedidos, os produtos podem se repetir na tabela de itens (observe que não obstante não termos nenhum pedido com o mesmo item grafado duas vezes, este item pode ser adquirido em outro pedido). Assim descobrimos nossa quarta tabela, a **Tabela de Produtos** e a chave primária **Código do Produto.**

SQL - Structured Query Language

Introdução

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas à sua manipulação. O Departamento de Pesquisas da IBM, desenvolveu a SQL como forma de interface para o sistema de BD relacional denominado SYSTEM R, início dos anos 70. Em 1986 o American National Standard Institute (ANSI), publicou um padrão SQL.

A SQL estabeleceu-se como linguagem padrão de Banco de Dados Relacional.

SQL apresenta uma série de comandos que permitem a definição dos dados, chamada de DDL (Data Definition Language), composta entre outros pelos comandos Create, que é destinado a criação do Banco de Dados, das Tabelas que o compõe, além das relações existentes entre as tabelas. Como exemplo de comandos da classe DDL temos os comandos Create, Alter e Drop.

Os comandos da série DML (Data Manipulation Language), 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.

Uma subclasse de comandos DML, a DCL (Data Control Language), dispõe de comandos de controle como Grant e Revoke.

A Linguagem SQL tem como grandes virtudes 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. Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens independente das tabelas e organização lógica dos dados.

Outra característica interessante na linguagem SQL é 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.

Devemos notar que a linguagem SQL consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices.

O Ambiente SQL

Dispomos na Ibero de dois softwares destinados a linguagem SQL o ISQL e o WinSQL.

O ISQL faz parte do pacote Ideo e permite construirmos Banco de Dados e tabelas diretamente pelo interpretador SQL, bem como acessarmos as Bases de Dados construídas no Ideo. O ISQL pode gerar Banco de Dados em seu ambiente proprietário (Watcom, hoje pertencente a Symantech) ou ainda nos consagrados Banco de Dados Oracle, SyBase, Ingres (Computer Associates), DB/2 (IBM) e Informix. Devido as origens do Ideo, o Banco de Dados SQL Server da Microsoft não é suportado, pois este Banco de Dados originou-se na microinformática e somente recentemente a Sapiens migrou seu software dos Ambientes Mainframe e Unix.

Já o WinSQL é um ambiente inteiramente gráfico (ao contrário do ISQL que guarda fortes características do ambiente em Mainframe onde se originou), destinado ao apredinzado, portanto somente pode criar Banco de Dados em seu formato proprietário.

Os comandos do WinSQL por serem visuais, não necessitam de maior esclarecimento além daqueles já contidos no Help. Já o ISQL apesar de possuir um Help bastante completo necessita, em nosso entender, de alguns esclarecimentos iniciais.

Uma série de comandos do interpretador, que funciona de forma análoga àquela existente no dBase modo interativo, podem ser utilizados pelo usuário. Não obstante alguns comandos tenham nome idêntico a alguns comandos do DOS, devemos notar que muitas vezes sua sintaxe é bastante diversa daquele sistema operacional. Vamos destacar os seguintes comandos:

\EDIT - Carrega o editor de bloco de notas do windows, o qual serve para a criação de arquivos para serem executados no Ideo.
Ex: \edit teste.sql

\CD - Mostra o diretório onde serão gravados os arquivos *.sql, *.dic *.dat.
Permite alterar para determinado diretório (\CD DADO, fará com que o diretório corrente passe a ser C:\DADO, caso o diretório corrente fosse a raiz. Permite retornar ao diretório de nível inferior (\CD ..).
Atenção este comando não é análogo ao Change Dir do DOS, na medida em que não permite a mudança direta de um subnível do diretório X para um diretório Y por exemplo.

\DEFAULT <drive> - permite alterarmos o drive corrente.
Ex: \DEFAULT F:

\INCLUDE - Executa arquivos *.sql. O arquivo .sql deverá conter uma série de instruções SQL.
Ex: \include teste.sql

@< file > ; - Também executa arquivos *.sql.
Ex: @teste.sql;

EXIT; - Finaliza a sessão do ISQL. ou (**\QUIT**)

COMMIT; - Confirma a transação.

ROLLBACK; - Desfaz a transação.

SHOW <tabela>; - Mostra os nomes das tabelas existentes em determinado banco de dados.
Ex: SHOW tables;

SHOW FIELDS FOR <tabela>; - Mostra os campos de determinada tabela.
Ex: SHOW FIELDS FOR ATOR;

SHOW INDEXES FOR <tabela>; - Lista de indices da tabela.

SHOW RELATIONSHIPS FOR <tabela>; - Lista de relacionamentos da tabela.

LIST <tabela> ; - Lista conteúdo da tabela.

Estudo Dirigido

Consideramos a linguagem SQL eminentemente prática, desta forma criamos um exmplo completo e propomos um exercício análogo, para tornar o estudante apto a manipular a linguagem SQL de maneira prática, em conformidade a filosofia eminentemente prática da Linguagem SQL.

O exemplo apresentado nesta apostila já está disponível para sua utilização do diretório \IDEO\SQL, bastando para isso você copiar este exemplo para seu diretório e iniciar os testes de forma simultânea a sua apresentação pelo professor.

É conveniente que você procure montar o exercício clássico (mundo), de forma a testar todos os conhecimentos adquiridos. Para tanto analise cuidadosamente o exercício proposto a seguir, e construa as relações, tabelas e queries adequadas ao final de cada exemplo.

Exercício: Elabore Banco de Dados Mundo que contenha as seguintes tabelas: Continente, País e Cidade. Observe que uma cidade deverá pertencer exclusivamente a um país e que cada país deverá estar cadastrado no continente onde se localizar sua área mais importante. Assim não obstante grande parte do território russo fazer parte Ásia, a Rússia será considerada fazendo parte da Europa. Assim teríamos basicamente uma relação do tipo:

Cidade --> País --> Continente

PARTE I - Comandos de Modificações do Esquema e Criação de Banco de Dados

Comando Create
Este comando permite a criação de tabelas no banco de dados ou mesmo de sua criação.

Sintaxe:
CREATE DATABASE < nome_db >;

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

Sintaxe:
```
CREATE TABLE < nome_tabela >
( nome_atributo1 < tipo > [ NOT NULL ],
  nome_atributo2 < tipo > [ NOT NULL ],
      ......
      nome_atributoN < tipo > [ NOT NULL ] ) ;
```

onde:

nome_table -	indica o nome da tabela a ser criada.
nome_atributo -	indica o nome do campo a ser criado na tabela.
tipo -	indica a definição do tipo de atributo (integer(n), char(n), real(n,m), date...).

n- número de dígitos ou de caracteres
m- número de casas decimais

Agora vamos criar uma tabela. Use o editor para salvar em um arquivo ou digite na linha de comando do ISQL.

CREATE DATABASE TRABALHO;

O comando acima criou um Banco de Dados, porém este na verdade não passa de uma abertura no diretório, pois não conta com nenhuma tabela.

Agora criaremos as tabelas que estarão contidas no Banco de Dados TRABALHO.

A primeira Tabela será a de Departamentos (DEPT). Esta tabela conterá além dos campos também sua chave primária, suas chaves estrangeiras e também seus índices. A segunda tabela será a de Empregados (EMP), que também será criada.

Não devemos esquecer de primeiramente abrirmos o Banco de Dados. Diferentemente do que ocorre em alguns aplicativos, em SQL o fato de criarmos um Banco de Dados, não significa que o banco recém criado já está preparado para utilização. A instrução a seguir, providencia a abertura do Banco de Dados criado.

OPEN DATABASE TRABALHO;

Agora estamos prontos para criarmos as tabelas necessárias. Lembramos aos Estudantes, que o Arquivo TABS.SQL, contém todas as instruções necessárias para criação do Banco de Dados Trabalho e de suas tabelas. Já o Arquivo DADOS.SQL irá popular estas tabelas. Para efeitos didáticos, criamos as tabelas de forma que sua população, em outras palavras os dados, sejam facilmente referenciáveis pelos estudantes. Assim sendo, na tabela de departamentos, contamos com 5 departamentos, cada um deles tendo seu gerente. Todos os “gerentes” tem nomes de cantoras brasileiras (Gal Costa, Marina Lima, etc), todos os “operários”

tem nomes de jogadores de futebol, todas as vendedoras tem nomes de jogadoras de volei, todas as balconistas tem nome de jogadoras de basquete e o presidente da empresa exemplo, tem o mesmo nome do presidente do Brasil. Desta forma os testes devem resultar em grupos bastante definidos. Assim se você estiver listando Gerentes e aparecer um homônimo da Ana Paula (jogadora de volei), verifique sua query atentamente, pois muito provavelmente a mesma estará errada.

A seguir código necessário a criação da tabela Departamento e seu índice:

```
create table Dept
    (DepNume    integer(4)   not null,
     DepNome    char(20)     not null,
     DepLoca    char(20)     not null,
     DepOrca    integer(12,2),
     primary key (DepNume)
  );

create unique index DepNum on Dept (DepNume asc);
```

Note-se que a chave primária já está definida juntamente com o registro da tabela. A criação do índice, que por razões óbvias deve ser criado após a tabela, naturalmente é um comando totalmente independente do primeiro create, que serviu para criar a tabela e suas característica básicas.

Vamos analisar o código necessário para a criação da tabela de empregados, apresentado a seguir:

```
create table Emp
    (EmpNume            integer(5) not null,
     EmpNome            char(30)   not null,
     EmpGere            integer(5) ,
     EmpServ            char(20)   ,
     DepNume            integer(4) not null,
     EmpAdmi            date       not null,
     EmpSala            integer(10,2),
     EmpComi            integer(10,2),
     primary key (EmpNume),
     foreign key has (DepNume)
          references Dept
          on delete restrict
          on update cascade
  );

create unique index EmpNum on Emp (EmpNume asc);
create index EmpDep on Emp (DepNume asc);
```

A Tabela de Empregados não poderia ter sido criada *antes* da Tabela de Departamento, pois contém uma referência direta àquela tabela. Quando declaramos que *DepNume* é chave estrangeira, promovemos de fato a ligação do cadastro de empregados como o cadastro de departamentos. Ao restringirmos as exclusões, permitimos a existência de funcionários não alocados a nenhum departamento. Apesar desta prática ser contrária a tese de que devemos possuir apenas tuplas perfeitamente relacionáveis em nossas tabelas, podemos deixar esta pequena abertura, pois um usuário que excluisse inadivertidamente determinado departamento, acabaria por excluir também uma grande quantidade de funcionários, que estivessem ligados a este departamento.

Já a atualização em cascata dos códigos de departamento é uma boa providência, na medida em que teremos, uma vez alterado algum código de departamento, a atualização imediata de todos os funcionários pertencentes ao departamento cujo código foi modificado.

Observações:

1- Observar que os índices são parte intrínseca das tabelas.
2- A integridade relacional é garantida pelo Banco de Dados e não pelo aplicativo.
3- Exclusões ou Alterações em Chaves Primárias, podem acarretar exclusões, anulações ou até mesmo perda de integridade nas tabelas onde esta chave primária existir como chave estrangeira. Portanto é imprescindível muito cuidado quando da elaboração do Banco de Dados. Uma tentação muito comum ao estudante é começar criando as tabelas do Banco de Dados sem prévia Normalização. Este talvez seja o melhor caminho para perder-se tempo em vão, pois quando você terminar de projetar suas telas de entrada de dados, notará "que nada funciona!". Esta será a senha para usar o velho comando DEL do DOS e depois começar tudo de novo ...

Comando Drop

Este comando elimina a definição da tabela, seus dados e referências.

Sintaxe:
DROP TABLE < nome_tabela > ;

Ex:
DROP TABLE EMP;

Comando Alter

Este comando permite inserir/eliminar atributos nas tabelas já existentes.

Comando:
ALTER TABLE < nome_tabela > ADD / DROP (
nome_atributo1 < tipo > [NOT NULL],
nome_atributoN < tipo > [NOT NULL]) ;

Não existe nenhum comando SQL que permita eliminar algum atributo de uma relação já definida. Assim caso você desejar eliminar uma chave primária devidamente referenciada em outra tabela como chave estrangeira, ao invés de obter a eliminação do campo, obterá apenas um erro.

Além do comando DROP que poderá eliminar uma tabela e suas relações, também podemos criar uma relação que tenha os atributos que se deseja, copiar-se a relação antiga sobre a nova e apgando-se então a relação que originalmente desejávamos eliminar.

Ex:
ALTER TABLE DEPT (
ADD DEPSALA DECIMAL (10,2));

Exercício: Criar o Banco de Dados Mundo. Observar que se um continente for excluído, todos os países contidos em tal continente também o serão. Esta situação é conhecida como exclusão em Cascata. Observar também que a exclusão de um País eliminará todas as Cidades contidas no mesmo.

Prática

O Exemplo Trabalho já possue pequeno programa destinado a construção das tabelas contidas no Banco de Dados TRABALHO. Execute "trabalho.sql" de forma a obter as tabelas acima sem necessidade de digitar as instruções SQL de maneira interativa.

Para tanto, você deverá copiar para seu diretório de trabalho o arquivo "trabalho.sql" do diretório \IDEO\SQL.

Execute: "@trabalho;" que deverá:

- Criar o banco de dados Trabalho.
- Abrir o banco de dados Trabalho.
- Criar as Tabelas, Indices e Relações contidas neste Banco de Dados.

Posteriormente execute o comando "show tables", que deverá exibir as tabelas "dept" e "emp".
E ao executar "show fields dept" serão exibidos os campos da tabela "dept".

Copie e execute enchetra.sql do diretório \IDEO\SQL de forma a obter um conjunto de dados preparados para os testes a seguir apresentados.

Na próxima etapa de nosso curso, estaremos realizando pesquisas utilizando a instrução Select. Julgamos conveniente que os estudantes populem seu exercício e realizem exercícios análogos aos apresentados na Base de Dados Trabalho no Banco de Dados Mundo.

Parte II - Comandos de Consulta ao Esquema

Devemos ressaltar que a linguagem SQL é utilizada tanto pelos profissionais responsáveis pelos dados, onde é ressaltada a figura do Administrador do Banco de Dados e dos Analistas de Dados, como também pelos desenvolvedores de Aplicações. Enquanto àqueles estão preocupados com o desempenho, integridade do Banco de Dados e utilizam toda gama de recusos disponíveis no SQL, estes estão preocupados apenas em "transformar dados em informações", portanto para os desenvolvedores costuma-se dizer que conhecer o "select" já basta. Em nosso curso enfatizaremos a importância de TODOS os comandos do SQL, mas sabemos de antemão que os professores responsáveis pelas linguagens IDEO, VB e Delphi, ressaltarão a preponderância da instrução "select", que será apresentada a seguir e não no final do curso de SQL como geralmente acontece, pelo fato de que diversas disciplinas necessitam especificamente deste comando, que passaremos a apresentar:

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

Resp:

```
SELECT  *  FROM  DEPT;
```

O exemplo utiliza o coringa "*" para selecionar as colunas na ordem em que foram criadas. A instrução *Select*, como pudemos observar seleciona um grupo de registros de uma (ou mais) tabela(s). No caso a instrução *From* nos indica a necessidade de pesquisarmos tais dados apenas na tabela Dept.

Where como base das Restrição de tuplas.

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

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

Operadores lógicos

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

Exemplos:

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

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

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

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

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

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

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

Resp: Neste exemplo deveremos denominar colunas por apelidos. Os nomes das colunas mostradas por uma consulta, são geralmente os nomes existentes no Dicionário de Dado, 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) Apresente todos os salários existentes na empresa, porém omita eventuais duplicidades.

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

```
SELECT DISTINCT EMPSERV
   FROM EMP;
```

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

Resp: Desejamos um tratamento diferenciado para valores nulos. Qualquer coluna de uma tupla que não contenha informações é denominada de nula, portanto informação não existente. Isto não é o mesmo que "zero", pois zero é um número como outro qualquer, enquanto que um valor nulo utiliza um "byte" de armazenagem interna 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.

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

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

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

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

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

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

Resp:

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

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

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

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

Demais Operadores

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

Exemplos:

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

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

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

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

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

Operadores Negativos

operador	descrição
<>	diferente
not nome_coluna =	diferente da coluna
not nome_coluna >	não maior que
not between	não entre dois valores informados
not in	não existente numa dada lista de valores
not like	diferente do padrao de caracteres informado
is not null	não é um valor nulo

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

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

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

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

Resp: Necessitaremos de consultas com condições múltiplas.

Operadores "AND" (E) e "OR" (OU).

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

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

Resp:

```
SELECT EMPNOME, EMPSALA, EMPSERV
   FROM  EMP
```

```
    WHERE EMPSALA  BETWEEN  700 AND 2000
            OR  EMPSERV =  'VENDEDOR';
```

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

Resp:

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

Funções de Caracteres

Lower -	força caracteres maiúsculos aparecerem em minúsculos.
Upper -	força caracteres minúsculos aparecerem em maiúsculos.
Concat(x,y)-	concatena a string "x" com a string "y".
Substring(x,y,str)-	extrai um substring da string "str", começando em "x", e termina em "y".
To_Char(num)-	converte um valor numérico para uma string de caracteres.
To_Date(char,fmt)-	converte uma string caracter em uma data.
^Q -	converte data para o formato apresentado.

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

Resp:

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

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

Resp:

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

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

Resp:

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

ou

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

Funções Agregadas (ou de Agrupamento)

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

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

Resp:

```
SELECT  AVG(EMPSALA)  FROM  EMP;

SELECT  MIN(EMPSALA)  FROM  EMP;

SELECT  MAX(EMPSALA)  FROM EMP;

SELECT  SUM(EMPSALA) FROM  EMP;
```

Agrupamentos

As funções de grupo operam sobre grupos de tuplas(linhas). Retornam resultados baseados em grupos de tuplas em vez de resultados de funções por tupla individual. A claúsula "group by" do comando "select" é utilizada para dividir tuplas em grupos menores.

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

16) Apresente a média de salário pagos por departamento.

Resp:

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

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

Having

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

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

Resp:

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

Obs.: A 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.

Exemplificando ERRO típico - Restringindo Média Maior que 1000:

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

(Esta seleção está ERRADA!)

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

(Seleção Adequada)

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 "sql" fará a seguinte avaliação:

a) WHERE, para estabelecer tuplas individuais candidatas (não pode conter funções de grupo)
b) GROUP BY, para fixar grupos.
c) HAVING, para selecionar grupos para exibiçao.

Equi-Junção (Junção por igualdade)

O relacionamento existente entre tabelas é chamado de equi-junção, pois os valores de colunas das duas tabelas são iguais. A Equi-junção é possível apenas quando tivermos definido de forma adequada a chave estrangeira de uma tabela e sua referência a chave primária da tabela precedente. Apesar de admitir-se em alguns casos, a equi-junção de tabelas, sem a correspondência Chave Primária-Chave Estrangeira, recomendamos fortemente ao estudante não utilizar este tipo de construção, pois certamente em nenhum

momento nos exemplos propostos em nossa disciplina ou nas disciplinas de Análise e Projeto de Sistemas, serão necessárias tais junções.

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

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

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

Obs.: Note que as tabelas quando contém colunas com o mesmo nome, usa-se um apelido "alias" para substituir o nome da tabela associado a 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 aliás se encarregaria de evitar que uma ambiqüidade fosse verificada. Embora SQL resolva de forma muito elegante o problema da nomenclatura idêntica para campos de tabelas, recomendamos que o estudante fortemente evite tal forma de nomear os campos. O SQL nunca confundirá um A.NOME com um B.NOME, porém podemos afirmar o mesmo de nós mesmos?

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

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

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

As Sub-Consultas

Uma sub-consulta é um comando "select" que é aninhado dentro de outro "select" e que devolve resultados intermediários.

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

Resp:

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

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

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

Resp:

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

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

Uniões

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

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

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

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

Inserções, Alterações e Exclusões

Uma linguagem direcionada a extração de informações de um conjunto de dados, em tese não deveria incorporar comandos de manipulação dos dados. Devemos observar contudo que a mera existência de uma linguagem padronizada para acesso aos dados "convidava" os desenvolvedores a aderirem a uma linguagem "padrão" de manipulação de tabelas. Naturalmente cada desenvolvedor coloca "um algo mais" em seu SQL (SQL PLUS, SQL *, ISQL, e toda sorte de nomenclaturas), por um lado desvirtuando os objetivos da linguagem (padronização absoluta), mas em contrapartida otimiza os acessos ao seu banco de dados e por maior que sejam estas mudanças, jamais são tão importantes que impeçam que um programador versado em SQL tenha grandes dificuldades em se adaptar ao padrão de determinada implementação. De fato as diferenças entre o SQL da Sybase, Oracle, Microsoft, são muito menores dos que as existentes entre o C, o BASIC e o Pascal, que são chamadas de linguagens "irmãs", pois todas originam-se conceitualmente no FORTRAN. Podemos observar que todas as três linguagens mencionadas possuem estruturas de controle tipo "para" (for), "enquanto" (while) e repita (do..while, repeat..until). Todas trabalham com blocos de instrução, todas tem regras semelhantes para declaração de variáveis e todas usam comandos de tomada decisão baseadas em instruções do tipo "se" ou "caso", porém apesar de tantas semelhanças (sic), é praticamente impossível que um programador excelente em uma linguagem consiga rapidamente ser excelente em outra linguagem do grupo. Poderíamos arriscar a dizer que um excelente programador C que utilize a implementação da Symantech terá que passar por um breve período de adaptação para adaptar-se ao C da Microsoft.

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

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

Inserir (Insert)

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

Ex:
INSERT INTO DEPT;
Possibilita a inserção de registros de forma interativa.
INSERT INTO DEPT (DEPNUME,DEPNOME,DEPLOCA) VALUES (70,"PRODUCAO","RIO DE JANEIRO");
Possibilita a inserção de registros em tabelas sem digitação dos dados.

Atualizar (Update)

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

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

Excluir (Delete)

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

Ex:
```
 DELETE FROM emp WHERE EMPSALA > 5000;
```

Transações

Muitas vezes gostaríamos que determinado processo, caso fosse abortado por qualquer motivo, pudesse ser inteiramente cancelado. Imaginemos por exemplo um usuário digitando um pedido. Imaginemos ainda que o sistema possa reservar cada item solicitado de maneira "on line", ou seja ao mesmo tempo em que estou digitando a quantidade o sistema já "empenhe" uma quantidade equivalente no estoque. Imaginemos ainda que o sistema deve cancelar todas as operações se apenas um dos itens não puder ser atendido. Grande problema, caso não pudéssemos anular todos os processos a partir de determinada condição.

Vamos simular tal ocorrência com nosso banco de dados EMP. Imaginemos que ao invés de digitarmos DELETE FROM emp WHERE salario > 5000; tivéssemos digitado DELETE FROM emp WHERE salario > 500; Ao invés de eliminarmos 2 registros, praticamente teríamos eliminado o banco de dados todo. Para evitarmos que um erro de digitação, ou um processo iniciado porém sem condição de ser completado integralmente comprometa todos nossos dados podemos criar uma transação que nos assegurará que nossos testes sejam bem sucedidos ou cancelados sem comprometer nossos dados.

```
begin transaction;
  delete from emp where salario > 500;
  if SQL_RECORDCOUNT > 20 THEN;
     ROLLBACK TRASACTION;
  else
     COMMIT;
  endif;
end transaction;
```

Visões

Uma visão consiste basicamente de uma tabela derivada de outras tabelas. Considerando o exemplo TRABALHO, poderíamos criar uma visão baseada na Tabela de Empregados (EMP) e na Tabela de Departamentos (DEPT) onde tivéssemos somente os Nomes dos Funcionários e os Departamenos nos quais estes trabalhassem. Teríamos algo assemelhado ao abaixo representado

```
CREATE VIEW EMP_DEP
AS SELECT E.EMPNOME, D.DEPNOME
     FROM EMP E, DEPT D
     WHERE E.DEPNUME = D.DEPNUME;
```

Devemos observar que:

1- Uma visão definida sobre uma única tabela somente será atualizável se os atributos da tal visão contiverem a chave primária de tal tabela.
2- Visões sobre várias tabelas não são passíveis de atualizações.

3- Visões que se utilizam de funções de agrupamentos, também não poderão ser atualizadas.

PARTE III - Relatórios

Comando:

```
REPORT DISTINCT / UNIQUE
   [ atributo(s) ]
   REPORTTOP
   PAGETOP
   TOP
   DETAIL
   NONE
   BOTTOM
   PAGEBOTTOM
   REPORTBOTTOM
 FROM [ tabela(s) ]
 [ WHERE clausula-where ]
 [ GROUP BY clausula-grupo ]
 [ ORDER BY clausula-order by ];
```

Como exemplo converteremos um simples Select em um Report, temos:

```
 SELECT EMPNOME
   FROM EMP
   WHERE  DEPNUME = 1000;

 REPORT
   DETAIL EMPNOME
   WHERE DEPNUME = 1000;
```

Podemos direcionar a saida de um relatório tanto para um arquivo como para uma impressora.

Para um arquivo:

```
  REPORT ON “RELAT.DAT” ...
```

Para uma impressora:

```
  REPORT ON LP:” ...
```

Agora incrementando um report temos:

```
 REPORT
 REPORTTOP COL 10, “*** RELATORIO DE FUNCIONARIOS *** “,
                TODAY  %Q”DD/MM/YY”, SKIP,
                COL 10, “================================“, SKIP 2

 DETAIL  COL 10, NOME %C22, SALARIO %FS, ADMISSAO %Q”DD/MM/YY”
 REPORTBOTTOM COL 10,
                        “================================“, SKIP,
                 COL 20, “TOTAL:”, TOTAL(SALARIO)
```

```
  FROM EMP
   ORDER BY NOME;
```

Onde:

REPORTTOP - O que sera impresso no topo do relatório.
PAGETOP - Impresso em cada topo de pagina.
TOP - Impresso em cada Topo do Sort-Grupo do relatório.
DETAIL - O que sera impresso em cada linha.
NONE - Se não tiver resultado o select, não sera impresso o relatório.
BOTTOM - Impresso em cada Bottom do Sort-Grupo do relatório
PAGEBOTTOM - O que sera impresso no rodapé de cada pagina.
REPORTBOTTOM - O que sera impresso no rodape do relatório.

Formatos:

%C - caracter
%D - data
  y - ano,
  n - mes numérico,
  a - mes alfanumérico,
  d - dia,
  j - dia e ano juliano
Exemplo: %D"dd/mm/yy"

%I - inteiro
%F - ponto flutuante

%FSZ onde: S - separador de 3 digitos e decimal point
Z - zeros serão suprimidos

%Q - data
%J - Hora
  h - hora, m - minutos, s - segundos

%T - hora

E temos as funções: TOTAL, AVERAGE, MAXIMUM, MINIMUM.