Formação e Novas Tecnologias

Quality & Wisdom
Qualidade e Sabedoria

manual NHK

Computer Associates

CIW CERTIFIED INTERNET WEBMASTER

Cisco Systems Authorized Reseller

CertQuest.com

prosofttraining

Microsoft CERTIFIED Partner

Brainbench the measure of achievement Distribuition Partners

VUE

intel

hp invent COMPAQ

CompTIA

Adobe Training Provider

CTP Authorized Training Provider

Transcender APPROVED TRAINING PARTNER

Self Test SOFTWARE

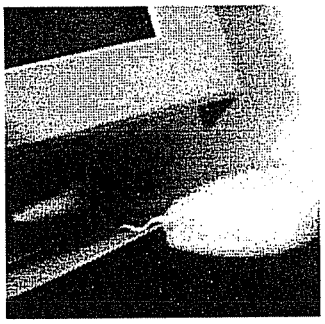
Authorized PROMETRIC TESTING CENTER

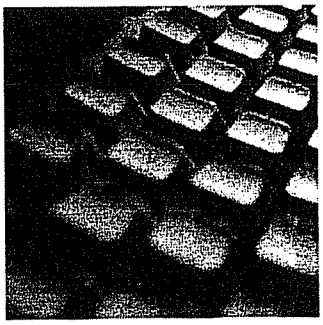
symantec.

autodesk Authorining center training

COREL TRAINING PARTNER

Microsoft SQL 2000





Curso:	MS SQL Server 2000
Nome do Manual:	MI01-0308 - Manual de MS SQL Server 2000
Versão:	1
Data:	12/1/2004
Elaborado Por:	Sérgio Santos
Verificado Por:	Sérgio Gouveia

Empresa:	NHK - Formação e Novas Tecnologias, Unipessoal, Lda
Morada:	Av. Duque D' Ávila , 72 B
	1050 - 083 Lisboa
Telefone:	213 301 888
Fax:	213 301 886
Internet:	www.nhk.pt
Email:	cursos@nhk.pt

Presidente:	Dr. Karen Bolbol
Dir Qualidade:	Dra Fátima Gil
Dir Pedagógica	Dra Eva Rosa
Dir Tec. Informação	Sérgio Gouveia

ATENÇÃO:

Este manual destina-se a apoiar os formados da NHK no curso.

Não pretendendo ser um manual exaustivo do curso em questão, apresenta-se como uma ferramenta de consulta quer durante a duração do curso, quer após a conclusão do curso.

O manual foi criado para satisfazer os requisitos das formações da NHK. Contudo poderá não corresponder na integra ao conteúdo programático do curso, sendo que será possível que alguns temas aqui abordados não façam parte do conteúdo programático do curso, assim como poderá acontecer que durante o curso sejam dados alguns temas que não se encontram neste manual.

A NHK reserva o direito de alterar o conteúdo do curso ou do manual de forma a satisfazer a sua oferta de cursos.

Índice:

BASES DE DADOS

Todos nós sabemos que existem gigantescas bases de dados a gerir as nossas vidas. De facto sabemos que a nossa conta bancária faz parte de uma colecção imensa de contas bancárias do nosso banco. O nosso título eleitoral ou o nosso cadastro de pessoa física, certamente estão armazenados em Bases de dados colossais. Sabemos também que quando levantamos dinheiro, na Caixa Multibanco, o nosso saldo e as movimentações existentes na nossa conta bancária já estão á nossa disposição.

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

Estas bases de dados, além de manterem todo este volume de dados organizado, também devem permitir actualizações, inclusões e exclusões do volume de dados, sem nunca perder a consistência. E não podemos esquecer que na maioria das vezes estamos a lidar com acessos concorrentes a várias tabelas das nossas bases de dados, algumas vezes com mais de um acesso ao registo de uma mesma tabela!

Uma base de dados é antes de mais nada, uma colecção lógica e coerente de dados com um determinado significado intrínseco. Por outras palavras, é um arquivo que contém uma série de dados de um cliente, um arquivo com dados aleatoriamente gerados, não pode ser considerada uma base de dados real.

Uma base de dados contém os dados dispostos numa ordem predeterminada em função de um projecto de sistema, sempre para um objectivo muito bem definido.

Uma base de dados representará sempre aspectos do Mundo Real. Assim sendo, uma base de dados(ou BD) é uma fonte de onde podemos extrair uma vasta game de informações derivadas, que possui um nível de interacção com eventos como o Mundo Real que representa. A forma mais comum da interacção entre utilizador e uma base de dados, dá-se através de sistemas específicos que por sua vez acedam ao volume de informações, geralmente através da linguagem SQL.

Os administradores de bases de dados (DBA) são responsáveis pelo controle de acesso aos dados e pela coordenação da utilização da BD. Já os projectistas de bases de dados(DBP) são analistas que identificam os dados a serem armazenados numa base de dados e pela forma como estes serão representados.

Os analistas e os programadores de desenvolvimento, criam sistemas que acedam aos dados da forma necessária ao Utilizador final, que é aquele que interage directamente com a base de dados.

SGBD x GA

Um SGBD – Sistema de Gestão de Bases de Dados – é uma colecção de programas que permitem ao utilizador definir, construir e manipular Bases de Dados para as mais diversas finalidades.

Um conceito que deverá fica bastante claro inicialmente é o que envolve a separação clara entre os Gestores de Bases de Dados dos Gestores de Arquivo.

Sistemas baseados em "Bases de Dados", com base 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 o padrão dBase) e o VB(que utiliza o Access), recomendam a utilização de Bases de Dados reais, porém utilizam aquelas "Bases de Dados" que possuem algumas características de Bases de Dados, mas possuem características típicas de Gestores de Arquivo.

Vamos definir algumas regras básicas e clara 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 novo "candidato" a SGBD, este poderá ser um GA (Gestor 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. Num 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 registo no seu programa, você não está a lidar com um SGBD.

Regra 2: Independência dos Dados - Quando as aplicações estiverem realmente imunes a mudanças na estrutura de armazenamento ou na estratégiade 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 de alterar o código do seu aplicativo, você não está a lidar com um SGBD.

Regra 3: Abstracção dos dados - Num SGBD real é fornecida ao utilizador somente uma representação conceptual dos dados, o que não inclui maiores detalhes sobre a sua forma de armazenamento real. O chamado Modelo de dados é um tipo de abstracção utilizada para fornecer esta representação conceptual. Neste modelo, um esquema das tabelas, os seus relacionamentos e suas chaves de acesso são exibidas ao utilizador, 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 num cliente inexistente e esta entrada não for automaticamente rejeitada, você não está a lidar com um SGBD.

Regra 4: Visões - Um SGBD deve permitir que cada utilizador visualize os dados de forma diferente daquela existente previamente na Base de Dados. Uma visão consiste num subconjunto de dados da Base de Dados, porém estes não deverão estar explicitamente armazenados. Portanto, toda a vez que você é obrigado a replicar uma estrutura, para fins de acesso de forma diferenciada por outros aplicativos, você não está a lidar com um SGBD.

Regra 5: Transacções - Um SGBD deve gerir completamente a integridade referencial definida em seu esquema, sem precisar em tempo algum, do auxilio do programa aplicativo. Desta forma exige-se que a base de dados, tenha ao menos uma instrução que permita a gravação de uma série de modificações simultâneas e uma instrução capaz de cancelar uma série de modificações. Por exemplo, imaginemos que estamos a cadastrar um pedido para um cliente, que este deseje reservar 5 itens do nosso stock, que estão disponíveis e portanto são reservados, porém existe um bloqueio financeiro(duplicatas em atraso) que impede a venda. A transacção deverá ser desfeita com apenas uma instrução á base de dados, sem quaisquer modificações suplementares nos dados. Caso você se obrigue a corrigir as reservas, através de acessos complementares, você não está a lidar 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 sempre que um utilizador, o primeiro utilizador bloqueou um registo numa tabela e o seu próximo passo será bloquear um registo numa tabela relacionada com a primeira, porém se este registo estiver previamente bloqueado por outro utilizador, o primeiro utilizador ficará paralisado, pois, estará à espera que o ségundo utilizador liberte o registo em uso, para que os outros o possam bloquear e prosseguir as suas tarefas. Se por hipótese o segundo utilizador necessitar de bloquear o registo bloqueado pelo primeiro utilizador, afirmamos que ocorreu um Dead-Lock, pois cada utilizador bloqueou um registo e precisa de bloquear outro! Imaginemos um caso onde o responsável pelos pedidos acabou de bloquear o Registo Item de Pedido, e necessita bloquear um registo no Cadastro de Produtos, para indicar uma nova reserva. Se a responsabilidade de evitar esta ocorrência for responsabilidade da aplicação, você não está a lidar com um SGBD.

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

Considerações Finais:

Actualmente, existe uma tendência de mercado em se dizer que qualquer problema será resolvido, caso a empresa adquira uma Base de Dados. Naturalmente, em um ambiente com acesso constante á Base 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 uma Base 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 impulsionar os negócios da empresa, e no geral causaram mais frustração do que solução, a aquisição da Base de Dados, pode gerar o mesmo tipo de problema.

É fundamental que a empresa candidata a utilizar uma Base de Dados, normalize-se totalmente, pois soluções "quebra-galho", típicas do ambiente que dispõe de um Gestor de Arquivo, tendem a ser impossíveis num ambiente estruturado sobre a Base de Dados. Portanto, sob pena de se realizar um grande investimento, e não se colher fruto nenhum, é muito conveniente, que a empresa *antes* de adquirir uma Base de Dados, passe por um processo de adaptação, preferencialmente contando com pessoal especializado, geralmente consultores, que *não* tenham qualquer ligação com fabricantes de Bases de Dados.

CARACTERÍSTICAS GERAIS DE UM SGBD

Os SGBD têm sete características operacionais elementares sempre observadas, que passaremos a listar:

Característica 1: Controle de redundâncias - A redundância consiste no armazenamento de uma mesma informação em locais diferentes, provocando inconsistências. Num Banco de Dados as informações só se encontram armazenadas num 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 sobre controle da base de dados.

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

Característica 3: Controle de Acesso – O SGBD deve dispor de recursos que possibilitem seleccionar a autoridade de cada utilizador. Assim um utilizador poderá realizar qualquer tipo de acesso, outros poderão ler alguns dados e actualizar outros, e outros ainda poderão somente aceder a um conjunto restrito de dados para escrita e leitura.

Característica 4: Interface - Uma base 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 acedida por aplicações.

Característica 5: Esquematização - Uma Base de Dados deverá fornecer mecanismos que possibilitam a compreensão dos relacionamentos existentes entre as tabelas e da sua eventual manutenção.

Característica 6: Controle de Integridade – Uma base de Dados deverá impedir que aplicações ou acesso 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 encontrarmos Bases de Dados que não satisfaçam completamente todas as características acima, o que não o invalida como Base de Dados. Na prática podemos encontrar situações onde a primeira característica não seja importante, pois podemos ter a Base de Dados baseada totalmente num único servidor, e as redundâncias podem ser aceites 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 o 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 incumbir desta

característica, no caso de pequenas empresas , com reduzido número de pessoas na área operacional.

O Interface e a esquematização, são características sempre disponíveis, o que varia neste caso é a qualidade destes componentes, que vai desde o sofrível até ao estado da arte. É muito conveniente que esta característica seja muito boa numa Base de Dados, onde estiverem em actuação mais do que um Administrador de Bases de Dados e tivermos um número relativamente alto de sistemas desenvolvidos ou em desenvolvimento neste ambiente.

De facto, quanto maior o número de pessoas envolvidas no desenvolvimento de aplicações e gestão da Base de Dados, mais importante se tornam estas duas características, pois cada novo sistema desenvolvido precisará sempre de 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 com uma que tenha uma interface mais recursiva. A esquematização existente na Base de Dados é muito melhor do que aquela mantida em alguma pasta, que sempre está "um pouquinho" desactualizada.

O controle de Integridade, é outra característica sempre presente nas Bases de Dados, mas existem diferenças aquando da implementação desta característica. Assim, é comum encontrarmos Bases 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 em termos de 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 base 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 uma base 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 vencedor para a nossa instalação.

Isto ocorre pois, na prática, todo o utilizador deseja um tempo de resposta muito pequeno. O chamado "prazo de entrega" muito comum em Bases de Dados operando nos limites de sua capacidade, ou nos casos onde o Hardware está muito desactualizado, é fonte de inúmeros problemas para o pessoal de informática. Neste caso é melhor abrirmos mão de uma Interface Amigável, de um gestão Automática de Backups ou ainda de outras características que não julgamos fundamentais, para nos livrarmos do problema típico de ambiente extremamente comprometido , por má performance da base de dados.

A escolha da Base de dados da empresa, portanto é uma decisão muito delicada, na medida em que esta irá acarretar troca de aplicativos e troca de Hardware. Os investimentos directamente aplicados na Base de Dados costumam ser infinitamente menores do que aqueles a serem aplicados na empresa, visando a sua perfeita adequação ao novo SGBD. Esta decisão, sempre que possível, deve ser tomada por especialistas em Bases de Dados com profundos conhecimento de Análise de Sistemas, de Bases de Dados e de Software de gestão de Bases de Dados, de forma a evitar que a empresa escolha uma Base de Dados inadequada aos seus propósitos e que pouco

tempo depois, seja obrigada a perder todo o investimento realizado em Software e Hardware.

ARQUITECTURA DE UM SGBD

ESTRUTURA

Podemos dizer que a base de dados tem um nível interno, onde é descrita a estrutura de armazenamento físico dos dados, um Nível intermediário onde temos a descrição lógica e um Nível Externo onde são descritas as visões para grupos de utilizadores.

Não podemos deixar de lembrar ainda que a Base de Dados garante a Independência lógica e Física dos Dados, portanto podemos alterar o esquema conceptual dos Dados, sem alterar as visões dos utilizadores ou mesmo alterar os esquema interno, sem contudo alterar o seu esquema conceptual.

MODELOS DE DADOS

O Modelo de Dados é Basicamente um conjunto de conceitos utilizados para descrever uma Base de Dados. Não existe uma única forma de representação deste modelo, porém qualquer forma que permita a correcta compreensão das estruturas de dados compreendidas na Base de Dados, pode ser considerada adequada. Vamos descrever sucintamente este modelo, pois estes serão objectos de outras disciplinas.

Modelo Orientado ao Registo: São Modelos que representam esquematicamente as estruturas das tabelas de forma bastante próxima a existente fisicamente. Basicamente são apresentados os registos de cada tabela(inclusive os seus campos) e os 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 Objecto: São modelos que procuram representar as informações através dos conceitos típicos da Programação Orientada por Objectos, utilizando o conceito de Classes que irão conter os objectos. Citamos os Modelos O2 e o de representação de Objectos como exemplos típicos desta abordagem.

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

Por 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.

BANCO DE DADOS RELACIONAL

O Modelo de Dados relacional, representa os dados contidos num Banco de Dados através de relações. Estas relações contêm informações sobre as entidades representadas e os seus relacionamentos. O Modelo Relacional, é claramente baseado no conceito de matrizes, onde as chamadas linhas(das matrizes) seriam os registos e as colunas, das mesmas, seriam os campos. Os nomes das tabelas e dos campos são de fundamental importância para a nossa compreensão entre o que estamos a armazenar, onde o fazemos e qual a relação existente entre os dados armazenados.

Cada linha da nossa relação será chamada de *TUPLA* e cada coluna terá o nome de *ATRIBUTO*. O conjunto de valores passíveis de serem assumidos por um atributo, será intitulado de *DOMÍNINO*.

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

No nosso curso, voltado á construção prática das Bases de Dados, e não da sua construção teórica, apenas serão citados os aspectos básicos da construção teórica, de forma a facilitar ao estudante o relacionamento entre Análise de Sistemas e Bases de Dados.

O domínio consiste num grupo de valores atómicos a partir dos quais um ou mais atributos retiram os seus valores lógicos(reais). Assim sendo, Rio de Janeiro, Paraná e Pará são estados válidos para o Brasil, enquanto que Corrientes não é um estado válido(Pertence á Argentina e não ao Brasil).

O esquema de uma relação, nada mais é do que os campos(colunas) existentes numa tabela. Já que a instância da relação consiste no conjunto de valores que cada atributo assume num determinado instante. Portanto, os dados armazenados na Base 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 dos dados na Base, não deverá ter qualquer importância para as relações, no que diz respeito ao seu tratamento. Os atributos deverão ser atómicos, isto é, não pode haver mais divisões.

Vamos dar o nome de chave primária ao Atributo que definir um registo, de entre uma colecção de registos. Chave Secundária, serão as 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 que um atributo(Por exemplo um cadastro ordenado alfabeticamente por estado, cidade e nome do cliente, necessitaria de uma chave composta com estes três atributos). A Chave Estrangeira, será aquela que permitir a ligação lógica entre uma tabela(onde ela se encontra) com outra na qual essa chave é a chave primária.

Exemplo:

```
Cidade              Estado
  *CidCodi            *EstCodi
   CidNome             EstNome
   EstCodi(E)
```

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

SQL – STRUCTURED QUERY LANGUAGE

INTRODUÇÃO

Quando as Bases de Dados Relacionais estavam a ser desenvolvidas, 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 de 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 Bases de Dados Relacional.

A 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 á criação da Base de Dados, das tabelas que o compõe, além das relações existentes entre as tabelas. Como por 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 num ou mais registos de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML temos os comandos, Select, Insert, Update e Delete.

A linguagem SQL tem como grandes virtudes a sua capacidade de gerir índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo, registo a registo. Outra característica muito importante, disponível em SQL é a sua capacidade de construção de visões, que são formas de visualisarmos os dados na forma de listagens independentemente das tabelas e organização lógica dos dados.

Outra característica interessante na linguagem SQL é a capacidade que dispomos de cancelar uma série de actualizações ou de as gravarmos, depois de iniciarmos uma sequência de actualizações. Os comandos Commit e Rollback são responsáveis por estas facilidades.

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

PARTE I – COMANDOS E MODIFICAÇÕES DO ESQUEMA E CRIAÇÃO DO BANCO DE DADOS.

Comando Create

Este comando permite a criação de tabelas na base de dados ou mesmo da sua criação.

Sintaxe:

```
CREATE DATABASE < nome_db >;
```

Onde:

nome_db – indica o nome da Base de Dados a ser criada

Sintaxe:

```
CREATE TABLE < nome_tabela >
( nome_atributo1 < tipo > [ NOT NULL ],
  nome_atrinuto2 < 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 criada 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 num arquivo ou digite na linha de comando do ISQL.

CREATE DATABASE TRABALHO;

O comando acima criou numa Base de Dados, porém este na verdade não passa de uma abertura no directório, pois não conta com nenhuma tabela.

Agora criaremos as tabelas que estarão contidas na Base de Dados TRABALHO.

A Primeira Tabela será a de Departamentos (DEPT). Além dos campos esta tabela conterá também a sua chave primária, as suas chaves estrangeiras e também os seus índices. A segunda tabela será a de empregados (EMP), que também será criada.

Não devemos esquecer de se abrir primeiramente a Base de Dados. Diferentemente do que ocorre em alguns aplicativos, em SQL o facto de criarmos uma Base de Dados, não significa que a base recém criada já está preparada para utilização. A instrução a seguir, providencia a abertura da Base de Dados.

OPEN DATABASE TRABALHO;

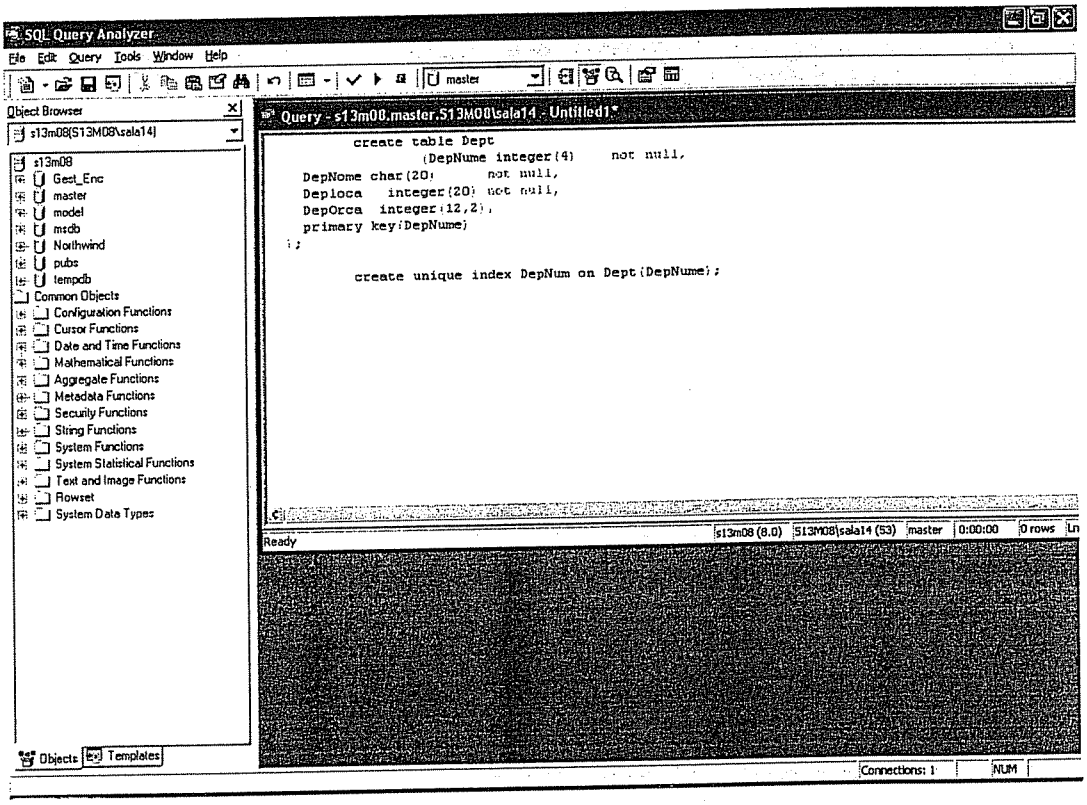
Agora estamos prontos para criarmos as tabelas necessárias. Lembramos aos Estudantes, que o Arquivo TABS.SQL, contém todas as instruções necessárias para criação da Base de Dados Trabalho e de suas tabelas. Já o arquivo DADOS.SQL irá popular estas tabelas. Para efeitos didácticos, criamos as tabelas 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 o seu gerente. Todos os "gerentes" têm nomes de cantoras brasileiras(Gal Costa, Marina Lima, etc), todos os "operários" têm nomes de jogadores de futebol, todas as vencedoras têm nomes de jogadoras de vólei, todas as balconistas têm nomes de jogadoras de basquete e o presidente da empresa exemplo, tem o nome do presidente do Brasil. Desta forma os testes devem resultar em grupos bastante definidos. Assim se se estiver a listar Gerentes e aparecer um homónimo da Ana Paula(Jogadora de vólei), verifique a sua query atentamente, pois muito provavelmente poderá estar errada.

A seguir está o código necessário à criação da tabela *Departamento* e o seu índice:

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

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

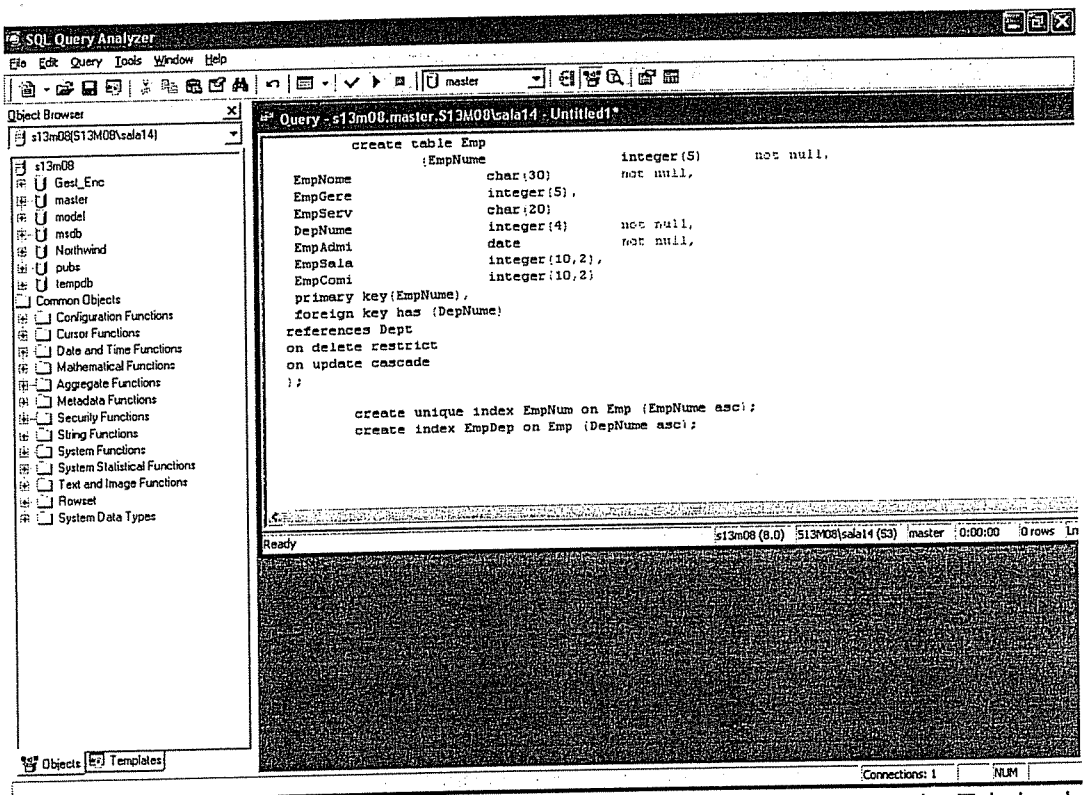


Note-se que a chave primária já está definida juntamente com o registo da tabela. A criação do índice, que por razões óbvias deve ser criado após a tabela, naturalmente é um comando totalmente independente do primeiro create, que serviu para criar a tabela e as suas características básicas.

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 directa àquela tabela. Quando declaramos que *DepNume* é uma chave estrangeira, promovemos de facto a ligação do cadastro de empregados, assim como o cadastro de departamentos. Ao restringirmos as exclusões, permitimos a existência de funcionários não alocados a nenhum departamento. Apesar desta prática ser contrária à tese de que devemos possuir apenas tuplas perfeitamente relacionáveis, nas nossas tabelas, podemos deixar esta pequena abertura, pois um utilizador que excluísse inadvertidamente determinado departamento , acabaria por excluir também uma grande quantidade de funcionários, que estivessem ligados a este departamento.

Já a actualização em cascata dos códigos de departamento é uma boa providência, na medida em que teremos, uma vez alterado algum código de departamento, a actualização 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 pela Base 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, que se tenha muito cuidado, aquando da elaboração da Base de Dados. Uma tentação muito comum ao estudante é começar a criar as tabelas da Base de Dados sem prévia Normalização. Este talvez seja o melhor caminho para perder-se tempo em vão, pois quando terminar de projectar as tabelas de entrada de dados, notará "que nada funciona". Esta será a palavra chave para usar o velho comando DEL do DOS e começar tudo novamente.

Comando Drop

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

Sintaxe:

```
DROP TABLE <nome_tabela>;
```

Ex:

```
DROP TABLE EMP;
```

Comado 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 deseje eliminar uma chave primária devidamente referenciada, noutra 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 as suas relações, também podemos criar uma relação que tenha os mesmos atributos que se deseja, copia-se a relação antiga sobre a nova e apaga-se a relação que originalmente se desejava eliminar.

Ex:

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

Exercício:

Criar a Base 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.

PARTE II – COMANDOS DE CONSULTA AO ESQUEMA

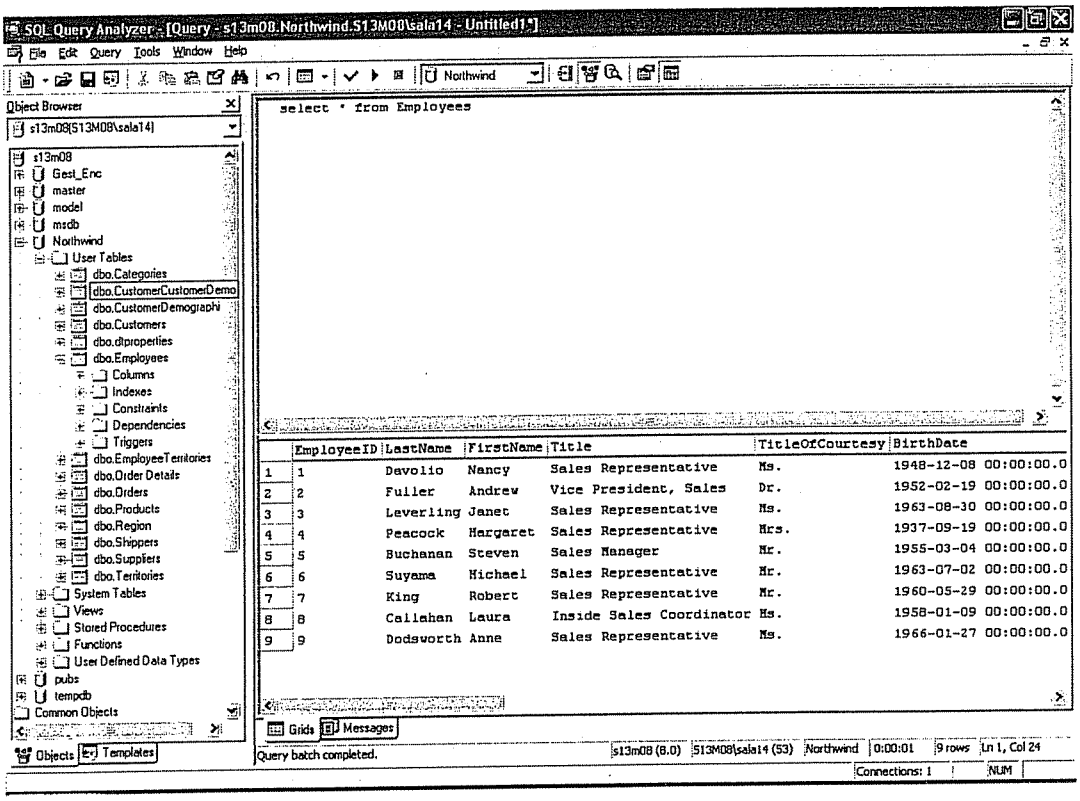
Devemos ressaltar que a linguagem SQL é utilizada tanto pelos profissionais responsáveis pelos dados, onde é ressaltada a figura do Administrador do Banco de Dados e dos Analistas de Dados, como também pelos programadores das Aplicações. Enquanto aqueles que estão preocupados com o desempenho, integridade da Base de Dados e utilizam toda a gama de recursos disponíveis no SQL, estes estão preocupados apenas em "transformar dados em informações", portanto para os programadores costuma-se dizer que conhecer o "select" já é suficiente. No nosso curso, enfatizaremos a importância de TODOS os comandos do SQL.

1) Selecção de todos os campos(ou colunas) da tabela de Employees (Empregados).

Resp;

```
SELECT * FROM employees;
```

O exemplo utiliza o asterisco "*" para seleccionar as colunas na ordem em que foram criadas. A instrução *Select*, como podemos observar, selecciona um grupo de registos de uma (ou mais) tabela(s). No caso, a instrução *From*, indica-nos a necessidade de pesquisar-mos tais dados apenas na tabela employees.



Where como base das Restrições de tuplas.

A cláusula "where" corresponde ao operador restrição da álgebra relacional. Contém a condição á qual 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 plicas (') na cláusula "where".

2) Seleccione todos os departamentos cujo orçamento mensal seja maior que 100000. Apresente o Nome de tal departamento e o ser 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 num valor. Esta expressão poderá conter nomes de colunas, valores numéricos, constantes e operadores aritmétricos.

```
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 todos sabem que CliCodi significa Código de Cliente. É possível (e provável) que o utilizador desconheça estes símbolos, portanto devemos apresentar, dando apelidos às colunas "contaminadas" pelo

informatiquês, que apesar de fundamental para os analistas, somente são vistos como enigmas para os utilizadores.

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

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

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 a sua existência física diferente da sua existência lógica (ou seja devidamente inicializado).

Resp: Desejamos um tratamento diferenciado para valores nulos. Qualquer coluna duma 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 de 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 EMPNOME DESC;
```

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

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

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

"Seleccione algo de algum lugar Onde se verificam tais relações"

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

Operadores

Operador	Significado
Between ... and ...	Entre dois valores (inclusive)
In (...)	Lista de valores
Like	Com um padrão de caracteres
Is null	É um valor nulo

Exemplos:

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

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


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


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

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

Operadores Negativos

Operador	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 padrão de caracteres informado
Is not null	Não é valor nulo

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

Resp: Aqui iremos precisar de utilizar expressões negativas. A seguir apresentaremos operadores negativos.

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

9) Apresente todos os funcionários entre 200 e 700 e que sejam vendedores.

Resp: Necessitaremos de consultas com condições múltiplas.

Operadores "AND" e "OR".

```
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

Função	Descrição
lower	Força caracteres maiúsculos a aparecerem como minúsculos
upper	Força caracteres minúsculos a aparecerem como maiúsculos
Concat(x,y)	Concatena a string "x" com a string "y"
Substring(x,y,str)	Extrai uma substring, da string "str", começando em "x" e terminando em "y"
To_Char(num)	Converte um valor numérico para uma string de caracteres.
To_Date(char,fmt)	Converte uma string caractere, numa 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-YYY"("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 algo, não nulo
Max(expr)	Maior valor da expr
Min(expr)	Menor valor da expr
Sum(n)	Soma dos valores de n, ignorando os nulos

15) Apresente a média, o Maior, o menor e também o somatório 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 baseados em grupos de tuplas em vez de resultados de funções por tupla individual. A Cláusula "group by" do comando "select" é utilizada para dividir tuplas em grupos menores.

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

16) Apresente a média de salários pagos por departamento.

Resp:

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

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

Having

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

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

Resp:

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

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

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

Dando um exemplo de um erro típico – Restringindo Média maior que 1000:

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

(**Esta selecção está errada**)

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

(**selecção correcta**)

Sequência no comando " Select":

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

A "SQL" fará a seguinte avaliação:

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

b) GROUP BY, para fixar grupos.

c) HAVING, para seleccionar grupos para exibição.

Equi-Junção(Junção por igualdade)

O relacionamento existente entre tabelas é chamado de equi-junção, pois os valores de colunas das duas tabelas são iguais. A equi-junção é possível apenas quando tivermos definido de forma adequada a chave estrangeira de uma tabela e a sua referência é a chave primária da tabela que a precede. Apesar de se admitir 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 momento algum nos exemplos propostos, na nossa disciplina de Análise e Projecto de Sistemas, serão necessárias tais junções.

18) Listar nome 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 aceder aos dados, restringindo convenientemente as relações existentes entre as tabelas. De facto sabemos que DEPNUME é uma chave primária da tabela de Departamentos e também é chave estrangeira da Tabela de Empregados. Portanto, este campo será o responsável pela equi-junção.

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

Obs: Note que as tabelas quando contêm colunas com o mesmo nome, usa-se um apelido "alias" para substituir o nome da tabela associado á coluna. Imagine que alguém tivesse definido NOME para ser o Nome do Empregado na Tabela de Departamentos. Tudo funcionaria de forma adequada, pois o "alias" encarregar-se-ia de evitar que essa ambiguidade fosse verificada.

Embora o SQL resolva de uma forma muito elegante o problema da nomenclatura idêntica para campos de tabelas, recomendamos que o estudante evite esse método de nomear os campos. O SQL nunca confundirá um A.NOME com um B.NOME, porém, será que podemos afirmar o mesmo de nós?

19) Liste os códigos de cada funcionário, os seus Nomes, os seus cargos e o nome do Gerente ao qual este se relaciona.

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

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

Inserções, Alterações e Exclusões

Uma linguagem direccionada á extracção de informações de um conjunto de dados, em tese não deveria incorporar comandos de manipulação de dados. Devemos observar contudo que a mera existência de uma linguagem padronizada, para acesso aos dados "convidava" os programadores a aderirem a uma linguagem "padrão" de manipulação de tabelas. Naturalmente cada programador coloca um extra no SQL em que trabalha(ex: SQL PLUS, SQL *, ISQL, e todas as outras nomenclaturas), por um lado desvirtuando os objectivos da linguagem(padronização absoluta), mas em contrapartida optimiza os acesso ao seu banco de dados e por maior que sejam estas mudanças, jamais são tão importantes que impeçam que um programador habituado em SQL tenha grandes dificuldades em adaptar-se ao padrão de determinada implementação. De facto as diferenças entre o SQL da Sybase, Oracle e 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 conceptualmente no FORTRAN. Podemos observar que todas as três linguagens mencionadas possuem estruturas de controle tipo "para" (for), "enquanto"(while) e repita(do...while, repeat .... until). Todas trabalham com blocos de instrução, todas têm regras semelhantes para declaração de variáveis e todas usam comandos de decisão baseados em instruções do tipo "se" ou "caso", porém apesar de tantas semelhanças (sic), é praticamente impossível que um programador excelente numa linguagem consiga rapidamente ser excelente noutra linguagem do grupo.

Poderíamos arriscar a dizer que um excelente programador em C que utilize a implementação da Symantec terá que passar por um breve período de adaptação para adaptar-se aos 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 ***aprendizado***, mas este programador poderá ir adaptando-se aos poucos sem precisar de ser reeducado, o que é um aspecto extremamente vantajoso para as empresas.

Inserir(Insert)

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

Exemplo:

INSERT INTO DEPT;

Possibilita a inserção de registos de forma interactiva.

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

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

Actualizar(Update)

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

Exemplo:

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

Excluir (Delete)

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

Exemplo:

DELETE FROM emp WHERE EMPSALA > 5000;

BIBLIOGRAFIA

Microsoft SQL SERVER 2000
Remata De Oliveira Leão
João Carlos da Silva
Editora Erica

Ajuda Online e do programa