Colégio La Salle São Paulo
Curso Técnico em Informática

Disciplina: **Banco de Dados**
Professor: **Marcelo Cabral Ghilardi**
Módulo: **III**
Ano Letivo: **2003**

VIII – Comandos SQL

A Linguagem SQL (Search and Query Language)

Em bancos de dados relacionais as informações são guardadas em entidades (tabelas).

Para recuperar uma informação necessária ao usuário, deve-se buscá-la em uma ou em várias tabelas diferentes, estabelecendo-se um relacionamento entre elas. Esta é a origem do nome deste paradigma de banco de dados.

As tabelas são conjuntos que utilizamos em teoria dos conjuntos, por exemplo, quando em um sistema existe uma tabela de vendas, esta tabela corresponde ao conjunto de todas as vendas feitas por uma empresa. A tabela de vendedores corresponde ao conjunto de vendedores que trabalham em uma empresa. Cada linha ou registro da tabela corresponde a um elemento do conjunto.

Consultas e alterações na base de dados correspondem a operações realizadas sobre os conjuntos (tabelas) existentes. Estas operações são definidas pela álgebra relacional. Por exemplo, determinar quais os vendedores com tempo de casa maior que um determinado patamar, significa determinar um subconjunto do conjunto de vendedores, onde todos os elementos possuam uma determinada propriedade.

Consultas em banco de dados não passam de problemas de álgebra relacional. Assim como acontece com a álgebra "tradicional", os operadores possuem algumas propriedades.

Sabemos que 2 x 3 = 3 x 2. Isto significa que, quando precisamos contar uma expressão de álgebra relacional para chegar a um determinado resultado, podemos faze-lo de mais de uma forma, pois várias expressões levam ao mesmo resultado. Em outras palavras, quando o banco de dados precisa montar uma expressão algébrica para encontrar um resultado, ele deve escolher uma entre várias. Apesar de apresentarem o mesmo resultado, as expressões são diferentes, e a diferença fará com que o banco de dados adote um diferente caminho para resolver cada uma expressão. Escolher o caminho mais curto é uma das grandes atribuições do banco de dados.

Está é a missão do otimizador, um subsistema do banco da dados, responsável por determinar o plano de execução para uma consulta.

A linguagem SQL (Search and Query Language) é um grande padrão de banco de dados, isto decorre da sua simplicidade e facilidade de uso. Ela se opõe a outras linguagens no sentido em que uma consulta SQL especifica a forma do resultado e não o caminho para chegar a ele.

Ela é um linguagem declarativa em oposição a outras linguagens procedurais. Isto reduz o ciclo de aprendizado daqueles que se iniciam na linguagem.

Vamos comparar:

Descrição declarativa:

"quero saber todas as vendas feitas por vendedores com mais de 10 anos de casa."

Descrição procedural:

"Para cada um dos vendedores, da tabela vendedores, com mais de 10 anos de casa, determine na tabela de vendas todas as vendas destes vendedores. A união de todas estas vendas será o resultado final do problema."

A declaração procedural tem muito de declarativa, o que a torna mais simples. Se ela fosse realmente procedural, seria ainda mais complicada.

O fato é, ter que informar o como fazer, torna as coisas mais difíceis. Neste sentido, SQL facilitou muito o trabalho dos desenvolvedores de banco de dados.

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 de cancelar uma série de atualizações ou de gravação, 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.

Comandos de criação

1. Criar banco de dados

Create database nome_do_banco_de_dados
Ex. create database teste

2. Criar tabelas

```
Create table nome_tabela (nome_da_coluna tipo_da_coluna[ ] ).....
        [Primary key (nome_da_coluna[,nome_da_coluna,,,,,])]
        [foreing key[nome_da_chave] (nome_da_coluna[,nome_da_coluna,....])
        references nome_da_tabela)

Ex1. create table jogo ( tipo_jogo character(15) not null,
                         local_jogo character(20) not null,
                        [primary key tipo_jogo]

Ex2. Create table aluno (  matricula smallint not null,
                           Nome_aluno character(40) not null,
                           Cod_curso character(02) not null,
                           Cod_campus character(02) not null,
```

Primary key (matricula),
Foreing key curso(cod_curso) reference cursos,
Foreing key campus(cod_campus) reference campi)

3. Criar um índice

Create index [unique] nome_do_indice
On nome_da_tabela(nome_da_coluna[,....])

Unique - indica que o índice não permitira valores repetidos para uma coluna específica ou combinação de colunas.

Ex. create unique index index1 on aluno(nome_aluno)

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.

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 *Selec*t, 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 CR$ 10.000,00. 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

Caracteres	*Significado*
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 treinado novamente, o

que é um aspecto extremamente vantajoso para as empresas.

Inserir (Insert)

INSERT INTO <tabela> [<campos>] [VALUES <valores>]
Ex:
INSERT INTO DEPT;
Possibilita a inserção de registros de forma interativa.
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.

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:

Comando	***Significado***
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:

Simbolo	***Significado***
%C	caracter
%D	data
y	ano
m	mês numérico,
a	mês alfanumérico
d	dia
j -	dia e ano juliano
%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

Exemplo: %D"dd/mm/yy"

RESUMO DOS COMANDOS SQL

1 Comandos DML:

Seleção: Select <colunas>
From <tabelas>
Where <condições>
Group by < colunas de agrupamento>
Having <condições sobre funções de grupo>
Order By <colunas para ordenação>

1. Selecionar uma coluna específica: identificar a coluna
2. Selecionar múltiplas colunas: identificar as colunas
3. Reordenar colunas: identificar as colunas, mudando ordem em relação a ordem de criação na tabela
4. Selecionar registro inteiro: *
5. Ordenando registros em seqüência: order by
6. Ordenando por múltiplos critérios: asc e desc
7. Selecionando um registro específico: where
8. Operadores Lógicos: = <> < > <= >=
9. Outros Operadores: IN, BETWEEN low AND high, LIKE (%, _), IS NULL, NOT
10. Múltiplas condições de consulta: precedência de operadores: (), AND, OR
11. Expressões: +, -, *, /
12. Expressões com data: data + dias = data, data - data = dias
13. Uso de alias para colunas e tabelas
14. Funções de grupo: sum, max, min, count
15. Funções de string: legth(), substr(), lower(), upper(), to_char(), to_date(), to_number()
16. Equi-Join, Self-Join, Outer-Join
17. Union, Intersect, Minus
18. Subqueries

Inserção de Regsitros:
Insert comum: Insert into <tabela> (<colunas>) values <valores das colunas>
Inser através de select: Insert into <tabela> (<colunas>) (select ...)

Atualização de Registros:
Update <tabela> set <coluna=valor> where <condições>

Remoção de Registros:
Delete from <tabela> where <condições>

Controle de Atualização: definir conceito de transação
Commit
Rollback
Savepoint X
Rollback to X

2 Comandos DDL:

1. Catálogo de tabelas do banco de dados: all/user
2. Tipos de dados disponíveis: char, varchar2, date, number
3. Create table <tabela> (<colunas>, <primary key (colunas chave)>, <foreign key (colunas chave) references <tabela referenciada> (<colunas chave da tabela referenciada>)
4. Alter table <tabela> add (<colunas>,
   <primary key <nome constraint > (<colunas>),
   <foreign key <nome constraint> (colunas) references <tabela referente> (<colunas ref.>),
   constraint <nome constraint> check (<coluna a ser validada> in (<valores validos>)))
5. Alter table <tabela> modify (<colunas>)
6. Alter table <tabela> drop constraint <nome constraint>
7. Drop table <tabela>
8. Create view <nome view> as <select>
9. Drop View <nome view>

Permissões de Acesso: grant/revoke

1. Usuário: create user <nome usr> identified by <senha>
2. Privilégios de sistema: connect, resource, dba, all => grant <privilégios> to <usuário> / revoke <privilegio> from usuario
3. Privilégios sobre tabelas: select, insert, update, delete, alter, index => grant <privilegios> on <tabela> to <usuário>
4. Sinônimos: create synonym <sinônimo> for <user>.<tabela>

Índices:

Create index <indice> on <tabela> (<colunas>)
Drop index <indice>

Triggers, Functions, Procedures, Packages

create or replace trigger <nome> after/before insert/update/delete on <tabela> ...
create or replace procedure <nome> (<parametro> in/out <tipo dado>) ...
create or replace function <nome> (<parametro> in/out <tipo par>) return <tipo retorno> is ..
create or replace package <nome> as ...