SQL – Structured Query Language
(Linguagem de Consulta Estruturada)

1. Criação de Tabelas.

Create Table

Este comando permite a criação de tabelas no banco de dados.

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, varchar(n), numeric(n,m), date...).

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

Exemplos:

```
CREATE TABLE DEPT
(
      DEPCODIGO          INTEGER          NOT NULL,
      DEPNOME            VARCHAR (20)      NOT NULL,
      DEPLOCAL           VARCHAR (20),
      DEPORCAMENTO       NUMERIC (12,2),
      PRIMARY KEY (DEPCODIGO)
);

CREATE TABLE EMPREGADO
(
      EMPNUMERO          INTEGER          NOT NULL,
      EMPNOME            VARCHAR (40)     NOT NULL,
      DEPCODIGO          INTEGER          NOT NULL,
      EMPSERVICO         VARCHAR (20),
      EMPGERENTE         INTEGER,
      EMPADMISSAO        DATE,
      EMPSALARIO         NUMERIC (9,2),
      EMPCOMISSAO        NUMERIC (3,2),
      PRIMARY KEY (EMPNUMERO),
      FOREIGN KEY  (DEPCODIGO) REFERENCES DEPT (DEPCODIGO)
);
```

2. Alteração na Estrutura da Tabela

Alter Table

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

Sintaxe:

```
ALTER  TABLE  < nome_tabela >  ADD / DROP
          nome_atributo          < tipo >        NOT NULL;
```

Exemplo:

```
ALTER TABLE DEPT  ADD
          DEPDIRETOR  VARCHAR (50);

ALTER TABLE DEPT  DROP
          DEPDIRETOR  VARCHAR (50);
```

3. Exclusão de Tabela

Drop Table

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

Sintaxe:

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

Exemplo:

```
DROP  TABLE   EMP;
```

4. Inclusão de Registros em Tabelas

Insert (Inserir)

Sintaxe:

```
INSERT INTO <tabela> (<campos>) VALUES (<valores>);
```

Exemplo:

```
INSERT INTO DEPT (DEPCODIGO, DEPNOME, DEPLOCAL, DEPORCAMENTO)
            VALUES (70,  'PESSOAL',  'BARRA BONITA',  10000.00);

INSERT INTO EMPREGADO (EMPNUMERO, EMPNOME, DEPCODIGO,
              EMPSERVICO, EMPADMISSAO, EMPSALARIO)
            VALUES (1, 'JOÃO', 70, 'OPERARIO', '10/01/2006', 900.90);
```

5. Alteração de Registros das Tabelas

Update (Atualização)

Sintaxe:

```
UPDATE <tabela> SET <campo> = <expressão> WHERE <condição>;
```

Exemplo:

```
UPDATE EMPREGADO SET EMPSALARIO = EMPSALARIO* 1.2
                     WHERE EMPSALARIO < 1000;
```

O exemplo altera os salários de todos os empregados que recebem menos de R$ 1000,00, aumentando os salários em 20%.

6. Exclusão de Registros das Tabelas

Delete (Excluir)

Sintaxe:

```
DELETE FROM <tabela> WHERE <condição>;
```

Exemplo:

```
DELETE FROM EMPREGADO WHERE EMPSALARIO > 5000;
```

O exemplo exclui todos os funcionários com Salário maior que R$ 5000.

7. Seleção de Registros. Comandos de Consulta aos Registros.

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

Resposta:

```
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 Registros.

A cláusula "where" corresponde ao operador restrição da álgebra relacional. Contém a condição que os registros devem obedecer a fim de serem listados. 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

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

Exemplos:

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

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

2) Selecione todos os departamentos cujo orçamento mensal seja maior que R$ 100.000,00. Apresente o Nome de tal departamento e seu orçamento anual, que será obtido multiplicando-se o orçamento mensal por 12.

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

```
SELECT  DEPNOME,  (DEPORCAMENTO * 12)  FROM  DEPT
   WHERE DEPORCAMENTO >= 10000;
```

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

Resposta: Neste exemplo deveremos denominar colunas por apelidos. Os nomes das colunas mostradas por uma consulta, são geralmente os nomes existentes no Dicionário de Dados, porém geralmente estão armazenados na forma do mais puro "informatiquês", onde "todo mundo" sabe que CliCodi significa Código do Cliente. É possível (e provável) que o usuário desconheça estes símbolos, portanto devemos os apresentar dando apelidos às colunas "contaminadas" pelo informatiquês, que apesar de fundamental para os analistas, somente são vistos como enigmas para os usuários.

```
SELECT  DEPNOME "DEPARTAMENTO", (DEPORCAMENTO * 12) "ORCAMENTO
ANUAL"  FROM  DEPT
        WHERE DEPORCAMENTO >= 10000;
```

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

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

```
SELECT DISTINCT EMPSERVICO FROM EMP;
```

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

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

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

SELECT  EMPNOME, EMPSERVICO FROM  EMP
     ORDER  BY  EMPNOME  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.

6) Selecione os Nomes dos Departamentos que estejam na fábrica, em Barra Bonita.

Resposta:

```
SELECT DEPNOME FROM DEPT
       WHERE DEPLOCAL = 'BARRA BONITA';
```

O exemplo exigiu uma restrição (Barra Bonita) que nos obrigou a utilizar a 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

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

Exemplos:

1. Selecione o Nome e o Salário dos funcionários cujo salário esteja entre R$ 500,00 e R$ 1500,00.

```
SELECT EMPNOME, EMPSALARIO FROM EMPREGADO
    WHERE EMPSALARIO BETWEEN 500 AND 1500;
```

2. Selecione o Nome e o Código do Departamento dos funcionários que trabalham nos Departamentos de Código: 1, 10, 30, 40 e 70.

```
SELECT EMPNOME, DEPCODIGO FROM EMPREGADO
    WHERE DEPCODIGO IN (1,10,30,40,70);
```

3. Selecione o Nome e o Serviço dos funcionários cujo nome comecem com "J".

```
SELECT EMPNOME, EMPSERVICO FROM EMPREGADO
    WHERE EMPNOME LIKE 'J%';
```

4. Selecione o Nome e o Serviço dos funcionários cuja Comissão seja Nula.

```
SELECT EMPNOME, EMPSERVICO FROM EMPREGADO
    WHERE EMPCOMISSAO IS NULL;
```

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

Operadores Negativos

<>	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

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

Resposta: Necessitaremos aqui a utilização de expressão negativa. A seguir apresentamos operadores negativos.

```
SELECT EMPNOME, EMPSALARIO FROM EMPREGADO
    WHERE EMPSALARIO NOT BETWEEN 1000 AND 3500;
```

8) Apresente todos os funcionários com salários entre 700 e 2000 e que sejam Operários.

Resposta: Necessitaremos de consultas com condições múltiplas.

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

```
SELECT EMPNOME, EMPSALARIO, EMPSERVICO FROM  EMPREGADO
      WHERE  EMPSALARIO  BETWEEN  700 AND 2000
         AND  EMPSERVICO =  'OPERARIO';
```

9) Apresente todos os funcionários com salários entre 700 e 2000 ou que sejam Operários.

Resposta:

```
SELECT EMPNOME, EMPSALARIO, EMPSERVICO FROM  EMPREGADO
      WHERE  EMPSALARIO  BETWEEN  700 AND 2000
          OR EMPSERVICO =  'OPERARIO';
```

10) Apresente todos os funcionários com salários entre 700 e 2000 e que sejam Operários ou Vendedores.

Resposta:

```
SELECT EMPNOME, EMPSALARIO, EMPSERVICO FROM  EMPREGADO
      WHERE  EMPSALARIO  BETWEEN  700 AND 2000
         AND (EMPSERVICO =  'OPERARIO' OR EMPSERVICO = 'VENDEDOR');
```

Funções Agregadas (ou de Agrupamento)

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

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

Resposta:

```
SELECT  AVG (EMPSALARIO)  FROM  EMPREGADO;

SELECT  MIN (EMPSALARIO)  FROM  EMPREGADO;

SELECT  MAX (EMPSALARIO)  FROM EMPREGADO;

SELECT  SUM (EMPSALARIO) FROM  EMPREGADO;
```

Agrupamentos

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

A cláusula "GROUP BY" pode ser usada para dividir os registros 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.

Resposta:

```
SELECT DEPCODIGO, AVG(EMPSALARIO) FROM EMPREGADO
    GROUP BY DEPCODIGO;
```

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.

Resposta:

```
SELECT DEPCODIGO, AVG(EMPSALARIO) FROM EMPREGADO
    GROUP BY DEPCODIGO
    HAVING COUNT(*) > 10;
```

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 DEPCODIGO AVG(EMPSALARIO)  FROM EMPREGADO
    WHERE  AVG(EMPSALARIO) > 1000
    GROUP BY DEPCODIGO;
```

(Esta seleção está ERRADA!)

```
SELECT  DEPCODIGO, AVG(EMPSALARIO) FROM EMPREGADO
    GROUP BY DEPCODIGO
    HAVING AVG(EMPSALARIO) > 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 registros individuais candidatos (não pode conter funções de grupo)
b) GROUP BY, para fixar grupos.
c) HAVING, para selecionar 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 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.

Resposta: Observemos que dois dos três dados solicitados estão na Tabela **Empregado**, 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 DEPCODIGO 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.EMPSERVICO,  B.DEPNOME
FROM  EMPREGADO A,  DEPT  B
        WHERE  A.DEPCODIGO = B.DEPCODIGO;
```

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.

Resposta: 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 registros da tabela a outro registro da mesma tabela.

```
SELECT  A.EMPNUMERO, A.EMPNOME, A.EMPSERVICO, B.EMPNOME
FROM  EMPREGADO A, EMPREGADO B
      WHERE  A.EMPGERENTE  = B.EMPNUMERO;
```

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 10000.

Resposta:

```
SELECT  EMPNOME, EMPSERVICO FROM EMPREGADO A
WHERE  10000 IN (SELECT  DEPORCAMENTO  FROM  DEPT
                      WHERE DEPT.DEPCODIGO = A.DEPCODIGO);
```

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.

Resposta:

```
SELECT DEPNOME FROM DEPT A
      WHERE EXISTS (SELECT * FROM EMPREGADO
      WHERE EMPSALARIO > 3500 AND EMPDEPCODIGO = A.DEPCODIGO);
```

Nota: Observe que a cláusula EXISTS indica se o resultado de uma pesquisa contém ou não registros. Observe também que poderemos verificar 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.

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

```
(SELECT * FROM EMP WHERE EMPNUME > 10)
UNION
(SELECT * FROM EMP WHERE DEPNUME > 10);
```