SQL - Structured Query Language

O que é SQL ?

SQL significa Structured Query Language. SQL é usado para a comunicação com base de dados. É um grupo de facilidade para definição, manipulação e controle de dados em um banco de dados relacional. De acordo com a ANSI (American National Standards Institute), esta é a linguagem padrão para a administração de base de dados relacional.

O que o SQL pode fazer ?

- ***DDL-Data Description Language***
  Exemplo : CREATE TABLE ……
- ***DML-Data Manipulation Language***
  Exemplos : INSERT/UPDATE/DELETE/SELECT
- ***DCL-Data Control Language***
  Exemplos : GRANT / REVOKE

Base de Dados Relacional

Uma base de dado relacional é uma coleção de tabelas contendo dados. Uma tabela consiste de colunas (atributos que descrevem a tabela) e linhas (a atual ocorrência de dados). Tais dados, podem ser acessados facilmente e rapidamente em uma base de dados relacional e visualizado em formato tabular.

Conectando-se ao BD

USUÁRIO

Um usuário é um nome definido no banco de dados que pode conectar-se e acessar objetos. Um *schema* é o nome dado para a coleção de objetos, como tables, views, clusters, procedures, e packages, associados a um usuário em particular.

Para acessar o banco de dados , o usuário deve executar uma aplicação de banco (como Oracle Forms SQL*Plus, etc...) e conectar-se usando o username definido no banco.

DML – Data Manipulation Language

Data Manipulation Language (DML)

Você usa comandos DML para manipular dados em tabelas. Existem quatro comandos DML básicos:

- SELECT
- INSERT
- UPDATE
- DELETE

SELECT

```
SELECT      [ALL | DISTINCT ]                    - Seleciona colunas
        {* | [[user.].table.column | expression,...]}
        FROM        table_name ,[table2]         - Define tabelas
        [WHERE]     condition                    - Restringe linhas
        [GROUP BY]  .....                        - Agrupamento
        [HAVING]    .....                        - Restringe linhas de grupo
        [ORDER BY]  .....                        - Ordena o resultado
```

Select *

Ex.: SELECT * FROM departamento;

COD_DEPT	NOME_DEPT
5200	MARKETING
4600	MANUTENÇÃO
5100	INFORMÁTICA

Select em uma coluna específica

Ex.: SELECT cod_dept FROM departamento;

COD_DEPT
5200
4600
5100

Alterando o cabeçalho da coluna

Ex.: SELECT cod_dept AS "Código do Departamento"
FROM departamento;

```
Código do Departamento
------------------------------
5200
4600
5100
```

Mostrando cálculos sobre colunas (1/2)

Você pode usar expressões aritméticas para calcular novos valores de uma coluna.

Ex.: SELECT cod_emp, salario, salario * 1.30 AS "Novo Salário"
FROM empregado;

COD_EMP	SALARIO	Novo Salário
5200	1200	1560
4600	1500	1950
5100	5000	6500

3 rows selected.

Mostrando cálculos sobre colunas (2/2)

Use os seguinte símbolos para operações aritméticas:

Símbolo	Significado
*	Multiplicação
/	Divisão
+	Adição
-	Subtração

Atenção: o resultado de qualquer coluna envolvendo um campo NULL é sempre NULL.

Eliminando linhas duplicadas

Algumas vezes o resultado de um comando SELECT pode retornar linhas idênticas. Para eliminar a duplicidade no retorno da consulta (QUERY), adicione o DISTINCT após o SELECT.

```
SELECT cod_dept
    FROM empregado;

COD_DEPT
--------------
        5200
        4600
        5100
        5100

 4 rows selected.
```

```
SELECT DISTINCT cod_dept
   FROM empregado;

COD_DEPT
--------------
        5200
        4600
        5100


 3 rows selected.
```

Organizando dados (ORDER BY) 1/3

Se você quiser trazer o resultado de uma consulta em uma ordem específica, use a cláusula ORDER BY. A Cláusula ORDER BY deve ser o último comando do SELECT.

Ex.: SELECT *
FROM empregado
ORDER BY nome_emp;

COD_EMP	NOME_EMP	COD_DEPT	SALARIO	TELEFONE
2437	CARLOS	4600	1500	312-3232
2096	JOÃO	5200	1200	232-1232
2899	MARIA	5100	6000	
2898	PEDRO	5100	5000	323-3232

Organizando dados (ORDER BY) 2/3

Você pode especificar ainda se a ordem será ascendente ou descendente. Para isto adicione as cláusulas ASC ou DESC após a coluna especificada no ORDER BY. O DEFAULT é ASC.

Ex.: SELECT *
FROM empregado
ORDER BY nome_emp DESC;

COD_EMP	NOME_EMP	COD_DEPT	SALARIO
2898	PEDRO	5100	5000
2899	MARIA	5100	6000
2096	JOÃO	5200	1200
2437	CARLOS	4600	1500

Organizando dados (ORDER BY) 3/3

Você pode especificar mais colunas na cláusula ORDER BY. Para isto, adicione uma virgula e a segunda coluna desejada. Esta segunda coluna determinará a ordem caso os valores das primeira coluna sejam iguais.

Ex.: SELECT *
FROM empregado
ORDER BY cod_dept, nome_emp desc;

COD_EMP	NOME_EMP	COD_DEPT	SALARIO
2437	CARLOS	4600	1500
2096	JOÃO	5200	1200
2898	PEDRO	5100	5000
2899	MARIA	5100	6000

Identificando colunas pelo número

Ex.:

```
                1          2          3
SELECT cod_emp, nome_emp, cod_dept
  FROM empregado
 ORDER BY 2;

SELECT cod_emp, cod_dept, salario * 1.30
   FROM empregado
  ORDER BY  3;
```

ORDER BY - Outras formas

```
SELECT cod_emp AS "codigo"
   FROM empregado
 ORDER BY "codigo";
```

```
SELECT cod_emp codigo
  FROM empregado
ORDER BY codigo;
```

```
SELECT cod_emp AS "codigo"
  FROM empregado
ORDER BY cod_emp;
```

```
SELECT cod_emp
   FROM empregado
 ORDER BY salario;
```

Usando condições de busca (WHERE) 1/5

Sintaxe: WHERE condição

Operador	Significado
=	Igual
<>	Diferente
>	Maior que
<	Menor que
>=	Maior ou Igual
<=	Menor ou igual

Palavra	Significado
IS NULL	Verifica se o valor é nulo (NULL)
BETWEEN	Verifica o valor dentro de um intervalo
IN	Verifica o valor dentro de uma lista
LIKE	Verifica colunas baseado em uma combinação de caracteres

Usando condições de busca (WHERE) 2/5

```
1) SELECT nome_emp, salario
       FROM empregados
    WHERE cod_dept = 5100;

2) SELECT nome_emp,salario
       FROM empregados
    WHERE salario > 5000;

3) SELECT nome_emp,salario
       FROM empregados
    WHERE salario >= 5000;

4) SELECT nome_emp,salario
       FROM empregados
    WHERE salario <= 5000;
```

Usando condições de busca (WHERE) 3/5

5) SELECT nome_emp, salario
FROM empregados
WHERE NOT salario = 5000 (ou salario <> 5000);

6) SELECT nome_emp, salario
FROM empregados
WHERE salario IS NULL;

7) SELECT nome_emp, salario
FROM empregados
WHERE salario IS NOT NULL;

8) SELECT nome_emp, salario
FROM empregados
WHERE salario BETWEEN 5000 AND 6000;

Usando condições de busca (WHERE) 4/5

```
9) SELECT nome_emp, salario
       FROM empregados
    WHERE salario NOT BETWEEN 5000 and 6000;

10) SELECT nome_emp, salario
        FROM empregados
      WHERE salario IN (1200, 1500);

11) SELECT nome_emp
        FROM empregados
      WHERE nome_emp IN ('JOÃO','MARIA');

12) SELECT nome_emp
        FROM empregados
      WHERE nome_emp LIKE 'J%';
```

Usando condições de busca (WHERE) 5/5

```
13) SELECT nome_emp
        FROM empregados
      WHERE nome_emp LIKE '%O';

14) SELECT nome_emp
        FROM empregados
      WHERE nome_emp LIKE '%O%';

15) SELECT nome_emp
        FROM nome_emp LIKE '_A%';

16) SELECT nome_emp
        FROM empregados
      WHERE NOT LIKE 'J%';
```

Usando valores calculados na cláusula WHERE

Você pode usar uma expressão aritmética para calcular um valor para uma condição de busca.

```
SELECT nome_emp, salario
    FROM empregados
WHERE salario * 1.30  > 1800;
```

Combinando condições (OR e AND)

1)
```
SELECT nome_emp
    FROM empregados
  WHERE salario > 1000 AND cod_dept  = 5100;
```

2)
```
SELECT nome_emp
    FROM empregados
  WHERE salario > 1000 OR cod_dept  = 5100;
```

3)
```
SELECT nome_emp
    FROM empregados
  WHERE salario > 1000 AND cod_dept  = 5100
                OR nome_emp = 'MARIA';
```

Aggregate Functions (1/4)

Você pode usar AGGREGATE FUNCTIONS AVG, COUNT, MAX, MIN e SUM para executar cálculos dentro do seu SELECT para sumarizar informações sobre um grupo de linhas de uma tabela.

Função	Significado
AVG	Retorna a média de todos os valores de uma determinada coluna
COUNT	Retorna o total de linhas que satisfazem uma condição
MAX	Retorna o maior valor de uma determinada coluna
MIN	Retorna o menor valor de uma determinada coluna
SUM	Retorna o somatório de uma determinada coluna

Aggregate Functions (2/4)

1) SELECT AVG (salario) AS "Média dos salários"
FROM empregados;

```
Média dos salários
---------------------
                3425
```

2) SELECT SUM (salario) AS "Total dos salários"
FROM empregados;

```
Total dos salários
---------------------
              13700
```

Aggregate Functions (3/4)

```
3) SELECT MAX(salario) AS "Maior salário"
      FROM empregados;

     Maior salário
     ---------------
                6000

4) SELECT  MIN (salario) AS "Menor salário"
      FROM empregados;

    Menor salário
    ----------------
                  1200
```

Aggregate Functions (4/4)

5) SELECT COUNT(*) AS "Total de empregados"
FROM empregados;

Total de empregados
4

Usando o WHERE em Aggregate Functions

Você pode usar o WHERE para restringir o conjunto de linhas usada em uma AGGREGATE FUNCTION.

```
SELECT MIN (salario) AS "Menor salário"
   FROM empregados
 WHERE cod_dept = 5100;

Menor salário
----------------
          5000
```

Especificando uma coluna no COUNT

Se você especificar um nome de coluna no COUNT, unicamente linhas contendo valores (NOT NULL) serão contadas. Quando você usa COUNT(*), todas as linhas serão contadas.

```
SELECT COUNT(telefone) AS "Empregados com Telefone"
  FROM empregados
  WHERE cod_dept = 5100;

Empregados com Telefone
------------------------------
                             3

1 row selected.
```

Eliminando linhas duplicadas em Aggregate Functions

```
SELECT COUNT(DISTINCT COD_DEPT)
            as "Total de Departamentos"
   FROM empregados

Total de Departamentos
----------------------------
                           3
1 row selected.
```

Agrupando Informações (GROUP BY)

Você pode usar AGGREGATE FUNCTIONS para mostrar informações de grupos de linhas. Para sumarizar informações de um grupos linhas, use a cláusula GROUP BY seguida da coluna que contém valores para ser agrupados juntos.

```
SELECT cod_dept, count(*) as "total"
   FROM empregados
  GROUP BY cod_dept;
```

cod_dept	total
4600	1
5100	2
5200	1

Usando ORDER BY com Aggregate Functions

```
SELECT cod_dept, AVG(salario) AS "Média Salário"
  FROM empregados
 GROUP BY cod_dept
 ORDER BY 2;

cod_dept    Média Salário
----------  -----------------
      5200             1200
      4600             1500
      5100             5500

    3 row selected.
```

Usando HAVING

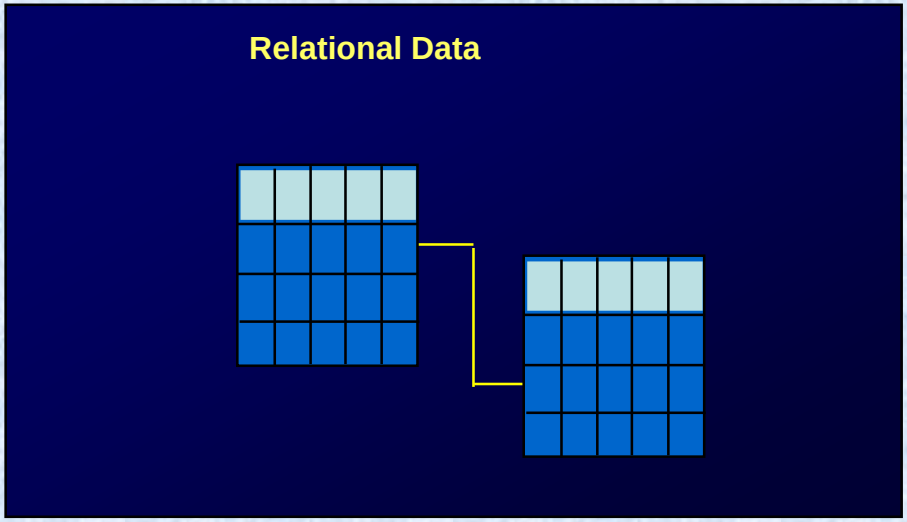
A cláusula HAVING permite adicionar um condição para cada grupo. Tem o mesmo comportamento do WHERE. Você usa a clausula HAVING para eliminar grupos do resultado, assim como usa o WHERE para eliminar linhas.

```
SELECT cod_dept,count(*) AS "total"
  FROM empregados
GROUP BY cod_dept
HAVING COUNT(*) > 1;

 cod_dept     total
 -----------  ------
      5100         2

1 row selected.
```

Acessando múltiplas tabelas (JOIN)



Para ligar duas tabelas, você precisa associar uma ou mais colunas da tabela com uma ou mais colunas da segunda tabela.

Acessando múltiplas tabelas (JOIN)

Exemplo:

```
SELECT cod_emp, nome_emp, departamento.cod_dept,
                    departamento.nome_dept
          FROM empregados, departamentos
        WHERE empregados.cod_dept = departamentos.cod_dept;
```

COD_EMP	NOME_EMP	COD_DEPT	NOME_DEPT
2898	PEDRO	5100	INFORMATICA
2899	MARIA	5100	INFORMATICA
2096	JOÃO	5200	MARKETING
2437	CARLOS	4600	MANUTENÇÃO

Acessando múltiplas tabelas (JOIN)

Lembretes sobre JOIN

- você está combinando linhas de duas ou mais tabelas

- um JOIN é baseado em colunas iguais (DATATYPES)

- se uma coluna no SELECT tem o mesmo nome em mais de uma tabela, é necessário um prefixo (alias ou nomes da tabela)

- a cláusula FROM contém o nome de todas as tabelas do JOIN

- você deveria usar o WHERE para limitar o resultado da tabela

Acessando múltiplas tabelas (JOIN)

PRODUTO CARTESIANO

- Um dos erros mais freqüentes referente a utilização de um JOIN é o **produto cartesiano**. Tal problema acontece quando duas tabelas estão na cláusula FROM mas não foram ligadas no WHERE. Isto gera um grande volume de informações, visto que para cada registros de uma tabela são buscados todos os registros da outra tabela, gerando assim um produto cartesiano
- Um produto cartesiano sempre gera muitas linhas e raramente é proveitoso, além de ter baixo índice de performance. Por exemplo, um produto cartesiano de duas tabelas, cada uma com 100 linhas, irá gerar 10.000 linhas acarretando um tempo de processamento muito alto. Sempre inclua uma condição de join, a menos que você necessariamente precise usar um produto
cartesiano.

- ***O join se torna necessário em 99,9% das consultas que efetuamos no banco de dados*****.**

Usando Alias

Você usa um alias para qualificar um nome de tabela. Especifica-se um alias na cláusula FROM.

from departamentos dep, empregados emp

```
SELECT emp.cod_emp, emp.nome_emp,
       dep.cod_dept, dep.nome_dept
  FROM  empregados emp, departamentos dep
WHERE emp.cod_dept = dep.cod_dept;
```

Usando UNION

Você usa o UNION para unir o resultado de duas ou mais consultas (QUERY). Para isto, o total de colunas e o tipo das colunas envolvidas nas consultas devem ser iguais.

```
SELECT emp.cod_emp, emp.nome_emp,
   FROM empregados emp
 WHERE  emp.cod_emp  = 2898
UNION
SELECT emp.cod_emp, emp.nome_emp,
  FROM empregados emp
 WHERE emp.cod_emp  = 2899;
```

Obs.: o UNION combina as colunas do SELECT e remove as linhas duplicadas.

Usando uma subquery com IN

```
SELECT a.nome_emp
   FROM empregados a
 WHERE a.cod_dept IN (SELECT  b.cod_dept
                        FROM departamento b
                      WHERE b.nome_dept like 'M%');
```

Usando EXISTS

```
SELECT a.nome_cli
      FROM clientes a
    WHERE EXISTS (SELECT debitos b
                                  FROM banco b where a.cod_cli =
b.cod_cli);
```

Considerações sobre o uso de subqueries

♦Quando uma SUBQUERY não possui nenhum vínculo com a QUERY externa, esta é executada primeira seguida da QUERY mais externa. A SUBQUERY serve como um filtro para a QUERY mais externa. Sendo assim, se a SUBQUERY é retorna um volume grande de informações, a performance pode ficar ameaçada.

♦Quando uma SUBQUERY possui vínculo com a QUERY externa, esta é executada a cada execução da QUERY mais externa. Evite a utilização deste tipo de SUBQUERY.

♦SUBQUERIES devem ser evitadas, visto que podem degradar a performance.

Inserindo dados em uma tabela

Para adicionar novas linhas em uma tabela, use o comando INSERT e especifique o valores que você que adicionar.

Sintaxe:

```
INSERT INTO table [(column [,column…])]
VALUES (value [,value…]);
```

Ex.:

```
INSERT INTO departamento
VALUES (5300, 'FINANCEIRO');
        ou
INSERT INTO departamento (cod_dept, nome_dept)
VALUES  (5300, 'FINANCEIRO');
```

Inserindo linhas através de um SELECT

Você pode inserir linhas oriundas de um resultado de um SELECT.

Sintaxe:

```
INSERT INTO table_name [(column,column...)]
Subquery
```

Ex.:

```
INSERT INTO departamento_copia
                (cod_dept, nome_dept )
(SELECT cod_dept,nome_dept
        FROM departamento);
```

Alterando dados de uma tabela

Use o comando UPDATE para modificar colunas de uma tabela. Este comando modifica linhas específicas, se a cláusula WHERE for especificada.

Sintaxe:

```
UPDATE table
SET column = value [,column= value]
[WHERE condition]
```

Ex.:

```
UPDATE empregados
        SET salario = salario * 1.30
WHERE salario < 1300;
```

Removendo dados de uma tabela

Use o comando DELETE para remover linhas de uma tabela.

Sintaxe:

```
DELETE FROM table
[WHERE condition]
```

Ex.:

```
DELETE empregados
    WHERE salario < 1300;
```

DDL – Data Definition Language

Tables

Base de dados relacionais apresentam informações através de uma coleção de tabelas. As tabelas abaixo representam uma base de dados relacional.

EMPREGADO

COD_EMP	NOME_EMP	COD_DEPT
2096	JOÃO	5200
2437	CARLOS	4600
2898	PEDRO	5100
2899	MARIA	5100

DEPARTAMENTO

COD_DEPT	NOME_DEPT
5200	MARKETING
4600	MANUTENÇÃO
5100	INFORMÁTICA

Tables

Sintaxe - Criação:

CREATE TABLE table_name

({column_name data_type [NOT] [NULL] [DEFAULT expression]})

DEFAULT expression => define que quando não for atribuído um valor para a coluna no insert, o campo assumirá o valor da expression.

Exemplo:

```
CREATE TABLE EMPREGADOS
(ID             NUMBER(4) NOT NULL,
DEPT_ID    NUMBER(2),
CIDADE      VARCHAR2(20) DEFAULT 'Porto Alegre');
```

Tables

Sintaxe - Exclusão:

DROP TABLE table_name

Exemplo:

DROP TABLE EMPREGADOS;

Tables

Sintaxe - Alteração:

```
ALTER TABLE table_name
[ADD {(column_name data_type  [NOT] [NULL] [DEFAULT expression])}]
[MODIFY {(column_name data_type  [NOT] [NULL] [DEFAULT expression])}]
[DROP COLUMN  column_name]
```

Exemplo:

```
ALTER TABLE EMPREGADOS ADD(IDADE NUMBER(2) NOT NULL);
ALTER TABLE EMPREGADOS MODIFY(CIDADE VARCHAR2(30));
ALTER TABLE EMPREGADOS DROP COLUMN CIDADE;
```

Primary Key (Chave primária)

Para assegurar que linhas duplicadas são sejam armazenadas, uma coluna ou a combinação de colunas é identificado como *PRIMARY KEY* de uma tabela quando definida. Cada entrada na coluna PRIMARY KEY precisa ser única.

COD_EMP	NOME_EMP	COD_DEPT	SALARIO
2096	JOÃO	5200	1200
2437	CARLOS	4600	1500
2898	PEDRO	5100	5000
2899	MARIA	5100	

PRIMARY KEY

Primary Key (Chave primária)

Exemplo – Criação/Exclusão:

```
CREATE TABLE EMPREGADOS
(ID              NUMBER(4) NOT NULL CONSTRAINT EM_PK PRIMARY KEY,
DEPT_ID    NUMBER(2),
CIDADE      VARCHAR2(20));

ALTER TABLE EMPREGADOS DROP CONSTRAINT EM_PK;

ALTER TABLE EMPREGADOS ADD CONSTRAINT EM_PK PRIMARY KEY(ID);
```

Foreign Keys

O relacionamento entre tabelas é feito pela definição de **FOREIGN KEYS.** Uma FOREIGN KEY é um valor ou combinação de valores em uma tabela que existe como uma PRIMARY KEY em outra tabela.

COD_EMP	NOME_EMP	**COD_DEPT**	SALARIO
2096	JOÃO	5200	1500
2437	CARLOS	4600	1200
2898	PEDRO	5100	5000
2899	MARIA	5100	6000

COD_DEPT	NOME_DEPT
5200	MARKETING
4600	MANUTENÇÃO
5100	INFORMÁTICA

Foreign Keys

Exemplo – Criação/Alteração:

```
CREATE TABLE EMPREGADOS
(ID              NUMBER(4) NOT NULL,
DEPT_ID    NUMBER(2) CONSTRAINT DEPT_FK REFERENCES DEPT(DEPTNO),
CIDADE      VARCHAR2(20));

ALTER TABLE EMPREGADOS DROP CONSTRAINT DEPT_PK;
ALTER TABLE EMPREGADOS ADD CONSTRAINT DEPT_FK FOREIGN KEY(DEPT_ID)
REFERENCES DEPT(DEPTNO);
DROP TABLE EMP CASCADE CONSTRAINTS; – exclui a tabela EMP e as
FOREIGN KEY´S que apontam para ela
```

Criando Tabelas Através de um Select

Sintaxe:

```
CREATE TABLE table_name AS query;
```

Ex.:

```
CREATE TABLE GERENTES AS
SELECT  EMPNO, ENAME, MGR, HIREDATE, SAL, COMM, DEPTNO
FROM EMP
WHERE JOB = 'MANAGER';
```