Linguagem SQL (Structured Query Language)

A Linguagem SQL

- A linguagem SQL implementa de uma maneira mais amigável as operações da álgebra relacional.
- Embora SQL seja conhecida como uma *linguagem de consulta*, ela contém muitos outros recursos como, por exemplo, recursos para definição da estrutura de dados, recursos para modificação de dados no banco de dados e recursos para especificação de restrições de segurança.

A Linguagem SQL

- Originalmente a SQL foi chamada de SEQUEL (Structured English QUEry Language) e foi projetada e implementada pela IBM como uma interface para um sistema de banco de dados relacional chamado SYSTEM R.
- Hoje SQL é uma linguagem padrão para bancos de dados relacionais.

A Linguagem SQL

- Em 1986 foi definida a primeira versão padrão do SQL (ANSI 1986), que foi revisada e expandida para o SQL2 (1992).
- Várias outras versões surgiram posteriormente, expandindo a SQL com conceitos de orientação a objetos e XML.
- SQL usa os termos *Table* (Tabela), *Row* (Linha) *e Column* (Coluna) para relação, tupla e atributo, respectivamente.

Conceitos de Esquema em SQL

- As primeiras versões da SQL não incluíam o conceito de esquema de banco de dados relacional, desta forma todas as tabelas eram consideradas parte do mesmo esquema.
- O conceito de esquema SQL é utilizado para agrupar tabelas e outros objetos que pertencem à mesma aplicação de banco de dados.

Conceitos de Esquema

- Um esquema SQL é definido por um nome e inclui uma identificação de autorização para identificar o usuário ou a conta que é dona do esquema.
- Os elementos de um esquema incluem tabelas, restrições, visões, domínios, dentre outros.
- O esquema é criado através do comando CREATE SCHEMA
- Ex.: CREATE SCHEMA Company

  AUTHORIZATION Jsmith;

Comando CREATE TABLE, Tipos de Dados e Restrições

- CREATE TABLE é usado para especificar uma nova relação dando um nome e informando os seus atributos e restrições.
- Os atributos são especificados primeiro informando o nome, o tipo de dado e qualquer restrição para o atributo, tais como NOT NULL.

Comando CREATE TABLE, Tipos de Dados e Restrições

- As restrições de chave e de integridade referencial podem ser especificadas no comando CREATE TABLE ou no comando ALTER TABLE.
- **EXEMPLOS**:

```
CREATE TABLE  empregado
    (codemp       INTEGER          NOT  NULL,
     nomeemp      VARCHAR(30)  NOT  NULL,
     salario        NUMERIC(8,2)  NOT NULL,
     dataadm      DATE               NOT NULL,
     coddpto      INTEGER          NOT NULL,
     CONSTRAINT PKEMP PRIMARY KEY (codemp),
     CONSTRAINT FKEMPDEP FOREIGN KEY (coddpto) REFERENCES
departamento(coddpto));
```

Comando CREATE TABLE, Tipos de Dados e Restrições

CREATE TABLE *r*

(*A_1* *D_1* **[NOT NULL],**

A_2 *D_2* **[NOT NULL],**

.....

A_n *D_n* **[NOT NULL],**

CONSTRAINT *restricao1* **PRIMARY KEY** (chave_primaria)**,**

CONSTRAINT *restricao2* **FOREIGN KEY** (chave_estrangeira_1)
REFERENCES tabela2 (atributo2),

. . .

CONSTRAINT *restricaon* **FOREIGN KEY (**chave_estrangeira_n**)**
REFERENCES tabelan (atributon))

Tipos de Dados

- Tipos de dados numéricos:
  - Inteiro (Integer ou Int e Smallint);
  - Real (Float, real, double precision);
  - Números formatados (decimal(i, j) ou Dec(i, j) ou numeric(i, j)) - i é número total de dígitos e j o número de dígitos a direita do ponto decimal.
  - Cadeias de caracteres de tamanho fixo (char(n) ou character(n)).
  - Cadeias de caracteres de tamanho variável (varchar(n), char varying(n) ou character varying(n).

Tipos de Dados

- Data (date - aaaa-mm-dd)
- Hora (time - hh:mm:ss)
- Alternativamente, pode-se declarar um domínio e usar o nome do domínio como tipo de dado de cada atributo.
- Ex.: CREATE DOMAIN TipoCPF AS CHAR(14)

Especificando Restrições e Valores Default

- Pelo fato do SQL permitir valores nulos como valores de atributos, uma restrição NOT NULL pode ser especificada se o valor nulo não pode ser definido para um determinado atributo.
- Para definir um valor default para um atributo utiliza-se a cláusula DEFAULT <valor> na definição do atributo.

Especificando Restrições e Valores Default

- Ex.:

```
CREATE TABLE  empregado
(   codemp        INTEGER       NOT  NULL,
    . . .
    coddpto       INTEGER NOT NULL DEFAULT 1,
    CONSTRAINT PKEMP PRIMARY KEY (codemp),
    CONSTRAINT FKEMPDEP FOREIGN KEY (coddpto)
    REFERENCES departamento(coddpto));
```

Especificando Restrições e Valores Default

- Chaves primárias são definidas com a cláusula PRIMARY KEY.
- Chaves alternativas ou secundárias são definidas com a cláusula UNIQUE.
- A restrição de integridade referencial é definida com a cláusula FOREIGN KEY.

Especificando Restrições e Valores Default

- As chaves estrangeiras incluem as opções de bloqueio (*default*), propagação (cascade), substituição por nulo (set null) e substituição pelo valor default (set default) que devem ser especificadas com a cláusula ON DELETE ou ON UPDATE.
- Ex.:

CREATE TABLE empregado
(codemp INTEGER **NOT NULL, . . .**
coddpto INTEGER **NOT NULL DEFAULT 1**,
CONSTRAINT PKEMP **PRIMARY KEY** (codemp),
CONSTRAINT FKEMPDEP **FOREIGN KEY** (coddpto) **REFERENCES** departamento(coddpto) **ON DELETE SET DEFAULT ON UPDATE CASCADE**);

Exclusão de Tabela

- Para remover uma relação de um banco de dados SQL, usa-se o comando **DROP TABLE,** cuja sintaxe é a seguinte:
  - **DROP TABLE** *r*, onde *r* é o nome da tabela.
- **Ex.: DROP TABLE** EMPREGADO.
- **OBS**: Deve-se usar as opções **RESTRICT** e **CASCADE** para verificar restrições de integridade referencial.
  - **DROP TABLE** *r* **RESTRICT**, não elimina a tabela se houver alguma restrição ou visão para a ela.
  - **DROP TABLE** *r* **CASCADE**, elimina a relação e todas as restrições e visões.

Exclusão de Esquema

- Se todo o esquema não for mais necessário o comando DROP SCHEMA é utilizado.
- Opções:
  - CASCADE - remove todos os objetos do esquema;
  - RESTRICT - elimina o esquema somente se não houver objetos no esquema.
  - Ex.: DROP SCHEMA Company CASCADE;

Comando de Alteração da Definição da Tabela

- Para adicionar ou remover atributos de uma relação, usa-se o comando **ALTER TABLE.**
  - **ALTER TABLE** *r* **ADD** *A D*, adiciona à tabela existente *r*, o atributo *A*, que pertence ao domínio *D*.
  - **Ex.: ALTER TABLE** EMPREGADO **ADD** CPFEMP CHAR(15);
  - **OBS**: Se já existirem tuplas na relação *r*, o novo atributo receberá valores nulos nessas tuplas.

Comando de Alteração da Definição da Tabela

- **ALTER TABLE** *r* **DROP** *A*, elimina o atributo *A* da tabela *r*. Para verificar restrições de integridade referencial, usa-se as opções **RESTRICT** e **CASCADE.**
  - **ALTER TABLE** *r* **DROP** *A* **CASCADE**, elimina o atributo *A* e todos as restrições e visões que envolvem o atributo *A* .
  - **ALTER TABLE** *r* **DROP** *A* **RESTRICT,** não elimina o atributo *A* se a ele tiver alguma restrição ou visão.
  - **EXEMPLO: ALTER TABLE** EMPREGADO **DROP** CODDPTO

Consultas básicas em SQL

- A estrutura básica de uma consulta em SQL consiste de três cláusulas básicas: **SELECT (projeção) , FROM (produto cartesiano)** e **WHERE (seleção).**

  SELECT $A_1,, A_2,,..., A_n,$

  FROM $r_1,, r_2,,..., r_m$

  WHERE P

- onde cada A_i, representa um atributo, cada r_i é uma relação e P é um predicado.
- Esta consulta é equivalente à seguinte expressão em álgebra relacional:

$$\pi_{A1, A2, ..., An}(\sigma_P(r_1 \ X \ r_2 \ X \ r_m))$$

Consultas básicas em SQL

- **OBS:**
  - A cláusula **WHERE** pode ser omitida, se não houver nenhuma condição de seleção.
  - A lista de atributos A_1,, A_2,,..., A_n pode ser substituída por um asterisco (*) quando todos os atributos de todas as relações da cláusula **FROM** forem selecionados.
  - O resultado de uma consulta SQL, como era de se esperar, é uma relação.

Consultas básicas em SQL

- **EXEMPLOS:**
  - Selecionar o nome de todos os empregados.

```
SELECT  NOMEEMP
FROM    EMPREGADO
```

  - Obter o nome dos empregados cujo salário seja superior a 500 reais.

```
SELECT  NOMEEMP
FROM    EMPREGADO
WHERE   SALARIO > 500
```

Consultas básicas em SQL

- Obter os empregados admitidos depois de “01/05/96”.

```
SELECT CODEMP, NOMEEMP, SALARIO,
   DATAADM, CODDPTO
FROM     EMPREGADO
WHERE   DATAADM > “01/05/96”
```

- Ou, alternativamente,

```
SELECT  *
FROM     EMPREGADO
WHERE   DATAADM > “01/05/96”
```

Tuplas Duplicadas

- A linguagem SQL não faz a eliminação automática de tuplas duplicadas. Para isto deve-se usar a palavra **DISTINCT** depois do **SELECT.**

- **Ex.:**

```
SELECT DISTINCT NOMEEMP
FROM EMPREGADO
```

- Para especificar que as duplicações não devem ser removidas deve-se usar a palavra **ALL.** No entanto, isto não é necessário pois é o *default.*

- **Ex.:**

```
SELECT ALL NOMEEMP
FROM EMPREGADO
```

Junções

- Pode-se definir uma junção em termos das operações de produto cartesiano e seleção. $\pi_{nomeemp,dataadm}$(EMPREGADO $\bowtie_{empregado.codemp=alocacao.codemp}$ ALOCACAO)

- Esta consulta pode ser escrita em SQL como:

SELECT DISTINCT NOMEEMP, DATAADM

FROM EMPREGADO, ALOCACAO

WHERE EMPREGADO.CODEMP = ALOCACAO.CODEMP

- **<u>OBS</u>:**
- Na cláusula **WHERE** pode-se usar os conectivos **AND, OR** e **NOT.**
- Pode-se usar o operador **BETWEEN** para especificar um intervalo de valores.
- Pode-se usar também o operador **LIKE,** combinado com **%** e **_**. O caractere **%** substitui vários caracteres, e o caractere **_** substitui um caractere.

- **Ex.:**
  - Encontrar os empregados que ganham entre 500 e 1000 reais.

```
SELECT  *
FROM    EMPREGADO
WHERE   SALARIO  BETWEEN  500 AND  1000
```

  - Ou, alternativamente,

```
SELECT  *
FROM    EMPREGADO
WHERE   SALARIO  >= 500 AND  SALARIO >= 1000
```

– Selecionar o nome dos empregados que foram admitidos a partir de “01/04/96” ou que recebem menos de 500 reais.

```
SELECT  NOMEEMP
FROM    EMPREGADO
WHERE  DATAADM >= “01/04/96”  OR
  SALARIO < 500
```

– Selecionar o nome dos empregados que possuem “Silva” no nome.

```
SELECT  NOMEEMP
FROM    EMPREGADO
WHERE   NOMEEMP  LIKE  “%Silva%”
```

Operações de Conjuntos

- A linguagem SQL inclui cláusulas para as operações tradicionais de conjuntos **union (união), intersect (interseção) e minus (diferença)**.

- **OBS:** Na operação de **união** as tuplas duplicadas são eliminadas automaticamente. Para manter as tuplas duplicadas deve-se usar a cláusula **union all.**

- **EXEMPLO:** Selecionar o nome dos empregados que trabalham no departamento de “INFORMÁTICA” ou participam de projetos com duração superior a “01/08/96”.

```
(SELECT  NOMEEMP
 FROM      EMPREGADO, DEPARTAMENTO
 WHERE   EMPREGADO.CODDPTO =
   DEPARTAMENTO.CODDPTO AND NOMEDPTO =
   “INFORMATICA”)
UNION   (SELECT  NOMEEMP
FROM      EMPREGADO E, ALOCACAO A, PROJETO P
WHERE   E.CODEMP = A.CODEMP AND
   A.CODPROJ =  P.CODPROJ  AND DURACAO >
   “01/08/96”)
```

- **<u>OBS</u>:** Pode-se utilizar o operador **in** para substituir as operações de conjunto **intersect** e **minus.**
  - **Ex:** Selecionar o nome dos empregados que trabalham no departamento de “INFORMATICA” e estão alocados a projetos com duração superior a “01/08/96”.

```
(SELECT  NOMEEMP FROM      EMPREGADO,
   DEPARTAMENTO WHERE   EMPREGADO.CODDPTO =
   DEPARTAMENTO.CODDPTO AND NOMEDPTO =
   “INFORMATICA”)

INTERSECT

(SELECT  NOMEEMP FROM      EMPREGADO,
   ALOCACAO, PROJETO WHERE
   EMPREGADO.CODEMP = ALOCACAO.CODEMP AND
   ALOCACAO.CODPROJ =  PROJETO.CODPROJ
         AND   DURACAO > “01/08/96”)
```

- A consulta anterior pode ser escrita como:

```
SELECT  NOMEEMP
FROM      EMPREGADO, DEPARTAMENTO
WHERE EMPREGADO.CODDPTO =
   DEPARTAMENTO.CODDPTO AND NOMEDPTO =
   “INFORMATICA” AND NOMEEMP
IN
(SELECT  NOMEEMP
FROM  EMPREGADO, ALOCACAO, PROJETO
WHERE
   EMPREGADO.CODEMP=ALOCACAO.CODEMP
        AND ALOCACAO.CODPROJ =
   PROJETO.CODPROJ   AND DURACAO >
   “01/08/96”)
```

Variáveis tuplas

- Selecionar o nome dos empregados que trabalham no departamento de “INFORMATICA”.

```
SELECT  NOMEEMP
FROM EMPREGADO AS E,
  DEPARTAMENTO AS D
WHERE   E.CODDPTO = D.CODDPTO AND
  NOMEDPTO = “INFORMATICA”
```

Variáveis Tuplas

– Selecionar o nome dos empregados que trabalham no mesmo departamento do empregado “JOSÉ DA SILVA”.

SELECT F.NOMEEMP

FROM EMPREGADO **AS** E, EMPREGADO **AS** F

WHERE E.CODDPTO = F.CODPTO **AND** E.NOMEEMP = “JOSÉ DA SILVA”

Comparação de Conjuntos

- O operador **in,** visto anteriormente, funciona apenas quando a comparação da cláusula **WHERE** for uma igualdade.
- **Ex.:** Selecionar o nome dos empregados que tem salário superior a algum empregado do departamento de “INFORMÁTICA”.

```
SELECT  E.NOMEEMP
FROM    EMPREGADO E,  EMPREGADO F,
        DEPARTAMENTO D
WHERE   E.SALARIO > F.SALARIO  AND
        F.CODDPTO = D.CODDPTO AND
        D.NOMEDPTO = “INFORMATICA”
```

Comparação de Conjuntos

```
SELECT  NOMEEMP
FROM    EMPREGADO
WHERE   SALARIO > SOME
    (SELECT  SALARIO
    FROM     EMPREGADO E, DEPARTAMENTO D
    WHERE E.CODDPTO=D.CODDPTO AND
      NOMEDPTO = "INFORMATICA")
```

- **OBS**: A construção **>SOME** representa "maior que algum", e a construção **>ALL** significa "maior que todos".

Comparação de Conjuntos

– **Ex.:** Selecionar o nome dos empregados que tem salário superior a todos os empregados do departamento de “INFORMÁTICA”.

```
SELECT  NOMEEMP
FROM      EMPREGADO
WHERE   SALARIO > ALL
(SELECT  SALARIO
  FROM EMPREGADO, DEPARTAMENTO
   WHERE EMPREGADO.CODDPTO =
   DEPARTAMENTO.CODDPTO  AND
   NOMEDPTO = “INFORMATICA”)
```

Função EXISTS

- Ex.: Selecione o nome de todos os empregados que tem dependentes.

```
SELECT NomeEmp
FROM Empregado as E
WHERE EXISTS
(SELECT * FROM Dependente as D
WHERE E.CodEmp = D.CodEmp)
```

Função EXISTS

- Ex.: Selecione o nome de todos os empregados que não tem dependentes.

```
SELECT NomeEmp
FROM Empregado as E
WHERE NOT EXISTS
(SELECT * FROM Dependente as D
WHERE E.CodEmp = D.CodEmp)
```

Conjuntos Explícitos

- Ex.: Selecione o código e o nome de todos os empregados que trabalham nos projetos 1, 2 ou 3

```
SELECT CodEmp, NomeEmp
FROM Empregado AS E, Trabalha AS T
WHERE E.CodEmp = T.CodEmp AND
   T.CodProj IN (1,2,3)
```

Trabalhando com valores Nulos

- Selecione os nomes de todos os empregados que não tem supervisores.

```
SELECT NomeEmp
FROM Empregado
WHERE CodSupervisor IS NULL;
```

- Para selecionar os empregados que tem supervisor usa-se o IS NOT NULL.

Junção entre Tabelas

- SQL permite especificar a condição da junção na cláusula FROM, usando a cláusula JOIN (INNER JOIN).
- Ex.: Recupere o nome e o endereço dos empregados do departamento Pesquisa.

```
SELECT NomeEmp, Endereco
FROM Empregado AS E JOIN Departamento AS D
    ON E.CodDpto = D.CodDpto
WHERE NomeDpto = ‘Pesquisa’
```

OUTER JOIN

- Ex.: Liste o nome de todos os empregados com o nome dos seus supervisores.

```
SELECT E.NomeEmp, S.NomeEmp
FROM Empregado AS E LEFT OUTER JOIN
    Empregado AS S ON E.CodSupervisor =
    S.CodEmp
```

- Há também o RIGHT OUTER JOIN e FULL OUTER JOIN.

Funções de agregação e agrupamento

- A linguagem SQL possui construções que incorporam os conceitos de agrupamento e funções agregadas, presentes em muitas aplicações de bancos de dados.
- As principais funções agregadas de SQL são:
  - **COUNT:** conta o número de tuplas recuperadas em uma consulta;
  - **SUM:** soma os valores de um atributo recuperado em uma consulta;
  - **MAX:** recupera o valor máximo para um atributo em uma consulta;
  - **MIN:** recupera o valor mínimo para um atributo em uma consulta;
  - **AVG:** calcula a média dos valores de um atributo recuperado em uma consulta.

Cláusula GROUP BY e HAVING

- A cláusula **GROUP BY** serve para agrupar tuplas que possuem o mesmo valor para os atributos relacionados no **GROUP BY.** A cláusula **HAVING** pode ser usada em conjunto com a cláusula **GROUP BY** para especificar uma condição de seleção no grupo de atributos recuperados. Somente os grupos que satisfizerem a condição serão recuperados.

Exemplos

- Liste a soma dos salários de todos os empregados, o maior, o menor e a média dos salários deles.

  SELECT SUM(Salario), MAX (Salario), MIN (Salario), AVG (Salario)

  FROM Empregado
- Liste o número de empregados da empresa.

  SELECT COUNT(*) FROM Empregado

Exemplos

- Para cada departamento recupere o código, o número de empregados e a média dos salários.

```
SELECT CodDpto, COUNT(*), AVG(Salario)
FROM Empregado
GROUP BY CodDpto
```

- Conte o número de salários distintos no banco de dados.

```
SELECT COUNT(DISTINCT Salario)
FROM Empregado
```

Exemplos

- Para cada projeto com mais do que dois empregados trabalhando, liste o código e o nome do projeto e o número de empregados que trabalham nele.

```
SELECT CodProj, ProjNome, COUNT(*)
FROM Projeto P, Trabalha T
WHERE P.CodProj = T.CodProj
GROUP BY CodProj, ProjNome
HAVING COUNT (*) > 2
```

Cláusula ORDER BY

- Para ordenar o resultado usa-se a cláusula ORDER BY.
  - Listar os empregados em ordem alfabética

    SELECT * FROM Empregado

    ORDER BY NomeEmp
  - Listar os empregados em ordem decrescente de salário. Se houver empate, usar o nome em ordem alfabética.

    SELECT * FROM Empregado

    ORDER BY Salario DESC, NomeEmp ASC

Comando INSERT

- O comando **INSERT** é usado para inserir uma tupla ou conjunto de tuplas no banco de dados. Exemplos:
  - **INSERT INTO** EMPREGADO **VALUES** (003, “José Silva”, 500, “19/05/96”, 01)
  - **INSERT INTO** EMPREGADO (CODEMP, NOMEEMP, DATAADM) **VALUES** (003,“Jose Silva”, “19/05/96”)

Comando INSERT

```
CREATE TABLE InfoDepart (
DptoNome   VARCHAR(15) NOT NULL
   PRIMARY KEY,
QtdeEmp   INTEGER,
TotalSal INTEGER)

INSERT INTO  InfoDepart
SELECT  DptoNome, COUNT(*), SUM(Salario)
   FROM (Departamento D JOIN Empregado E ON
   D.CodDpto = E.CodDpto)
GROUP BY DptoNome
```

Comando DELETE

- O comando **DELETE** é usado para remover uma tupla ou conjunto de tuplas do banco de dados. Forma geral do comando:

  DELETE FROM *r*

  WHERE *P*

- onde *r* representa uma relação e *P* um predicado.

Comando DELETE - Exemplos

- Remover do banco de dados os empregados que ganham menos que 200 reais.

```
DELETE FROM  EMPREGADO
WHERE  SALARIO < 200
```

- Remover do banco de dados os empregados que trabalham no departamento de “Informática”.

```
DELETE FROM EMPREGADO
WHERE  CODDPTO  IN
(SELECT  CODPTO FROM DEPARTAMENTO    WHERE
   NOMEDPTO = “INFORMÁTICA”)
```

- Remover todas as tuplas da relação empregado.

```
DELETE FROM  EMPREGADO
```

Comando UPDATE

- O comando **UPDATE** é usado para modificar os valores dos atributos de uma ou mais tuplas selecionadas.
- **Ex.:** Dar a todos os empregados do departamento de “INFORMÁTICA” um aumento de 10%.

```
UPDATE  EMPREGADO
SET  SALARIO = SALARIO * 1.10
WHERE  CODDPTO  IN (SELECT  CODDPTO
   FROM  DEPARTAMENTO
   WHERE  NOMEDPTO = “INFORMÁTICA”)
```

Comando UPDATE - Exemplo

- Alterar a data de admissão do empregado de código 10 para “19/05/96”.

UPDATE EMPREGADO
SET DATAADM = “19/05/96”
WHERE CODEMP = 10

Visões (Tabelas Virtuais)

- Uma visão em SQL é uma relação simples derivada de outras relações, ou mesmo de outras visões previamente definidas. A forma do comando para criar uma visão é a seguinte:
  - **CREATE VIEW** *v* **AS** *E*
  - onde *v* é o nome da visão e *E* uma expressão de consulta.

Visões - Exemplos

- Criar a visão EMP-PROJ com o nome do empregado, o nome do projeto, a data de alocação e a duração do projeto.

```
CREATE VIEW EMP-PROJ
AS SELECT NOMEEMP, NOMEPROJ,
   DATAALOC, DURACAO
FROM    EMPREGADO, ALOCACAO, PROJETO
WHERE EMPREGADO.CODEMP =
   ALOCACAO.CODEMP AND
   ALOCACAO.CODPROJ = PROJETO.CODPROJ
```

Visões - Exemplos

- Criar a visão DEPTO_INFO com o nome do departamento, o número de empregados e a soma total dos salários de todos os empregados.

```
CREATE VIEW  DEPTO-INFO (NOMEDPTO,
N_EMPS, TOTAL_SAL)  AS   (SELECT
NOMEDPTO, COUNT(*),  SUM(SALARIO)
FROM  EMPREGADO, DEPARTAMENTO
WHERE  EMPREGADO.CODDPTO =
DEPARTAMENTO.CODDPTO
GROUP BY  NOMEDPTO
```

Visões

- Pode-se fazer consultas diretamente em visões.
- **Ex.:** Selecionar o nome do departamento, o número de empregados e o salário total para departamentos com mais de 20 empregados.

```
SELECT  NOMEDPTO, N_EMPS,
   TOTAL_SAL
FROM      DEPTO_INFO
WHERE   N_EMPS > 20
```

Visões - DROP VIEW e Atualização

- Para eliminar uma visão usa-se o comando **DROP VIEW**.
- **Ex.:** Eliminar do banco de dados a visão EMP_PROJ.

  DROP VIEW EMP_PROJ
- Operações de atualização em visões são permitidas, embora nem sempre façam sentido.
- **Ex.:** Alterar o total de salário do departamento de "INFORMÁTICA" para 100000 reais.

  UPDATE DPTO_INFO **SET** TOTAL_SAL = 100000
  WHERE NOMEDPTO = "INFORMÁTICA"

  Ocorre um erro pois não existe a coluna Total_Sal em uma tabela real.

Visões Atualizáveis

- Regras sobre atualização em visões:
  - Uma visão definida sobre uma única tabela é atualizável se os atributos da visão contêm a chave primária, as chaves candidatas da relação base e todos os atributos NOT NULL.
  - Uma visão definida em múltiplas tabelas envolvendo junções geralmente não é atualizável. O comando UPDATE atualiza dados somente de uma tabela.
  - Uma visão definida usando funções agregadas ou agrupamento de atributos não é atualizável.