SQL

Banco de Dados I

Prof. Guilherme Tavares de Assis

Universidade Federal de Ouro Preto – UFOP

Instituto de Ciências Exatas e Biológicas – ICEB

Departamento de Computação – DECOM

SQL (*Structured Query Language*)

- SQL é uma linguagem de consulta que implementa as operações da álgebra relacional de forma bem amigável.

- Além de permitir a realização de consultas, SQL possibilita:
  - definição da estrutura de dados;
  - definição de restrições de integridade;
  - modificação de dados no banco de dados;
  - especificação de restrições de segurança e controle de transações;
  - utilização em linguagens hospedeiras.

- De uma forma geral, SQL utiliza os termos tabela, linha e coluna para relação, tupla e atributo, respectivamente.

SQL (*Structured Query Language*)

- SQL foi projetada e implementada pela IBM, como uma interface para o sistema de banco de dados relacional SYSTEM R, tendo sido chamada inicialmente de SEQUEL (*Structured English QUEry Language*).

- Em 1986, um trabalho conjunto entre o ANSI (*American National Standards Institute*) e o ISO (*International Standards Organization*) conduziu a primeira versão padrão de SQL (ANSI 1986), chamada SQL1.
  - Em 1992, tal padrão foi revisado e mais expandido, gerando a SQL2.
  - A SQL3 deverá estender a SQL2 com banco de dados orientados a objetos.

Esquema e Catálogo

- A SQL1 não contemplava o conceito de esquema de banco de dados relacional.
  - Todas as tabelas eram consideradas parte do mesmo esquema.
- Esquema, incorporado à SQL2, é utilizado para agrupar tabelas e outros componentes que pertencem à mesma aplicação de banco de dados.
  - Um esquema é definido por um nome e inclui um identificador de autorização para indicar o usuário que é dono do esquema.

    CREATE SCHEMA Empresa **AUTHORIZATION** JSilva;
  - Os elementos de um esquema incluem tabelas, restrições, visões, domínios e outros componentes.
- Catálogo é uma coleção de esquemas num ambiente SQL.
  - Esquemas de um mesmo catálogo podem compartilhar certos elementos como, por exemplo, definições de domínio.

Comando CREATE TABLE

- CREATE TABLE é o comando usado para especificar uma nova relação, fornecendo um nome e informando os seus atributos e as suas restrições.
  - Inicialmente, os atributos são especificados, informando o nome, o tipo de dado e qualquer restrição para o atributo, como, por exemplo, NOT NULL.
  - Depois, são especificadas as restrições (CONSTRAINT) de chave (PRIMARY KEY, UNIQUE) e de integridade referencial (FOREIGN KEY).
    - Tais restrições podem ser especificadas também no comando ALTER TABLE, que permite a alteração da definição de uma relação.

Comando CREATE TABLE

```
CREATE TABLE  Empregado
(       PrimeiroNome        VARCHAR(15)       NOT NULL,
        InicialMeio         CHAR,
        UltimoNome          VARCHAR(15)       NOT NULL,
        NumEmpregado        CHAR(9)           NOT NULL,
        DataNascimento      DATE,
        Endereco            VARCHAR(30),
        Sexo                CHAR,
        Salario             DECIMAL(10,2),
        NumSupervisor       CHAR(9),
        NumDepto            INT               NOT NULL,
    CONSTRAINT PK_Emp PRIMARY KEY (NumEmpregado),
    CONSTRAINT FK_NumSup FOREIGN KEY (NumSupervisor)
      REFERENCES  Empregado (NumEmpregado),
    CONSTRAINT FK_EmpDep FOREIGN KEY (NumDepto)
      REFERENCES  Departamento (NumDepto)
);
```

Comando CREATE TABLE

- Tipos de dados numéricos:
  - inteiro: integer ou int, smallint;
  - real: float, real, double precision;
  - números formatados: decimal(i, j) ou numeric(i, j) ou dec(i,j) onde "i" é número de dígitos a esquerda e "j" é o número de dígitos a direita do ponto decimal.

- Tipos de dados alfanuméricos:
  - cadeia de caracteres de tamanho fixo: char(n) ou character(n);
  - cadeia de caracteres de tamanho variável: varchar(n) ou char varying(n) ou character varying(n), onde "n" é o tamanho máximo de caracteres.

Comando CREATE TABLE

- Tipo data: date (aaaa-mm-dd)
- Tipo hora: time (hh:mm:ss)
- Pode-se declarar um domínio e usar o nome do domínio como tipo de dado de algum atributo.

  CREATE DOMAIN TipoNumEmp **AS** CHAR(9);

Comando CREATE TABLE

- Pelo fato da SQL permitir valores nulos como valores de atributos, uma restrição NOT NULL pode ser especificada se o valor nulo não puder ser definido para um atributo.

- Para definir um valor *default* para um atributo, utiliza-se a cláusula DEFAULT <valor> na definição do atributo.
  - O valor *default* é incluído em uma nova tupla, se um valor explícito não for fornecido para o atributo em questão.

```
CREATE TABLE  Empregado
(     . . .
      NumDepto        INT       NOT NULL        DEFAULT 1,
      . . .
 );
```

Comando CREATE TABLE

- Em SQL, pode-se especificar a ação a ser tomada se uma restrição de integridade referencial for violada, mediante à exclusão de uma tupla ou à atualização do valor de uma chave primária.
  - As opções incluem 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 (em uma operação de exclusão) ou ON UPDATE (em uma operação de atualização), em cada restrição de integridade referencial.

Comando CREATE TABLE

```
CREATE TABLE  Empregado
(     PrimeiroNome      VARCHAR(15)      NOT NULL,
      . . .
      NumDepto          INT              NOT NULL      DEFAULT 1,
   CONSTRAINT PK_Emp PRIMARY KEY (NumEmpregado),
   CONSTRAINT FK_NumSup FOREIGN KEY (NumSupervisor)
     REFERENCES  Empregado (NumEmpregado)
     ON DELETE SET NULL   ON UPDATE CASCADE,
   CONSTRAINT FK_EmpDep FOREIGN KEY (NumDepto)
     REFERENCES  Departamento (NumDepto)
     ON DELETE SET DEFAULT   ON UPDATE CASCADE
);
```

Comando DROP SCHEMA

- Para remover um esquema de um banco de dados, SQL usa o comando DROP SCHEMA, cuja sintaxe é:

  DROP SCHEMA <E> <opção>;

  onde <E> é o nome do esquema a ser removido e <opção> pode ser RESTRICT (não elimina o esquema se houver algum elemento nele) ou CASCADE (elimina o esquema e todos os seus elementos).

- Ex.: **DROP SCHEMA** Empresa **CASCADE**;

Comando DROP TABLE

- Para remover uma relação de um banco de dados, SQL usa o comando DROP TABLE, cuja sintaxe é:

  DROP TABLE <R> <opção>;

  onde <R> é o nome da relação a ser removida e <opção> pode ser RESTRICT (não elimina a relação se houver alguma restrição ou visão associada a ela) ou CASCADE (elimina a relação e todas as restrições e visões associadas a ela).

- Ex.: **DROP TABLE** Dependente **CASCADE**;

Comando ALTER TABLE

- O comando ALTER TABLE é usado para adicionar ou remover atributos e restrições de uma relação, e para alterar a definição de um atributo. Possui as seguintes sintaxes:
  - **ALTER TABLE** <R> **ADD** <A> <D> : adiciona o atributo <A>, cujo domínio é <D>, na relação existente <R>.

    ALTER TABLE Empregado **ADD** Serviço VARCHAR(12);

    Neste caso, se já existirem tuplas na relação <R>, o novo atributo receberá valores nulos para essas tuplas.

Comando ALTER TABLE

- **ALTER TABLE** <R> **DROP** <A> <opção> : remove o atributo <A> da relação existente <R>.

  A <opção> pode ser RESTRICT (não elimina o atributo se houver alguma restrição ou visão referenciando-o) ou CASCADE (elimina o atributo, as visões e as restrições que o referenciam).

  ALTER TABLE Empregado **DROP** Endereco **CASCADE**;

Comando ALTER TABLE

- **ALTER TABLE** <R> **ALTER** <A> **DROP DEFAULT** : remove a cláusula de *default* referente ao atributo <A> da relação existente <R>.

  ALTER TABLE Empregado **ALTER** NumDepto **DROP DEFAULT**;

- **ALTER TABLE** <R> **ALTER** <A> **SET DEFAULT** <V>: adiciona uma cláusula de *default*, referente ao atributo <A> da relação existente <R>, com o valor <V>.

  ALTER TABLE Empregado **ALTER** NumDepto **SET DEFAULT** 2;

Comando ALTER TABLE

- **ALTER TABLE** <R> **DROP CONSTRAINT** <C>: remove a restrição <C> referente à relação existente <R>.

  ALTER TABLE Empregado **DROP CONSTRAINT** FK_EmpDep;

- **ALTER TABLE** <R> **ADD CONSTRAINT** <C>: adiciona a restrição <C> na relação existente <R>.

  ALTER TABLE Empregado **ADD CONSTRAINT** FK_NumSup **FOREIGN KEY** (NumSupervisor) **REFERENCES** Empregado (NumEmpregado) **ON DELETE** SET NULL **ON UPDATE** CASCADE;

Consultas Básicas em SQL

- A estrutura básica de uma consulta SQL é:

  SELECT <A_1>, <A_2>, ... , <A_n> {projeção}
  FROM <R_1>, <R_2>, ... , <R_m> {produto cartesiano}
  WHERE <cond>; {seleção}

  - cada <A_i>, para $1 \leq i \leq n$, representa um atributo;
  - cada <R_j>, para $1 \leq j \leq m$, representa uma relação;
  - <cond> é uma condição de seleção (expressão lógica);
  - a cláusula WHERE pode ser omitida, se não houver condição de seleção;
  - a lista de atributos A_1,, A_2, ... , A_n pode ser substituída por um asterisco (*) indicando que todos os atributos de todas as relações da cláusula FROM serão projetados.

- Esta consulta é equivalente à seguinte expressão em álgebra relacional: $\pi_{A1, A2, ...,An}(\sigma_{cond}(R_1 \times R_2 \times R_m))$

Consultas Básicas em SQL

- Exemplos de consultas básicas:
  - Número e nome de todos os empregados:

```
SELECT NumEmpregado, PrimeiroNome, UltimoNome
FROM   Empregado;
```

  - Nome dos empregados cujo salário seja superior a 500 reais:

```
SELECT PrimeiroNome, InicialMeio, UltimoNome
FROM   Empregado
WHERE  Salario > 500;
```

  - Empregados de sexo feminino:

```
SELECT *
FROM   Empregado
WHERE  Sexo = 'F';
```

Palavras-chave DISTINCT e ALL

- Diferentemente do modelo relacional formal, SQL permite duas tuplas idênticas em uma mesma relação.
  - Uma relação em SQL não é um conjunto de tuplas.

- Para remover as tuplas duplicadas no resultado de uma consulta, deve-se usar a palavra-chave DISTINCT na cláusula SELECT.

  SELECT DISTINCT UltimoNome

  FROM Empregado;

- Para especificar que as tuplas duplicadas não devem ser removidas, deve-se usar a palavra-chave ALL (*default*).

  SELECT ALL UltimoNome

  FROM Empregado;

Operador LIKE

- Na SQL, o operador de comparação LIKE permite condições de comparação em partes de uma cadeia de caracteres.
  - O caractere '%' substitui um número arbitrário de caracteres.
  - O caractere '_' substitui um único caractere.

- Listar todos os empregados cujo nome começa com 'A':

```
SELECT *
FROM   Empregado
WHERE  PrimeiroNome LIKE 'A%';
```

- Listar todos os empregados que nasceram na década de 50:

```
SELECT *
FROM   Empregado
WHERE  DataNascimento LIKE '_ _ 5%';
```

Operador BETWEEN

- Por meio do operador BETWEEN, é possível especificar um intervalo de valores para um atributo.

- Listar todos os empregados do departamento 5 cujo salário esteja entre 5000 e 8000:

```
SELECT  *
FROM    Empregado
WHERE   (Salário BETWEEN 5000 AND 8000) AND
        NumDepto = 5;
```

Valor NULL

- SQL permite consultas que verificam se o valor de um atributo é NULL; para isto, utiliza-se o operador IS ou IS NOT.

```
SELECT PrimeiroNome, UltimoNome
FROM   Empregado
WHERE  NumSupervisor IS NULL;
```

Renomeando Relações

- Em consultas SQL, é possível "renomear" nomes de relações, usando o operador AS.

- Obter o nome e o endereço dos empregados que trabalham no departamento 'Pesquisa':

```
SELECT E.PrimeiroNome, E.UltimoNome, E.Endereco
FROM   Empregado AS E, Departamento AS D
WHERE  (E.NumDepto = D.NumDepto) AND
       (D.NomeDepto = 'Pesquisa');
```

Renomeando Relações

- para cada empregado, obter o último nome dos empregados e dos seus supervisores diretos:

```
SELECT E.UltimoNome, M.UltimoNome
FROM   Empregado AS E, Empregado AS M
WHERE  (E.NumSupervisor = M.NumEmpregado);
```

- É possível também "renomear" nomes de atributos.

```
...
FROM Empregado AS E(PN,IM,UN,Num,DN,End,Sexo,Sal,NumS,NumD)
...
```

Operadores aritméticos

- Outra característica da SQL é a permissão em se utilizar operadores aritméticos ('+', '-', '*', '/') nas consultas.

- Apresentar os salários resultantes de um aumento de 10% a todos os empregados do projeto 'Produto Y' :

```
SELECT  PrimeiroNome, UltimoNome, 1.1 * Salario
FROM    Empregado, Trabalha_em, Projeto
WHERE   (Empregado.NumEmpregado = Trabalha_em.NumEmpregado)
        AND (Projeto.NumProj = Trabalha_em.NumProj) AND
        (NomeProj = 'Produto Y');
```

Cláusula ORDER BY

- A SQL permite ordenar as tuplas no resultado de uma consulta pelos valores de um ou mais atributos, utilizando a cláusula ORDER BY.

```
SELECT  NomeDepto, PrimeiroNome, UltimoNome, NomeProj
FROM    Departamento, Empregado, Trabalha_em, Projeto
WHERE   (Empregado.NumDepto=Departamento.NumDepto) AND
        (Empregado.NumEmpregado=Trabalha_em.NumEmpregado)
        AND (Projeto.NumProj = Trabalha_em.NumProj)
ORDER BY NomeDepto, PrimeiroNome;
```

Operações de Conjunto

- A SQL incorporou as seguintes operações de conjunto da álgebra relacional:
  - união (UNION);
  - interseção (INTERSECT);
  - diferença (EXCEPT).
- Na operação de união, as tuplas duplicadas são eliminadas automaticamente.
  - Para manter as tuplas duplicadas, deve-se usar a cláusula UNION ALL.

Operações de Conjunto

```
SELECT DISTINCT  NumProj
FROM    Projeto, Departamento, Empregado
WHERE (Projeto.NumDepto=Departamento.NumDepto) AND
          (NumGerente=NumEmpregado) AND
          (UltimoNome= 'Silva')
UNION
SELECT DISTINCT NumProj
FROM    Trabalha_em, Empregado
WHERE (Empregado.NumEmpregado=Trabalha_em.NumEmpregado)
          AND (UltimoNome= 'Silva');
```

Junção entre Tabelas

- SQL permite especificar a condição da junção na cláusula FROM, usando a cláusulas JOIN (INNER JOIN) e ON.

- Obter o nome e o endereço dos empregados que trabalham no departamento 'Pesquisa':

```
SELECT PrimeiroNome, UltimoNome, Endereco
FROM   (Empregado JOIN Departamento ON
        Empregado.NumDepto=Departamento.NumDepto)
WHERE  NomeDepto= 'Pesquisa';
```

- Pode-se também utilizar a junção natural da álgebra:

```
SELECT PrimeiroNome, UltimoNome, Endereco
FROM   Empregado NATURAL JOIN Departamento
WHERE  NomeDepto= 'Pesquisa';
```

Junção entre Tabelas

- Em SQL, é possível utilizar também as operações de junção externa (OUTER JOIN) da álgebra relacional.

```
SELECT E.UltimoNome AS NomeEmp, M.UltimoNome AS NomeSup
FROM   (Empregado AS E LEFT OUTER JOIN Empregado AS M
        ON E.NumSupervisor = M.NumEmpregado);
```

- O exemplo exemplificou uma junção externa à esquerda (LEFT OUTER JOIN); mas existem também a junção externa à direita (RIGHT OUTER JOIN) e a junção externa completa (FULL OUTER JOIN).

Subconsultas

- Uma subconsulta é um bloco completo (SELECT ... FROM ... WHERE) que existe dentro da cláusula WHERE de uma outra consulta, chamada consulta externa.

```
SELECT DISTINCT  NumProj
FROM     Projeto
WHERE  NumProj IN ( SELECT NumProj
                    FROM  Projeto, Departamento, Empregado
                    WHERE (NumGerente=NumEmpregado) AND
                   (Projeto.NumDepto=Departamento.NumDepto)
                   AND (UltimoNome= 'Silva'))
                OR
                NumProj IN ( SELECT T.NumProj
                             FROM Trabalha_em AS T, Empregado AS E
                             WHERE (E.NumEmpregado=T.NumEmpregado)
                             AND (E.UltimoNome= 'Silva'));
```

O operador IN funciona como o operador '∈' da teoria de conjuntos: verifica se um valor pertence a um conjunto.

Operadores SOME e ALL

- Além do operador IN, existem outros operadores que são usados para comparar um valor 'v' com um conjunto 'V':
  - <operador lógico> SOME (ou ANY): retorna verdadeiro se 'v' é <operador lógico> que algum valor do conjunto 'V';
  - <operador lógico> ALL: retorna verdadeiro se 'v' é <operador lógico> que todos os valores do conjunto 'V'.
- Os operadores SOME e ALL precisam acompanhar algum operador lógico: =, <>, >, >=, <, <=.
- O operador '= SOME' é equivalente ao operador IN.

Operadores SOME e ALL

```
SELECT PrimeiroNome, UltimoNome
FROM   Empregado
WHERE Salario > ALL (SELECT Salario FROM Empregado
                     WHERE NumDepto=5);
```

```
SELECT E.PrimeiroNome, E.UltimoNome
FROM   Empregado AS E
WHERE E.Salario > SOME ( SELECT M.Salario
                         FROM Empregado AS M, Departamento AS D
                         WHERE (M.NumDepto=D.NumDepto) AND
                               (D.NomeDepto= 'Pesquisa'));
```

Conjuntos Explícitos de Valores

- Em SQL, é possível utilizar um conjunto explícito de valores na cláusula WHERE ao invés de uma subconsulta.

```
SELECT DISTINCT NumEmpregado
FROM    Trabalha_em
WHERE  NumProj IN (1, 2, 3);
```

Subconsultas Correlacionadas

- Sempre que uma condição na cláusula WHERE de uma subconsulta faz referência a algum atributo de uma relação declarada na consulta externa, diz-se que as duas consultas são correlacionadas.
  - Neste caso, a subconsulta é avaliada uma vez para cada tupla (ou combinação de tuplas) da consulta externa.

```
SELECT  E.PrimeiroNome, E.UltimoNome
FROM    Empregado AS E
WHERE   E.NumEmpregado IN (SELECT D.NumEmpregado
                           FROM Dependente AS D
                           WHERE E.PrimeiroNome = NomeDependente);
```

Função EXISTS

- A função EXISTS é utilizada para verificar se o resultado de uma subconsulta correlacionada é vazio ou não.

```
SELECT  E.PrimeiroNome, E.UltimoNome
FROM    Empregado AS E
WHERE   NOT EXISTS (SELECT * FROM Dependente AS D
                    WHERE E.NumEmpregado = D.NumEmpregado);
```

Função EXISTS

```
SELECT E.PrimeiroNome, E.UltimoNome
FROM   Empregado AS E
WHERE  EXISTS (SELECT * FROM Dependente AS D
                    WHERE E.NumEmpregado=D.NumEmpregado)
       AND
       EXISTS (SELECT * FROM Departamento
            WHERE E.NumEmpregado = NumGerente);
```

Agrupamento e Funções de Agregação

- A linguagem SQL incorpora os conceitos de agrupamento e funções de agregação definidos na álgebra relacional.

- As principais funções de agregação são:
  - COUNT: número de tuplas recuperadas em uma consulta;
  - SUM: soma dos valores de um atributo em uma consulta;
  - MAX: valor máximo de um atributo em uma consulta;
  - MIN: valor mínimo de um atributo em uma consulta;
  - AVG: média dos valores de um atributo em uma consulta.

Agrupamento e Funções de Agregação

- As funções de agregação podem ser usadas em uma cláusula SELECT ou em uma cláusula HAVING.
- A cláusula GROUP BY serve para agrupar tuplas que possuem o mesmo valor para os atributos relacionados em tal cláusula.
- A cláusula HAVING pode ser usada em conjunto com a cláusula GROUP BY para especificar uma condição de seleção a ser aplicada em cada grupo de tuplas recuperadas em uma consulta.
  - Somente os grupos que satisfizerem a condição serão retornados.

Agrupamento e Funções de Agregação

```
SELECT SUM(Salario), MAX(Salario), MIN(Salario), AVG(Salario)
FROM   (Empregado JOIN Departamento ON
        Empregado.NumDepto=Departamento.NumDepto)
WHERE  NomeDepto= 'Pesquisa';
```

```
SELECT COUNT(*)
FROM   Empregado AS E, Departamento AS D
WHERE  (E.NumDepto = D.NumDepto) AND
       (D.NomeDepto = 'Pesquisa');
```

Agrupamento e Funções de Agregação

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

-----------------------------------------------------------------------

SELECT  PrimeiroNome, UltimoNome
FROM    Empregado AS E
WHERE   (SELECT COUNT(*)
         FROM  Dependente AS D
         WHERE E.NumEmpregado=D.NumEmpregado) >= 2;
```

Agrupamento e Funções de Agregação

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

Agrupamento e Funções de Agregação

```
SELECT    P.NumProj, P.NomeProj, COUNT(*)
FROM      Projeto AS P, Trabalha_em AS T
WHERE     P.NumProj=T.NumProj
GROUP BY  P.NumProj, P.NomeProj;

-------------------------------------------------------------

SELECT    P.NumProj, P.NomeProj, COUNT(*)
FROM      Projeto AS P, Trabalha_em AS T
WHERE     P.NumProj=T.NumProj
GROUP BY  P.NumProj, P.NomeProj
HAVING    COUNT(*) > 2;
```

Agrupamento e Funções de Agregação

```
SELECT D.NumDepto, COUNT(*)
FROM   Empregado AS E, Departamento AS D
WHERE  E.NumDepto=D.NumDepto AND E.Salario>5000 AND
       E.NumDepto IN ( SELECT NumDepto
                       FROM  Empregado
                       GROUP BY  NumDepto
                       HAVING COUNT(*) > 5)
GROUP BY  D.NumDepto;
```

Comando INSERT

- Em sua forma mais simples, o comando INSERT é utilizado para adicionar uma única tupla a uma relação.
  - Deve-se especificar o nome da relação e uma lista de valores para a tupla.
  - Os valores da tupla devem estar relacionados na mesma ordem em que foram definidos no CREATE TABLE.

```
INSERT INTO Empregado
VALUES   ('Rosana', 'P', 'Souza', '653298653', '1962-12-30',
          'R.Alagoas, 1230', 'F', 3000, '987654321', 4);
```

Comando INSERT

- Uma segunda forma permite especificar nomes explícitos de atributos que correspondem aos valores fornecidos no comando INSERT.

```
INSERT INTO Empregado(PrimeiroNome, UltimoNome,
                      NumEmpregado, NumDepto)
VALUES    ('Rosana', 'Souza', '653298653', 4);
```

Comando INSERT

- Uma variação do comando INSERT inclui múltiplas tuplas numa relação, conjuntamente com a criação da relação e a carga da mesma com o resultado de uma consulta.

```
CREATE TABLE Info_Deptos
(   Nome_Depto    VARCHAR(15),
    Num_de_Emps INTEGER,
    Total_Sal       INTEGER );

INSERT INTO Info_Deptos
SELECT      NomeDepto, COUNT(*), SUM(Salario)
FROM        (Departamento JOIN Empregado ON
            Departamento.NumDepto=Empregado.NumDepto)
GROUP BY    NomeDepto;
```

Comando DELETE

- O comando DELETE remove tuplas de uma relação.
- Possui a cláusula WHERE para selecionar as tuplas a serem excluídas.
  - Quando não especificada, todas as tuplas da relação determinada são excluídas; entretanto, a tabela permanece no banco de dados como uma tabela vazia.
- As tuplas são explicitamente excluídas de uma só relação.
  - Entretanto, a exclusão pode se propagar para tuplas de outras relações de acordo com as restrições de integridade referencial definidas.

Comando DELETE

```
DELETE FROM Empregado
WHERE   UltimoNome = 'Silva';

DELETE FROM Empregado
WHERE   Salario < 800;

DELETE FROM Empregado
WHERE   NumDepto IN  (SELECT NumDepto
                      FROM Departamento
                      WHERE NomeDepto = 'Pesquisa');

DELETE FROM Empregado;
```

Comando UPDATE

- O comando UPDATE é utilizado para modificar valores de atributos de uma ou mais tuplas em uma única relação.
  - Cláusula WHERE para selecionar as tuplas a serem modificadas.
  - Cláusula SET para especificar os atributos a serem modificados e o seus novos valores.
  - A modificação do valor de uma chave primária pode se propagar para chaves estrangeiras de outras relações, de acordo com as restrições de integridade referencial definidas.

- Alterar a localização e o número do departamento controlador do projeto número 10 para 'Anchieta' e 1, respectivamente:

```
UPDATE  Projeto
SET     Localização = 'Anchieta', NumDepto = 1
WHERE   NumProj = 10;
```

Comando UPDATE

- Diversas tuplas podem ser modificadas com um único comando UPDATE.

- Aumentar em 10% os salários de todos os empregados do Departamento 'Pesquisa':

```
UPDATE   Empregado
SET      Salario = Salario * 1.1
WHERE    NumDepto IN  (SELECT NumDepto
                       FROM Departamento
                       WHERE  NomeDepto='Pesquisa');
```

Visões em SQL

- Uma visão em SQL é uma relação única que é derivada de outras relações ou de outras visões previamente definidas.
  - Uma visão não existe necessariamente na forma física: é considerada uma relação virtual, limitando assim, as operações de atualização que podem ser aplicadas.
  - Não há limitação na consulta de uma visão.

- O comando para especificar uma visão é CREATE VIEW.
  - Deve-se especificar o nome da visão, uma lista de nomes de atributos e uma consulta para especificar o conteúdo da visão.
  - Se nenhum dos atributos da visão resultar da aplicação de funções ou operações aritméticas, não há necessidade de especificar os nomes de atributos para a visão, já que seriam os mesmos nomes de atributos das relações definidoras da visão.

Visões em SQL

```
CREATE VIEW  V_Trabalha_em
AS SELECT  PrimeiroNome, UltimoNome, NomeProj, Horas
   FROM    Empregado AS E, Projeto AS P, Trabalha_em AS T
   WHERE   (E.NumEmpregado = T.NumEmpregado) AND
           (P.NumProj = T.NumProj);

CREATE VIEW  V_Info_Deptos (NomeDepto, QteEmps, TotalSal)
AS SELECT  NomeDepto, COUNT(*), SUM(Salario)
   FROM    (Departamento JOIN Empregado ON
           Departamento.NumDepto=Empregado.NumDepto)
   GROUP BY  NomeDepto;
```

Visões em SQL

- Uma vez definida uma visão, pode-se realizar consultas utilizando a mesma.

- Obter o primeiro e o último nome de todos os empregados que trabalham no projeto 'Projeto X':

```
SELECT PrimeiroNome, UltimoNome
FROM   V_Trabalha_em
WHERE  NomeProj = 'Projeto X';
```

- Uma das principais vantagens de uma visão é simplificar a especificação de certas consultas. As visões também são utilizadas como mecanismos de segurança e autorização.

Visões em SQL

- Uma visão está sempre atualizada, pois não efetua alguma ação no momento em que é definida, e sim no momento em que se especifica uma consulta utilizando-a.

- Para remover uma visão, utiliza-se o comando DROP VIEW.

DROP VIEW V_Trabalha_em;

Visões em SQL

- Com relação à atualização de dados das relações definidoras de uma visão, a partir da mesma, pode-se dizer que:
  - uma visão definida a partir de uma única relação é atualizável se os atributos da visão contiverem a chave primária (ou possivelmente alguma outra chave candidata) de tal relação, pois isso mapeia cada tupla da visão para uma única tupla da relação definidora;
  - uma visão definida a partir de múltiplas relações utilizando junções geralmente não é atualizável, pois a atualização pela visão deve ser mapeada em diversas atualizações nas relações definidoras para fornecer o efeito desejado;
  - uma visão definida com o uso de agrupamento e funções agregadas não é atualizável, pois alguns atributos não existem nas relações definidoras.

Conexão entre PostgreSQL e C++

```
#include <iostream>
#include "libpq-fe.h" //biblioteca necessária
using namespace std;

int main()
{
   int nTuplas, nAtributos;

   //declara uma variavel logica que representara o
   //o banco de dados
   PGconn *conn = NULL;

   //Cria uma conexao com um banco do PostgreSQL
   conn = Pqconnectdb ("host=localhost port=5432
      dbname=empresa user=postgres password=postgres");
```

Conexão entre PostgreSQL e C++

```
//Verifica a conexao feita
if (PQstatus(conn) != CONNECTION_OK)
{
   //Retorna uma mensagem de erro do PostgreSQL
   cout << PQerrorMessage(conn);

   //Encerra a conexao com o banco de dados,
   //liberando a memoria
   PQfinish(conn);
   return 1;
}
else
   cout << "Conexao executada com sucesso!" << endl;
```

Conexão entre PostgreSQL e C++

```
//Declara um ponteiro para resultado de uma consulta
PGresult *res;

//Faz uma consulta ao banco de dados
res = PQexec(conn, "select * from funcionario;");

//Verifica a validade da consulta
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
   cout << PQerrorMessage(conn);
   PQclear(res);
   PQfinish(conn);
   return 2;
}
```

Conexão entre PostgreSQL e C++

```
// Obtem o numero de tuplas
nTuplas = PQntuples(res);
// Obtem o numero de atributos
nAtributos = PQnfields(res);

// Percorre todos as tuplas
for (int i = 0; i < nTuplas; i++)
{
   // Percorre todos os atributos
   for (int j = 0; j < nAtributos; j++)
   {
      // Imprime o valor do atributo j da tupla i
      cout << PQgetvalue(res, i, j) << "\t";
   }
   cout << endl;
}
```

Conexão entre PostgreSQL e C++

```
    // Fecha o acesso ao resultado da consulta
    PQclear(res);

    // Fecha a conexao com o banco de dados
    PQfinish(conn);

}
```

Conexão entre PostgreSQL e C++

Compilação:

```
g++ -o exemplo -I /usr/include/postgresql/ exemplo.cpp
-L /usr/lib/ -lpq
```

Considerando que:

- o SGBD eh o PostgreSQL 9.2;
- o arquivo libpq-fe.h encontra-se no diretorio "/usr/include/postgresql";
- a biblioteca libpq encontra-se no diretorio "/usr/lib".