Universidade Federal
de Campina Grande

Banco de Dados I
Unidade 5: A linguagem SQL - DML - Parte 2

Prof. Cláudio de Souza Baptista, Ph.D.
Laboratório de Sistemas de Informação – LSI
UFCG

SQL-DML

- Esquemas do BD Empresa:
  - **Empregado**(matricula, nome, endereco, salario, supervisor, depto)
  - **Departamento**(coddep, nome, gerente, dataini)
  - **Projeto**(codproj, nome, local, depart)
  - **Alocacao**(matric,codproj, horas)

SQL-DML - Exemplos

Quantificadores

- `ANY` (ou `SOME`) e `ALL` (ou `EVERY`) comportam-se como quantificadores existencial ("ao menos um") e universal, respectivamente.

- **Exemplo**

```
SELECT mat, salario
FROM empregado
WHERE salario >= all
        (SELECT salario FROM empregado)
```

Definição de ALL

f <comp> all R $\Leftrightarrow \forall\, t \in R$ (f <comp> *t*)

onde <comp> pode ser : <, ≤, >, =, ≠

(5< all | 0, 5, 6) = false

(5< all | 6, 10) = true

(5 = all | 4, 5) = false

(5 ≠ all | 4, 6) = true (pois 5 ≠ 4 and 5 ≠ 6)

Definição de ANY (SOME)

f <comp> ANY R $\Leftrightarrow \exists\, t \in R$ **(*f* <comp> *t*)**

onde <comp> pode ser : $<, \leq, >, =, \neq$

(5< any

0
5
6

) = true

(lê-se: 5 < alguma tupla na relação)

(5< any

0
5

) = false

(5 = any

0
5

) = true

(5 ≠ any

0
5

) = true (uma vez que 0 ≠ 5)

SQL-DML - Exemplos

Quantificadores

- Exemplo com agrupamento
  - Quais departamentos têm mais empregados?

```
SELECT depto
FROM empregado
GROUP BY depto
HAVING COUNT(*) >= ALL
                    (SELECT COUNT(*)
                     FROM empregado GROUP BY depto)
```

SQL-DML - Exemplos

Quantificadores

- Exemplo com agrupamento
  - Quais empregados não ganham o menor salário pago pela empresa?

```
SELECT mat
FROM empregado
WHERE salario > ANY
        (SELECT salario FROM empregado)
```

Junção em SQL:1999

- Vimos como fazer junção em SQL-92. O padrão SQL:1999 (e o 92) especifica vários tipos de junção:
  - Clássica (tabelas separadas por vírgulas como vimos)
  - cross-joins
  - natural joins
  - conditions joins
  - column name join
  - outer joins (left, right, ou full)

Junções

- Exemplos: Natural Join
- Sejam as tabelas T1 e T2

T1

C1	C2
10	15
20	25

T2

C1	C4
10	BB
15	DD

Junção Natural de T1 com T2

C1	C2	C4
10	15	BB

```
SELECT *
FROM T1 NATURAL JOIN T2
```

- OBS: a junção será feita por colunas de mesmo nome

Junções

- **Exemplos:** Cross Join
- Implementa o produto cartesiano

```
SELECT *
FROM T1 CROSS JOIN T2
```

Junções

- **Exemplos:** Condition Join
  - usa a cláusula ON para especificar a condição de junção

```
SELECT *
FROM T1 JOIN T2
ON T1.C1 = T2.C1
```

é equivalente a:

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
```

Junções

- **Exemplos:** Outer Join
  - Preserva no resultado valores que não casam na comparação da junção
  - Motivação: às vezes precisamos mostrar estes valores que não casam
  - **ex.** Tabelas empregado e departamento onde o código do departamento em empregado é chave estrangeira, portanto, pode haver valores nulos. Se quisermos uma lista de todos os empregados com os nomes dos respectivos departamentos, usando uma junção natural eliminaria os empregados sem departamento (com valores null)

Junções

- **Exemplos:** Left Outer Join

T1

C1	C2
10	15
20	25

T2

C3	C4
10	BB
15	DD

Junção left outer de T1 com T2

C1	C2	C3	C4
10	15	10	BB
20	25	Null	Null

```
SELECT *
FROM T1 LEFT OUTER JOIN T2
ON T1.C1 = T2.C3
```

Junções

- **Exemplos:** Right Outer Join

T1

C1	C2
10	15
20	25

T2

C3	C4
10	BB
15	DD

Junção right outer de T1 com T2

C1	C2	C3	C4
10	15	10	BB
Null	Null	15	DD

```
SELECT *
FROM T1 RIGHT OUTER JOIN T2
ON T1.C1 = T2.C3
```

Junções

- **Exemplos:** Full Outer Join

T1

C1	C2
10	15
20	25

T2

C3	C4
10	BB
15	DD

Junção full outer de T1 com T2

C1	C2	C3	C4
10	15	10	BB
20	25	Null	Null
Null	Null	15	DD

```
SELECT *
FROM T1 FULL OUTER JOIN T2
ON T1.C1 = T2.C3
```

A Cláusula WITH

- Permite visões serem definidas localmente a uma query, ao invés de globalmente como veremos adiante.

Ex.: Mostre os funcionários que ganham o maior salário

```
WITH max_sal_temp(sal) as
            SELECT MAX(salario)
            FROM empregado
SELECT mat
FROM    empregado e, max_sal_temp m
WHERE e.salario = m.sal
```

Relações Diversas

No SQL:1999

```
SELECT depto
FROM
        (SELECT depto, AVG(salario)
        FROM empregado
        GROUP BY depto) resultado(depto, media)
WHERE media > 100;
```

No Oracle:

```
SELECT dep
FROM
        (SELECT depto as dep, AVG(salario) as
media
        FROM empregado
        GROUP BY depto) Resultado
WHERE Resultado.media > 100;
```

- **O comando INSERT**
  - Usado para adicionar uma tupla a uma relação
  - Sintaxe:

```
INSERT INTO tabela [ (lista colunas) ] fonte
```

  - Onde fonte pode ser uma especificação de pesquisa (SELECT) ou uma cláusula VALUES da forma:

```
VALUES (lista de valores atômicos)
```

OBS.: Se o comando INSERT incluir a cláusula VALUES então uma única tupla é inserida na relação.

Ex.create table empregado (mat int, nome char(50), end char(150), salario number(9,2), depto int)

```
INSERT INTO Empregado(mat, nome) VALUES(9491,1010 'Ana');
INSERT INTO Empregado VALUES (1000, 'Ana Silva', null, 8740, null)
```

SQL–DML - Exemplos

Obs.: A inserção será rejeitada se tentarmos omitir um atributo que não permite valores nulos (NOT NULL)

Ex.:

```
INSERT INTO Empregado (nome, salario) VALUES ('Flávia', 960);
```

- Podemos inserir várias tuplas numa relação através de uma query.

SQL–DML - Exemplos

Exemplo:

```
CREATE TABLE DEPTO_INFO
        (nome character(15),
        numemp integer,
        totsal real);

INSERT INTO DEPTO_INFO(nome, numemp, totsal)
        SELECT d.nome, COUNT(*), SUM(salario)
        FROM Departamento d JOIN Empregado e
        ON d.coddep = e.depto
        GROUP BY d.nome
```

SQL–DML - Exemplos

- **O comando DELETE**
  - Remove tuplas de uma relação
  - Sintaxe:

```
DELETE
FROM nome da
tabela
[WHERE condição]
```

```
DELETE
FROM Estudante
WHERE sexo = 'M'
```

Obs.: Se omitirmos a cláusula WHERE, então o DELETE deve ser aplicado a todas as tuplas da relação. Porém, a relação permanece no BD como uma relação vazia.

SQL–DML - Exemplos

- **O comando UPDATE**
  - Modifica o valor de atributos de uma ou mais tuplas.
  - Sintaxe:

```
UPDATE tabela
SET lista_atributos com atribuições de valores
[WHERE condição]
```

Obs.: omitir a cláusula WHERE implica que o UPDATE deve ser aplicado a todas as tuplas da relação

SQL–DML - Exemplos

- **O comando UPDATE**

  Ex. Modifique o nome do Departamento de Computação para Departamento de Informática, e o telefone

```
UPDATE Departamento
SET nome='Informatica',
      tel = 333222
WHERE nome='Computação'
```

OBS.: se houver mais de um atributos a serem alterados, os separamos por vírgula (,) na cláusula SET

SQL–DML - Exemplos

- **O comando UPDATE**

  Ex. Dê um aumento de 10% a todos os empregados do departamento de Pesquisa

```
UPDATE Empregado
SET SALARIO = 1.1*SALARIO
WHERE depto in (SELECT coddep
                FROM Departamento
                WHERE nome='Pesquisa')
```

SQL–DML - Exemplos

- **O comando CASE**
  - Permite mudar o valor de um dado, por exemplo, poderiamo ter codificado o atributo sexo como 1 = masculino, 2 = feminino, 0 = indefinido , e então ao fazermos um select queremos expressar os valores por extenso ao invés de usar código.

```
SELECT mat, nome,
        CASE
                WHEN sexo=1 THEN 'Masculino'
                WHEN sexo=2 THEN 'Feminino'
                WHEN sexo=0 THEN 'Indefinido'
        END,
        endereco, salario
FROM Empregado
```

Visões (Views)

Visões

- Não é desejável que todos os usuários tenham acesso a todo o esquema => visões precisam ser definidas.
- **Visão:** é uma relação virtual que não faz parte do esquema conceitual mas que é visível a um grupo de usuários.
- A visão é definida por uma DDL e é computada cada vez que são realizadas consultas aos dados daquela visão.
- O catálogo do SGBD é o repositório que armazena as definições das visões.
- Uma visão possui nome, uma lista de atributos e uma query que computa a visão.

Visões

- Uma visão é uma tabela virtual que é definida a partir de outras tabelas, contendo sempre os dados atualizados.

- Visão em SQL:
  - Sintaxe:

```
CREATE VIEW nomeVisão AS expressão_de_consulta
```

  - Exemplo:

```
CREATE VIEW Alocacao1(nomeE, nomeP, Horas)
        AS SELECT e.nome, p.nome, horas
            FROM Empregado e, Projeto p, Alocacao a
            WHERE e.matricula = a.matricula and
                  p.codproj = a.codproj
```

  - Cria uma relação virtual **Alocacao1(nomeE, nomeP, horas)**

Visões

Podemos escrever consultas na visão definida.

- **Ex.:** Obter o nome dos empregados que trabalham no projeto ‘Informatização’

```
SELECT nomeE
FROM Alocacao1
WHERE nomeP = ‘Informatizacao’
```

Visões

- **Ex.2:** Criar uma visão que contém informações gerenciais sobre um departamento, contendo o nome do depto, total de empregados e total de salários.

```
CREATE VIEW InfoDepto (nome, numemp, totsal)
        AS SELECT D.nome, COUNT(*), SUM(salario)
            FROM Departamento d, Empregado e
            WHERE d.coddep = e.depto GROUP BY d.nome
```

Visões

- **Eliminando uma visão**
  - Usamos o comando DROP VIEW
  - Sintaxe:

```
DROP VIEW nomeVisão
```

  - Ex.:

```
DROP VIEW Alocacao1
```

```
DROP VIEW InfoDepto
```

Visões

- Atualizando uma visão
  - Para ilustrarmos alguns problemas, considere a visão Alocacao1 e suponha que queiramos atualizar o atributo nomeP da tupla que contém ‘João’ de ‘ProdutoX’ para ‘Produto Y’.
  - Esta atualização de visão é expressa da seguinte forma:

```
UPDATE Alocacao1
SET nomeP = ‘ProdutoY’
WHERE nomeE = ‘João’ and nomeP = ‘ProdutoX’
```

Visões

- O update anterior pode ser mapeado em vários updates nas relações base. Dois possíveis updates, com resultados diferentes são:

```
UPDATE Alocacao
SET codproj = (SELECT codproj FROM Projeto
                    WHERE nome = 'ProdutoY')
WHERE matricula IN (SELECT matricula FROM Empregado
                    WHERE nome = 'João')
      AND codproj IN (SELECT codproj FROM Projeto
                      WHERE nome = 'ProdutoX')
```

ou

```
UPDATE Projeto
SET nome = 'ProdutoX'
WHERE nome = 'ProdutoY'
```

=> Como o SGBD vai escolher qual UPDATE computar?

Visões

- Considere a visão alocação1 se tentarmos fazer:

```
INSERT INTO Alocacao1
VALUES ('José', 'SIG', 10)
```

Visões

- Outro problema em update de visão: suponha a seguinte visão

```
CREATE VIEW Emp2
        AS SELECT mat, nome, dataNasc
            FROM Empregado
            WHERE depto = 1
```

  - O que aconteceria se fizéssemos:

```
INSERT INTO Emp2 VALUES (100, 'Ana', '1978/10/02')
```

⇒ depto terá valor nulo, portanto o que acontece com

SELECT * FROM emp2 ?

Visões

- Alguns updates de visões não fazem sentido para relação base.

Ex.:

```
UPDATE InfoDepto
SET totsal = 10.000
WHERE nomed = ‘Pesquisa’
```

Visões

- Observações:

1) Uma visão definida numa única tabela é atualizável se os atributos da visão contêm a chave primária.

2) Visões definidas sobre múltiplas tabelas usando junção geralmente não são atualizáveis

3) Visões usando funções de agrupamento e agregados não são atualizáveis.

Valores Nulos

- Interpretação de um valor nulo:
  - o atributo não se aplica a tupla
  - o valor do atributo para esta tupla é desconhecido
  - o valor é conhecido, mas está ausente (não foi posto ainda)

- Problemas com valores nulos:
  - problemas com junções (informações são perdidas)
  - problemas com funções aritméticas, pois os resultados com nulos serão sempre nulos.
  - O count(*) conta a linhas com null, mas count(coluna) conta apenas os não null

Valores Nulos

Lógica de Nulls

- Terceiro valor booleano `DESCONHECIDO`.

- Uma consulta somente produz valores se a condição da cláusula `WHERE` for `VERDADE` (`DESCONHECIDO` não é suficiente).

Valores Nulos

Cuidado:

- Se x é um atributo inteiro com valor null:

  x * 0 = NULL

  x - x = NULL

  x + 3 = NULL

- Quando comparamos um valor nulo com outro valor nulo usando um operador relacional o resultado é DESCONHECIDO!

  x = 3 => DESCONHECIDO (unknown)

  x > 2 => DESCONHECIDO (unknown)

Valores Nulos

Lógica de três valores:

verdade = 1; falso = 0, e desconhecido = 1/2.

Então:

- AND = min.
- OR = max.
- NOT(x) = 1 – x.

Algumas Leis não Funcionam

Exemplo: *p* `OR NOT` *p* `= verdade`

- Para a lógica dos 3-valores: se *p* = desc., então lado esquerdo = max(1/2,(1–1/2)) = 1/2 ≠ 1.

Valores Nulos

Ex.: seja a tabela

Bar	Cerveja	Preço
Rubronegro	Carlsberg	Null

```
SELECT bar
FROM Vende
WHERE preço < 2,00 OR preço >= 2,00
```

DESCONHECIDO DESCONHECIDO

DESCONHECIDO

- O bar Rubronegro não é selecionado, mesmo se a cláusula `WHERE` é uma tautologia.

NULL representa valores UNKNOWN

- Uma operação aritmética que envolve NULL, retorna NULL.
  - coluna minus NULL → NULL
    (e não zero)
- Comparação booleana com NULL retorna UNKNOWN
  (e não true/false)
  - NULL = NULL → UNKNOWN
  - valor {<>,>,<,>=,<=} NULL → UNKNOWN
- Testar se um valor é NULL
  - where valor **IS NULL**
- Testar se um valor não é NULL
  - where valor **IS NOT NULL**

- SQL SELECT retorna valores para consultas cujo WHERE seja TRUE
- SQL GROUP retorna grupos para consultas cujo HAVING seja TRUE
- A função de agregação COUNT(*) conta todas as tuplas
- Já COUNT(coluna) conta as tuplas cujo valor para coluna não seja NULL
- Outras funções de agregação ignoram valores NULL