

MAC0439 - Laboratório de Bancos de Dados

Aula 11 - SQL

Consultas Mais Avançadas

e

Operações de Modificação do Banco de Dados

Inserção, Remoção e Alteração

16 de setembro de 2016

Profa. Kelly Rosa Braghetto

(Adaptação dos slides do prof. Jeffrey Ullman, da *Stanford University*)

Exemplo para a aula

- ◆ Todas as nossas consultas SQL serão baseadas no seguinte esquema de BD:

Refrigerante(nome, fabricante)

Lanchonete(nome, endereco, cnpj)

Cliente(nome, endereco, telefone)

Apreciador(nome_cliente, nome_refri)

Venda(nome_lanch, nome_refri, preco)

Frequentador(nome_cliente, nome_lanch)

Subconsultas

- ◆ Um comando SELECT-FROM-WHERE parentizado (= *subconsulta*) pode ser usado como um valor em diferentes lugares de uma consulta, incluindo nas cláusulas FROM e WHERE.
- ◆ **Exemplo:** no lugar de uma relação na cláusula FROM, nós podemos usar uma subconsulta, e então consultar o seu resultado.
 - ▶ Para isso, faz-se necessário o uso de uma variável-tupla para nomear as tuplas do resultado.

Exemplo: uma subconsulta no FROM

- ◆ Encontre os refrigerantes apreciados por pelo menos uma pessoa que frequenta o Sujinhos.

```
SELECT nome_refri
FROM Apreciador, (SELECT nome_cliente
                  FROM Frequentador WHERE
                  nome_lanch = 'Sujinhos') SC
WHERE Apreciador.nome_cliente =
      SC.nome_cliente;
```

Clientes que frequentam o Sujinhos

Subconsultas que devolvem uma tupla

- ◆ Se uma subconsulta garantidamente produz uma única tupla, então a subconsulta pode ser usada como um valor.
 - ▶ Geralmente, a tupla tem um único componente.
 - ▶ Um erro é gerado em tempo de execução se não há nenhuma tupla no resultado ou se o resultado contém mais do que uma tupla.

Exemplo: subconsulta com tupla única

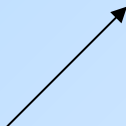
- ◆ Usando `Venda(nome_lanch, nome_refri, preco)`, encontre as lanchonetes que servem Fanfa pelo mesmo preço que o Sujinhos cobra pela Sprife.
- ◆ A combinação de duas consultas certamente resolve a questão:
 1. Encontre o preço da Sprife no Sujinhos.
 2. Encontre as lanchonetes que vendem Fanfa por esse preço.

Solução com consulta + subconsulta

```
SELECT nome_lanch  
FROM Venda  
WHERE nome_refri = 'Fanfa' AND  
preco = (SELECT preco
```

```
FROM Venda  
WHERE nome_lanch = 'Sujinhos'  
AND nome_refri = 'Sprife');
```

Preço da
Sprife no
Sujinhos



O operador IN

- ◆ A expressão

`<tupla> IN (<subconsulta>)`

é verdadeira se e somente se a tupla é membro da relação produzida pela subconsulta.

- ◆ Oposto:

`<tupla> NOT IN (<subconsulta>).`

- ◆ Expressões com IN podem aparecer na cláusula WHERE.

Exemplo: IN

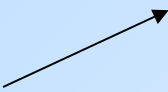
- ◆ Usando `Refrigerante(nome, fabricante)` e `Apreciador(nome_cliente, nome_refri)`, encontre o nome e o fabricante de cada refri que o Fred gosta.

```
SELECT *
```

```
FROM Refrigerante
```

```
WHERE nome IN (SELECT nome_refri  
FROM Appreciador  
WHERE nome_cliente = 'Fred');
```

Conjunto de
refris que o
Fred gosta



Estas consultas são equivalentes?

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

a	b
1	2
3	4

R

b	c
2	5
2	6

S

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

IN é um predicado sobre as tuplas de R

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Laço sobre as tuplas de S

Laço sobre as tuplas de R

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) satisfaz
a condição;
1 aparece uma
vez no resultado.

Esta consulta “pareia” tuplas de R e S

```
SELECT a  
FROM R, S  
WHERE R.b = S.b;
```

Laço duplo sobre
as tuplas de R e S

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) com (2,5) e
(1,2) com (2,6) -
ambos pares
satisfazem a
condição; 1 é
incluído na
resposta 2 vezes!

O operador EXISTS

- ◆ A expressão EXISTS(<subconsulta>) é verdadeira se e somente se o resultado da subconsulta não é vazio.
- ◆ **Exemplo:** A partir de **Refrigerante(nome, fabricante)**, encontre os refris que são os únicos fabricados por seus fabricantes.

Exemplo: EXISTS

```
SELECT nome  
FROM Refrigerante r1  
WHERE NOT EXISTS (
```

Observe a regra do escopo:
fabricante se refere à
relação na cláusula FROM
mais próxima que possua
o atributo.

Cjto de refris
com o mesmo
fabricante de
r1, mas que
não é o mesmo refri.

```
SELECT *  
FROM Refrigerante  
WHERE fabricante =  
r1.fabricante AND  
nome <> r1.nome);
```

Operador de
“diferente”
da SQL

O operador ANY

- ◆ $x = \text{ANY}(\langle \text{subconsulta} \rangle)$ é uma condição booleana que é verdadeira sse x é igual a pelo menos uma tupla no resultado do subconjunto.
 - ◆ “=” pode ser substituído por qualquer operador de comparação.
- ◆ **Exemplo:** $x \geq \text{ANY}(\langle \text{subconsulta} \rangle)$ significa que x não é sozinha a menor tupla produzida pela subconsulta.
 - ◆ Observe que as tuplas resultantes na subconsulta precisam possuir um único componente.

O operador ALL

- ◆ $x \langle \rangle \text{ALL}(\langle \text{subconsulta} \rangle)$ é verdadeira sse para toda tupla t no resultado da subconsulta, x não é igual a t .
 - ◆ Ou seja, x não está no resultado da subconsulta.
- ◆ “ $\langle \rangle$ ” pode ser substituído por qualquer operador de comparação.
- ◆ **Exemplo:** $x \geq \text{ALL}(\langle \text{subconsulta} \rangle)$ significa que não há no resultado da subconsulta tupla maior do que x .

Exemplo: ALL

- ◆ A partir de Venda(nome_lanch, nome_refri, preco), encontre o(s) refri(s) vendidos pelo maior preço.

```
SELECT nome_refri
FROM Venda
WHERE preco >= ALL(
    SELECT preco
    FROM Venda);
```

preco de Venda mais "externo" não pode ser menor do que qualquer outro preco.

Modificações no banco de dados

- ◆ Um comando de **modificação** não devolve um (multi)conjunto de tuplas como resultado (como uma consulta faz); ele **modifica o estado do BD de alguma forma**
- ◆ Existem 3 tipos de modificações:
 1. **Inserção** de tupla(s)
 2. **Remoção** de tupla(s)
 3. **Modificação** do(s) valor(es) dos componentes de tupla(s) existente(s)

Inserção

- ◆ Para inserir uma única tupla:

```
INSERT INTO <relação>  
VALUES ( <lista de valores> );
```

- ◆ **Exemplo:** adicione a `Apreciador(nome_cliente, nome_refri)` o fato de que Ana gosta de Fanfa.

```
INSERT INTO Appreciador  
VALUES ( 'Ana' , 'Fanfa' );
```

Especificando atributos no INSERT

- ◆ Nós podemos adicionar ao nome da relação uma lista de atributos.
- ◆ Há duas razões para se fazer isso:
 1. Quando esquecemos a ordem de criação dos atributos na relação.
 2. Quando não temos valores para todos os atributos e queremos que o SGBD preencha os componentes faltantes com NULL ou um valor *default*.

Exemplo: especificando atributos

- ◆ Outra forma de adicionar o fato de que Ana gosta de Fanfa a `Apreciador(nome_cliente, nome_refri)`:

```
INSERT INTO
    Apreciador(nome_refri, nome_cliente)
VALUES ('Fanfa', 'Ana');
```

Adicionando valores padrão

- ◆ Em um comando CREATE TABLE, podemos indicar um valor padrão para um atributo, por meio da cláusula DEFAULT.
- ◆ Quando uma tupla inserida não possui valor para esse atributo, o valor padrão será usado.

Exemplo: valores padrão

```
CREATE TABLE Cliente (  
    nome CHAR(30) PRIMARY KEY,  
    endereco CHAR(50)  
        DEFAULT 'Av. Paulista, 123',  
    telefone CHAR(16)  
);
```

Exemplo: valores padrão

```
INSERT INTO Cliente(nome)  
VALUES ('Ana');
```

Tupla resultante:

nome	endereco	telefone
Ana	Av. Paulista 123	NULL

Inserção de várias tuplas

- ◆ Podemos inserir o resultado todo de uma consulta em uma relação usando a forma:

```
INSERT INTO <relação>  
( <subconsulta> );
```

Exemplo: inserção de subconsultas

- ◆ Usando `Frequentador(nome_cliente, nome_refri)`, insira em uma nova relação `Colegas(nome)` todos os colegas “em potencial” da Ana, i.e., os clientes que frequentam pelo menos uma lanchonete frequentada pela Ana.

Nome dos
potenciais colegas

Solução

Pares de tuplas de Clientes onde a primeira é para Ana e a segunda é para um outro cliente qualquer, e as lanchonetes são a mesma.

```
INSERT INTO Colegas  
(SELECT f2.nome_cliente  
FROM Frequentador f1, Frequentador f2  
WHERE f1.nome_cliente = 'Ana' AND  
f2.nome_cliente <> 'Ana' AND  
f1.nome_lanch = f2.nome_lanch  
);
```

Remoção

- ◆ Para remover tuplas que satisfazem uma condição em uma relação:

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

Exemplo: remoção

- ◆ Remova de `Apreciador(nome_cliente, nome_refri)` o fato de que Ana gosta de Fanfa:

```
DELETE FROM Appreciador
```

```
WHERE nome_cliente = 'Ana'  
      AND nome_refri = 'Fanfa';
```

Exemplo: remoção de todas as tuplas

- ◆ Esvazie a relação *Apreciador*:

```
DELETE FROM Apreciador;
```

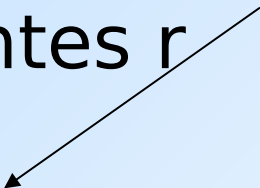
- ◆ Observe que nenhuma cláusula *WHERE* é necessária.

Exemplo: remoção de algumas tuplas

- ◆ Remova de **Refrigerante(nome, fabricante)** todos os refris para os quais existe um outro refri feito pelo mesmo fabricante.

```
DELETE FROM Refrigerantes r  
WHERE EXISTS (
```

Refris com o mesmo
Fabricante e um nome
diferente do nome
do refri representado
pela tupla r.



```
SELECT nome FROM Refrigerante  
WHERE fabricante = r.fabricante AND  
nome <> r.nome);
```

Semântica do comando de remoção do slide anterior (1)

- ◆ Suponha que a Cola-Coca produza somente Fanfa e Fanfa Diet.
- ◆ Suponha que o processamento da remoção “passe” primeiro pela tupla da Fanfa.
 - ▶ A subconsulta é não vazia, por causa da tupla da Fanfa Diet, então remove-se a tupla da Fanfa.
- ◆ Agora, quando r é a tupla para Fanfa Diet, essa tupla será removida também?
 - ▶ Essa dúvida é válida já que, após a remoção da tupla de Fanfa, Fanfa Diet passaria a ser o único refri de seu fabricante.

Semântica do comando de remoção do slide anterior (2)

- ◆ **Resposta:** Fanfa Diet será removida também!
- ◆ A razão para isso é o fato de que a remoção acontece em dois estágios:
 1. Marcação de todas as tuplas para as quais a condição WHERE é satisfeita.
 2. Remoção das tuplas marcadas.

Alterações

- ◆ Para mudar alguns atributos em algumas tuplas de uma relação:

UPDATE <relação>

SET <lista de atribuições a atributos>

WHERE <condição sobre as tuplas>;

Assim como no comando DELETE, a cláusula WHERE é opcional.

Exemplo: alteração

- ◆ Mude o número do telefone do cliente Mário para 555-1212:

```
UPDATE Cliente  
SET telefone = '555-1212'  
WHERE nome = 'Mário';
```

Exemplo: modificação de várias tuplas

- ◆ Faça com que R\$4 seja o preço máximo para um refrigerante:

```
UPDATE Venda  
SET preco = 4.00  
WHERE preco > 4.00;
```

Aggregações

- ◆ **SUM, AVG, COUNT, MIN e MAX** podem ser aplicados a uma coluna na cláusula **SELECT** para produzir a agregação da referida coluna.
- ◆ Além disso, **COUNT(*)** conta o número de tuplas.

Exemplo: Agregação

- ◆ A partir de `Venda(nome_lanch, nome_refri, preço)`, encontre o preço médio de Fanfa:

```
SELECT AVG(preço)
```

```
FROM Venda
```

```
WHERE nome_refri = 'Fanfa';
```

Eliminando duplicações em uma agregação

- ◆ Use DISTINCT dentro de uma agregação.
- ◆ **Exemplo:** encontre o número de preços *diferentes* cobrados pela Fanfa:

```
SELECT COUNT(DISTINCT preço)
FROM Venda
WHERE nome_refri = 'Fanfa';
```

Valores NULL são ignorados na agregação

- ◆ Um NULL nunca contribui para uma soma, média ou contagem, e nunca pode ser nem o mínimo, nem o máximo de uma coluna.
- ◆ Mas se não existir valores não nulos em uma coluna, então o resultado da agregação é NULL.
 - ▶ **Exceção:** COUNT de um conjunto vazio é 0.

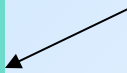
Exemplo: efeito de NULLs

```
SELECT count(*)  
FROM Venda WHERE  
nome_refri = 'Fanfa';
```

O número de lanchonetes que vendem Fanfa.

```
SELECT count(preço)  
FROM Venda WHERE  
nome_refri = 'Fanfa';
```

O número de lanchonetes que vendem Fanfa a um preço conhecido (ou seja, diferente de NULL).



Agrupamento

- ◆ Depois de uma expressão SELECT-FROM-WHERE, podemos adicionar **GROUP BY** e uma lista de atributos.
- ◆ A relação resultante do SELECT-FROM-WHERE com GROUP BY é agrupada de acordo com os valores de todos os referidos atributos e qualquer agregação é aplicada somente dentro de cada grupo.

Exemplo: agrupamento

◆ A partir de

Venda(nome_lanch, nome_refri, preço),
encontre o preço médio de cada refri:

```
SELECT nome_refri, AVG(preço)
```

```
FROM Venda
```

```
GROUP BY nome_refri;
```

nome_refri	AVG(preço)
Fanfa	2.33
...	...

Exemplo: agrupamento

- ◆ A partir de `Venda(nome_lanch, nome_refri, preço)` e `Frequentador(nome_cliente, nome_lanch)`, encontre, para cada cliente, o preço médio da Fanfa nas lanchonetes que ele frequenta:

```
SELECT nome_cliente, AVG(preço)
FROM Frequentador, Venda
WHERE nome_refri = 'Fanfa' AND
Frequentador.nome_lanch =
                Venda.nome_lanch
GROUP BY nome_cliente;
```

Computa todas as tuplas cliente-lanch-preço para Fanfa.

Depois, as agrupa por cliente.

Restrição no SELECT: listas com agregação

- ◆ Se um agrupamento é usado, então cada elemento da lista do SELECT precisa ser:
 1. Uma agregação, ou
 2. Um atributo da lista do GROUP BY.

Exemplo de consulta incorreta

- ◆ Alguém pode pensar que é possível encontrar a lanchonete que vende Fanfa mais barato usando:

```
SELECT nome_lanch, MIN(preço)  
FROM Venda  
WHERE nome_refri = 'Fanfa';
```

- ◆ Mas essa consulta **NÃO** é permitida em SQL.

Cláusulas **HAVING**

- ◆ **HAVING** <condição> pode aparecer depois da cláusula **GROUP BY**.
- ◆ Se aparecer, a condição é aplicada sobre cada grupo. Grupos que não satisfazem a condição são eliminados da resposta da consulta.

Exemplo: HAVING

- ◆ A partir de `Venda(nome_lanch, nome_refri, preço)` e `Refrigerante(nome, fabricante)`, encontre o preço médio dos refri que são servidos em pelo menos 3 lanchonetes ou que são fabricados pela Cola-Coca.

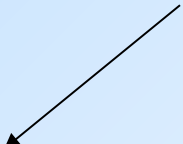
Solução

```
SELECT nome_refri, AVG(preço)  
FROM Venda  
GROUP BY nome_refri
```


```
HAVING COUNT(nome_lanch) >= 3 OR  
nome_refri IN
```

```
(SELECT nome  
FROM Refrigerantes  
WHERE fabricante = 'Cola-Coca');
```

Grupos de refri com pelo menos 3 lanchonetes não nulas e também grupos em que o fabricante é a Cola-Coca.



Refris fabricados pela Cola-Coca.



Requisitos para as condições do **HAVING**

- ◆ Vale qualquer coisa dentro de uma subconsulta.
- ◆ Fora de subconsultas, o **HAVING** pode referenciar um elemento somente se ele for:
 1. Um atributo agrupador, ou
 2. Uma agregação
(essa é a mesma regra usada para cláusulas **SELECT** com agregação).

Referências Bibliográficas

- ◆ *Database Systems - The Complete Book*, Garcia-Molina, Ullman e Widom. 2002.
Capítulo 6
- ◆ *Sistemas de Bancos de Dados (6ª edição)*, Elmasri e Navathe. 2010.
Capítulos 4 e 5