

# **[MAC0313] Introdução aos Sistemas de Bancos de Dados**

## **Aula 18 Linguagem SQL (Parte 2)**

### **Consultas Básicas e Consultas Envolvendo Múltiplas Relações**

13 de outubro de 2016

Prof<sup>a</sup> Kelly Rosa Braghetto

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

# Comandos

## Select-From-Where

**SELECT** <lista de atributos>  
**FROM** <lista de tabelas>  
**WHERE** <condição>

## Exemplo para a aula

- ◆ As 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)

Vendedor(nome\_lanch, nome\_refri, preco)

Frequentador(nome\_cliente, nome\_lanch)

## Exemplo

- ◆ Usando `Refrigerante(nome, fabricante)`, quais “refris” são feitos por *Cola-Coca* ?

```
SELECT nome
```

```
FROM Refrigerante
```

```
WHERE fabricante = 'Cola-Coca';
```

# Resultado da consulta

nome
Fanfa
Kuaif
Sprife
...

A resposta é uma relação com um único atributo, **nome**, e tuplas com o nome de cada refrigerante produzido pela *Cola-Coca*.

# “Processamento” de uma consulta sobre uma única relação

- ◆ Começa com a relação na cláusula FROM.
- ◆ Aplica-se a seleção indicada na cláusula WHERE.
- ◆ Aplica-se a projeção indicada pela cláusula SELECT.

# Semântica operacional - visão geral

- ◆ Processamento de uma consulta:
  - ▶ Considere que há uma *variável-tupla* percorrendo cada tupla da relação mencionada na cláusula FROM.
  - ▶ Verifique se a tupla “atual” satisfaz a cláusula WHERE.
  - ▶ Se sim, compute os atributos ou expressões da cláusula SELECT usando os componentes dessa tupla.

# Semântica operacional

nome	fabricante
Fanfa	Cola-Cola

Verifica se é  
*Cola-Coca*

Se sim, inclui  
t.nome no resultado

A variável-tupla  $t$   
percorre todas as  
tuplas

```
SELECT nome
FROM Refrigerante
WHERE fabricante = 'Cola-Coca';
```

## O \* em cláusulas SELECT

- ◆ Quando há apenas uma relação na cláusula FROM, um \* na cláusula SELECT equivale a “todos os atributos dessa relação”.
- ◆ Exemplo:  
Usando Refrigerante(nome, fabricante)

```
SELECT *  
FROM Refrigerante  
WHERE fabricante = 'Cola-Coca';
```

## Resultado da consulta

nome	fabricante
Fanfa	Cola-Coca
Kuaif	Cola-Coca
Sprife	Cola-Coca
. . .	. . .

Agora, o resultado possui todos os atributos de Refrigerante.

## Renomeando atributos

- ◆ Para modificar os nomes dos atributos no resultado, use **“AS < novo nome >”** para renomear um atributo.

- ◆ Exemplo:

usando Refrigerante(nome, fabricante)

```
SELECT nome AS refri, fabricante  
FROM Refrigerante  
WHERE fabricante = 'Cola-Coca';
```

## Resultado da consulta

refri	fabricante
Fanfa	Cola-Coca
Kuaif	Cola-Coca
Sprife	Cola-Coca
. . .	. . .

# Expressões em cláusulas SELECT

◆ Atributos, constantes e funções podem aparecer como elementos na cláusula SELECT.

◆ Exemplo:

Usando `Venda(nome_lanch, nome_refri, preco)`

```
SELECT nome_lanch, nome_refri,  
       preço*32.32 AS preco_em_iene  
FROM Venda;
```

## Resultado da consulta

nome_lanch	nome_refri	preco_em_iene
Sujinhos	Fanfa	161,65
Bar do Zé	Sprife	145,48
...	...	...

## **Exemplo: Constantes como expressões**

Usando `Apreciador(nome_cliente, nome_refri)`:

```
SELECT nome_cliente,  
        'aprecia Fanfa' AS descricao  
FROM Appreciador  
WHERE nome_refri = 'Fanfa';
```

# Resultado da consulta

cliente	descricao
Sally	aprecia Fanfa
Fred	aprecia Fanfa
...	...

# Condições complexas para a cláusula WHERE

- ◆ Operadores booleanos AND, OR, NOT.
- ◆ Comparações =, <>, <, >, <=, >=.
- ◆ E muitos outros operadores que produzem valores booleanos como resultado.

## Exemplo: Condição “complexa”

- ◆ Usando `Venda(nome_lanch, nome_refri, preco)`, encontre o preço cobrado no *Sujinhos* pela *Fanfa*:

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

# Busca de padrões em cadeias de caracteres

- ◆ Uma condição pode comparar uma *string* com um padrão (~ expressão regular) usando:
  - ▶ <Atributo> **LIKE** <padrão> ou <Atributo> **NOT LIKE** <padrão>
- ◆ *Padrão* é uma *string* contendo caracteres especiais:
  - ▶ '%' “casa” com qualquer *string*
  - ▶ '\_' “casa” com qualquer (um) caractere

## Exemplo: LIKE

### ◆ Usando

Cliente(nome, endereço, telefone),  
encontre os clientes com DDD de *São Paulo*:

```
SELECT nome  
FROM Cliente  
WHERE telefone LIKE ' (11) % ' ;
```

## Exemplo(2): LIKE

### ◆ Usando

Cliente(nome, endereço, telefone),  
encontre os clientes cujo primeiro nome  
tem 3 letras:

```
SELECT nome  
FROM Cliente  
WHERE nome LIKE '___ %';
```

# Caracteres especiais em expressões com o LIKE

- ◆ Para usar '%' ou o '\_' em um padrão sem que eles exerçam a função de caractere especial, é preciso fazer o “*scape*” deles.
- ◆ A SQL nos permite usar qualquer caractere como *scape*.
- ◆ **Exemplo:** padrão que “casa” o valor do atributo *s* com uma *string* iniciada e finalizada por '%'

```
s LIKE 'x%%x%' ESCAPE 'x'
```

# Comparação de *strings*, datas e horários

- ◆ Também podemos usar os operadores  $>$ ,  $>=$ ,  $<$  e  $<=$  para comparar *strings*, datas e horários.
- ◆ Quando comparamos *strings* com o  $<$ , por exemplo, estamos perguntando se uma *string* precede a outra na ordem lexicográfica.
- ◆ **Exemplos:**  
'facada'  $<$  'farpa' e 'bar'  $<$  'barganha'

## Qual é o resultado da consulta a seguir?

- ◆ A partir da relação Venda a seguir:

nome_lanch	nome_refri	preco
Sujinhos	Fanfa	NULL

```
SELECT nome_lanch FROM Venda
WHERE preco < 2.00 OR preco >= 2.00;
```

# Comparando NULL com outros valores

- ◆ Tuplas em relações SQL podem ter o NULL como valor para um ou mais de seus atributos.
- ◆ A lógica das condições em SQL é uma lógica ternária: **TRUE, FALSE, UNKNOWN**.
- ◆ Comparar qualquer valor (incluindo o próprio NULL) com NULL resulta em **UNKNOWN**.
- ◆ Uma tupla é incluída no conjunto resposta de uma consulta se e somente se a cláusula WHERE é **TRUE** .

## Lógica ternária (ou *trivalente*)

- ◆ Para entender como o **AND**, **OR** e o **NOT** funcionam na lógica ternária, pense que TRUE = 1, FALSE = 0 e UNKNOWN =  $\frac{1}{2}$ .
- ◆ AND = MIN; OR = MAX, NOT(x) = 1-x.
- ◆ Exemplo:

$$\begin{aligned} \text{TRUE AND (FALSE OR NOT(UNKNOWN))} &= \\ \text{MIN(1, MAX(0, (1 - } \frac{1}{2} \text{)))} &= \\ \text{MIN(1, MAX(0, } \frac{1}{2} \text{))} &= \text{MIN(1, } \frac{1}{2} \text{)} = \frac{1}{2}. \end{aligned}$$

## Um exemplo surpreendente

- ◆ A partir da relação Venda a seguir:

nome_lanch	nome_refri	preco
Sujinhos	Fanfa	NULL

```
SELECT nome_lanch FROM Venda
```

```
WHERE preco < 2.00 OR preco >= 2.00;
```

← UNKNOWN →      ← UNKNOWN →

← UNKNOWN →

**Resultado:** nenhuma tupla é selecionada!

# Razão: Leis para a lógica binária != Leis para a lógica ternária

- ◆ Algumas leis comuns, como a comutatividade do AND, valem na lógica ternária.
- ◆ Mas outras **não valem**
  - ▶ Exemplo: *lei do meio excluído*  
 $p \text{ OR NOT } p = \text{TRUE}$
  - ▶ Quando  $p = \text{UNKNOWN}$ , o lado esquerdo é  $\text{MAX}( \frac{1}{2}, (1 - \frac{1}{2}) ) = \frac{1}{2} \neq 1$ .

# Ordenação do resultado de uma consulta

- ◆ É possível ordenar as tuplas da relação resultante de uma consulta por meio da cláusula

**ORDER BY <lista de atributos> [ASC | DESC]**

- ◆ A ordenação ascendente (ASC) é a padrão

- ◆ Exemplos:

```
SELECT * FROM Cliente ORDER BY  
                                nome, telefone;
```

ou

```
SELECT * FROM Cliente ORDER BY nome DESC;
```

# Consultas envolvendo múltiplas relações

- ◆ Consultas interessantes frequentemente combinam dados de mais de uma relação.
- ◆ Podemos considerar várias relações em uma consulta listando-as na cláusula FROM.
- ◆ Para distinguir atributos de relações diferentes que possuem o mesmo nome: “<relação>.<atributo>” .

## Exemplo: junção de duas relações

- ◆ Usando a relação `Apreciador(nome_cliente, nome_refri)` e `Frequentador(nome_cliente, nome_lanch)`, encontre os refrigerantes apreciados por pelo menos uma pessoa que frequenta a lanchonete Sujinhos.

```
SELECT nome_refri
FROM Appreciador, Frequentador
WHERE nome_lanch = 'Sujinhos' AND
Frequentador.nome_cliente =
Appreciador.nome_cliente;
```

# Semântica formal

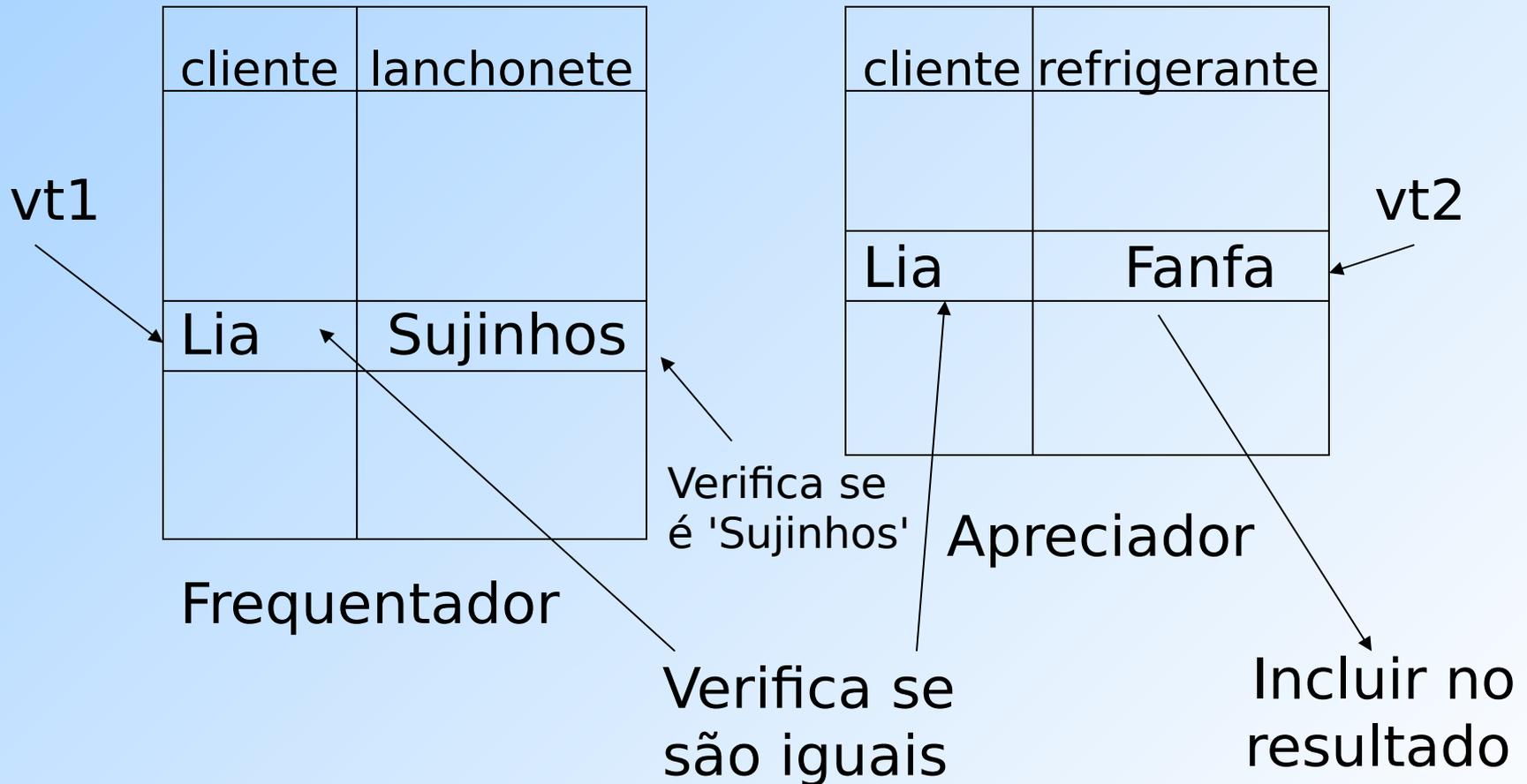
- ◆ Quase a mesma que a das consultas sobre uma única relação:
  1. Comece com o produto cartesiano de todas as relações da cláusula FROM.
  2. Aplique a condição de seleção da cláusula WHERE.
  3. Projete sobre a lista de atributos e expressões da cláusula SELECT.

# Semântica operacional

- ◆ Imagine uma variável-tupla para cada relação na cláusula FROM.
  - ◆ Essas variáveis visitam cada combinação possível de tuplas, uma de cada relação.
- ◆ Se as variáveis-tuplas apontam para tuplas que satisfazem a cláusula WHERE, envie essas tuplas para a cláusula SELECT.

# Exemplo

```
SELECT nome_refri
FROM Apreciador, Frequntador
WHERE nome_lanch = 'Sujinhos' AND
Frequntador.nome_cliente = Apreciador.nome_cliente;
```



## Variáveis-tuplas explícitas

- ◆ Às vezes, uma consulta precisa usar duas cópias de uma mesma relação.
- ◆ Para diferenciar as cópias, acrescenta-se o nome de uma variável-tupla na frente do nome da relação na cláusula FROM.
- ◆ É sempre possível renomear uma relação desta forma (mesmo quando isso não é indispensável para a consulta).

## Exemplo: auto-junção

- ◆ A partir de `Refrigerante(nome_refri, fabricante)`, encontre todos os pares de refri feitos por um mesmo fabricante. Restrições:
  - ▶ Não produza pares como *(Fanfa, Fanfa)*.
  - ▶ Produza pares em ordem alfabética, p.e., *(Fanfa, Sprife)*, mas não *(Sprife, Fanfa)*.

```
SELECT r1.nome, r2.nome
FROM Refrigerante r1, Refrigerante r2
WHERE r1.fabricante = r2.fabricante
      AND r1.nome < r2.nome;
```

# Subconsultas

- ◆ Um comando SELECT-FROM-WHERE parentizado (= *subconsulta*) pode ser usado 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 (atributo).
  - ▶ Um erro é gerado em tempo de execução se não há nenhuma tupla no resultado da subconsulta 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) -  
as duas combinações  
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.

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

```
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 (coluna).

## O operador ALL

- ◆  $x \langle \rangle \mathbf{ALL}(\langle \mathbf{subconsulta} \rangle)$  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:**  $x \geq \mathbf{ALL}(\langle \mathbf{subconsulta} \rangle)$  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.

# Referências bibliográficas

- ◆ *A First Course in Database Systems*,  
Ullman e Widom. 1997.  
Capítulo 5
- ◆ *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