

Sistemas de Informação
GSI016 Banco de Dados 1



SQL: Mais sobre DML

Prof. Humberto Luiz Razente
humberto@facom.ufu.br
Bloco B - sala 1B144

Roteiro



- ◆ Junções

- ◆ Visões

SQL DML

◆ SELECT ... FROM ... WHERE ...

- lista atributos de uma ou mais tabelas de acordo com alguma condição

◆ INSERT INTO ...

- insere dados em uma tabela

◆ DELETE FROM ... WHERE ...

- remove dados de tabelas já existentes

◆ UPDATE ... SET ... WHERE ...

- altera dados específicos de uma tabela

Junção

Relembrando os tipos

- ◆ Existem **dois tipos de condição de junção**
 - **Equi-junções** empregam o operador de igualdade
 - **Junções theta** empregam outros operadores como <, >, BETWEEN, similaridade, etc.
- ◆ Existem **três tipos de junção**
 - **Junção interna:** retornam uma linha somente quando os atributos na junção contêm valores que satisfazem a condição de junção
 - **Junção externa:** retornam uma linha mesmo quando um dos atributos (ou ambas) na condição de junção contém um valor nulo
 - **Auto junção:** retornam o resultado da junção de uma tabela com ela mesma

Auto Junção

- ◆ É necessário empregar um apelido (*alias*) para identificar cada referência para a tabela na consulta

Cláusula AS

◆ Renomeia

- atributos

- ◆ deve aparecer na cláusula SELECT
- ◆ **útil para a visualização das respostas na tela**

- relações

- ◆ deve aparecer na cláusula FROM
- ◆ **útil quando a mesma relação é utilizada mais do que uma vez na mesma consulta**

◆ Sintaxe

- nome_antigo AS nome_novo

Auto Junção

Empregando apelidos no PostgreSQL

◆ Exemplo:

- Para selecionar o nome de todos os funcionários de uma empresa juntamente com o nome de seus gerentes:

```
■ SELECT t1.nome funcionario,  
        t2.nome gerente  
FROM   empregado t1, empregado t2  
WHERE  t1.id_gerente = t2.id_funcionario;
```

Junção

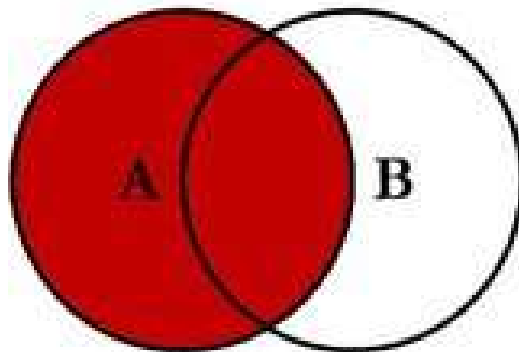
◆ SQL-92

- inclusão de operações adicionais na **cláusula FROM**

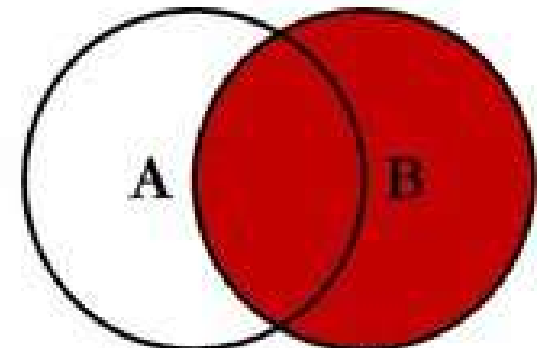
◆ Operações adicionais no PostgreSQL

- ... [INNER] JOIN ... ON ...
- ... LEFT [OUTER] JOIN ... ON ...
- ... RIGHT [OUTER] JOIN ... ON ...
- ... FULL [OUTER] JOIN ... ON ...

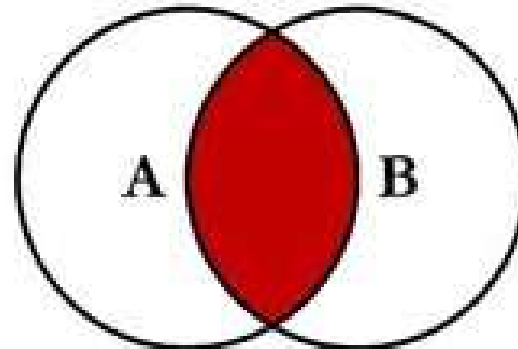
SQL JOINS



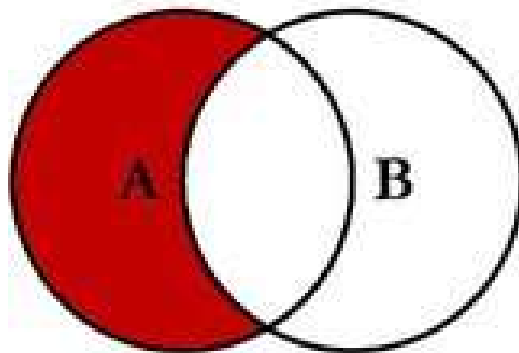
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



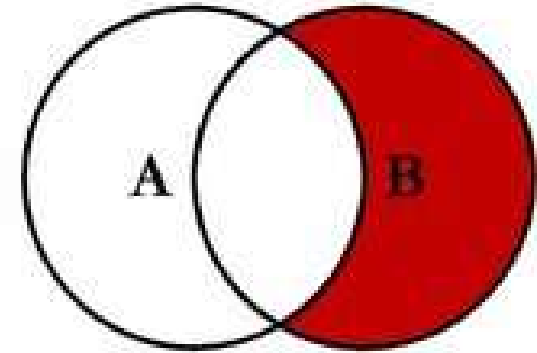
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



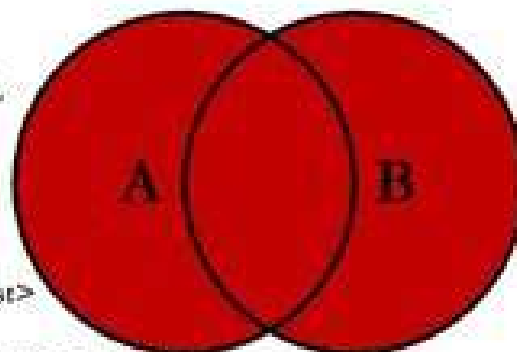
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



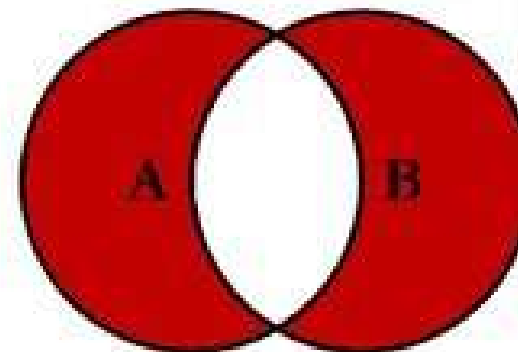
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

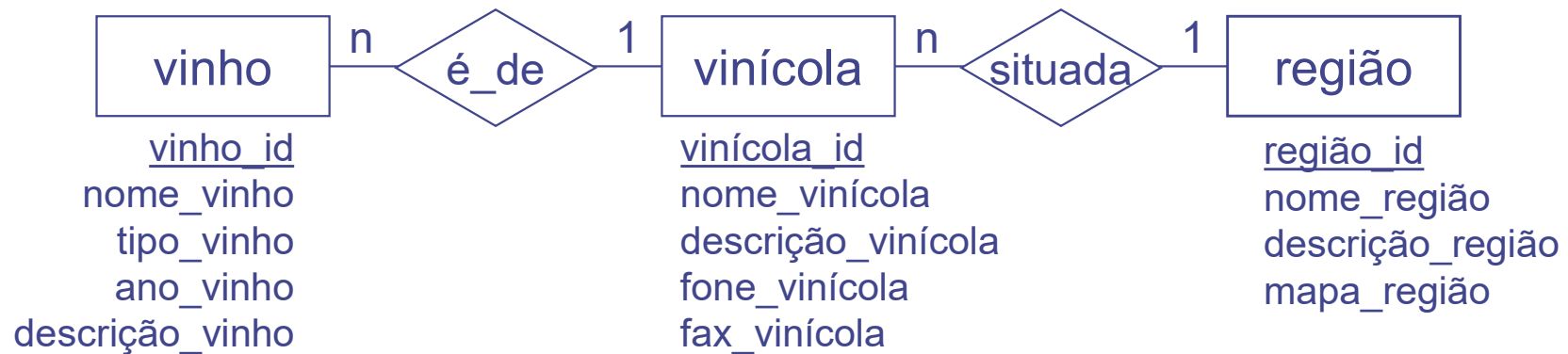


```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Relações Base



- ◆ **região** (região_id, nome_região, mapa_região, descrição_região)
- ◆ **vinícola** (vinícola_id, nome_vinícola, descrição_vinícola, fone_vinícola, fax_vinícola, **região_id**)
- ◆ **vinho** (vinho_id, nome_vinho, tipo_vinho, ano_vinho, descrição_vinho, **vinícola_id**)

Exemplos

- ◆ `SELECT nome_vinícola, nome_região
FROM vinícola, região
WHERE vinícola.região_id = região.região_id;`
- ◆ `SELECT nome_vinícola, nome_região,
FROM vinícola LEFT OUTER JOIN região
ON vinícola.região_id = região.região_id;`

Existe diferença entre os comandos?

Exemplos

- ◆

```
SELECT nome_vinícola, nome_região, nome_vinho  
FROM vinícola, região, vinho  
WHERE vinícola.região_id = região.região_id AND  
vinho.vinícola_id = vinícola.vinícola_id;
```
- ◆

```
SELECT nome_vinícola, nome_região, nome_vinho  
FROM vinícola JOIN região JOIN vinho  
ON vinho.vinícola = vinícola.vinícola_id  
ON vinícola.região_id = região.região_id;
```

Existe diferença entre os comandos?

Exemplo

◆ SELECT nome_vinícola AS nome_da_vinícola ,
nome_região AS localizada_na_região ,
nome_vinho AS produz_o_vinho ,
FROM vinícola AS V,
região AS R,
vinho AS Vi
WHERE V.região_id = R.região_id AND
Vi.vinícola_id = V.vinícola_id;

Subconsultas: SELECTs aninhados

◆ São blocos SELECT...FROM...WHERE completos dentro da cláusula WHERE de outra consulta


◆ Exemplo:

- Selecionar o nome do vinho mais velho

```
SELECT nome_vinho  
FROM vinho
```

```
WHERE ano_vinho IN (SELECT MIN(ano_vinho)  
                    FROM vinho);
```

SELECT



```
SELECT <lista de atributos e funções>  
FROM <lista de tabelas>  
[ WHERE predicado ]  
[ GROUP BY <atributos de agrupamento> ]  
[ HAVING <condição para agrupamento> ]  
[ ORDER BY <lista de atributos> ] ;
```

<https://www.postgresql.org/docs/current/sql-select.html>

Cláusula GROUP BY

- ◆ Podemos dividir o conjunto de tuplas de uma relação em grupos de acordo com algum critério, baseado nos valores dos atributos
 - Por exemplo, na tabela abaixo as tuplas podem ser agrupadas de acordo com o nome do autor
 - Exemplo: PAULO COELHO, MACHADO DE ASSIS e JOSÉ MARIA

Output pane

	titulo character(35)	nome character varying(20)
1	MAKTUB	PAULO COELHO
2	BRIDA	PAULO COELHO
3	HISTÓRIAS PARA PAIS, FILHOS E NETOS	PAULO COELHO
4	DOM CASMURRO	MACHADO DE ASSIS
5	SOL	JOSÉ MARIA

Cláusula GROUP BY

-- autores de todos os livros da livraria

```
SELECT NOME
FROM LIVRO, AUTOR, ESCREVE
WHERE COD_AUTOR = COD_AUTOR_ESC AND
      COD_LIVRO_ESC = COD_LIVRO
GROUP BY NOME;
```

- ◆ Observe que na cláusula SELECT só podem constar os atributos presentes no GROUP BY
- ◆ Isso faz sentido pois, por exemplo, existem 3 livros do PAULO COELHO cadastros, qual deles apareceria no resultado?

- ◆ Tente adicionar o campo TITULO na cláusula SELECT do comando acima e observe o resultado

Data Output		Explain	Messa
	nome	character varying(20)	
1	PAULO COELHO		
2	MACHADO DE ASSIS		
3	JOSÉ MARIA		

Cláusula GROUP BY

-- autores de todos os livros da livraria

```
SELECT NOME
  FROM LIVRO, AUTOR, ESCREVE
 WHERE COD_AUTOR = COD_AUTOR_ESC AND
        COD_LIVRO_ESC = COD_LIVRO
GROUP BY NOME;
```

◆ Tente adicionar o campo TITULO na cláusula SELECT do comando acima e observe o resultado

◆ ERRO: coluna "livro.titulo" deve aparecer na cláusula GROUP BY ou ser utilizada em uma função de agregação

Data Output		Explain	Messa
	nome		
	character varying(20)		
1	PAULO COELHO		
2	MACHADO DE ASSIS		
3	JOSÉ MARIA		

Cláusula GROUP BY

- ◆ Mais de um atributo pode ser usado no agrupamento

	titulo character(35)	nome character varying(20)	valor numeric(7,2)
1	SQL	JOSÉ MARIA	10.90
2	DOM CASMURRO	MACHADO DE ASSIS	10.90
3	O ALIENISTA	MACHADO DE ASSIS	10.90
4	CONTOS FLUMINENSES	MACHADO DE ASSIS	24.00
5	CONTOS	MACHADO DE ASSIS	24.00
6	MAKUB	PAULO COELHO	24.00
7	BRIDA	PAULO COELHO	29.50
8	HISTÓRIAS PARA PAIS, FILHOS E NETOS	PAULO COELHO	35.00

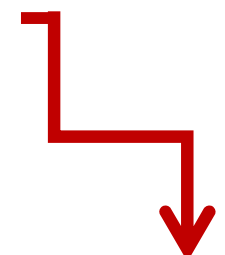
-- agrupando os livros por autor/preço

```
SELECT NOME, VALOR FROM LIVRO, AUTOR, ESCRIVE
WHERE COD_AUTOR = COD_AUTOR_ESC AND
      COD_LIVRO_ESC = COD_LIVRO
GROUP BY NOME, VALOR
ORDER BY NOME
```

Cláusula GROUP BY

- ◆ Mais de um atributo pode ser usado no agrupamento

	titulo character(35)	nome character varying(20)	valor numeric(7,2)
1	SQL	JOSÉ MARIA	10.90
2	DOM CASMURRO	MACHADO DE ASSIS	10.90
3	O ALIENISTA	MACHADO DE ASSIS	10.90
4	CONTOS FLUMINENSES	MACHADO DE ASSIS	24.00
5	CONTOS	MACHADO DE ASSIS	24.00
6	MAKUB	PAULO COELHO	24.00
7	BRIDA	PAULO COELHO	29.50
8	HISTÓRIAS PARA PAIS, FILHOS E NETOS	PAULO COELHO	35.00



-- agrupando os livros por autor/pr

```
SELECT NOME, VALOR FROM LIVRO, AU
WHERE COD_AUTOR = COD_AUTOR_ESC
      COD_LIVRO_ESC = COD_LIVRO

GROUP BY NOME, VALOR

ORDER BY NOME
```

	nome character varying(20)	valor numeric(7,2)
1	JOSÉ MARIA	10.90
2	MACHADO DE ASSIS	10.90
3	MACHADO DE ASSIS	24.00
4	PAULO COELHO	24.00
5	PAULO COELHO	29.50
6	PAULO COELHO	35.00

Cláusula GROUP BY

- ◆ As funções agregadas (e.g., COUNT, MIN, MAX, AVG) podem ser usadas para cálculos com subgrupos de tuplas definidos pela cláusula GROUP BY

- ◆

	titulo character(35)	nome character varying(20)	valor numeric(7,2)
1	SQL	JOSÉ MARIA	10.90
2	DOM CASMURRO	MACHADO DE ASSIS	10.90
3	O ALIENISTA	MACHADO DE ASSIS	10.90
4	CONTOS FLUMINENSES	MACHADO DE ASSIS	24.00
5	CONTOS	MACHADO DE ASSIS	24.00
6	MAKTUB	PAULO COELHO	24.00
7	BRIDA	PAULO COELHO	29.50
8	HISTÓRIAS PARA PAIS, FILHOS E NETOS	PAULO COELHO	35.00

-- qual o número de livros por autor?

```
SELECT NOME, COUNT(*)  
FROM AUTOR, ESCREVE  
WHERE COD_AUTOR = COD_AUTOR_ESC  
GROUP BY NOME
```

Cláusula GROUP BY

- ◆ As funções agregadas (e.g., COUNT, MIN, MAX, AVG) podem ser usadas para cálculos com subgrupos de tuplas definidos pela cláusula GROUP BY

- ◆

	titulo character(35)	nome character varying(20)	valor numeric(7,2)
1	SQL	JOSÉ MARIA	10.90
2	DOM CASMURRO	MACHADO DE ASSIS	10.90
3	O ALIENISTA	MACHADO DE ASSIS	10.90
4	CONTOS FLUMINENSES	MACHADO DE ASSIS	24.00
5	CONTOS	MACHADO DE ASSIS	24.00
6	MAKTUB	PAULO COELHO	24.00
7	BRIDA	PAULO COELHO	29.50
8	HISTÓRIAS PARA PAIS, FILHOS E NETOS	PAULO COELHO	35.00

-- qual o número de livros por autor

```
SELECT NOME, COUNT(*)
```

```
FROM AUTOR, ESCREVE
```

```
WHERE COD_AUTOR = COD_AUTOR_ESC
```

```
GROUP BY NOME
```

	nome character varying(20)	count bigint
1	PAULO COELHO	3
2	MACHADO DE ASSIS	4
3	JOSÉ MARIA	1

Cláusula HAVING

- ◆ **HAVING:** é semelhante à cláusula WHERE. HAVING elimina tuplas *agrupadas que não satisfazem a uma determinada condição*
- ◆ **Diferença com WHERE:** WHERE filtra tuplas individuais antes da aplicação do GROUP BY, enquanto HAVING filtra grupo de tuplas criadas por GROUP BY. As condições de filtragem do HAVING devem ser feitas baseando-se nos atributos agrupados por GROUP BY

Cláusula HAVING

-- Selecionar o nome das editoras cujo total de publicações é maior que 1 ?

-- Solução

```
SELECT RAZAO, L.COD_EDITORA
  FROM EDITORA E, LIVRO L
 WHERE E.COD_EDITORA = L.COD_EDITORA
 GROUP BY RAZAO, L.COD_EDITORA
 HAVING COUNT(L.COD_EDITORA) > 1;
```


Subconsultas: SELECTs aninhados

- ◆ São blocos SELECT...FROM...WHERE completos dentro da cláusula WHERE de outra consulta
- ◆ Essa construção possibilita a realização de consultas sobre os resultados obtidos em outras consultas
- ◆ Podem aparecer na cláusulas
 - SELECT
 - FROM
 - WHERE

Subconsultas: SELECTs aninhados

- ◆ Na cláusula FROM: visão inline

- ◆ Exemplo:

```
/* Listar informações sobre livros e editoras*/
```

```
SELECT *
```

```
FROM LIVRO L JOIN (SELECT * FROM EDITORA) E ON  
(L.COD_EDITORA = E.COD_EDITORA)
```

Subconsultas: SELECTs aninhados

◆ Na cláusula FROM: visão inline

◆ Exemplo:

/ Listar informações sobre livros e editoras*/*

SELECT *

**FROM LIVRO L JOIN (SELECT * FROM EDITORA) E ON
(L.COD_EDITORA = E.COD_EDITORA)**

Output pane

	cod_livro smallint	titulo character(35)	valor numeric(7,2)	volume smallint	cod_editora smallint	cod_editora smallint	razao character varying(20)	endereco character(20)	cidade character(70)
1	31	MAKTUB	24.00		1	1	ROCCO	R. RODRIGO S	RIO DE JANEI
2	55	BRIDA	29.50		1	1	ROCCO	R. RODRIGO S	RIO DE JANEI
3	63	HISTÓRIAS PA	35.00		2	2	GLOBO		RIO DE JANEI
4	14	DOM CASMURRO	10.90		3	3	ATICA		SÃO PAULO
5	13	SQL	10.90		4	4	USP/ICMC		

Subconsultas: SELECTs aninhados

- ◆ Na cláusula SELECT: função inline
- ◆ Exemplo:

/ Listar informações sobre livros e o horário da consulta*/*

```
SELECT *, (SELECT now())  
FROM LIVRO L
```

Output pane

	cod_livro smallint	titulo character(35)	valor numeric(7,2)	volume smallint	cod_editora smallint	now timestamp with time zone
1	31	MAKTUB	24.00		1	2017-04-26 07:09:01.83
2	55	BRIDA	29.50		1	2017-04-26 07:09:01.83
3	63	HISTÓRIAS PA	35.00		2	2017-04-26 07:09:01.83
4	14	DOM CASMURRO	10.90		3	2017-04-26 07:09:01.83
5	13	SQL	10.90		4	2017-04-26 07:09:01.83

Subconsultas: SELECTs aninhados

- ◆ Na cláusula WHERE: subconsulta aninhada
 - não correlacionada; ou
 - correlacionada: avaliada 1 vez para cada linha processada pela consulta superior

- ◆ Exemplo:

```
/* Selecionar o título com valor mais alto */  
SELECT TITULO, VALOR  
FROM LIVRO  
WHERE VALOR IN (  
    SELECT MAX(VALOR)  
    FROM LIVRO );
```

Subconsultas: SELECTs aninhados

- ◆ Exemplo – subconsulta não correlacionada:

/ Selecionar o título com valor mais alto */*

Query - postgres on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

SQL Editor Graphical Query Builder

Previous queries

```
set search_path to livraria;  
  
SELECT TITULO, VALOR  
FROM LIVRO  
WHERE VALOR IN (SELECT MAX(VALOR)  
                FROM LIVRO );
```

Output pane

Data Output Explain Messages History

	max numeric
1	35.00

Query - postgres on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

SQL Editor Graphical Query Builder

Previous queries

```
set search_path to livraria;  
  
SELECT TITULO, VALOR  
FROM LIVRO  
WHERE VALOR IN (SELECT MAX(VALOR)  
                FROM LIVRO );
```

Output pane

Data Output Explain Messages History

	título character(35)	valor numeric(7,2)
1	HISTÓRIAS PARA PAIS,	35.00

Subconsultas: SELECTs aninhados

◆ Outros exemplos:

/ Selecionar os títulos mais baratos do que o título cujo código é 31 */*

```
SELECT TITULO, VALOR
FROM LIVRO
WHERE VALOR < (SELECT VALOR
                FROM LIVRO
                WHERE COD_LIVRO = 31) ;
```

Subconsultas: SELECTs aninhados

◆ Outros exemplos:

/ Selecionar os títulos mais baratos do que o título cujo código é 31 */*

```
SELECT TITULO, VALOR
FROM LIVRO
WHERE VALOR < (SELECT VALOR
FROM LIVRO
WHERE COD_LIVRO = 31) ;
```

Output pane

Data Output	Explain	Messages	History
	valor numeric(7,2)		
1	24.00		

Subconsultas: SELECTs aninhados

◆ Outros exemplos:

/ Selecionar os títulos mais baratos do que o título cujo código é 31 */*

```
SELECT TITULO, VALOR
FROM LIVRO
WHERE VALOR < (SELECT VALOR
                FROM LIVRO
                WHERE COD_LIVRO = 31) ;
```

Output pane			
Data Output		Explain	Messages
	titulo character(35)	valor numeric(7,2)	
1	DOM CASMURRO	10.90	
2	SQL	10.90	

Subconsultas: SELECTs aninhados

◆ Outros exemplos:

/ Selecionar os títulos de livros da editora com
COD_EDITORA = 3 que possuam valor menor do que todos
os livros da editora com COD_EDITORA = 1*/*

```
SELECT TITULO, VALOR
FROM LIVRO
WHERE COD_EDITORA = 3 AND
      VALOR < ALL (SELECT VALOR
                   FROM LIVRO
                   WHERE COD_EDITORA = 1);
```

Subconsultas: SELECTs aninhados

◆ Outros exemplos:

/ Selecionar os títulos de livros da editora com
COD_EDITORA = 3 que possuam valor menor do que todos
os livros da editora com COD_EDITORA = 1*/*

```
SELECT TITULO, VALOR
FROM LIVRO
WHERE COD_EDITORA = 3 AND
      VALOR < ALL (SELECT VALOR
                   FROM LIVRO
                   WHERE COD_EDITORA = 1);
```

Output pane

	valor numeric(7,2)
1	24.00
2	29.50

Output pane

	titulo character(35)	valor numeric(7,2)
1	DOM CASMURRO	10.90

Subconsultas: SELECTs aninhados

◆ Pode-se utilizar com os operadores ANY e ALL

- >
- >=
- <
- <=
- =
- <>

◆ Curiosidade: = ANY equivale ao IN

Subconsultas: SELECTs aninhados

- ◆ Consultas aninhadas correlacionadas
 - Quando condição na cláusula WHERE de uma consulta aninhada **referencia** algum atributo de uma relação declarada na consulta externa

Subconsultas: SELECTs aninhados

◆ Exemplo – consultas aninhadas correlacionadas

```
CREATE TABLE FUNCIONARIO (  
  COD INTEGER PRIMARY KEY,  
  NOME VARCHAR(20) );
```

```
CREATE TABLE DEPENDENTE (  
  COD INTEGER PRIMARY KEY,  
  NOME VARCHAR(20),  
  COD_FUNC INTEGER REFERENCES FUNCIONARIO)
```

```
INSERT INTO FUNCIONARIO VALUES (1, 'JOSÉ'), (2,  
  'MARIA'), (3, 'JOÃO');
```

```
INSERT INTO DEPENDENTE VALUES (1, 'MARIA', 2);
```

Subconsultas: SELECTs aninhados

◆ Exemplo - consultas aninhadas correlacionadas

/ Recuperar o nome de cada funcionário que tem um dependente com o mesmo nome do funcionário */*

```
SELECT F.NOME
      FROM FUNCIONARIO F, DEPENDENTE D
     WHERE F.COD = D.COD_FUNC AND
           F.NOME = D.NOME
```

```
SELECT F.NOME
      FROM FUNCIONARIO F
     WHERE F.COD IN (SELECT D.COD_FUNC
                     FROM DEPENDENTE D
                    WHERE F.NOME = D.NOME)
```

Subconsultas: SELECTs aninhados

- ◆ A função EXISTS em SQL é usada para verificar se o resultado de uma consulta aninhada é vazio (não contém tuplas) ou não
 - O resultado de EXISTS é TRUE se o resultado tiver pelo menos uma tupla

Subconsultas: SELECTs aninhados

◆ Exemplo - consultas aninhadas correlacionadas

/ Recuperar o nome de cada funcionário que tem um dependente com o mesmo nome do funcionário */*

```
SELECT F.NOME
FROM FUNCIONARIO F
WHERE EXISTS (SELECT D.COD_FUNC
               FROM DEPENDENTE D
               WHERE F.COD = D.COD_FUNC AND
                     F.NOME = D.NOME)
```

para cada tupla de FUNCIONARIO,
avaliar a consulta aninhada, que
recupera todas as tuplas
DEPENDENTE com os mesmos
COD_FUNC e NOME que a tupla
FUNCIONARIO;

Subconsultas: SELECTs aninhados

◆ Exemplo - consultas aninhadas correlacionadas

/ Recuperar os nomes dos funcionários que não tem dependentes */*

```
SELECT F.NOME
FROM FUNCIONARIO F
WHERE NOT EXISTS (SELECT *
                  FROM DEPENDENTE D
                  WHERE F.COD = D.COD_FUNC)
```

Output pane

	nome character varying(20)
1	JOSÉ
2	JOÃO



VISÕES

Visão

◆ Em SQL

- tabela simples que é derivada de outras tabelas
- as tabelas base podem ser tabelas ou outras visões
- não existe necessariamente em sua forma física → tabela virtual
- a definição de uma visão é armazenada no dicionário de dados que guarda a consulta que gerou a tabela

Visão

◆ Utilidade

- forma de se especificar uma tabela que precisa ser acessada frequentemente, embora essa tabela não exista fisicamente
- facilita a escrita de consultas complexas
- Pode restringir a visualização do conteúdo de uma tabela por meio da limitação das colunas que são exibidas e das linhas que são filtradas

CREATE VIEW

```
CREATE VIEW nome_visão (lista_de_atributos)  
AS <expressão_da_consulta>
```

- ◆ Especifica uma visão

- ◆ Características

- lista_de_atributos: opcional (função de renomeação)
- expressão_da_consulta: consulta para especificar o conteúdo da visão (SELECT ... FROM ... WHERE ...)

Exemplo

◆ Criação da visão

```
CREATE VIEW vinhos_da_vinícola  
AS SELECT vinícola.vinícola_id, nome_vinícola,  
        vinho_id, nome_vinho  
FROM vinho, vinícola  
WHERE vinho.vinícola_id = vinícola.vinícola_id
```

◆ Perguntas

- quantos atributos a visão possui?
- qual a ordem desses atributos?

Exemplo

◆ Criação da visão

```
CREATE VIEW tipos_de_vinho (código, nome, tipo)  
AS SELECT vinho_id, nome_vinho, tipo_vinho  
FROM vinho  
WHERE descrição_vinho LIKE "%delicioso%"
```

◆ Perguntas

- quantos atributos a visão possui?
- qual a ordem desses atributos?

DROP VIEW

DROP VIEW nome_visão

◆ Remove a definição de uma visão

◆ Exemplos:

- DROP VIEW vinhos_da_vinícola;
- DROP VIEW tipos_de_vinho;

Usando o comando `SELECT` em uma visão

- ◆ A visão funciona como uma tabela normal para a maioria das operações, e o comando `SELECT` funciona da maneira usual com uma visão

Usando o comando `SELECT` em uma visão - Exemplos

```
CREATE VIEW vinhos_da_vinicola
AS SELECT vinicola.vinicola_id, nome_vinicola,
           vinho_id, nome_vinho
FROM vinho, vinicola
WHERE vinho.vinicola_id =
       vinicola.vinicola_id
```

◆ **SELECT** * **FROM** vinhos_da_vinicola

◆ **SELECT** vinicola_id, **COUNT** (vinho_id)
 FROM vinhos_da_vinicola
 GROUP BY vinicola_id

Eliminando uma visão

- ◆ Por meio do comando DROP

- `DROP VIEW nome_visão`

- ◆ As tabelas base não são afetadas por esse comando

Visão – Comandos adicionais

◆ Para visualizar no dicionário de dados

- Selecionar a tabela `pg_views`

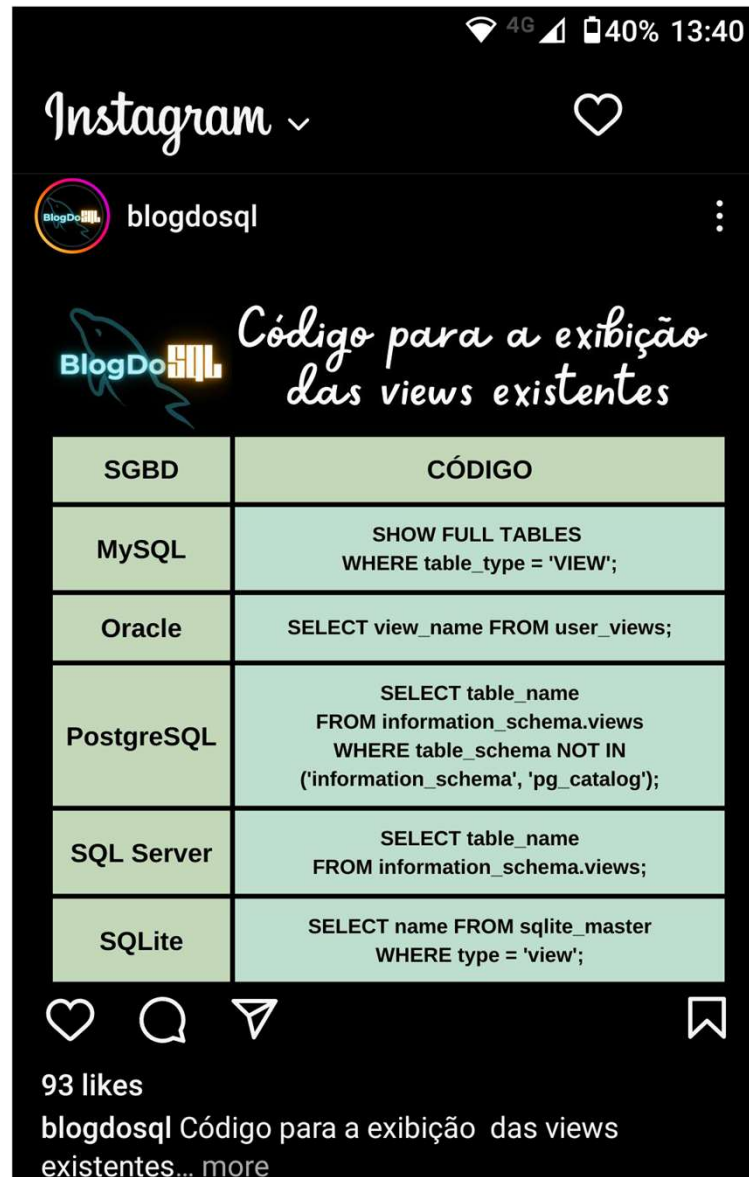
◆ Para substituir uma visão

- Eliminar com o comando `DROP` e depois empregar o comando `CREATE VIEW`

■ Particularidade Oracle:

- ◆ Usar a opção `OR REPLACE` do comando `CREATE VIEW`
 - essa opção preserva as autorizações de segurança existentes

Visões



Bibliografia

- ◆ Elmasri, Ramez; Navathe, Shamkant B. **Sistemas de banco de dados**. 4 ed. São Paulo: Addison Wesley, 2005, 724 p. Bibliografia: p. [690]-714.
- ◆ Deitel, H. M. **Java: como programar**. 6 ed. São Paulo: Pearson Prentice Hall, 2005.

Leitura complementar para casa

◆ Elmasri – Navathe

- Capítulo 8
- Capítulo 9 - Seção 9.2 – Visões

◆ Manual do SGBD PostgreSQL

- Explorar as outras particularidades dos comandos apresentados
- <http://www.postgresql.org/docs/manuals/>