



Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

SQL

Structured Query Language

Prof. Dr. Halley Wesley Gondim
halley.was@gmail.com

SQL - Tirinha



SQL - Histórico

- ✓ Versão original desenvolvida pela IBM (Lab. De Pesquisa San José)
- ✓ Originalmente chamada de Sequel ("Structured English Query Language" (Linguagem de Consulta Estruturada em Inglês))
- ✓ SQL (Structured Query Language – Linguagem de Consulta Estruturada)
- ✓ 1986 – American National Standards Institute (ANSI) e a International Standards Organization (ISO) publicaram padrões para SQL. (SQL-86)

SQL - Histórico

- ✓ SQL (ISO/IEC 9075-x) foi revisto em:
 - SQL-99
 - SQL-2003
 - SQL-2008
 - SQL-2016
- ✓ Linguagem de Definição de Dados (DDL)
 - Comandos definição de esquemas, exclusão, criação de índices e modificação nos esquemas de relações

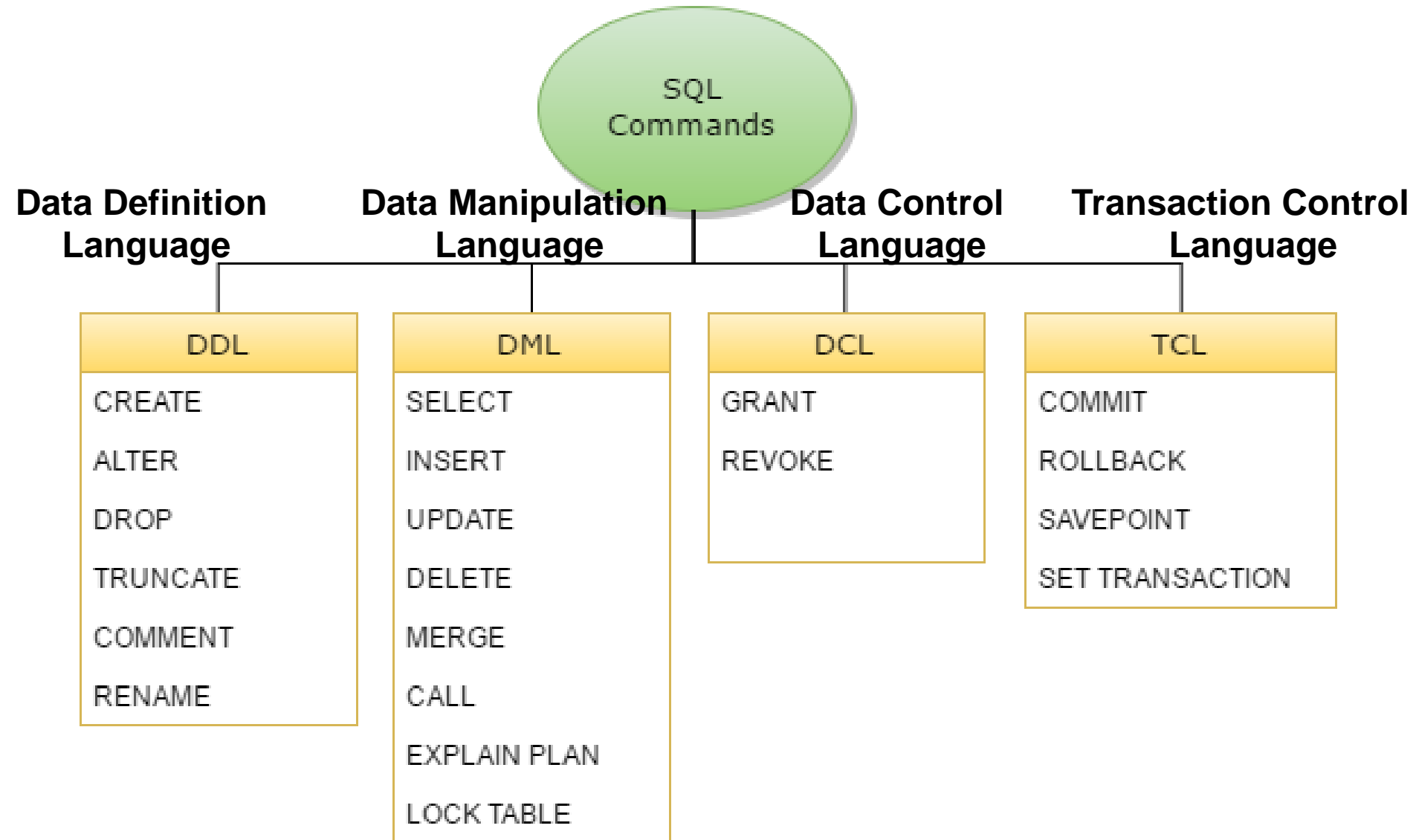
SQL - Histórico

- ✓ Linguagem Interativa de Manipulação de Dados (DML)
 - Linguagem de consulta, inserção, exclusão e modificação
- ✓ Definição de visões
- ✓ Autorização
- ✓ Integridade
- ✓ Controle de transações
 - Inclui comandos para especificar iniciação e finalização de transações.

SQL – Estruturas Básicas

- ✓ SQL permite uso de valores nulos
 - (desconhecidos / inexistentes)
- ✓ A estrutura básica de expressão em SQL consiste:
 - **Select** (Projeção de álgebra relacional – relacionar atributos desejados)
 - **From** (Produto cartesiano, associação entre relações pesquisadas)
 - **Where** (Seleção do predicado - condições)

SQL – Categorias



SQL – DDL

✓ Uma **DDL (Linguagem de Definição de Dados)** permite ao usuário definir novas tabelas e elementos associados.

Obs: A maioria dos bancos de dados de SQL comerciais tem extensões proprietárias no DDL.

✓ SQL DDL permite não só **especificação** de um conjunto de relações, como também **informações** acerca de cada uma **das relações**

SQL – DDL

- ✓ O **esquema** de cada relação (tabela).
- ✓ O **domínio dos valores** associados a cada atributo
- ✓ **Regras de integridade**
- ✓ O conjunto de **índices** para manutenção de cada relação
- ✓ Informações sobre **segurança** e autoridade sobre cada relação
- ✓ A estrutura de **armazenamento** físico de cada relação no disco.

SQL – Definição de Esquema em SQL

Base de Dados – Criar nossos bancos.

```
1  /*CRIAR NOVO DATABASE*/
2  CREATE DATABASE nome;
3
4  /*ALTERAR O NOME DO DATABASE*/
5  ALTER DATABASE nome RENAME TO novo_nome
6
7  /*ALTERAR O PROPRIETÁRIO DO DATABASE*/
8  ALTER DATABASE nome OWNER TO novo_dono
9
10 /*APAGAR DATABASE*/
11 DROP DATABASE novo_nome;
12
```

SQL – Tipos de domínio em SQL

- ✓ **Char(n)** – cadeia de caractere – tamanho fixo
 - ✓ **Varchar(n)** – cadeia caractere - variável – ($\leq n$)
 - ✓ **Integer** – inteiro
 - ✓ **Numeric** - ponto flutuante, precisão em cálculos
 - ✓ **Serial / BigSerial** – inteiro com incremento automático
 - ✓ **Date** – Ano(4 dig.), mês e dia
 - ✓ **Time** – horas, minutos e segundos
 - ✓ **Clob** – texto “infinito”
 - ✓ **Blob** – armazenamento de até 4Gb de dados.
 - ✓ Como é booleano?
- (<https://www.postgresql.org/docs/11/index.html>)

SQL – Definição de Esquema em SQL

✓ Criando uma tabela

▪ **CREATE TABLE** r ($A_1D_1, A_2D_2, \dots, A_ND_N$

< REGRAS DE INTEGRIDADE₁> ,

...

< REGRAS DE INTEGRIDADE_k>)

R = Tabela

A = Atributos

D = Domínio

SQL – Definição de Esquema em SQL

- ✓ Regras de integridades permitidas englobam:
 - **PRIMARY KEY** ($A_{j1}, A_{j2}, \dots, A_{j1m}$)
 - **CHECK** (P)

Os atributos $A_{j1}, A_{j2}, \dots, A_{j1m}$ formam a chave primária da relação

Check especifica um predicado P que precisa ser satisfeito por todas as tuplas em uma tabela/relação

SQL – Definição de Esquema em SQL

```
1 CREATE TABLE nome_da_tabela (  
2 atributo_chave SERIAL PRIMARY KEY,  
3 atributo_1 VARCHAR(80),  
4 atributo_2 NUMERIC(7,2)  
5 )
```

```
1 CREATE TABLE nome_da_tabela (  
2 atributo_chave SERIAL PRIMARY KEY,  
3 atributo_1 VARCHAR(80),  
4 atributo_2 NUMERIC(7,2) CHECK(atributo_2 > 0),  
5 atributo_3 VARCHAR(80) CHECK (atributo_3 IN ('M','F','A')),  
6 atributo_estrangeiro INTEGER,  
7 FOREIGN KEY (atributo_estrangeiro)  
8 REFERENCES nome_tabela_estrangeira  
9 (atributo_chave_tabela_estrangeira)  
10 )  
11
```

SQL – Tipos de domínio em SQL

✓ **Serial / BigSerial** – inteiro com incremento automático

```
1  /*INCREMENTO*/
2  CREATE SEQUENCE nome_tabela_seq;
3  CREATE TABLE nome_da_tabela (
4      nome_da_coluna integer PRIMARY KEY
5      DEFAULT nextval('nome_tabela_seq') NOT NULL
6  );
7
8  /*INCREMENTO COM SERIAL*/
9  CREATE TABLE nome_da_tabela(
10 nome_da_coluna SERIAL PRIMARY KEY
11 )
```

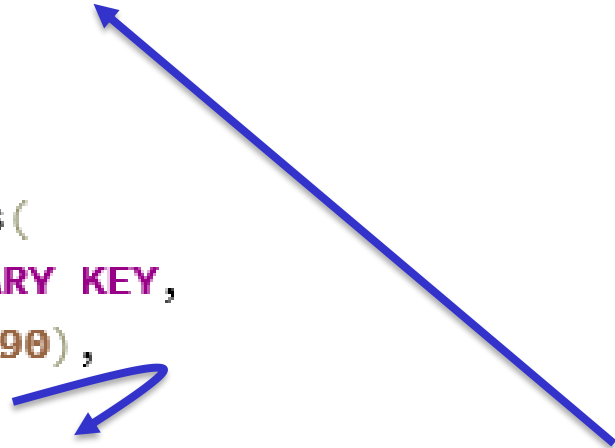
SQL – Sequence

✓Corpo de uma sequence

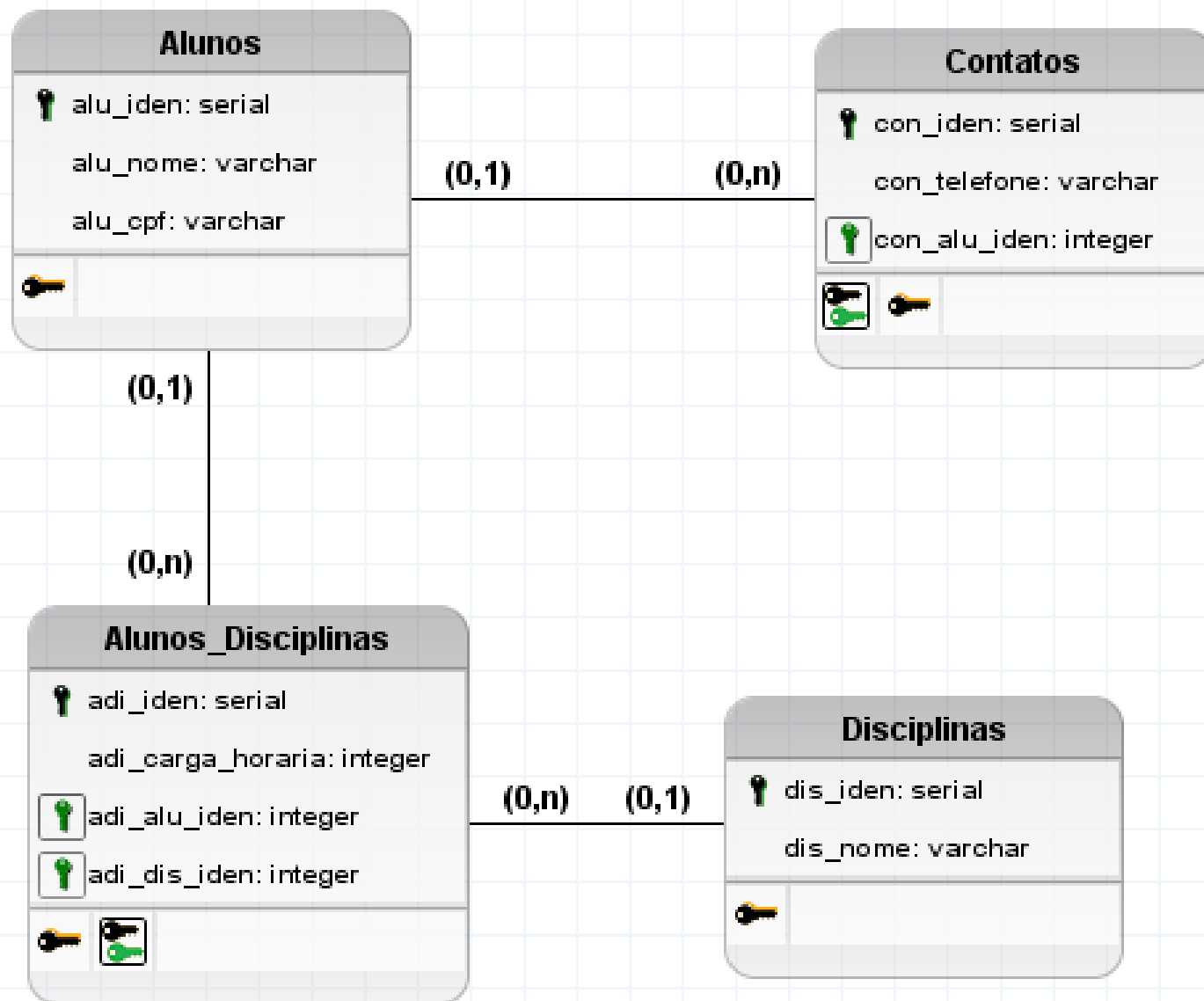
```
1 CREATE SEQUENCE public.nome_da_tabela_seq
2     INCREMENT 1
3     START 1
4     MINVALUE 1
5     MAXVALUE 2147483647
6     CACHE 1;
7
```


SQL – Create table exemplo

```
1  /*TABELA COM ATRIBUTO AUTOINCREMENTO SERIAL*/
2  CREATE TABLE alunos(
3  alu_iden serial PRIMARY KEY,
4  alu_nome varchar(80)
5  )
6
7  CREATE TABLE contatos(
8  con_iden serial PRIMARY KEY,
9  con_telefone varchar(90),
10 con_alu_iden integer,
11 FOREIGN KEY (con_alu_iden) REFERENCES alunos (alu_iden)
12 )
```



SQL – Crie as seguintes tabelas



SQL – DDL tabelas

```
1  CREATE TABLE alunos (  
2      alu_iden serial PRIMARY KEY,  
3      alu_nome varchar,  
4      alu_cpf varchar  
5  );  
6  
7  CREATE TABLE contatos (  
8      con_iden serial PRIMARY KEY,  
9      con_telefone varchar,  
10     con_alu_iden integer,  
11     FOREIGN KEY (con_alu_iden) REFERENCES alunos (alu_iden)  
12 );  
13
```

SQL – DDL tabelas

```
14 CREATE TABLE disciplinas (  
15     dis_iden serial PRIMARY KEY,  
16     dis_nome varchar  
17 );  
18  
19 CREATE TABLE alunos_disciplinas (  
20     adi_iden serial PRIMARY KEY,  
21     adi_carga_horaria integer,  
22     adi_alu_iden integer,  
23     adi_dis_iden integer,  
24     FOREIGN KEY (adi_alu_iden) REFERENCES alunos (alu_iden),  
25     FOREIGN KEY (adi_dis_iden) REFERENCES disciplinas (dis_iden)  
26 );  
27
```

SQL – Definição de Esquema em SQL

✓ Para remoção de uma relação/tabela de um banco de dados SQL, usamos o comando **DROP TABLE**.

✓ **DROP TABLE** nome_tabela

```
1  /*APAGA SOMENTE A TABELA*/  
2  DROP TABLE contatos;  
3  
4  /*APAGA NÃO SÓ A TABELA, MAS QUEM DEPENDE DELA*/  
5  DROP TABLE alunos CASCADE;  
6
```

✓ CASCADE = Remove as restrições

SQL – Definição de Esquema em SQL

✓ Usamos o comando **ALTER TABLE** para adicionar atributos a uma relação existente.

✓ **ALTER TABLE r ADD COLUMN A D**

R = Relação / tabela

A = Atributo

D = Domínio

Podemos encontrar variância para ALTER TABLE.

Ex:

ALTER TABLE r DROP A

ALTER TABLE r RENAME COLUMN C1 TO C2

SQL – Definição de Esquema em SQL

```
1  /*ADICIONANDO NOVA COLUNA A UMA TABELA EXISTENTE*/
2  ALTER TABLE nome_tabela ADD COLUMN nova_coluna tipo_dado;
3
4  /*REMOVENDO UMA COLUNA DA TABELA*/
5  ALTER TABLE nome_tabela DROP COLUMN nome_coluna;
6
7  /*ADICIONANDO RESTRIÇÕES*/
8  ALTER TABLE nome_tabela ADD CHECK (nome_coluna <> '');
9  ALTER TABLE nome_tabela ADD CONSTRAINT nome_constraint UNIQUE (nome_coluna);
10 ALTER TABLE nome_tabela ADD FOREIGN KEY (coluna_chave_estrangeira)
11     REFERENCES tabela_relacionamento (coluna_chave_primaria);
12 /*REMOVENDO RESTRIÇÕES*/
13 ALTER TABLE nome_tabela DROP CONSTRAINT nome_constraint;
14
15 /*RENOMEANDO COLUNAS E TABELA*/
16 ALTER TABLE nome_tabela RENAME COLUMN nome_coluna TO novo_nome_coluna;
17 ALTER TABLE nome_tabela RENAME TO novo_nome_tabela;
18
```

SQL – Inserção

✓ **INSERÇÃO**

- Utiliza-se o comando **INSERT INTO** para incluir dados nas relações.

// não definimos a ordem, por padrão seguir ordem dos atributos no banco.

```
INSERT INTO nome_tabela  
VALUES (ordem_atributo1, ordem_atributo2,...)
```

// se definirmos a ordem dos atributos devemos inserir seus dados respectivamente

```
INSERT INTO nome_tabela (atributo1, atributo2, atributo 3)  
VALUES (valor_atributo1, valor_atributo2, valor_atributo3)
```

```
INSERT INTO nome_tabela (atributo2, atributo3, atributo 1)  
VALUES (valor_atributo2, valor_atributo3, valor_atributo1)
```



SQL – Inserção

✓ **INSERÇÃO**

➤ É possível, na inserção de tuplas, fornecer valores somente para alguns atributos do esquema

```
INSERT INTO conta  
VALUES (NULL, "A-401",1200)
```

SQL – Inserção



| |
|-------------|
| Tables (4) |
| alunos |
| Columns (3) |
| alu_iden |
| alu_nome |
| alu_cpf |
| Constraints |

```
1  /*SEM DEFINIR ORDEM, PEGAR ORDEM PADRÃO DA TABELA*/
2  INSERT INTO alunos VALUES (1,'MARIA JOAQUINA','874.963.111-87');
3
4  /*DEFININDO A ORDEM DOS ATRIBUTOS, TENHO QUE INSERIR NA MESMA SEQUENCIA*/
5  INSERT INTO alunos (alu_nome, alu_cpf) VALUES ('MARCELA','887.698.321-87');
6  INSERT INTO alunos (alu_cpf, alu_nome) VALUES ('222.632.541-87','RAMBO');
7  INSERT INTO alunos (alu_nome, alu_cpf) VALUES ('TARANTINO','414.587.321-99');
8
9  /*DEFININDO INSERÇÃO DE UM CAMPO NULO*/
10 INSERT INTO alunos (alu_nome, alu_cpf) VALUES ('TARANTINO',null);
11
```

➤Obs. Caso execute e a sequence afirmar que já exista o valor 1, rode novamente. Ela vai incrementar em um e tudo volta ao normal.

SQL – DML

✓ Cláusula Select

- O resultado de uma consulta de SQL é, naturalmente, uma relação/tabela.

"mostre todos os dados da tabela alunos"

| | | |
|---|--------|--------|
| 1 | SELECT | * |
| 2 | FROM | alunos |

| Data Output | Explain | Messages | Notifications |
|-------------|--------------------------|-------------------------------|------------------------------|
| | alu_iden [PK] integer | alu_nome character varying | alu_cpf character varying |
| 1 | 1 | MARIA JOAQUINA | 874.963.111-87 |
| 2 | 2 | MARCELA | 887.698.321-87 |
| 3 | 3 | RAMBO | 222.632.541-87 |
| 4 | 4 | TARANTINO | 414.587.321-99 |
| 5 | 5 | TARANTINO | [null] |

✓ Cláusula Select

- Nos casos em que desejamos forçar a eliminação de duplicidade, podemos inserir a palavra chave **DISTINCT** depois de **SELECT**

```
SELECT DISTINCT atributos  
FROM nome_tabela
```

```
SELECT ALL atributos  
FROM nome_tabela
```

Com ALL as duplicidades não serão eliminadas

SQL – DML

✓ Cláusula Select

- Nos casos em que desejamos forçar a eliminação de duplicidade, podemos inserir a palavra chave **DISTINCT** depois de **SELECT**

```
1 SELECT alu_nome
2 FROM alunos
```

| | Data Output | Explain | Message |
|---|--------------------------------------|---------|---------|
| | alu_nome character varying | | |
| 1 | MARIA JOAQUINA | | |
| 2 | MARCELA | | |
| 3 | RAMBO | | |
| 4 | TARANTINO | | |
| 5 | TARANTINO | | |

```
1 SELECT DISTINCT alu_nome
2 FROM alunos
```

| | Data Output | Explain | Messages | No |
|---|--------------------------------------|---------|----------|----|
| | alu_nome character varying | | | |
| 1 | TARANTINO | | | |
| 2 | MARCELA | | | |
| 3 | RAMBO | | | |
| 4 | MARIA JOAQUINA | | | |

SQL – DML

✓ Cláusula Select

- Também pode conter expressões aritméticas envolvendo os operadores **+**, **-**, ***** e **/**

```
SELECT atributo1, atributo2 * 100  
FROM nome_tabela
```

| | | |
|---|---------------|---|
| 1 | SELECT | adi_carga_horaria, adi_carga_horaria * 10 |
| 2 | FROM | alunos_disciplinas |
| 3 | | |

| | Data Output | Explain | Messages | Notifications |
|---|------------------------------|---------|---------------------|---------------|
| | adi_carga_horaria integer | | ?column? integer | |
| 1 | 60 | | 600 | |
| 2 | 100 | | 1000 | |
| 3 | 150 | | 1500 | |
| 4 | 65 | | 650 | |
| 5 | 61 | | 610 | |
| 6 | 40 | | 400 | |

✓ Cláusula Where

- A SQL usa conectores lógicos AND, OR e NOT ao invés de símbolos matemáticos.
- Operadores dos conectivos lógicos podem ser expressões envolvendo operações de comparação: <, <=, >, >=, = e <>

```
SELECT atributos  
FROM nome_tabela  
WHERE atributo1 <= 100000  
        AND atributo2 >= 90000
```

SQL – Valores nulos

- Podemos utilizar a palavra chave **NULL** como predicado para testar a existência de valores nulos.

```
SELECT numero_emprestimo  
FROM emprestimo  
WHERE total IS NULL
```

```
SELECT numero_emprestimo  
FROM emprestimo  
WHERE total IS NOT NULL
```



SQL – DML

✓ Cláusula Where

- Obter cargas horárias (maior que 65 e menor igual a 150)

```
1  SELECT adi_carga_horaria
2  FROM alunos_disciplinas
3  WHERE adi_carga_horaria > 65 AND adi_carga_horaria <=150
```

Data Output Explain Messages Notifications

| | adi_carga_horaria | |
|---|-------------------|--|
| | integer |  |
| 1 | 100 | |
| 2 | 150 | |

SQL – DML

✓ Cláusula Where

- A SQL possui um operador de comparação **between** para simplificar a cláusula where.

```
SELECT atributos  
FROM nome_Tabela  
WHERE atributo1 BETWEEN 900 AND 1000
```

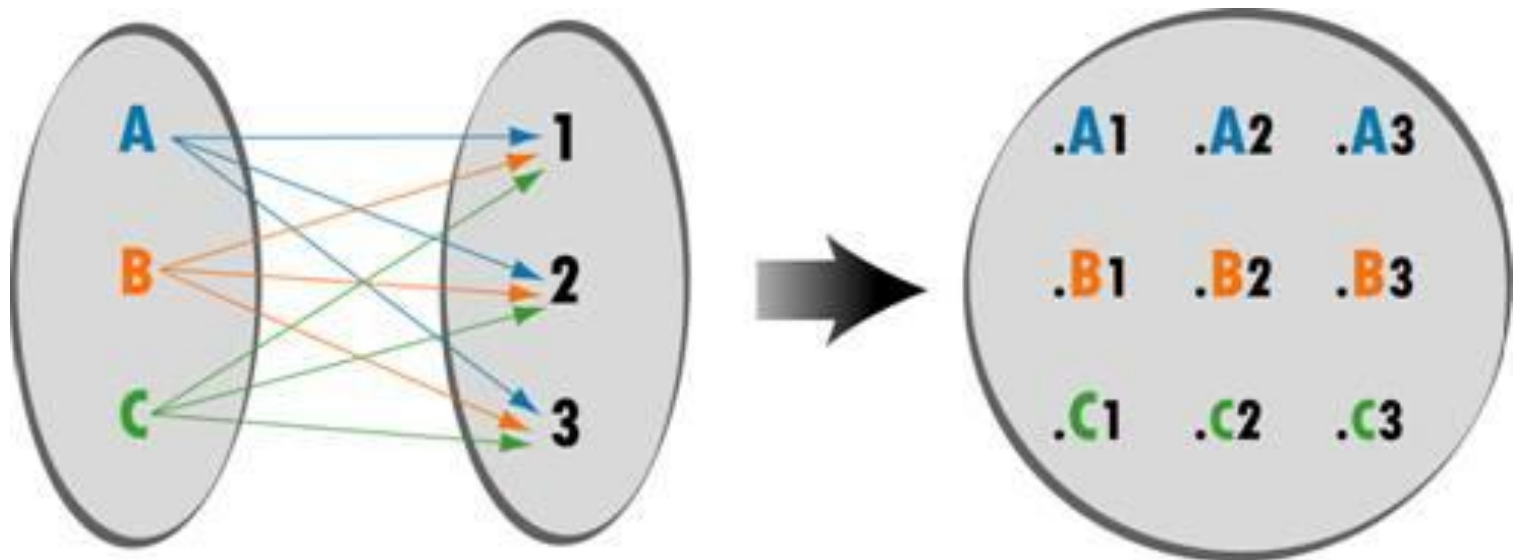
```
1 SELECT adi_carga_horaria  
2 FROM alunos_disciplinas  
3 WHERE adi_carga_horaria BETWEEN 65 AND 150
```

Data Output Explain Messages Notifications

| | adi_carga_horaria integer | |
|---|------------------------------|--|
| 1 | 100 | |
| 2 | 150 | |
| 3 | 65 | |

✓ Cláusula FROM

- Deve se informar qual(is) tabela(s) são necessárias para se realizar a consulta.
- **É um produto cartesiano**

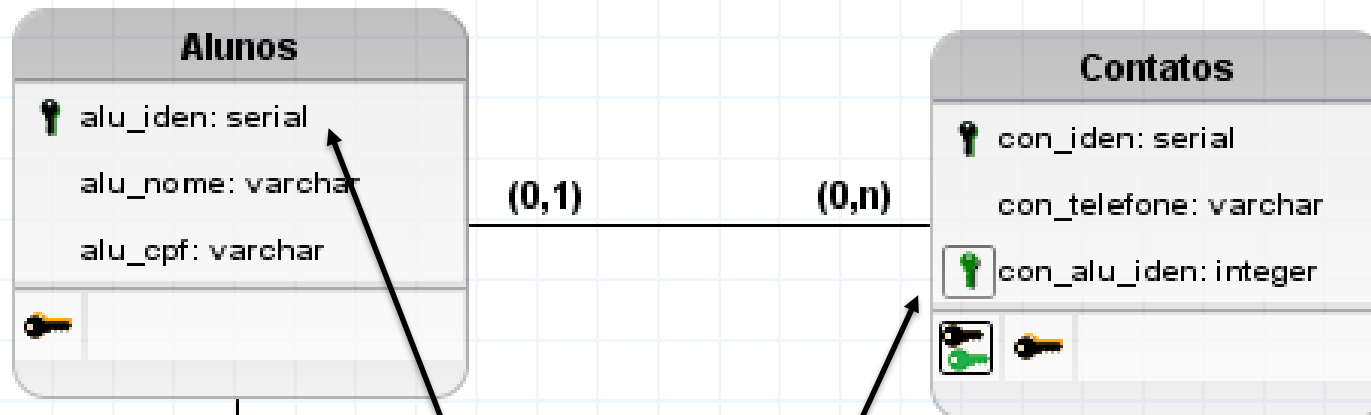


✓ Cláusula FROM

- Note que SQL usa a notação ***nome_relacionamento.nome_atributo*** para evitar ambigüidades.
- Usou no from mais de uma tabela, deve-se realizar essa comparação de chave estrangeira com chave primária da outra tabela.

```
SELECT atributos  
FROM nome_tabela1, nome_tabela2  
WHERE chave_primaria_tab1 = chave_estrangeira_tab2
```

SQL – DML

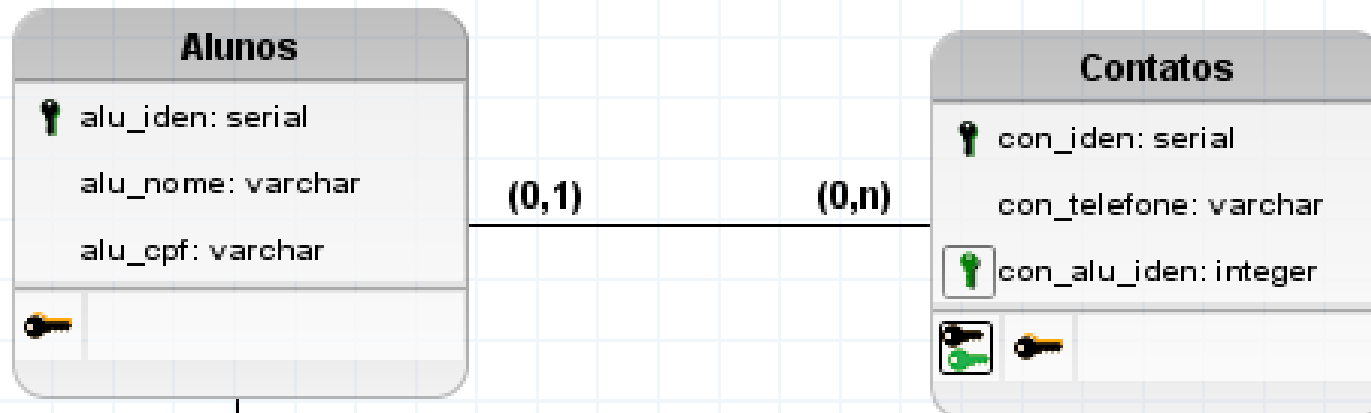


```
1  SELECT alu_nome, contatos.*
2  FROM alunos, contatos
3  WHERE alunos.alu_iden = contatos.con_alu_iden
```

[Data Output](#) [Explain](#) [Messages](#) [Notifications](#)

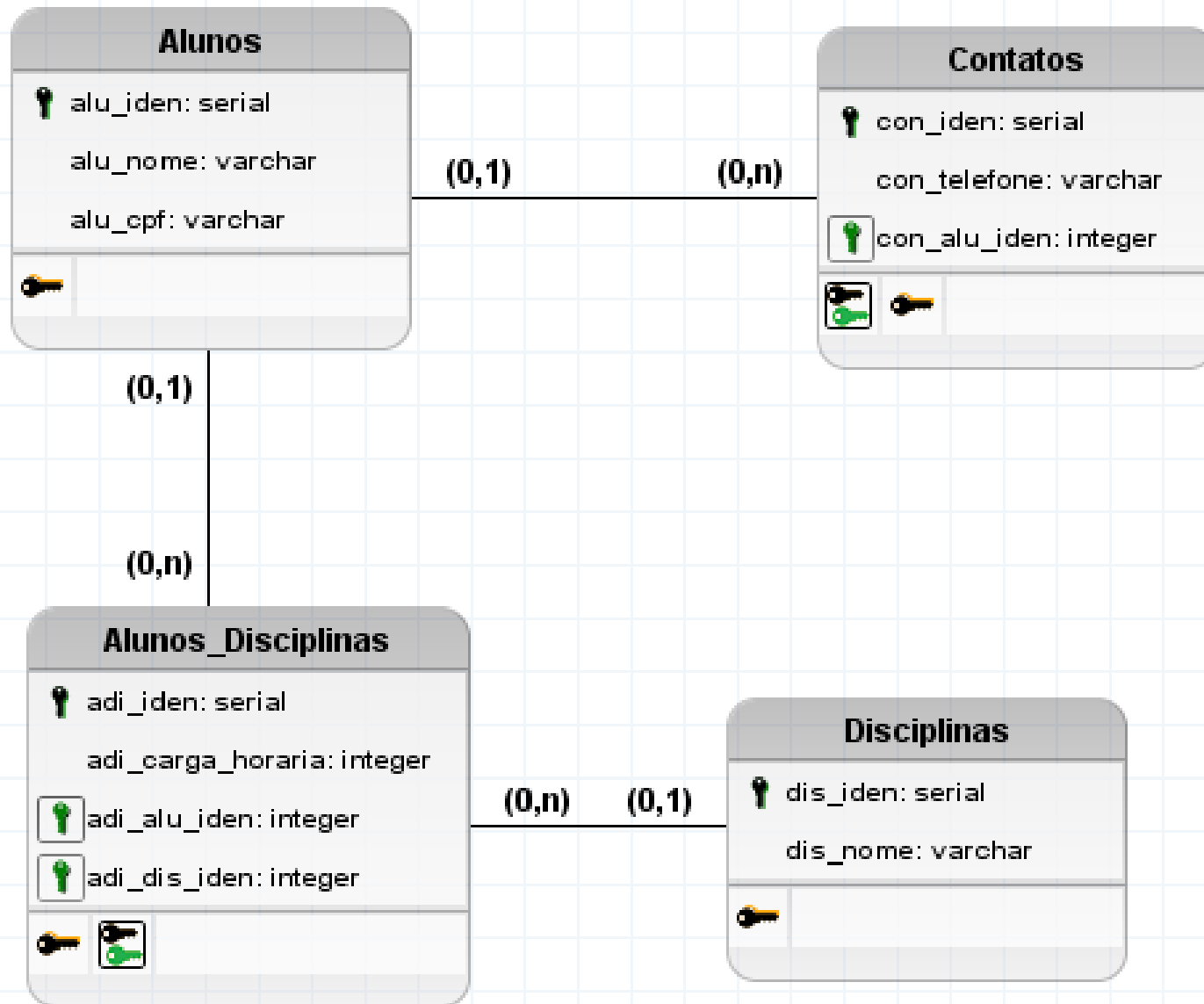
| | alu_nome character varying | con_iden integer | con_telefone character varying | con_alu_iden integer |
|---|--------------------------------------|----------------------------|--|--------------------------------|
| 1 | MARIA JOAQUINA | 7 | 5555-5555 | 1 |
| 2 | MARIA JOAQUINA | 8 | 5444-5555 | 1 |
| 3 | RAMBO | 9 | 5333-5555 | 3 |

SQL – DML



```
1  /**JUNÇÃO SEM COMANDO JOIN*/
2  SELECT *
3  FROM alunos a, contatos c
4  WHERE a.alu_iden = c.con_alu_iden
5
6  /**USANDO JOIN*/
7  SELECT *
8  FROM alunos a JOIN contatos c
9       ON a.alu_iden = c.con_alu_iden
10
```

SQL – DML



SQL – DML

```
1  /*JUNÇÃO NA UNHA*/
2  SELECT *
3  FROM alunos a, contatos c, alunos_disciplinas ad, disciplinas d
4  WHERE a.alu_iden = c.com_alu_iden AND
5         a.alu_iden = ad.adi_alu_iden AND
6         d.dis_iden = ad.adi_dis_iden
7
8  /*USO DO JOIN*/
9  SELECT *
10 FROM alunos a JOIN contatos c ON c.com_alu_iden = a.alu_iden
11      JOIN alunos_disciplinas ad ON a.alu_iden = ad.adi_alu_iden
12      JOIN disciplinas d ON d.dis_iden = ad.adi_dis_iden
13
```


SQL – DML

SELECT * FROM a
INNER JOIN b ON a.key = b.key



SELECT * FROM a
LEFT JOIN b ON a.key = b.key



SELECT * FROM a
RIGHT JOIN b ON a.key = b.key



POSTGRESQL JOINS



SELECT * FROM a
LEFT JOIN b ON a.key = b.key
WHERE b.key IS NULL



SELECT * FROM a
RIGHT JOIN b ON a.key = b.key
WHERE a.key IS NULL



SELECT * FROM a
FULL JOIN b ON a.key = b.key



SELECT * FROM a
FULL JOIN b ON a.key = b.key
WHERE a.key IS NULL OR b.key IS NULL



SQL – Operação Rename

- ✓ Proporciona um mecanismo para rebatizar tanto relações quanto atributos, usando a cláusula **AS**.

Nome_antigo **AS** nome_novo

- ✓ Pode estar tanto cláusula **SELECT** como **FROM**

SQL – Operação Rename

Query Editor Query History

```
1  /*COM USO DO 'AS'*/
2  SELECT usu_nome as nome, usu_renda as renda
3  FROM usuarios
4
5  /*SEM USO DO 'AS'*/
6  SELECT usu_nome nome, usu_renda renda
7  FROM usuarios
8
9
10
```

Data Output Explain Messages Notifications

| | nome | renda |
|---|------------------------|---------------|
| | character varying (80) | numeric (7,2) |
| 1 | MARIA | 8455.66 |
| 2 | JOAO | 1000.50 |

SQL – Variáveis Tuplas

- ✓ Uma variável tupla em SQL precisa estar associada a uma relação em particular (palavra chave **AS** opcional)

```
1  /*APELIDO PARA TABELA, OPCIONAL USAR 'AS'*/  
2  SELECT u.usu_nome, c.con_telefone  
3  FROM usuarios u, contatos c  
4  WHERE u.usu_iden = c.con_usu_iden
```

The diagram illustrates the tuple variables **u** and **c** used in the SQL query. Red boxes highlight the variables **u** and **c** in the **FROM** clause. Red arrows point from these variables to the **WHERE** clause, showing the join condition **u.usu_iden = c.con_usu_iden**. The **SELECT** clause shows the attributes **u.usu_nome** and **c.con_telefone** being retrieved.

SQL – Operações em Strings

- ✓ Operações sobre Strings mais usadas são as verificações de coincidências de pares
 - Operador **LIKE**
 - **(%)**: Compara qualquer substring
 - **(_)** : Compara qualquer caracter
 - Sensíveis(Maiúsculas / Minúsculas)
 - SQL **<>** sql

SQL – Operações em Strings

```
1  /*TENHA A EM QUALQUER PARTE */
2  SELECT * FROM usuarios u
3  WHERE u.usu_nome LIKE '%A%'
4
5  /*COMECE COM J */
6  SELECT * FROM usuarios u
7  WHERE u.usu_nome LIKE 'J%'
8
9  /*TERMINE COM IA */
10 SELECT * FROM usuarios u
11 WHERE u.usu_nome LIKE '%IA'
12
13 /*COMECE COM QUALQUER CARACTERE
14 TENHA O SEGUNDO COMO 'A' E
15 QUALQUER FINAL */
16 SELECT * FROM usuarios u
17 WHERE u.usu_nome LIKE '_A%'
```

Funções e operadores disponíveis para examinar e manipular valores cadeia de caracteres

Veja mais...

<https://www.postgresql.org/docs/11/functions-string.html>

SQL – Ordenação e apresentação de Tuplas


- ✓ SQL oferece ao usuário algum controle sobre a ordenação.

ORDER BY nome_atributo


- ✓ A cláusula ORDER BY relaciona os itens em ordem ascendente. Para especificar a forma de ordenação, devemos indicar:
 - ☐ **DESC** (Ordem Descendente)
 - ☐ **ASC** (Ordem Ascendente)

SQL – Ordenação e apresentação de Tuplas

```
1  /*ORDENAR NOME DE MÉDICO
2  ORDEM CRESCENTE - ASC É OPCIONAL*/
3  SELECT *
4  FROM medicos m
5  ORDER BY m.med_nome ASC
6
7  /*ORDENAR NOME DE MÉDICO
8  ORDEM DECRESCENTE*/
9  SELECT *
10 FROM medicos m
11 ORDER BY m.med_nome DESC
```



| | | | |
|---|-----------------|--------|--|
| 1 | Aarão Câmara | [null] | |
| 2 | Aarão Ribeiro | [null] | |
| 3 | Abílio Camacho | [null] | |
| 4 | Abílio Novalles | [null] | |



| | med_nome character varying | med_crm character varyir |
|---|-------------------------------|-----------------------------|
| 1 | Zuriel Fragoso | [null] |
| 2 | Zuriel Carrasqueira | [null] |
| 3 | Zuleide Robalinho | [null] |

SQL – Modificações no BD

✓ **REMOÇÃO**

Podemos remover somente tuplas inteiras; mas não podemos excluir valores de uma célula em particular

```
DELETE FROM r  
WHERE P
```



SQL – Modificações no BD

✓ REMOÇÃO



```
1  /*REMOVER O MÉDICO ALCIDES VILLANUEVA*/  
2  DELETE FROM medicos m  
3  WHERE m.med_nome = 'Alcides Villanueva'  
4
```

```
1  /*REMOVER FILMES ENTRE 2015 E 2019  
2  QUE O TÍTULO COMEÇEM COM A*/  
3  DELETE FROM filmes f  
4  WHERE f.fil_ano BETWEEN 2015 AND 2019  
5  AND f.fil_titulo LIKE 'A%'
```

SQL – Modificações no BD

✓ REMOÇÃO



SQL – Modificações no BD

✓ **Não confunda!**

✓ **DROP TABLE** r

✓ ***DELETE FROM** r

✓ ***TRUNCATE FROM** r

Delete mantém a tabela r, mas remove todas as suas tuplas. **Drop** não remove apenas todas as tuplas de r, mas também seu esquema.

SQL – Modificações no BD

✓ **ATUALIZAÇÕES**

- Em determinadas situações, podemos querer modificar valores das tuplas sem, no entanto, alterar todos os valores. Utilizando **UPDATE**

UPDATE tabela

SET campo = novo_valor



SQL – Modificações no BD

✓ ATUALIZAÇÕES



[Redacted] added 3 new photos — with [Redacted] and 3 others.

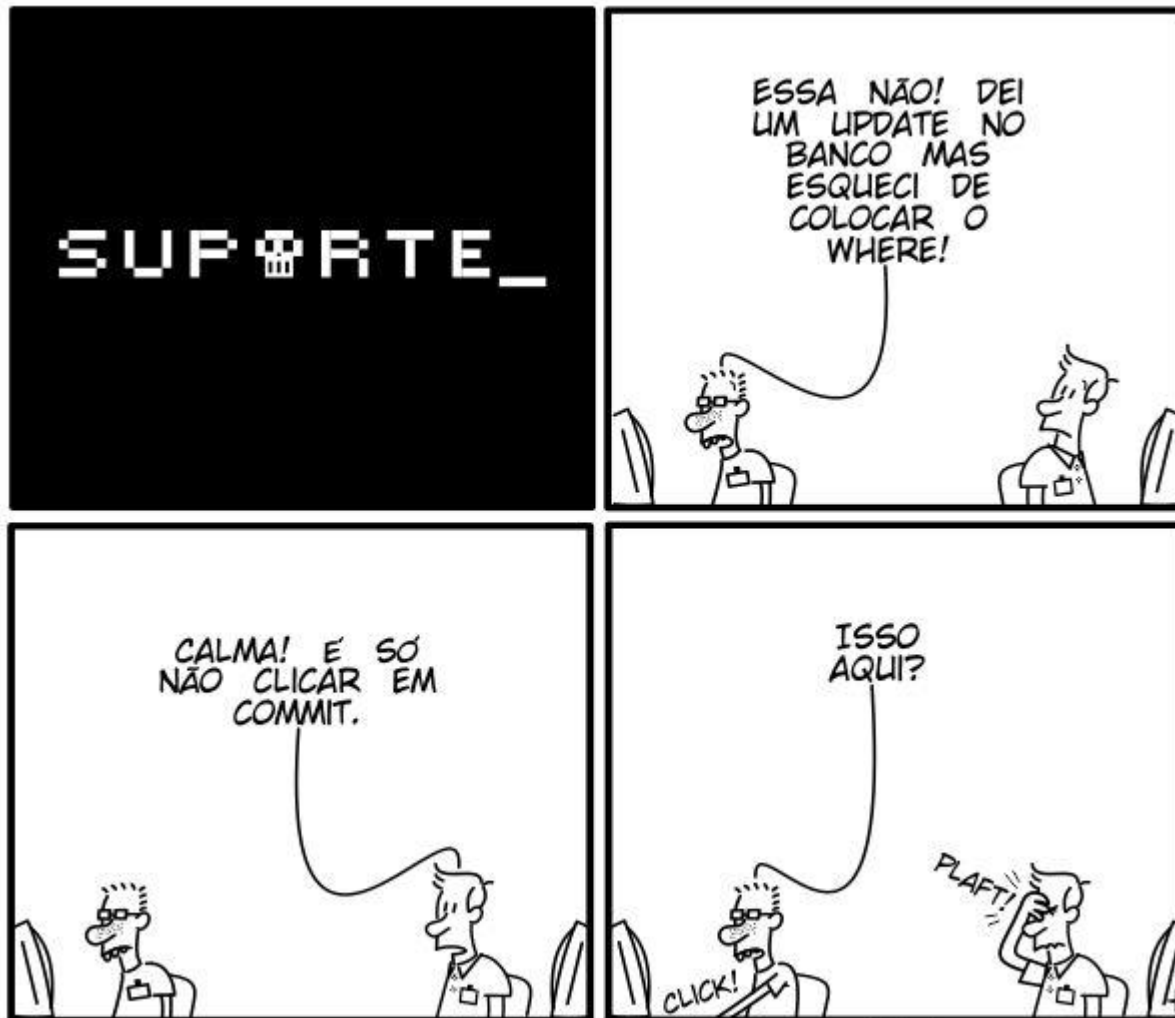
2 hrs · Joinville, SC · 📍

Hoje mais um certificado foi entregue na empresa... Vamos comemorar com o querido colaborador V [Redacted] ... (foi em base TESTE)



SQL – Modificações no BD

✓ ATUALIZAÇÕES



SQL – Modificações no BD

✓ ATUALIZAÇÕES

```
1  /*ATRIBUIR 10000 PARA MARIA*/
2  UPDATE usuarios
3  SET usu_renda = 10000
4  WHERE usu_nome = 'MARIA'
5
6  /*ATRIBUIR MAIS 50 NO VALOR
7  CONTIDO EM USU_RENDA*/
8  UPDATE usuarios
9  SET usu_renda = usu_renda + 50
10 WHERE usu_nome = 'MARIA'
```


SQL – Funções Agregadas

- São funções que tomam uma coleção (um conjunto ou subconjunto) de valores como entrada, retornando um valor simples.
- **AVG** - Média (Average)
- **MIN** - Mínimo (Minimum)
- **MAX** - Máximo (Maximum)
- **SUM** - Total (Soma)
- **COUNT** - Contagem (Count)

SQL – Funções Agregadas

- A entrada para SUM e AVG precisa ser um conjunto de números.

```
SELECT AVG (atributo_numerico)  
FROM tabela  
WHERE condição
```


```
SELECT SUM (atributo_numerico)  
FROM tabela  
WHERE condição
```

```
SELECT COUNT (*)  
FROM tabela
```

SQL – Funções Agregadas

```
1  /*SOMA*/
2  SELECT SUM(vin_salario)
3  FROM vinculos;
4
5  /*MÉDIA*/
6  SELECT AVG(vin_salario)
7  FROM vinculos;
8
9  /*CONTADOR*/
10 SELECT COUNT(vin_salario)
11 FROM vinculos;
12
```

Conta a
quantidade de
tuplas (linhas)



SQL – Funções Agregadas

- Podemos utilizar **funções agregadas** em um **grupo de conjuntos** de tuplas, utilizando **GROUP BY**

Tem que
aparecer
aqui

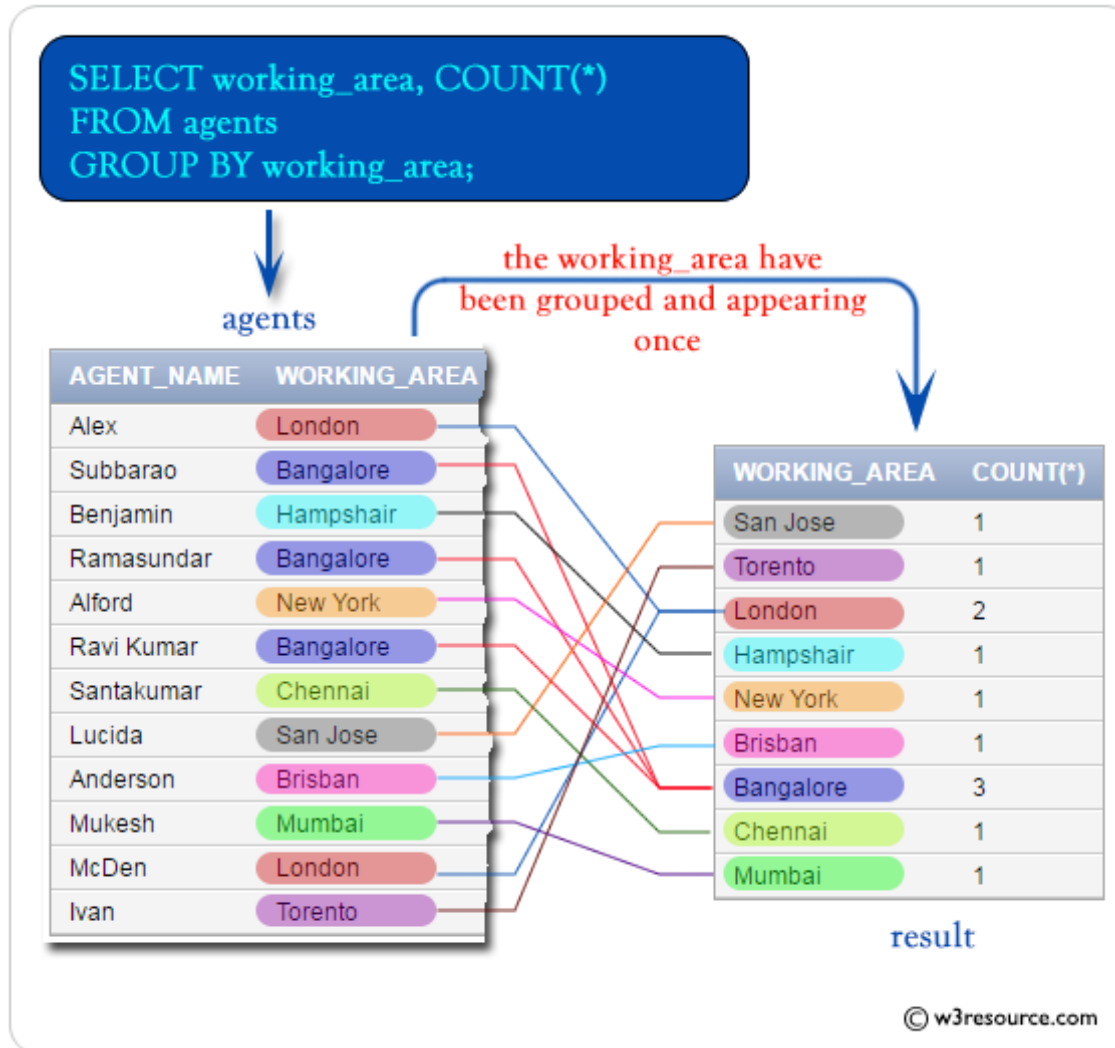
```
SELECT atributo1, AVG (atributo2)  
FROM tabela  
GROUP BY atributo1
```

Agrupa
colunas da
tabela

```
1  /*AGRUPA O SALÁRIO DOS MÉDICOS POR CLÍNICA*/  
2  SELECT c.cli_nome, SUM(vin_salario)  
3  FROM clinicas c JOIN vinculos v ON c.cli_iden = v.vin_cli_iden  
4  GROUP BY c.cli_nome;
```

- Obs. Atributo apareceu no Select com função agregada deve-se usar o GROUP BY.

SQL – Funções Agregadas



SQL – Funções Agregadas

Conta

```
1 SELECT c.cat_nome, count(*)  
2 FROM filmes f JOIN filmes_categorias fc ON fc.fca_fil_iden = f.fil_iden  
3 JOIN categorias c ON c.cat_iden = fc.fca_cat_iden  
4 GROUP BY c.cat_nome
```

Grupo

| filme character varying (255) | categoria character varying (25) |
|---------------------------------------|-------------------------------------|
| Godzilla Resurgence | Ação |
| The Hobbit: An Unexpected Journey | Aventura |
| The Fast and the Furious | Ação |
| The Curious Case of Benjamin Button | Drama |
| X-Men: First Class | Ação |
| The Hunger Games: Mockingjay - Part 2 | Aventura |
| The Sorcerers Apprentice | Ação |
| Poseidon | Ação |

5

1

2

SQL – Funções Agregadas

- Definir condições a grupos podemos utilizar a cláusula **HAVING**.
 - Predicados da cláusula **HAVING** são aplicadas depois da formação de grupos.

```
SELECT atributo1, AVG (atributo2)
FROM tabela
GROUP BY atributo1
HAVING AVG (atributo2) > 1200
```

Obs. Having são where dos grupos, ou seja, Restringir as linhas oriundas do Group By

<https://www.w3resource.com/sql/aggregate-functions/Max-having.php>

```
SELECT cust_city, cust_country,
MAX(outstanding_amt)
FROM customer
GROUP BY cust_country, cust_city
HAVING MAX(outstanding_amt)>10000;
```

| CUST_CITY | CUST_COUNTRY | OUTSTAND |
|-----------|--------------|----------|
| Bangalore | India | 12000 |
| London | UK | 4000 |
| New York | USA | 6000 |
| New York | USA | 6000 |
| Bangalore | India | 8000 |
| London | UK | 6000 |
| London | UK | 11000 |
| New York | USA | 3000 |
| Brisban | Australia | 5000 |
| Brisban | Australia | 7000 |
| Chennai | India | 8000 |
| Mumbai | India | 11000 |
| Chennai | India | 9000 |

customer

```
SELECT cust_city, cust_country,
MAX(outstanding_amt)
FROM customer
GROUP BY cust_country, cust_city;
```

| CUST_CITY | CUST_CO | MAX(OUTSTA |
|-----------|-----------|------------|
| Bangalore | India | 12000 |
| Brisban | Australia | 7000 |
| Chennai | India | 11000 |
| Hampshair | UK | 5000 |
| London | UK | 11000 |
| Mumbai | India | 12000 |
| New York | USA | 6000 |
| San Jose | USA | 3000 |
| Toronto | Canada | 11000 |

```
HAVING MAX(outstanding_amt)>10000;
```

| CUST_CITY | CUST_CO | MAX(OUTSTA |
|-----------|-----------|------------|
| Bangalore | India | 12000 |
| Brisban | Australia | 7000 |
| Chennai | India | 11000 |
| Hampshair | UK | 5000 |
| London | UK | 11000 |
| Mumbai | India | 12000 |
| New York | USA | 6000 |
| San Jose | USA | 3000 |
| Toronto | Canada | 11000 |

| CUST_CITY | CUST_COUNTRY | MAX(OUTSTA |
|-----------|--------------|------------|
| Bangalore | India | 12000 |
| Chennai | India | 11000 |
| London | UK | 11000 |
| Mumbai | India | 12000 |
| Toronto | Canada | 11000 |

<https://www.w3resource.com/sql/aggregate-functions/Max-having.php>

SQL – Funções Agregadas

- Definir condições a grupos podemos utilizar a cláusula **HAVING**.
 - Predicados da cláusula **HAVING** são aplicadas depois da formação de grupos.

```
1  /*RETORNAR OS MÉDICOS QUE POSSUEM MAIS DE UM ACOMPANHAMENTO*/
2  SELECT count(*), m.med_nome
3  FROM medicos m JOIN acompanhamentos a ON m.med_iden = a.acom_med_iden
4         JOIN pacientes p ON a.acom_pac_iden = p.pac_iden
5  GROUP BY m.med_nome
6  HAVING COUNT(*) > 1
7
```

SQL – Funções Agregadas

- Se uma cláusula **WHERE** e uma cláusula **HAVING** aparecem na mesma consulta, o predicado que aparece em **WHERE** é aplicado primeiro
- Tuplas que satisfazem a cláusula **WHERE** são, agrupados por meio da cláusula **GROUP BY**. A cláusula **HAVING** aplica ao grupo.

SQL – Funções Agregadas

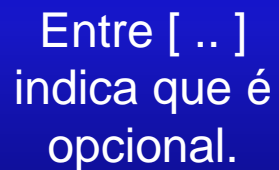
Diferenças

- **Where:** seleciona as linhas antes dos grupos e funções agregadas sejam computadas (Ou seja, seleciona as linhas que poderão ser computadas)
- **Having:** Seleciona linhas do grupo após o cálculo das funções agregadas.

SQL – LIMIT

Podemos selecionar a **quantidade exata de linhas** que vão ser retornadas a partir de um SELECT.

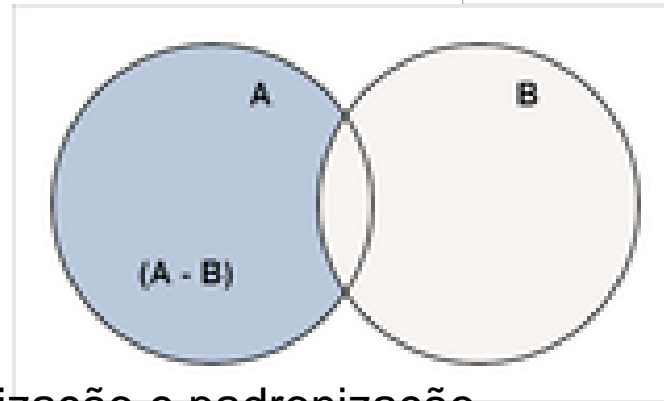
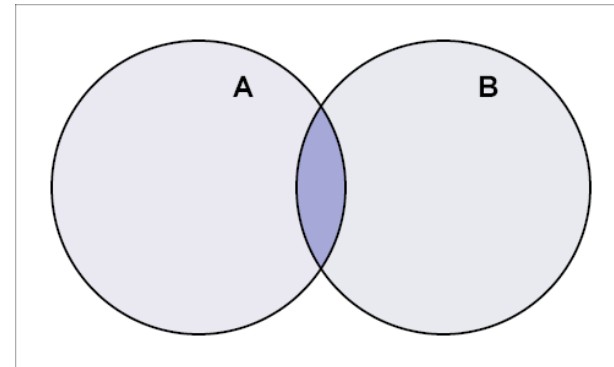
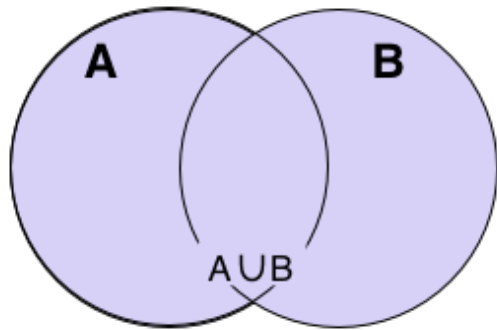
```
SELECT coluna1, coluna2,...  
FROM tabela  
[LIMIT { quantidade | ALL }]  
[OFFSET quantidade_pular_inicio]
```



Entre [..]
indica que é
opcional.

SQL – Operações de Conjuntos

- ✓ Operadores SQL-92* (UNION, INTERSECT e EXCEPT) correspondem a operações da álgebra relacional



*Entidades de normalização e padronização

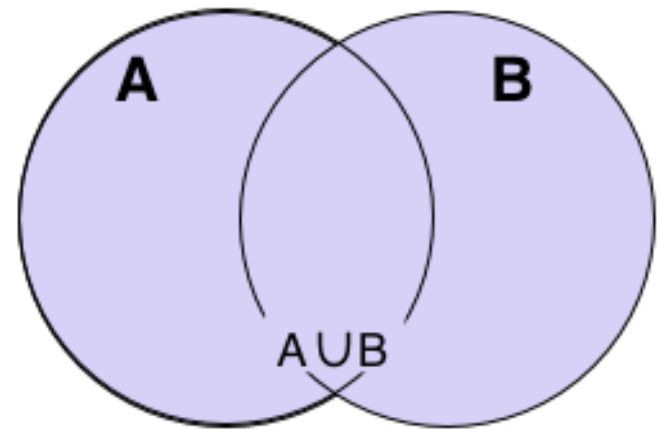
SQL – Operação de União

- A operação de **UNION**, ao contrário de cláusula **SELECT**, automaticamente elimina as repetições / Variação **ALL**

(**SELECT** atributos
FROM tabela)

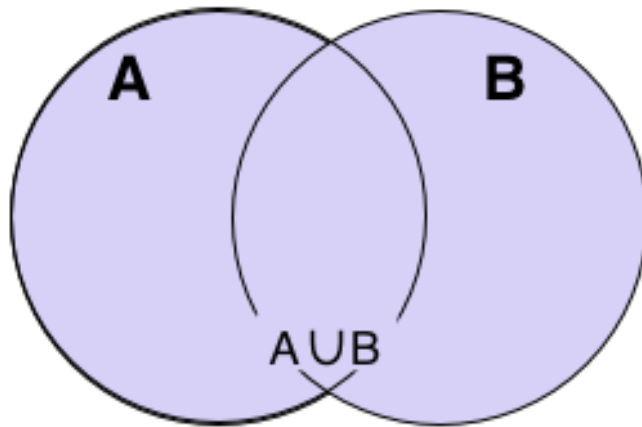
UNION / ALL

(**SELECT** atributos
FROM tabela)



- Obs. Deve-se sempre comparar os mesmos atributos de cada tabela contendo os mesmos tipos de dados.

SQL – Operação de União



+



A+B

```
1  (SELECT m.med_nome, m.med_crm
2  FROM medicos m
3  WHERE m.med_crm IS NOT null)
4  UNION
5  (SELECT m.med_nome, m.med_crm
6  FROM medicos m WHERE m.med_crm IS NULL)
7  ,
```

SQL – Operação de União

| Table 1 | |
|----------|----------|
| Column 1 | Column 2 |
| a | a |
| a | b |
| a | c |

UNION



| Table 2 | |
|----------|----------|
| Column 1 | Column 2 |
| b | a |
| a | b |
| b | c |

| Result | |
|----------|----------|
| Column 1 | Column 2 |
| a | a |
| a | b |
| a | c |
| b | a |
| b | c |

Duplicações são exibidas em uma única linha

| Table 1 | |
|----------|----------|
| Column 1 | Column 2 |
| a | a |
| a | b |
| a | c |

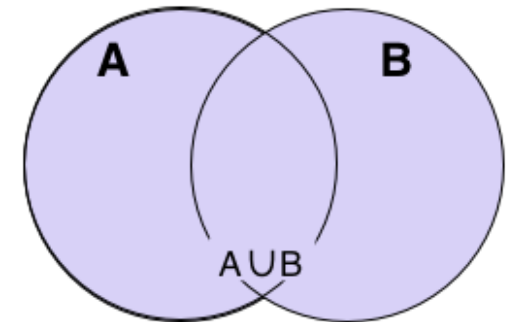
UNION ALL



| Table 2 | |
|----------|----------|
| Column 1 | Column 2 |
| b | a |
| a | b |
| b | c |

| Result | |
|----------|----------|
| Column 1 | Column 2 |
| a | a |
| a | b |
| a | b |
| a | c |
| b | a |
| b | c |

As duplicações são exibidas no resultado



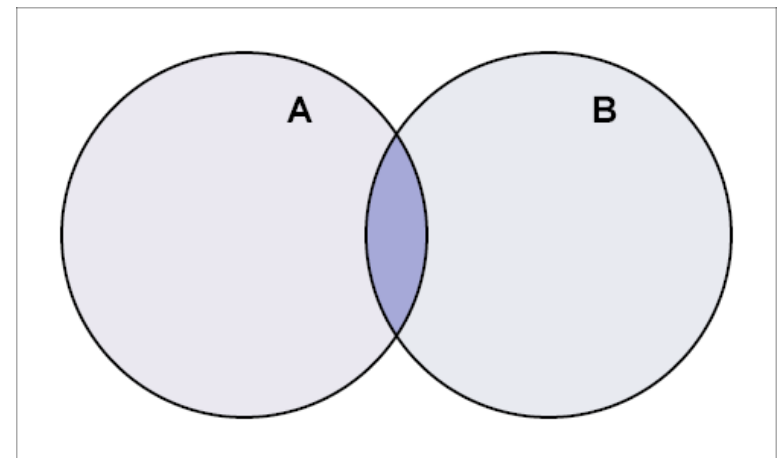
SQL – Operação de Interseção

- A operação de **INTERSECT** automaticamente elimina todas as repetições. Variante **ALL**

(**SELECT** atributos
FROM tabela)

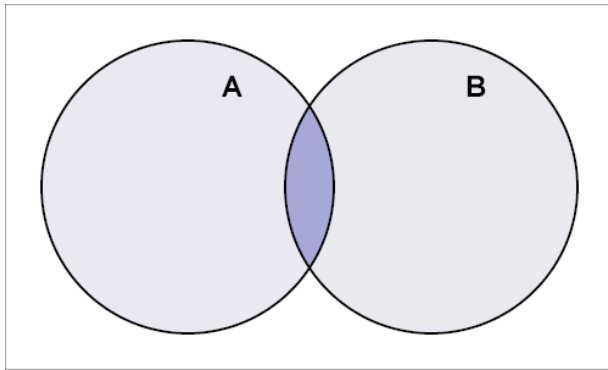
INTERSECT / ALL

(**SELECT** atributos
FROM tabela)



Jones tem diversas contas e empréstimos no banco, então Jones aparecerá somente uma vez, caso contrário usar **ALL**

SQL – Operação de Interseção



CASAS



VEÍCULOS

```
1 (SELECT m.med_nome
2 FROM medicos m
3 WHERE m.med_crm IS NOT null)
4 INTERSECT
5 (SELECT m.med_nome
6 FROM medicos m WHERE m.med_crm IS NULL)
7
```

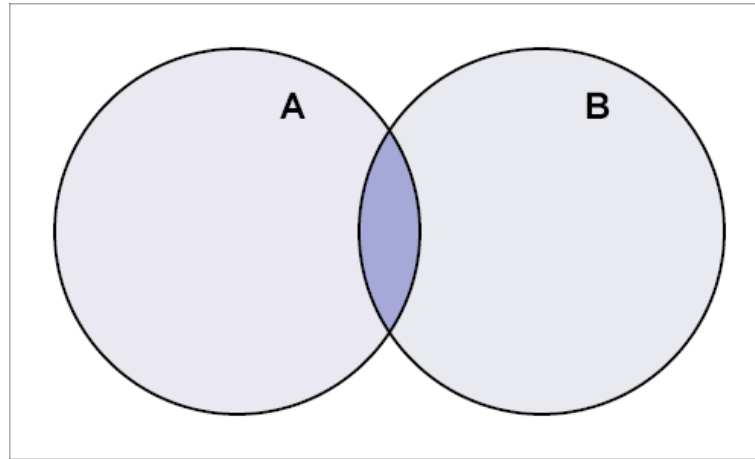
Data Output Explain Messages Notifications

| | med_nome | |
|---|-------------------|---|
| | character varying | 🔒 |
| 1 | MARCIO SOUZA | |



CASA / VEÍCULO

SQL – Operação de Interseção

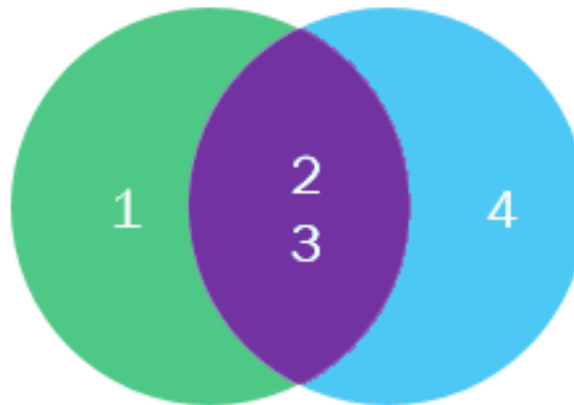


| |
|---|
| 1 |
| 2 |
| 3 |

A

| |
|---|
| 2 |
| 3 |
| 4 |

B



A INTERSECT B



| |
|---|
| 2 |
| 3 |

RESULT

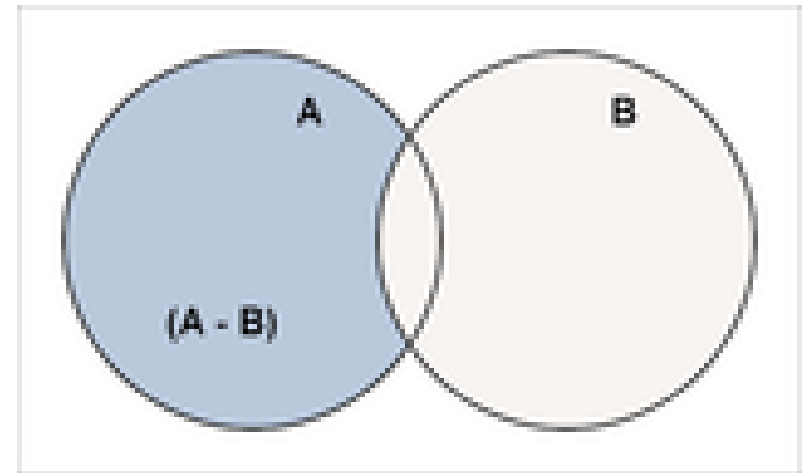
SQL – Operação de Diferença

Except irá seguir a mesma lógica dos exemplos anteriores.

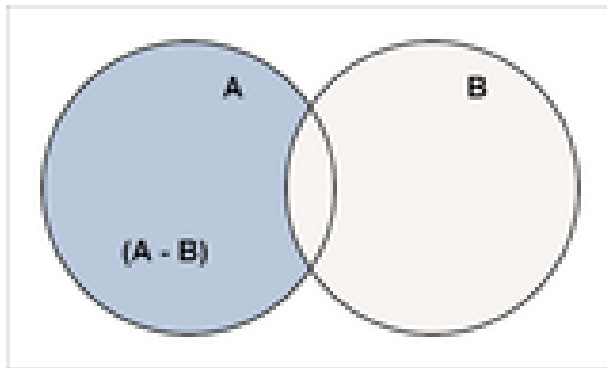
(SELECT atributos
FROM tabela)

EXCEPT / ALL

(SELECT atributos
FROM tabela)



SQL – Operação de Diferença



CASAS



VEÍCULOS

```
1 (SELECT m.med_nome
2  FROM medicos m
3  WHERE m.med_crm IS NOT null)
4  EXCEPT
5  (SELECT m.med_nome
6   FROM medicos m WHERE m.med_crm IS NULL)
7
```

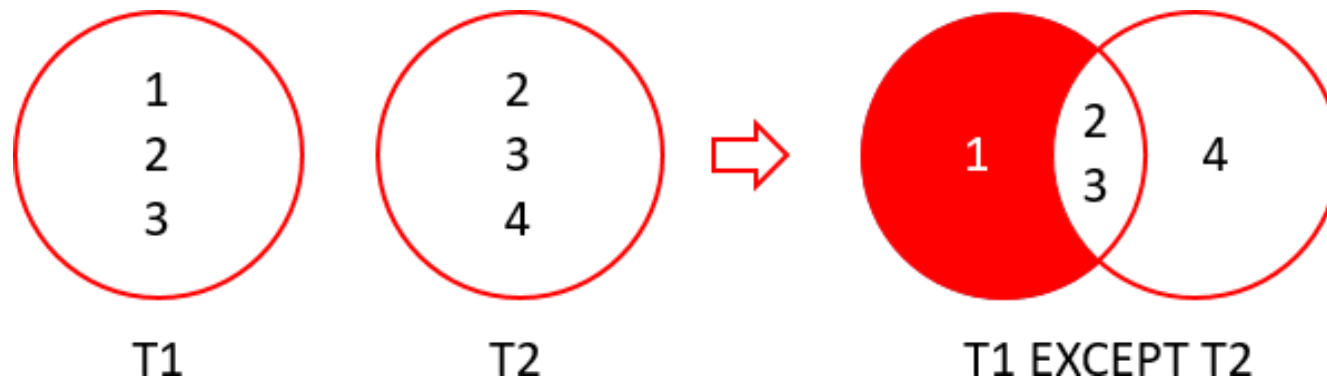
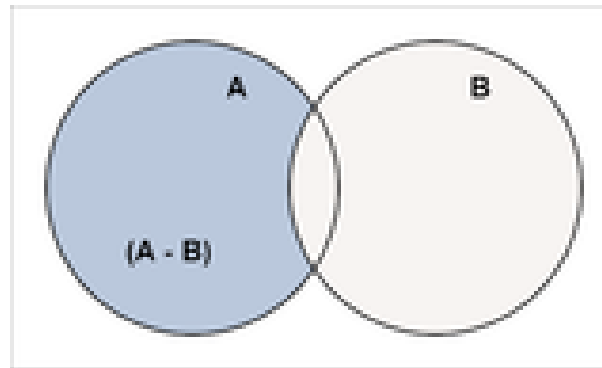
Data Output Explain Messages Notifications

| | med_nome character varying | |
|---|-------------------------------|--|
| 1 | MARIA APARECIDA | |
| 2 | AMADO BATISTA | |
| 3 | JOANA GOMES | |

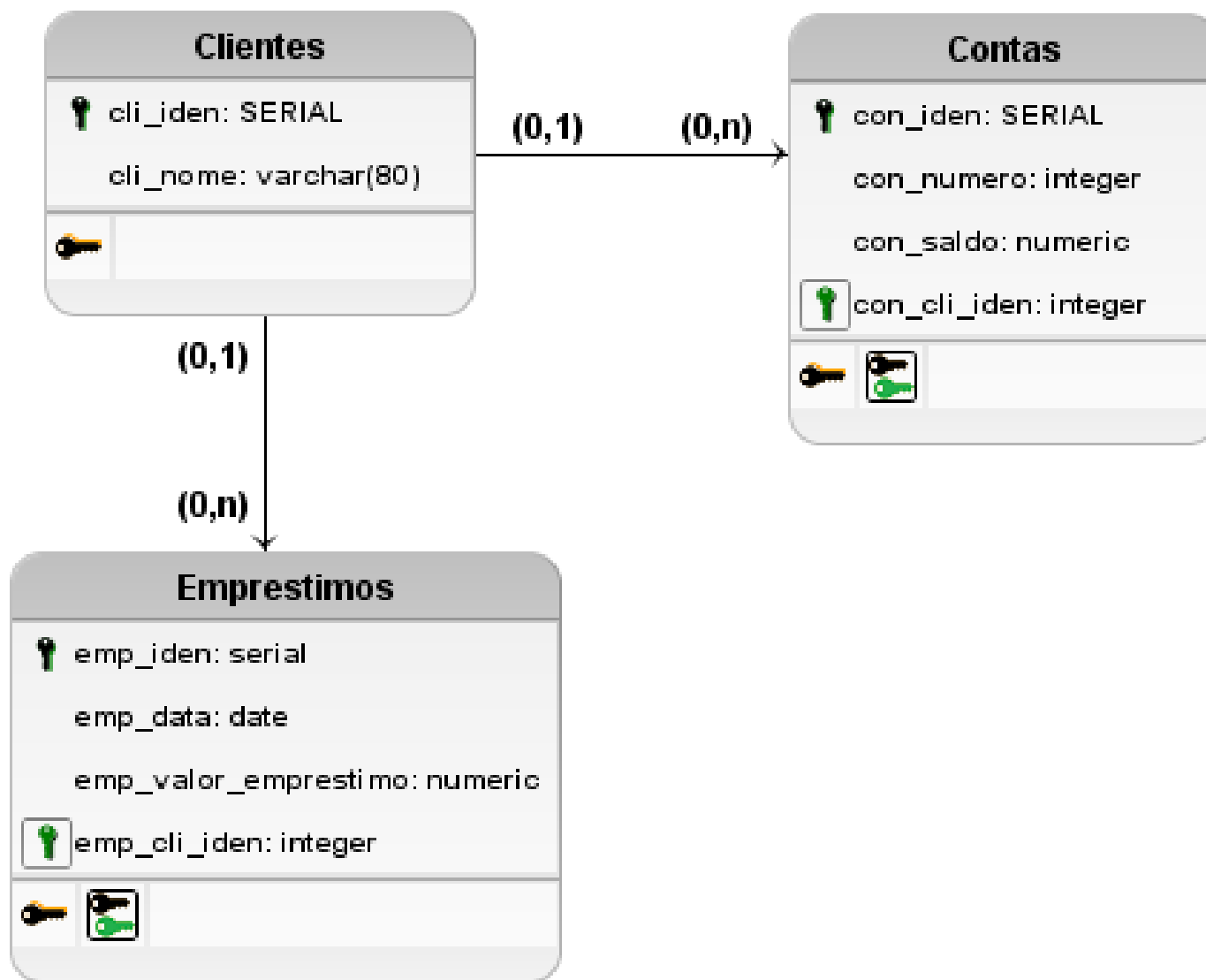


CASA / VEÍCULO

SQL – Operação de Diferença



SQL – Modificações no BD



SQL – Modificações no BD

✓ **INSERÇÃO**

- Pode-se inserir vários dados com base em uma consulta SELECT. Obs, deve-se retornar a sequência dos atributos de acordo com a definição na inserção.
- Passos:
 - Faça a sua consulta

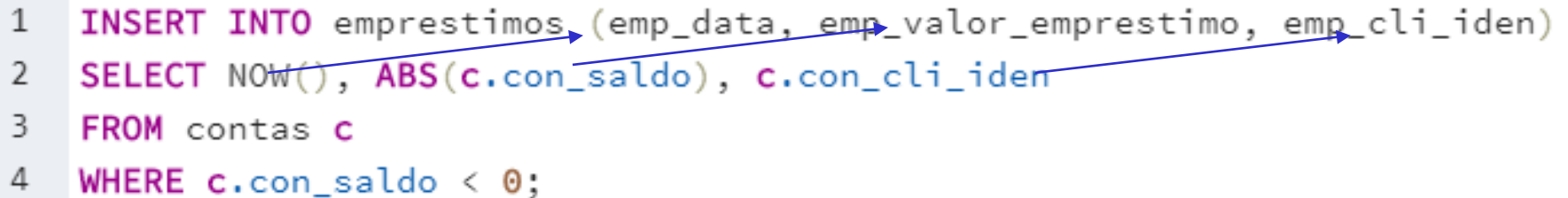
```
1  /*RETORNAR CONTAS COM SALDO NEGATIVO*/  
2  SELECT * FROM contas c  
3  WHERE c.con_saldo < 0
```


SQL – Modificações no BD

✓ **INSERÇÃO**

2. Escolha a tabela destino: Empréstimos
3. Faça uma adaptação do SELECT (da consulta 1) para ser compatível com os campos da tabela destino (passo 2)

```
1  INSERT INTO empréstimos (emp_data, emp_valor_emprestimo, emp_cli_iden)
2  SELECT NOW(), ABS(c.con_saldo), c.con_cli_iden
3  FROM contas c
4  WHERE c.con_saldo < 0;
```



SQL – Modificações no BD

✓ **INSERÇÃO**

- Podemos usar uma **cláusula SELECT** para especificar um conjunto de tuplas. O comando SELECT é realizado primeiro, resultando em um conjunto de tuplas que é então **inserido em uma relação** conta.
- É possível, na inserção de tuplas, fornecer valores somente para **alguns atributos** do esquema

SQL – Modificações no BD

✓ **REMOÇÃO**

- Desejamos remover os registros de todos os empréstimos com saldos abaixo da média do saldo da conta.
 - 1) Monte à consulta.

```
1  /*OBTER A MÉDIA DAS CONTAS*/
2  /*TRUNCA*/
3  SELECT TRUNC(AVG(ABS(c.con_saldo)),2)
4  FROM contas c
5
6  /*ARREDONDA*/
7  SELECT ROUND(AVG(ABS(c.con_saldo)),2)
8  FROM contas c
```

SQL – Modificações no BD

✓ **REMOÇÃO**

- Associe a consulta ao comando DELETE.

```
1 DELETE FROM empréstimos e
2 WHERE e.emp_valor_empréstimo
3       <= (SELECT ROUND(AVG(ABS(c.con_saldo)),2)
4           FROM contas c)
5
```

SQL – Modificações no BD

✓ ATUALIZAÇÕES

- Disponibilize mais R\$500,00 a cada empréstimo cujo o valor seja menor que o valor máximo dos saldos da tabela contas.

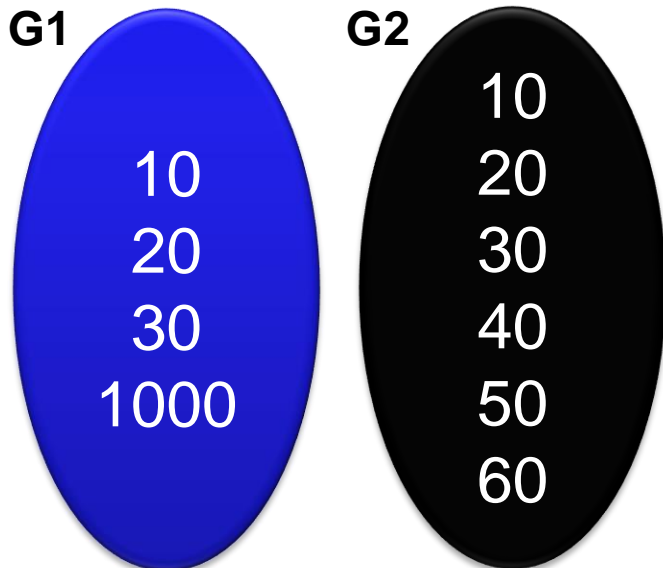
```
1  UPDATE emprestimos e
2  SET emp_valor_emprestimo = emp_valor_emprestimo + 500
3  WHERE e.emp_valor_emprestimo
4         <= (SELECT ROUND(MAX(ABS(c.con_saldo)),2)
5              FROM contas c)
```

SQL – Subconsultas Aninhadas

Expressão **IN** (Subconsulta)

Lado direito deverá retornar uma consulta com uma **única coluna**.

A expressão (esquerda) é comparada linha a linha com a subconsulta. Será **true** se qualquer linha da subconsulta for **igual** a expressão



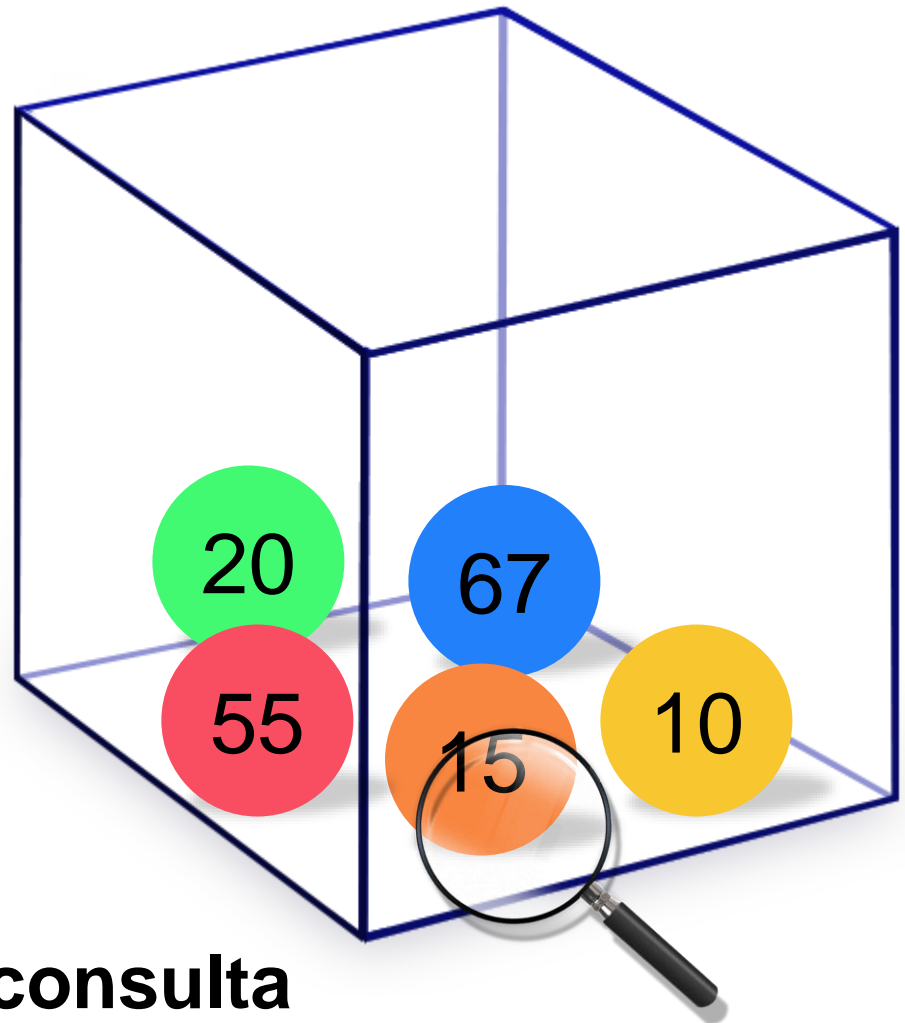
```
SELECT * FROM grupo g1  
WHERE g1.gru_valor IN  
(SELECT g2.gru_valor FROM grupo2 g2)
```

Resultado: 10, 20 e 30

SQL – Subconsultas Aninhadas

SELECT 15 IN

**Ir  comparar 15 com
cada resultado da subconsulta**



SQL – Subconsultas Aninhadas

- ❑ Caso contrário ao (IN), o NOT IN será verdadeiro caso não retorne nenhuma linha.

```
SELECT * FROM grupo g1
WHERE g1.gru_valor NOT IN
(SELECT g2.gru_valor FROM grupo2 g2)
```

Conjunto enumerado

```
SELECT * FROM grupo g1
WHERE g1.gru_valor NOT IN (20,30)
```


SQL – Comparação de Conjuntos

expressão **operador** **ANY/SOME** (subconsulta)

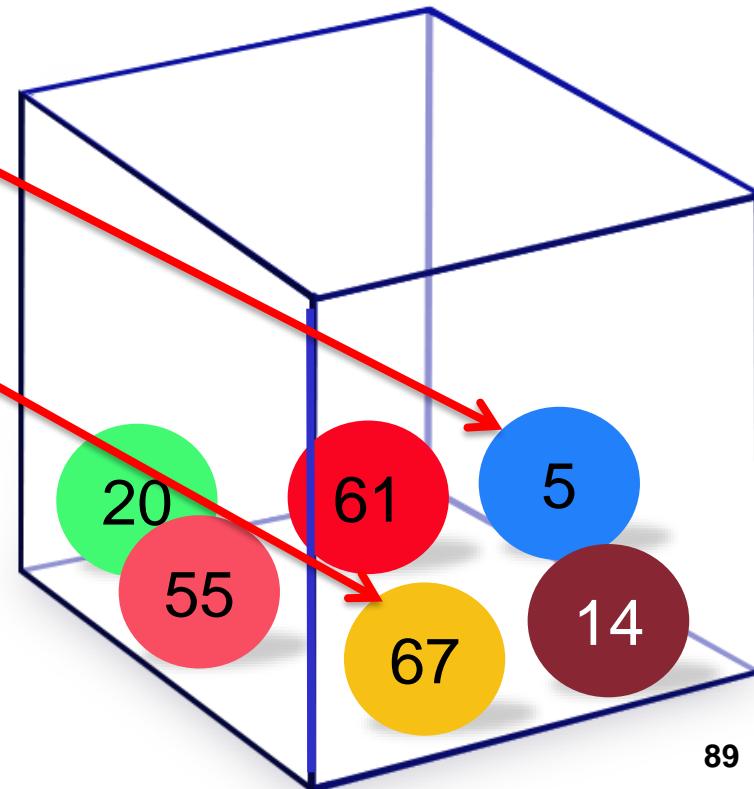
SOME é sinônimo de ANY
IN é equivalente a ANY

WHERE 70 > SOME (
Pelo menos um valor
maior que o mínimo

WHERE 70 < SOME (
Pelo menos um valor
menor que o máximo

WHERE 2 > SOME (
Pelo menos um valor
maior que o mínimo

WHERE 2 < SOME (
Pelo menos um valor
menor que o máximo



SQL – Comparação de Conjuntos

```
/*PELO MENOS UM VALOR MAIOR QUE O MÍNIMO*/  
SELECT * FROM grupo g1  
WHERE g1.gru_valor > SOME  
(SELECT g2.gru_valor FROM grupo2 g2)
```

```
/*PELO MENOS UM VALOR MENOR QUE O MÁXIMO*/  
SELECT * FROM grupo g1  
WHERE g1.gru_valor < SOME  
(SELECT g2.gru_valor FROM grupo2 g2)
```

- A frase “maior que apenas uma” é representada em SQL por **> SOME**

SQL – Comparação de Conjuntos

expressão operador **ALL** (subconsulta)

True, se todas as linhas produzem true

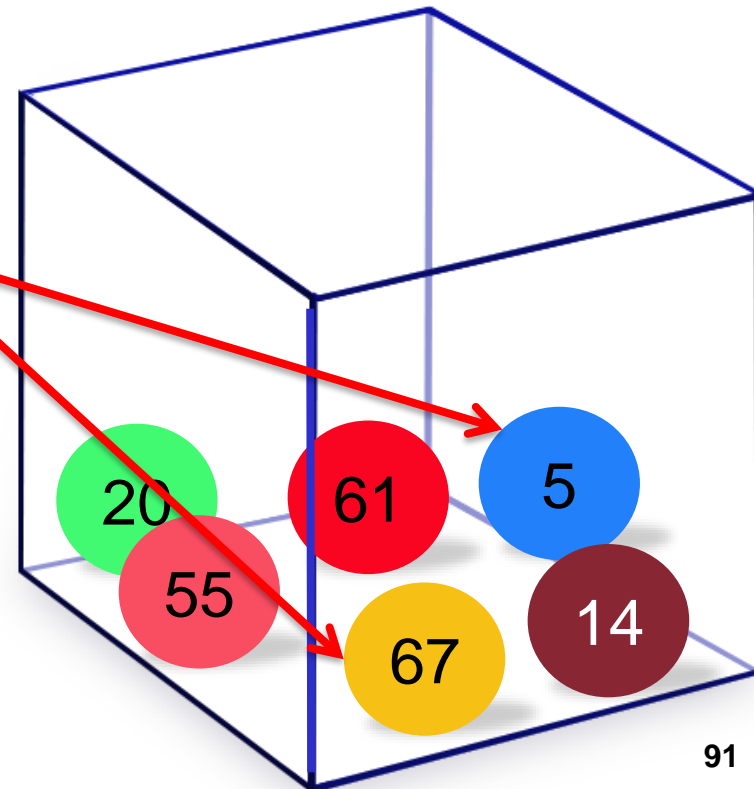
NOT IN é equivalente $<>$ ALL

WHERE 70 > ALL (
É maior que o valor
máximo

WHERE 70 < ALL (
Menor que o valor
mínimo

WHERE 2 > ALL (
É maior que o valor
máximo

WHERE 2 < ALL (
Menor que o valor
mínimo



SQL – Comparação de Conjuntos

expressão operador **ALL** (subconsulta)

True, se todas as linhas produzem true

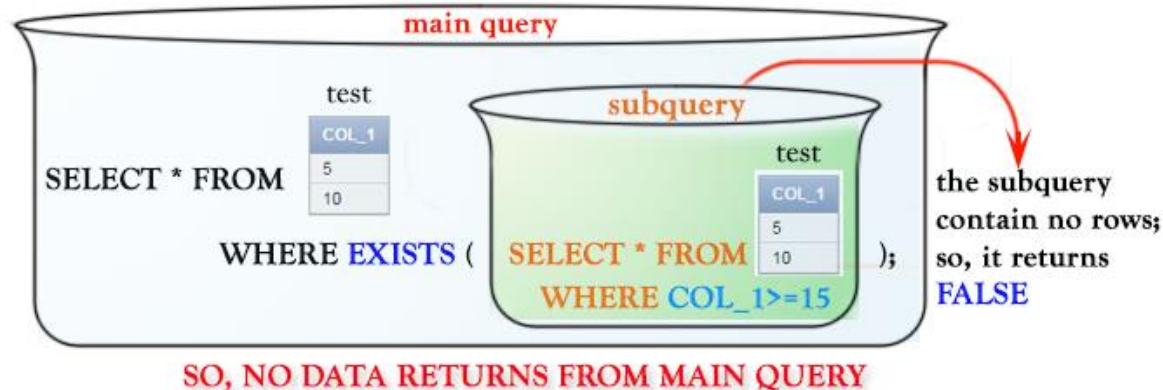
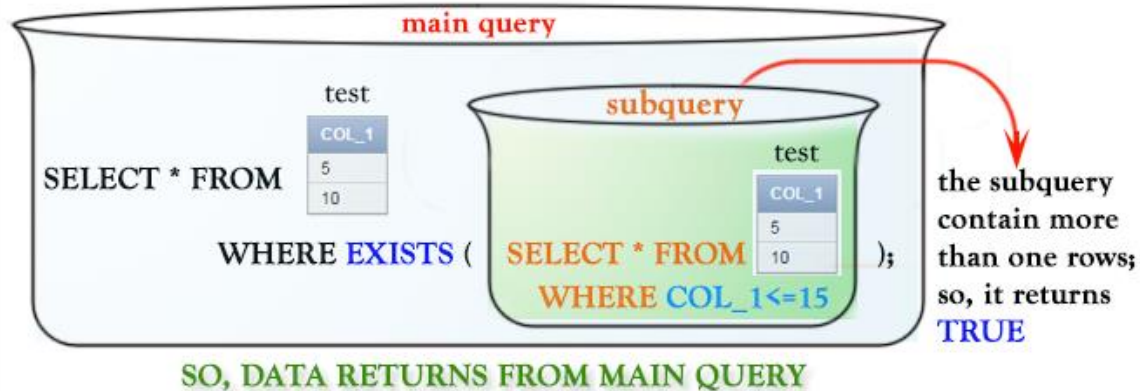
NOT IN é equivalente <> ALL

```
/*RETORNA O VALOR QUE É MAIOR QUE TODOS*/  
SELECT * FROM grupo g1  
WHERE g1.gru_valor > ALL  
(SELECT g2.gru_valor FROM grupo2 g2)
```

WHERE EXISTS (**subquery**);

EXISTS operator

- EXISTS is a Comparison operator
- Used in WHERE clause to validate an "IT EXISTS" condition.
- EXISTS will tell you whether a query returned any results.
- Returns a BOOLEAN, (TRUE or FALSE).
- Returns TRUE if a subquery contains any rows.



SQL – Comparação de Conjuntos

```
--EXECUTA POR ÚLTIMO
```

```
SELECT * FROM grupo gr  
WHERE EXISTS
```

```
--EXECUTA PRIMEIRO
```

```
(SELECT * FROM grupo g  
WHERE g.gru_valor >= 1000)
```

**Retorna true se a subquery
retornar um ou mais
resultados**

```
SELECT * FROM grupo gr  
WHERE gr.gru_valor > 30 AND  
EXISTS (SELECT * FROM grupo g  
WHERE g.gru_valor >= 1000)
```

```
--EXECUTA POR ÚLTIMO
```

```
SELECT * FROM grupo gr  
WHERE EXISTS
```

```
--EXECUTA PRIMEIRO
```

```
(SELECT 1 FROM grupo g  
WHERE g.gru_valor >= 1000  
LIMIT 1)
```