



Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

SQL

Structured Query Language

Prof. Dr. Halley Wesley Gondim

halley.was@gmail.com

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

- ✓ 1987 - IBM publicou seus próprios padrões
- ✓ SQL foi revisto em SQL-1992, 99(SQL-3) e 2003
 - SQL-99 (consultas recursivas e gatilhos)
 - SQL-2003 (facilidade em manipulação xml)
- ✓ Linguagem de Definição de Dados (DDL)
 - Comandos definição de esquemas, exclusão, criação de índices e modificação nos esquemas de relações

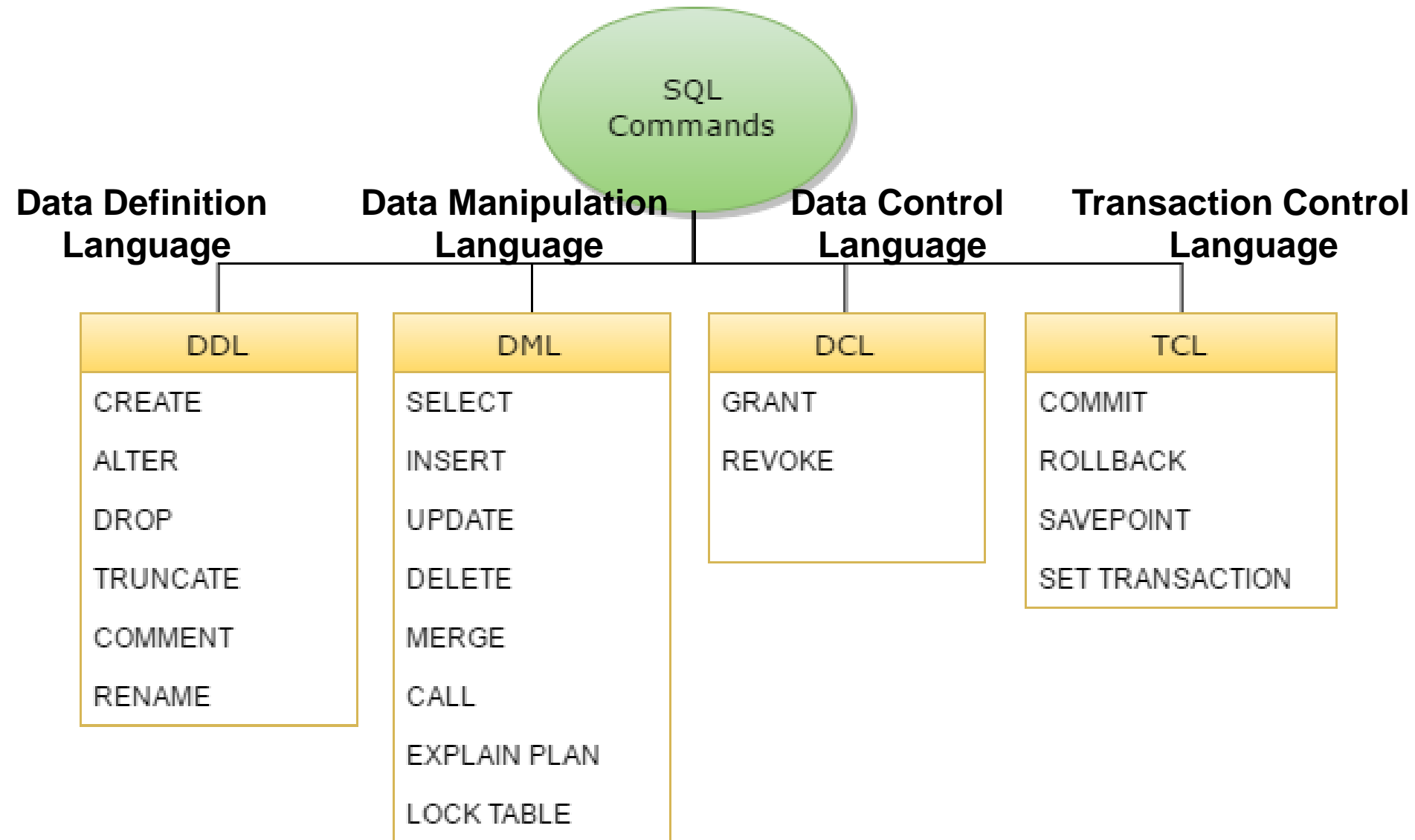
SQL - Histórico

- ✓ 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 )
```

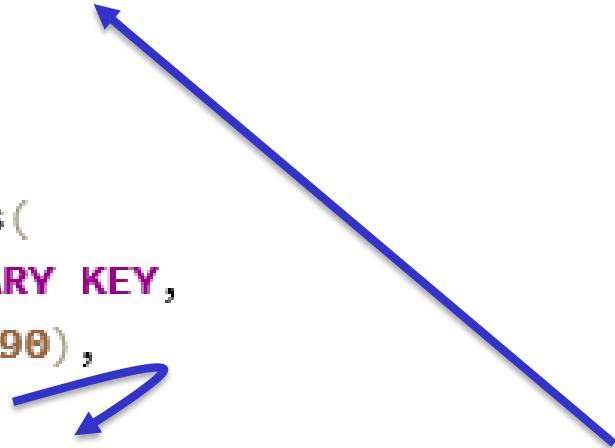
SQL – Sequence

✓Corpo de uma sequence

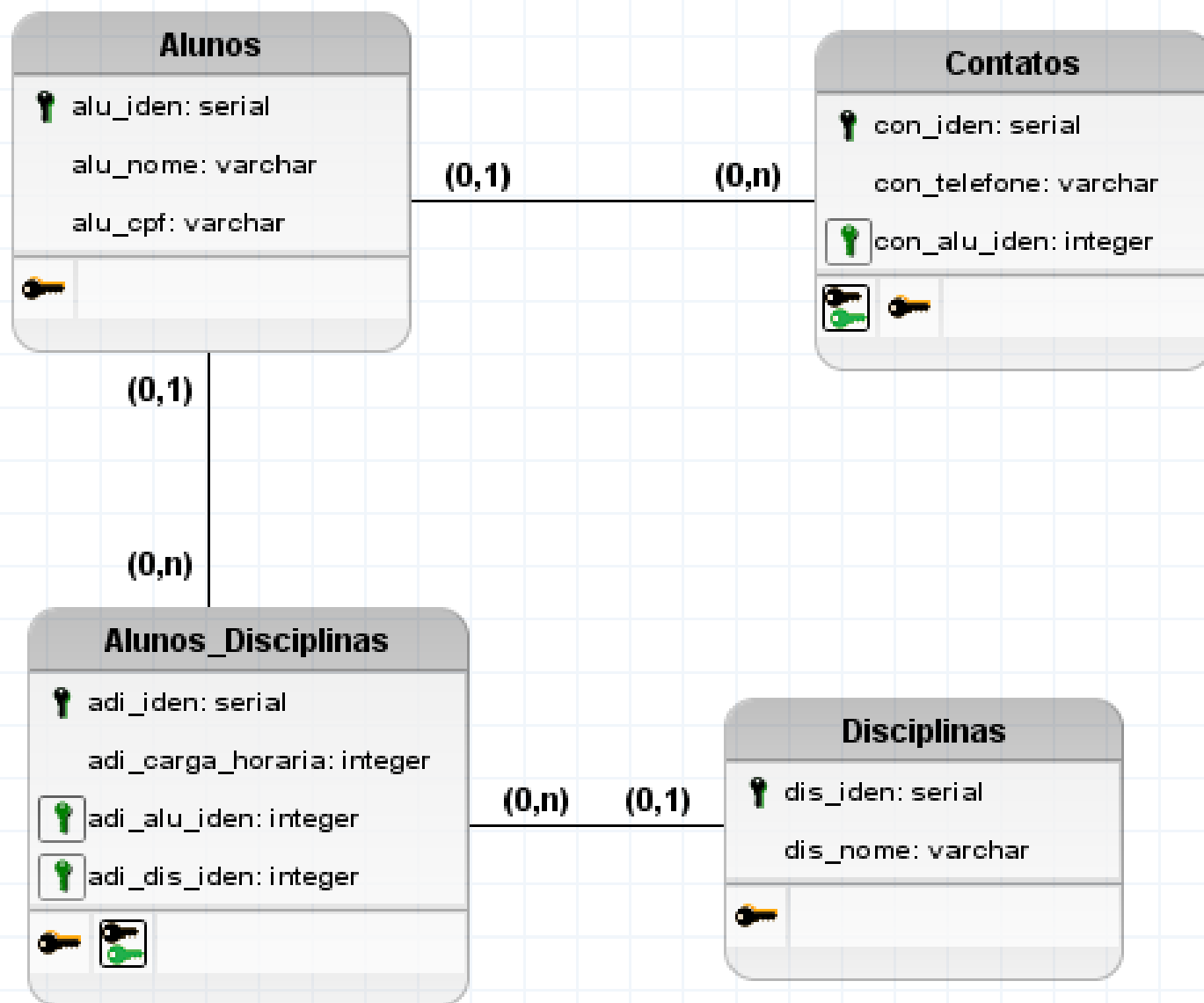
```
1 CREATE SEQUENCE public.nome_da_tabela_seq
2     INCREMENT 1
3     START 1
4     MINVALUE 1
5     MAXVALUE 2147483647
6     CACHE 1;
7
```

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)
```

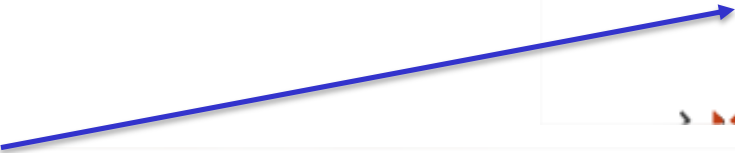
SQL – Inserção

✓ **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	Messages
 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
```

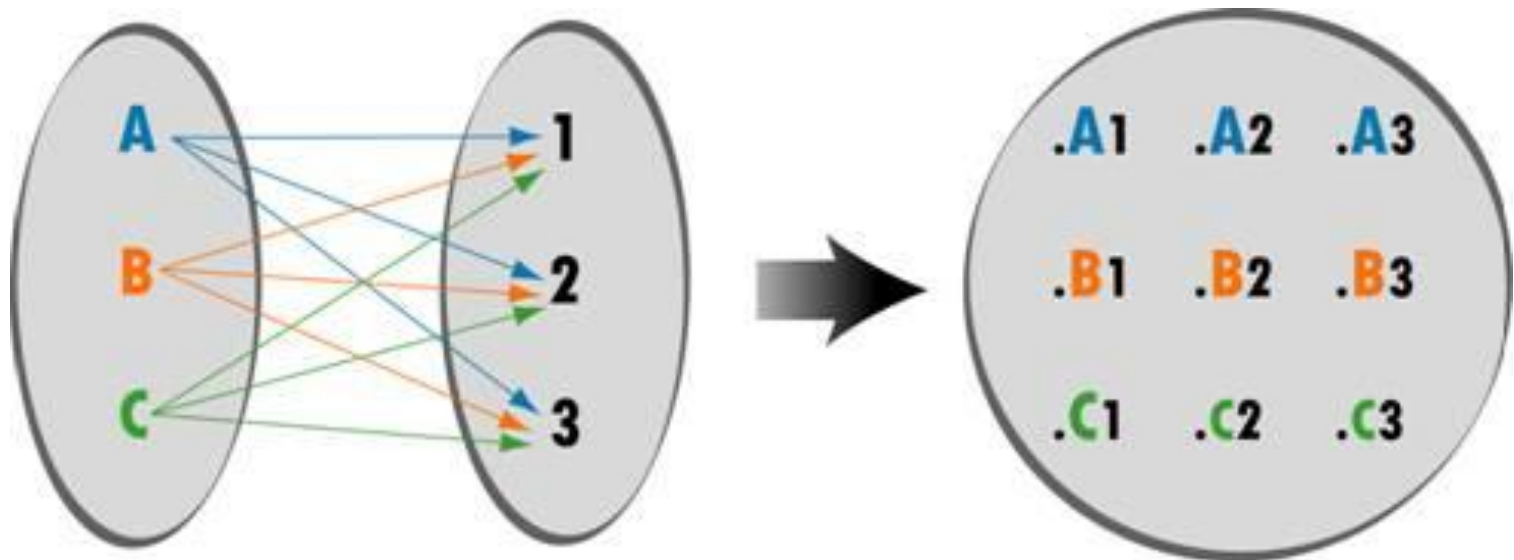
```
1 SELECT adi_carga_horaria  
2 FROM alunos_disciplinas  
3 WHERE adi_carga_horaria BETWEEN 65 AND 150
```

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**

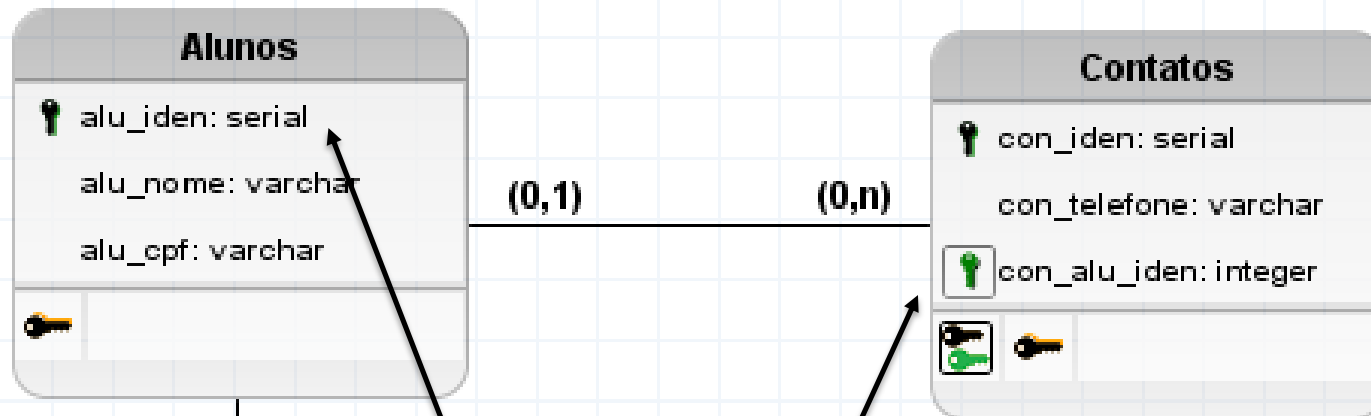


✓ Cláusula FROM

- 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

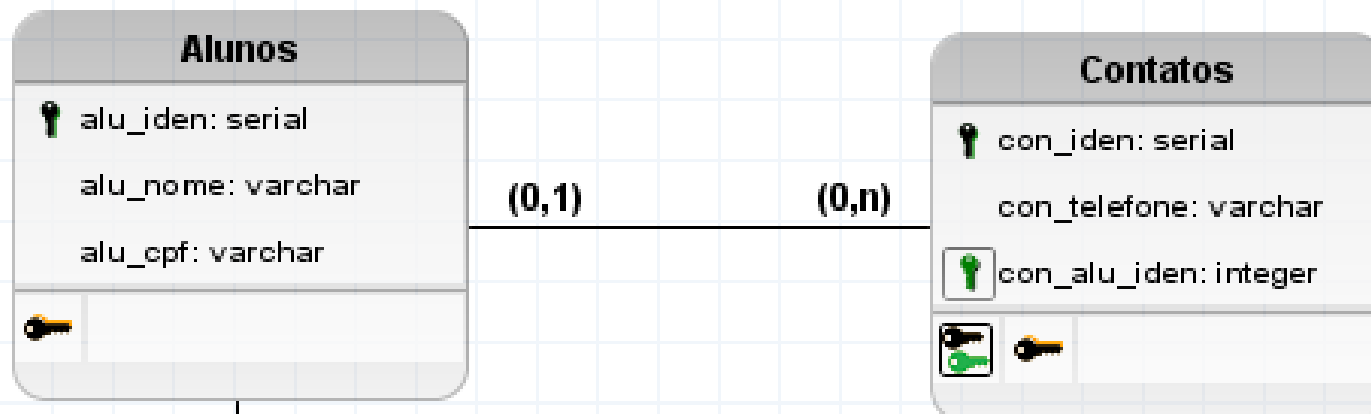


```
1  SELECT alu_nome, contatos.*
2  FROM alunos, contatos
3  WHERE alunos.alu_iden = contatos.con_alu_iden
```

Data Output Explain Messages Notifications

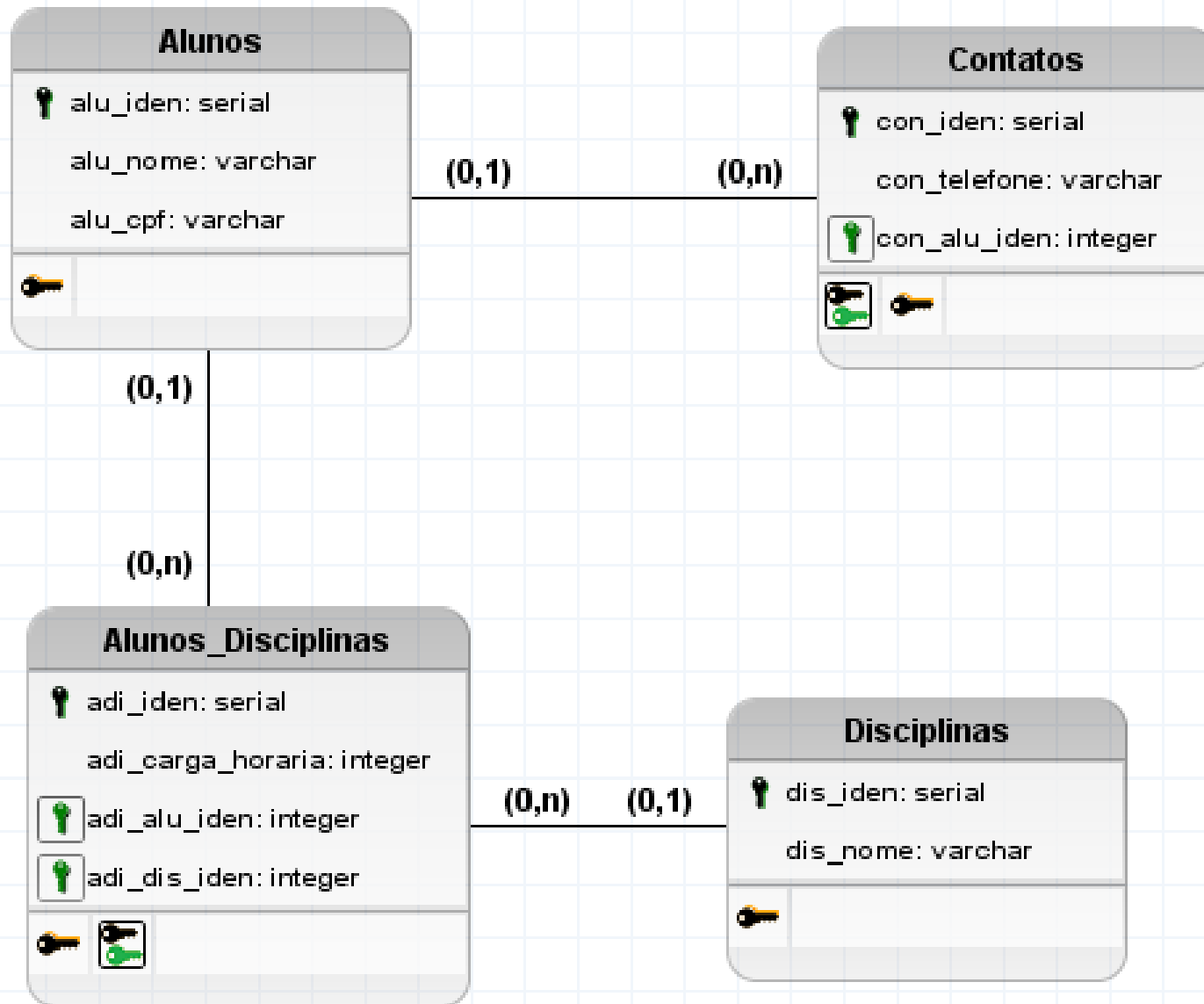
	alu_nome character varying	con_iden integer	con_telefone character varying	con_alu_iden integer
1	MARIA JOAQUINA	7	5555-5555	1
2	MARIA JOAQUINA	8	5444-5555	1
3	RAMBO	9	5333-5555	3

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
```