



101010101  
001010101  
101010100

E-BOOK

# GUIA RÁPIDO DE SQL



# SUMÁRIO



<b>INTRODUÇÃO</b>	03
<b>1. DQL</b>	04
1.1. SELECT	04
1.2. WHERE	05
1.3. AND, OR, NOT	06
1.4. COUNT	07
1.5. SUM	08
1.6. AVG	08
1.7. MIN	09
1.8. MAX	09
1.9. GROUP BY	09
1.10. ORDER BY	10
1.11. INNER JOIN	10
1.12. LEFT JOIN	11
1.13. RIGHT JOIN	12
1.14. HAVING	12
<b>2. DDL</b>	13
2.1. CREATE	13
2.2. ALTER	14
2.3. DROP	15
<b>3. DML</b>	16
3.1. INSERT	16
3.2. UPDATE	17
3.3. DELETE	17

# INTRODUÇÃO

Este é um guia rápido de SQL criado pela equipe do TreinaWeb e serve principalmente para iniciantes na linguagem.

SQL é a sigla em inglês para “Structured Query Language”, que traduzindo significa “Linguagem de Consulta Estruturada”, e é vastamente utilizada no dia a dia dos desenvolvedores. Com a SQL você pode executar comandos para criar, alterar, consultar ou gerenciar bases de dados relacionais de forma otimizada, simples e segura.

A linguagem SQL é dividida em três conjuntos: o DQL (Data Query Language), o DML (Data Manipulation Language) e o DDL (Data Definition Language). Neste e-book você poderá explorar e entender melhor qual a utilidade prática de cada um deles, bem como, conhecer as principais formas de consulta e manipulação que podem ser realizadas dentro de um ambiente de banco de dados.

**Boa leitura e bons estudos!**

# 1. DQL

DQL é a sigla de “Data Query Language”, ou em português, “Linguagem de Consulta de Dados”, o conjunto mais utilizado do SQL. Com ele você pode realizar consultas de vários tipos utilizando o comando SELECT e suas cláusulas. Cláusulas são especificações que você define para indicar os dados que uma consulta deve trazer.

## 1.1 SELECT

O comando SELECT é o principal comando do SQL pois com ele você realiza desde consultas mais básicas quanto as mais avançadas. A sintaxe básica do SELECT é:

```
1 SELECT <coluna> FROM <tabela>
```

Onde:

- **coluna:** lista de colunas que você deseja trazer;
- **tabela:** nome da tabela de onde serão extraídos os dados a serem consultados.

Imaginando um banco de dados de uma escola, primeiramente, vamos consultar todos os alunos que estudam nessa escola, utilizando o seguinte comando:

```
1 SELECT * FROM alunos;
```

Result Grid								
Filter Rows:								
	ra	nome	idade	data_nasc	rg	endereco	telefone	turma
▶	20	Sonia	9	2008-09-07	492345987	Av. Almeida	33012489	4º ano A
	45	José	16	2001-09-02	435643268	Av. Cardoso	33098282	9º ano C
	90	Maria	15	2002-06-02	474930021	Rua Treze	33332390	8º ano C
	100	Marcos	12	2005-01-04	487621235	Rua João	33089090	6º ano A
	121	João	10	2006-10-10	491236542	Rua Dois	33330202	5º ano B

O caractere asterisco (\*) é um coringa que traz todos os campos da tabela.

Se quisermos filtrar os campos e trazer apenas o nome e o telefone de todos os alunos, o comando seria o seguinte:

```
1 SELECT nome, telefone FROM alunos;
```

Result Grid		
Filter Rows:		
	nome	telefone
▶	Sonia	33012489
	José	33098282
	Maria	33332390
	Marcos	33089090
	João	33330202

Para facilitar as consultas SQL, podemos utilizar o pseudônimo ALIAS para dar nome a uma coluna ou tabela. Isso é comumente utilizado para encurtar a escrita da consulta, caso você tenha nomes de colunas ou tabelas muito grandes. Por isso, o nome do

ALIAS é geralmente curto e você pode dar a ele qualquer nome.

Sua sintaxe é:

```
1 SELECT <coluna> FROM <tabela> AS <nome_alias>
```

Onde:

- **coluna:** lista de colunas que você deseja trazer;
- **tabela:** nome da tabela de onde serão extraídos os dados a serem consultados;
- **nome\_alias:** Nome encurtado da tabela ou coluna.

Se quisermos trazer o nome e a idade dos alunos utilizando o ALIAS, podemos realizar da seguinte maneira:

```
1 SELECT al.nome, al.idade FROM alunos AS al
```

Result Grid		
	nome	idade
▶	Clara	9
	Sonia	9
	José	16
	Maria	15
	Marcos	9
	João	10

Neste caso, a palavra “alunos” foi substituída por “al”.

## 1.2. WHERE

O comando WHERE é um comando que especifica uma condição que você pode utilizar em suas consultas, onde o resultado deve reunir apenas os que foram selecionados por ela.

A sintaxe do WHERE é:

```
1 SELECT <coluna> FROM <tabela> WHERE <condicao>
```

Onde:

- **coluna:** lista de colunas que você deseja trazer;
- **tabela:** nome da tabela de onde serão extraídos os dados a serem consultados;
- **condicao:** filtragem dos dados.

Se agora quisermos trazer apenas os alunos que estudam no 5º ano B, podemos realizar a seguinte consulta utilizando a cláusula WHERE.

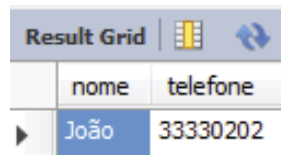
```
1 SELECT al.nome FROM alunos AS al
2 WHERE turma = '5º ano B';
```

Result Grid	
	nome
▶	João

Nesta consulta, nós traremos somente o que foi especificado: os nomes dos alunos que estão

matriculados na turma do 5º ano B. Caso for necessário trazer mais campos além do nome, podemos realizar a consulta da seguinte maneira:

```
1 SELECT al.nome, al.telefone FROM alunos AS al
2 WHERE turma = '5º ano B';
```

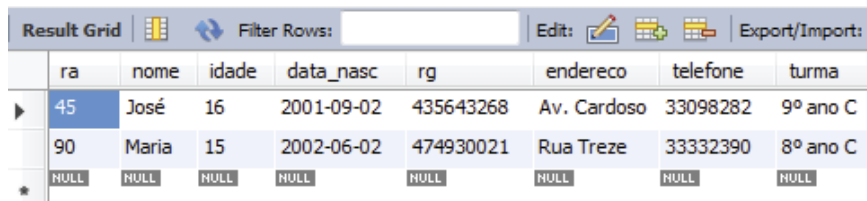


	nome	telefone
▶	João	33330202

Neste caso, a consulta irá trazer também o telefone dos alunos da turma 5º ano B.

Se quisermos realizar uma consulta que traga todos os alunos, porém, somente os que tiverem a idade maior ou igual a 15 anos, também podemos realizar essa consulta utilizando a cláusula WHERE:

```
1 SELECT * FROM alunos AS al
2 WHERE al.idade >= 15;
```



	ra	nome	idade	data_nasc	rg	endereco	telefone	turma
▶	45	José	16	2001-09-02	435643268	Av. Cardoso	33098282	9º ano C
	90	Maria	15	2002-06-02	474930021	Rua Treze	33332390	8º ano C
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

### 1.3. AND, OR E NOT

AND, OR e NOT são operadores lógicos. Eles avaliam a condição presente no seu comando e devolvem um resultado.

- **AND** – Significa “E”, avalia a condição e devolve um valor verdadeiro caso todas sejam corretas;
- **OR** – Significa “OU”, avalia a condição e devolve um valor verdadeiro caso uma das condições sejam corretas;
- **NOT** – É um operador de negação. Devolve o valor contrário do que foi solicitado.

A sintaxe básica desses operadores são:

```
1 SELECT <coluna> FROM <tabela> WHERE <condicao> AND <condicao>
2 SELECT <coluna> FROM <tabela> WHERE <condicao> OR <condicao>
3 SELECT <coluna> FROM <tabela> WHERE NOT <condicao>
```

Onde:

- **coluna:** lista de colunas que você deseja trazer;
- **tabela:** nome da tabela de onde serão extraídos os dados a serem consultados;
- **condicao:** filtragem dos dados.

Voltando aos exemplos, se quisermos trazer todos os alunos que tenham 9 anos e estejam no “4º ano A”, utilizaríamos o operador AND, como podemos ver abaixo:

```
1 SELECT nome FROM alunos AS al
2 WHERE al.idade = 9 AND al.turma = '4º ano A';
```

Result Grid	
	nome
▶	Sonia
	Marcos

Vamos utilizar o mesmo exemplo acima, só que com o operador OR.

```
1 SELECT nome FROM alunos AS al
2 WHERE al.idade = 9 OR al.turma = '4º ano A';
```

Result Grid	
	nome
▶	Clara
	Sonia
	Marcos

Neste caso, a consulta irá trazer todos os alunos que tenham 9 anos ou que estejam matriculados na turma do 4º ano A.

Como já foi descrito, o operador NOT é um operador de negação. Então, se quisermos listar os alunos de todas as turmas exceto a turma “4º ano A”, podemos realizar a consulta da seguinte maneira:

```
1 SELECT nome FROM alunos AS al
2 WHERE NOT al.turma = '4º ano A';
```

Result Grid	
	nome
▶	Clara
	José
	Maria
	João

## 1.4. COUNT

COUNT é uma função de agregação que retorna o número de linhas de uma determinada condição. Essa função ignora valores nulos, ou seja: se você especificar uma coluna que contenha valores nulos, o COUNT irá ignorar as linhas correspondentes durante a contagem.

Sua sintaxe básica é:

```
1 SELECT COUNT (<coluna>) FROM <tabela>
```

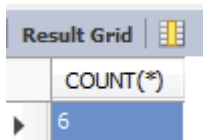
Onde:

- **coluna:** lista de colunas que você deseja trazer. Caso queira trazer todas, você pode utilizar o caractere asterisco;
- **tabela:** nome da tabela de onde serão extraídos os dados a serem consultados.

Primeiramente, vamos verificar quantos registros

existem na tabela alunos.

```
1 SELECT COUNT(*) FROM alunos;
```



Result Grid
COUNT(*)
6

Neste caso, ele contará todos os registros da tabela de alunos, inclusive os que tem campos nulos, já que não foi especificada nenhuma coluna. Neste caso, ele só ignoraria se todas as colunas fossem nulas.

Se quisermos filtrar essa consulta trazendo todos os alunos que contenham telefone, ou seja, que não tenham o campo telefone em branco, podemos realizar da seguinte maneira:

```
1 SELECT COUNT(al.telefone) FROM alunos AS al;
```

Neste caso, todos os alunos que tenham o campo telefone preenchido serão retornados.

## 1.5. SUM

SUM também é uma função de agregação onde é realizada a somatória do valor das colunas que forem especificadas.

Sua sintaxe básica é:

```
1 SELECT SUM(<coluna>) FROM <tabela>
```

Onde:

- **coluna:** lista de colunas que você deseja realizar a somatória;
- **tabela:** nome da tabela de onde serão extraídos os dados.

Se quisermos realizar a somatória de quatro notas do aluno cujo registro (RA) seja número “45”, podemos realizar da seguinte maneira:

```
1 SELECT SUM(nota) FROM alunos WHERE ra = 45;
```

Neste caso, serão procuradas as quatro notas do aluno cujo RA seja “45” e, logo após, será realizada a somatória das mesmas.

## 1.6. AVG

A função AVG extrai a média aritmética de um grupo de linhas determinadas. Neste caso, os valores nulos são sempre ignorados.

Sua sintaxe básica é:

```
1 SELECT AVG(<coluna>) FROM <tabela>
```

Onde:

- **coluna:** lista de colunas que você deseja realizar a média aritmética;
- **tabela:** nome da tabela de onde serão



extraídos os dados.

Voltando ao exemplo acima, após realizar a somatória das notas do aluno cujo RA é 45, podemos realizar também a média aritmética dessas 4 notas com o seguinte comando:

```
1 SELECT AVG(nota) FROM alunos WHERE ra = 45;
```

### 1.7. MIN

MIN retorna o menor valor de um grupo de linhas.

Sua sintaxe básica é:

```
1 SELECT MIN (<coluna>) FROM <tabela>
```

Onde:

- **coluna:** lista de colunas que você deseja realizar a consulta para encontrar o valor mínimo;
- **tabela:** nome da tabela de onde serão extraídos os dados.

Se quisermos agora saber a menor nota que o aluno do nosso exemplo teve, podemos realizar o seguinte comando:

```
1 SELECT MIN(nota) FROM alunos WHERE ra = 45;
```

### 1.8. MAX

MAX retorna o maior valor de um grupo de linhas.

Sua sintaxe básica é:

```
1 SELECT MAX(<coluna>) FROM <tabela>
```

Onde:

- **coluna:** lista de colunas que você deseja realizar a consulta para encontrar o valor máximo;
- **tabela:** nome da tabela de onde serão extraídos os dados.

Assim como verificamos anteriormente a menor nota, vamos verificar agora a maior nota:

```
1 SELECT MAX(nota) FROM alunos WHERE ra = 45;
```

### 1.9. GROUP BY

GROUP BY é utilizado para agrupar dados de linhas ou colunas, baseado nas semelhanças entre elas. A cláusula GROUP BY é geralmente utilizada com as cláusulas SUM, AVG, COUNT, MAX, MIN.

Sua sintaxe é:

```
1 SELECT <colunas> FROM <tabela> GROUP BY <coluna>
```

Onde:

- **colunas:** lista de colunas que você deseja consultar;
- **tabela:** nome da tabela de onde serão extraídos os dados;
- **coluna:** coluna pela qual se deseja agrupar.

Se quisermos trazer a contagem de quantos alunos tem em cada turma, precisamos agrupá-los por turma.

```
1 SELECT COUNT(*), al.turma FROM alunos AS al
2 GROUP BY al.turma;
```

Result Grid		
	COUNT(*)	turma
▶	2	4º ano A
	1	4º ano B
	1	5º ano B
	1	8º ano C
	1	9º ano C

## 1.10. ORDER BY

ORDER BY determina a ordem de apresentação do resultado da consulta, podendo ser em ordem ascendente ou descendente.

Sua sintaxe básica é:

```
1 SELECT <colunas> FROM <tabela>
2 ORDER BY <coluna> <ASC/DESC>
```

Onde:

- **colunas:** lista de colunas que você deseja selecionar;
- **tabela:** nome da tabela de onde serão

extraídos os dados;

- **coluna:** coluna pela qual a ordenação deverá ser realizada;
- **<ASC/DESC>:** ordem de ordenação, sendo ASC para ordenação ascendente e DESC para ordenação descendente.

Se quisermos trazer todos os alunos de forma crescente, podemos realizar a consulta da seguinte maneira:

```
1 SELECT al.nome FROM alunos AS al
2 ORDER BY al.nome ASC;
```

Result Grid	
	nome
▶	Clara
	João
	José
	Marcos
	Maria
	Sonia

Se quisermos trazer todos os nomes só que em ordem decrescente, podemos utilizar o mesmo comando, trocando apenas o ASC pelo DESC.

## 1.11. INNER JOIN

O INNER JOIN é um comando que retornará todos os registros em comum de duas tabelas.

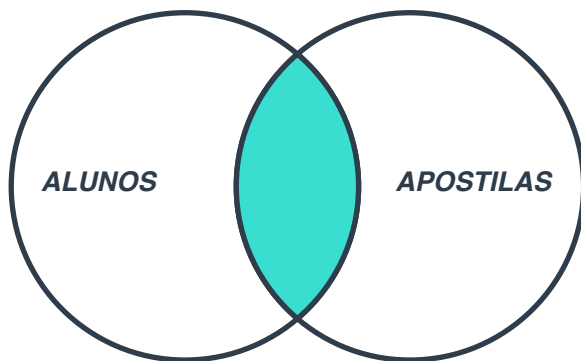
Sua sintaxe básica é:

```
1  SELECT *
2  FROM <tabela1>
3  INNER JOIN <tabela2>
4      ON <tabela1.coluna> = <tabela2.coluna>
```

Onde:

- **tabela1:** nome da tabela de onde serão extraídos os dados. Essa tabela será considerada à esquerda;
- **tabela2:** nome da outra tabela de onde serão extraídos os dados. Essa tabela será considerada à direita;
- **tabela1.coluna e tabela2.coluna:** lista de colunas que serão utilizadas para se realizar a ligação das duas tabelas.

Graficamente, o INNER JOIN ficará da seguinte maneira:



Agora, além de utilizarmos a tabela de alunos, também utilizaremos a tabela de apostilas. Essas apostilas podem ou não estar associadas a um aluno. Por exemplo, se quisermos trazer todos os alunos que possuam ao menos uma apostila, podemos realizar o seguinte comando:

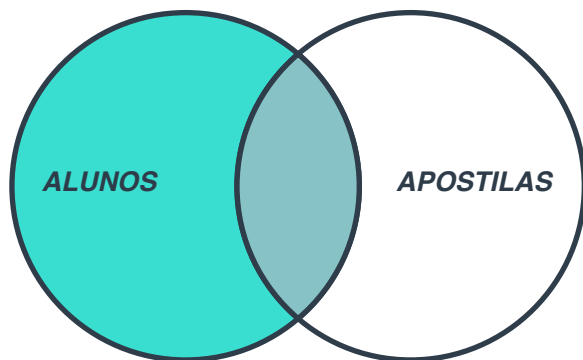
```
1  SELECT nome
2  FROM alunos AS al
3  INNER JOIN apostilas AS ap
4      ON al.ra = ap.ra_aluno;
```

Assim ele trará todos os alunos que contenham uma ou mais apostilas. Os alunos que não tenham nenhuma apostila no momento não serão mostrados no resultado da nossa consulta.

### 1.12. LEFT JOIN

O LEFT JOIN retorna todos os registros da tabela da esquerda e os registros da tabela da direita, desde que existam registros em comum na tabela da esquerda

Graficamente, ele pode ser representado da seguinte maneira:



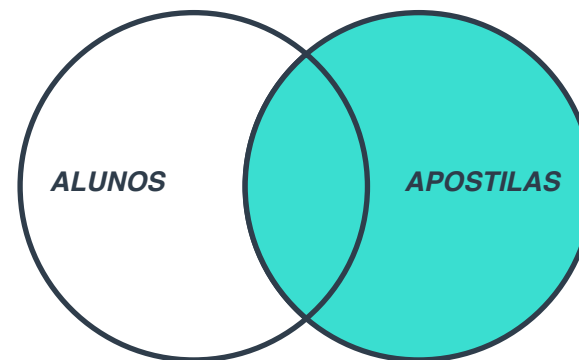
Para compreender melhor o uso do LEFT JOIN, vamos ao seguinte exemplo: suponhamos que precisamos realizar uma consulta que traga todos os alunos e as apostilas que estão com cada um, porém precisamos também trazer os alunos que não estejam com nenhuma apostila. Sendo assim, podemos realizar a consulta da seguinte maneira:

```
1 SELECT nome FROM alunos AS al
2 LEFT JOIN apostilas AS ap
3 ON al.ra = ap.ra_aluno;
```

### 1.13. RIGHT JOIN

O RIGHT JOIN é o contrário do LEFT JOIN. RIGHT JOIN retorna todos os dados da tabela da direita mais os dados em comum com a tabela da esquerda.

Graficamente, ele pode ser representado da seguinte maneira:



Podemos utilizar um exemplo similar ao que usamos no LEFT JOIN: dessa vez, vamos retornar todas as apostilas, independente se elas tiverem vínculo com algum aluno ou não.

```
1 SELECT nome FROM alunos AS al
2 RIGHT JOIN apostilas AS ap
3 ON al.ra = ap.ra_aluno;
```

Apesar de o comando ser parecido com o LEFT JOIN, este comando trará todas as apostilas, independente se houver vínculo com aluno ou não, pois o RIGHT JOIN atribui a prioridade para a tabela da direita.

### 1.14. HAVING

O HAVING determina uma condição de busca. Essa condição somente pode ser utilizada com o comando SELECT, quando houver um grupo ou função de agregação, por isso é geralmente utilizada com uma cláusula GROUP BY.

É importante ressaltar que o HAVING é diferente da cláusula WHERE. O WHERE restringe os resultados obtidos sempre após o uso da cláusula FROM, enquanto o HAVING filtra o resultado do agrupamento.

Sua sintaxe é:

```
1 SELECT <colunas> FROM <tabela>
2 GROUP BY <coluna> HAVING <condicao>
```

Onde:

- **colunas:** lista de colunas que você deseja realizar a consulta;
- **tabela:** nome da tabela de onde serão extraídos os dados;
- **coluna:** coluna pela qual os dados da consulta serão agrupados;
- **condicao:** filtragem dos dados.

Para melhor entendimento, vamos usar o seguinte exemplo: suponhamos que precisamos saber quantas apostilas tem cada aluno. Para isso, primeiro precisamos realizar um INNER JOIN das tabelas de alunos e apostilas para obter as informações em comum. Após isso, será realizada a contagem de apostilas por aluno, porém, queremos neste caso que somente sejam mostrados os alunos que contenham duas ou mais apostilas. O comando ficará da seguinte maneira:

```
1 SELECT COUNT(*), nome FROM alunos AS al
2 INNER JOIN apostilas AS ap
3     ON al.ra = ap.ra_aluno
4 GROUP BY al.ra
5 HAVING COUNT(*)>=2;
```

## 2. DDL

DDL é a sigla para “Data Definition Language” ou “Linguagem de Definição de Dados”. Os comandos DDL são utilizados para definir estruturas de dados no SQL, fazendo o uso dos comandos CREATE, ALTER e o DROP.

### 2.1. CREATE

O comando CREATE é utilizado para criar objetos, como um novo banco de dados ou mesmo tabelas.

Suas sintaxes são:

- Para criar um banco de dados:

```
1 CREATE DATABASE <nome_banco>
```

- Para criar tabela no banco de dados:

```
1 CREATE TABLE <nome_tabela>
2 <definicao_colunas> <parametros>
```



Onde:

- **nome\_banco:** nome que você deseja dar ao novo banco de dados;
- **nome\_tabela:** nome que você deseja dar à nova tabela;
- **definicao\_colunas:** define as colunas que você deseja inserir na nova tabela;
- **parametros:** você pode definir os parâmetros de cada coluna, como o seu tipo, tamanho, se será uma coluna obrigatória, dentre outros.

Para criar nosso banco de dados escolar, podemos criá-lo da seguinte maneira:

## 1 CREATE DATABASE escola;

Para criar uma tabela para armazenar os dados dos nossos alunos, podemos realizar com o seguinte comando:

```
1 CREATE TABLE alunos
2 (
3     ra            INTEGER      PRIMARY KEY,
4     nome          VARCHAR(20)  NOT NULL,
5     idade         INTEGER      NOT NULL,
6     data_nasc     DATE          NOT NULL,
7     rg            INTEGER      NOT NULL,
8     endereço      VARCHAR(100) NOT NULL,
9     telefone      INTEGER      NOT NULL,
10    turma         VARCHAR      NOT NULL
11 );
```

Nesse caso, a tabela de alunos irá conter estes campos com estes parâmetros determinados. Perceba que um dos parâmetros do campo RA é a definição PRIMARY KEY.

O PRIMARY KEY é a definição de uma chave primária que garante um valor único dentro da tabela. Neste caso, cada aluno terá um valor de RA único, não podendo ser repetido, nem nulo. A utilização da chave primária também pode ser utilizada como um índice de referência para a criação de relacionamentos com outras tabelas.

Após executado o comando, teremos como resultado a seguinte tabela:

Result Grid								
Filter Rows:								
	ra	nome	idade	data_nasc	rg	endereço	telefone	turma
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

## 2.2. ALTER

O comando ALTER altera propriedades de um objeto dentro do banco de dados. Com ele você pode alterar propriedades adicionando, alterando ou até mesmo excluindo um objeto.

Sua sintaxe é:



```
1 ALTER <tipo_objeto>
2 <nome_objeto> <parametros>
```

Onde:

- **tipo\_objeto:** É o tipo do objeto que você deseja trabalhar, podendo ser um banco (database) ou uma tabela (table);
- **nome\_objeto:** Nome do banco ou tabela;
- **parametros:** comandos que você deseja realizar, podendo alterar, adicionar ou excluir.

Se quisermos, por exemplo, inserir a coluna “nome\_responsavel” na tabela de alunos, podemos realizar esta alteração da seguinte forma:

```
1 ALTER TABLE alunos ADD nome_responsavel
2 VARCHAR(15) NOT NULL;
```

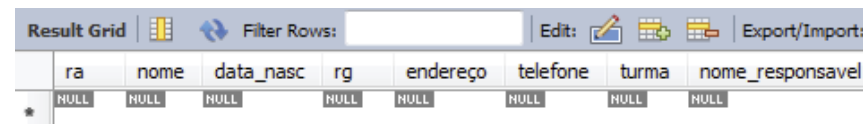
Depois de executado o comando a nova coluna irá aparecer na tabela.



ra	nome	idade	data_nasc	rg	endereço	telefone	turma	nome_responsavel
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Também podemos excluir uma coluna em uma tabela caso necessário. Vamos então excluir a coluna “idade”. O comando ficará da seguinte maneira:

```
1 ALTER TABLE alunos DROP COLUMN idade;
```



ra	nome	data_nasc	rg	endereço	telefone	turma	nome_responsavel
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

## 2.3. DROP

DROP é o comando utilizado para excluir dados, podendo ser uma tabela ou até mesmo o banco de dados.

Sua sintaxe é:

```
1 DROP <tipo_objeto> <nome_objeto>
```

Onde:

- **tipo\_objeto:** Pode ser um banco (database) ou uma tabela (table);
- **nome\_objeto:** Nome do banco ou tabela.

Se quisermos excluir de vez a tabela de alunos, utilizaremos o seguinte comando:

```
1 DROP TABLE alunos;
```

Assim, nossa tabela de alunos será removida do banco de dados.

## 3. DML

DML é a sigla de “Data Manipulation Language” ou “Linguagem de Manipulação de Dados”. São instruções utilizadas para inserir, atualizar ou excluir dados de um banco de dados SQL.

### 3.1. INSERT

O comando INSERT é utilizado para inserir dados em uma tabela, adicionando uma ou mais linhas (ou tuplas como são chamadas as linhas no banco de dados).

Sua sintaxe é:

```
1  INSERT INTO <tabela> <colunas> <valores>
```

Onde:

- **tabela:** nome da tabela na qual deseja inserir os dados;
- **coluna:** colunas que receberão os novos valores;
- **valores:** dados que serão inseridos nas colunas especificadas da tabela.

Por exemplo: vamos inserir um novo aluno na nossa tabela de alunos:

```
1  INSERT INTO alunos
2      (ra, nome, idade, data_nasc, rg,
3      endereco, telefone, turma)
4  VALUES
5      (121, 'João', 10, '10/10/2006',
6      491236542, 'Rua Um', 33330101,
7      '5º ano B');
```

Neste caso, todos os valores das colunas das tabelas foram especificados. Sendo assim, podemos utilizar uma abreviação, onde não são especificadas as colunas. Por isso deve-se sempre levar em conta a ordem das colunas a serem preenchidas.

Vamos então adicionar mais um aluno, só que utilizando essa forma mais abreviada:

```
1  INSERT INTO alunos
2  VALUES
3      (90, 'Maria', 15, '21/06/2002',
4      474930021, 'Rua Treze', 33332390,
5      '8º ano C');
```



ra	nome	idade	data_nasc	rg	endereco	telefone	turma
90	Maria	15	2002-06-02	474930021	Rua Treze	33332390	8º ano C
121	João	10	2006-10-10	491236542	Rua Um	33330101	5º ano B
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Perceba que, no comando, foi seguida a mesma ordem de colunas na tabela.

### 3.2. UPDATE

O comando UPDATE é utilizado para atualizar registros no nosso banco de dados.

Sua sintaxe é:

```
1 UPDATE <tabela> SET coluna = "novo_valor"
2 WHERE <condicao>
```

Onde:

- **tabela:** nome da tabela onde deseja alterar os dados;
- **coluna:** coluna que receberá o novo valor;
- **novo\_valor:** novo valor a ser atribuído para a coluna especificada para as linhas que corresponderem à filtragem em WHERE;
- **condicao:** filtragem dos dados.

Suponhamos que o aluno cujo RA é de número 121 tenha trocado seu número de telefone. Sendo assim, precisamos atualizá-lo:

```
1 UPDATE alunos SET telefone = 33330202
2 WHERE ra = 121;
```

ra	nome	idade	data_nasc	rg	endereco	telefone	turma
90	Maria	15	2002-06-02	474930021	Rua Treze	33332390	8º ano C
121	João	10	2006-10-10	491236542	Rua Um	33330101	5º ano B
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Com o UPDATE, também podemos atualizar mais de um campo na mesma linha. Se além do telefone, o aluno também tiver mudado de endereço, podemos realizar o comando da seguinte forma:

```
1 UPDATE alunos SET endereco = 'Rua
2 Dois', telefone = 33330202
3 WHERE ra = 121;
```

ra	nome	idade	data_nasc	rg	endereco	telefone	turma
90	Maria	15	2002-06-02	474930021	Rua Treze	33332390	8º ano C
121	João	10	2006-10-10	491236542	Rua Dois	33330202	5º ano B
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Neste caso, o comando acima irá atualizar os dois campos a partir de um único comando.

### 3.3. DELETE

O comando DELETE é utilizado para remover dados.

## 1 DELETE FROM <tabela> WHERE <condicao>

Onde:

- **tabela:** nome da tabela que contém as linhas a serem removidas;
- **condicao:** filtragem dos dados, limitando as linhas a serem removidas.

Por exemplo: se quisermos excluir o aluno cujo RA é 90, podemos realizar a operação da seguinte maneira:

## 1 DELETE FROM alunos WHERE ra = 90;

Result Grid								
Filter Rows:			Edit:			Export/Import:		
	ra	nome	idade	data_nasc	rg	endereco	telefone	turma
▶	121	João	10	2006-10-10	491236542	Rua Dois	33330202	5º ano B
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Neste caso, o aluno de RA igual a 90 foi excluído da nossa base de dados.

# Acelere sua carreira na melhor escola para desenvolvedores Full Stack e DevOps do Brasil!

Há mais de 12 anos formando desenvolvedores de ponta! São mais de 4.000 horas de conteúdo, com formações completas e com foco no mercado de trabalho.

Conheça nossos cursos.

 **TREINAWEB**

