Sumário

[ SELECT 2](#_Toc82471465)

[ INSERT INTO 7](#_Toc82471466)

[ UPDATE 8](#_Toc82471467)

[ DELETE 9](#_Toc82471468)

[SQL TOP, LIMITE, FETCH FIRST or ROWNUM Clause: 9](#_Toc82471469)

[MIN () e MAX (): 10](#_Toc82471470)

[COUNT (), AVG (), SUM (). 10](#_Toc82471471)

[ LIKE 11](#_Toc82471472)

[IN 13](#_Toc82471473)

[ BETWEEN 14](#_Toc82471474)

[ALIASES 17](#_Toc82471475)

[JOIN 18](#_Toc82471476)

[GROUP BY 27](#_Toc82471477)

[WHERE 29](#_Toc82471478)

[DATABASE 30](#_Toc82471479)

[CREATE DB 30](#_Toc82471480)

[DROPDATABASE 30](#_Toc82471481)

[BACKUP DATABASE 30](#_Toc82471482)

[CREATE TABLE 30](#_Toc82471483)

[DROP TABLE 31](#_Toc82471484)

[ALTER TABLE 31](#_Toc82471485)

[CONSTRAINSTS 31](#_Toc82471486)

[UNIQUE 32](#_Toc82471487)

[PRIMARY KEY 32](#_Toc82471488)

[FOREIGN KEY 33](#_Toc82471489)

[CHECK 34](#_Toc82471490)

[DEFAULT 34](#_Toc82471491)

[INDEX 35](#_Toc82471492)

[AUTO INCREMENT 35](#_Toc82471493)

[DATES 35](#_Toc82471494)

[VIEWS 36](#_Toc82471495)

[DATA TYPES 37](#_Toc82471496)

* **INTRODUÇÃO/COMANDOS:**

**SELECT** \* **FROM** Customers;

O SELECT seleciona a tabela Customers do DataBase, o “\*” quer dizer que você pegara TODAS as colunas da Tabela, e o FROM referência a tabela de onde você pegara os dados.

Os comandos principais do SQL são: **SELECT**, **UPDATE**, **INSERT** e **DELETE**.

* SELECT:

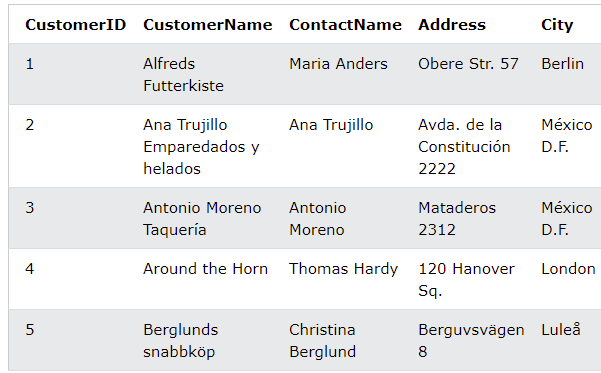
O SELECT do SQL seleciona colunas das tabelas que você deseja ver.

**SELECT** \* Column 1, Column 2, ...

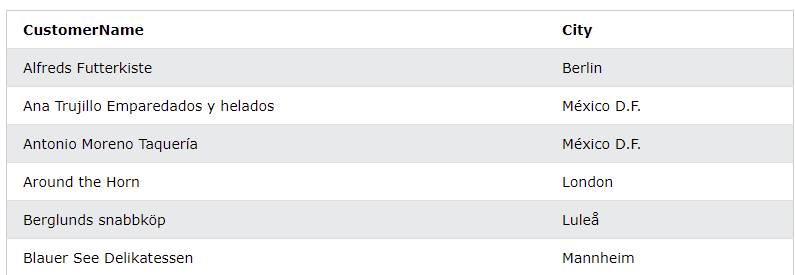
**FROM** table\_name

Exemplo:

**SELECT** \* **FROM** Customers:



**SELECT** CustomerName, City **FROM** Customers;



Outro tipo de SELECT é o **SELECT DISTINCT** que é usado para retornar apenas **valores diferentes.**

1. **SELECT DISTINCT:**

**SELECT DISTINCT** column1, column 2...

**FROM** table\_name

1. **WHERE:**

O **WHERE** pode ser uma clausura usada para realizar **FILTROS**. Ele filtra apenas os registros que atendem àquela condição específica. Pode ser usado no **UPDATE** e **DELETE** também

**SELECT** column 1, column 2...

**FROM** table\_name

**WHERE** condition;

Exemplo:

**SELECT** \* **FROM** Customers

**WHERE** Country = ”México”;



Ele pegou todos os dados (\*) da Tabela Customers e filtrou só os dados do país México.

1. **AND, OR e NOT:**

**AND**, **OR** e **NOT** podem ser combinados com o **WHERE** para filtrar ainda mais os dados desejados.

**SELECT** column 1, column 2...

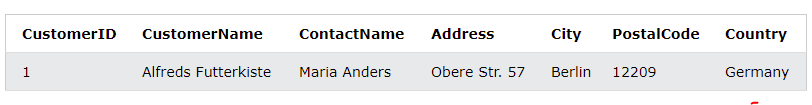
**FROM** table\_name

**WHERE** codition1 **AND** condition 2...;

Exemplo AND:

**SELECT** \* **FROM** Customers

**WHERE** Country=”Germany” **AND** City = “Berlin”



Pegou no DB só os dados que tem o país Alemanha e a cidade Berlin.

**SELECT** column 1, column 2...

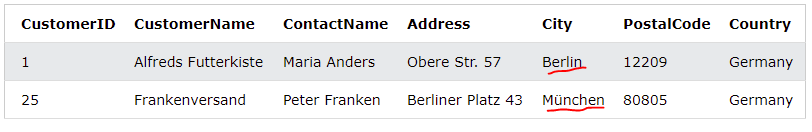
**FROM** table\_name

**WHERE** codition1 **OR** condition 2...;

Exemplo OR:

**SELECT** \* **FROM** Customers

**WHERE** City = “Berlin” **OR** City = “Munchen”



Selecionou só os dados da cidade de Berlin e Munchen

**SELECT** column 1, column 2...

**FROM** table\_name

**WHERE** **NOT** codition;

EXEMPLO NOT:

**SELECT** \* **FROM** Customers

**WHERE** **NOT** Country=”Germany”

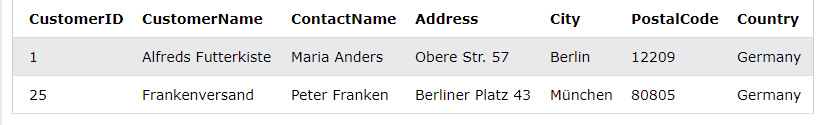


Sem os países da Alemanha.

Exemplo AND, OR and NOT:

**SELECT** \* **FROM** Customers

**WHERE** Country = “Germany” **AND** (City=”Berlin” **OR** City=”Munchen”)



**SELECT**\***FROM** Customers

**WHERE** **NOT** Country=”Germany” **AND** **NOT** Country=”USA”



1. **ORDER BY:**

Para ordenar os resultados em Ascendentes ou descendentes. Usa a palavra DESC para descendente, e ASC para ascendente.

**SELECT** column1, column2,..

**FROM** table\_name

**ORDER** **BY** column1, column2,... **ASC**|**DESC**;

Exemplo:

**SELECT** \* **FROM** Customers

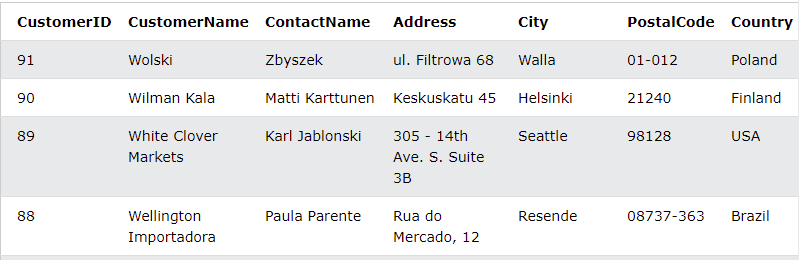
**ORDER** **BY** Country;



Ordenou por país em ordem Alfabética.

**SELECT** \* **FROM** Customers

**ORDER** **BY** Customer ID **DESC**



Ordenou de forma descendente os Customers ID.

* INSERT INTO**:**

É usado para adicionar novos dados na Tabela. Há dois jeitos de inserir dados

**INSERT** **INTO** table\_name (column1, column2,..)

**VALUES** (value1, value2,..);

Os valores são inseridos na ordem que das colunas.

Exemplo:

**INSERT** **INTO** Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

**VALUES** (“Cardinal”, “Tom B. Erichsen”, “Skagen 21”, Stavanger”, “4006”, “Norway”)



OU

**INSERT** **INTO** table\_name

**VALUES** (value1, value2, value3,...);

Nesse caso os valores são adicionados em todas as colunas da Tabela

1. **NULL VALUES:**

Null Values tem valores vazios e é impossível inserir ou atualizar com novos valores, sem adicionar um valor para o mesmo. Null Value não é zero. Os operadores **IS** **NULL** e **IS** **NOT** **NULL** são úteis para verificar os valores null

**SELECT** column\_names

**FROM** table\_name

**WHERE** column\_name **IS** **NULL**;

**SELECT** column\_names

**FROM** table\_name

**WHERE** column\_name **IS** **NOT** **NULL**;

* UPDATE**:**

Útil para atualizar os dados da tabela

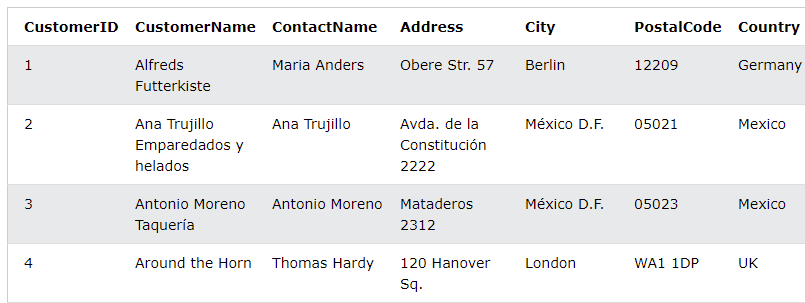
**UPDATE** table\_name

**SET** column1 = value1, column2 = value2,...

**WHERE** condition;

\***OBS**: o WHERE serve para direcionar em qual lugar da tabela os valores serão atualizados, se omiti-lo vai atualizar a tabela **INTEIRA**.

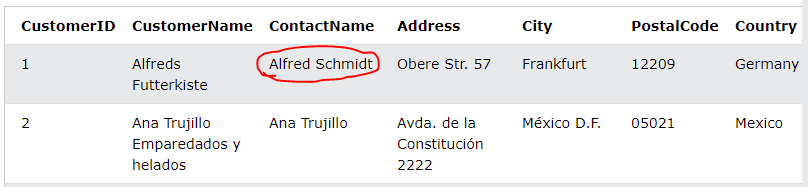
ANTES



**UPDATE** Customers

**SET** ContactName = “Alfred Schmidt”, City = “Frankfurt”

**WHERE** CustomerID=1;

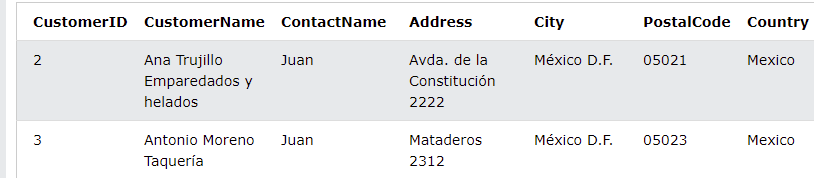


Para atualizar vários dados simultâneos, utiliza-se o **WHERE** também.

**UPDATE** Customers

**SET** ContactName = “Juan”

**WHERE** Country = “México”;



Todo país México terá o Juan como contactname.

* DELETE:

Útil para deletar dados da tabela.

**DELETE** **FROM** table\_name

**WHERE** condition;

\***OBS**: O **WHERE** é utilizado para indicar quais valores deletar. Se não for mencionado, ele deletará **todos** os dados.

**DELETE** **FROM** table\_name irá deletar **todos** os dados

# **SQL TOP, LIMITE, FETCH FIRST or ROWNUM Clause**:

SELECT TOP é usada para especificar o número de dados a ser retornado. É bem utilizado para uma **base de dados grande** com **várias tabelas.**

**MySQL**:

**SELECT** column\_name (s)

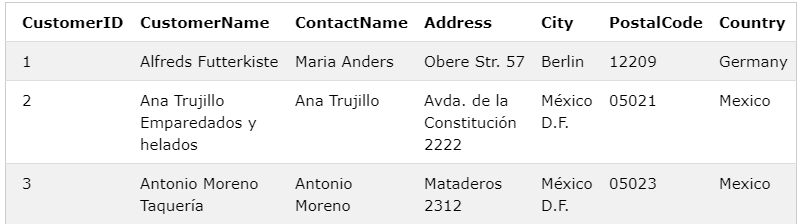
**FROM** table\_name

**WHERE** condition

**LIMIT** number;

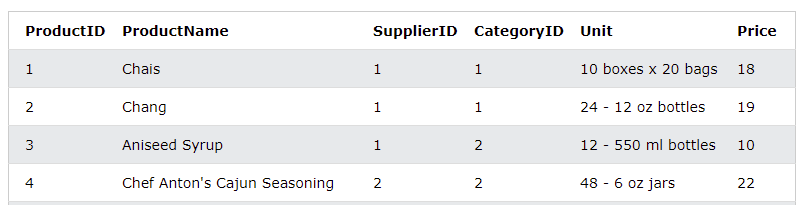
Exemplo:

**SELECT** \* **FROM** Customers **LIMIT** 3;



Exibe as 3 primeiras **linhas** da Tabela.

# MIN () e MAX ():



**SELECT** **MIN** (column\_name)

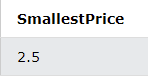
**FROM** table\_name

**WHERE** condition;

Exemplo:

**SELECT** **MIN** (price) **AS** SmallestPrice

**FROM** products;



Retorna o valor com o nome de **Smallest Price**

**SELECT** **MAX** (column\_name)

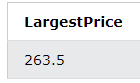
**FROM** table\_name

**WHERE** condition;

Exemplo:

**SELECT** **MAX** (Price) **AS** LargestPrice

**FROM** Products;



# COUNT (), AVG (), SUM ().

**SELECT** **COUNT** (column\_name)

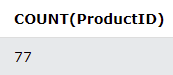
**FROM** table\_name

**WHERE** condition;

Exemplo:

**SELECT** **COUNT** (ProductID)

**FROM** Products;



**SELECT** **AVG** (column\_name)

**FROM** (table\_name)

**WHERE**;

**SELECT** **AVG** (Price)

**FROM** Products;



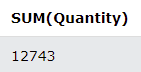
**SELECT** **SUM** (column\_name)

**FROM** table\_name

**WHERE** condition;

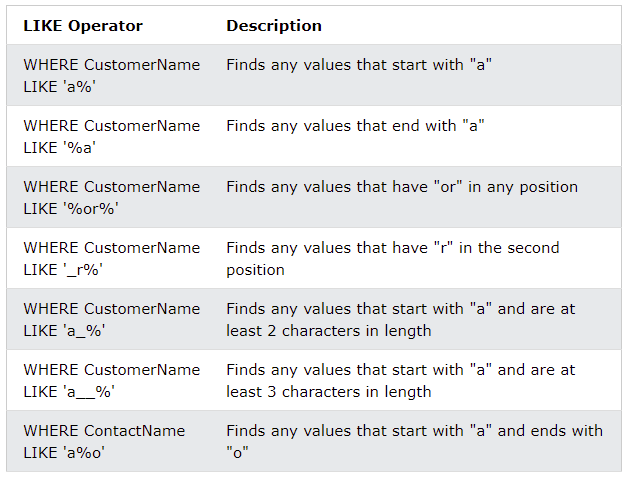
**SELECT** **SUM** (Quantity)

**FROM** OrderDetails;



* LIKE:

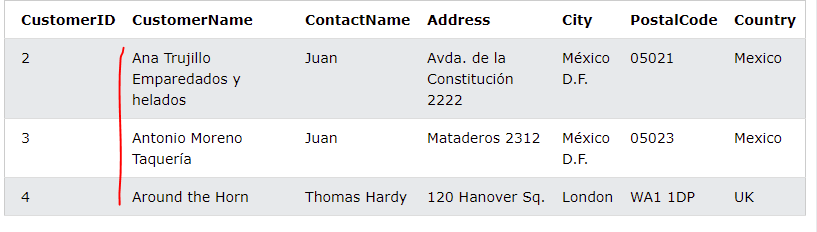
O operador ‘%” representa o zero, um, ou vários caracteres, e o “\_” representa um caractere.



**Exemplo1**:

**SELECT** \* **FROM** Customers

**WHERE** CustomerName **LIKE** “a%’;

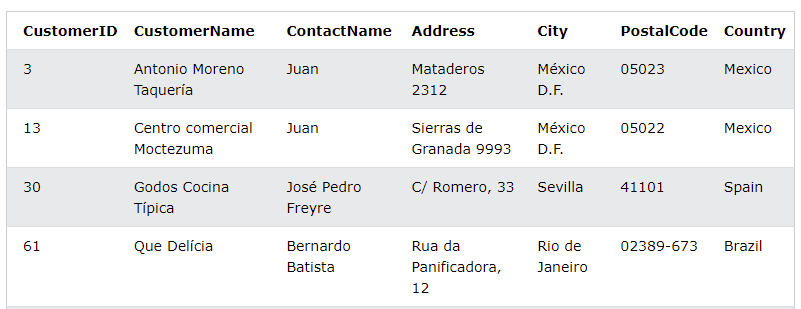


Selecionou valores que começam com “A” para os customers name.

**Exemplo** 2:

**SELECT** \* **FROM** Customers

**WHERE** CustomerName **LIKE** '%a';



Termina tudo em a os CustomerName.

**Exemplo** **3**:

**SELECT** \* **FROM** Customers

**WHERE** CustomerName **LIKE** '%or%'



Acha qualquer “or” no CustomerName.

**Só seguir a Tabela e procurar da forma que for necessário para sua escolha.**

# IN

O operador In permite você especificar em múltiplas variáveis, usando o WHERE. É como se fosse um operador para várias condições (OR).

**Syntax In:**

**SELECT** column\_name(s)

**FROM** table\_name

**WHERE** column\_name **IN** (value1, value2, ...);

**Syntax Or:**

**SELECT** column\_name(s)

**FROM** table\_name

**WHERE** column\_name **IN** (SELECT STATEMENT);

**Exemplo 1:**

**SELECT** \* **FROM** Customers

**WHERE** Country **IN** ('Germany', 'France', 'UK');



Selecionou só os países UK, Germany e France. É como se fosse um **FILTRO** muito bom.

**Exemplo 2:**

**SELECT** \* **FROM** Customers

**WHERE** Country **NOT** **IN** ('Germany', 'France', 'UK');

Seleciona todos os países exceto esses 3.

**Exemplo 3:**

**SELECT** \* **FROM** Customers

**WHERE** Country **IN** (**SELECT** Country **FROM** Suppliers);

Vai selecionar os customers que são do mesmo país dos suppliers.

* BETWEEN:

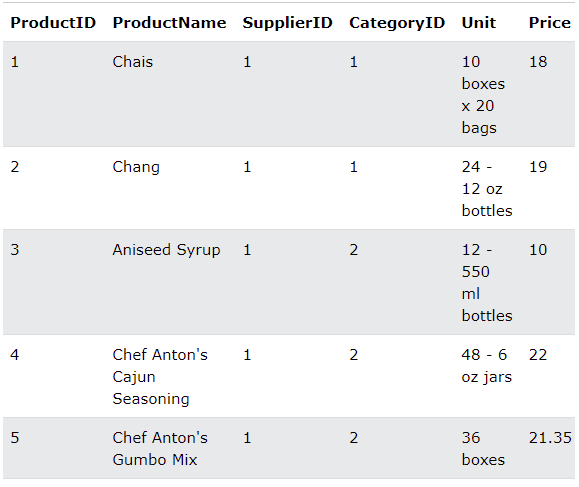
O operador BETWEEN seleciona os valores que dão um range. Esses valores podem ser números, textos ou datas. Sendo uma ferramenta poderosa para filtrar dados. É um operador inclusivo, ou seja, valores iniciais e finais são incluídos.

**SELECT** column\_name(s)

**FROM** table\_name

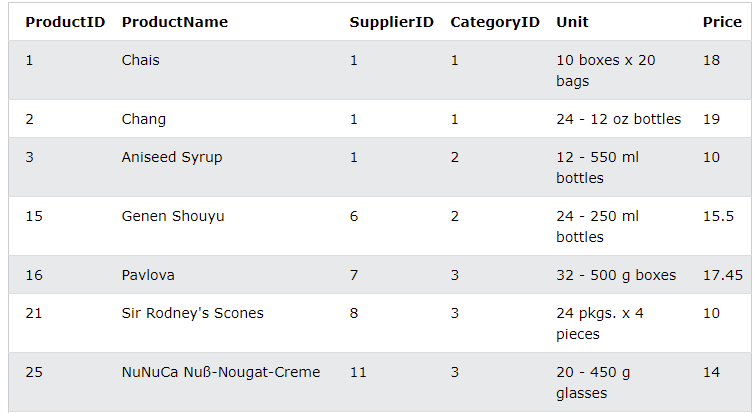
**WHERE** column\_name **BETWEEN** value1 **AND** value2;

**Tabela Base**



**Exemplo 1:**

**SELECT** \* **FROM** Products  
**WHERE** Price **BETWEEN** 10 **AND** 20;

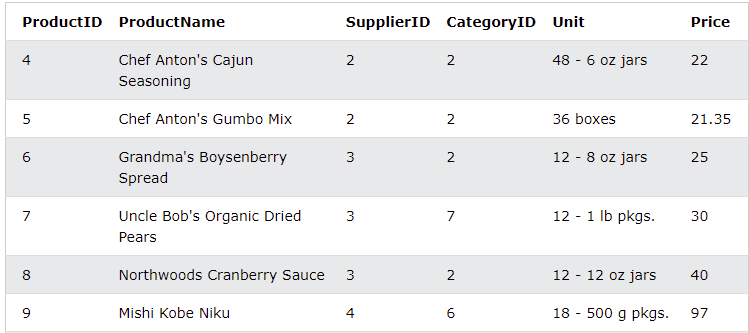


Filtrou apenas os valores entre 10 e 20 da coluna Price.

**Exemplo 2:**

Pode-se usar o **NOT BETWEEN** para não incluir na range.

**SELECT** \* **FROM** Products  
**WHERE** Price **NOT** **BETWEEN** 10 **AND** 20;

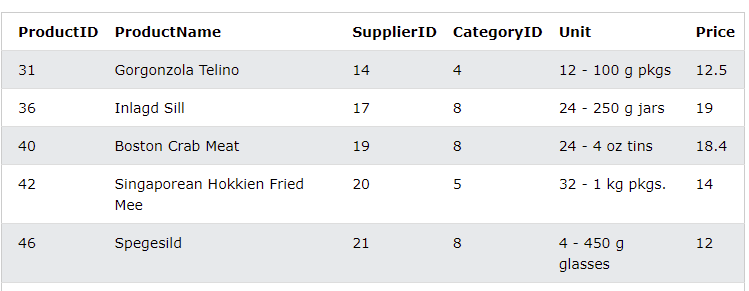


Incluiu apenas valores que estão fora do range de 10 a 20, ou seja, maiores ou menores que esse intervalo.

**Exemplo 3:**

Usa-se o **BETWEEN IN**  para realizar um filtro mais poderoso.

**SELECT** \* **FROM** Products  
**WHERE** Price **BETWEEN** 10 **AND** 20  
**AND** CategoryID **NOT** **IN** (1,2,3);



Além de filtrar os valores da coluna Price que estejam no intervalo de 10 à 20, retirar também do filtro os dados que possuem valor de 1 à 3 na coluna Category ID. A linha 1 por exemplo tem um preço de 18 e poderia entrar no filtro, porém o Category ID tem o valor 1, e sai do filtro.

**Exemplo 4:**

**SELECT** \* **FROM** Products  
**WHERE** ProductName **BETWEEN** 'Carnarvon Tigers' **AND** 'Mozzarella di Giovanni'  
**ORDER** **BY** ProductName;

Seleciona da tabela Produtos, os ProductName’s de Carnavon Tigers até Mozzarela di Giovanni e ordena por ordem alfabética.

# ALIASES

É usado para dar à tabela, ou coluna numa tabela, um nome temporário.

**Alias Column Syntax**

**SELECT** column\_name **AS** alias\_name  
**FROM** table\_name;

**Alias Table Syntax**

SELECT column\_name(s)  
FROM table\_name AS alias\_name;

**Exemplo 1:**

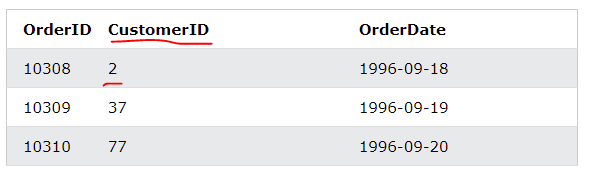
SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

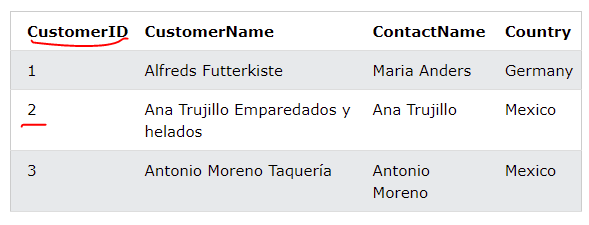


Seleciona as colunas e ainda coloca os nomes temporários.

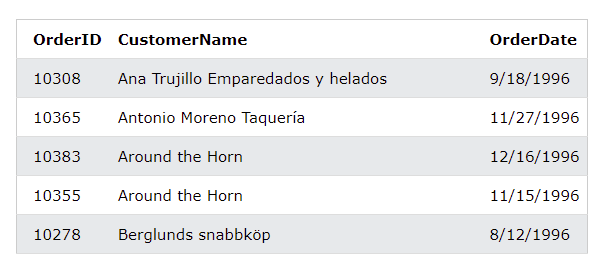
# JOIN

Join é usada para combinar linhas de 2 ou mais tabelas, baseada na coluna entre elas. Essa tabela é a Orders e a outra é a Customers.



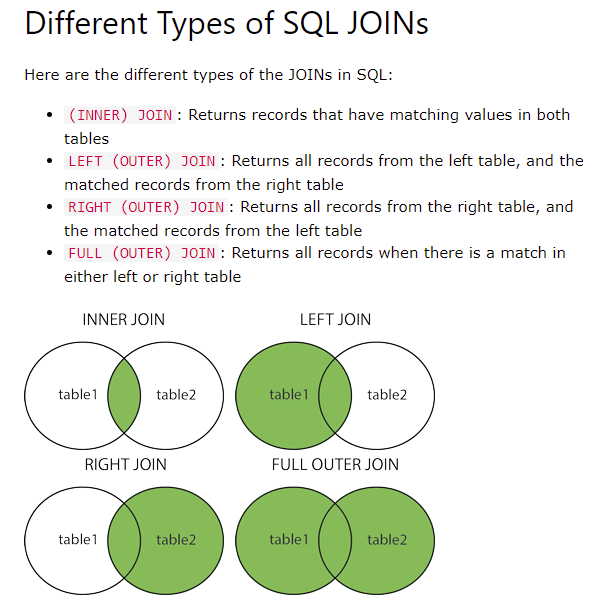


SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;



O resultado foi que criou-se uma tabela com o Order ID, e o OrderDate da tabela Orders, e o nome na coluna CustomerName da tabela Customers. Ou seja, antes uma tabela Orders que tinha apenas o número ID, agora relacionou-se com a tabela Customers para ter o nome.

Na primeira linha do código temos a escolha das colunas para a tabela nova (Order ID, Customer Name e Order Date), na segunda linha de onde pegaremos essas informações para fazer a tabela nova, que é o Orders, e na terceira linha dar o Inner Join, que é o match de informações similares entre as 2 tabelas.



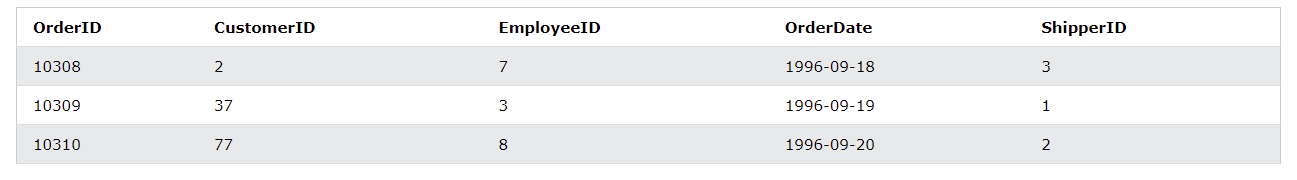
Como usou o INNER JOIN, pegou os valores em comuns das 2 tabelas. Ambas as tabelas possuíam o mesmo Customer ID, assim deu para pegar o que elas possuíam em comum e criar uma nova, porém como resultado de saída colocou o nome do Customer.

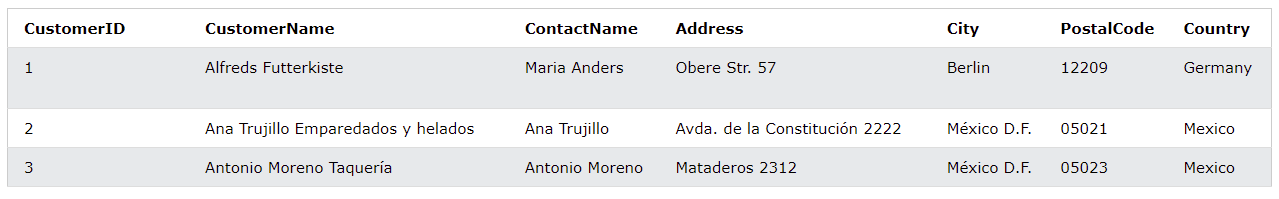
**Exemplo INER JOIN:**

Como mencionado antes, só pega os valores em comuns nas duas tabelas.

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;

Seleciona a coluna desejada a dar o join, na segunda linha menciona a tabela 1, de onde será retirado, e de um inner join na tabela 2, onde tabela1.nome da coluna desejada = tabela 2.nome da coluna desejada.



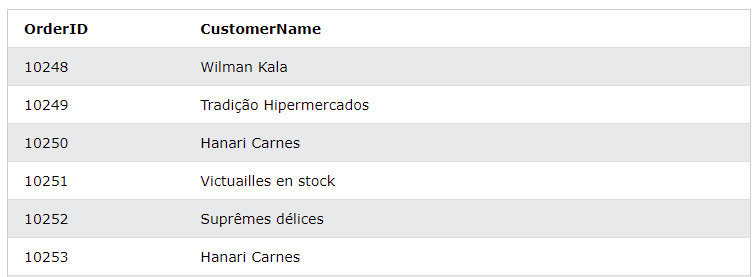


Em ambas as tabelas temos os CustomerID em comum, então vamos relacionar o número do Customer ID da tabela 1 com o nome por exemplo, ou endereço.

SELECT Orders.OrderID, Customers.CustomerName  
FROM Orders  
INNER JOIN Customers

ON Orders.CustomerID = Customers.CustomerID;

Selecionou da tabela Orders a coluna Order ID, e da tabela Customers a coluna CustomerName. Partindo-se da tabela Orders, e dando um Inner Join na tabela Customers, onde da tabela Orders o Customer ID vai ter seus valores ligados pelo Customer ID da Tabela Customers.



O mesmo vale para 3 tabelas.

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName  
FROM ((Orders  
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)  
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);



O objetivo é retornar uma tabela com valores que as **DUAS TABELAS DÃO MATCH**. Olhar o esquema e o desenho.

**Exemplo LEFT JOIN:**

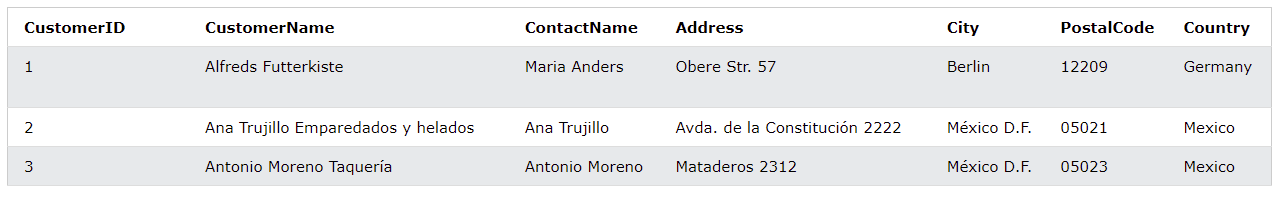
Retorna valores da tabela à esquerda, e os valores da tabela de direita que dão match.

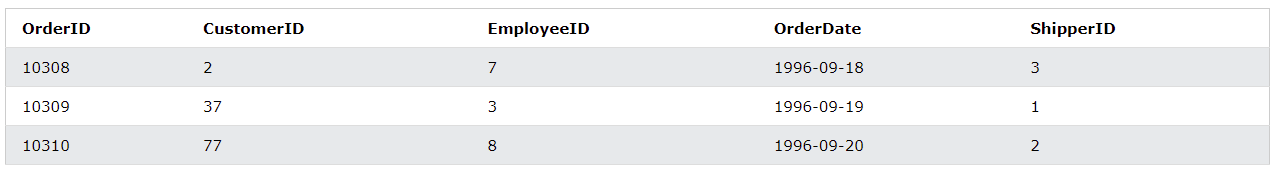
**SELECT** column\_name(s)

**FROM** table1

**LEFT JOIN** table2

**ON** table1.column\_name = table2.column\_name;





Temos a tabela customer e a tabela orders.

**SELECT** Customers.CustomerName, Orders.OrderID

**FROM** Customers

**LEFT** **JOIN** Orders

**ON** Customers.CustomerID = Orders.CustomerID

**ORDER** **BY** Customers.CustomerName;

Primeira linha selecionou-se a coluna CustomerName da tabela Customer, e a coluna Order ID da tabela Orders. Segunda linha indica que vem da tabela Customers, e a terceira irá realizar o Left join na tabela Orders. A quarta linha indica onde será realizado, ou seja, o customer ID da tabela customers vai receber os valores do Customer ID da tabela orders.



**OBS**: Se digitar INNER JOIN ao invés de LEFT a tabela resultara nessas 2 mesmas colunas, por conta do que você escolheu na primeira linha, porém os dados serão diferentes, pois inner join pega os valores em comum entre as 2, e o left pega todos de uma tabela e ainda os em comum com a outra. Por isso o left join apresentou 214 resultados e o outro 192.

**Exemplo Right JOIN:**

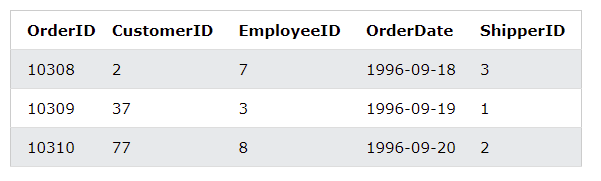
Mesmo esquema do left join, porém agora do lado direito, obviamente.

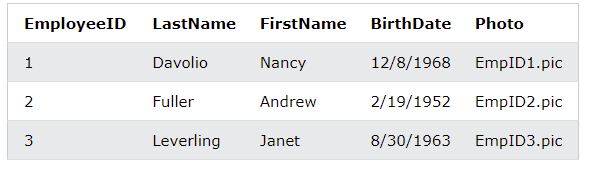
**SELECT** column\_name(s)

**FROM** table1

**RIGHT** **JOIN** table2

**ON** table1.column\_name = table2.column\_name;





**SELECT** Orders.OrderID, Employees.LastName, Employees.FirstName

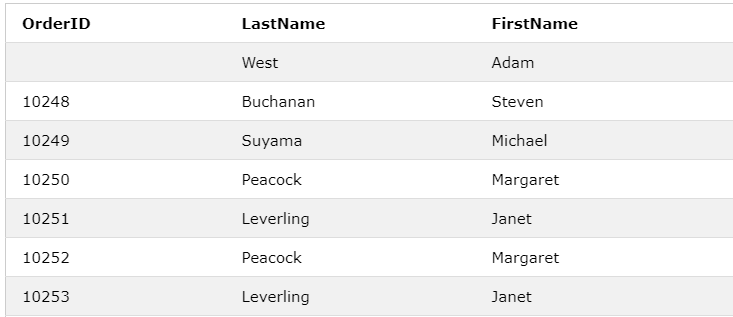
**FROM** Orders

**RIGHT** **JOIN** Employees

**ON** Orders.EmployeeID = Employees.EmployeeID

**ORDER** **BY** Orders.OrderID;

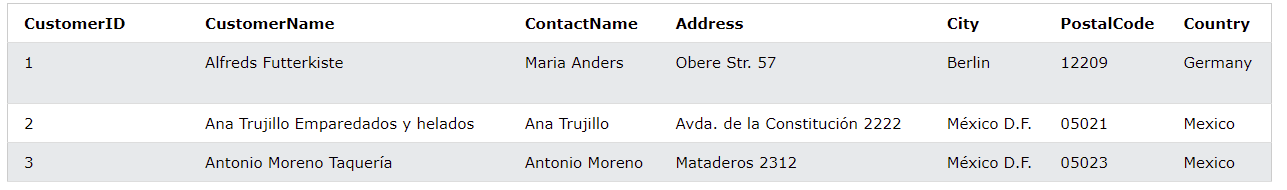
Primeira linha selecionou as colunas desejadas para a nova tabela: Order ID, Last Name e First Name. Segunda linha parte da Orders e da um right join em Employees. Na quarta coluna, como os 2 tem em comum o ID, então troca o Employee ID da tabela orders pelo da tabela de Employee. O On meio que indica quais são as colunas em comum das 2.

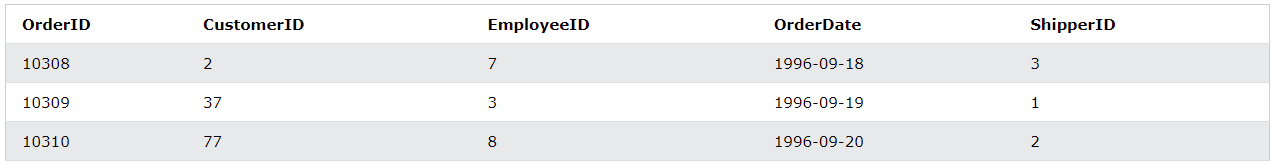


**Exemplo Full Join.**

Agora é hora de unir todos os valores das 2 tabelas, independente de estarem em comum ou não.

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition;





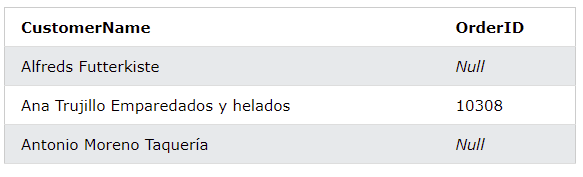
**SELECT** Customers.CustomerName, Orders.OrderID

**FROM** Customers

**FULL** **OUTER** **JOIN** Orders

**ON** Customers.CustomerID=Orders.CustomerID

**ORDER** **BY** Customers.CustomerName;



Esse é o resultado, pq na linha 4 o customer ID da tabela Customers vira o customer ID da tabela Orders, e em comum só tem o order id 10308.

**Exemplo SELF JOIN**

Une as tabelas.

**SELECT** column\_name(s)

**FROM** table1 T1, table1 T2

**WHERE** condition;



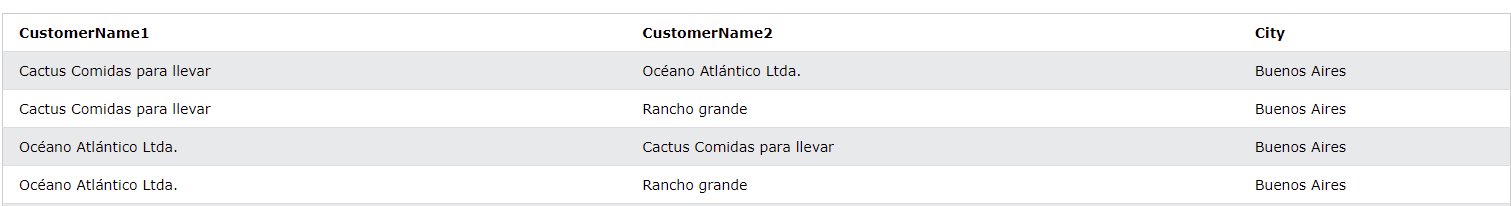
**SELECT** A.CustomerName **AS** CustomerName1, B.CustomerName **AS** CustomerName2, A.City

**FROM** Customers A, Customers B

**WHERE** A.CustomerID <> B.CustomerID

**AND** A.City = B.City

**ORDER** **BY** A.City;



# GROUP BY

Serve para organizar os valores das tabelas mas com base nas funções: COUNT(), MAX(), MIN(), SUM(), AVG().

**SELECT** column\_name(s)

**FROM** table\_name

**WHERE** condition

**GROUP** **BY** column\_name(s)

**ORDER** **BY** column\_name(s);



Exemplo: Seleciona os países em grupos e conta eles.

**SELECT** **COUNT**(CustomerID), Country

**FROM** Customers

**GROUP** **BY** Country;



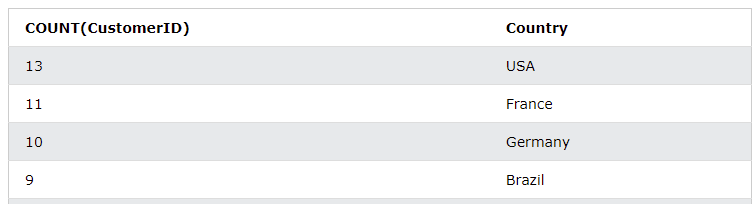
Da para arrumar a tabela na ordem de contagem do maior para o menor

**SELECT** **COUNT**(CustomerID), Country

**FROM** Customers

**GROUP** **BY** Country

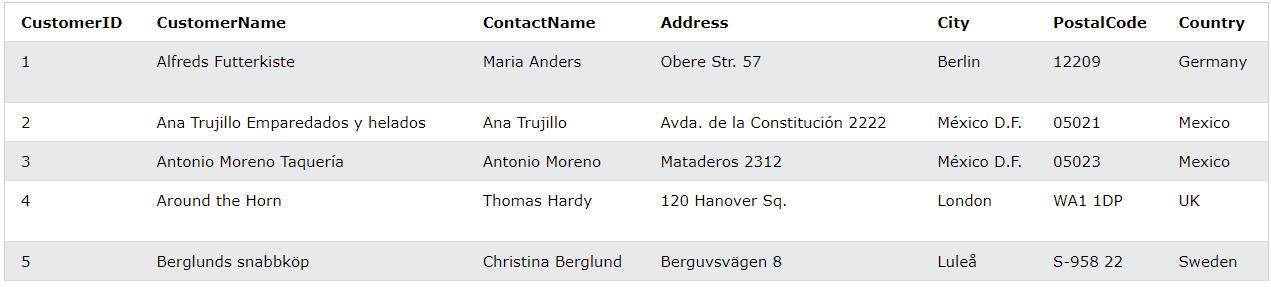
**ORDER** **BY** **COUNT**(CustomerID) DESC;



* **HAVING**

Having é usado para realizar um filtro onde o Where não pode ser usado. É bem usado com o Group by.

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

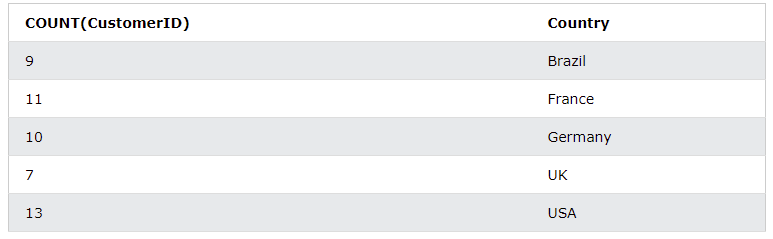


**SELECT** **COUNT**(CustomerID), Country

**FROM** Customers

**GROUP** **BY** Country

**HAVING** **COUNT**(CustomerID) > 5;

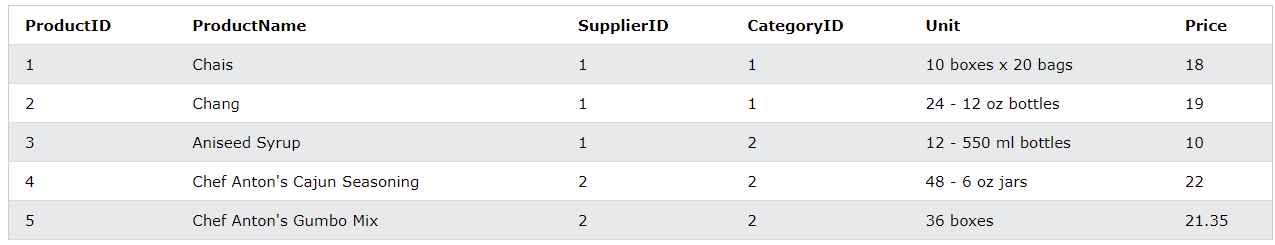


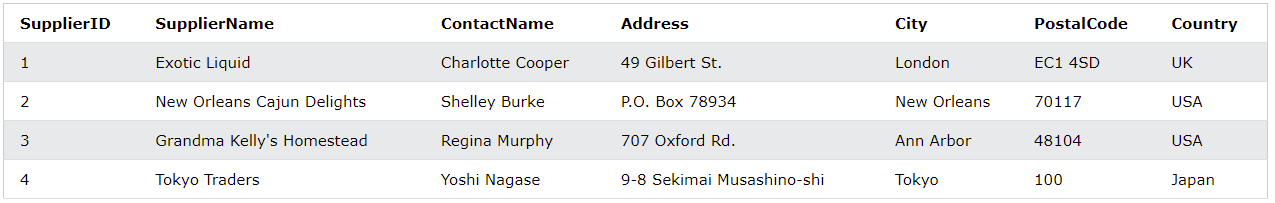
Selecionou os países com count maior que 5.

# WHERE

É usado para testar se existe algum dado na subquery. Ele retorna true or false.

SELECT column\_name(s)  
FROM table\_name  
WHERE EXISTS  
(SELECT column\_name FROM table\_name WHERE condition);



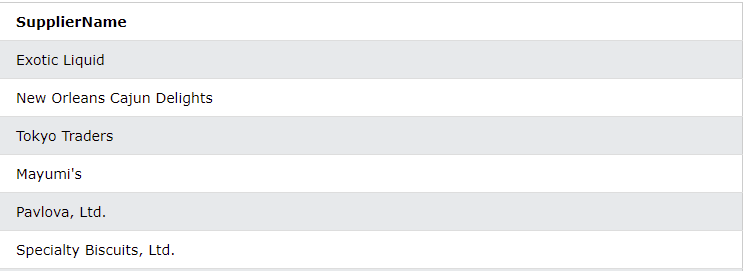


Temos as 2 tabelas: Products e Suppliers.

**SELECT** SupplierName

**FROM** Suppliers

**WHERE** **EXISTS** (**SELECT** ProductName **FROM** Products **WHERE** Products.SupplierID = Suppliers.supplierID AND Price < 20);



Retornou os SupllierName da tabela Suppliers, mas filtrando os dados da tabela Products, onde os preços são menores que 20.

# DATABASE

# CREATE DB

Para criar um Database apenas use o comando:

**CREATE** **DATABASE** databasename;

# DROPDATABASE

Serve para deletar a database criada. Tomar cuidado pois deletará completamente todos os dados existentes.

**DROP** **DATABASE** databasename;

# BACKUP DATABASE

Para criar um backup da database, utilize o comando

**BACKUP** **DATABASE** databasename

**TO** **DISK** = 'filepath'; (C:\ backups\teste...)

# CREATE TABLE

Use os seguintes comandos para criar a tabela e suas colunas

**CREATE** **TABLE** table\_name (

column1 datatype,

column2 datatype,

column3 datatype,

....

);

Exemplo:

**CREATE** **TABLE** Persons (

PersonID int,

LastName varchar(255),

FirstName varchar(255),

Address varchar(255),

City varchar(255)

);

# DROP TABLE

Usado para deletar apenas a tabela

DROP TABLE *table\_name*;

# ALTER TABLE

É um comando usado para realizar alteração na tabela, como adicionar ou retirar colunas e dados.

ALTER TABLE table\_name  
ADD column\_name datatype;

# CONSTRAINSTS

Usado em tabelas que foram criadas com o comando CREATE TABLE ou alteradas com comando ALTER TABLE.

**CREATE** **TABLE** table\_name (

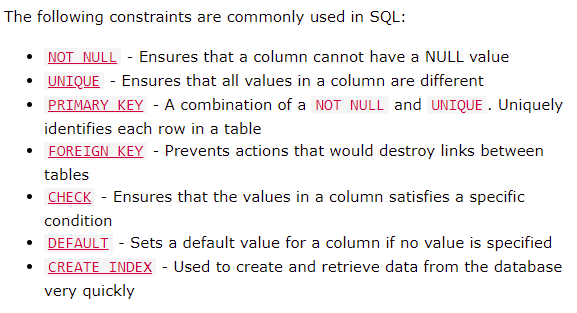
column1 datatype constraint,

column2 datatype constraint,

column3 datatype constraint,

....

);



# UNIQUE

É uma constraint que faz com que os valores dessa coluna sejam diferentes, e relacionados com uma Primary Key. Pode-se ter várias em uma mesma tabela

**CREATE** **TABLE** Persons (

ID int NOT NULL UNIQUE,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int

);

# PRIMARY KEY

Em uma tabela só pode ter UMA primary key e não pode ter valores tipo Null.

**CREATE** **TABLE** Persons (

ID int **NOT** **NULL**,

LastName varchar(255) **NOT** **NULL**,

FirstName varchar(255),

Age int,

**PRIMARY** **KEY** (ID)

);

Criou-se na tabela Persons a primary key na coluna ID, e observa-se que ID tem a constraint NOT NULL.

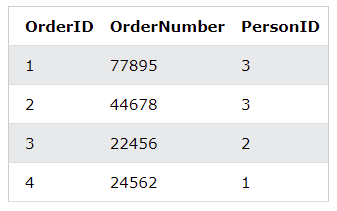
Pode-se adicionar uma Primary Key em uma tabela, a alterando. Usa-se o comando:

ALTER TABLE Persons  
ADD PRIMARY KEY (ID);

# FOREIGN KEY

A foreign key faz a relação com as Primary Keys de outra tabela. A Foreign é a tabela filha e a primary é a tabela mãe.





A primeira tabela é a de Pessoas e a segunda é de Orders.

A coluna Person ID está presente entre as 2 tabelas, onde a coluna da tabela de Pessoas se liga à de orders, ou seja, é uma Primary Key, e a outra uma Foreign Key. A foreign key previne dados inválidos de serem colocados numa coluna foreign key.

**CREATE** **TABLE** Orders (

OrderID int NOT NULL,

OrderNumber int NOT NULL,

PersonID int,

**PRIMARY** **KEY** (OrderID),

**FOREIGN** **KEY** (PersonID) **REFERENCES** Persons(PersonID)

);

# CHECK

A constraint CHECK é usada para limitar o valor do range que pode ser colocado em uma coluna. Se definiar o check numa coluna so vai permitir a inserção de valores já definidos naquela coluna, limitando-a.

**CREATE** **TABLE** Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

**CHECK** (Age>=18)

);

# DEFAULT

Adiciona valores já definidos, quando não forem adicionados novos valores na tabela.

**CREATE** **TABLE** Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

City varchar(255) **DEFAULT** 'Sandnes'

);

Adicionará a palavra Sandness na coluna City quando não for adicionado nenhum outro valor.

# INDEX

Usado para criação de índices nas tabelas. São muito úteis para a realização rápida de pesquisas e queries.

CREATE INDEX index\_name  
ON table\_name (column1, column2, ...);

Para deletar:

DROP INDEX index\_name ON table\_name;

# AUTO INCREMENT

Auto incrementa-se um valor único para ser gerado automaticamente quando um novo registro é colocado na tabela.

**CREATE** **TABLE** Persons (

Personid int NOT NULL AUTO\_INCREMENT,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

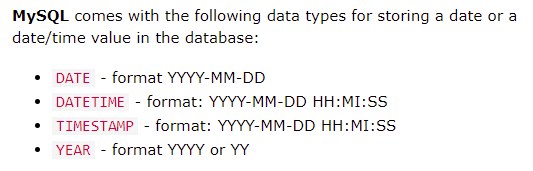
Age int,

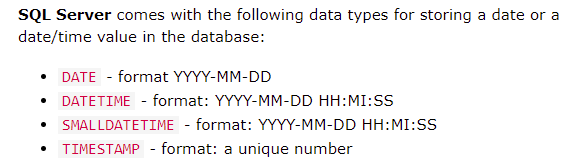
**PRIMARY** **KEY** (Personid)

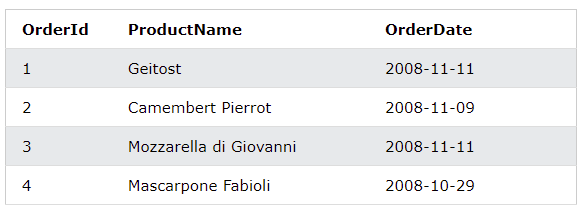
);

Assim o person Id terá o incremento de 1,2,3...automaticamente.

# DATES

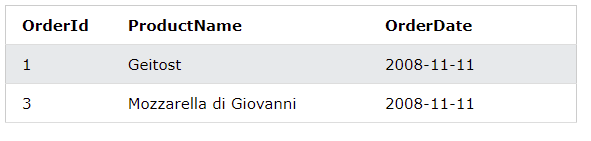






Pode-se selecionar as datas que deseja-se observar:

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'



# VIEWS

View é uma tabela virtual baseada no resultado de um statement do SQL.

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

Exemplo:

**CREATE** **VIEW** [Brazil Customers] **AS**

**SELECT** CustomerName, ContactName

**FROM** Customers

**WHERE** Country = 'Brazil';

Criou-se uma tabela onde tem os Customers do Brazil, com os dados de customer name e contact name tirados da tabela Customers.

# DATA TYPES

**STRINGS:**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHAR(size) | A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1 |
| VARCHAR(size) | A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535 |
| BINARY(size) | Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1 |
| VARBINARY(size) | Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes. |
| TINYBLOB | For BLOBs (Binary Large Objects). Max length: 255 bytes |
| TINYTEXT | Holds a string with a maximum length of 255 characters |
| TEXT(size) | Holds a string with a maximum length of 65,535 bytes |
| BLOB(size) | For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data |
| MEDIUMTEXT | Holds a string with a maximum length of 16,777,215 characters |
| MEDIUMBLOB | For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data |
| LONGTEXT | Holds a string with a maximum length of 4,294,967,295 characters |
| LONGBLOB | For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data |
| ENUM(val1, val2, val3, ...) | A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them |
| SET(val1, val2, val3, ...) | A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list |

**NUMERIC**:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| BIT(*size*) | A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1. |
| TINYINT(*size*) | A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255) |
| BOOL | Zero is considered as false, nonzero values are considered as true. |
| BOOLEAN | Equal to BOOL |
| SMALLINT(*size*) | A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255) |
| MEDIUMINT(*size*) | A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255) |
| INT(*size*) | A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255) |
| INTEGER(*size*) | Equal to INT(size) |
| BIGINT(*size*) | A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255) |
| FLOAT(*size*, *d*) | A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions |
| FLOAT(*p*) | A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE() |
| DOUBLE(*size*, *d*) | A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter |
| DOUBLE PRECISION(*size*, *d*) |  |
| DECIMAL(*size*, *d*) | An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0. |
| DEC(*size*, *d*) | Equal to DECIMAL(size,d) |

**DATE:**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| DATE | A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31' |
| DATETIME(*fsp*) | A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time |
| TIMESTAMP(*fsp*) | A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT\_TIMESTAMP and ON UPDATE CURRENT\_TIMESTAMP in the column definition |
| TIME(*fsp*) | A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59' |
| YEAR | A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format. |