1. SELECT

A instrução **SELECT é usada para selecionar dados de um banco de dados.**
Os dados retornados **são armazenados em uma tabela de resultado**.

Exemplo:

```
SELECT column1, column2, ...
FROM table_name;
```

Aqui, column1, column2, ... são os **nomes de campo da tabela da qual você deseja selecionar dados**, tabela essa escrita no **FROM**.

1.1. SELECT *

A seguinte instrução SQL seleciona **todas (*)** as **colunas da tabela "Clientes"**:

```
SELECT * FROM Customers;
```

1.2. SELECT DISTINCT

A instrução **SELECT DISTINCT** é usada para **retornar apenas valores distintos (diferentes)**.

Dentro de uma tabela, uma coluna geralmente contém muitos valores duplicados; E às vezes você deseja listar apenas os diferentes valores (distintos).

```
SELECT DISTINCT Country FROM Customers;
```

Essa instrução SQL seleciona apenas os valores distintos da coluna "país" na tabela "clientes".

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

Essa declaração SQL **conta (COUNT) e retorna o número de países diferentes (distintos)** da tabela "clientes".

2. WHERE

A cláusula WHERE é usada para filtrar registros.

É usado para **extrair apenas os registros que atendem a uma condição especificada**.

Exemplo:

```
SELECT * FROM Customers
WHERE Country = 'Mexico';
```

Essa declaração SQL seleciona todos os dados dos clientes do "México".

2.1. Operadores na Cláusula WHERE

2.1.1. = –> Igual

2.1.2. > –> Maior que

2.1.3. < –> Menor que

2.1.4. >= –> Maior/Igual que

2.1.5. <= –> Menor/Igual que

2.1.6. != ou <> –> Not Equal, no caso Diferente que

2.1.7. BETWEEN –> Entre um certo intervalo

Exemplo:
SELECT * FROM Products
WHERE Price **BETWEEN** 50 AND 60;

2.1.8. LIKE –> Procura um padrão

Exemplo:
SELECT * FROM Customers
WHERE City **LIKE** 's%';
(Aqui pega tudo dos clientes onde sua cidade começa com “s”).

2.1.9. IN –> Para especificar vários valores possíveis para uma coluna

Exemplo:
SELECT * FROM Customers
WHERE City **IN** ('Paris','London');
(Pega tudo onde as cidades são ou Paris ou London)

Exemplo 2:

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

Seleciona todos os clientes que não estão localizados na "Alemanha", "França" ou "Reino Unido".

Exemplo 3:

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

Seleciona todos os clientes que são dos mesmos países que os fornecedores.

3. AND, OR e NOT

A cláusula **WHERE** pode ser combinada com **AND, OR e NOT**.

Exemplos:

```
SELECT * FROM Customers
WHERE Country = 'Germany' AND (City = 'Berlin' OR City =
'Stuttgart');
```

Essa declaração SQL seleciona todos os campos de "clientes" onde o país é "Alemanha" e a cidade deve ser "Berlim" ou "Stuttgart" (use parênteses para formar expressões complexas).

```
SELECT * FROM Customers
WHERE NOT Country = 'Germany' AND NOT Country = 'USA';
```

E essa declaração SQL seleciona todos os campos de "clientes" onde o país não é "Alemanha" e nem "EUA".

4. ORDER BY

O ORDER BY é **usado para classificar o conjunto de resultados em ordem ascendente ou decrescente.**

O **ORDER BY classifica os registros em ordem crescente por padrão**. Para classificar os registros em **ordem descendente**, use a palavra-chave **DESC.**

Exemplos:

```
SELECT * FROM Customers
ORDER BY Country;
```

Essa instrução SQL seleciona todos os clientes da tabela "clientes", classificados pela coluna "país". (ordem alfabética)

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

A seguinte instrução SQL seleciona todos os clientes da tabela "Clientes", classificados por ordem **crescente** pelo "país" e **decrescente** pela coluna "CustomerName".

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
64	Rancho grande	Sergio Gutiérrez	Av. del Libertador 900	Buenos Aires	1010	Argentina
54	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
59	Piccolo und mehr	Georg Pipps	Geislweg 14	Salzburg	5020	Austria
20	Ernst Handel	Roland Mendel	Kirchgasse 6	Graz	8010	Austria
76	Suprêmes délices	Pascale Cartrain	Boulevard Tirou, 255	Charleroi	B-6000	Belgium
50	Maison Dewey	Catherine Dewey	Rue Joseph-Bens 532	Bruxelles	B-1180	Belgium
88	Wellington Importadora	Paula Parente	Rua do Mercado, 12	Resende	08737-363	Brazil

5. UNION

O operador **UNION** é usado para **combinar o conjunto de resultados de duas ou mais instruções selecionadas.**

- Cada SELECT que tenha UNION **deve ter o mesmo número de colunas;**
- As colunas também **devem ter tipos de dados semelhantes**;
- As colunas em todos SELECT's também devem estar na mesma ordem.

Exemplos:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Se alguns clientes ou fornecedores tiverem a mesma cidade, cada cidade será listada **apenas uma vez**, porque a **UNION seleciona apenas valores distintos.**

5.1. UNION ALL

Use a **Union All** para selecionar também valores duplicados!

Exemplo:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

Esse SQL retorna as cidades alemãs (também os valores duplicados) da tabela "clientes" e "fornecedores":

6. MIN e MAX

A função **MIN()** retorna o **menor valor da coluna selecionada.**

A função **MAX()** retorna o **maior valor da coluna selecionada.**

Exemplos:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;

SELECT MAX(Price) AS LargestPrice
FROM Products;
```

7. COUNT, AVG e SUM

A função **COUNT() retorna o número de linhas que correspondem a um critério especificado.**

Exemplo:

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

A função **AVG() retorna o valor médio de uma coluna numérica.**

Exemplo:

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

A função **SUM() retorna a soma total de uma coluna numérica.**

Exemplo:

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

8. LIKE

O operador **LIKE** é usado em uma cláusula **WHERE para procurar um padrão especificado em uma coluna.**

Existem dois coringas frequentemente usados em conjunto com o operador LIKE:

- **O sinal percentual (%) representa zero, um ou vários caracteres;**
- **O sinal de sublinhamento (_) representa um, um único caractere.**

O sinal percentual e o sublinhado também podem ser usados em combinações!

Operador LIKE	Descrição
WHERE CustomerName LIKE 'a%'	Encontra quaisquer valores que começam com "A"
WHERE CustomerName LIKE '%a'	Encontra quaisquer valores que terminem com "A"
WHERE CustomerName LIKE '%or%'	Encontra quaisquer valores que tenham "or" em qualquer posição
WHERE CustomerName LIKE '_r%'	Encontra quaisquer valores que tenham "r" na segunda posição
WHERE CustomerName LIKE 'a_%'	Encontra quaisquer valores que começam com "A" e têm pelo menos 2 caracteres de comprimento
WHERE CustomerName LIKE 'a__%’	Encontra quaisquer valores que começam com "A" e têm pelo menos 3 caracteres de comprimento
WHERE ContactName LIKE 'a%o'	Encontra quaisquer valores que começam com "A" e termina com "O"

9. GROUP BY

O **GROUP BY agrupa linhas que possuem os mesmos valores em linhas resumidas**, como "Encontre o número de clientes em cada país".

O **GROUP BY** é frequentemente usado com funções agregadas (**count (), max (), min (), sum (), avg ()**) para agrupar o conjunto de resultados por uma ou mais colunas.

Exemplo:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

Esse SQL lista o número de clientes em cada país.

10. HAVING

A cláusula **HAVING** foi adicionada ao SQL porque a palavra-chave **WHERE não pode ser usada com funções agregadas.**

Exemplo:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

Esse Select lista o número de clientes em cada país. Inclui somente países com mais de 5 clientes.

11. Valores NULL

Não é possível testar valores nulos com operadores de comparação, como =, <, ou <>.

Teremos que usar o **IS NULL** e o **NOT NULL** ao invés desses.

- Conectivos lógicos AND, OR e NOT
  - NOT desconhecido = desconhecido
  - OR = verdadeiro, se um dos dois argumentos for verdadeiro e outro desconhecido
  - OR = desconhecido, se um dos dois argumentos for falso e o outro desconhecido
  - OR = desconhecido, se os dois argumentos forem desconhecidos
  - AND = falso, se um dos dois argumentos for falso e o outro desconhecido
  - AND = desconhecido, se um dos dois argumentos for verdadeiro e o outro desconhecido
  - AND = desconhecido, se os dois argumentos forem desconhecidos

Desconhecido = NULL

- Os operadores aritméticos +, -, * e / retornam NULL se um de seus operandos for NULL
- Mas, count(*) trata NULL exatamente como outros valores, ou seja, eles são contados!
- Todas as outras operações agregadas (SUM, AVG, MIN, MAX, e variações usando DISTINCT) desconsideram o valor NULL

11.1. IS NULL

O operador **IS NULL** é usado para **testar valores vazios (valores nulos)**.

Exemplo:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

Esse SQL lista todos os clientes com um valor nulo no campo "Endereço".

11.2. NOT NULL

O operador **NOT NULL** é usado para testar valores não vazios (valores não nulos).

Exemplo:

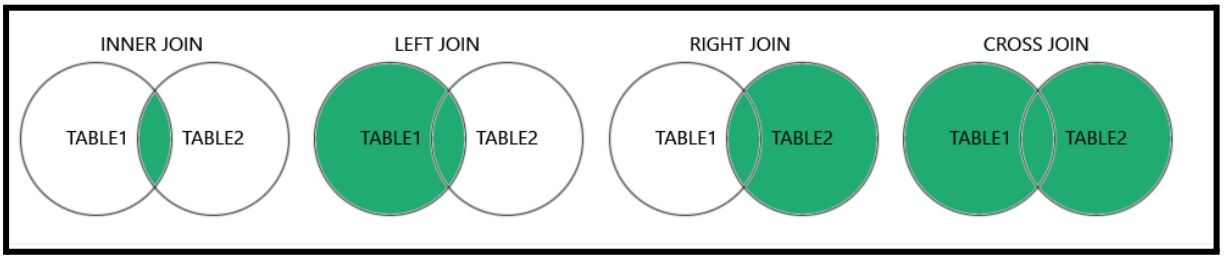
```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

O seguinte SQL lista todos os clientes com um valor no campo "Endereço".

12. JOINS

A cláusula **JOIN** é usada **para combinar linhas de duas ou mais tabelas, com base em uma coluna relacionada entre elas**.

- **INNER JOIN**: Retorna registros que têm **valores correspondentes em ambas as tabelas**;
- **LEFT JOIN**: Retorna todos os **registros da tabela esquerda e os registros correspondentes da tabela direita**;
- **RIGHT JOIN**: Retorna todos os **registros da tabela certa e os registros correspondentes da tabela esquerda**;
- **CROSS JOIN**: **Retorna todos os registros de ambas as tabelas.**



Exemplo:

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

Importante lembrar que quando tiver o **JOIN**, o seu “WHERE” seria usando o **ON**.

Sintaxe:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

12.1. USING

- INNER JOIN e USING
  - Nomes dos campos usados na junção não precisam ser especificados se forem iguais nas duas tabelas
  - Pode ser usado quando outros campos das tabelas tem o mesmo nome e não se deseja usá-los na junção
  - Requer que o nome do campo seja colocado sem o nome da tabela

```
SELECT      Marinheiros.nome-marin
FROM        Marinheiros
INNER JOIN  Reservas
    USING (id-marin)
```

O USING() é apenas uma sintaxe mais conveniente, enquanto que o ON permite um pouco mais de flexibilidade quando os nomes das colunas não são idênticos.

12.2. NATURAL JOIN

- NATURAL JOIN
  - Nomes dos campos usados na junção não precisam ser especificados se forem iguais nas duas tabelas
  - Pode ser usado apenas quando todos campos das tabelas tem o mesmo nome participam da junção

```
SELECT          Marinheiros.nome-marin
FROM            Marinheiros
NATURAL JOIN    Reservas
```

12.3. OUTER JOIN

- Junções descartam valores não correspondentes
  - Junção Interna
- Junções externas funcionam de maneira diferente
  - Uma junção externa não requer que os registros de uma tabela possuam registros equivalentes em outra
    - NULL

- Junção Externa
  - Esquerda
  - Direita
  - Completa
- É possível desabilitar valores nulos quando criamos campos em tabelas, usando NOT NULL

Existem **3 tipos de OUTER JOIN**, e eles são:

- LEFT OUTER JOIN
  - O resultado desta seleção sempre contém todos os registros da tabela esquerda (isto é, a primeira tabela mencionada na consulta), mesmo quando não exista registros correspondentes na tabela direita
  - Pode-se usar ON, USING ou NATURAL

```
SELECT          Marinheiros.nome-marin
FROM            Marinheiros
LEFT OUTER JOIN Reservas
        ON (Marinheiros.id-marin = Reservas.id-marin)
```

- RIGHT OUTER JOIN
  - O resultado desta seleção sempre contém todos os registros da tabela direita (isto é, a segunda tabela mencionada na consulta), mesmo quando não exista registros correspondentes na tabela esquerda
  - Pode-se usar ON, USING ou NATURAL

```
SELECT              Marinheiros.nome-marin
FROM                Marinheiros
RIGHT OUTER JOIN    Reservas
    USING (id-marin)
```

- FULL OUTER JOIN
  - O resultado desta seleção apresenta todos os dados das tabelas à esquerda e à direita, mesmo que não possuam correspondência entre as tabelas

```
SELECT  Marinheiros.nome-marin
FROM    Marinheiros
NATURAL FULL OUTER JOIN Reservas
```

13. EXISTS

O operador **EXISTS** é usado para testar a existência de qualquer registro em uma subconsulta.

O operador **EXISTS** retorna true se a subconsulta retornar um ou mais registros.
Sintaxe:

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

Ex:

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

Retorna True e lista os fornecedores com um preço do produto inferior a 20

14. ANY e ALL

ANY e **ALL** permitem executar uma comparação entre um único valor da coluna e uma variedade de outros valores.

- **ANY**:
  - Retorna um valor booleano como resultado;
  - Retorna **True** se algum dos valores da subquery atende à condição;
  - **ANY** significa que a condição será verdadeira se a operação for verdadeira para qualquer um dos valores no intervalo.
  - Ex:

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
   (SELECT ProductID
   FROM OrderDetails
   WHERE Quantity = 10);
```

  - Lista o nome do produto se encontrar algum registro na tabela OrderDetails tem quantidade igual a 10 (isso retornará verdadeiro porque a coluna de quantidade possui alguns valores de 10)

- **ALL:**
  - Retorna um valor booleano como resultado;
  - Retorna **True** se **todos** os valores da subquery atenderem à condição;
  - É usado com **SELECT, WHERE e HAVING**;
  - **ALL** significa que a condição será verdadeira apenas se a operação for verdadeira para todos os valores no intervalo.
  - Ex:

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
   (SELECT ProductID
   FROM OrderDetails
   WHERE Quantity = 10);
```

  - Lista o nome do produto se **todos** os registros na tabela OrderDetails têm quantidade **igual a 10**. Isso retornará,

obviamente, **false** porque a coluna de quantidade possui muitos valores diferentes (não apenas o valor de 10).

1º Forma Normal

- Primeira Forma Normal
  - Todos os atributos admitem apenas valores atômicos
  - Não admite-se atributos compostos, multi-valorados e relações aninhadas

- Não está na Primeira Forma Norma:
  - Atributo composto (Endereço)
  - Atributo Multi-valorado (Telefone)

ID	Nome	Telefone	Endereço
1	José	9838-0021 9838-0021	Avenida Brasil, 2033 - São Paulo/SP
2	Maria	8837-0012 2234-1121	Avenida Maranhão, 2933 - Rio de Janeiro/RJ
3	João	9739-0023	Rua Mato Grosso, 44 Belo Horizonte/MG
4	Carlos	3334-1022	Avenida Paulista, 1000 - São Paulo/SP

- Atributo composto é decomposto.

ID	Nome	Telefone	Logradouro	Cidade	UF
1	José	9838-0021 9838-1221	Avenida Brasil, 2033	São Paulo	SP
2	Maria	8837-0012 2234-1121	Avenida Maranhão, 2933	Rio de Janeiro	RJ
3	João	9739-0023	Rua Mato Grosso, 44	Belo Horizonte	MG
4	Carlos	3334-1022	Avenida Paulista, 1000	São Paulo	SP

- Atributo multi-valorado é decomposto.
- Agora a relação está na Primeira Forma Normal

ID	Nome	Logradouro	Cidade	UF
1	José	Avenida Brasil, 2033	São Paulo	SP
2	Maria	Avenida Maranhão, 2933	Rio de Janeiro	RJ
3	João	Rua Mato Grosso, 44	Belo Horizonte	MG
4	Carlos	Avenida Paulista, 1000	São Paulo	SP

ID	Telefone
1	9838-0021
1	9838-1221
2	8837-0012
2	2234-1121
3	9739-0023
4	3334-1022

2º Forma Normal

- Segunda Forma Normal
  - Estar na 1FN
  - Todos os atributos que não fazem parte da chave são totalmente dependentes da chave primária
    - Ou seja, dependentes de todos os atributos que compõem a chave primária
  - Se uma relação estiver na 1FN e sua chave primária for simples (ou seja, não composta), ela já está na 2FN
  - Nada é dito a respeito de dependências mútuas entre os atributos que pertencem às chaves
  - Nada é dito a respeito de dependências mútuas entre os atributos que não pertencem às chaves

- Não está na Segunda Forma Normal
  - $\{\text{id-curso}\} \rightarrow \{\text{descrição-curso}\}$

id-aluno	**id-curso**	**nota**	**descrição-curso**
1	1	5.9	Banco de dados
1	2	8.1	Engenharia de software
2	2	9.3	Engenharia de software
3	1	4.6	Banco de dados

- Agora, ambas estão na Segunda Forma Normal

id-aluno	**id-curso**	**nota**
1	1	5.9
1	2	8.1
2	2	9.3
3	1	4.6

id-curso	**descrição-curso**
1	Banco de dados
2	Engenharia de software

3º Forma Normal

- Terceira Forma Normal
  - Estar na 2FN
  - Nenhum atributo que não faz parte de uma chave pode ser transitivamente dependente da chave primária
    - Ou seja, atributos que não pertencem à alguma chave não dependem de nenhum atributo que também não pertence à nenhuma chave
    - Ou seja (mais uma vez), todos os atributos não chave são completamente dependentes dos atributos chave e são independentes entre si
  - Todas as relações na 2FN com apenas um atributo não chave estão na 3FN
  - Nada é dito a respeito de dependências mútuas entre os atributos que pertencem às chaves

- Relação Funcionário
- Há atributos que são funcionalmente dependentes de outros atributos não chave
  - {id-cargo} $\rightarrow$ {descrição-cargo}

id	**nome**	**id-cargo**	**descrição-cargo**
1	João	2	Professor
2	José	1	Recepcionista
3	Maria	1	Recepcionista
4	Zeca	2	Professor

- Relação Funcionário

id	nome	id-cargo
1	João	2
2	José	1
3	Maria	1
4	Zeca	2

- Relação Cargo

id	descrição
1	Recepcionista
2	Professor

FNBC

- A FNBC é uma versão mais restrita da 3FN
  - Na 3FN, calcular todas as chaves de uma relação usando dependências funcionais é um problema NP-Completo
  - Na FNBC é necessário apenas verificar se o lado esquerdo de cada DF em F é uma superchave (computando o fechamento de atributo)
- É possível garantir que considerando todos os campos de uma relação em FNBC, uma parte da informação não pode ser inferida (usando apenas DFs) a partir dos valores em todos os outros campos na instância da relação