POWER BI + SQL SERVER

SQL Server → Aprender como usar o sql server como fonte de dados para o power bi.

O que é?

O SQL Server é um sistema gerenciador de banco de dados relacional (SGBD) criado pela Microsoft.
Ele serve para armazenar, organizar e gerenciar grandes volumes de dados, permitindo que aplicativos, sistemas e usuários acessem, modifiquem e consultem essas informações usando a linguagem SQL (Structured Query Language).

Ótima opção para tratar dados e trazer transformados para o Power Query

O que o SQL Server faz?

- Armazena dados de forma estruturada (em tabelas, linhas e colunas).
- Permite consultas e manipulação de dados com SQL.
- Garante segurança, integridade e performance no acesso a dados.
- Suporta transações, procedimentos armazenados, triggers, views, etc.
- Pode ser usado por empresas para gerenciar dados de sistemas, sites, ERPs, etc.

Qual a ligação entre SQL Server e Power BI?

O Power BI é uma ferramenta da Microsoft para análise e visualização de dados.

- O Power BI se conecta diretamente ao SQL Server para buscar dados.
- Você pode criar relatórios e dashboards interativos com os dados armazenados no SQL Server.
- A conexão pode ser feita:
    - Local (se o banco estiver na sua máquina ou rede)
    - Online (se o banco estiver na nuvem, como no Azure SQL Server)
- É muito comum em empresas usarem SQL Server como fonte de dados principal para relatórios no Power BI.

Processo → Foi baixado um banco de dados fictício da Microsoft, Contoso Corporation. Após baixado foi importado dentro do **SQL Management Studio.** Abrir uma nova consulta para inserir os códigos e funções.

FUNÇÕES

SELECT

SELECT → Define as colunas que quer retornar (pode ser * para todas).
Ex: SELECT * FROM CLIENTES (todas as colunas)
SELECT nome, idade FROM clientes (Coluna específica)

FROM → Define a tabela ou visão de onde os dados virão.

É possível executar mais de uma consulta na mesma ação. Se você selecionar uma ou mais linhas de código com o mouse e clicar em executar, apenas o trecho selecionado será processado.
É valido usar SELECT + TOP(N), no qual N é um número natural que junto que com o SELECT faz um filtro selecionado de determinada coluna/tabela com base no TOP 5 (Os primeiros N elementos).

COMANDOS

DISTINCT → SELECT DISTINCT “COLUNA”/* FROM “TABELA” mostra as linhas diferentes, distintas.

AS → SELECT FIRST_NAME AS NOME, DATE AS “ANIVERSÁRIO” FROM TABELA
AS renomeia colunas de uma tabela (Se tiver acento usar aspas)

ORDER BY → Order by ordenada uma coluna (no caso a coluna renomeada para “NOME”) em order ascendente (ASC ou DESC).

SELECT FIRST_NAME AS NOME, DATE AS “ANIVERSÁRIO” FROM TABELA
ORDER BY NOME ASC.

WHERE → Permite fazer uma filtragem simples e poderosa.
SELECT * FROM TABELA
WHERE FIRST_NAME = ‘ENZO ARRUE JUAN FUSO’ // outro exemplo: WHERE END_DATE IS NULL

WHERE, permite fazer uma filtragem de tabela de acordo com um texto exato da tabela. Caso quisermos buscar apenas um palavra sem saber sem saber exatamente o conteúdo escrito fazemos da seguinte maneira:

LIKE + % % →
SELECT * FROM TABELA
WHERE FIRST_NAME LIKE = ‘%ARRUE%’

**Usar logo após o SELECT

OPERADORES LÓGICOS

NOT → Negação de uma lógica
WHERE END_DATE IS NOT NULL → Contrário de quando o valor for NULL

AND → E. Todas as condições precisam ser verdadeiras ao mesmo tempo.
SELECT * FROM TABELA
WHERE Price >= 1000 AND Brand_Name = 'Contoso'

OR→ Ou. Basta **uma** das condições ser verdadeira para o resultado ser incluído.
SELECT * FROM TABELA
WHERE NAME 'Fabrikam' OR NAME = 'Contoso'
Pode ser feito assim:
WHERE NAME IN ('Contoso', 'Fabrikam')

BETWEEN → Intervalo de duas datas selecionadas
SELECT * FROM DimEmployee
WHERE StartDate BETWEEN '2000-01-01' AND '2000-12-31'

FUNÇÕES AGREGADORAS

DAX e SQL Server possuem semelhança porque ambos usam conceitos do SQL e operam sobre dados tabulares (organizados **em linhas e colunas)**, mantendo funções universais de agregação com nomes iguais.

SUM → Somatória dos elementos de uma coluna

AVERAGE → Média a partir dos elementos de uma coluna

MAX → Retorna o valor máximo, mais alto de uma coluna
Ex: SELECT SUM(SalesAmount) AS 'Total Vendido' FROM FactSales

COUNT → Conta o número de registros em um conjunto de resultados. Ele pode contar todas as linhas ou apenas valores não nulos de uma coluna.

COMANDOS + AGREGAÇÕES

GROUP BY → O GROUP BY reúne linhas que têm valores iguais numa ou mais colunas, formando grupos, para que possamos calcular algo (como soma, contagem, média) para cada grupo separado.

SELECT BrandName, COUNT(*) AS 'Qt produtos' FROM DimProduct GROUP BY BrandName

O GROUP BY junta os produtos que têm a mesma marca e a função COUNT(*) conta quantos produtos tem em cada grupo (marca).

Ou seja: para cada marca, ele soma quantos produtos existem e mostra essa quantidade junto com o nome da marca

Outro exemplo útil:
SELECT ClassName, AVG(UnitPrice) FROM DimProduct

GROUP BY ClassName

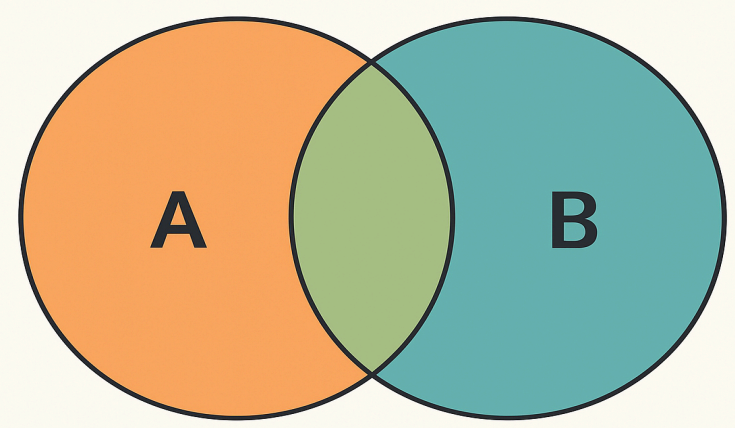
Seleciona a coluna ClassName faz a média do preço e por conto do GROUP BY ele liga os grupos com a média dos produtos internos.

CONTINUAÇÃO COMANDOS

JOIN → O JOIN no SQL serve para juntar dados de tabelas diferentes usando um campo que elas têm em comum (geralmente um ID).
Pensa que você tem:

- Uma tabela com clientes
- Outra com pedidos

O JOIN vai ligar cada cliente com seus pedidos usando o campo que existe nas duas, tipo ClienteID.



INNER JOIN

Pega **apenas o que tem correspondência nas duas tabelas** (Interseção do diagrama)
➡ Se não tiver ligação, some da consulta.

LEFT JOIN

Pega **tudo da tabela da esquerda** (LEFT) e o que tem a combinar com a tabela da direita. (Interseção mais círculo da esquerda)
➡ Se não achar nada na direita, preenche com **NULL**.
Pega todos os clientes que têm pedidos e clientes que não possuem.
Ex:

Círculo Clientes	Círculo ID Pedidos
Alan	2
Faber	32212
João	444
Gabi	NULL

RIGHT JOIN

Igual ao LEFT, mas pega **tudo da tabela da direita** e o que combinar da esquerda.
➡ O que não tiver correspondência na esquerda, fica **NULL**.

FULL JOIN

Pega **tudo das duas tabelas**.
➡ Se não achar correspondência, coloca NULL no lado que faltar.

Exercício

Objetivo: Fazer uma tabela com o nome do produto (DimProduct), nome da subcategoria (DimSubCategory) e nome da categoria (DimCategory) usando o JOIN.

Contexto:

Tabela DimProduct → Nome do produto, Chave da Subcategoria Produto

Tabela DimProductSubCategory → Chave da Subcategoria Produto, Nome da Subcategoria, Chave da Categoria Produto

Tabela DimProductCategory → Chave da Categoria Produto, Nome Categoria Produto

Código:

```
 SELECT ProductName,ProductSubCategoryName, ProductCategoryName

FROM DimProduct AS p

INNER JOIN DimProductSubcategory AS s

ON p.ProductSubcategoryKey = s.ProductSubcategoryKey

INNER JOIN DimProductCategory AS c

ON s.ProductSubcategoryKey = c.ProductCategoryKey
```

Lógica:

Produto sabe a qual subcategoria pertence.
→ `p.ProductSubcategoryKey = s.ProductSubcategoryKey`

Subcategoria sabe a qual categoria pertence.
→ `s.ProductCategoryKey = c.ProductCategoryKey`

Assim, ao ligar Produto → Subcategoria → Categoria, você consegue chegar do produto até sua categoria.

Chave primária (Primary Key – PK)
É um campo que identifica de forma única cada registro em uma tabela.
Exemplo na tabela DimProductSubcategory:

- ProductSubcategoryKey → cada subcategoria tem um número único.

Chave estrangeira (Foreign Key – FK)
É um campo que faz referência à chave primária de outra tabela.
Exemplo na tabela DimProduct:

-ProductSubcategoryKey → aponta para qual subcategoria o produto pertence.
Ele não é único aqui (porque vários produtos podem ter a mesma subcategoria), mas o valor dele deve existir na tabela DimProductSubcategory.

Passo 1 — Definir a tabela principal (tabela "base")

No exemplo, queremos ver produtos, suas subcategorias e categorias.
A pergunta é: *"Onde está o produto?"*
➡ Resposta: Na tabela **DimProduct**.

Essa será a tabela **base** do FROM.
A partir dela, sabemos que da para pegar:

- **ProductName** (existe diretamente nela, não precisa do JOIN para buscar isso).

Passo 2 — Identificar colunas que não estão na base

Também queremos **ProductSubCategoryName**.
➡ Ele não está na **DimProduct**.

Mas na DimProduct existe um campo **ProductSubcategoryKey**, que é uma *chave estrangeira* apontando para a tabela **DimProductSubcategory**.
Logo, para buscar o **nome da subcategoria**, temos que:

- Fazer um JOIN usando ProductSubcategoryKey (campo em comum entre as tabelas).

- Só assim conseguimos acessar ProductSubCategoryName.

- **Passo 3 — Repetir a lógica para a categoria**

Você também quer **ProductCategoryName**.
➡ Ele não está nem na **DimProduct** nem na **DimProductSubcategory**?
Na verdade, ele está na **DimProductCategory**, mas **DimProduct** não liga diretamente a **DimProductCategory**.

O caminho é:

- DimProduct ➡ DimProductSubcategory ➡ DimProductCategory

- A ligação é feita pelo campo **ProductCategoryKey** (que está em DimProductSubcategory e em DimProductCategory).

! Na **DimProduct**, temos apenas o código da subcategoria (`ProductSubcategoryKey`).

Na **DimProductSubCategory**, cada código (`Key`) tem um nome único (`Name`).

O `JOIN` liga as duas tabelas pelo código para trazer o nome correspondente.

IDEIA DE TABELA:

1 Tabela `DimProduct` (produtos)

ProductKey	ProductName	ProductSubcategoryKey
1	Camiseta Azul	10
2	Tênis Runner	20
3	Jaqueta Jeans	10
4	Bicicleta Speed	30

2 Tabela `DimProductSubcategory` (subcategorias)

ProductSubcategoryKey	ProductSubcategoryName	ProductCategoryKey
10	Roupas	100
20	Calçados	100
30	Bicicletas	200

3 Tabela `DimProductCategory` (categorias)

ProductCategoryKey	ProductCategoryName
100	Vestuário
200	Esportes

MONTAGEM DA DIMENSÃO PRODUTO:

CÓDIGO:

```
SELECT ProductKey AS 'ID Produto',

ProductName AS 'Nome',

ProductSubcategoryName AS 'Subcategoria',

ProductCategoryName AS 'Categoria',

BrandName AS 'Marca',
```

```
UnitPrice AS 'Preço Único',

UnitCost AS 'Custo Único'

FROM DimProduct AS p

INNER JOIN DimProductSubcategory AS s

ON p.ProductSubcategoryKey = s.ProductSubcategoryKey

INNER JOIN DimProductCategory as c

ON c.ProductCategoryKey = s.ProductCategoryKey
```

FUNÇÃO CONDICIONAL

CASE → Função condicional que permite criar várias condições.
Começa com CASE e termina com END. A condição usa WHEN, após a condição retorna um valor verdadeiro usando THEN.
ELSE para um valor fora das acimas.

Ex:

```
CASE

    WHEN CustomerType = 'Person' THEN CONCAT(FirstName,' ', LastName)

        ELSE CompanyName

END AS Nome
```

Isso acima é apenas uma coluna usando condições para retornar valores seguindo uma lógica.

CREATE VIEW, CREATE DROP → Quando quisermos colocar um código sql no pesquisador de objetos colocamos na primeira linha “CREATE VIEW “Nome” AS.
DROP remove.