# Artigo científico sobre banco de dados

Antonio Marcos Siqueira
Guilherme Banzati Viana Ribeiro

30 de Junho de 2022

## Resumo

Artigo cientifico sobre o uso de Views, Triggers, Procedimentos e Joins na linguagem SQL no trabalho, e manipulação, de bancos de dados relacionais.

## 1 Introdução

A linguagem SQL (Struct Query Language, ou Linguagem de Consulta Estruturada) é uma linguagem padrão para a manipulação de dados dentro de um sistema de gerenciamento de banco de dados.
Ela possibilita o armazenamento, organização, atualização e exclusão de informações dentro de um determinado banco de dados, e é uma linguagem declarativa e que não necessita de profundos conhecimentos de programação para que alguém possa começar a manipula-la.
Neste artigo científico, sera explicado as definições, aplicações e exemplos práticos seguintes tópicos contidos na linguagem SQL: Views, Triggers, Procedimentos e Join (junções); além de ser explicados e analisados cenários em que esses tópicos podem, ou não, serem aplicados.

## 2 Views

### 2.1 Definição

Em teoria de banco de dados, uma view é um conjunto resultado de uma consulta armazenada sobre os dados, as linhas e colunas da view são geradas dinamicamente no momento em que é feita uma referência a ela.

### 2.2 Para que serve

Uma view é uma maneira alternativa de observação de dados de uma ou mais tabelas, que compõem uma base de dados. Então ela pode ser considerada como uma tabela virtual (ou uma consulta armazenada).
Sendo assim, as views nos possibilitam mais que simplesmente "visualizar dados", pois elas podem ser implementadas também com algumas aplicações de restrição

### 2.3 Para que deve ser aplicado

As views podem ser usadas em casos como: criar uma restrição entre os usuários e os dados, criar uma restrição entre os usuários e os domínios, associar vários domínios formando uma única entidade e agregar informações, em vez de fornecer maiores detalhes do banco de dados.

### 2.4 Locais onde não deve ser aplicado

Existem alguns locais que as views não devem ser aplicadas, elas escondem uma complexidade da query, podendo enganar o desenvolvedor quanto à performance necessária para acessar determinada informação especifica.
Também pode ser mais complexo quando views usam outras views. Em alguns casos, o usuário pode estar fazendo consultas desnecessárias (sem saber disso) e de forma muito intensiva.
As views também podem ser "mal utilizadas" criando camadas extras, e assim, mais objetos para serem administrados, em alguns casos, isso pode limitar exageradamente o que o usuário pode acessar impedindo certas tarefas.

### 2.5 Exemplos práticos

Um exemplo prático sobre um uso de uma view: abaixo vemos uma view criada para facilitar o acesso de vendas e produtos de um determinado banco de dados.

```
CREATE VIEW vwVendas AS
SELECT * FROM Vendas V
JOIN Clientes C ON V.Cliente_id = C.id_Cliente
JOIN Produtos P ON V.Produto_id = P.id_Produto
GO
```

A view criada no exemplo anterior pode ser executada da seguinte forma

```
SELECT * FROM vwVendas
```

## 3 Triggers

### 3.1 Definição

Os triggers (gatilhos) definem uma estrutura do banco de dados que funciona, como o nome sugere, como uma função que é disparada mediante alguma ação. Geralmente essas ações que disparam os triggers são alterações nas tabelas por meio de operações de inserção, exclusão e atualização de dados (insert, delete e update).
Um gatilho está diretamente relacionado a uma tabela, sempre que uma dessas ações é efetuada sobre determinada tabela, é possível dispará-lo para executar alguma tarefa.

### 3.2 Para que serve

A principal funcionalidade de um trigger é a automatização de tarefas no banco de dados após ocorrer alguma ação.

## 3.3 Para que deve ser aplicado

A aplicação de triggers dependerá muito de como esta sendo desenvolvendo determinada aplicação. O melhor cenário seria quando é necessário tirar algumas funções de uma aplicação e colocá-las no banco de dados, por exemplo.
Um exemplo seria o armazenamento de acessos à determinada aplicação, visto que o trigger dispara-rá uma função que vai registrar os dados do usuário visitante (como por exemplo: IP, data de acesso, entre outros dados) e criará um log de usuário, registrando no banco de dados.

## 3.4 Locais onde não deve ser aplicado

Existem alguns cenários em que se deve analisar a necessidade de utilizar um trigger, ao adota-lo, resultará na quase impossibilidade de migração de banco de dados, visto que os triggers utilizam uma linguagem proprietária.
É consenso também que a utilização de triggers acarreta em uma queda de perfmormance.

## 3.5 Exemplos práticos

Para exemplificar na prática o uso de triggers, será usado como cenário uma certa aplicação financeira que contém um controle de caixa e efetua vendas. Abaixo, serão criadas as tabelas para usar de exemplo.

```
CREATE TABLE CAIXA
(
    DATA            DATETIME,
    SALDO_INICIAL   DECIMAL(10,2),
    SALDO_FINAL     DECIMAL(10,2)
)
GO
INSERT INTO CAIXA
VALUES (CONVERT(DATETIME, CONVERT(VARCHAR, GETDATE(), 103)), 100, 100)
GO
CREATE TABLE VENDAS
(
    DATA    DATETIME,
    CODIGO  INT,
    VALOR   DECIMAL(10,2)
)
GO
```

Pelo código abaixo, sempre que forem registradas ou excluídas vendas, essas operações devem ser automaticamente refletidas na tabela de caixa, aumentando ou reduzindo o saldo.

```
CREATE TRIGGER TGR_VENDAS_AI
ON VENDAS
FOR INSERT
AS
BEGIN
```

```
    DECLARE
    @VALOR  DECIMAL(10,2),
    @DATA   DATETIME
    SELECT @DATA = DATA, @VALOR = VALOR FROM INSERTED
    UPDATE CAIXA SET SALDO_FINAL = SALDO_FINAL + @VALOR
    WHERE DATA = @DATA
END
GO
```

No exemplo acima, o trigger reflete diretamente sobre a tabela de vendas, que reduzirá o saldo final do caixa na data da venda quando uma venda for inserida.
O resultado será mostrado na tabela abaixo:

| **DATA** | **SALDO_INICIAL** | **SALDO_FINAL** |
|---|---|---|
| 2022-06-29 00:00:00.000 | 100.00 | 100.00 |

# 4 Procedimentos

## 4.1 Definição

Procedimentos, basicamente são um conjunto de comandos em SQL que podem ser executados de uma única vez (como uma função).

## 4.2 Para que serve

Os procedimentos servem para armazenar tarefas repetitivas e aceitar parâmetros de entrada para que determinada tarefa seja efetuada de acordo com a necessidade individual do usuário.

## 4.3 Para que deve ser aplicado

Os procedimentos ajudam a reduzir o tráfego na rede, melhorar a performance de um banco de dados, criar tarefas agendadas, diminuir riscos, criar rotinas de processamento, entre outras aplicações.
Procedimentos devem podem aplicados quando temos várias aplicações escritas em diferentes linguagens, ou rodam em plataformas diferentes, porém executam a mesma função.

## 4.4 Locais onde não deve ser aplicado

Os procedimentos, apesar de ser a opção mais rápida em determinadas situações (quando em comparação com um trigger, por exemplo), eles também tiram muito o controle geral do que esta sendo processado pelo sistema, ou seja, isso pode ser uma desvantagem em determinadas situações.

## 4.5 Exemplos práticos

No exemplo abaixo, é executado uma consulta utilizando um filtro por descrição, em uma tabela específica de um determinado banco de dados.

```
USE BancoDados
GO
CREATE PROCEDURE Busca
@CampoBusca VARCHAR (20)
AS
SELECT Codigo, Descrição
FROM NomeTabela
WHERE Descricao = @CampoBusca
```

Para executar esse procedimento, basta declarar "EXECUTE" seguido pelo nome dele, e na frente o valor a ser utilizado como parâmetro.

```
EXECUTE Busca 'Exemplo'
```

Para excluir um procedimento, basta utilizar a cláusula "DROP PROCEDURE" como no exemplo abaixo.

```
DROP PROCEDURE Busca
```

# 5 Joins

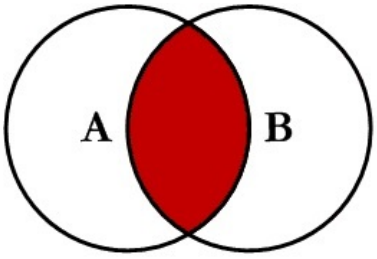
## 5.1 Definição

Joins são métodos de Junção, que tem a finalidade de unir duas tabelas do banco de dados.

## 5.2 Para que serve

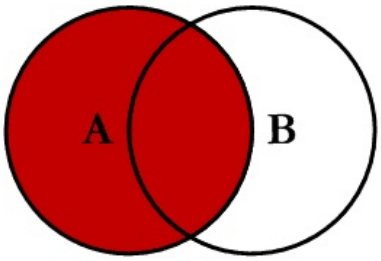
Como o própio nome sugere, o método Join serve para juntar tabelas. Pode-se dizer que o método Join se retrata na Teoria dos Conjuntos, muito conhecida na matemática. Existem vários tipos de comandos "join", e cada um possui um efeito diferente em cima das tabelas selecionadas.

Figura 1 – Inner Join



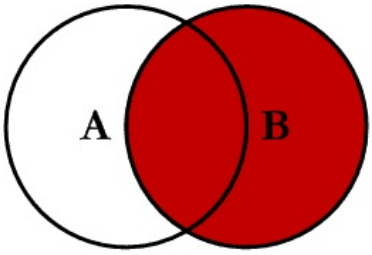

O Inner Join é o método de junção mais conhecido, e como ilustra a Figura 1, retorna os registros que são comuns às duas tabelas.

Figura 2 – Left Join



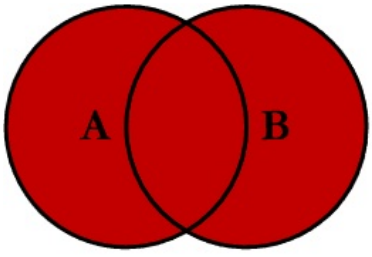

O Left Join, como ilustra a Figura 2, retorna todos os registros que estão na tabela A (mesmo que não estejam na tabela B) e os registros da tabela B que são comuns à tabela A.

Figura 3 – Right Join



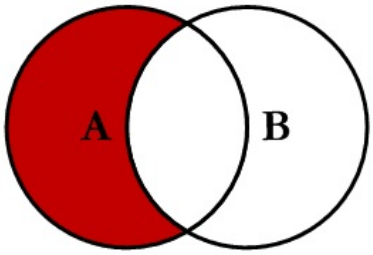

O Right Join, como ilustra a Figura 3, retorna todos os registros que estão na tabela B (mesmo que não estejam na tabela A) e os registros da tabela A que são comuns à tabela B.

Figura 4 – Outer Join



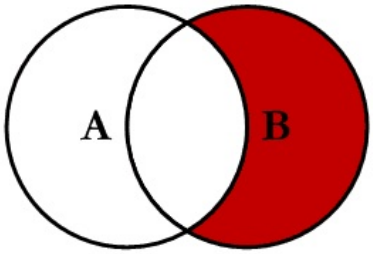

O Outer Join, como ilustra a Figura 4, retorna todos os registros que estão na tabela A e todos os registros da tabela B.

Figura 5 – Left Excluding Join



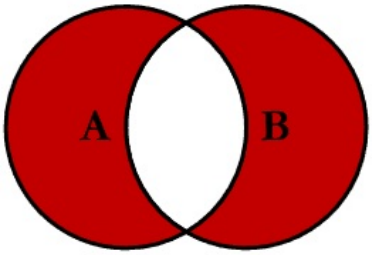

O Left Excluding Join, como ilustra a figura 5, retorna todos os registros que estão na tabela A e que não estejam na tabela B.

Figura 6 – Right Excluding Join




O Right Excluding Join, como ilustra a Figura 6, retorna todos os registros que estão na tabela B e que não estejam na tabela A.

Figura 7 – Outer Excluding Join




O Outer Excluding Join, como ilustra a Figura 7, retorna todos os registros que estão na tabela B, mas que não estejam na tabela A, e todos os registros que estão na tabela A, mas que não estejam na tabela B.

## 5.3 Para que deve ser aplicado

Em resumo, os joins servem para qualquer tipo de junção de tabelas. Há diversos usos para a aplicação de joins, mas para isso, deve se saber utilizar o tipo adequado para um query, por exemplo, pois pode resultar em um retorno diferente do esperado, além de gerar até problemas de performance no banco de dados.

## 5.4 Locais onde não deve ser aplicado

Pode se resultar em um problema de performance a utilização do join onde não é necessário seu uso, pois basicamente ele é um mecanismo automático que pode buscar coisas que você o usuário precisa. Quanto mais há tabelas envolvidas, mais pode prejudicar a performance, até mesmo de forma exponencial. Na prática, quanto mais tabelas, mais combinações são possíveis, mais dados serão gerados, mais fontes a serem consultadas, e se resulta em menos oportunidades de otimizações.

## 5.5 Exemplos práticos

Abaixo será mostrado, em exemplos, o funcionamento e o retorno de cada join, em tabelas previamente montadas, que possuam algum relacionamento para que os dados possam ser "cruzados". Serão criadas duas tabelas contendo uma coluna Nome, que será comum entre elas.

```
CREATE TABLE TabelaA(
  Nome varchar(50) NULL
)
GO
CREATE TABLE TabelaB(
  Nome varchar(50) NULL
```

Em seguida, serão adicionados, nas tabelas recém criadas, alguns dados para que permitam colocar à prova as junções.

```
INSERT INTO TabelaA VALUES('Fernanda')
INSERT INTO TabelaA VALUES('Josefa')
INSERT INTO TabelaA VALUES('Luiz')
INSERT INTO TabelaA VALUES('Fernando')

INSERT INTO TabelaB VALUES('Carlos')
INSERT INTO TabelaB VALUES('Manuel')
INSERT INTO TabelaB VALUES('Luiz')
INSERT INTO TabelaB VALUES('Fernando')
```

- Exemplo de junção com o comando Inner Join:

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
INNER JOIN TabelaB as B
                on a.Nome = b.Nome
```

Resultado:

| **Nome** | **Nome** |
|---|---|
| Luiz | NULL |
| Fernando | NULL |

- Exemplo de junção com o comando Left Join:

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
LEFT JOIN TabelaB as B
  on a.Nome = b.Nome
```

Resultado:

| **Nome** | **Nome** |
|---|---|
| Fernanda | NULL |
| Josefa | NULL |
| Luiz | Luiz |
| Fernando | Fernando |

- Exemplo de junção com o comando Right Join:

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
RIGHT JOIN TabelaB as B
  on a.Nome = b.Nome
```

Resultado:

| **Nome** | **Nome** |
|---|---|
| NULL | Carlos |
| NULL | Manuel |
| Luiz | Luiz |
| Fernando | Fernando |

- Exemplo de junção com o comando Outer Join:

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
FULL OUTER JOIN TabelaB as B
    on a.Nome = b.Nome
```

Resultado:

| **Nome** | **Nome** |
|---|---|
| Fernanda | NULL |
| Josefa | NULL |
| Luiz | Luiz |
| Fernando | Fernando |
| NULL | Carlos |
| NULL | Manuel |

*Observação: nesse código, a palavra reservada OUTER é opcional. Portanto, se a removermos, deixando apenas a expressão FULL JOIN, o resultado será o mesmo.

- Exemplo de junção com o comando Left Excluding Join:

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
LEFT JOIN TabelaB as B
                on a.Nome = b.Nome
WHERE b.Nome is null
```

Resultado:

| **Nome** | **Nome** |
|---|---|
| Fernanda | NULL |
| Josefa | NULL |

- Exemplo de junção com o comando Right Excluding Join:

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
RIGHT JOIN TabelaB as B
    on a.Nome = b.Nome
WHERE a.Nome is null
```

Resultado:

| **Nome** | **Nome** |
|---|---|
| NULL | Carlos |
| NULL | Manuel |

- Exemplo de junção com o comando Outer Excluding Join:

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
FULL OUTER JOIN TabelaB as B
    on a.Nome = b.Nome
WHERE a.Nome is null or b.Nome is null
```

Resultado:

| **Nome** | **Nome** |
|---|---|
| Fernanda | NULL |
| Josefa | NULL |
| NULL | Carlos |
| NULL | Manuel |

## 6 Considerações finais

Como visto nesse artigo científico, as aplicações de views, triggers, procedimentos e joins são recursos importantes a serem implementados em um banco de dados, pois eles simplificam diversas operações a serem realizadas por ele. Assim sendo, se torna mais

simples a codificação de um determinado sistema, definindo uma camada intermediária de controle entre o usuário, o banco de dados físico e o código fonte da aplicação.

## Referências


BEAULIEU, A. *Aprendendo SQL: Dominando os Fundamentos de SQL*. [S.l.]: Novatec Editora, 2019.

IBM. Create procedure (procedimentos sql). Disponível em: <https://www.ibm.com/docs/pt-br/netcoolomnibus/8.1?topic=reference-create-procedure-command-sql-procedures>. Acesso em: 25 jun. 2022.

MARCHIORI, L. Trigger: o que é, para que serve e como usar em sql? Disponível em: <https://blog.betrybe.com/tecnologia/o-que-e-como-usar/>. Acesso em: 25 jun. 2022.

MEDIA, D. Conceitos e criação de views no sql server. Disponível em: <https://www.devmedia.com.br/conceitos-e-criacao-de-views-no-sql-server/22390>. Acesso em: 24 jun. 2022.

MEDIA, D. Guia completo de sql. Disponível em: <https://www.devmedia.com.br/guia/guia-completo-de-sql/38314>. Acesso em: 24 jun. 2022.

MEDIA, D. Inner, cross, left, rigth e full joins. Disponível em: <https://www.devmedia.com.br/inner-cross-left-rigth-e-full-joins/21016>. Acesso em: 26 jun. 2022.

MEDIA, D. Introdução aos stored procedures no sql server. Disponível em: <https://www.devmedia.com.br/introducao-aos-stored-procedures-no-sql-server/7904>. Acesso em: 25 jun. 2022.

MEDIA, D. Sql join: Entenda como funciona o retorno dos dados. Disponível em: <https://www.devmedia.com.br/sql-join-entenda-como-funciona-o-retorno-dos-dados/31006>. Acesso em: 26 jun. 2022.

MEDIA, D. Triggers no sql server: teoria e prática aplicada em uma situação real. Disponível em: <https://www.devmedia.com.br/triggers-no-sql-server-teoria-e-pratica-aplicada-em-uma-situacao-real/28194>. Acesso em: 25 jun. 2022.

MICROSOFT. Create view (transact-sql). Disponível em: <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql>. Acesso em: 25 jun. 2022.

SILVEIRA, P. O que é sql? Disponível em: <https://www.alura.com.br/artigos/o-que-e-sql>. Acesso em: 24 jun. 2022.

TRYBE. Sql join(inner, left, right e full) combinando tabelas! Disponível em: <https://blog.betrybe.com/sql/sql-join/>. Acesso em: 28 jun. 2022.