

Escola de Artes, Ciências e Humanidades – Universidade de São Paulo

Parte 2: Implementação de Regras de Negócios

Prof. Fátima Nunes

Matheus Percoraro de Carvalho Santos	Nº USP: 11917271
Ryan Brenno Ramos	Nº USP: 11215772
Sungwon Yoon	Nº USP: 9822261
Wendel Fernandes de Lana	Nº USP: 11796722

Artefato a)

As regras de negócio de negócio levadas em consideração foram as seguintes:

1. A fim de limitar abusos, os cupons de promoção só podem ser usados 5 vezes. Após utilizado pela 5ª vez, o cupom automaticamente expira.
2. O estabelecimento é de médio porte e, por isso, deve manter um controle de teto de salários para determinados tipos de funcionários. Em especial, a verba mensal alocada para o pagamento de garçons efetivos é de R\$ 50000,00.

Regra 1

Pode ser modelada a partir de um gatilho. Toda vez que um novo registro for inserido na tabela Pedido (evento), caso o cupom utilizado por aquele pedido tenha sido utilizado 4 vezes anteriormente (condição), a data de validade daquele cupom é atualizada para a data corrente do sistema.

- Solução em SQL padrão:

```
CREATE TRIGGER cupom_limite AFTER INSERT ON pedido
REFERENCING NEW ROW AS nrow
FOR EACH ROW
BEGIN ATOMIC
UPDATE cupom SET vencimento = CURRENT_TIMESTAMP
WHERE codigo = (SELECT p.fk_Cupom_codigo FROM pedido p
WHERE p.fk_Cupom_codigo = nrow.fk_Cupom_codigo
GROUP BY p.fk_Cupom_codigo HAVING COUNT(p.fk_Cupom_codigo) =5)
END
```

- Solução em código implementado no SQL Server:

```
CREATE TRIGGER cupom_limite
ON pedido
AFTER INSERT
AS
BEGIN
    UPDATE cupom SET vencimento = GETDATE()
    WHERE codigo = (SELECT p.fk_Cupom_codigo FROM Pedido p, INSERTED i
WHERE p.fk_cupom_codigo = i.fk_cupom_codigo GROUP BY p.fk_cupom_codigo
HAVING COUNT(p.fk_cupom_codigo) = 5)
END
```

Não houve nenhuma dificuldade encontrada na implementação da solução no SGBD utilizado, fazendo-se necessário, para aplicar a solução em SQL padrão, apenas procurar na documentação como alguns comandos e cláusulas são utilizadas no SQL Server.

- Casos de teste:

Adotamos que as tabelas Pedido e Cupom apresentam os seguintes registros:

```
SELECT * FROM Pedido
SELECT * FROM Cupom
```

	id	data	fk_Cupom_codigo	forma_pagamento	status
1	1	2022-05-20 15:29:43.000	cup1	dinheiro	1
2	2	2022-05-21 15:29:43.000	cup1	credito	1
3	3	2022-05-22 15:29:43.000	cup1	credito	1
4	4	2022-05-23 15:29:43.000	cup2	dinheiro	1
5	5	2022-05-24 15:29:43.000	cup1	credito	1

	codigo	porcentagem	vencimento
1	cup1	10	2022-08-15 23:59:59.000
2	cup 2	20	2023-01-13 23:59:59.000

Após inserirmos um pedido que utiliza cup2, não há alteração nenhuma em nenhum outro registro além do registro inserido:

```
INSERT INTO Pedido VALUES (6, GETDATE(), 'cup2', 'dinheiro', 0);
```

Messages
(0 rows affected)
(1 row affected)
Completion time: 2022-05-30T20:59:18.9244851-03:00

Após a inserção de um pedido que utiliza cup1, sendo essa a quinta utilização de tal cupom, o registro do cupom é atualizado e a sua data de validade alterada para a do momento corrente:

```
INSERT INTO Pedido VALUES (7, GETDATE(), 'cup1', 'debito', 0);
```

Messages
(1 row affected)
(1 row affected)
Completion time: 2022-05-30T21:00:54.5075287-03:00

SELECT * FROM Cupom

100 %

Results Messages

	codigo	porcentagem	vencimento
1	cup1	10	2022-05-30 21:07:23.157
2	cup2	20	2023-01-13 23:59:59.000

Por fim, caso seja feita mais uma inserção em Pedido utilizando cup1, a data de vencimento não se alterará novamente, pois o valor seria reatualizado e passaria a valer para o atual pedido:

INSERT INTO Pedido VALUES (8, GETDATE(), 'cup1', 'dinheiro', 0);

100 %

Messages

(0 rows affected)

(1 row affected)

Completion time: 2022-05-30T21:10:59.5494860-03:00

Regra 2

Pode ser modelada a partir de uma asserção. Essa asserção deve recuperar a soma de todos os salários de garçons efetivos e garantir que o valor total não passe de 50000.

- Solução em SQL padrão:

```
CREATE ASSERTION teto_gastos
CHECK (NOT EXISTS (SELECT SUM(salario) FROM Funcionario WHERE cargo = 'garcom' AND tipo = 'efetivo'
AND data_egresso IS NULL
HAVING SUM(salario) >= 50000));
```

Solução em código implementado no SQL Server:

```
CREATE TRIGGER teto_gastos
ON funcionario
INSTEAD OF INSERT, UPDATE
AS
BEGIN
    IF (EXISTS(SELECT 1 FROM inserted i WHERE i.tipo = 'efetivo' AND i.cargo = 'garcom' AND i.salario +
(SELECT SUM(f.salario)
FROM Funcionario f WHERE f.cargo = 'garcom' AND f.tipo = 'efetivo'
AND f.data_egresso IS NULL) >= 50000))
BEGIN
    RAISERROR ('Teto de gastos atingido.',10,1)
    ROLLBACK TRANSACTION
END
ELSE
BEGIN
```

```

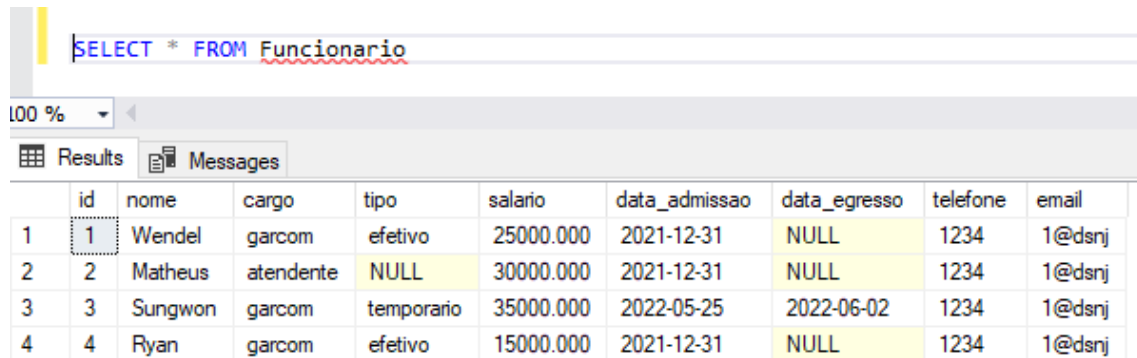
INSERT INTO Funcionario
SELECT * FROM INSERTED
END
END

```

O SQL Server, assim como outros SGBDs, não apresenta suporte para asserções. Sendo assim, tivemos que modelar a regra de negócio para funcionar como um gatilho, garantindo que o registro não seja inserido na tabela caso quebre a condição.

- Casos de teste:

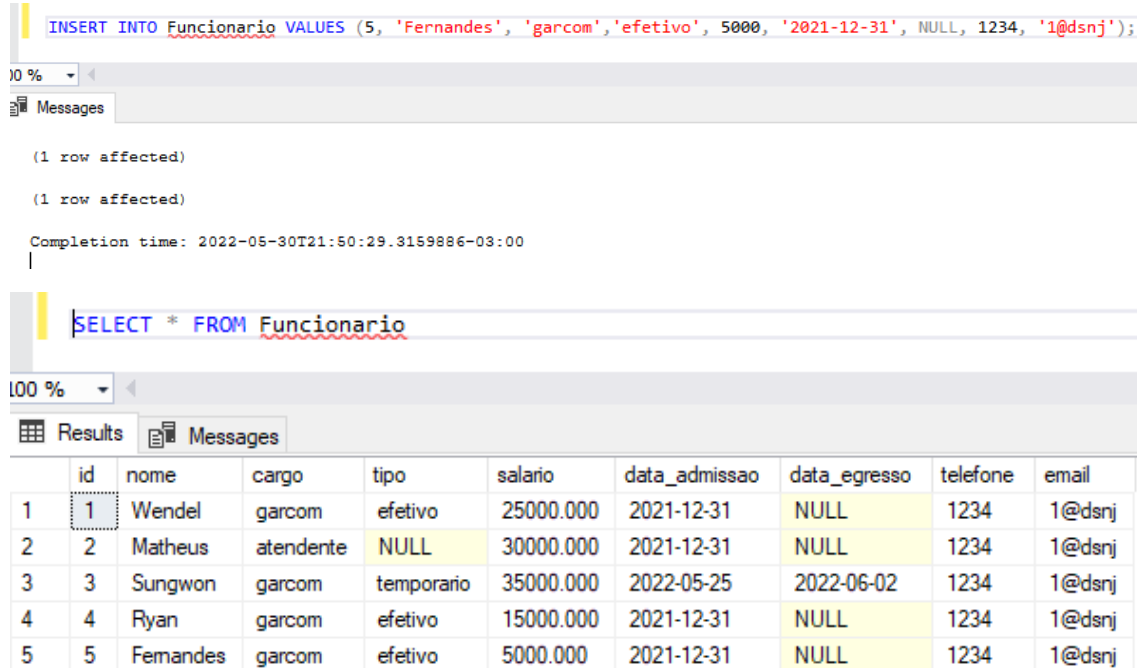
Adotamos que a tabela Funcionario apresenta os seguintes registros:



The screenshot shows a SQL query window with the command `SELECT * FROM Funcionario`. Below the query, the 'Results' tab displays a table with 10 columns: id, nome, cargo, tipo, salario, data_admissao, data_egresso, telefone, and email. There are 4 rows of data.

	id	nome	cargo	tipo	salario	data_admissao	data_egresso	telefone	email
1	1	Wendel	garcom	efetivo	25000.000	2021-12-31	NULL	1234	1@dsnj
2	2	Matheus	atendente	NULL	30000.000	2021-12-31	NULL	1234	1@dsnj
3	3	Sungwon	garcom	temporario	35000.000	2022-05-25	2022-06-02	1234	1@dsnj
4	4	Ryan	garcom	efetivo	15000.000	2021-12-31	NULL	1234	1@dsnj

Após inserirmos um garçom efetivo com salário de 5000, o teto máximo não será atingido, portanto, a inserção ocorrerá normalmente:



The screenshot shows a SQL query window with the command `INSERT INTO Funcionario VALUES (5, 'Fernandes', 'garcom', 'efetivo', 5000, '2021-12-31', NULL, 1234, '1@dsnj');`. Below the query, the 'Messages' tab shows two messages: `(1 row affected)` and `(1 row affected)`. The 'Results' tab shows the updated table with 5 rows.

Completion time: 2022-05-30T21:50:29.3159886-03:00

	id	nome	cargo	tipo	salario	data_admissao	data_egresso	telefone	email
1	1	Wendel	garcom	efetivo	25000.000	2021-12-31	NULL	1234	1@dsnj
2	2	Matheus	atendente	NULL	30000.000	2021-12-31	NULL	1234	1@dsnj
3	3	Sungwon	garcom	temporario	35000.000	2022-05-25	2022-06-02	1234	1@dsnj
4	4	Ryan	garcom	efetivo	15000.000	2021-12-31	NULL	1234	1@dsnj
5	5	Fernandes	garcom	efetivo	5000.000	2021-12-31	NULL	1234	1@dsnj

Após tentar inserir um garçom efetivo que finalmente atingiria o teto de gastos, a condição do gatilho impede a inserção de ocorrer:

```
INSERT INTO Funcionario VALUES (6, 'Ramos', 'garcom', 'efetivo', 15000, '2021-12-31', NULL, 1234, '1@dsnj');
```

100 %

Messages

Teto de gastos para garcom atingido.
Msg 3609, Level 16, State 1, Line 2
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2022-05-30T21:52:28.5914803-03:00

Outros funcionários que não garçons efetivos podem ser inseridos normalmente (caberia a outras regras e restrições, por sua vez, limitarem o teto de gasto para outras categorias de funcionários):

```
INSERT INTO Funcionario VALUES (6, 'Lana', 'gerente', NULL, 60000, '2021-12-31', NULL, 1234, '1@dsnj');
```

100 %

Messages

(1 row affected)
(1 row affected)

Completion time: 2022-05-30T22:07:21.6749094-03:00

```
SELECT * FROM funcionario
```

100 %

Results Messages

	id	nome	cargo	tipo	salario	data_admissao	data_egresso	telefone	email
1	1	Wendel	garcom	efetivo	25000.000	2021-12-31	NULL	1234	1@dsnj
2	2	Matheus	atendente	NULL	30000.000	2021-12-31	NULL	1234	1@dsnj
3	3	Sungwon	garcom	temporario	35000.000	2022-05-25	2022-06-02	1234	1@dsnj
4	4	Ryan	garcom	efetivo	15000.000	2021-12-31	NULL	1234	1@dsnj
5	5	Fernandes	garcom	efetivo	5000.000	2021-12-31	NULL	1234	1@dsnj
6	6	Lana	gerente	NULL	60000.000	2021-12-31	NULL	1234	1@dsnj

Artefato b)

É de interesse do restaurante, por questões de segurança, limitar as informações que os funcionários que têm acesso ao sistema podem consultar. O restaurante possui um funcionário responsável pelo atendimento de pedidos de delivery. Uma das possibilidades de exibição de informações para esse usuário é uma tabela que mostre os endereços já utilizados pelos clientes para que ele possa inserir no cadastro do novo pedido.

- Solução em SQL padrão e SQL Server:

```
CREATE VIEW historico_delivery AS  
SELECT p.data, c.nome, d.endereco, d.numero, d.complemento, d.cep, c.telefone  
FROM cliente c, delivery d, pedido p  
WHERE d.fk_Cliente_id = c.id AND p.id = d.fk_Pedido_id;
```

A visão fornece informações do pedido realizado da tabela Delivery, a data do pedido da tabela Pedido e o número de telefone da tabela Cliente. Com esses dados, uma consulta pode ser feita sobre a visão para recuperar os endereços utilizados por um cliente por meio da pesquisa do telefone, possibilitando também a ordenação pela data para facilitar a visualização.

No caso da materialização dessa visão, a consulta sobre ela se tornaria mais rápida, mas o overhead de manutenção e armazenamento, dado que a visão recupera todos os pedidos já feitos, tornaria insatisfatória a relação custo/benefício.

A atualização de dados sobre a visão incorreria em inconsistência de dados nas tabelas base. E a visão não possui intuito de propiciar a atualização dos dados por quem a utiliza.

Artefato c)

Faz-se útil checar se os itens cadastrados no sistema vendidos pelos fornecedores foram entregues com uma data de validade posterior à data de compra. Os pagamentos aos fornecedores ocorrem somente ao fim do mês, sendo necessário também a realização de uma consulta para recuperar a soma total do valor das compras daquele mês de um determinado fornecedor. Tais consultas podem ser otimizadas com a criação de uma visão que una as tabelas Fornecedor, Compra, Item e Produto. Ambas as consultas, então, poderão ser realizadas sobre a visão criada.

- Solução em SQL padrão da visão e das consultas:

```
CREATE VIEW fornecedores_produtos AS
SELECT p.nome, i.quantidade, i.data_validade, c.data data_compra, c.preco, f.cnpj, f.nome_fantasia,
f.nome_contato, f.contato_telefone
FROM Fornecedor f, Compra c, Item i, Produto p
WHERE f.id = c.fk_Fornecedor_id AND c.id = i.fk_Compra_id AND i.fk_Produto_id = p.id
```

```
SELECT nome, data_validade, data_compra, nome_fantasia, nome_contato, contato_telefone
FROM fornecedores_produtos
WHERE data_validade <= data_compra
```

```
SELECT cnpj, nome_fantasia, SUM(preco), nome_contato, contato_telefone
FROM fornecedores_produtos
WHERE DATE_TRUNC(data_compra, 'MONTH') = DATE_TRUNC(CURRENT_DATE, 'MONTH')
GROUP BY cnpj, nome_fantasia, nome_contato, contato_telefone
```

- Solução em código implementado no SQL Server:

```
CREATE VIEW fornecedores_produtos AS
SELECT p.nome, i.quantidade, i.data_validade, c.data AS data_compra, c.preco, f.cnpj, f.nome_fantasia,
f.nome_contato, f.contato_telefone
FROM Fornecedor AS f
JOIN Compra AS c ON f.id = c.fk_Fornecedor_id
JOIN Item AS i ON c.id = i.fk_Compra_id
JOIN Produto AS p ON i.fk_Produto_id = p.id
```

```
SELECT nome, data_validade, data_compra, nome_fantasia, nome_contato, contato_telefone
FROM fornecedores_produtos
WHERE DATEDIFF(day, data_validade, data_compra) >= 0
```

```
SELECT cnpj, nome_fantasia, SUM(preco), nome_contato, contato_telefone
FROM fornecedores_produtos
WHERE MONTH(data_compra) = MONTH(GETDATE())
GROUP BY cnpj, nome_fantasia, nome_contato, contato_telefone
```

Decidimos que a visão deve retornar todos os itens cadastrados no sistema com a sua data de validade e a quantidade, assim como o nome do produto associado ao item para fácil identificação, a data da compra para comparação com a validade, o preço da compra para soma posterior das compras realizadas no mês de um fornecedor, e as informações do fornecedor para contato facilitado.

Para que as consultas de fato sejam otimizadas, faz-se necessário a materialização dessa visão. Teremos overhead de custo de manutenção e armazenamento, porém, como a visão estará materializada no disco, não haverá necessidade do join entre as tabelas, aumentando a velocidade da recuperação dos dados.

A atualização de dados sobre a visão incorreria em inconsistência de dados nas tabelas base. E a visão não possui intuito de propiciar a atualização dos dados por quem a utiliza.