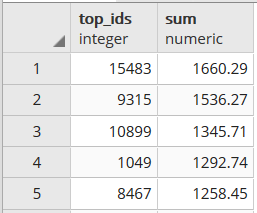
**Operação 1 – Problemas de Concorrência**

O problema de concorrência identificado na operação 2 ocorre quando:

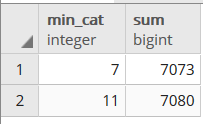
1. A operação é chamada múltiplas vezes concorrentemente para clientes que cumpram os requisitos, que comprem itens da categoria menos vendida no instante da compra e que a quantidade dos itens seja suficiente para que essa categoria deixe de ser a menos vendida. Se dois clientes executam uma compra simultaneamente nas condições indicadas, ambos irão ter desconto nos produtos apesar da categoria já não ser a menos vendida (um comportamento que não desejável). O comportamento desejável consiste em não atribuir desconto ao cliente que compre o produto após a categoria inicial deixar de ser a menos vendida.
2. Um cliente que reúna todas as condições para obter o desconto na próxima compra exceto não pertencer ao top 1000 de melhores clientes, ou seja, caso este cliente execute duas compras concorrentemente em que a primeira o coloca na lista dos 1000 melhores clientes, na 2ª compra ele terá de ter direito ao desconto sendo um comportamento que não ocorre.

**Problema 1:**



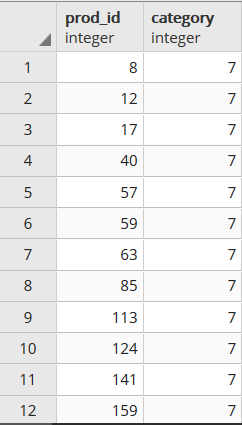
Escolhemos alguns dos clientes que são do top 1000 dos clientes com mais de 50 anos

Figura 1 -Alguns top clientes



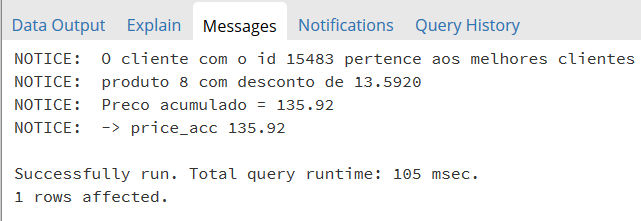
A categoria menos vendida é a 7, com uma diferença de 7 unidades para a segunda categoria menos vendida.

Figura 2 - Categorias menos vendida



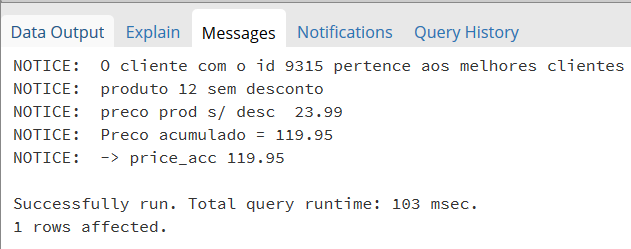
Estes são os produtos que neste momento, caso sejam comprados por um top cliente, têm desconto.

Figura 3 - Produtos da Categoria 7



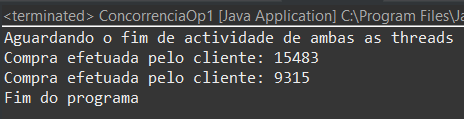
Ao efetuarmos uma compra, o cliente obtém desconto no produto. Mas a compra foi superior a 7 logo a categoria 11 passa a ser a menos vendida.

Figura 4 - Primeira compra



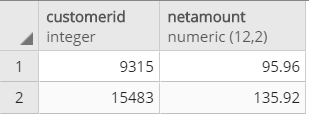
Ao efetuarmos outra compra, o cliente já não obtém desconto no produto, pois já não pertence à categoria menos vendida. (Comportamento esperado)

Figura 5 - Segunda compra



Se utilizarmos a classe Java conseguimos efetuar as duas compras concorrentemente

Figura 6 - Compras concorrentes



Tal como esperado o segundo cliente obteve desconto apesar do produto já não ser da categoria menos vendida. (Comportamento não desejável)

Figura 7 – Preço das compras concorrentes

**Solução?**

Para resolver este problema de concorrência utilizámos um Lock na tabela *inventory* do tipo “Share Row Exclusive” antes da obtenção da melhor categoria, uma vez que a primeira thread que chega escolhe a categoria menos vendida e procede com o resto da transação e a segunda thread tem de ler a categoria mas como foi alterada pela primeira thread, a segunda thread necessita de esperar que a primeira termine daí escolhermos um mecanismo exclusive. Por outro lado, atualizamos também na tabela *inventory* os valores das quantidades e das vendas e para isso necessitamos de um mecanismo do tipo *Share* e ao nível da linha.

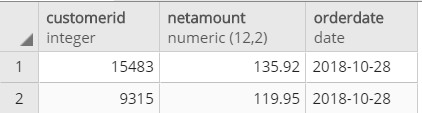
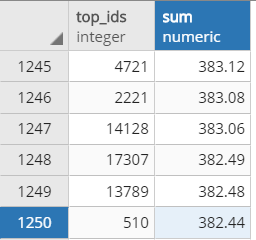


Figura - Aplicação correta do desconto

**Problema 2:**

****

Fomos procurar um cliente que não estando no top 1000 de melhores clientes está bastante perto. Neste caso o cliente com id = 510. Depois de ter feito as compras no mês anterior este cliente faz a primeira compra e verificamos que entrou no top de 1000 melhores clientes (Figura 11). Depois fazemos uma 2ª compra e verificamos se é aplicado desconto ou não.

Figura - Encontramos um cliente que ainda não está no top 1000

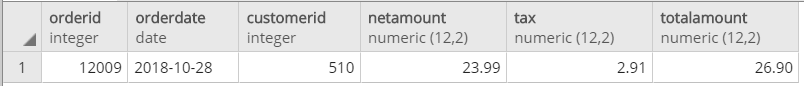
****

Figura - O cliente faz uma primeira compra

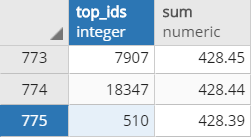


Figura - O cliente entra no top 1000

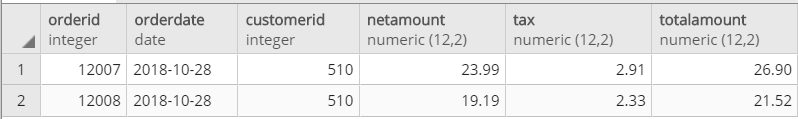


Figura - É feita a 2ª compra e recebe o desconto na mesma (Comportamento desejável)

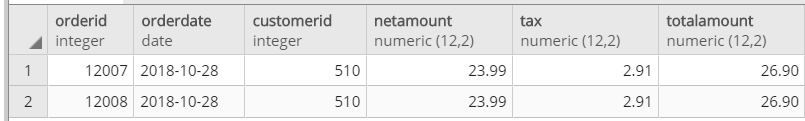
****

Figura - Compras feitas concorrentemente sem mecanismos de lock (Comportamento não desejável)

**Solução:**

Para resolvermos este problema utilizámos um mecanismo de lock do tipo “Share Row Exclusive” na tabela customers antes do select que vai buscar a lista de melhores clientes com as condições da promoção uma vez que necessitamos que a 2ª transação espere que a primeira atualize a lista dos melhores clientes e para isso necessitamos de lock de leitura e escrita e exclusiva para que a outra thread espere pelo desenrolar da primeira.

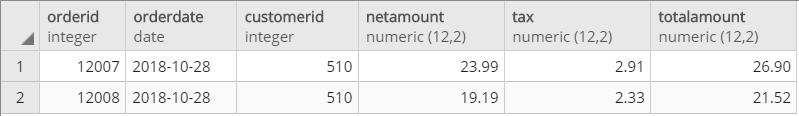
****

Figura - Compras concorrentes com lock (Comportamento desejável)

**Operação 2 – Problemas de Concorrência**

O problema de concorrência identificado na operação 2 ocorre quando:

1. Um cliente faz “duas primeiras compras” ao mesmo tempo, i.e., faz simultaneamente duas compras não tendo nenhuma compra feita até ao momento.
2. Quando dois clientes fazem a primeira compra de um certo produto ao mesmo tempo e um deles esgota o stock.

**Problema 1:**

No exemplo seguinte o cliente com ID 1 não tem nenhuma compra efetuado, o produto com ID 10000 custa 10,99 euros e tem direito a desconto quando comprado na primeira compra do cliente (custa menos de 15 euros e é um dos produtos com maior stock).

Assim que o cliente com o ID 1 faz uma compra de uma unidade do produto 10000 é aplicado um desconto de 15% à compra do produto.

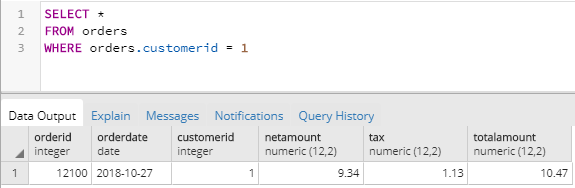




Figura 15 -preço da primeira compra

Depois de realizar uma compra igual à anterior o valor da compra é superior pois não é aplicado o desconto porque já não é a primeira compra do cliente.

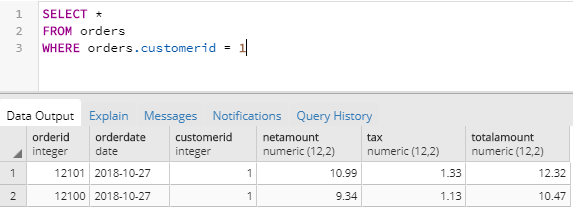


Figura 16 – Segunda compra do cliente

Depois deste exemplo voltamos ao estado anterior da base de dados, ou seja, cliente com ID 1 não tem compras e o stock e sales do produto 10000 voltaram ao valor antigo. Desta vez foram feitas duas compras concorrentes, iguais às anteriores recorrendo às funções feitas em Java:

Como se pode observar ao efetuar duas compras concorrentemente o desconto é aplicado às duas compras, ou seja, as duas compras estão a ser feitas como se fossem ambas as primeiras compras do cliente.

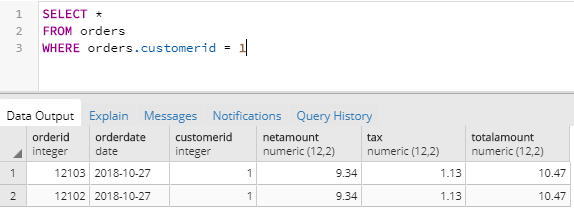
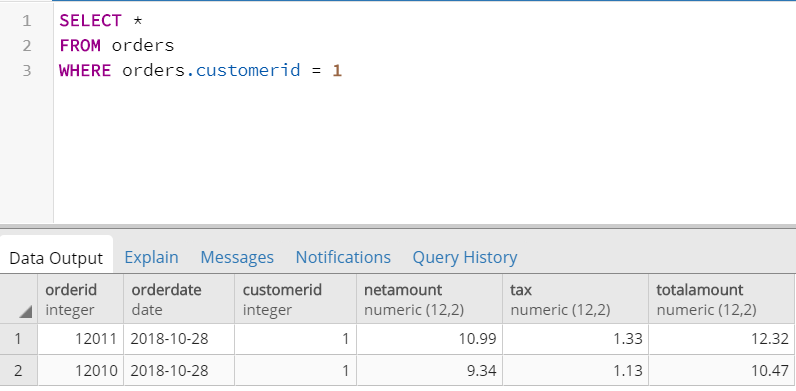


Figura 17 – Preço das duas compras concorrentes

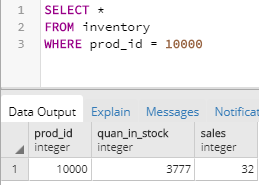
**Solução?**

Decidimos utilizar um Lock do tipo “Share Row Exclusive” na tabela Cust\_hist colocado antes da verificação da primeira compra. Isto permite que a primeira thread que chegue bloqueie os dados e só os liberte depois da transação ser *commited* assim para quando a 2ª compra chega já verifica que não é a primeira compra e não aplica o desconto transitando a base de dados para um estado correto. Como podemos comprovar no print em baixo, não foi aplicado o desconto duas vezes.

****

**Problema 2:**

Para realizar este teste usamos os clientes com os ID’s 1 e 4, que não têm nenhuma compra feita.



Como se pode observar existem 3777 unidades em stock do produto com o ID 10000. Iremos fazer duas compras em que o total de unidades compradas serão superiores às existentes.

Figura 18 - Quantidade de stock do produto com ID 10000

Se fizermos duas compras não concorrentes:

* Cliente 1 faz uma compra de 3750 unidades do produto 10000
* Cliente 4 faz uma compra de 50 unidades do produto 10000

O cliente 1 foi o primeiro a efetuar a compra e conseguiu fazer a compra, deixando apenas 27 unidades do produto.

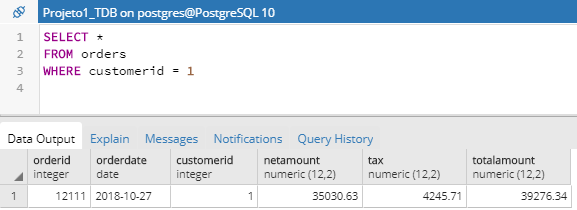
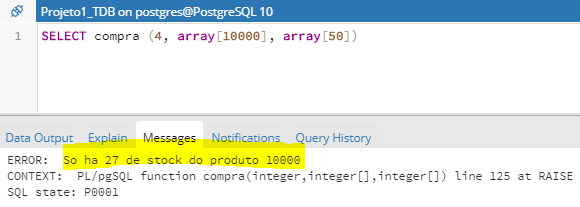


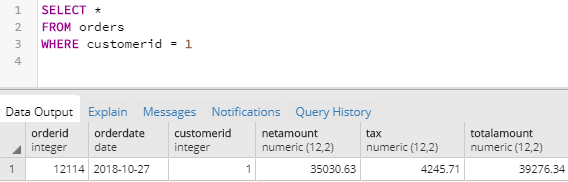
Figura 19 - Valor da compra do cliente 1



Como o cliente 4 fez a compra depois e a quantidade que pretendia era superior ao stock existe a compra não foi efetuada.

Figura 20 - Tentativa de compra do cliente 4

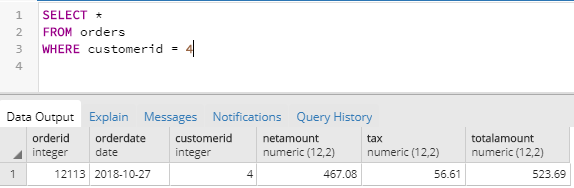
E se os dois clientes fizeram as compras que foram demonstradas ao mesmo tempo? Para fazer esse teste voltamos ao estado anterior da base de dados, ou seja, os clientes 1 e 4 não têm qualquer compra e o stock do produto 10000 é 3777.



Podemos ver que o cliente 1 conseguiu fazer a compra e teve o desconto aplicado.

(10,99 \* 0,85) \* 3750 = 35030,63 euros

Figura 21 - Compra do cliente 1

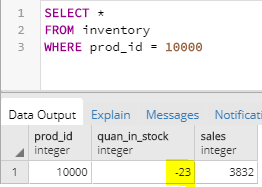


Podemos ver que o cliente também conseguiu fazer a compra e teve o desconto aplicado.

(10,99 \* 0,85) \* 50 = 467,08 euros

Figura 22 - Compra do cliente 4

Se os dois clientes conseguiram fazer as compras e se o total dos produtos comprados excede o stock existe, o que aconteceu ao stock?



Mesmo controlando com um IF\* que a quantidade dos produtos comprados tem de ser igual ou inferior ao stock existente, as duas compras foram efetuadas e o stock ficou negativo. (Dirty Read)

Figura 23 - Stock do produto 10000 depois de duas compras concorrentes

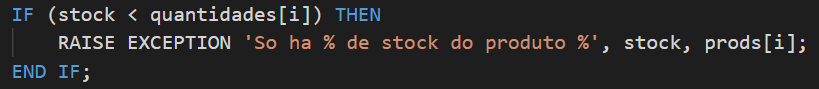
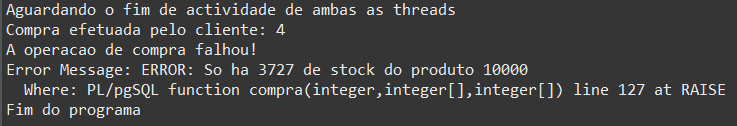


Figura 24 - \* IF colocado na função da compra para controlar que um cliente não compra mais produtos do que a quantidade que está em stock.

**Solução?**

A solução que utilizámos para resolver o primeiro problema de concorrência desta operação também resolve este segundo problema de concorrência uma vez que o lock do tipo “SHARE ROW EXCLUSIVE” na verificação da primeira compra permite que a primeira thread que chegue, apanhar os dados e obter um lock exclusivo. Assim a 2ª thread não consegue começar a sua operação enquanto a 1ª thread não acabar a sua transação.



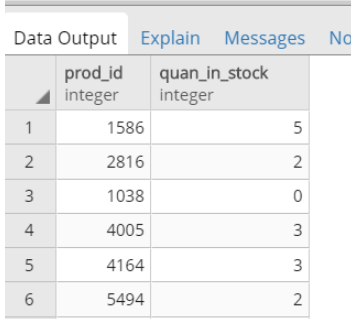
Como podemos observar no output da classe Java, neste exemplo o cliente que conseguiu fazer a compra foi o cliente 4 comprando as 50 unidades do produto 10000 e quando a primeira Thread chega posteriormente já não consegue comprar pois já não há stock suficiente e a sua compra é cancelada. Este é o comportamento desejado.

**Operação 3 – Problemas de Concorrência**

O problema de concorrência identificado na operação 3 ocorre quando:

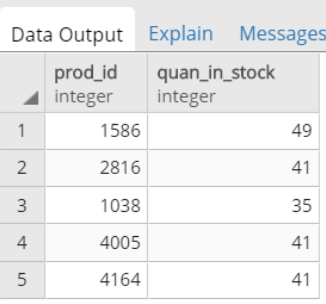
- A operação é chamada múltiplas vezes concorrentemente. Isto acontece, pois, todas as transações obtém a mesma lista de produtos com menos de 20% de stock e fazem a reposição do stock de tantas vezes quantas a operação for chamada para os mesmos produtos (este comportamento não é desejável).

Este resultado difere caso a operação seja chamada múltiplas vezes, mas individualmente (comportamento esperado). Em baixo apresentamos alguns prints dos dois comportamentos.



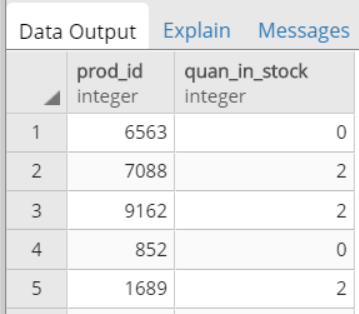
Começamos por observar alguns dos produtos que devem sofrer alteração do stock após chamar a operação.

Figura 25 -produtos com menos de 20% de stock



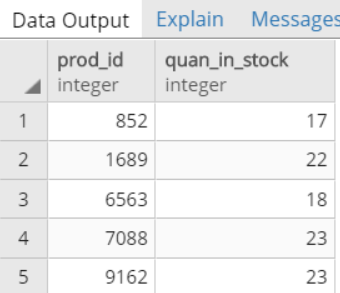
Após realizar a operação de reposição de stock podemos observar que o stock destes produtos foi reposto.

Figura 26 - Stock dos produtos reposto



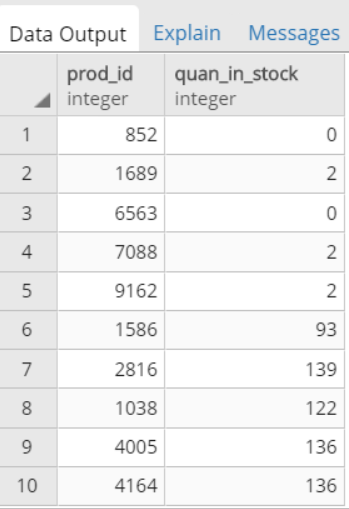
Observamos os novos produtos que devem sofrer alteração de stock quando chamarmos novamente a operação.

Figura 27 - Outros produtos com pouco stock



Tal como esperado o resultado de chamar a operação novamente foi incrementar o stock dos produtos corretamente (Comportamento esperado).

Figura 28 - Stock dos produtos reposto



O mesmo não acontece quando fazemos operações concorrentes. Ambas as threads repuseram stock nos mesmos produtos. (Comportamento não desejável)

Figura 29 - Stock dos produtos com operações concorrentes

**Solução?**

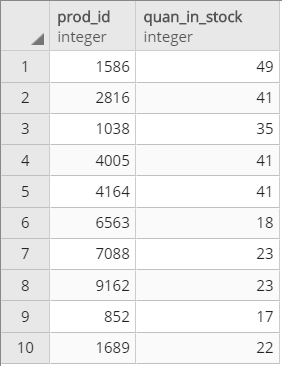
Uma vez que este problema de concorrência tem o seu foco na leitura dos produtos com menos de 20% de stock, para resolver o dito cujo utilizámos um Lock do tipo “Shared Row Exclusive”. Este modo permite-nos fazer com que a primeira Thread que adquira o lock não deixe que existam mais locks sobre os mesmos dados, isto leva a que apenas a primeira Thread apanhe os dados e os utilize e só depois quando terminar a transação é que a outra Thread tem acesso. Os modos de nível inferior não possuem esta funcionalidade de proteger os locks em operações de leitura (SELECT). Assim com um acesso em série controlado conseguimos aplicar corretamente o stock.

Figura - Aplicação correta dos stocks em operações concorrentes