**Anomalia 1:**

Se forem feitas duas encomendas em simultâneo, para o mesmo produto, podem haver problemas na atualização do inventário, i. e., não descontar o número correto de vezes, aceitando assim encomendas que excedam a quantidade em stock.

Na classe **ConcurrencyAnomaly1.java** é ilustrado um exemplo teste desta anomalia. Consideram-se duas threads, T1 e T2, com encomendas de um cliente com customer\_id=1, da quantidade 3 para o produto com prod\_id=2.

Ou seja, serão criadas duas threads da função create\_order(customer\_id, ordered\_products, quantity\_products, dtax) com

customer\_id=1

ordered\_products= {2}

quantity\_products= {3}

dtax=0.23

A thread 2 não é bem sucedida porque está a ser criada uma order com order\_id já existente, conforme se pode ver no output da execução do código Java:

Creating instance 1

Creating instance 2

Starting thread 1

Starting thread 2

Calling now 2

Calling now 1

The end of thread 1

Order of thread 2 was not successful.

ERROR: duplicate key value violates unique constraint "orders\_pkey"

Detail: Key (orderid)=(12001) already exists.

Where: SQL statement "INSERT INTO orders (orderid, customerid, orderdate, netamount, tax, totalamount)

VALUES (order\_id, customer\_id, orderdate, 0, 0, 0)"

PL/pgSQL function generate\_empty\_order(integer,date) line 6 at SQL statement

SQL statement "SELECT (SELECT generate\_empty\_order(customer\_id, order\_date))"

PL/pgSQL function create\_order(integer,integer[],integer[],numeric) line 15 at assignment

The end of thread 2

quan\_in\_stock of prod\_id 2= 115 and sales= 22

The end

Não se conseguiu assim observar o problema de concorrência antecipado e que poderia ser corrigido da seguinte forma:

T1: S LOCK inventory

T1: Lê inventory T2: S LOCK inventory

T1: inventory é atualizado T2: aguarda

T1: UNLOCK inventory T2: aguarda

T2: lê inventory

T2: inventory é atualizado

T2: UNLOCK inventory

**Operação 1**

A operação 1 não tem nenhum problema de concorrência com outra das operações definidas.  
Por outro lado, existe uma série de potênciais problemas:  
-Enquanto a operação 1 correr, a linha da tabela orders que inclui o pedido não pode ser alterada, nem lida, pois vais ser modificada (colocação de um X-LOCK nesta linha)  
-Para esta operação funcionar, ela vai ler a linhas da orderline com o orderline introduzido. Estas não podem ser alteradas enquanto a operação lê (S-LOCK nestas linhas)  
-Os produtos a que esta operação se refere não podem mudar as suas propriedades (nomeadamente o preço e o tipo). (colocaç  
ao de um S-LOCK enquanto a operação lê as tabelas dos produtos.) op1

**Operação 3**

A operação 3 é constituida por duas partes:

-Uma primeira onde lê a tabela inventory

-Uma segunda onde adiciona tuplos na tabela reorder

A terceira operação entra em conflito com a create\_orders, uma vez que a create\_orders actualiza a tabela inventory, e a operação lê a tabela, o que pode levar a que um produto fique abaixo da quantidade minima de stock ao mesmo tempo que a função lê a tabela, e não seja capaz de se aperceber desta mudança de stock:

EXPLAIN ANALYSE SELECT auto\_reorder\_with\_locks('2018-11-01','2018-11-01');

SELECT create\_order (20001,'{10001}','{5000}',1);

(supondo que o artigo prod\_id = 10001 fica abaixo da quantidade minima com a compra das unidades)

Ao executar estas duas queries de forma concorrênte no java, o create\_order efetua com sucesso a compra (como esperado).

Simultaneamente, a create order não dá ordens de reorder do artigo 10001.

Para resolver este problema, teorizou-se que se deveria implementar um S-LOCK na tabela inventory, de forma a impedir outra operação de a actualizar enquanto a operação 3 a lia.

Implementou-se a função com um "ACCESS EXCLUSIVE MODE", que apesar de resolver o problema de concorrência, não é um lock adequado ao problema. Ainda está por implementar a colocação um LOCK menos restritivo.