**T07 – Recuperando e modificando valores**

Igor Augusto Reis Gomes – 12011BSI290 – [igor.augusto@ufu.br](mailto:igor.augusto@ufu.br)

Heitor Guimarães Da Fonseca Filho – 12011BSI203 – [heitor.filho@ufu.br](mailto:heitor.filho@ufu.br)

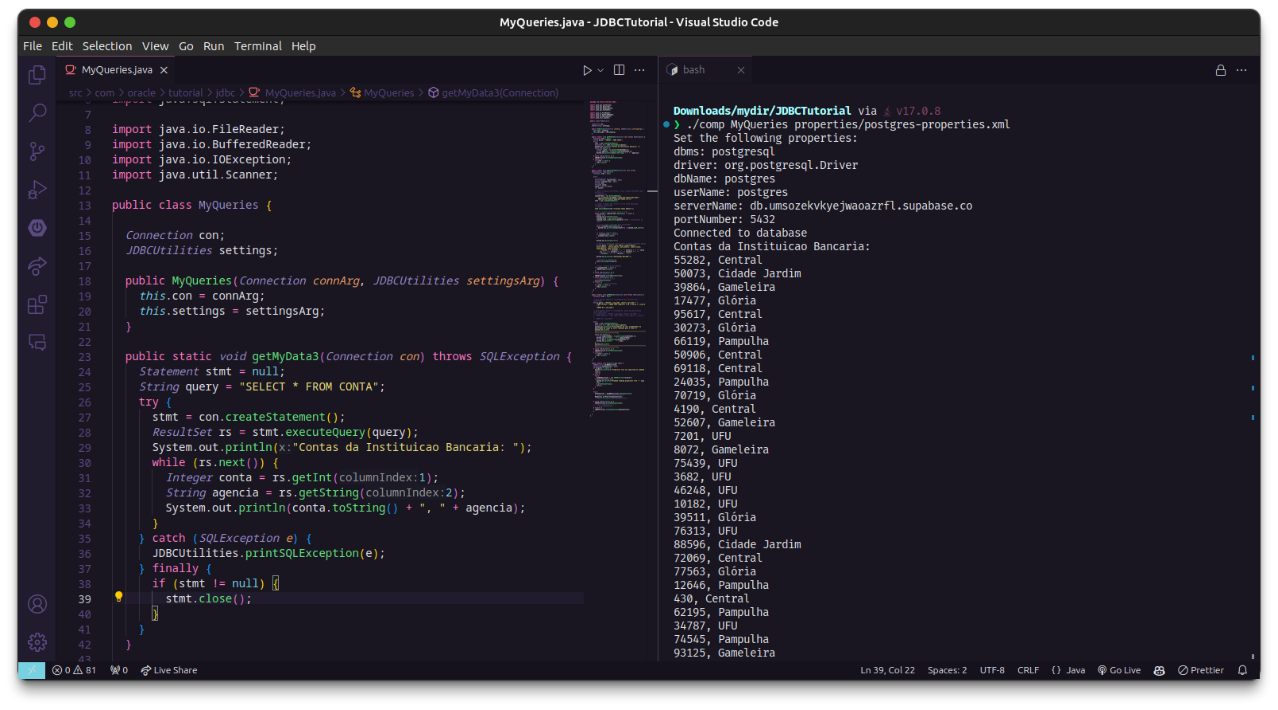
1. **Utilizando a interface do pgadmin3, conecte-se ao banco de dados IB2 e elabore a seguinte consulta no banco de dados:**

**“Retorne os nomes de todos os clientes do banco, com suas respectivas somas de depósitos e empréstimos, caso existam. O resultado das somas deve ser agrupado pelos nomes dos clientes, agência e conta".**

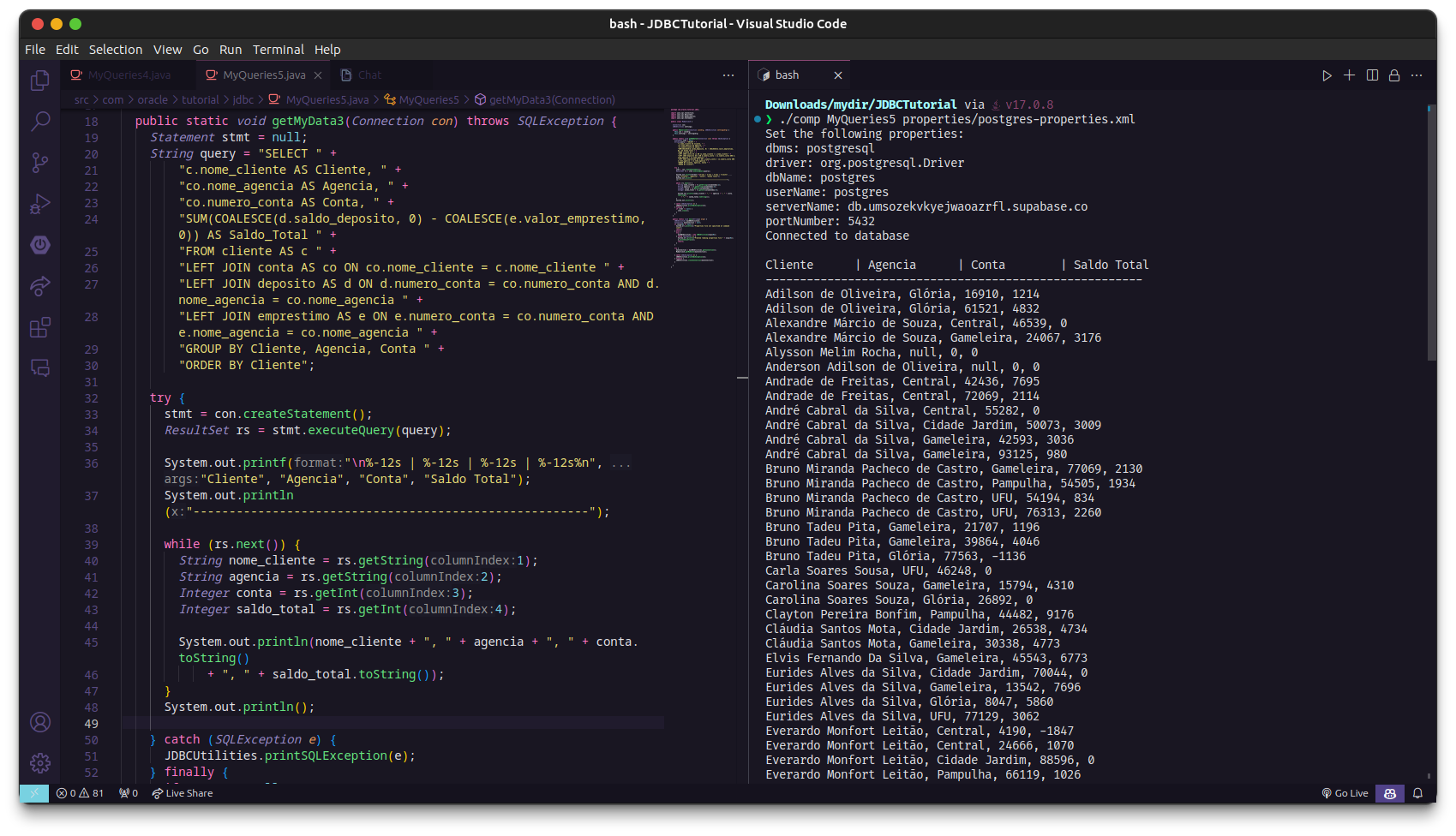
**Texto

Descrição gerada automaticamente com confiança baixa**

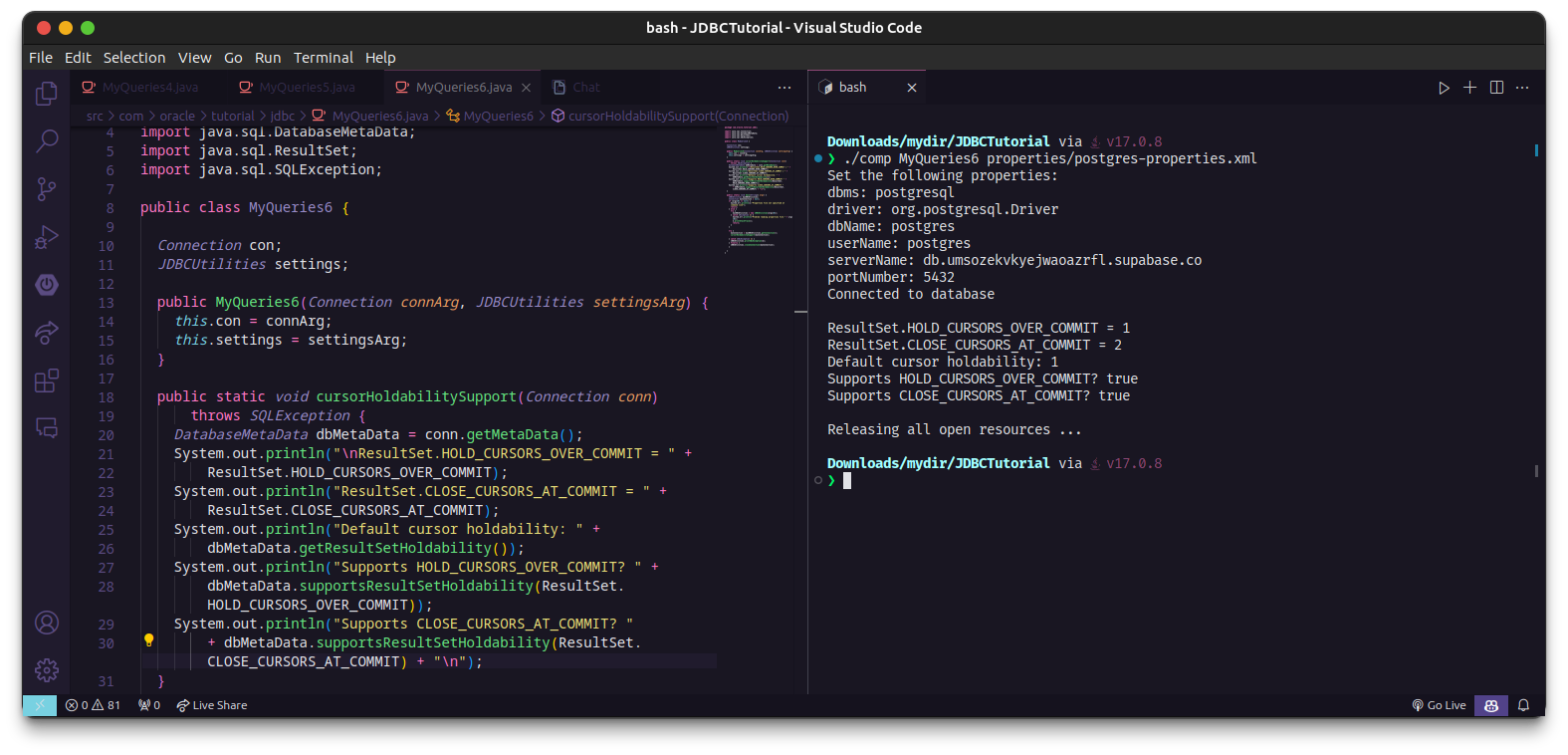
1. **Este código acessa o banco de dados IB2 no postgresql retornando alguns valores. Acrescente-o no seu programa MyQueries e execute-o com o comando comp:**
   1. **./comp MyQueries properties/postgres-properties.xml**



1. **Modifique o programa do item 2 para retornar os dados da consulta solicitada no item 1 utilizando nas funções getInt(), getString(), …, os seguintes indicadores de campos:**
   1. **Índices numéricos. É como está no exemplo acima, apenas lembre-se de tratar todos os campos retornados pela consulta;**
   2. **alias (cláusulas “AS ...”) dos nomes dos campos retornados pela consulta**
   3. **Os nomes dos campos das tabelas alvo;**



1. **Agora você vai explorar as possibilidades de configuração de um SBD para acesso a dados. Para consultar quais as possibilidades de um SGBD faz-se o uso de metadados, ou seja, dados que dizem como os dados são armazenados/consultados. Para início, inclua este código em seu programa MyQueries, execute-o e explique os resultados de acordo com a descrição das constantes documentadas neste site:**
   1. [**http://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html#rs\_interface**](http://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html#rs_interface)



O método imprime informações sobre as opções de retenção de cursores e verifica se o banco de dados suporta essas opções.

Constantes para retenção de cursores:

* 1. ResultSet.HOLD\_CURSORS\_OVER\_COMMIT = 1: Define a constante com valor 1, indicando que os cursores devem ser mantidos abertos após a confirmação de uma transação no banco de dados.
  2. ResultSet.CLOSE\_CURSORS\_AT\_COMMIT = 2: Define a constante com valor 2, indicando que os cursores devem ser fechados automaticamente após a confirmação de uma transação no banco de dados.

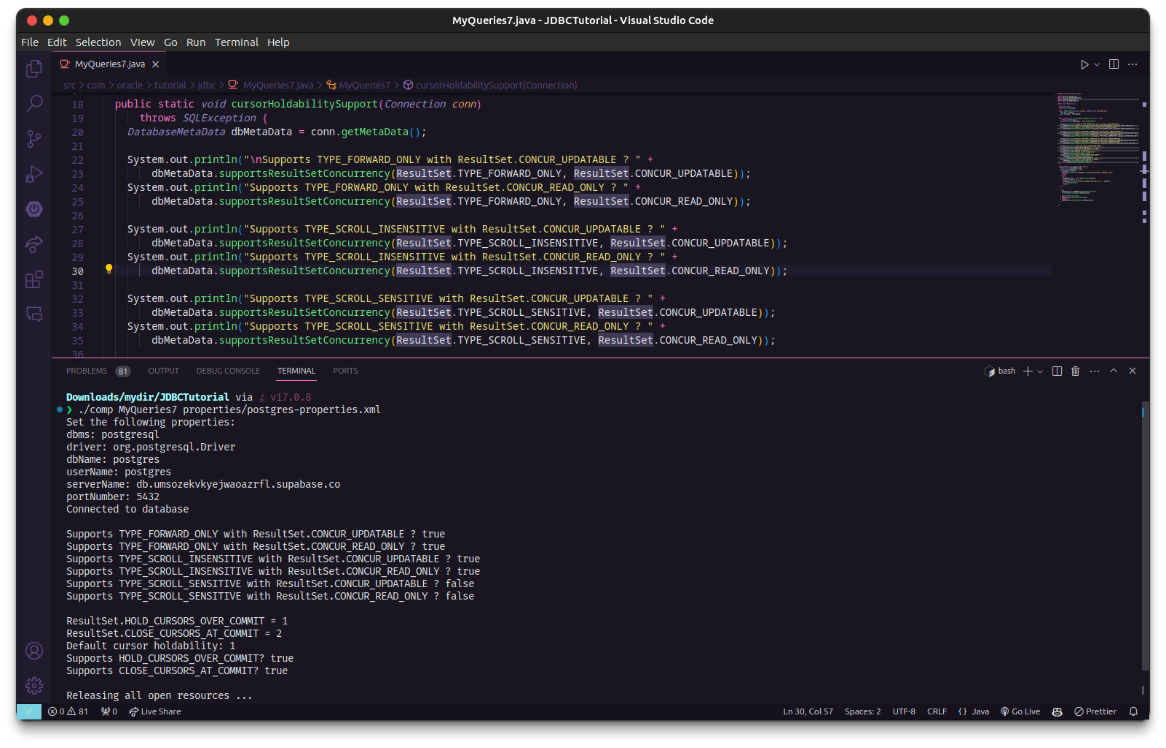
Configuração padrão de retenção de cursores:

* 1. Default cursor holdability: 1. Indica que a config padrão de retenção de cursores para a conexão com o banco de dados é 1, o que corresponde a HOLD\_CURSORS\_OVER\_COMMIT

Verificação de suporte às opções de retenção de cursores:

* 1. Supports HOLD\_CURSORS\_OVER\_COMMIT? true: Verifica se o banco de dados suporta a opção de retenção de cursores HOLD\_CURSORS\_OVER\_COMMIT. Neste caso, o resultado é true, indicando que o banco de dados suporta a retenção de cursores após a confirmação de uma transação.
  2. Supports CLOSE\_CURSORS\_AT\_COMMIT? true: Verifica se o banco de dados suporta a opção de retenção de cursores CLOSE\_CURSORS\_AT\_COMMIT. Mais uma vez, o resultado é true, indicando que o banco de dados suporta o fechamento automático de cursores após a confirmação de uma transação.

1. **Expanda o código do item 4 para verificar se o banco de dados ao qual você está se conectando suporta estes três tipos de ResultSet combinados com uma configuração para os dois tipos de acesso possíveis de serem passados como o segundo parâmetro da função supportsResultSetConcurrency. Perceba que como temos três tipos de ResultSet vezes dois tipos de acesso, então você deve criar seis diferentes acionamentos da função supportsResultSetConcurrency e imprimir o valor de retorno na tela, tal qual está sendo feito no item 4:**



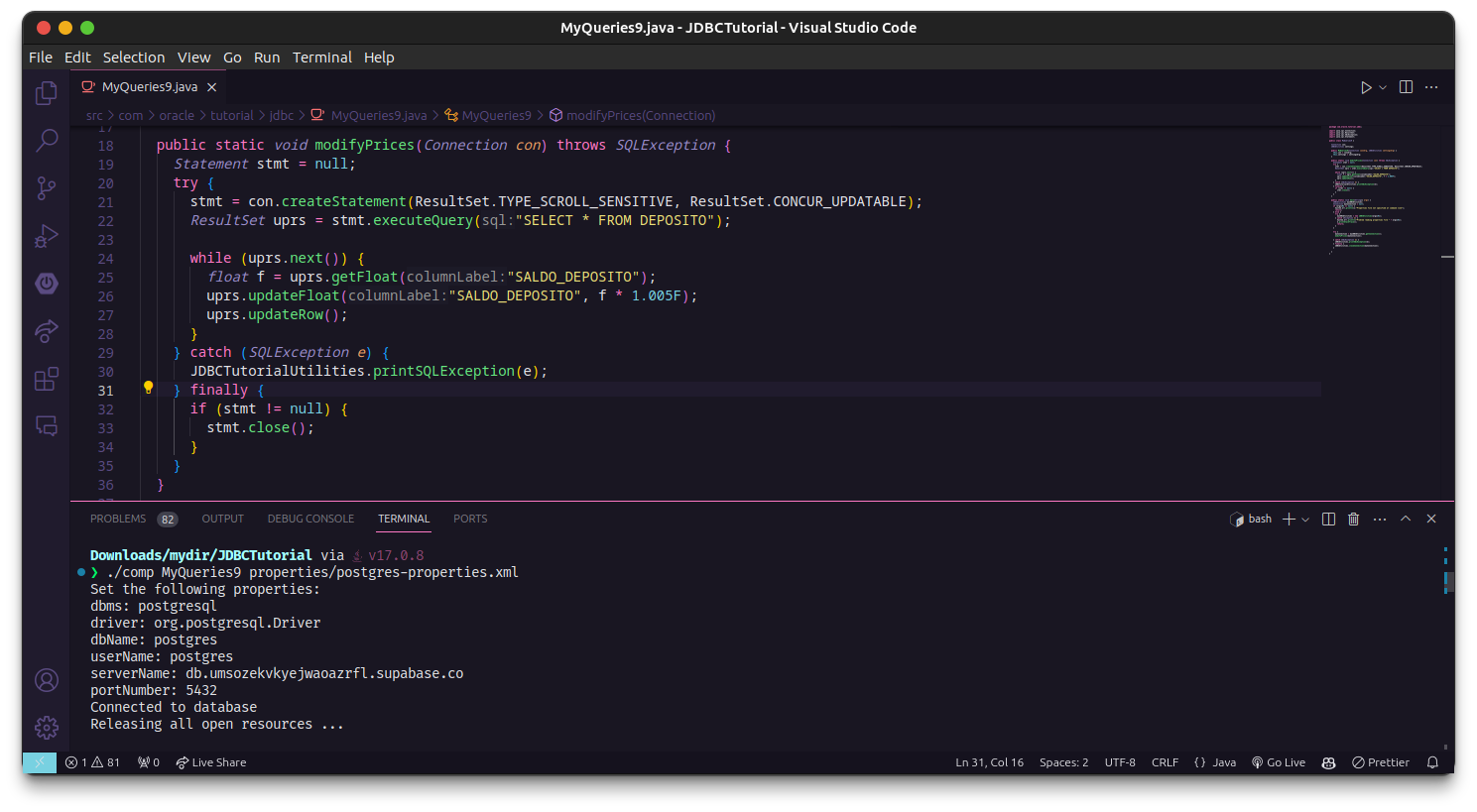
1. **Analise o seguinte código e entenda o que ele se propõe a fazer no banco de dados de exemplo do tutorial JDBC;**
   1. **Aviso que esse código possui um erro. Quero que você o identifique utilizando seu conhecimento adquirido pela execução e apreciação dos resultados do item 5.**

O propósito do método modifyPrices é recuperar dados da tabela "COFFEES" e modificar a coluna "PRICE" aumentando-a em 0,5% para cada linha no conjunto de resultados, utilizando um ResultSet para navegar pelos resultados da consulta e atualizar a coluna "PRICE" para cada linha.

Entretanto, há duas questões críticas no código:

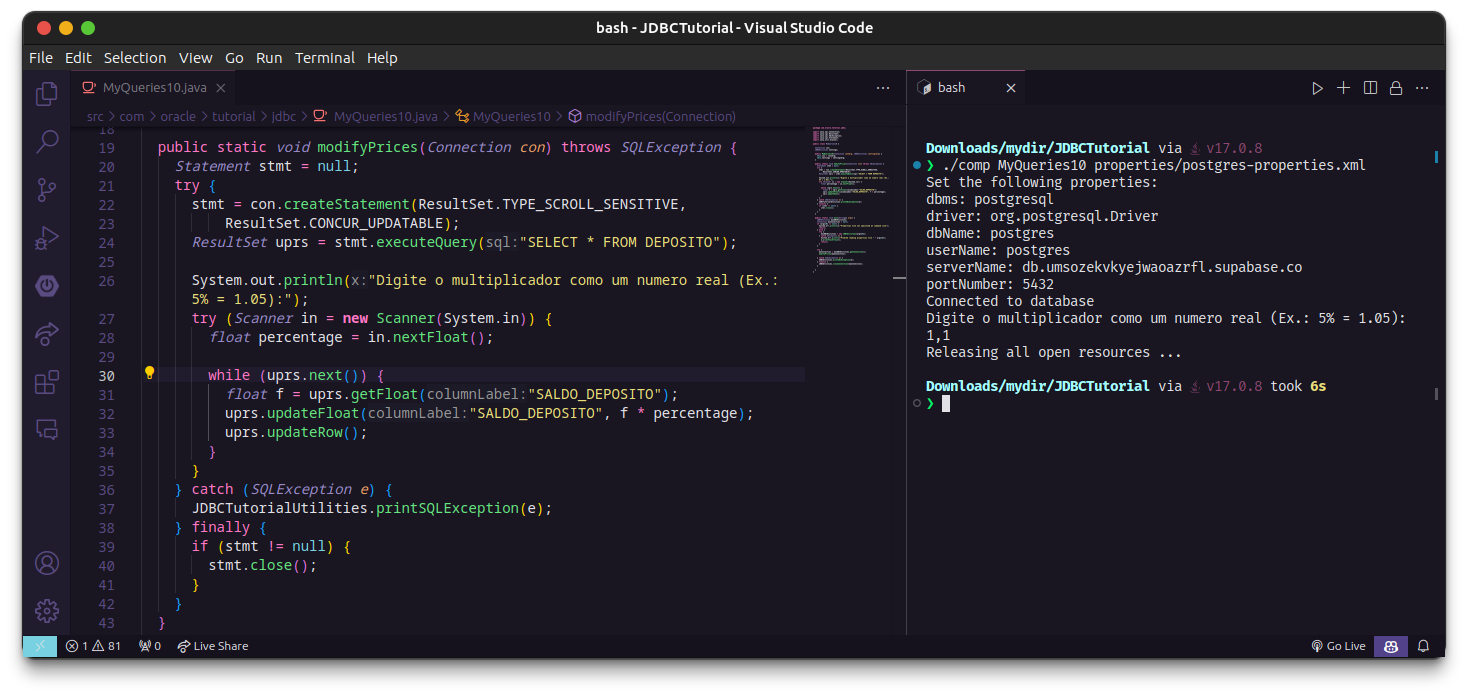
A primeira é que, ao definir ResultSet.CONCUR\_READ\_ONLY, indica-se explicitamente que o conjunto de resultados (ResultSet) é somente leitura, ou seja, não pode ser atualizado. No entanto, logo em seguida, o código tenta realizar uma atualização usando a linha uprs.updateFloat("PRICE", f \* 1.005);. Isso é logicamente inconsistente e resultará em um erro. A abordagem correta seria usar ResultSet.CONCUR\_UPDATABLE para permitir atualizações no ResultSet.  
  
A segunda questão está relacionada à utilização do método updateFloat. Esse método não possui uma assinatura que aceite uma string e um valor double como parâmetros. No código fornecido, é utilizado uprs.updateFloat("PRICE", f \* 1.005);, em que f \* 1.005 é um valor double. No entanto, o método updateFloat deve receber um valor float como parâmetro. Portanto, a maneira correta de utilizá-lo seria explicitar o valor como float, como por exemplo: uprs.updateFloat("PRICE", f \* 1.005F);.

1. **Supondo que a tabela de depósitos diz respeito a depósitos em contas poupança, modifique o código de exemplo no item 6 para atualizar a tabela de depósitos acrescentando juros de 0.5% a todas as linhas da tabela. Lembre-se que o erro no código do item 6 deve estar corrigido;**

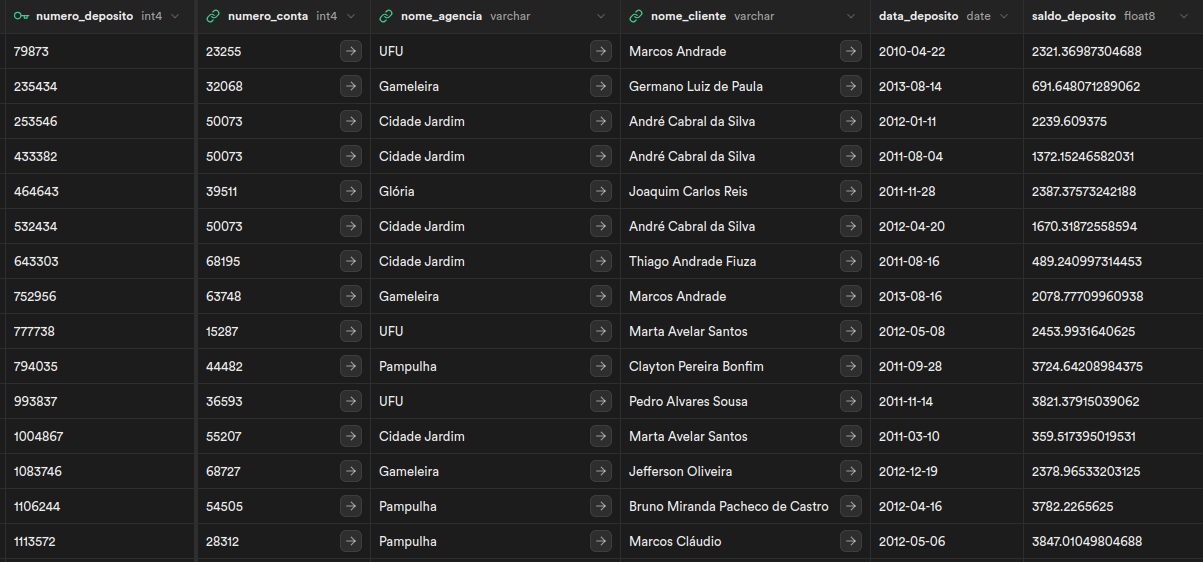


1. **Modifique a função do item 7 para ler do teclado um número que representa a porcentagem de juros que deve ser aplicada aos débitos. O código abaixo permite ler o número do teclado:.**

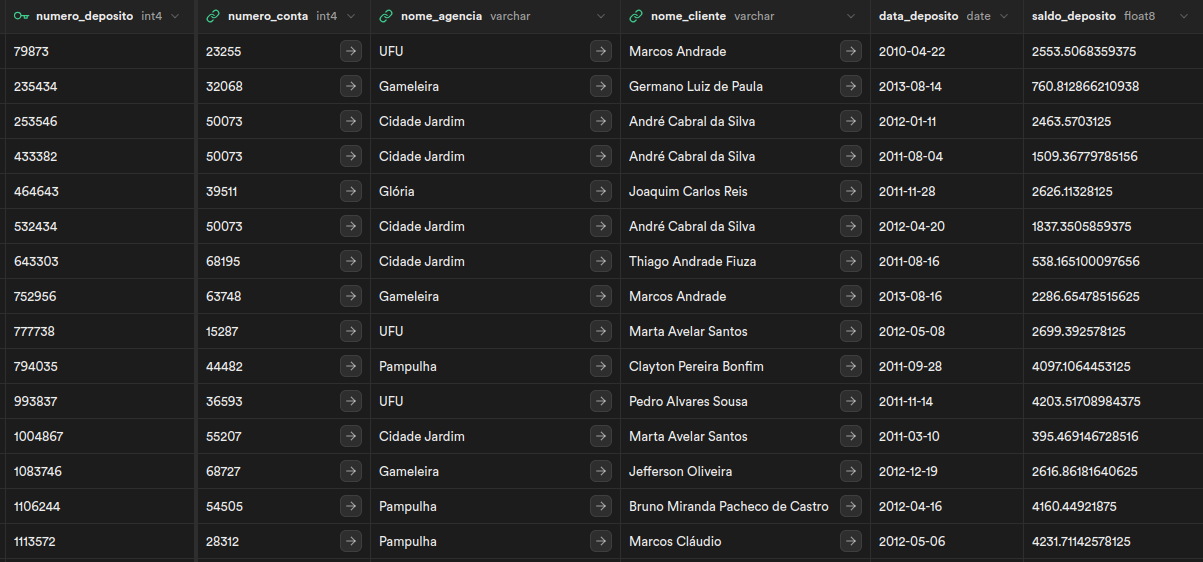
Execução do código



Antes do aumento de 10% (1,1) na tabela depóstios, coluna saldo\_deposito (ordenado de forma ascendente pela coluna numero\_deposito):



Após aumento:



1. **Operações em batch constituem outra forma de atualização de um banco de dados. Elas são um conjunto de operações executadas em bloco, ou seja, apesar da execução de vários comandos, por exemplo, para inserção de dados pelo programa em java, estes comandos somente passarão a figurar no banco de dados após a execução do comando setAutoCommit(true) acessado da variável que simboliza a conexão. Para que os comandos possam ser disparados pelo programa em java é necessário então desligar o commit do banco de dados com o comando setAutoCommit(false). Utilizando o primeiro exemplo do link** [**http://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html#batch\_updates**](http://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html#batch_updates)**, odifique o código da aula anterior (10) para realizar inserções das novas linhas na tabela débito em batch. Neste caso você deve considerar que todos os comandos de inserção da aula 10 constituem um único bloco, ou seja, desligar o commit no início das inserções e ligar novamente apenas após a inserção da última linha.**

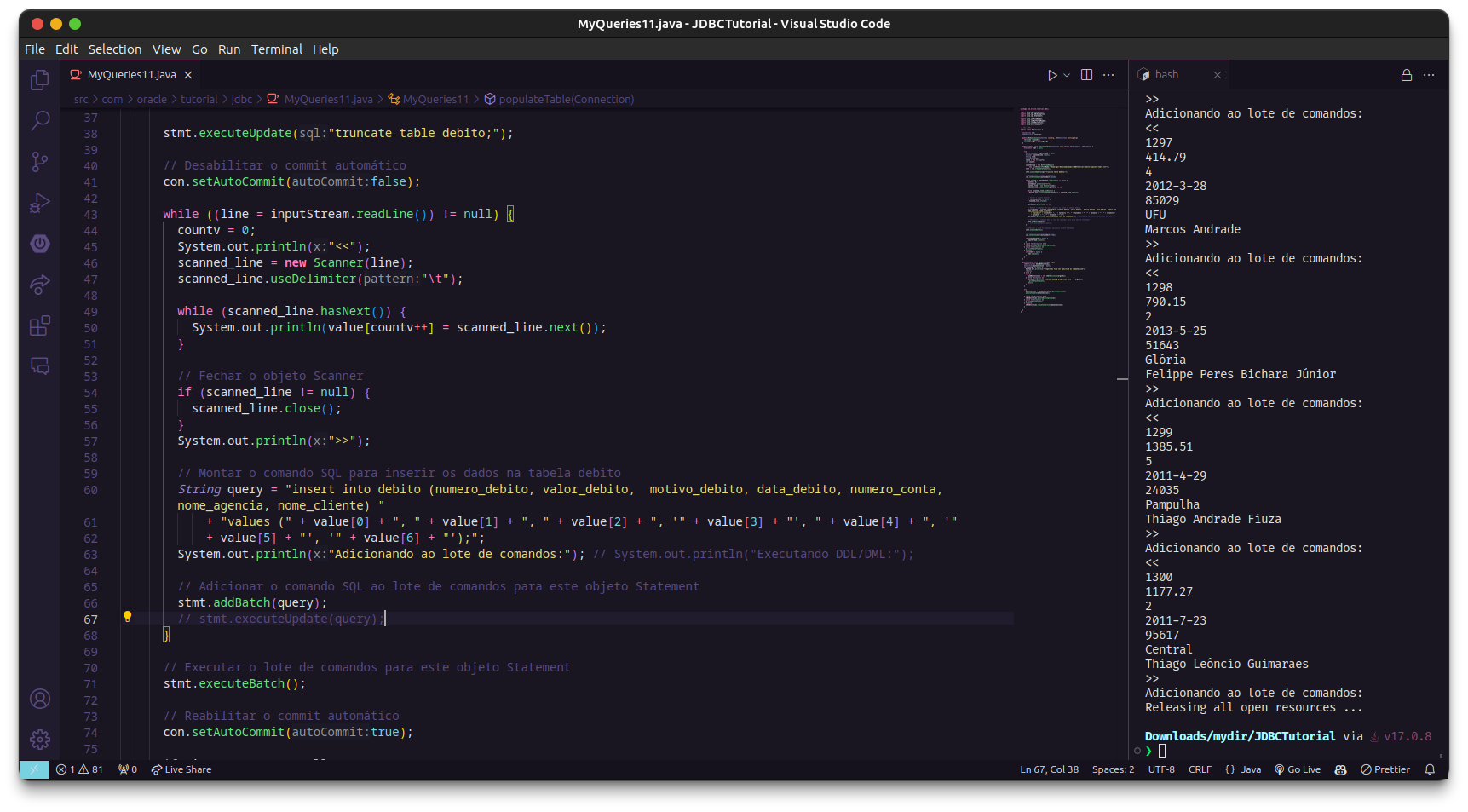
Fiz as seguintes alterações:

1) Adicionei a desabilitação do commit automático com con.setAutoCommit(false); no início do método.

2) Adicionei os comandos stmt.addBatch(query); e stmt.executeBatch(); para adicionar e executar o lote de comandos, respectivamente.

3) Removi o comando stmt.executeUpdate(query); que executava o comando SQL para inserir os dados na tabela debito pois o comando foi adicionado ao lote de comandos, então não é mais necessário executá-lo.

4) Adicionei a reabilitação do commit automático com con.setAutoCommit(true); no final do método.



1. **Uma forma de inserção de dados em um BD's muito comum em aplicações é o preenchimento de dados em uma interface gráfica (geralmente uma janela) com entidades de entrada de textos e números. Estes dados são separados isoladamente em variáveis no programa para posterior adição ao BD, após as devidas verificações de validade na sintaxe da entrada de dados. Nesse caso não usei a palavra “semântica” propositalmente, visto que podemos deixar o próprio SGBD criticar possíveis erros de integridade referencial, por exemplo. No código exemplo logo abaixo temos variáveis que podem ter sido lidas da interface e utilizadas para acionar a função de inserção de linhas. Modifique o código abaixo para inserir as seguintes tuplas na tabela débito do nosso BD da instituição bancária.**

Além da inserção dos dados propriamente ditos, acrescentei adicionalmente, uma verificação para caso os dados já existam no banco, os mesmos serão excluídos para que não ocorra erro de chave duplicada (integridade referencial). Por fim, imprimi os dados que foram inseridos no console.

