Sistemas de Bancos de Dados
Aula 11 – Recuperando e modificando valores

As atividades realizadas anteriormente eram referentes a criação de tabelas, inserção e recuperação de dados destas tabelas. Nas atividades desta aula você vai explorar um pouco mais de recuperação de dados, porém você vai recuperar os dados para depois modificá-los e devolver as modificações para o banco de dados, sempre que possível.

A possibilidade de uma modificação de dados ser bem ou mal sucedida depende das capacidades do SGBD ao qual você está conectado, conjugadas com as opções de acesso que você mesmo explicita no momento em que configura a conexão ao banco de dados. Você será conduzido a explorar estas possibilidades de configurações de um SBD. Também vai explorar as diferentes possibilidades de atualização de dados, por exemplo, uma linha por vez ou várias linhas por vez.

http://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html

Para esta aula utilizaremos o banco de dados IB2 criado e modificado na aula anterior;

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 devem ser agrupados pelos nomes dos clientes, agência e conta".

2) 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*:

```
public static void getMyData3(Connection con) throws SQLException {
  Statement stmt = null;
  String query = "SELECT * FROM CONTA";
 try {
    stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    System.out.println("Contas da Instituicao Bancaria: ");
    while (rs.next()) {
     Integer conta = rs.getInt(1);
      String agencia = rs.getString(2);
      System.out.println(conta.toString() +", " + agencia);
    }
  } catch (SQLException e) {
    JDBCUtilities.printSQLException(e);
  } finally {
    if (stmt != null) { stmt.close(); }
  }
}
```

./comp MyQueries properties/postgres-properties.xml

3) 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;

4) 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:

http://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html#rs_interface

```
import java.sql.DatabaseMetaData;

public static void cursorHoldabilitySupport(Connection conn)
    throws SQLException {

    DatabaseMetaData dbMetaData = conn.getMetaData();
    System.out.println("ResultSet.HOLD_CURSORS_OVER_COMMIT = " +
        ResultSet.HOLD_CURSORS_OVER_COMMIT);

    System.out.println("ResultSet.CLOSE_CURSORS_AT_COMMIT = " +
        ResultSet.CLOSE_CURSORS_AT_COMMIT);

    System.out.println("Default cursor holdability: " +
        dbMetaData.getResultSetHoldability());

    System.out.println("Supports HOLD_CURSORS_OVER_COMMIT? " +
        dbMetaData.supportsResultSetHoldability(
            ResultSet.HOLD_CURSORS_OVER_COMMIT));

    System.out.println("Supports CLOSE_CURSORS_AT_COMMIT? " +
        dbMetaData.supportsResultSetHoldability(
            ResultSet.CLOSE_CURSORS_AT_COMMIT));
}
```

5) Existe um método para identificar se um banco de dados aceita uma atualização ou não dos dados de um *ResultSet*, o método *supportsResultSetConcurrency.* Na figura abaixo está uma cópia de sua documentação assim como consta na documentação do JDBC:

supportsResultSetConcurrency

```
boolean supportsResultSetConcurrency(int type,
                                     int concurrency)
                                       throws SQLException
```

Retrieves whether this database supports the given concurrency type in combination with the given result set type.

Parameters:

type - defined in `java.sql.ResultSet`

concurrency - type defined in `java.sql.ResultSet`

Returns:

`true` if so; `false` otherwise

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

`Connection`

Para acionar a função *supportsResultSetConcurrency* você pode utilizar três tipos de *ResultSet* como sendo o primeiro parâmetro desta função: *ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE e ResultSet.TYPE_SCROLL_SENSITIVE.* 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. Apenas a leitura dos dados retornados no ResultSet, cuja constante é *ResultSet.CONCUR_READ_ONLY*;
2. Atualização de dados por meio da modificação do *ResultSet*, cuja constante é: *ResultSet.CONCUR_UPDATABLE*;

6) Analise o seguinte código e entenda o que ele se propõe a fazer no banco de dados de exemplo do tutorial JDBC;

```
public static void modifyPrices(Connection con) throws SQLException {

    Statement stmt = null;
    try {
        stmt = con.createStatement();
        stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                   ResultSet.CONCUR_READ_ONLY);
        ResultSet uprs = stmt.executeQuery( "SELECT * FROM COFFEES");

        while (uprs.next()) {
            float f = uprs.getFloat("PRICE");
            uprs.updateFloat( "PRICE", f * 1.005);
            uprs.updateRow();
        }

    } catch (SQLException e ) {
        JDBCTutorialUtilities.printSQLException(e);
    } finally {
        if (stmt != null) { stmt.close(); }
    }
}
```

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.

7) 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;

8) 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:

```
        System.out.println("Digite o multiplicador como um numero real (Ex.: 5% = 1,05):");
        Scanner in = new Scanner(System.in);
        double percentage = in.nextDouble();
```

9) 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, modifique 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.

10) 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.

Número do débito	Valor do débito	Motivo do débito	Data do débito	Número da Conta	Nome da Agência	Nome do Cliente
2000	150	1	"2014-01-23"	46248	"UFU"	"Carla Soares Sousa"
2001	200	2	"2014-01-23"	26892	"Glória"	"Carolina Soares Souza"
2002	500	3	"2014-01-23"	70044	"Cidade Jardim"	"Eurides Alves da Silva"

```
import java.sql.Date;

public static void insertRow(Connection con,  ...)
 throws SQLException {

    Statement stmt = null;
    try {
        stmt = con.createStatement( ..., ...);

        ResultSet uprs = stmt.executeQuery("SELECT * FROM ...");

        uprs.moveToInsertRow(); //posiciona no ponto de inserção da tabela

        uprs.updateInt("...", ...);
        uprs.updateDate("data_debito", Date.valueOf(data_debito) );
         ...

        uprs.insertRow(); //insere a linha na tabela

        uprs.beforeFirst(); //posiciona-se novamente na posição anterior ao primeiro registro
    } catch (SQLException e ) {
        JDBCTutorialUtilities.printSQLException(e);
    } finally {
        if (stmt != null) { stmt.close(); }
    }
}
```

Aqui a variável data_debito é do tipo "*String*", tal qual está na tabela de dados acima. Lembrem-se que também deve ser provida a chamada à função no método *main* da classe.

11) Crie um relatório sobre os resultados e conclusões de todas as práticas desta atividade e envie para o professor/tutor acompanhado de códigos-fonte em java da atividade. **É essencial** o envio de um relatório e dos códigos-fonte separados para avaliação;