Aula 4

JDBC - Java Database Connectivity

Java - JDBC

- Histórico
  - Diversos produtos de bases de dados com linguagens próprias de acesso e não compatíveis.
  - Solução: Criar uma API de acesso genérico para bases de dados.
  - Formação de um consórcio para definir esta API
    - SAG - Standard Query Language Access Group
  - Padrão unificado para acesso a base de dados
    - SQL CLI - Standard Query Language Call Level Interface
  - CLI é apenas uma maneira para submissão de sentenças SQL aos Sistemas Gerenciadores de Banco de Dados (SGBD);

Java - JDBC

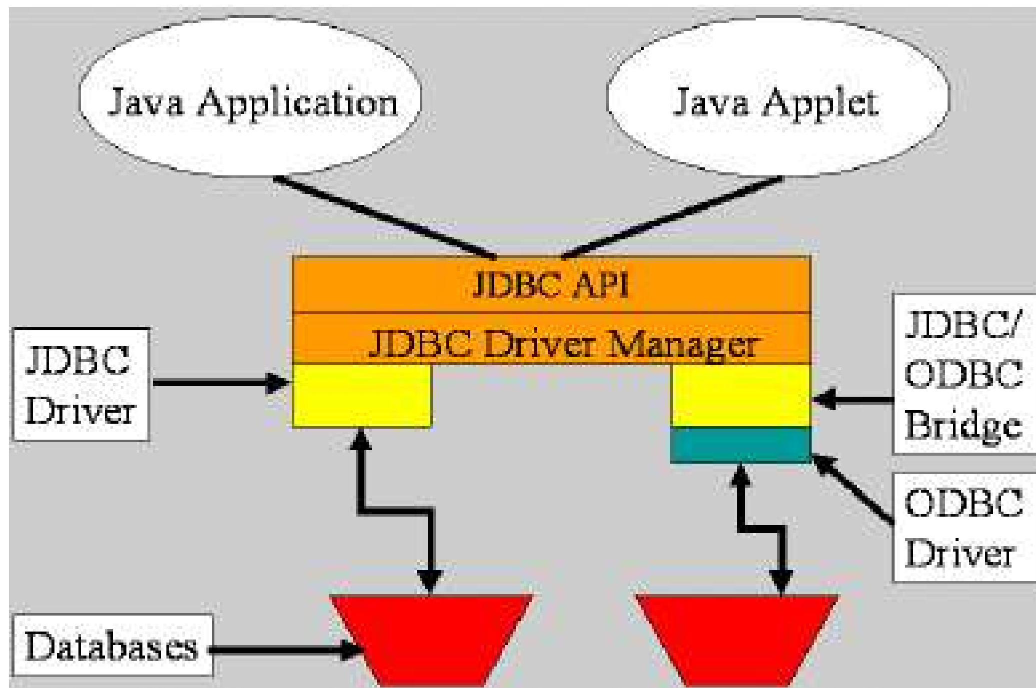
- CLI ODBC - Microsoft
  - ODBC - Open Database Connectivity - padrão estendido da CLI
  - Destina-se a plataforma Windows
  - 1992: ODBC 1.0
    - Acesso comum a base de dados distintas usando SQL
    - Problemas: lenta, defeituosa e documentação escassa.
  - 1994: ODBC 2.0
    - Solucionava alguns problemas
    - Adoção de drivers de 32 bits
  - 1996: ODBC 3.0
    - Novas chamadas de funções, documentação
    - Padrão totalmente controlado pela Microsoft

Java - JDBC

- CLI ODBC - Microsoft
  - Atualmente
    - ODBC 4.0 após muitas versões 3.x
    - O interface OBDC é baseada na X/Open SQL CLI
  - Desvantagens
    - Padrão controlado pela Microsoft e constantemente estendido
    - Sobrecarga devido as camadas ODBC
- X/Open SAG CLI
  - Originalmente definida pelo SAG
  - Recebeu este nome em 1994 para diferenciar do ODBC
  - As APIs são baseadas em SLQ dinâmico
  - Em 1996 tornou-se um padrão internacional ISO 9075-3

Java - JDBC

- Arquitetura JDBC



Java - JDBC

- JDBC API
  - Fornece um framework para manipular os dados armazenados em forma tabular, geralmente em um banco de dados relacional.
- JDBC Driver Manager
  - Responsável por gerenciar os drivers específicos de acesso a base de dados.
- JDBC Driver
  - Corresponde ao driver específico para tratar com as requisições e respostas ao SGBD.

Java - JDBC

- Tipos de Drivers
  - **JDBC-ODBC Bridge** (Tipo 1)
    - Permitem acessar base de dados através de ODBC
    - Faz parte do pacote padrão do java
    - Solução dependente de plataforma
  - **Native API** (Tipo 2)
    - APIs específica e nativa de base de dados (C e C++)
    - Traduz as chamadas JDBC em código específico do BD
    - As APIs são dependentes de plataforma
    - Melhor desempenho que o JDBC

Java - JDBC

- Tipos de Drivers
  - **JDBC-Net pure** (Tipo 3)
    - Chamadas traduzidas para um protocolo de rede que não é específico de banco de dados.
    - Chamadas são enviadas para um servidor que traduz para um protocolo específico de banco de dados.
  - **Native-protocol Pure Java Driver** (Tipo 4)
    - Chamadas traduzidas para um protocolo de rede específico para o banco de dados
    - Programas podem se conectar diretamente com o banco de dados
    - Não há camadas adicionais

Java - JDBC

- Processo básico para utilizar o JDBC
  - Carregar e registrar o driver JDBC junto ao Driver Manager
  - Configurar e obter uma conexão com o banco de dados
  - Preparar os dados para consulta (formatação, ordenação, ...) e o próprio SQL
  - Executar a consulta
  - Obter e verificar os resultados
  - Tratar possíveis erros
  - Formatar a saída para o usuário
  - Ao finalizar a aplicação, fechar a conexão

Java - JDBC

- Carregar e registrar um driver JDBC
  - Driver geralmente é uma classe java
    - ex: driver Jay Bird do Firebird
      - pacote: **firebirdsql-full.jar** e classe: **org.firebirdsql.jdbc.FBDriver**
  - Existem três maneiras para se registrar um driver:
    - Através de um *classloader* (mais comum)
      - Class.forName(nome_classe_driver);
    - Através de instanciação
      - nome_classe_driver driver = new nome_classe_driver();
    - Através de parâmetros pela linha de comando
      - java -Djdbc.drivers=nome_classe_driver nome_aplicação

Java - JDBC

- Carregar e registrar um driver JDBC
  - Exemplos utilizando o Firebird
    - por *classloader*

      Class.forName(**org.firebirdsql.jdbc.FBDriver**);

      Class.forName(**org.firebirdsql.jdbc.FBDriver**).newInstance();
    - por inicialização

      org.firebirdsql.jdbc.FBDriver driver = new

      org.firebirdsql.jdbc.FBDriver();
    - por propriedades na inicialização

      java -Djdbc.drivers=**org.firebirdsql.jdbc.FBDriver** nome_aplicação

Java - JDBC

- Obter uma conexão com o banco de dados
  - São necessários três elementos (depende da conf. do BD):
    - URL: utilizada para identificar a base de dados e o driver

      jdbc:<subprotocol>:<subname>

    - Nome de usuário: identifica o usuário
    - Senha de acesso: senha para o acesso

  - A conexão pode ser feita de duas maneiras e utiliza um objeto do tipo *Connection* para referenciá-la.
    - Utilizando o DriverManager

      ```
      Connection minhaCon = DriverManager.getConnection (url, usr, senha);
      ```

    - Utilizando o próprio driver

      ```
      Connection minhaCon = Driver.connect (url, java.util.Properties prop);
      ```

Java - JDBC

- Obter uma conexão com o banco de dados
  - Exemplos utilizando o Firebird

```
String dburl = "jdbc:firebirdsql:localhot/3050://home/databases/aula_db.fdb";
String usr = "aluno_cd";
String senha = "aluno_pw";

java.sql.Connection con = java.sql.DriverManager.getConnection (dburl, usr, senha);
```

ou ...

```
java.util.Properties connectionProperties = new java.util.Properties ();
connectionProperties.put ("user", usr);
connectionProperties.put ("password", senha);
java.sql.Connection con = driver.connect (dburl, connectionProperties);
```

Java - JDBC

- Executar consultas
  - Existem três maneiras:
    - Statements (java.sql.Statement)
      - Executa uma instrução SQL fixa e retorna um resultado. Por padrão, somente um resultado pode permaner aberto por consulta.
    - PreparedStatements (java.sql.PreparedStatement)
      - Executa uma instrução SQL pré-compilada. Fornece meios para passagem de parâmetros para essa instrução.
    - CallableStatements (java.sql.CallableStatement)
      - Executa SQL stored Procedure.

Java - JDBC

- Executar consultas
  - Exemplos
    - Statements

```
Statement stmt = con.createStatement();
```

    - PreparedStatements

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO
tabela_aluno (id, nome) VALUES (?, ?)");
```

    - CallableStatements

```
callableStatement cstmt = con.prepareCall("{call test_sp(?, ?)}");
```

Java - JDBC

- Statement - Métodos
  - ResultSet **executeQuery**(String sql);
    - Executa uma operação SQL que retorna dados como resultados. Esses dados devem ser armazenados em um objeto *ResultSet* para serem manipulados.
  - int **executeUpdate**(String sql);
    - Executa uma operação SQL que retorna o número de linhas afetadas. Utilizado para operações SQL que não devem retornar dados (inserção, deleção ou atualização) ou ainda para funções de manipulação (criação de tabelas, ...).
  - boolean **execute**(String sql);
    - Executa uma operação SQL que pode retornar diversos resultados.

Java - JDBC

- PreparedStatement - Métodos
  - Os métodos de execução são os mesmos do *Statement.*
  - Os métodos abaixo são alguns dos quais podem ser utilizados para substituir o caracter coringa "?"
    - void **setBlob**(int i, Blob x);
    - void **setBoolean**(int parameterIndex, boolean x);
    - void **setByte**(int parameterIndex, byte x);
    - void **setDouble**(int parameterIndex, double x);
    - void **setInt**(int parameterIndex, int x);
    - void **setString**(int parameterIndex, String x);
    - void **setTime**(int parameterIndex, Time x);
    - void **setDate**(int parameterIndex, Date x, Calendar cal);

Java - JDBC

- Statement - Exemplos

```
java.sql.ResultSet rs = stmt.executeQuery ("SELECT * FROM tabela_aluno");
stmt.executeUpdate("CREATE TABLE tabela_aluno (id INT, nome CHAR(25));
stmt.executeUpdate("INSERT INTO tabela_aluno VALUES (1, ´Bicho Papão´);
stmt.execute("INSERT INTO tabela_aluno VALUES (2, 'Curupira');
```

- PreparedStatement - Exemplos

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO tabela_aluno
    (id, nome) VALUES (?, ?)");
pstmt.setInt (1, 1);
pstmt.setString (2, 'Bicho Papão');
pstmt.executeUpdate();
```

Java - JDBC

- CallableStatement - Métodos e exemplo
  - Os métodos são herdados do *Statement* e do *PreparedStatement*
  - A sintaxe geral para utilizá-lo é:

```
{?= call <procedure-name>[<arg1>,<arg2>, ...]}
{call <procedure-name>[<arg1>,<arg2>, ...]}
```

  - Exemplo:

```
CallableStatement cstmt = con.prepareCall( "{ ? = call sp_B( ? ) }" );
cstmt.registerOutParameter( 1, Types.INTEGER );  /* paremetro OUT */
cstmt.setString( 2, "M-O-O-N" );   /* parametro IN */

cstmt.execute();    // ou executeUpdate()

int iRP = cstmt.getInt( 1 );   /* obtém o parâmetro de saída */
```

Java - JDBC

- Obter e tratar os resultados
  - *ResultSet*
    - Tabela de dados que representa um conjunto resposta a uma consulta ao banco de dados.
    - Mantém um cursor apontando para a linha atual de dados
    - Inicialmente esse cursor não aponta para nenhuma linha
    - Por padrão, a movimentação do cursor é somente para frente e os dados não podem ser atualizados diretamente no *ResultSet*.
    - O objeto *ResultSet* é automaticamente fechado se o objeto *Statement* que o gerou é fechado.
    - Fornece métodos para obter os dados da linha selecionada.

Java - JDBC

- ResultSet - Métodos
  - boolean **absolute**(int row): move o cursor para o registro especificado
  - void **close**(): fecha o ResultSet e libera o BD.
  - boolean **first**(): move o cursor para o primeiro registro do ResultSet
  - boolean **last**(): move o cursor para o último registro do ResultSet
  - boolean **next**(): move o cursor para o próximo registro
  - boolean **previous**(): move o cursor para o registro anterior
  - Funções **getTYPE** (int columnIndex) ou **getTYPE** (String columnName)
    - Obtém o valor do campo especificado pelo índice ou pelo nome da coluna do registro corrente.

Java - JDBC

- ResultSet - Métodos (JDBC 2.0)
  - void **cancelRowUpdates**(): cancela as alterações realizadas no registro corrente.
  - void **deleteRow**(): deleta o registro corrente do ResultSet e do BD.
  - void **insertRow**(): insere o conteúdo do registro atual no ResultSet e no BD.
  - void **refreshRow**(): atualiza o valor do registro atual do ResultSet com o valor do registro mais recente do BD.
  - void **updateRow**(): atualiza o BD com os valores do registro corrente.
  - void **updateTYPE**(String columnName, TYPE value): atualiza o BD com o valor da coluna especificada do registro corrente.

Java - JDBC

- ResultSet - Exemplos

```
/* configura o ResultSet para suportar movimentação do cursor bidirecional e atualização */
Statement stmt = con.createStatement (ResultSet.TYPE_SCROLL_INSENSITIVE,
                                      ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT * FROM tabela_aluno");

rs.absolute(5);  /* move o cursor para a linha 5 */
rs.updateString("nome", "Drácula");   /* atualiza a coluna "nome"  */
rs.updateRow();  /* atualiza o registro no base de dados */

/* forma muito usada para navegação em um ResultSet */
while (rs.next( )) {
    int id = rs.getInt(1);
    String s = rs.getString("nome");
} //while
```

Java - JDBC

- Tratar possíveis erros
  - *java.sql.SQLException*
    - permite detectar e tratar os erros causados pelos métodos dos objetos JDBC ao acessar a base de dados.
    - fornece dois métodos para obter mais informações sobre o erro.
  - SQLException - Métodos
    - String **getMessage**(): retorna uma string descrevendo o erro.
    - String **getSQLState**(): retorna um identificador de estado, descrito na especificação X/Open SQL.
    - int **getErrorCode()**: retorna um descritor de erro específico do fabricante do SGDB.
    - SQLException **getNextException**(): recupera a próxima exceção.

Java - JDBC

- Tratar possíveis erros
  - *java.sql.SQLWarning*
    - Exceção que fornece informações sobre alertas ao acesso ao banco de dados.
    - Podem ser recuperados através dos objetos *Statement*, *Connection* e *ResultSet*;
    - Exemplo:

```
stmt = con.createStatement();
sqlw = con.getWarnings();
while( sqlw != null)  {
    /* código para tratar os warnings */
    sqlw = sqlw.getNextWarning();
} //while
con.clearWarnings();
```

Java - JDBC

- Metadados
  - java.sql.DatabaseMetaData
    - Recupera informações sobre o banco de dados
    - Possui cerca de 150 métodos
    - Um SQLException será disparado se o driver não suporta algum dos métodos

```
DatabaseMetaData dbmd = con.getMetaData();
```

  - java.sql.ResultSetMetaData
    - Recupera informações sobre um determinado ResultSet
    - Possui cerca de 25 métodos
    - Obtém informações sobre tipos e propriedades das colunas

```
ResultSetMetaData rsmd = rs.getMetaData();
```

Exercícios

- Utilizando como base a GUI das aulas sobre Swing, crie um banco de dados para armazenar as informações. Inicialmente implemente utilizando os métodos da interface Statement.

- Crie uma janela de consulta que permita executar diversos tipos de seleção e exiba os resultados em uma JTable. Utilize os métodos da interface PreparedStatement para construir as consultas.

- Implemente uma Stored Procedure e teste os métodos da interface CallableStatement.

- Implemente um teste que obtenha metadados do ResultSet e do banco de dados e exiba na tela.

Para saber mais

- JDBC Technology
  http://www.javasoft.com/products/jdbc/index.html
- Java Database Programming
  http://java.sun.com/developer/onlineTraining/Database/JDBCShortCourse/contents.html
- Basic Tutorial
  http://java.sun.com/developer/Books/jdbc/Fisher/Fisher_ch02.pdf
- jGuru JDBC 2.0 Fundamentals
  http://java.sun.com/developer/onlineTraining/Database/JDBC20Intro/index.html