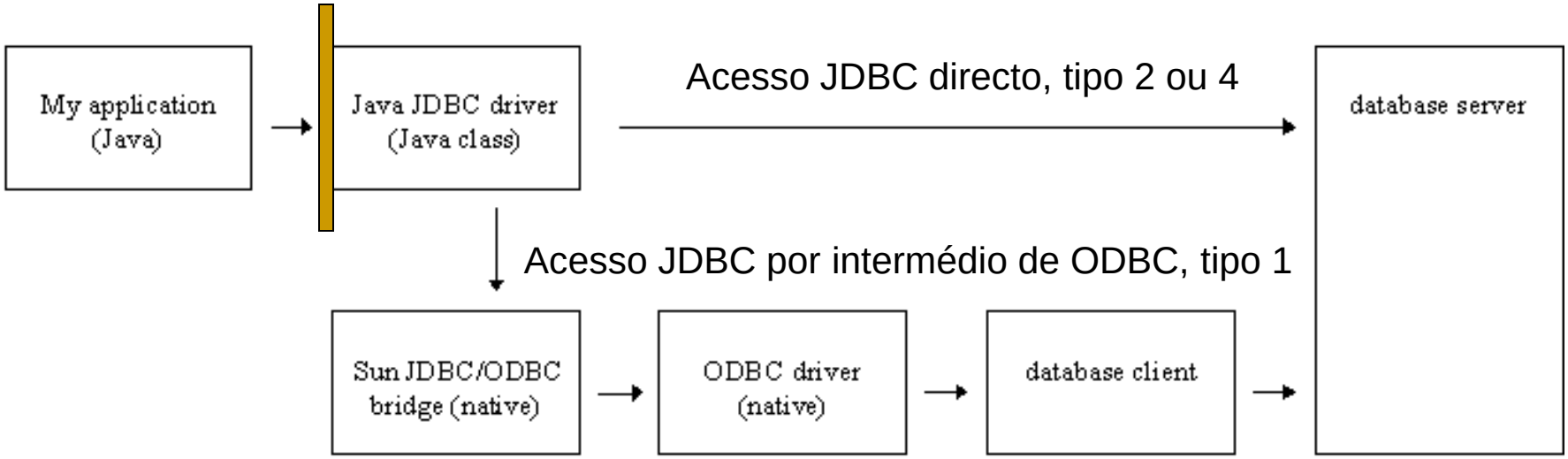
08 – JDBC

Baseado nos slides dos professores Paulo Trigo e Porfírio Filipe
Todas as alterações são da responsabilidade do professor António Teófilo

O que é o JDBC?

- JDBC – Java Database Connectivity
- É uma interface normalizada que permite o acesso, em código Java, a bases de dados relacionais.
- É uma implementação X/Open SQL CLI (Call Level Interface), compatível com o SQL 92
- Atualmente na versão 4.3 para java SE 9 ou superior

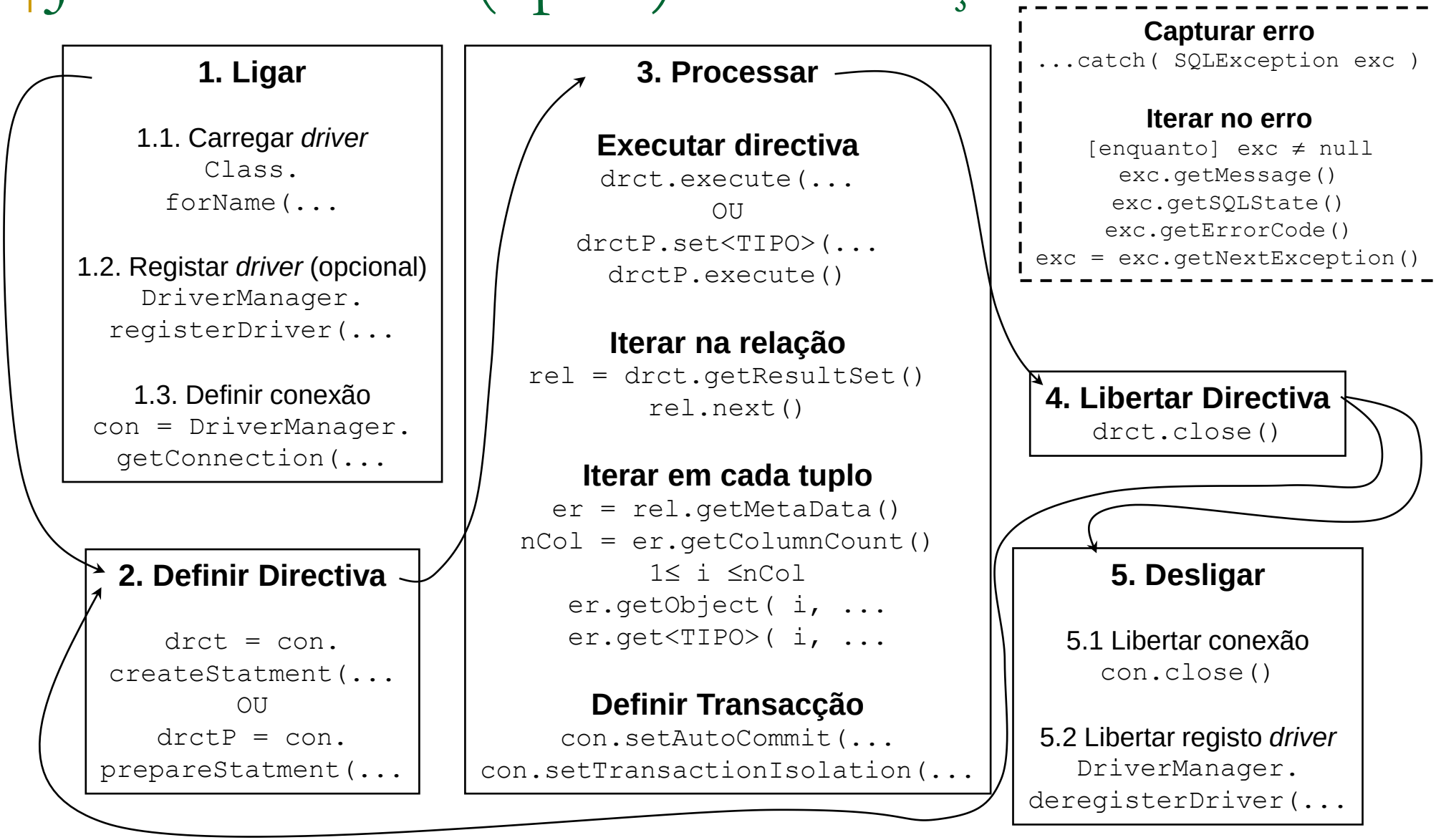
Arquitectura JDBC (*Java Database Connectivity*)



Tipos de Driver JDBC

- **Tipo 1 - JDBC-ODBC bridge**
  - É um driver que faz de ponte entre a interface JDBC e o driver ODBC
  - ODBC – Open Database Connectivity é outro standart para acesso a dados compatível com o X/Open CLI.
- Tipo 2 - Native-API Driver specification
  - É um driver que utiliza directamente as bibliotecas da base de dados para o seu acesso
- Tipo 3 - Network-Protocol Driver
  - É um driver que comunica com uma camada intermédia (middle-tier, ex: J2EE application server), a qual comunica com os servidores de dados, mas que acrescenta serviços como: caching, load balacing, logging, connection pooling, etc).
- **Tipo 4 - Native-Protocol Driver**
  - É um driver que utiliza o protocolo de comunicação via rede, para comunicar com a aplicação servidora da base de dados (concebida pelo fabricante da DB).

JBDC – Fluxo (típico) de utilização



1. Ligar (1.1. e 1.2.)

JDBC – ODBC (type 1)

```
// Classes da biblioteca JDBC
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.SQLException;


...
// 1.1. Carregar driver
// JDBC-ODBC "bridge driver"
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" ); //*1


...
// 1.2. Registar driver
// A database URL of the form jdbc:subprotocol:subname
Driver driver = DriverManager.getDriver( "jdbc:odbc:sun" );
DriverManager.registerDriver( driver );

...
```

MySQL JDBC (type 4)

```
*1 – com.mysql.jdbc.Driver
```

ClassNotFoundException

SQLException

Passo 1.2 já é automático no passo 1.1 (JDBC 3)
O passo 1.1 já carrega, regista e cria uma instância do driver

1. Ligar (1.3.)

MySQL JDBC (type 4)

```
*1 - jdbc:mysql:
*2 - //localhost/teste
```

```
// Classes da biblioteca JDBC
import java.sql.Connection;
import java.sql.SQLException;


...
// Formato do URL para conexao á base de dados
// jdbc:odbc:data-source-name
String prefixoURL = "jdbc:odbc:"; // *1
String fonteDados = prefixoURL + obterNomeFonteDados(); // *2


// 1.3. Definir conexão ---------> SQLException
Connection conexao = DriverManager.getConnection(
        fonteDados, // Fonte de Dados ("Data Source")
        obterNomeUtilizador(), // Nome do utilizador
        obterSenha() ); // Senha (password)
...
```

Interacção com o Utilizador:

*obterNomeFonteDados() // *2*
obterNomeUtilizador()
obterSenha()

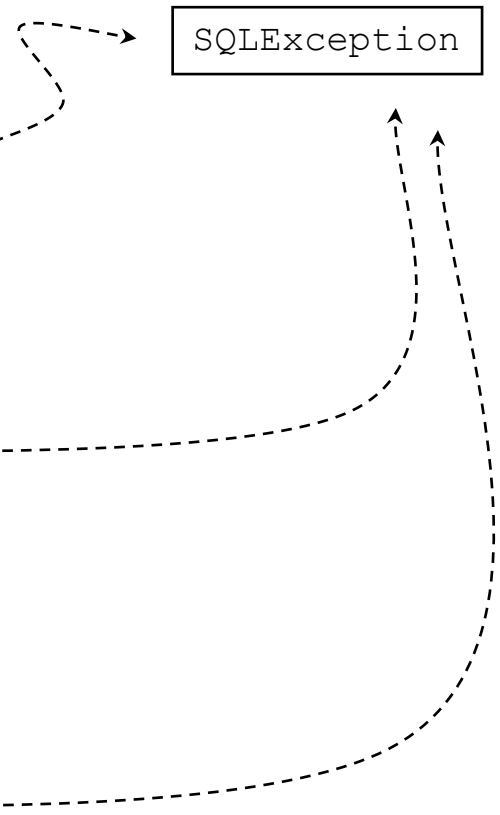
Capturar e tratar erros: SQLException

```
// Classes da biblioteca JDBC
import java.sql.SQLException;

...
try
{
   função que lance uma SQLException (e.g. DriverManager.getConnection())
}
catch( SQLException excepcao )
{
   while( excepcao != null )
   {
      System.out.println( excepcao.getMessage() );
      System.out.println( excepcao.getSQLState() );
      System.out.println( excepcao.getErrorCode() );

      // Obter a proxima excepcao contida no objecto
      excepcao = excepcao.getNextException();
   }
}
...
```

Em SQLException uma excepção pode conter uma cadeia de excepções
Ver método: setNextException(SQLException ex)

2. Execução directa de statements

```
// Classes da biblioteca JDBC
import java.sql.Statement;
import java.sql.SQLException;

... // obter conexao

// criar um comando para execução directa
Statement directiva = conexao.createStatement();

...

// executar o comando
String str_directiva = "SELECT * FROM person";
directiva.execute( str_directiva );

...

// processar resultado

directiva.close(); // libertar recursos
```



2. Execução preparada de statements

Os *prepared statements* são statements pre-compilados (se o driver o permitir) e podem ser executados várias vezes com valores diferentes.
Os ? Indicam os valores* que podem devem ser indicados mais tarde.

```
// Classes da biblioteca JDBC
import java.sql.PreparedStatement;
import java.sql.SQLException;

... // obter conexao

String directiva = "INSERT INTO Table1 VALUES (?, ?)";
PreparedStatement directivaP = conexao.prepareStatement( directiva );

...

directivaP.setInt( 1, valor1 );
directivaP.setString( 2, valor2 );
directivaP.execute();
...
// processar resultado

// colocar novos valores, executar, ...

directivaP.close(); //libertar recursos
```

SQLException

Os ? serão concretizados antes de ser solicitada a execução da directiva

Admitindo que:
1° atributo de T é tipo int
2° atributo de T é tipo char ou varchar
valor1, *valor2* contêm os valores a inserir

* Os '?' só podem estar em substituição de valores

3. Processamento de resultados

- A execução de comandos SQL pode devolver um conjunto de registos
- Para resolver este problema, as linguagens utilizam a noção do cursor
- Cursor – objecto que contém os resultados da execução de um comando SQL, e ao qual se pode pedir individualmente cada resultado (linha, tuplo)
- Classe Java que implementa um cursor: **java.sql.ResultSet**
- Utilização:

```
Statement s = …;
s.execute();
ResultSet rs = s.getResultSet(…);
while( rs.next() ) {
   // obter um atributo
   Object valorAtributo = rs.getObject( indiceAtributo );
}
```

3. Processar (Iterar na relação e em cada tuplo)

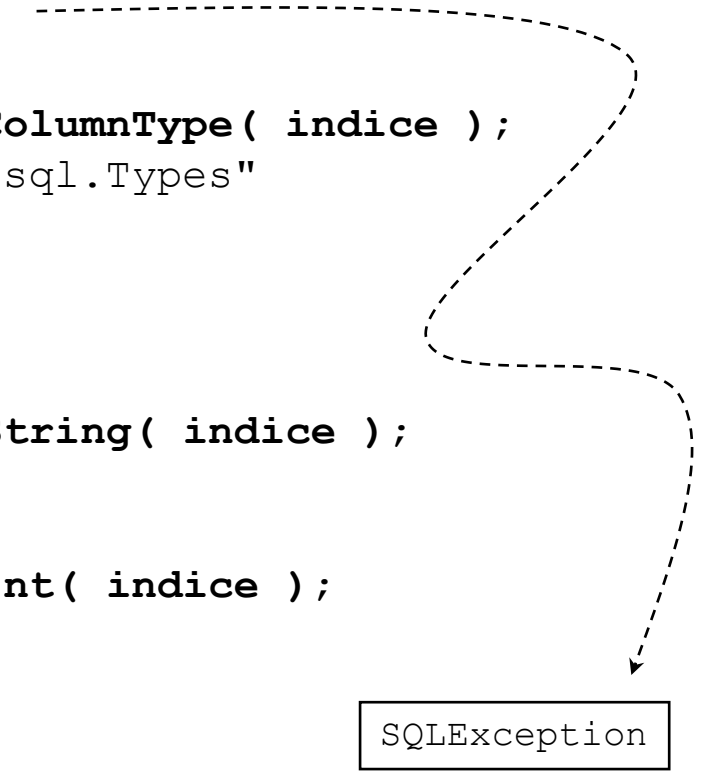
```
// Classes da biblioteca JDBC
import java.sql.ResultSet;
import java.sql.SQLException;

...
// Obter relação
ResultSet relacao = directiva.getResultSet();

// Obter numero de colunas do "esquema de relação"
ResultSetMetaData esquemaRelacao = relacao.getMetaData();
int numeroColunas = esquemaRelacao.getColumnCount();

// Iterar na relação: obter o 1º tuplo e cada um dos restantes
while( relacao.next() )
{
   // Iterar num "tuplo"
   for( int indice = 1; indice <= numeroColunas; indice++ )
   {
      // Tenta ler do "RecordSet" como "Object"
      Object objecto = relacao.getObject( indice );
```

SQLException

... continua na próxima folha

3. Processar (Iterar num tuplo – conversões de tipo)

```
... continuação da folha anterior

// se não conseguiu ler com sucesso, ou seja,
// se obteve null mas o valor do atributo não era null
if( ( objecto == null )
      &&
      ( ! relacao.wasNull() ) )
{
   int tipo = esquemaRelacao.getColumnType( indice );
   // Tal como definido em "java.sql.Types"
   switch( tipo )
   {
      case Types.VARCHAR:
      case Types.CHAR:
           objecto = relacao.getString( indice );
           break;
      case Types.INTEGER:
           objecto = relacao.getInt( indice );
           break;
      ...
   }
...
```



Exemplo completo de acesso a dados

```
Statement stmt = conn.createStatement();
try {
  ResultSet rs = stmt.executeQuery( "SELECT * FROM MyTable" );
  try {
    int numColumns = rs.getMetaData().getColumnCount();
    while ( rs.next() ) {
      for ( int i = 1 ; i <= numColumns ; i++ ) {
        // Column numbers start at 1.
        System.out.println( "COLUMN " + i + " = " + rs.getObject(i) );
      }
    }
  } finally {
      rs.close();
  }
} finally {
     stmt.close();
}
```

Nota: falta colocar o tratamento das excepções

4. Libertar Directiva; e 5. Desligar

```
// Classes da biblioteca JDBC
import java.sql.SQLException;

...
// 4. Libertar directiva
// =====================
directiva.close(); /* de execução directa */

directivaP.close(); /* de execução preparada */


// 5. Desligar
// ==========
// 5.1. Libertar conexao
conexao.close();

// 5.2 Libertar registo "driver"
DriverManager.deregisterDriver( driver );
...
```

SQLException

… e ainda … como obter Meta Informação

```
// Classes da biblioteca JDBC
import java.sql.SQLException;
import java.sql.DatabaseMetaData;


// Gets the metadata regarding this connection's database.
// A Connection's database is able to provide information describing its tables,
// its supported SQL grammar, its stored procedures,
// the capabilities of this connection, and so on.
// This information is made available through a DatabaseMetaData object
DatabaseMetaData metaInformacaoBD = conexao.getMetaData();

// Obter o nome do SGBD
String nomeSGBD = metaInformacaoBD.getDatabaseProductName();

// Obter o número máximo de conexões activas permitidas
int maximoConexoes = metaInformacaoBD.getMaxConnections();

// Obter informação sobre bases de dados geridas pelo SGBD
ResultSet catalogRS =  metaInformacaoBD.getCatalogs();
```

… obter Meta Informação – tabelas, vistas, etc

```
// Classes da biblioteca JDBC
import java.sql.SQLException;
import java.sql.DatabaseMetaData;

// Obter informação sobre tabelas, vistas, etc
String[] tiposEsquemaRelacao = { "TABLE", "VIEW" };
relacao = metaInformacaoBD.getTables(
        // a catalog name; "" retrieves those without a catalog;
        // null means drop catalog name from the selection criteria
        null,
        // a schema name pattern; "" retrieves those without a schema
        "%",
        // a table name pattern
        "%",
        // a list of table types to include; null returns all types
        // Typical types are:
        // "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY",
        // "LOCAL TEMPORARY", "ALIAS", "SYNONYM"
        tiposEsquemaRelacao
        );
...
```

Os resultados dependem do tipo do motor de base de dados utilizado

Equivalência de tipos: Java – SQL

Java method	***SQL Type***
`getInt`	`INTEGER`
`getLong`	`BIG INT`
`getFloat`	`REAL`
`getDouble`	`FLOAT`
`getBignum`	`DECIMAL`
`getBoolean`	`BIT`
`getString`	`VARCHAR`
`getString`	`CHAR`
`getDate`	`DATE`
`getTime`	`TIME`
`getTimestamp`	`TIME STAMP`
`getObject`	*any type*

Acerca de Transacções

Matéria extra

Consultar slides acerca de transacções para mais informação

… Meta Informação – suporte a transacções, etc

- Uma transacção define um conjunto de acções a ser executado como um todo

```
// Classes da biblioteca JDBC
import java.sql.SQLException;
import java.sql.DatabaseMetaData;

// Verificar se o SGBD suporta transacções
boolean existeSuporte;
existeSuporte = metaInformacaoBD.supportsTransactions();

// Verificar se o SGBD suporta transacções
existeSuporte = metaInformacaoBD.supportsTransactionIsolationLevel(
                                                    nivel de isolamento );

// Verificar qual o nível de isolamento de omissão usado pelo SGBD
int nIOmissao = metaInformacaoBD.getDefaultTransactionIsolation();
...
```

Processar (Definir transacção)

```
// Classes da biblioteca JDBC
import java.sql.Connection;
import java.sql.SQLException;

...
// Desligar "commit" automático
conexao.setAutoCommit( false );

// Definir Nivel de Isolamento (tal como definido em java.sql.Connection)*
int nivelIsolamento = Connection.TRANSACTION_SERIALIZABLE
conexao.setTransactionIsolation( nivelIsolamento );

// Exemplo de transacção com 3 directivas SQL
directiva.execute( string com a directiva SQL 1 );
directiva.execute( string com a directiva SQL 2 );
directiva.execute( string com a directiva SQL 3 ); ...

// Consolidar (commit) ou desfazer (rollback) a transacção
conexao.commit(); /* ou conexao.rollback() */
...
```

Connection.TRANSACTION_NONE ou
Connection.TRANSACTION_READ_UNCOMMITTED ou
Connection.TRANSACTION_READ_COMMITTED ou
Connection.TRANSACTION_REPEATABLE_READ ou
Connection.TRANSACTION_SERIALIZABLE

SQLException