Gerenciamento de Dados e Informação

Conectividade - JDBC

Fernando Fonseca
Ana Carolina
Robson Fidalgo

Conectividade

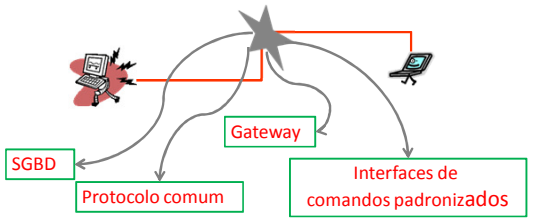
- O surgimento de ambientes com a arquitetura cliente-servidor compostos por diferentes plataformas vindas de diversos fabricantes requer das aplicações a necessidade de comunicar-se com outras aplicações para poderem compartilhar dados e/ou invocar processos que sejam comuns

Conectividade

- Nos primórdios dos SGBD, toda a conectividade a BD era feita por meio de aplicações escritas para trabalhar exclusivamente com um SGBD específico
  - Existiam dezenas de produtos para BD e cada um usava uma linguagem proprietária
  - As aplicações eram dependentes das linguagens dos sistemas de BD
  - Se uma aplicação existente necessitasse se conectar a um BD em outro SGBD, era necessário modificá-la para uma nova linguagem

Conectividade

Acesso a dados existentes
em múltiplas fontes independentes



Uso de SGBD

- Modelo de dados deve abrigar construções existentes nos modelos locais
- O SGBD escolhido deve possuir a funcionalidade necessária para atuar como um SGBD global
  - Multibanco de dados

Uso de Protocolo comum de acesso a dados

- Baseado na padronização do protocolo utilizado na comunicação dos diversos componentes do ambiente distribuído
  - RDA (Remote Database Access) - ISO
  - EDA/SQL (Enterprise Database Access/ SQL) - Information Builders
  - CORBA (Common Object Request Broker Architecture) – OMG
  - ...

Uso de Gateway

- Tradutores de comandos de acesso a dados
- Não possuem funcionalidades de SGBD
- Podem vir acoplados a SGBD
- Normalmente são soluções proprietárias para conectarem fontes de dados específicas

Interfaces de Comandos Padronizados

- SAG - SQL Access Group
  - Padrão de interoperabilidade
    - Qualquer cliente BD pode se comunicar diretamente com qualquer servidor de BD
    - Utilização de formatos de mensagens e protocolos comuns
    - Cria interface SQL com função de chamada (Call Level Interface - CLI)
      - Uma interface procedural para SQL
      - Para SGBD de diversos fabricantes

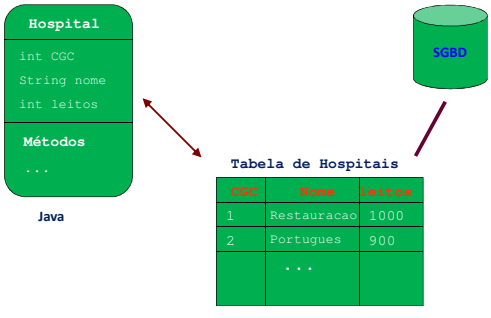
Interfaces de Comandos Padronizados

- Padrão de interoperabilidade (Cont.)
  - Conexão com BD por meio de driver local
  - Preparar solicitações SQL
  - Executar solicitações
  - Recuperar resultados
  - Encerrar conexão
  - Requer o uso de um driver específico para cada SGBD de modo a traduzir as chamadas da aplicação para a linguagem nativa de acesso ao SGBD
    - IDAPI, ODBC, JDBC, ...

Conectividade com Java

- Obtenção
  - Definição de uma camada de serviço de persistência que utiliza SGBD relacionais

Conectando Java a um BD Relacional

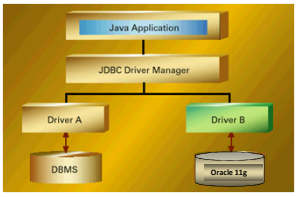


JDBC™ – Java Database Connectivity

- API para acesso a SGBD (Sistemas de Gerenciamento de Banco de Dados) utilizando Java
  - Estabelece uma conexão com o SGBD
  - Envia comandos SQL para o SGBD
  - Processa os resultados
- Protocolo JDBC
  - Define regras de comunicação entre uma aplicação Java e um SGBD
  - Necessidade de um driver para efetivar a comunicação (inserido no CLASSPATH)

Arquitetura de JDBC

- JDBC é composto por três componentes principais: API, Driver Manager e os Drivers JDBC



Tipos de Driver JDBC

- Tipo 1 (*JDBC-ODBC Bridge*)
  Transforma JDBC em ODBC e se utiliza desse último para comunicação com o SGBD
- Tipo 2 (*Driver parcial*)
  Mapeia chamadas JDBC para uma API nativa do SGBD. Precisa de código específico de plataforma além da biblioteca Java

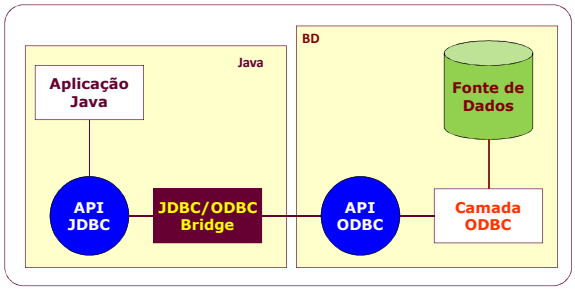
Tipos de Driver JDBC

- Tipo 3 (*Drivers Middleware*)
  Driver puro Java para um servidor middleware que dê suporte a clientes JDBC. Utiliza protocolo independente de SGBD particular
- Tipo 4 (*Direct-to-database*)
  Driver puro Java que permite a conexão com um servidor de banco de dados. Utiliza protocolo específico de um SGBD

Tipo 1: JDBC/ODBC Bridge

- Vem com o J2SE
- Faz acesso JDBC usando drivers ODBC
- Um driver ODBC deve estar disponível na máquina
- Usa código nativo
- Perde em desempenho porque passa por duas camadas (JDBC e ODBC)
- Recomendado apenas para testes ou para uso em aplicações não-críticas

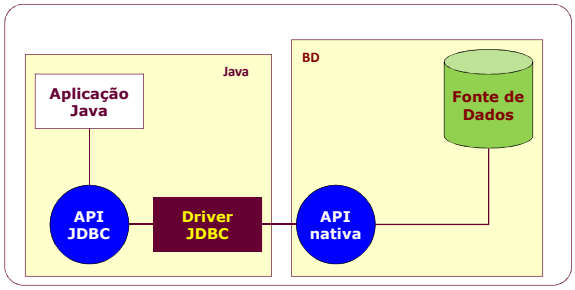
Tipo 1: JDBC-ODBC Bridge



Tipo 2: Parte Java, parte Nativo

- Um driver Tipo 2 converte chamadas à API JDBC em chamadas à API do SGBD
- Mais rápido que a JDBC/ODBC Bridge porque dispensa uma camada

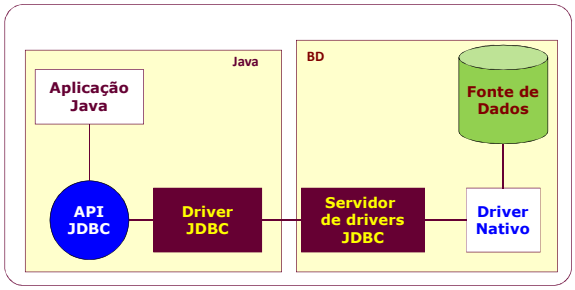
Tipo 2: Parte Java, parte nativo



Tipo 3: Servidor intermediário

- Traduz chamadas à API JDBC para um protocolo de rede independente de SGBD
- As mensagens independentes de SGBD são traduzidas por um servidor intermediário para o protocolo nativo do SGBD
- O servidor intermediário é capaz de se conectar com vários tipos de SGBD
- Arquitetura flexível oferecida por vários fabricantes servidores de aplicações

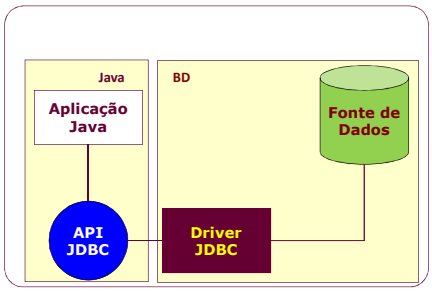
Tipo 3: Servidor intermediário



Tipo 4: Puro Java

- Converte chamadas à API JDBC diretamente para o protocolo de rede usado pelo SGBD
- São extremamente eficientes
- A maioria dos fabricantes fornece drivers tipo 4 junto com seus SGBD (Oracle, Sybase, IBM, Borland)

Tipo 4: Puro Java



Tipos de Drivers JDBC: Resumo

- Os drivers Tipo 1 e 2 são apenas soluções temporárias e rápidas, enquanto soluções puro Java estavam sendo desenvolvidas
- Os drivers tipo 3 e 4 são os recomendados por serem puro Java
- Drivers tipo 3 são mais flexíveis, mas exigem a quebra em camadas
- Drivers tipo 4 fazem acesso direto a SGBD e são os mais simples de usar e instalar

API de JDBC

- Pacote java.sql
  - Interfaces
    - Driver
    - Connection
    - Statement
    - ResultSet
    - CallableStatement
    - PreparedStatement
    - DatabaseMetaData
    - ResultSetMetaData
    - Array
    - Blob
    - Clob
    - Ref

API de JDBC

- Pacote java.sql
  - Interfaces (Cont.)
    - SQLData
    - SQLInput
    - SQLOutput
    - Struct
    - ...

API de JDBC

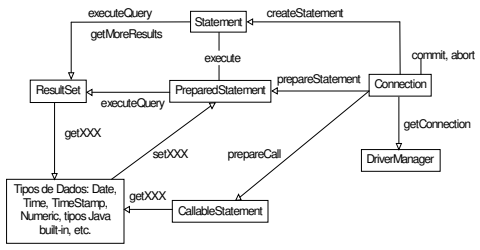
- Classes
  - DriverManager
  - Date
  - Timestamp
  - BatchUpdateException
  - ...

API JDBC Standard Extension

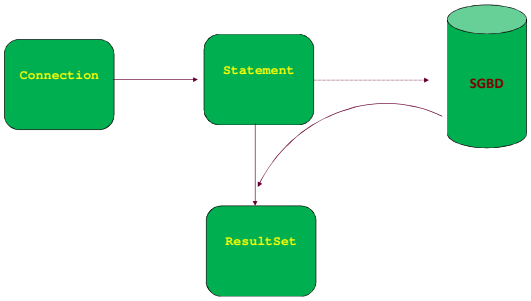
- Interfaces do pacote javax.sql
  - ConnectionEvent
  - ConnectionEventListener
  - ConnectionPoolDataSource
  - CursorMovedEvent
  - CursorMovedListener
  - DataSource
  - PooledConnection
  - RowSet
  - RowSetMetaData
  - ..

API de JDBC

- Pacote java.sql



Conhecendo as Principais Classes



Conhecendo as Principais Classes

- Connection(conexão)
  - isClosed( ) – testa se a conexão está fechada
  - close( ) – fecha a conexão
  - createStatement( ) – cria um comando
- Statement(comando)
  - close( ) – fecha o comando
  - executeQuery(String sql) – executa uma consulta em SQL
  - executeUpdate(String sql) – faz uma atualização de um registro numa tabela

Conhecendo as Principais Classes

- ResultSet (tabela)
  - Tipos
    - forward-only
      Não possui scroll e é insensitive
    - scroll-insensitive
      Não é sensível a mudanças no SGBD
    - scroll-sensitive
      É sensível a mudanças no SGBD

Conhecendo as Principais Classes

- ResultSet (tabela) (Cont.)
  - Habilidades
    - Scrolling (Rolagem)
      Permite ao usuário mover-se para frente e para trás no conjunto de resultados
    - Updatability
      Permite atualizar automaticamente o Banco de Dados, caso o usuário altere dados no conjunto de resultados

Conhecendo as Principais Classes

- ResultSet (tabela) (Cont.)
  - Scrolling
    Possibilita que a movimentação do cursor possa ser realizada em ambos os sentidos, por meio dos métodos
    - boolean previous( )
    - boolean next( )
    - boolean absolute(int row) throws SQLException
    - boolean relative(int rows)

Conhecendo as Principais Classes

- ResultSet (tabela) (Cont.)
  - Tipos de Concorrência
    Uma aplicação pode escolher entre dois tipos de controle de concorrência para um result set
    - Somente leitura
      Não permite atualização do seu conteúdo
    - Atualizável
      updateXXX()

Conhecendo as Principais Classes

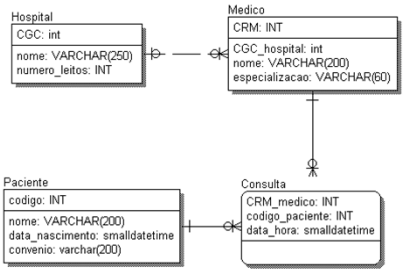
- ResultSet (tabela) (Cont.)
  - getInt(String coluna) – devolve o dado de determinada coluna como inteiro
  - getString(String coluna) - devolve o dado de determinada coluna como string
  - ...

Passos para Acesso ao SGBD

- Instalar um driver JDBC para o SGBD considerado e configurar o ambiente de execução
- Carregar e registrar o driver JDBC
- Estabelecer uma conexão com o SGBD
- Submeter a execução de comandos SQL
- Ler os resultados
- Fechar a conexão

Um BD Exemplo

Um Sistema de Saúde



Registrando Drivers JDBC

- Carregue o(s) driver(s) JDBC
  Class.forName(nome_do_driver)
  - O nome do driver é fornecido pelo provedor (Ex.: Oracle)
    - Exemplo
      - Class.forName("oracle.jdbc.driver.OracleDriver ")

ORACLE 11g

Estabelecendo uma conexão

- Utilize o método getConnection da classe java.sql.DriverManager para abrir uma conexão com o SGBD
- Assinatura de getConnection
  static Connection getConnection(String url, String user,String login) throws SQLException
  - Exemplo de url
    - "jdbc:oracle:thin:@itapissuma.cin.ufpe.br:1521 :dbdisc "

java.sql.Connection

- Assinatura dos Métodos
  Statement createStatement( ) throws SQLException
  void setAutoCommit(boolean autocommit) throws SQLException
  void commit( ) throws SQLException
  void rollback( ) throws SQLException
  void close( ) throws SQLException

Exemplo: Registrando Driver e Abrindo uma Conexão

```
//bloco necessario para abrir uma conexao
try{
 Class.forName(" oracle.jdbc.driver.OracleDriver ");
 Connection con =
        DriverManager.getConnection("jdbc:oracle:thin:
        @itapissuma.cin.ufpe.br:1521:dbdisc ", " user ",    " passwd
");
} catch(ClassNotFoundException ex1){
 ex1.printStackTrace();
} catch(SQLException ex2){
 ex2.printStackTrace();
}
```

Login e *Senha* no Banco de Dados

Executando Comandos SQL

- Utilizar a classe java.sql.Statement para enviar comandos SQL para o SGBD
  - Criar um Statement a partir da conexão
    - Exemplo

      Statement stmt = con.createStatement();

java.sql.Statement

- Assinatura dos Métodos

  int executeUpdate(String sql) throws SQLException
  - Para executar comandos DML - INSERT, UPDATE e DELETE - ou DDL - CREATE, DROP, etc.

  ResultSet executeQuery(String sql) throws SQLException
  - Para executar comandos SELECT

java.sql.ResultSet

- Assinatura dos Métodos

  boolean next( ) throws SQLException
  - Posiciona o cursor na próxima linha
  - Inicialmente o cursor está antes da primeira linha

  String getString(int col) throws SQLException

  String getString(String nomeColuna) throws SQLException

ResultSet:

getXXX métodos

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte	X	x	x	x	x	x	x	x	x	x	x	x	x						
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x						
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x						
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x						
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x						
getDouble	x	x	x	x	x	X	x	x	x	x	x	x	x						
getBigDecimal	x	x	x	x	x	x	X	x	x	x	x	x	x						
getBoolean	x	x	x	x	x	x	x	X	x	x	x	x	x						
getString	x	x	x	x	x	x	x	x	X	x	X	X	x	x	x	x	x	x	x
getBytes														X	X	x			
getDate											x	x	x				X		x
getTime											x	x	x					X	x
getTimestamp											x	x	x				x		X
getAsciiStream											x	x	X	x	x				
getUnicodeStream											x	x	X	x	x				
getBinaryStream														x	x	X			

Aperfeiçoando ResultSet

```
...
stmt = con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
stmt.setFetchSize(   );
rset = stmt.executeQuery( "SELECT * FROM
  HOSPITAL ");
```


Exemplo: Criando uma tabela

```
try {
        stmt = con.createStatement();
       String sql = " CREATE TABLE HOSPITAL ( CGC
              NUMBER, NOME VARCHAR2(40),
        NUMERO_LEITOS NUMBER) ";
        stmt.executeUpdate(sql);
} catch(SQLException se) { se.printStackTrace(); }
 catch(ClassNotFoundException ce) {
        ce.printStackTrace(); }
```

Exemplo: Inserindo Dados em uma Tabela

```
Statement stmt;  //Exemplo de INSERT
try{
 stmt = con.createStatement();
 stmt.executeUpdate(" INSERT INTO Hospital (CGC,
        nome,leitos) VALUES (1,'Restauração',1000) ");
 System.out.println(" Insercao realizada ");
 stmt.close();
} catch(SQLException ex2){
 ex2.printStackTrace(); }
```


Exemplo: Removendo Dados de uma Tabela

```
Statement stmt;  //Exemplo de DELETE
try{
 stmt = con.createStatement();
 stmt.executeUpdate(" DELETE FROM Hospital WHERE
        CGC=1 ");
 System.out.println(" Delecao realizada ");
 stmt.close();
} catch(SQLException ex2){
 ex2.printStackTrace(); }
```


Exemplo: Atualizando Dados em uma Tabela

```
Statement stmt;  //Exemplo de UPDATE
try{
 stmt = con.createStatement();
 stmt.executeUpdate(" UPDATE Hospital SET leitos=500
                WHERE CGC=1 ");
 System.out.println(" Atualizacao realizada");
 stmt.close();
} catch(SQLException ex2){
 ex2.printStackTrace(); }
```


Exemplo: Consultando Dados em uma Tabela

```
Statement stmt;  //Exemplo de SELECT
try{
 stmt = con.createStatement();
 ResultSet rs = stmt.executeQuery("SELECT * FROM
HOSPITAL ");
 while(rs.next()){
  int cgc = rs.getInt(" CGC ");
  String nome = rs.getString(" Nome ");
  int leitos = rs.getInt(" leitos ");
```


Exemplo: Consultando Dados em uma Tabela

```
 Hospital novo = new Hospital(cgc,nome,leitos);
    System.out.println(novo);
 }

rs.close();
 stmt.close();
} catch(SQLException ex2){
 ex2.printStackTrace();
}
```


java.sql.CallableStatement

- Executa chamadas a stored procedures no SGBD

java.sql.PreparedStatement

- Reduz o tempo de execução para comandos SQL que são executados várias vezes seguidas
- Os comandos são pré-compilados pelo SGBD

Exemplo: PreparedStatement

```
Try {  ....
    s = "SELECT * FROM USUARIO WHERE CODIGO
   = ?";
  pstmt = con.prepareStatement(s);
  pstmt.setInt(1,1);
  pstmt.executeQuery();
 ....

} catch(SQLException e)  { }
```


Metadados

- Informação a respeito de dados
  - DatabaseMetadata
    Dados a respeito da base de dados como um todo - estrutura do Banco de Dados
    - Fabricante e versão do SGBD
    - Descrição de tabelas (nomes das colunas, chaves,...)
  - Informações a respeito de características com suporte (ou não) pelo sistema

DatabaseMetaData

- Informações a respeito de características (Cont.)
  - Aproximadamente 150 métodos
  - Muitos deles retornam ResultSets
  - Não há padronização nas respostas dadas por cada implementação
  - Alguns drivers não dão suporte a todos os métodos

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


ResultSetMetaData

- Informação a respeito de dados (Cont.)
  - ResultSetMetadata
    Informações a respeito de tipos e propriedades das colunas de um ResultSet
    - Número de colunas retornadas
    - Nomes e tipos de cada coluna
  - Gerado a partir de um ResultSet, resultado de uma consulta

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


ResultSetMetaData

- Exemplo: Listar o nome de todas as colunas

```
...
for (int i=0;i<rsmd.getColumnCount();i++){
     nome=rsmd.getColumnName(i);
     System.out.println(nome);
}
```


Atualizações em Lote

- Permite que várias operações de update possam ser submetidas juntas para processamento do SGBD, ao invés de sozinhas
  - Exemplo

```
try { con.setAutoCommit( false );
    stmt.clearBatch();
    stmt.addBatch( sUpdate1 );
    stmt.addBatch( sUpdate2 );
    stmt.addBatch( sUpdate3 );
    int[ ]  conta = stmt.executeBatch();
} catch(SQLException e) {}
```

Tipos de Dados Avançados (SQL3)

- Large Objects (LOB)
  - Binary Large Objects (BLOB)
  - Character Large Objects (CLOB)
- Locator
  - Interface para associar um evento SAX com a localização de um documento (Processamento de XML)

java.sql.Blob

- Assinatura dos Métodos
  - InputStream getBinaryStream( ) throws SQLException
  - byte[ ] getBytes(long pos, int length) throws SQLException
  - long length( ) throws SQLException
  - ....

Exemplo: Blob

```
try {
    PreparedStatement getFotoStmt =
    con.prepareStatement("SELECT IMAGEM FROM IMAGENS
    WHERE CODIGO = ?");
    getFotoStmt.setInt(1, 1);
    rs = getFotoStmt.executeQuery();
    if (rs.next()) {
          fotoBlob = rs.getBlob("IMAGEM");
          byte teste[ ] = fotoBlob.getBytes(1,
    (int)fotoBlob.length());
          if (teste != null) {
          result =
    (Image)Toolkit.getDefaultToolkit().createImage(teste);  }
    } else System.out.println("erro");
    } catch (SQLException ex) { ex.printStackTrace();}
```


java.sql.Clob

- Assinatura dos Métodos
  - InputStream getAsciiStream( ) throws *SQLException*
    Reader getCharacterStream( ) throws SQLException
    String getSubString(long pos, int length) throws SQLException
  - ...

Transações

- Preservam a consistência dos dados do SGBD
- Propriedades das transações (ACID):
  - Atomicidade
  - Consistência
  - Isolamento
  - Durabilidade

Implementando Transações

- Propriedades auto-commit de Connection
  - true - realiza commit após cada operação executada pelo Statement ser completada(default)
  - false - não realiza commit automaticamente

Implementando Transações

- Métodos
  - setAutoCommit(boolean autoCommit)
    - Define o modo de commit
  - commit( )
    - Confirma a transação
  - rollback( )
    - Cancela a transação

Exemplo de Transação com JDBC

```
Try {  ...
  con.setAutoCommit(false);
  stmt = con.createStatement();
  String sql1 = "INSERT INTO HOSPITAL (5,'Neuro',800)";
  String sql2 = "INSERT INTO USUARIO (6,'Hope',750)";
  stmt.executeUpdate(sql1);
  stmt.executeUpdate(sql2);
  con.commit();
} catch(SQLException se) {
      try { con.rollBack(); }
      catch(SQLException e){e.printStackTrace(); }
      e.printStackTrace();
}
```


Transações - Isolamento

- Níveis de isolamento
  - READ COMMITTED
    - Nível default de alguns SGBD
    - Modificações realizadas por outras transações só se tornam visíveis depois do commit
    - Duas transações concorrentes podem modificar o mesmo registro

READ COMMITTED

- Exemplo
  - a=0;

T1	T2	comentário
Update t set a=1;		
	select a from t1;	Retorna 0
commit		
	select a from t1;	Retorna 1
	update t set a=2;	
Select a from t1;		Retorna 1
	Commit	
Select a from t1;		Retorna 2

Transações - Isolamento

- Níveis de isolamento (Cont.)
  - SERIALIZABLE
    - Nível mais estrito de isolamento
    - Menor nível de concorrência
    - Modificações realizadas por outras transações não são visíveis
    - Sempre se tem a visão do banco do início da transação
    - Duas transações concorrentes não podem modificar o mesmo registro

SERIALIZABLE

- Exemplo
  - a=0;

T1	T2	Comentário
Update t set a=1;		
	select a from t1;	Retorna 0
commit		
	select a from t1;	Retorna 0
Begin work		Inicia nova transação
Update t set a=3		Lock exclusivo em a
	update t set a=2;	T2 trava esperando o lock
commit		Rollback automático em T2

Transações e JDBC

```
con.setAutoCommit(false); // Inicia a transação
con.setTransactionIsolation
   (con.TRANSACTION_READ_COMMITTED);
      (...)
con.commit();
      (...)
}catch(SQLException   ){
     con.rollback(); // SEMPRE termine a transação!
}
```