

Gerenciamento de Dados e Informação

Fernando Fonseca
Ana Carolina
Robson Fidalgo



Cin.ufpe.br

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

Interoperabilidade



2

Conectividade

- Nos primórdios dos SGBD, toda a conectividade a BD era feita através 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



3

Conectividade

- Acesso a dados existentes em múltiplas fontes independentes pode ser feito
 - **Através de SGBD**
 - Abrigar construções existentes nos modelos locais
 - Deve ser utilizado por um SGBD que possua a funcionalidade requerida para atuar como um SGBD global
 - **Multibanco de dados**



4

Conectividade

- Acesso a dados existentes em múltiplas fontes independentes (Cont.)
 - **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
 - ...



5

Conectividade

- Acesso a dados existentes em múltiplas fontes independentes (Cont.)
 - **Uso de gateway**
 - Tradutores de comandos de acesso a dados
 - Não possuem funcionalidades de SGBD
 - Podem vir acoplados a SGBD
 - **Interfaces de comandos padronizados**
 - IDAPI, ODBC, JDBC, ...



6

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)



7

Interfaces de Comandos Padronizados

- Padrão de interoperabilidade (Cont.)
 - Uma CLI é, simplesmente, uma interface procedural para SQL
 - Uma CLI 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



8

Interfaces de Comandos Padronizados

- Conjunto comum de API (**Application Programming Interface**)
 - Para SGBD de diversos fabricantes
 - Conexão com BD através de driver local
 - Preparar solicitações SQL
 - Executar solicitações
 - Recuperar resultados
 - Encerrar conexão



9

ODBC

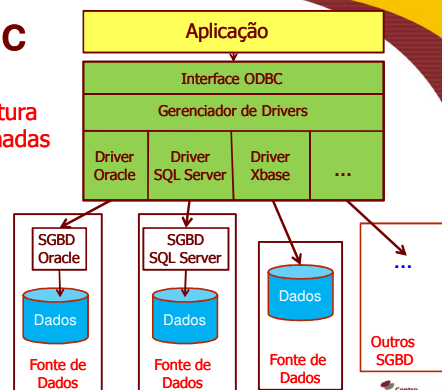
- Open Database Connectivity
 - Microsoft
 - Implementações de terceiros - Unix
 - Componentes
 - API
 - Funções para gerenciar os drivers
 - Drivers



10

ODBC

Arquitetura em Camadas



11

ODBC - Arquitetura

- Aplicação (Camada 1)
 - Um programa que chama funções ODBC para interagir com fontes de dados
 - Uma planilha Excel
 - Uma ferramenta de relatório
 - Um documento Word
 - Uma aplicação web
 - ...



12

ODBC - Arquitetura

- Gerenciador de Drivers (Camada 2)
 - Carrega os drivers, direcionando as chamadas de funções ao driver ODBC correto
 - Provê acesso aos drivers ODBC
 - Intercepta as chamadas da aplicação para o banco de dados
 - Permite que múltiplos drivers estejam ativos simultaneamente
 - Carrega e descarrega drivers, verifica status e administra múltiplas conexões entre aplicações e fontes de dados



13

ODBC - Arquitetura

- Drivers (Camada 3)
 - É uma biblioteca de funções que processa as solicitações ODBC, enviando instruções SQL específicas para cada fonte de dados
 - Traduz cada requisição para o formato apropriado do SGBD (Mediador)
 - Conecta uma fonte de dados, traduz comandos SQL e os submete à fonte de dados, recupera informações da referida fonte e retorna dados para a aplicação
 - Se a fonte é Xbase (não usa SQL), o driver também deve processar os comandos SQL



14

ODBC - Arquitetura

- Fontes de Dados (Camada 4)
 - Trata-se dos dados propriamente ditos
 - Cada fonte de dados deve possuir um driver apropriado para que a intermediação possa ser estabelecida



15

ODBC

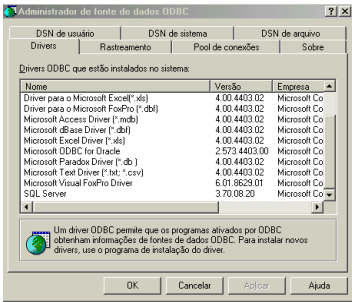
Configuração




16

ODBC

Configuração





Nome	Versão	Empresa
Driver para o Microsoft Excel® (.xls)	4.00.4403.02	Microsoft Co
Driver para o Microsoft FoxPro® (.dbf)	4.00.4403.02	Microsoft Co
Microsoft Access Driver® (.mdb)	4.00.4403.02	Microsoft Co
Microsoft dBase Driver® (.dbf)	4.00.4403.02	Microsoft Co
Microsoft Excel Driver® (.xls)	4.00.4403.02	Microsoft Co
Microsoft ODBC for Oracle	2.57.3.4403.00	Microsoft Co
Microsoft Paradox Driver® (.db)	4.00.4403.02	Microsoft Co
Microsoft Text Driver® (.txt; .csv)	4.00.4403.02	Microsoft Co
Microsoft Visual FoxPro Driver	6.01.8829.01	Microsoft Co
SQL Server	3.70.09.20	Microsoft Co



17

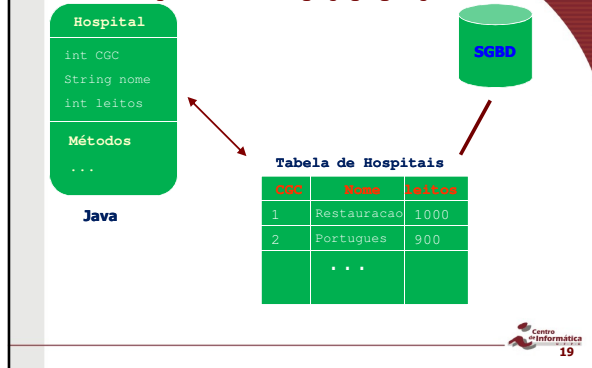
Conectividade com Java

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

18

Conectando Java a um BD Relacional

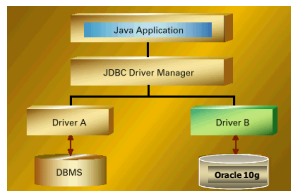


JDBCTM – Java Database Connectivity

- API para acesso a SGBD (Sistemas de Gerenciamento de Banco de Dados) com 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)
- Centro de Informática 20

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
- Centro de Informática 22

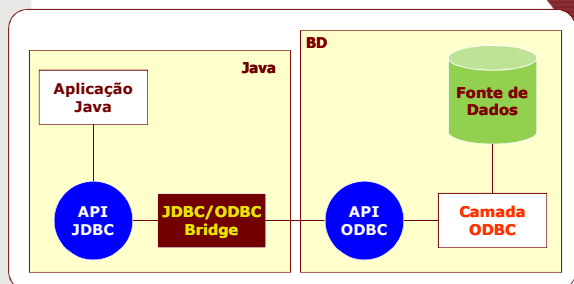
Tipos de Driver JDBC

- 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
- Centro de Informática 23

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
- Centro de Informática 24

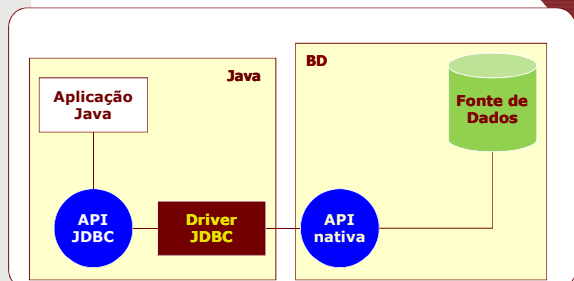
Tipo 1: JDBC-ODBC Bridge



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

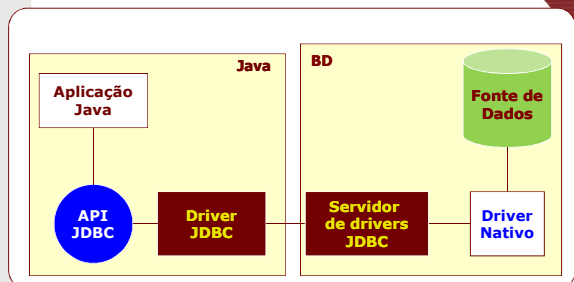
Tipo 2: Parte Java, parte nativo



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

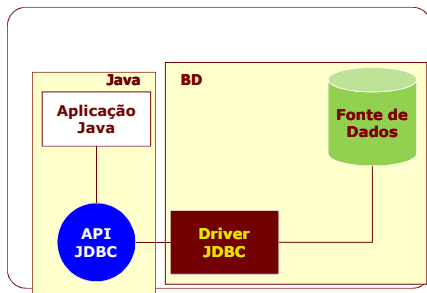
Tipo 3: Servidor intermediário



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

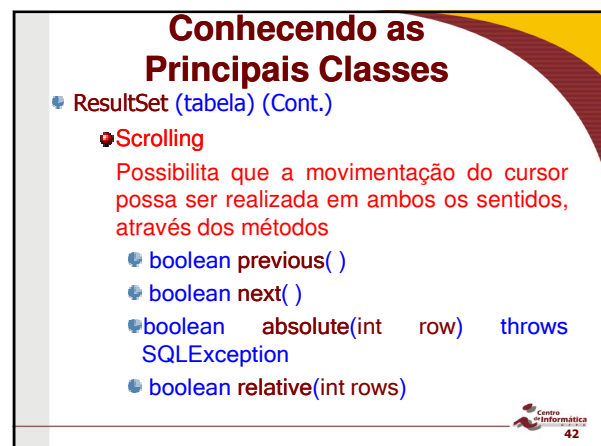
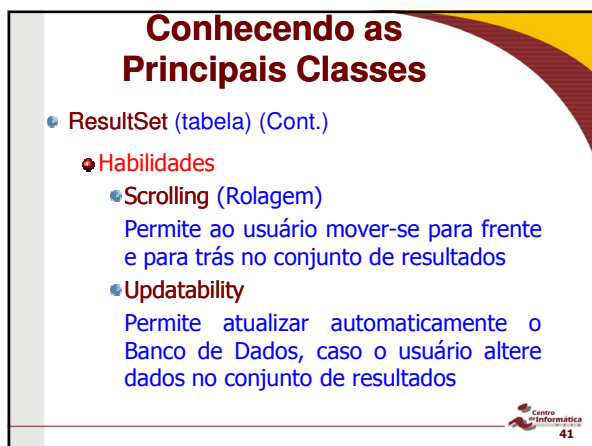
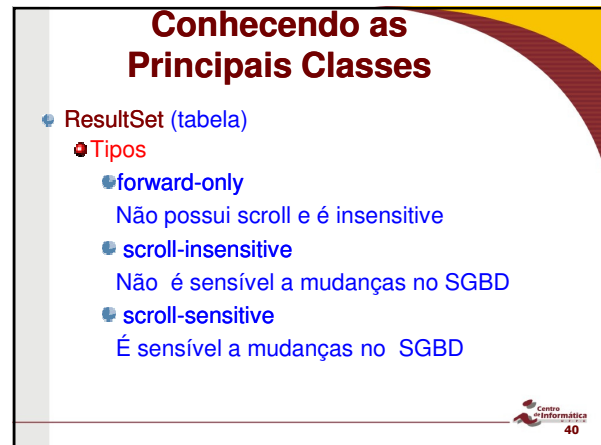
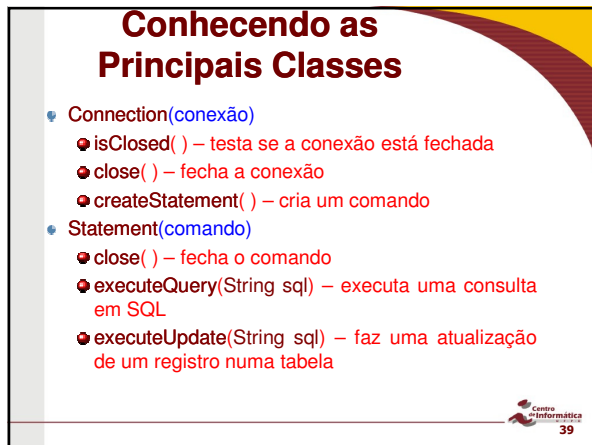
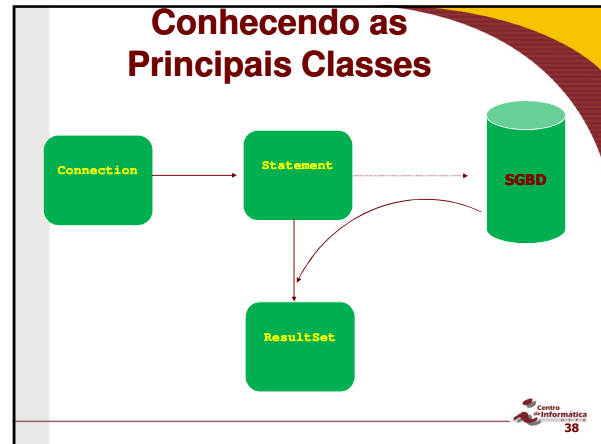
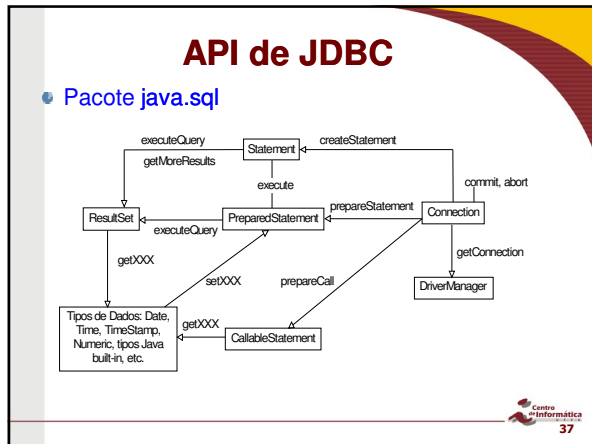
- Pacote java.sql
 - Interfaces (Cont.)
 - SQLData
 - SQLInput
 - SQLOutput
 - Struct
 - ...

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
 - ..



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()



43

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
 - ...



44

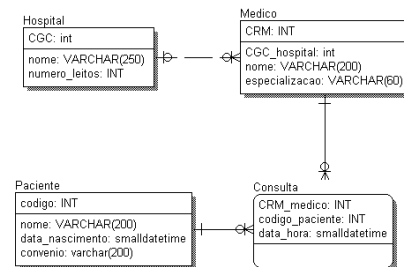
Passos para Acesso ao SGBD

- 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



45

Um BD Exemplo



46

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
10g



47

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 "




48

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
```

 49

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

 50

Executando Comandos SQL

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

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


 51

java.sql.Statement

- Assinatura dos Métodos

```
int executeUpdate(String sql) throws
SQLException
ResultSet executeQuery(String sql) throws
SQLException
```

- ❶ Para executar comandos DML - INSERT, UPDATE e DELETE - ou DDL - CREATE, DROP, etc.
- ❷ Para executar comandos SELECT

 52


java.sql.ResultSet

- Assinatura dos Métodos

```
boolean next( ) throws SQLException
```

- ❶ Posiciona o cursor na próxima linha
- ❷ Inicialmente o cursor está antes da primeira linha


```
String getString(int col) throws SQLException
String getString(String nomeColuna) throws
SQLException
```

 53

ResultSet:

getXXX métodos

	T	S	I	R	F	D	D	C	V	L	V	L	D	T
	M	N	A	T	A	O	U	C	H	O	A	O	A	I
	T	E	L	E	B	E	R	C	A	R	C	H	A	R
	T	N	R	T	E	L	C	A	R	C	H	A	R	P
getBytes	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	
getByte									X	X	X			
getDate													X	
getTime														X
getTimeStamp														X
getAsciiStream									X	X	X	X	X	
getUnicodeStream									X	X	X	X	X	
getBinaryStream									X	X	X	X	X	

 54

Aperfeiçoando ResultSet

```
...
stmt = con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
stmt.setFetchSize(25);
rset = stmt.executeQuery( "SELECT * FROM
HOSPITAL ");
```



55

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(); }
```



56

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(); }
```



57

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(); }
```



58

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(); }
```



59

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 ");
```



60

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();
}
```



61

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



62

Exemplo: PreparedStatement

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



63

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



64

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();
```



65

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();
```



66

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);
}
```



67

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) {}
```



68

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)



69

java.sql.Blob

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



70

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();}
```



71

java.sql.Clob


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



72

Transações


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



73

Implementando Transações

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



74

Implementando Transações

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




75

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();
}
    
```



76

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



77

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



78

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;		
commit	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 e){
    con.rollback(); // SEMPRE termine a transação!
}
  
```