

Programação Orientada a Objetos

Objetos de Acesso a Dados e introdução a
RDBMS

www.comp.ita.br/~pauloac/ces22/

Persistência de Dados

- Manter dados de forma não-volátil é um problema quase onipresente em programas estruturados ou OO
- Quando os dados correspondem a grande volume de bytes, a abordagem de utilizar arquivos simples para armazená-los pode ser ineficiente em termos de uso de espaço e tempo de recuperação
- Desenvolvedores de programas estruturados perceberam isso a muito tempo e por isso criaram sistemas para tratar o armazenamento e recuperação de dados de forma eficiente. São os chamados gerenciadores de banco de dados ou (Database management systems, DBMS)

DBMS

- Os primeiros DBMS não eram relacionais, mas baseados em conceitos de bancos hierárquicos.
- Os banco de dados relacionais (RDBMS) foram e ainda são muito bem sucedidos.
- Banco de dados orientados a objetos também são usados e vários estão disponíveis (Cache, Db4o,etc)
- Outros padrões de DBMS também estão disponíveis como document store, key-value, Graph e outros
- Esses assuntos serão aprofundados na disciplina CES-30, Técnicas de Banco de Dados
- Frameworks para mapeamento de DBMS não OO para programas OO são muitos comuns, especialmente Object Relational Mapping. Esses assuntos e outros serão estudados na disciplina Fundamentos de Engenharia de Software

DBMS 2

- Aqui, estudaremos um padrão de projeto para acessar dados através de objetos mantendo o conceito do objeto desacoplado da implementação da sua persistência em arquivos (flat files) ou em DBMS
- Em seguida, faremos um breve introdução ao acesso de RDBMS através de Java (JDBC) e usaremos o padrão de projeto para criar a persistência em RDBMS

Padrão de Projeto Data Access Object (DAO)

- **Contexto**

- Acesso a dados depende da fonte dos dados. Acesso a armazenamento permanente, como banco de dados, varia bastante dependendo do tipo de armazenamento (relational databases, object-oriented databases, flat files, etc.) e também da implementação dada pelo fabricante.

- **Problema**

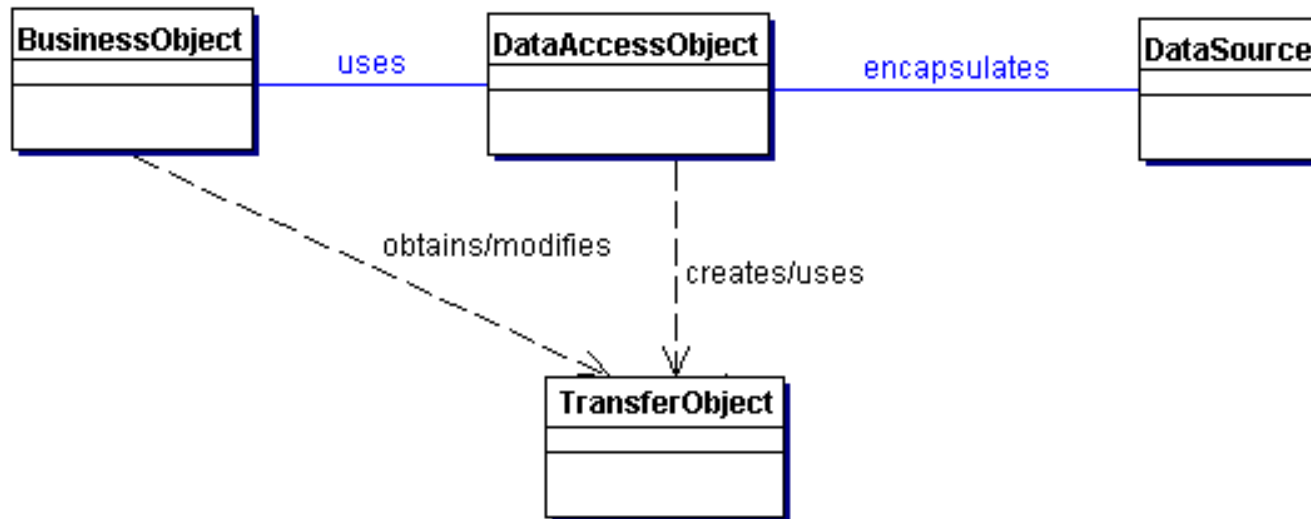
- Código que depende de características específicas da fonte de dados fica junto a lógica de acesso aos dados e da lógica de negócio. Isto torna mais difícil modificar ou substituir uma fonte de dados da aplicação.

Solution: DAO

- **Use a Data Access Object (DAO) to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data.**
 - Each DAO implementation deals with the access mechanism required to work with the data source.
 - The data source could be a persistent store like an RDBMS, an external service like a B2B exchange, a repository like an LDAP database, or a business service accessed via CORBA Internet Inter-ORB Protocol (IIOP) or low-level sockets.
 - The DAO completely hides the data source implementation details from its clients.
 - The business component that relies on the DAO uses the simpler interface exposed by the DAO for its clients.
 - Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, this pattern **allows the DAO to adapt to different storage schemes without affecting its clients** or business components.
 - Essentially, the **DAO acts as an adapter** between the component and the data source.

DAO: Data Access Object

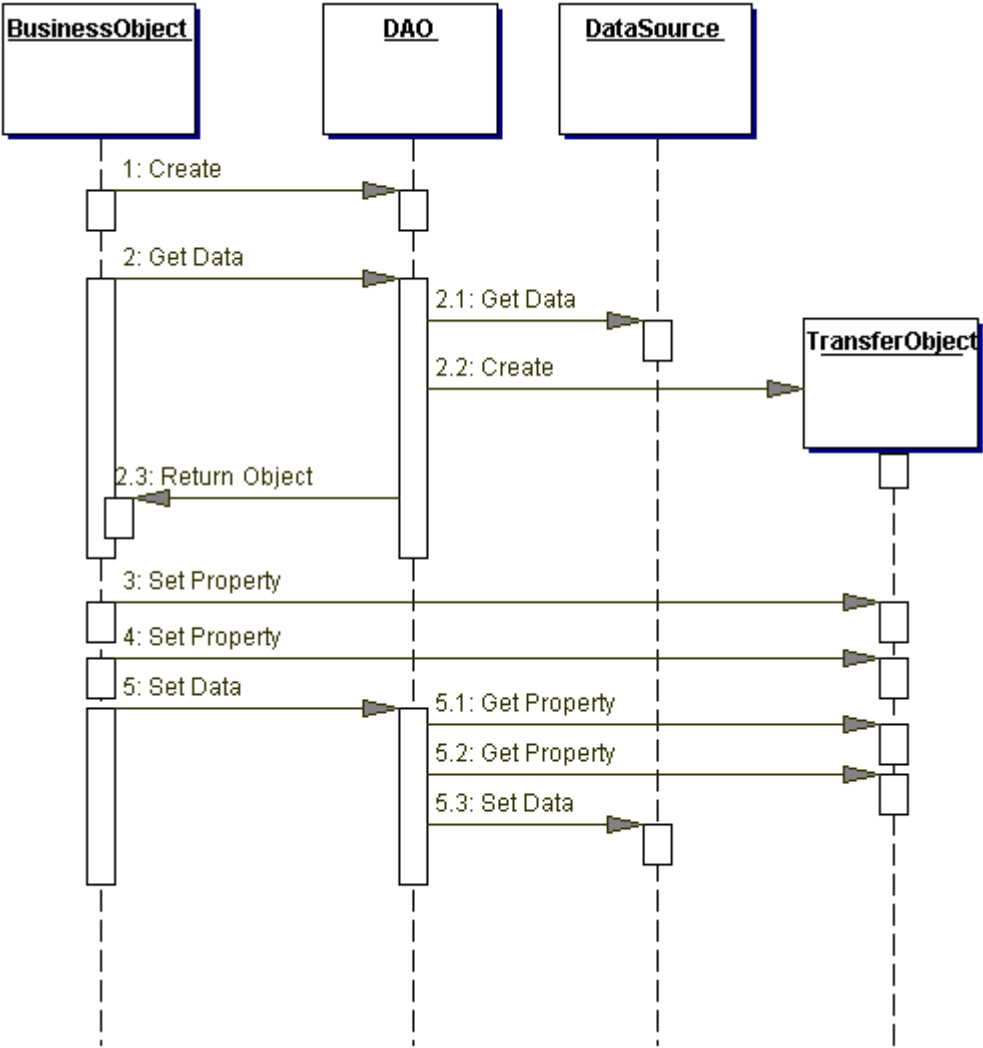
- Estrutura



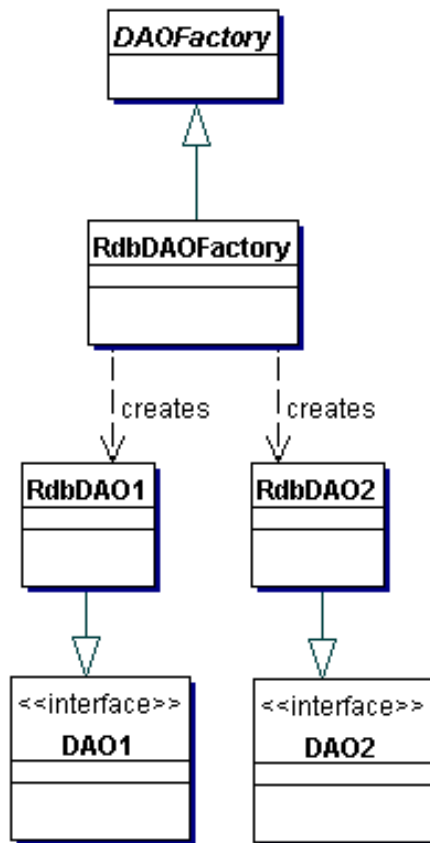
Componentes

- **BusinessObject**
 - The BusinessObject represents the data client. It is the object that requires access to the data source to obtain and store data. A BusinessObject may be implemented as a session bean, entity bean, or some other Java object, in addition to a servlet or helper bean that accesses the data source.
- **DataAccessObject**
 - The DataAccessObject is the primary object of this pattern. The DataAccessObject abstracts the underlying data access implementation for the BusinessObject to enable transparent access to the data source. The BusinessObject also delegates data load and store operations to the DataAccessObject.
- **DataSource**
 - This represents a data source implementation. A data source could be a database such as an RDBMS, OODBMS, XML repository, flat file system, and so forth. A data source can also be another system (legacy/mainframe), service (B2B service or credit card bureau), or some kind of repository (LDAP).
- **TransferObject**
 - This represents a Transfer Object used as a data carrier. The DataAccessObject may use a Transfer Object to return data to the client. The DataAccessObject may also receive the data from the client in a Transfer Object to update the data in the data source.

Diagrama de Seqüência - DAO



DAO e Abstract Factory

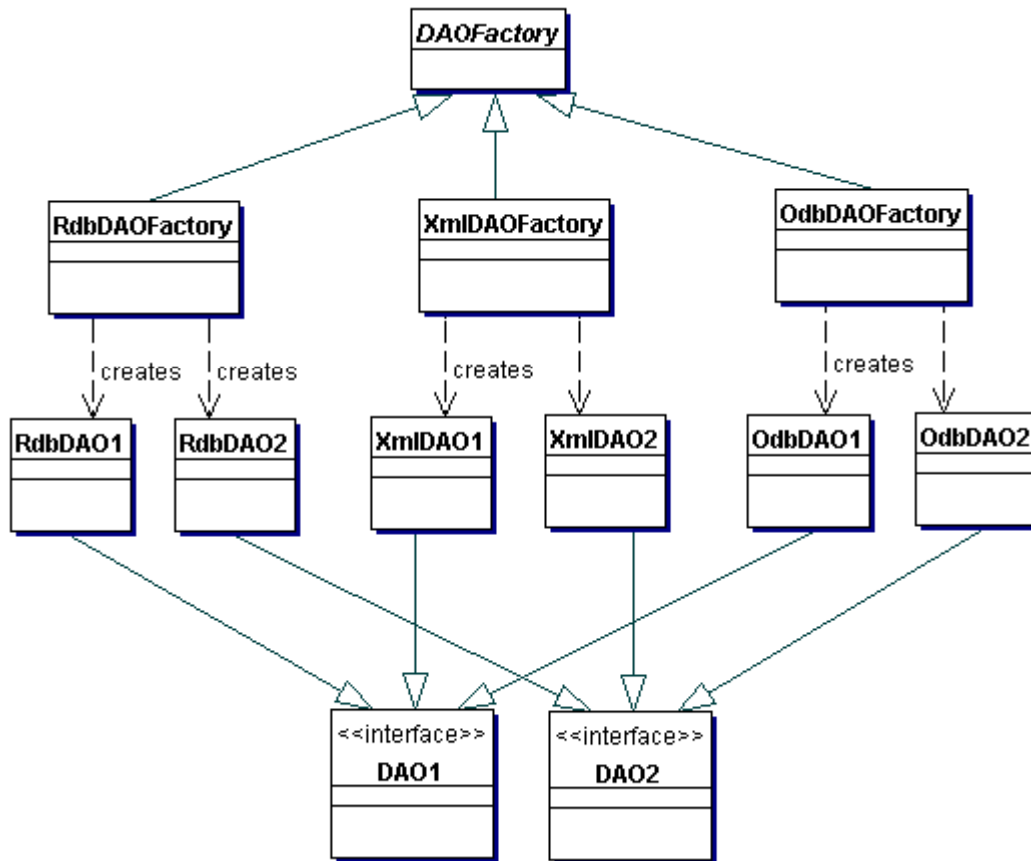


Exemplo:

Um DAOFactory é herdado por uma fábrica específica para um banco de dados Relacional(RdbDAOFactory).

Esta fábrica cria as várias instâncias que implementam as interfaces necessárias (DAO1 e DAO2)

Múltiplas implementações de DAOFactory



Exemplo:

Um DAOFactory é herdado por várias classes que implementam mecanismos acessos a diferentes Fontes de Dados (Rdb, XML e Odb)

Cada fábrica cria as várias instâncias que implementam as interfaces necessárias (DAO1 e DAO2)

Exemplo: DAO

- Problema: Criar um conjunto de classes capazes de armazenar e recuperar dados de autores de livros de uma editora.
- Considere como dados de autores apenas (id, first name e last name)
- Utilize o padrão Data Access Object

```
//Interface DAO

package DAO;

public interface AuthorDAO {
    public Authorsto getAuthors();
}
```

AuthorsDAO.java

```
package DAO;

public class AuthorTO {
    private int id;
    private String firstName;
    private String lastName;

    public AuthorTO(int id,String first,  String last) {
        firstName = first;
        this.id = id;
        lastName = last;
    }
    public String toString() {
        return id+"\t"+ firstName+"\t"+lastName;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
}
```

AuthorTO.java

```
package DAO;
import java.util.Iterator;
import java.util.Vector;

public class AuthorTO extends Vector<AuthorTO> {

    public String toString () {
        StringBuffer str=new StringBuffer();
        str.append("AUTHORID FIRSTNAME LASTNAME\n");
        for (Iterator iter = this.iterator(); iter.hasNext();) {
            AuthorTO element = (AuthorTO) iter.next();
            str.append(element);
            str.append("\n");
        }
        return str.toString();
    }
}
```

AuthorsTO.java

AuthorView.java

```
//      Displaying the contents of the authors table.
import java.awt.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

public class Authorsview extends JFrame {

    // constructor connects to database, queries database, processes
    // results and displays results in window
    public Authorsview()
    {
        super( "Authors Table of Books Database" );
        DAOFactory fac=DAOFactory.getDAOFactory(DAOFactory.FIREBIRD);
        AuthorDAO authors=fac.getAuthorDAO();
        AuthorsTO authorsTO=authors.getAuthors();
        // set up GUI and display window
        JTextArea textArea = new JTextArea( authorsTO.toString() );
        Container container = getContentPane();

        container.add( new JScrollPane( textArea ) );

        setSize( 300, 100 ); // set window size
        setVisible( true ); // display window

    } // end DisplayAuthors constructor

    // launch the application
    public static void main( String args[] )
    {
        Authorsview window = new Authorsview();
        window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }

} // end class
```

Vantagens e Desvantagens

- Desvantagens
 - Complexidade: Havia 1 classe e agora há 3 classes e 1 interface
- Vantagens
 - Desacoplamento entre o cliente e o acesso ao banco de dados
 - Apenas 1 classe é dependente do banco de dados
 - Facilita o reuso
 - Outras classes que desejarem acessar dados dos autores podem fazê-lo criando uma instância do DAO

Vantagens e Desvantagens

- Vantagens
 - Facilita a portabilidade
 - Portar o programa para outro banco de dados (SQL Server, Oracle, etc.) envolve a criação de apenas 1 classe que implemente AuthorDAO e instanciar a implementação correta (pode-se usar o DP Factory, para facilitar esta instanciação)

DAOFactory

```
// Abstract class DAO Factory
public abstract class DAOFactory {
// List of DAO types supported by the factory
public static final int HSQLDB = 1;
public static final int ORACLE = 2;
public static final int SYBASE = 3; ...
// There will be a method for each DAO that can be
// created. The concrete factories will have to
// implement these methods.
public abstract CustomerDAO getAuthorsDAO();

public static DAOFactory getDAOFactory( int ds) {
switch (whichFactory) {
    case FIREBIRD: return new FireBirdDAOFactory();
    case ORACLE : return new OracleDAOFactory();
    case SYBASE : return new SybaseDAOFactory();
    ...
    default : return null;
}
}
}
```

FileDAOFactory

```
// File concrete DAO Factory implementation

public class FileDAOFactory extends DAOFactory {

    AuthorDAO getAuthorDAO() {
        // AuthorDAOImpl implements AuthorDAO
        return new AuthorDAOImpl();
    }
}
```

Authors

```
package DAO;

//      Displaying the contents of the authors table.
import java.awt.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

public class Authors extends JFrame {

    // constructor connects to database, queries database, processes
    // results and displays results in window
    public Authors()
    {
        super( "Authors Table of Books Database" );
        DAOFactory fac=DAOFactory.getDAOFactory(DAOFactory.FIREBIRD);
        AuthorDAO authors=fac.getAuthorDAO();
        AuthorSTO authorsTO=authors.getAuthors();
        // set up GUI and display window
        JTextArea textArea = new JTextArea( authorsTO.toString() );
        Container container = getContentPane();

        .....
        .....
        .....
        .....
        .....
        .....
    }
}
```

Exercício

- Altere o programa para utilizar persistência em Properties files. (java.util.Properties).
- Crie uma Factory para properties Files e uma implementação de AuthorDAO para properties files. Uso de Properties:

```
Properties prop=new Properties();  
Prop.load(new FileInputStream(...));  
prop.getProperty("size");  
prop.getProperty("authorID1");  
prop.getProperty("firstName1");  
prop.getProperty("lastName1");
```

Conectividade com Banco de Dados em Java através de JDBC

Outline

- 1 Introdução
- 2 Modelo de Banco de Dados Relacional
- 3 Relational Database Overview: The `books` Database
- 4 SQL
 - 4.1 Basic `SELECT` Query
 - 4.2 `WHERE` Clause
 - 4.3 `ORDER BY` Clause
 - 4.4 Merging Data from Multiple Tables: `INNER JOIN`
 - 4.5 `INSERT` Statement
 - 4.6 `UPDATE` Statement
 - 4.7 `DELETE` Statement
- 5 Acessando o Banco de Dados `books` no Firebird
- 6 Modificando Databases com JDBC
 - 6.1 Connecting to and Querying a Database
 - 6.2 Querying the `books` Database
- 7 **Referências sobre JDBC**

1 Introduction

- Database
 - Collection of data
- DBMS
 - Database management system
 - Stores and organizes data
- SQL
 - Relational database
 - Structured Query Language

1 Introduction (Cont.)

- RDBMS
 - Relational database management system
 - Firebird
 - firebird.sourceforge.net
- JDBC
 - Java Database Connectivity
 - JDBC driver

2 Relational-Database Model

- Relational database
 - Table
 - Rows, columns
 - Primary key
 - Unique data
- SQL statement
 - Query

2 Relational-Database Model (Cont.)

	Number	Name	Department	Salary	Location
Row	23603	Jones	413	1100	New Jersey
	24568	Kerwin	413	2000	New Jersey
	34589	Larson	642	1800	Los Angeles
	35761	Myers	611	1400	Orlando
	47132	Neumann	413	9000	New Jersey
	78321	Stephens	611	8500	Orlando
		Primary key		Column	

Fig. 1 **Employee** table sample data.

2 Relational-Database Model (Cont.)

Department	Location
413	New Jersey
611	Orlando
642	Los Angeles

Fig. 2 Result of selecting distinct **Department** and **Location** data from the **Employee** table.

3 Relational Database

Overview: The books Database

- Sample books database
 - Four tables
 - authors, publishers, authorISBN and titles
 - Relationships among the tables

3 Relational Database Overview: The books Database (Cont.)

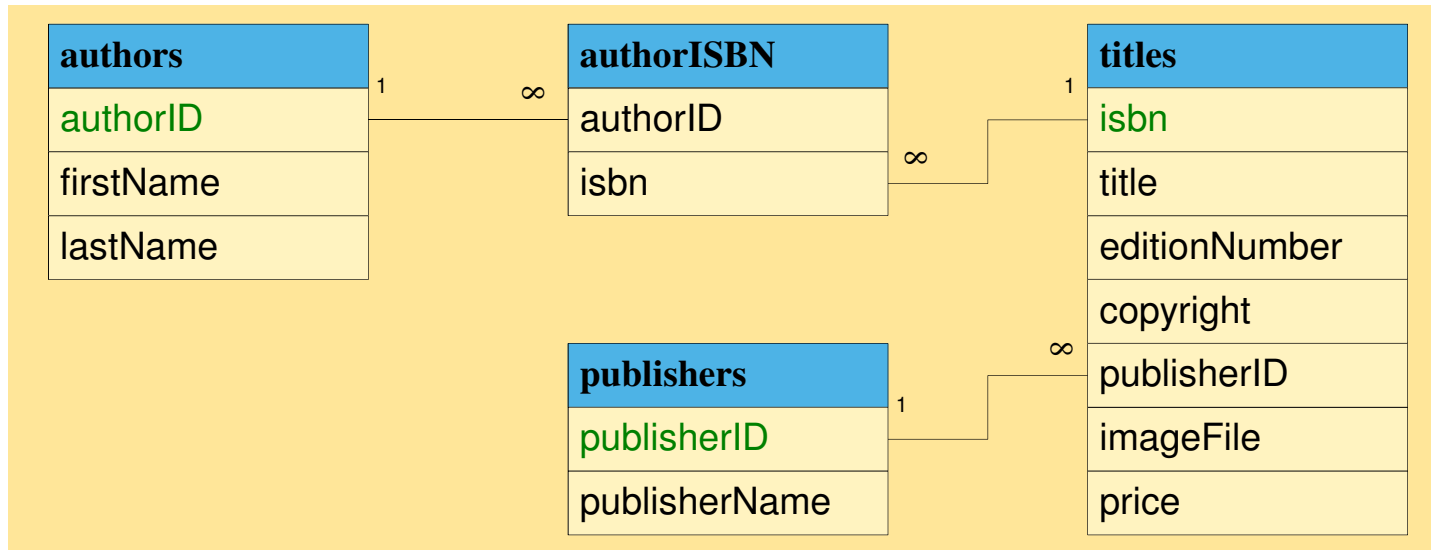


Fig. 11 Table relationships in **books**.

3 Relational Database Overview: The books Database (Cont.)

Column	Description
authorID	Author's ID number in the database. In the books database, this integer column is defined as <i>autoincremented</i> . For each row inserted in this table, the database automatically increments the authorID value to ensure that each row has a unique authorID. This column represents the table's primary key.
firstName	Author's first name (a string).
lastName	Author's last name (a string).

Fig. 23.3 authors table from books.

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 23.4 Sample data from the authors table.

3 Relational Database Overview: The books Database (Cont.)

Column	Description
publisherID	The publisher's ID number in the database. This autoincremented integer is the table's primary key.
publisherName	The name of the publisher (a string).
Fig. 23.5	publishers table from books.

publisherID	publisherName
1	Prentice Hall
2	Prentice Hall PTG
Fig. 6	Data from the publishers table.

3 Relational Database Overview: The books Database (Cont.)

Column	Description
isbn	ISBN of the book (a string). The table's primary key.
title	Title of the book (a string).
editionNumber	Edition number of the book (an integer).
copyright	Copyright year of the book (a string).
publisherID	Publisher's ID number (an integer). A foreign key to the publishers table.
imageFile	Name of the file containing the book's cover image (a string).
price	Suggested retail price of the book (a real number). [Note: The prices shown in this book are for example purposes only.]
Fig. 23.7	titles table from books.

3 Relational Database Overview: The books Database (Cont.)

isbn	title	edition- Number	copy- right	publish- erID	imageFile	price
0130895725	C How to Program	3	2001	1	chtp3.jpg	74.95
0130384747	C++ How to Program	4	2002	1	cpphtp4.jpg	74.95
0130461342	Java Web Services for Experienced Programmers	1	2002	1	jwsfep1.jpg	54.95
0131016210	Java How to Program	5	2003	1	jhtp5.jpg	74.95
0130852473	The Complete Java 2 Training Course	5	2002	2	javactc5.jpg	109.95
0130895601	Advanced Java 2 Platform How to Program	1	2002	1	advjhtp1.jpg	74.95

Fig. 23.8 Sample data from the titles table of books.

3 Relational Database Overview: The books Database (Cont.)

Column	Description
authorID	The author's ID number, a foreign key to the authors table.
isbn	The ISBN for a book, a foreign key to the titles table..

Fig. 23.9 authorISBN table from books.

authorID	isbn	authorID	isbn
1	0130895725	2	0139163050
2	0130895725	3	0130829293
2	0132261197	3	0130284173
2	0130895717	3	0130284181
2	0135289106	4	0130895601

Fig. 10 Sample data from the authorISBN table of books .

4 SQL

- SQL overview
- SQL keywords

SQL keyword	Description
SELECT	Retrieves data from one or more tables.
FROM	Tables involved in the query. Required in every SELECT.
WHERE	Criteria for selection that determine the rows to be retrieved, deleted or updated.
GROUP BY	Criteria for grouping rows.
ORDER BY	Criteria for ordering rows.
INNER JOIN	Merge rows from multiple tables.
INSERT	Insert rows into a specified table.
UPDATE	Update rows in a specified table.
DELETE	Delete rows from a specified table.

Fig. 12 SQL query keywords.

4.1 Basic SELECT Query

- Simplest form of a SELECT query
 - **SELECT** * **FROM** *tableName*
 - **SELECT** * **FROM** authors
- Select specific fields from a table
 - **SELECT** authorID, lastName **FROM** authors

authorID	lastName
1	Deitel
2	Deitel
3	Nieto
4	Santry

Fig. 13 Sample authorID and lastName data from the authors table.

4.2 WHERE Clause

- specify the selection criteria
 - **SELECT** *columnName1, columnName2, ...* **FROM** *tableName*
WHERE *criteria*
 - **SELECT** *title, editionNumber, copyright*
FROM *titles*
WHERE *copyright > 2000*
- **WHERE** clause condition operators
 - <, >, <=, >=, =, <>
 - **LIKE**
 - wildcard characters % and _

4.2 WHERE Clause (Cont.)

title	editionNumber	copyright
C How to Program	3	2001
C++ How to Program	4	2002
The Complete C++ Training Course	4	2002
Internet and World Wide Web How to Program	2	2002
Java How to Program	5	2003
XML How to Program	1	2001
Perl How to Program	1	2001
Advanced Java 2 Platform How to Program	1	2002
Fig. 23.14 Sampling of titles with copyrights after 2000 from table <code>titles</code> .		

4.2 WHERE Clause (Cont.)

- **SELECT** authorID, firstName, lastName
FROM authors

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel

Fig. 15 Authors whose last name starts with D from the authors table.

4.2 WHERE Clause (Cont.)

- **SELECT** authorID, firstName, lastName
FROM authors
WHERE lastName **LIKE** '_i%'

authorID	firstName	lastName
3	Tem	Nieto

Fig. 16 The only author from the authors table whose last name contains i as the second letter.

4.3 ORDER BY Clause

- Optional **ORDER BY** clause
 - **SELECT** *columnName1, columnName2, ...* **FROM** *tableName* **ORDER BY** *column* **ASC**
 - **SELECT** *columnName1, columnName2, ...* **FROM** *tableName* **ORDER BY** *column* **DESC**
- **ORDER BY** multiple fields
 - **ORDER BY** *column1 sortingOrder, column2 sortingOrder, ...*
- Combine the **WHERE** and **ORDER BY** clauses

4.3 ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
FROM authors
ORDER BY lastName **ASC**

authorID	firstName	lastName
2	Paul	Deitel
1	Harvey	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 17 Sample data from table authors in ascending order by lastName .

4.3 ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
FROM authors
ORDER BY lastName **DESC**

authorID	firstName	lastName
4	Sean	Santry
3	Tem	Nieto
2	Paul	Deitel
1	Harvey	Deitel

Fig. 18 Sample data from table authors in descending order by lastName .

4.3 ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
FROM authors
ORDER BY lastName, firstName

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 19 Sample author data from table authors in ascending order by lastName and by firstName .

4.3 ORDER BY Clause (Cont.)

- **SELECT** isbn, title, editionNumber, copyright, price
FROM titles **WHERE** title **LIKE** '%How to Program'
ORDER BY title **ASC**

isbn	title	edition - Number	copy- right	price
0130895601	Advanced Java 2 Platform How to Program	1	2002	74.95
0130895725	C How to Program	3	2001	74.95
0130384747	C++ How to Program	4	2002	74.95
0130308978	Internet and World Wide Web How to Program	2	2002	74.95
0130284181	Perl How to Program	1	2001	74.95
0134569555	Visual Basic 6 How to Program	1	1999	74.95
0130284173	XML How to Program	1	2001	74.95
013028419x	e-Business and e-Commerce How to Program	1	2001	74.95

Fig. 20 Sampling of books from table titles whose titles end with How to Program in ascending order by title .

4.4 Merging Data from Multiple Tables: Joining

- Split related data into separate tables
- Join the tables
 - Merge data from multiple tables into a single view
 - INNER JOIN
 - **SELECT** *columnName1, columnName2, ...*
FROM *table1*
INNER JOIN *table2*
ON *table1.columnName = table2.column2Name*
 - **SELECT** *firstName, lastName, isbn*
FROM *authors, authorISBN*
INNER JOIN *authorISBN*
ON *authors.authorID = authorISBN.authorID*
ORDER BY *lastName, firstName*

4.4 Merging Data from Multiple Tables: Joining (Cont.)

<code>firstName</code>	<code>lastName</code>	<code>isbn</code>	<code>firstName</code>	<code>lastName</code>	<code>isbn</code>
Harvey	Deitel	0130895601	Paul	Deitel	0130895717
Harvey	Deitel	0130284181	Paul	Deitel	0132261197
Harvey	Deitel	0134569555	Paul	Deitel	0130895725
Harvey	Deitel	0139163050	Paul	Deitel	0130829293
Harvey	Deitel	0135289106	Paul	Deitel	0134569555
Harvey	Deitel	0130895717	Paul	Deitel	0130829277
Harvey	Deitel	0130284173	Tem	Nieto	0130161438
Harvey	Deitel	0130829293	Tem	Nieto	013028419x
Paul	Deitel	0130852473	Sean	Santry	0130895601

Fig. 23.21 Sampling of authors and ISBNs for the books they have written in ascending order by `lastName` and `firstName`.

4.5 INSERT Statement

- Insert a row into a table
 - **INSERT INTO** *tableName* (*columnName1*, ... , *columnNameN*)
VALUES (*value1*, ... , *valueN*)
 - **INSERT INTO** authors (firstName, lastName)
VALUES ('Sue', 'Smith')

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Smith

Fig. 22 Sample data from table Authors after an INSERT operation.

4.6 UPDATE Statement

- Modify data in a table

- **UPDATE** *tableName*

- SET** *columnName1 = value1, ... , columnNameN = valueN*

- WHERE** *criteria*

- **UPDATE** authors

- SET** lastName = 'Jones'

- WHERE** lastName = 'Smith' **AND** firstName = 'Sue'

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Jones

Fig. 23 Sample data from table authors after an UPDATE operation.

4.7 DELETE Statement

- Remove data from a table
 - **DELETE FROM** *tableName* **WHERE** *criteria*
 - **DELETE FROM** authors
 - WHERE** lastName = 'Jones' **AND** firstName = 'Sue'

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 24 Sample data from table authors after a DELETE operation.

5 Creating Database books in Hsqldb

- Hsqldb
 - Banco de Dados open source, totalmente em Java
 - Arquivo hsqldb.jar dentro do projeto capítulo 8
- Arquivo de exemplos em
 - Capitulo8\hsqldb\authors_database
 - Script de Criação de banco de dados em: books.sql

Tabela Authors

```
CREATE TABLE AUTHORS (  
  AUTHORID VARCHAR(40) NOT NULL,  
  FIRSTNAME VARCHAR(40),  
  LASTNAME VARCHAR(40) );
```

```
insert into authors values ('1','Harvey','Deitel');  
insert into authors values ('2','Paul','Deitel');  
insert into authors values ('3','Tem','Nieto');  
insert into authors values ('4','Sean','Santry');
```

6 Modificando Databases com JDBC

- Connect to a database
- Query the database
- Display the results of the query

6.1 Connecting to and Querying a Database

- DisplayAuthors
 - Retrieves the entire `authors` table
 - Displays the data in a `JTextArea`

6.1 Connecting to and Querying a JDBC Data Source (Cont.)

Type	Description
1	The <i>JDBC-to-ODBC bridge driver</i> connects Java programs to Microsoft ODBC (Open Database Connectivity) data sources. The Java 2 Software Development Kit from Sun Microsystems, Inc. includes the JDBC-to-ODBC bridge driver (<code>sun.jdbc.odbc.JdbcOdbcDriver</code>). This driver typically requires the ODBC driver to be installed on the client computer and normally requires configuration of the ODBC data source. The bridge driver was introduced primarily for development purposes and should not be used for production applications.
2	<i>Native-API, partly Java drivers</i> enable JDBC programs to use database-specific APIs (normally written in C or C++) that allow client programs to access databases via the Java Native Interface. This driver type translates JDBC into database-specific code. Type 2 drivers were introduced for reasons similar to the Type 1 ODBC bridge driver.
3	<i>JDBC-Net pure Java drivers</i> take JDBC requests and translate them into a network protocol that is not database specific. These requests are sent to a server, which translates the database requests into a database-specific protocol.
4	<i>Native-protocol pure Java drivers</i> convert JDBC requests to database-specific network protocols, so that Java programs can connect directly to a database.

Fig. 23.26 JDBC driver types.

Um Programa bastante simples

- FirstJDBCProgram.java

```

public class FirstJDBCProgram {
public static void main(String args[]) {
    System.out.println("#testando acesso a banco de dado HSQLDB\n\n");
    Connection con = null;
    String teste = "SELECT * FROM authors;";

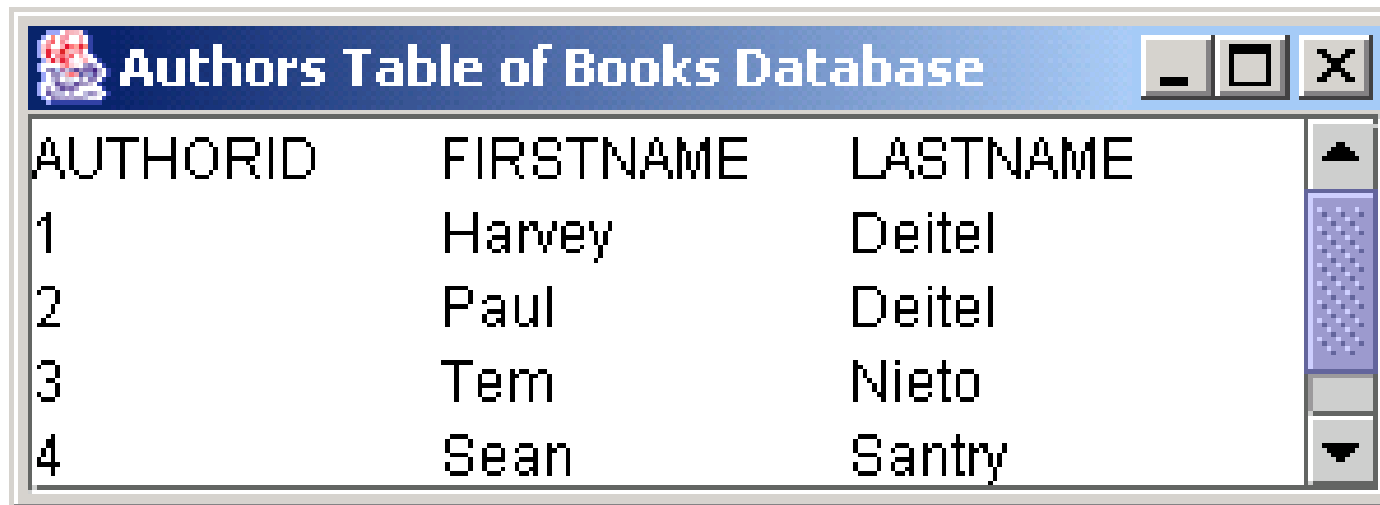
    con = HsqldbConnection.getConnection();
    if(con!=null)
        System.out.println("Conexão aberta com sucesso!");
    else {
        System.out.println("Erro ao tentar abrir Conexão!");
        System.exit(1);
    }
    try {
        Statement stm = con.createStatement();
        ResultSet rs = stm.executeQuery(teste);
        while (rs.next()) {
            Object id=rs.getInt("authorID");
            String linha = rs.getString("firstname");
            System.out.println("Cliente:" +id+ " nome:" +linha);
        }
    } catch (SQLException e) { e.printStackTrace(); }
    finally {
        try {
            con.close();
        }
        catch(SQLException onClose) { System.out.println("error on closing"); onClose.printStackTrace(); }
    }
}

```

Exercício

- Altere o programa para que escreva a lista de autores com nome e sobrenome em ordem decrescente de sobrenome

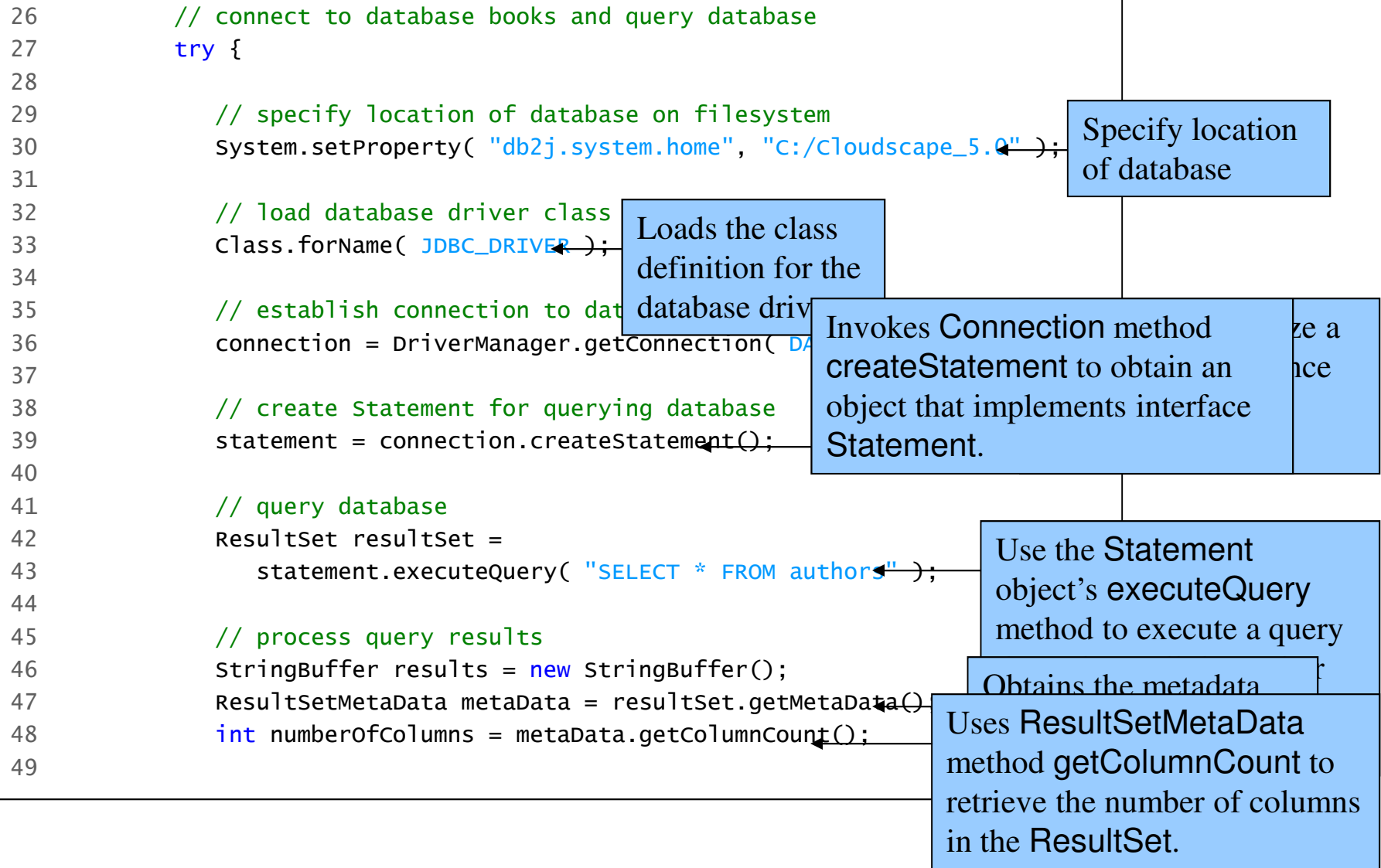
Authors Database



AUTHORID	FIRSTNAME	LASTNAME
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

```
1 // Fig. 26: DisplayAuthors.java
2 // Displaying the contents of the authors table.
3
4 import java.awt.*;
5 import java.sql.*;
6 import java.util.*;
7 import javax.swing.*;
8
9 public class DisplayAuthors extends JFrame {
10
11     // JDBC driver name and database URL
12     static final String JDBC_DRIVER = "com.ibm.db2j.jdbc.DB2jDriver";
13     static final String DATABASE_URL = "jdbc:db2j:books";
14
15     // declare Connection and Statement for accessing
16     // and querying database
17     private Connection connection;
18     private Statement statement;
19
20     // constructor connects to database, queries database, processes
21     // results and displays results in window
22     public DisplayAuthors()
23     {
24         super( "Authors Table of Books Database" );
25
```

Imports package `java.sql`,
which contains classes and
interfaces for the JDBC API.



```
50     for ( int i = 1; i <= numberOfColumns; i++ )
51         results.append( metaData.getColumnname( i ) + "\t" );
52
53     results.append( "\n" );
54
55     while ( resultSet.next() ) {
56
57         for ( int i = 1; i <= numberOfColumns; i++ )
58             results.append( resultSet.getObject( i ) + "\t" );
59
60         results.append( "\n" );
61     }
62
63     // set up GUI and display window
64     JTextArea textArea = new JTextArea( results.toString() );
65     Container container = getContentPane();
66
67     container.add( new JScrollPane( textArea ) );
68
69     setSize( 300, 100 ); // set window size
70     setVisible( true ); // display window
71
72 } // end try
73
```

Append the column names to the StringBuffer results.

Append the data in each ResultSet row to the StringBuffer results.

Create the GUI that displays the StringBuffer results, set the size of the application window and show the application window.

```
74 // detect problems interacting with the database
75 catch ( SQLException sqlException ) {
76     JOptionPane.showMessageDialog( null, sqlException.getMessage(),
77         "Database Error", JOptionPane.ERROR_MESSAGE );
78
79     System.exit( 1 );
80 }
81
82 // detect problems loading database driver
83 catch ( ClassNotFoundException classNotFound ) {
84     JOptionPane.showMessageDialog( null, classNotFound.getMessage(),
85         "Driver Not Found", JOptionPane.ERROR_MESSAGE );
86
87     System.exit( 1 );
88 }
89
90 // ensure statement and connection are closed properly
91 finally {
92
93     try {
94         statement.close();
95         connection.close();
96     }
97 }
```

Close the Statement and the
database Connection.

```
98     // handle exceptions closing statement and connection
99     catch ( SQLException sqlException ) {
100         JOptionPane.showMessageDialog( null,
101             sqlException.getMessage(), "Database Error",
102             JOptionPane.ERROR_MESSAGE );
103
104         System.exit( 1 );
105     }
106 }
107
108 } // end DisplayAuthors constructor
109
110 // launch the application
111 public static void main( String args[] )
112 {
113     DisplayAuthors window = new DisplayAuthors();
114     window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
115 }
116
117 } // end class DisplayAuthors
```

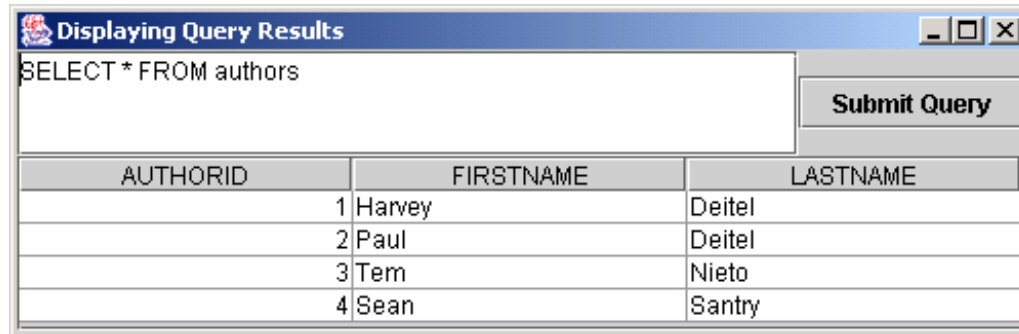
Exercícios

- Altere o programa para que este apresente a lista de livros (isbn, título, número da edição, preço) disponíveis no banco de dados
- Inclua à direita do preço o nome da Editora (publisher)

6.2 Querying the `books` Database

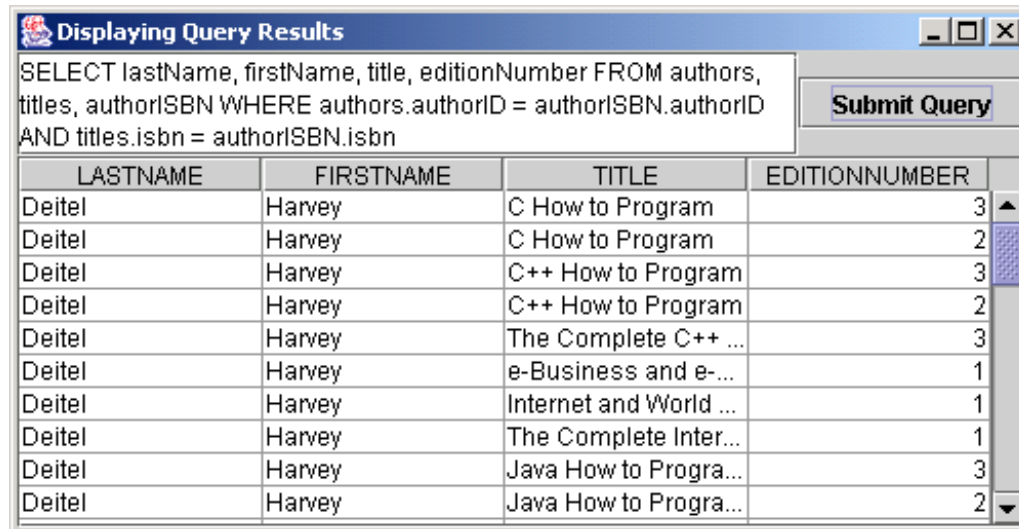
- Allow the user to enter any query into the program
- Display the results of a query in a `JTable`

6.2 Querying the books Database (Cont.)



The screenshot shows a window titled "Displaying Query Results" with a text area containing the query `SELECT * FROM authors` and a "Submit Query" button. Below the query is a table with three columns: AUTHORID, FIRSTNAME, and LASTNAME. The results are as follows:

AUTHORID	FIRSTNAME	LASTNAME
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry



The screenshot shows a window titled "Displaying Query Results" with a text area containing a complex query: `SELECT lastName, firstName, title, editionNumber FROM authors, titles, authorISBN WHERE authors.authorID = authorISBN.authorID AND titles.isbn = authorISBN.isbn`. Below the query is a "Submit Query" button and a table with four columns: LASTNAME, FIRSTNAME, TITLE, and EDITIONNUMBER. The results are as follows:

LASTNAME	FIRSTNAME	TITLE	EDITIONNUMBER
Deitel	Harvey	C How to Program	3
Deitel	Harvey	C How to Program	2
Deitel	Harvey	C++ How to Program	3
Deitel	Harvey	C++ How to Program	2
Deitel	Harvey	The Complete C++ ...	3
Deitel	Harvey	e-Business and e-...	1
Deitel	Harvey	Internet and World ...	1
Deitel	Harvey	The Complete Inter...	1
Deitel	Harvey	Java How to Progra...	3
Deitel	Harvey	Java How to Progra...	2

```
1 // Fig. 27: ResultSetTableModel.java
2 // A TableModel that supplies ResultSet data to a JTable.
3
4 import java.sql.*;
5 import java.util.*;
6 import javax.swing.table.*;
7
8 // ResultSet rows and columns are counted from 1 and JTable
9 // rows and columns are counted from 0. When processing
10 // ResultSet rows or columns for use in a JTable, it is
11 // necessary to add 1 to the row or column number to manipulate
12 // the appropriate ResultSet column (i.e., JTable column 0 is
13 // ResultSet column 1 and JTable row 0 is ResultSet row 1).
14 public class ResultSetTableModel extends AbstractTableModel {
15     private Connection connection;
16     private Statement statement;
17     private ResultSet resultSet;
18     private ResultSetMetaData metaData;
19     private int numberOfRows;
20
21     // keep track of database connection status
22     private boolean connectedToDatabase = false;
23
```

```
24 // initialize resultSet and obtain its meta data object;
25 // determine number of rows
26 public ResultSetTableModel( String driver, String url,
27     String query ) throws SQLException, ClassNotFoundException
28 {
29     // load database driver class
30     Class.forName( driver );
31
32     // connect to database
33     connection = DriverManager.getConnection( url );
34
35     // create Statement to query database
36     statement = connection.createStatement(
37         ResultSet.TYPE_SCROLL_INSENSITIVE,
38         ResultSet.CONCUR_READ_ONLY );
39
40     // update database connection status
41     connectedToDatabase = true;
42
43     // set query and execute it
44     setQuery( query );
45 }
46
```

Establishes a connection to the database.

Invokes Connection method createStatement to create a Statement object.

Invokes ResultSetTableModel method setQuery to perform the default query.

```

47 // get class that represents column type
48 public Class getColumnClass( int column ) throws IllegalStateException
49 {
50 // ensure database connection is available
51 if ( !connectedToDatabase )
52     throw new IllegalStateException( "Not Connected to Database" );
53
54 // determine Java class of column
55 try {
56     String className = metaData.getColumnClassName( column + 1 );
57
58 // return Class object that represents
59     return Class.forName( className );
60 }
61
62 // catch SQLExceptions and ClassNotFoundExceptions
63 catch ( Exception exception ) {
64     exception.printStackTrace();
65 }
66
67 // if problems occur always return default object
68 return object.getClass();
69 }
70

```

Obtains the fully qualified class name for the specified column.

Loads the class definition for the class and returns the corresponding Class object.

Returns the default type.

```

71 // get number of columns in ResultSet
72 public int getColumnCount() throws IllegalStateException
73 {
74     // ensure database connection is available
75     if ( !connectedToDatabase )
76         throw new IllegalStateException( "Not Connected to Database" );
77
78     // determine number of columns
79     try {
80         return metaData.getColumnCount();
81     }
82
83     // catch SQLExceptions and print error message
84     catch ( SQLException sqlException ) {
85         sqlException.printStackTrace();
86     }
87
88     // if problems occur above, return 0 for number of columns
89     return 0;
90 }
91
92 // get name of a particular column in ResultSet
93 public String getColumnName( int column ) throws IllegalStateException
94 {
95     // ensure database connection is available
96     if ( !connectedToDatabase )
97         throw new IllegalStateException( "Not Connected to Database" );

```

Obtains the number of columns in the ResultSet.

```
98
99     // determine column name
100    try {
101        return metaData.getColumnName( column + ← );
102    }
103
104    // catch SQLExceptions and print error message
105    catch ( SQLException sqlException ) {
106        sqlException.printStackTrace();
107    }
108
109    // if problems, return empty string for column name
110    return "";
111 }
112
113 // return number of rows in ResultSet
114 public int getRowCount() throws IllegalStateException
115 {
116     // ensure database connection is available
117     if ( !connectedToDatabase )
118         throw new IllegalStateException( "Not Connected to Database" );
119
120     return numberOfRows;
121 }
122
```

Obtains the column name
from the ResultSet.

```
123 // obtain value in particular row and column
124 public Object getValueAt( int row, int column )
125     throws IllegalStateException
126 {
127     // ensure database connection is available
128     if ( !connectedToDatabase )
129         throw new IllegalStateException( "Not Connected to Database" );
130
131     // obtain a value at specified ResultSet row and column
132     try {
133         resultSet.absolute( row + 1 );
134
135         return resultSet.getObject( column + 1 );
136     }
137
138     // catch SQLExceptions and print error message
139     catch ( SQLException sqlException ) {
140         sqlException.printStackTrace();
141     }
142
143     // if problems, return empty string object
144     return "";
145 }
146
```

Uses ResultSet method absolute to position the ResultSet cursor at a specific row

Uses ResultSet method getObject to obtain the Object in a specific column of the current row.

```

147 // set new database query string
148 public void setQuery( String query )
149     throws SQLException, IllegalStateException
150 {
151     // ensure database connection is available
152     if ( !connectedToDatabase )
153         throw new IllegalStateException( "Not Connected to Database" );
154
155     // specify query and execute it
156     resultSet = statement.executeQuery( query );
157
158     // obtain meta data for ResultSet
159     metaData = resultSet.getMetaData();
160
161     // determine number of rows in ResultSet
162     resultSet.last();
163     numberOfRows = resultSet.getRow();
164
165     // notify JTable that model has changed
166     fireTableStructureChanged();
167 }
168

```

Executes the query to obtain a new **ResultSet**.

Uses **ResultSet** method **last** to
position
at the
Res

Uses **ResultSet** method **getRow** to
obtain the row number for the current

Invokes method **fireTableStructureChanged** to
notify any **JTable** using this
ResultSetTableModel object as its model that
the structure of the model has changed.

```
169 // close Statement and Connection
170 public void disconnectFromDatabase()
171 {
172     // close Statement and Connection
173     try {
174         statement.close();
175         connection.close();
176     }
177
178     // catch SQLExceptions and print error message
179     catch ( SQLException sqlException ) {
180         sqlException.printStackTrace();
181     }
182
183     // update database connection status
184     finally {
185         connectedToDatabase = false;
186     }
187 }
188
189 } // end class ResultSetTableModel
```

Close the Statement and Connection if a ResultSetTableModel object is garbage collected.

6.2 Querying the books Database (Cont.)

ResultSet static type constant	Description
TYPE_FORWARD_ONLY	Specifies that a <code>ResultSet</code> 's cursor can move only in the forward direction (i.e., from the first row to the last row in the <code>ResultSet</code>).
TYPE_SCROLL_INSENSITIVE	Specifies that a <code>ResultSet</code> 's cursor can scroll in either direction and that the changes made to the <code>ResultSet</code> during <code>ResultSet</code> processing are not reflected in the <code>ResultSet</code> unless the program queries the database again.
TYPE_SCROLL_SENSITIVE	Specifies that a <code>ResultSet</code> 's cursor can scroll in either direction and that the changes made to the <code>ResultSet</code> during <code>ResultSet</code> processing are reflected immediately in the <code>ResultSet</code> .

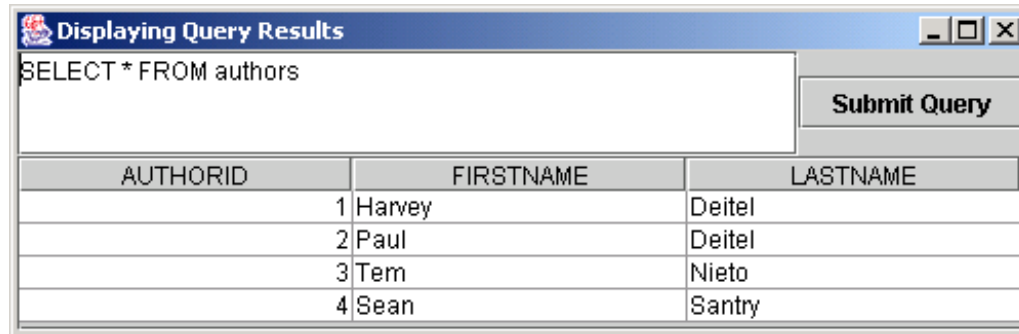
Fig. 23.28 `ResultSet` constants for specifying `ResultSet` type.

6.2 Querying the books Database (Cont.)

<code>ResultSet</code> static concurrency constant	Description
<code>CONCUR_READ_ONLY</code>	Specifies that a <code>ResultSet</code> cannot be updated (i.e., changes to the <code>ResultSet</code> contents cannot be reflected in the database with <code>ResultSet</code> 's <i>update</i> methods).
<code>CONCUR_UPDATABLE</code>	Specifies that a <code>ResultSet</code> can be updated (i.e., changes to the <code>ResultSet</code> contents can be reflected in the database with <code>ResultSet</code> 's <i>update</i> methods).

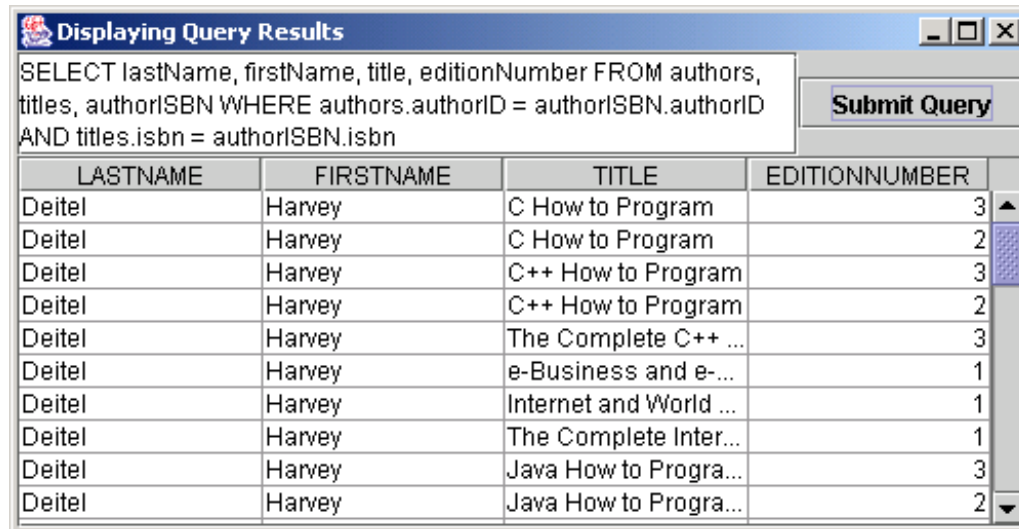
Fig. 23.29 `ResultSet` constants for specifying result properties.

6.2 Querying the books Database (Cont.)



The screenshot shows a window titled "Displaying Query Results" with a text area containing the query `SELECT * FROM authors` and a "Submit Query" button. Below the query is a table with three columns: AUTHORID, FIRSTNAME, and LASTNAME. The results are as follows:

AUTHORID	FIRSTNAME	LASTNAME
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

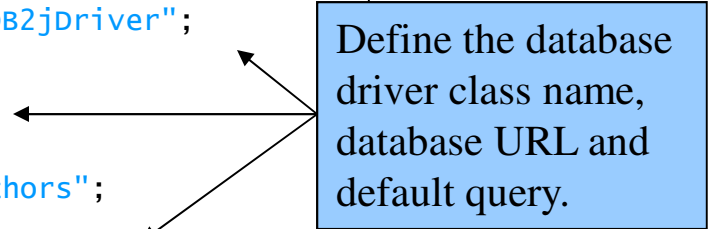


The screenshot shows a window titled "Displaying Query Results" with a text area containing a complex query: `SELECT lastName, firstName, title, editionNumber FROM authors, titles, authorISBN WHERE authors.authorID = authorISBN.authorID AND titles.isbn = authorISBN.isbn`. Below the query is a "Submit Query" button and a table with four columns: LASTNAME, FIRSTNAME, TITLE, and EDITIONNUMBER. The results are as follows:

LASTNAME	FIRSTNAME	TITLE	EDITIONNUMBER
Deitel	Harvey	C How to Program	3
Deitel	Harvey	C How to Program	2
Deitel	Harvey	C++ How to Program	3
Deitel	Harvey	C++ How to Program	2
Deitel	Harvey	The Complete C++ ...	3
Deitel	Harvey	e-Business and e-...	1
Deitel	Harvey	Internet and World ...	1
Deitel	Harvey	The Complete Inter...	1
Deitel	Harvey	Java How to Progra...	3
Deitel	Harvey	Java How to Progra...	2

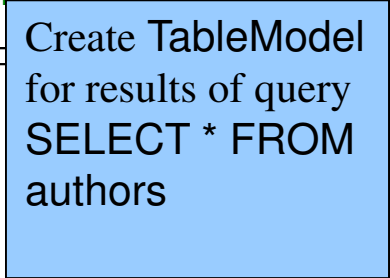
```
1 // Fig. 30: DisplayQueryResults.java
2 // Display the contents of the Authors table in the
3 // Books database.
4
5 import java.awt.*;
6 import java.awt.event.*;
7 import java.sql.*;
8 import java.util.*;
9 import javax.swing.*;
10 import javax.swing.table.*;
11
12 public class DisplayQueryResults extends JFrame {
13
14     // JDBC driver and database URL
15     static final String JDBC_DRIVER = "com.ibm.db2j.jdbc.DB2jDriver";
16     static final String DATABASE_URL = "jdbc:db2j:books";
17
18     // default query selects all rows from authors table
19     static final String DEFAULT_QUERY = "SELECT * FROM authors";
20
21     private ResultSetTableModel tableModel;
22     private JTextArea queryArea;
23
24     // create ResultSetTableModel and GUI
25     public DisplayQueryResults()
26     {
27         super( "Displaying Query Results" );
```

Define the database driver class name, database URL and default query.



```
28
29 // create ResultSetTableModel and display database table
30 try {
31
32 // specify location of database on filesystem
33 System.setProperty( "db2j.system.home", "C:/Cloudscape_5.0" );
34
35 // create TableModel for results of query SELECT * FROM authors
36 tableModel = new ResultSetTableModel( JDBC_DRIVER, DATABASE_
37     DEFAULT_QUERY );
38
39 // set up JTextArea in which user types queries
40 queryArea = new JTextArea( DEFAULT_QUERY, 3, 100 );
41 queryArea.setWrapStyleWord( true );
42 queryArea.setLineWrap( true );
43
44 JScrollPane scrollPane = new JScrollPane( queryArea,
45     ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
46     ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER );
47
48 // set up JButton for submitting queries
49 JButton submitButton = new JButton( "Submit Query" );
50
51 // create Box to manage placement of queryArea and
52 // submitButton in GUI
53 Box box = Box.createHorizontalBox();
54 box.add( scrollPane );
55 box.add( submitButton );
56
```

Create TableModel
for results of query
SELECT * FROM
authors



```

57 // create jTable delegate for tableModel
58 jTable resultTable = new jTable( tableModel );
59
60 // place GUI components on content pane
61 Container c = getContentPane();
62 c.add( box, BorderLayout.NORTH );
63 c.add( new JScrollPane( resultTable ), BorderLayout.CENTER );
64
65 // create event listener for submitButton
66 submitButton.addActionListener(
67
68     new ActionListener() {
69
70         // pass query to table model
71         public void actionPerformed( ActionEvent event )
72         {
73             // perform a new query
74             try {
75                 tableModel.setQuery( queryArea.getText() );
76             }
77
78             // catch SQLExceptions when performing
79             catch ( SQLException sqlException ) {
80                 JOptionPane.showMessageDialog( null,
81                     sqlException.getMessage(), "Database error",
82                     JOptionPane.ERROR_MESSAGE );
83

```

Create jTable delegate for tableModel

Register an event handler for the submitButton that the user clicks to submit a query to the database.

Invokes ResultSetTableModel method setQuery to execute the new query.

```
84         // try to recover from invalid user query
85         // by executing default query
86         try {
87             tableModel.setQuery( DEFAULT_QUERY );
88             queryArea.setText( DEFAULT_QUERY );
89         }
90
91         // catch SQLException when performing default query
92         catch ( SQLException sqlException2 ) {
93             JOptionPane.showMessageDialog( null,
94                 sqlException2.getMessage(), "Database error",
95                 JOptionPane.ERROR_MESSAGE );
96
97             // ensure database connection is closed
98             tableModel.disconnectFromDatabase();
99
100            System.exit( 1 ); // terminate application
101
102            } // end inner catch
103
104            } // end outer catch
105
106            } // end actionPerformed
107
108            } // end ActionListener inner class
109
110        ); // end call to addActionListener
111
```

```

112         // set window size and display window
113         setSize( 500, 250 );
114         setVisible( true );
115
116     } // end try
117
118     // catch ClassNotFoundException thrown by
119     // ResultSetTableModel if database driver not found
120     catch ( ClassNotFoundException classNotFound ) {
121         JOptionPane.showMessageDialog( null,
122             "Cloudscape driver not found", "Driver not found",
123             JOptionPane.ERROR_MESSAGE );
124
125         System.exit( 1 ); // terminate application
126     } // end catch
127
128     // catch SQLException thrown by ResultSetTableModel
129     // if problems occur while setting up database
130     // connection and querying database
131     catch ( SQLException sqlException ) {
132         JOptionPane.showMessageDialog( null, sqlException.getMessage(),
133             "Database error", JOptionPane.ERROR_MESSAGE );
134
135         // ensure database connection is closed
136         tableModel.disconnectFromDatabase();
137
138         System.exit( 1 ); // terminate application
139     }
140

```

```
141     // dispose of window when user quits application (this overrides
142     // the default of HIDE_ON_CLOSE)
143     setDefaultCloseOperation( DISPOSE_ON_CLOSE );
144
145     // ensure database connection is closed when user quits application
146     addWindowListener(
147
148         new WindowAdapter() {
149
150             // disconnect from database and exit when window has closed
151             public void windowClosed( WindowEvent event )
152             {
153                 tableModel.disconnectFromDatabase();
154                 System.exit( 0 );
155             }
156         }
157     );
158
159 } // end DisplayQueryResults constructor
160
161 // execute application
162 public static void main( String args[] )
163 {
164     new DisplayQueryResults();
165 }
166
167 } // end class DisplayQueryResults
```

Exercício: DAO e RDBMS

- Com base no exemplo do início do capítulo sobre autores (AuthorsDAO), faça o armazenamento e recuperação dos dados de autores em um banco de dados Hsqldb.
- Utilize os padrões DAO e abstract factory.
- Antes de iniciar a implementação defina os passos necessários a execução da tarefa e quais classes/interfaces deverão ser criadas....

Exercício: DAO e RDBMS

- Passos para a solução:
 - Crie uma classe abstrata DAOFactory com o método estático getDAOFactory(int ds). Onde ds é um indicador de Fonte de Data Source (ds). Quando ds=1 deve-se retornar uma HSQLDBDAOFactory. Retorne nulo em outros casos.
 - DAOFactory deve especificar o método abstrato getAuthorDAO que retorna um AuthorDAO
 - Crie uma classe HsqlDBDAOFactory que herde DAOFactory e implemente o método getAuthorsDAO e retorne um AuthorsDAOImpl

```

package DAO;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.swing.JOptionPane;

public class AuthorDAOImpl implements AuthorDAO {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "org.firebirdsql.jdbc.FBDriver";
    static final String host="126.0.0.1";
    static final String path="c:\\eclipse\\stefanini\\Aula7\\books.fdb";
    static final String DATABASE_URL = "jdbc:firebirdsql:"+host+"/3050:"+path;

    // declare Connection and Statement for accessing
    // and querying database
    private Connection connection;
    private Statement statement;

    public AuthorSTO getAuthors() {
        // connect to database books and query database
        try {
            // load database driver class
            Class.forName( JDBC_DRIVER );

            // establish connection to database
            connection = DriverManager.getConnection( DATABASE_URL , "SYSDBA",
"masterkey");

            // create Statement for querying database
            statement = connection.createStatement();

            // query database
            ResultSet resultSet =
                statement.executeQuery( "SELECT * FROM authors;" );

```

AuthorDAOImpl.java

AuthorDAOImpl.java

```
// process query results
AuthorSTO authors=new AuthorSTO();

//resultSet=statement.executeQuery( "SELECT * FROM authors" );
while ( resultSet.next() ) {
    int id=resultSet.getInt(1);
    String firstName=resultSet.getString(2);
    String lastName=resultSet.getString(3);
    authors.add(new AuthorTO(id,firstName,lastName));
}
return authors;
}
// detect problems interacting with the database
catch ( SQLException sqlException ) {
    JOptionPane.showMessageDialog( null, sqlException.getMessage(),
        "Database Error", JOptionPane.ERROR_MESSAGE );

    System.exit( 1 );
}
// detect problems loading database driver
catch ( ClassNotFoundException classNotFound ) {
    JOptionPane.showMessageDialog( null, classNotFound.getMessage(),
        "Driver Not Found", JOptionPane.ERROR_MESSAGE );

    System.exit( 1 );
}
// ensure statement and connection are closed properly
finally {
    try {
        statement.close();
        connection.close();
    }
    // handle exceptions closing statement and connection
    catch ( SQLException sqlException ) {
        JOptionPane.showMessageDialog( null,
            sqlException.getMessage(), "Database Error",
            JOptionPane.ERROR_MESSAGE );

        System.exit( 1 );
    }
}

return null;
}
}
```

Exercícios

- Altere o programa para que este execute inicialmente a consulta a tabela de livros (titles)
- Elabore e execute uma query SQL sobre o banco de dados books envolvendo pelo menos três tabelas
- Crie um botão chamado “Execute Update” que realiza operações de atualização no programa DisplayQueryResults: insert/update
 - Insira um nome autor
 - Atualize o preço do livro Core Java para 60.00

7 Referências sobre JDBC

- Sun Microsystems JDBC home page
 - [Java.sun.com/products/jdbc](http://java.sun.com/products/jdbc)
- SQL materials
 - www.sql.org
- Home page do Banco de Dados Firebird
 - <http://firebird.sourceforge.net/>
- Referências sobre DAO Design Pattern
 - <http://java.sun.com/blueprints/patterns/DAO.html>