Chapter 13: Database Basics

Database Basics

In this chapter we will learn about databases and their uses within Java programing. We will work with a more sophisticated data verse a flat files.

We will explore an open source database from the Apache Organization called Derby. In addition we review the basics of JDBC (Java Database Connectivity) drivers and SQL (Structured Query Language).

Database Basics

Java Database Connectivity (JDBC) is a set of classes that can be used to develop client/server applications that work with databases developed by many vendors.

With JDBC, we can use the same methods and classes that we've looked at to read and write records as well as perform other tasks in the database.

A class called a driver acts as the bridge to the database source. Each of the popular databases have a driver in Java.

CLIENT/SERVER software connects a user of information with a provider of that information. We (client) request a web page from a source (server) and the server provides us the information we are requesting.

Database Basics

When we need to request information of database, we use a programming language called **Structured Query Language (SQL).** When we request information from a database using SQL, (pronounced See-Quel), we ask the database a question to request information, such as “How many blue cars were made?”. This question is called a **QUERY**. Not only can we use SQL to request information, it is used to control the database in general such as Create, Update and Delete tables.

There are two types of SQL:

Data Manipulation Language (DML) and **Data Definition Language (DDL)**. DML is what we used to build, modify, delete and request information where as DDL is used by the **Database Administrator (DBA)** to build the database structures as well as control the security called **Access Controls**. SQL is the industry standard used to access a relational database.

Database Basics

The **JDBC library, java.sql** package, includes classes for each of the tasks commonly associated with database usage:

Making the connection to the database

Creating statements using SQL

Executing the SQL query in the database

Viewing the **RESULT SETS** (the records that are returned from the database)

Database Drivers

Since Java programs that use JDBC classes can follow the programming model standard for executing SQL statements and processing the result set, the format of the database and the platform it was created with is not of concern.

This is another reason why we call Java platform independent. As a result of the driver manager which keeps track of the drivers required for access to the database records we need a different driver for each database format that is used in a program.

We can also use the Open Database Connectivity driver (ODBC) to connect to other types of data sources such as Excel and the like.

Data Tools Platform (DTP)

- A cohesive set of frameworks and tools for creating, managing, and using data.
- Intended for use in data-centric development and administrative tasks
- Several projects, including Model Base, Enablement, Connectivity, and SQL Development Tools.

DTP Model Base

project provides the foundation for database development.

uses best practices, like model-driven development with UML, and the Eclipse Modeling Framework (EMF)

Database definition model

- Edit model
- SQL model
- SQL Query model
- SQL XML Query model

Connectivity

- includes components for defining, connecting to, and working with data sources
- Driver Management Framework: create driver definitions based on templates
- Connection Management Framework: is the basis for specific connection
- JDBC connection support: DTP includes a JDBC driver template
- Data Source Explorer: Data Source Explorer is an Eclipse view of connection instances
- Open Data Access:(ODA) is a data-access framework allowing applications to access data from standard and custom data sources

SQL Development Tools

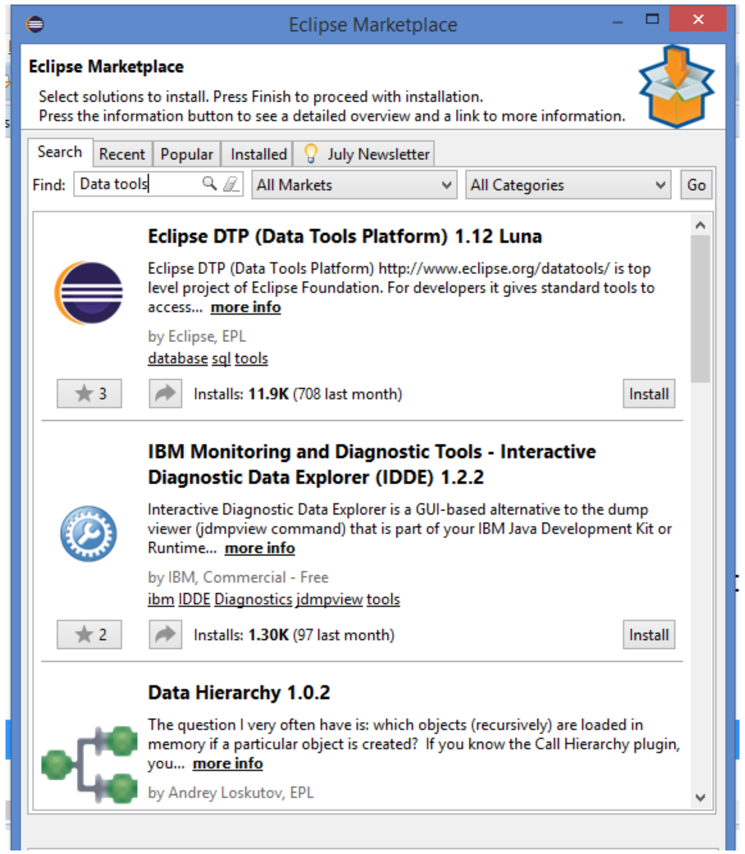
- Editor Framework is an extensible framework for editing database routines and SQL statements.
- Debugger Framework provides an extensible support base enabling debugging
- SQL Query Parser provides an extensible framework enabling SQL components and tools
- SQL Query Builder enables you to create, edit, or run SQL statements using the SQL Query Builder graphical interface
- SQL Execution Plan Framework provides a means for capturing and presenting execution plans in a generic fashion

Apache Derby

1. Appropriate versions of Eclipse, Data Tools Platform, EMF, and GEFDownload or access these builds from http://www.eclipse.org/datatools/downloads.php

2. Apache Derby Download the appropriate version from http://db.apache.org/derby/

Find and install DTP



Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Find: Data tools

Eclipse DTP (Data Tools Platform) 1.12 Luna

Eclipse DTP (Data Tools Platform) http://www.eclipse.org/datatools/ is top level project of Eclipse Foundation. For developers it gives standard tools to access... **more info**

by Eclipse, EPL
database sql tools

3 Installs: **11.9K** (708 last month)

IBM Monitoring and Diagnostic Tools - Interactive Diagnostic Data Explorer (IDDE) 1.2.2

Interactive Diagnostic Data Explorer is a GUI-based alternative to the dump viewer (jdmpview command) that is part of your IBM Java Development Kit or Runtime... **more info**

by IBM, Commercial - Free
ibm IDDE Diagnostics jdmpview tools

2 Installs: **1.30K** (97 last month)

Data Hierarchy 1.0.2

The question I very often have is: which objects (recursively) are loaded in memory if a particular object is created? If you know the Call Hierarchy plugin, you... **more info**

by Andrey Loskutov, EPL

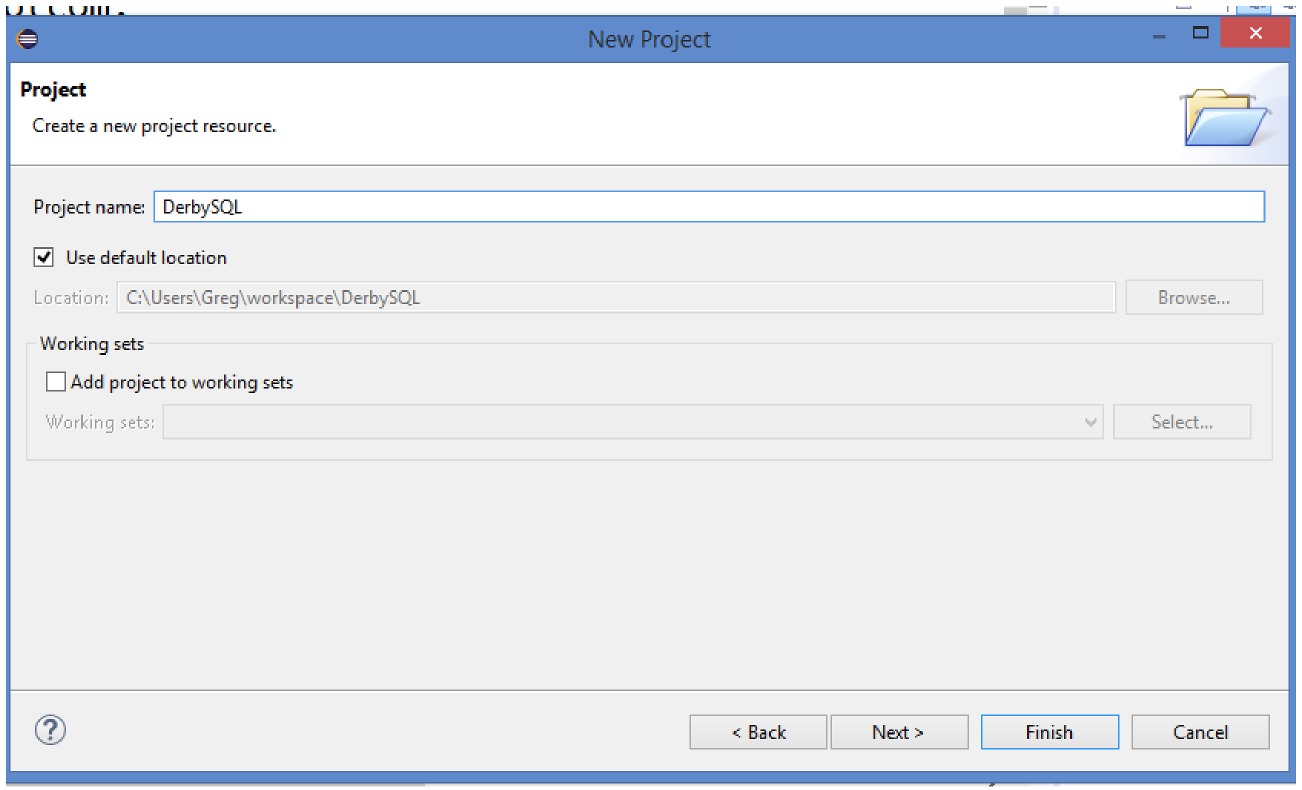
Find and Install Derby

http://db.apache.org/derby/

Download and Extract

Create a new General Project

"DerbySQL" via File-> New -> Other -> General -> Project.



New Project

Project

Create a new project resource.

Project name: DerbySQL

[✓] Use default location

Location: C:\Users\Greg\workspace\DerbySQL

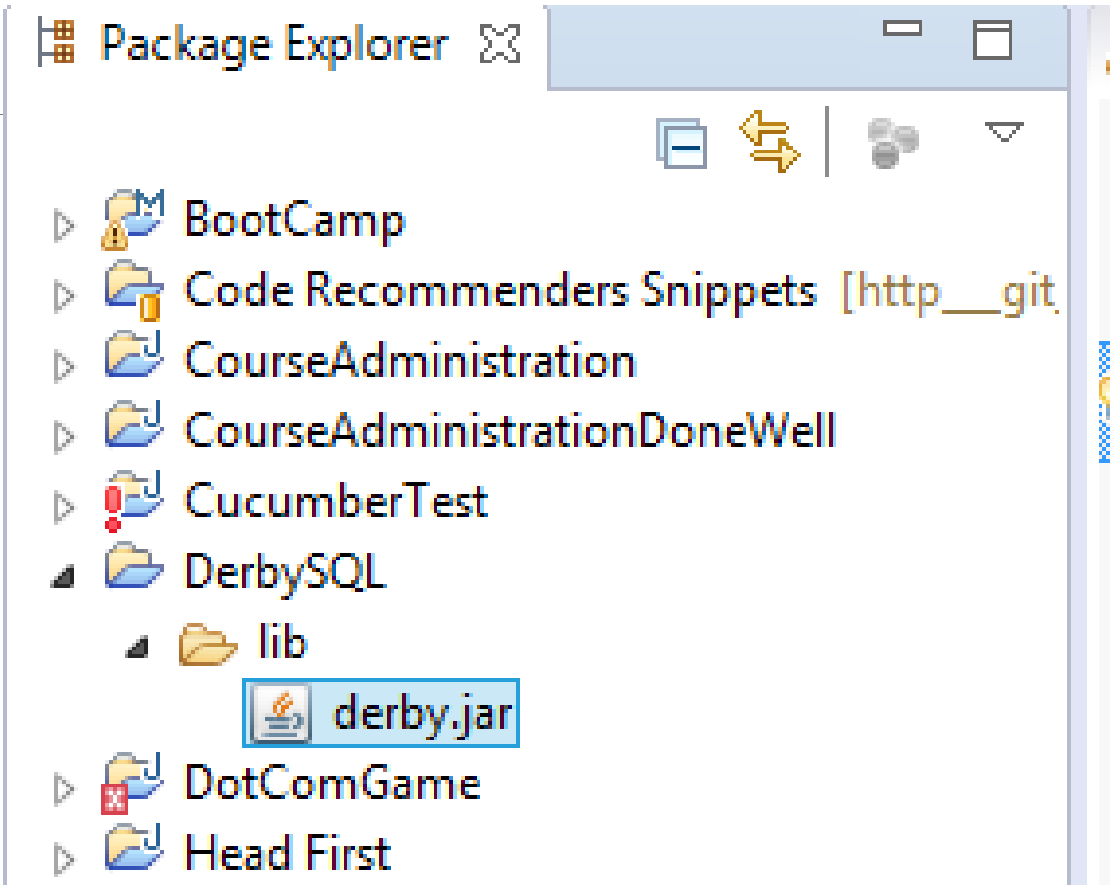
Working sets

[ ] Add project to working sets

Working sets:

Add Derby to Eclipse

Add a folder "lib" to your project. Copy the file derby.jar from your Derby download into this folder.



Define the driver for the Derby access

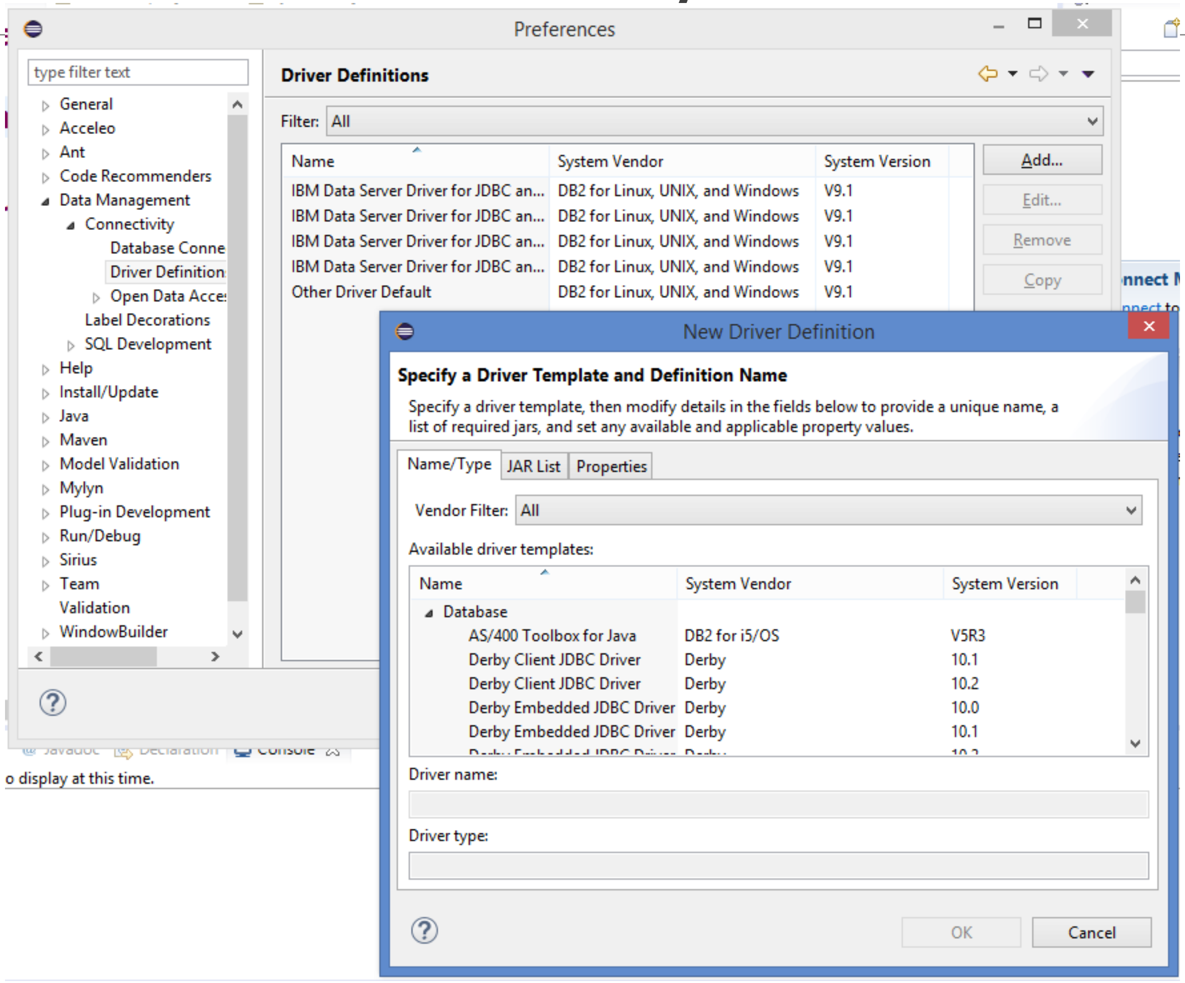
Go to Window-> Preferences

select "Data Management" -> Connectivity ->

Driver Definition.

Press Add.

Select Derby



Driver Definitions

Filter: All

Name	System Vendor	System Version
IBM Data Server Driver for JDBC an...	DB2 for Linux, UNIX, and Windows	V9.1
IBM Data Server Driver for JDBC an...	DB2 for Linux, UNIX, and Windows	V9.1
IBM Data Server Driver for JDBC an...	DB2 for Linux, UNIX, and Windows	V9.1
IBM Data Server Driver for JDBC an...	DB2 for Linux, UNIX, and Windows	V9.1
Other Driver Default	DB2 for Linux, UNIX, and Windows	V9.1

Specify a Driver Template and Definition Name

Specify a driver template, then modify details in the fields below to provide a unique name, a list of required jars, and set any available and applicable property values.

Vendor Filter: All

Available driver templates:

Name	System Vendor	System Version
Database		
AS/400 Toolbox for Java	DB2 for i5/OS	V5R3
Derby Client JDBC Driver	Derby	10.1
Derby Client JDBC Driver	Derby	10.2
Derby Embedded JDBC Driver	Derby	10.0
Derby Embedded JDBC Driver	Derby	10.1

Driver name:

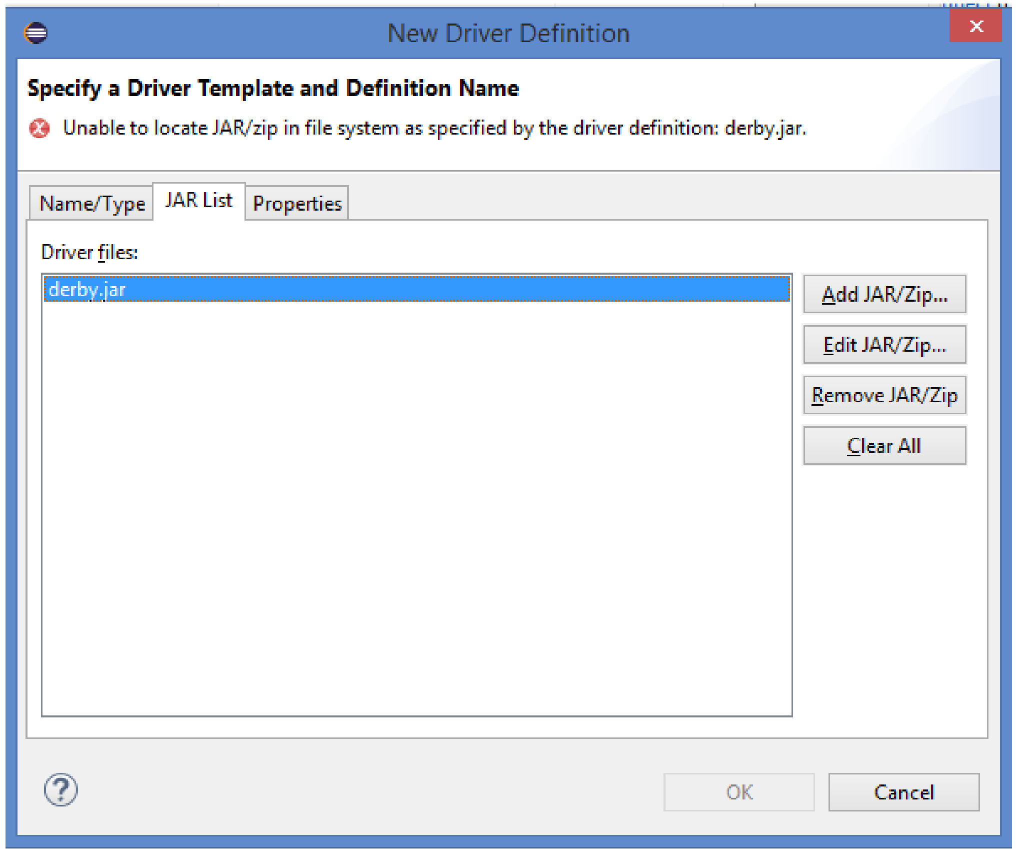
Driver type:

o display at this time.

Add Jar file from Lib

Select then the tab jar press Add... and select the derby.jar from your project folder "lib“

OK

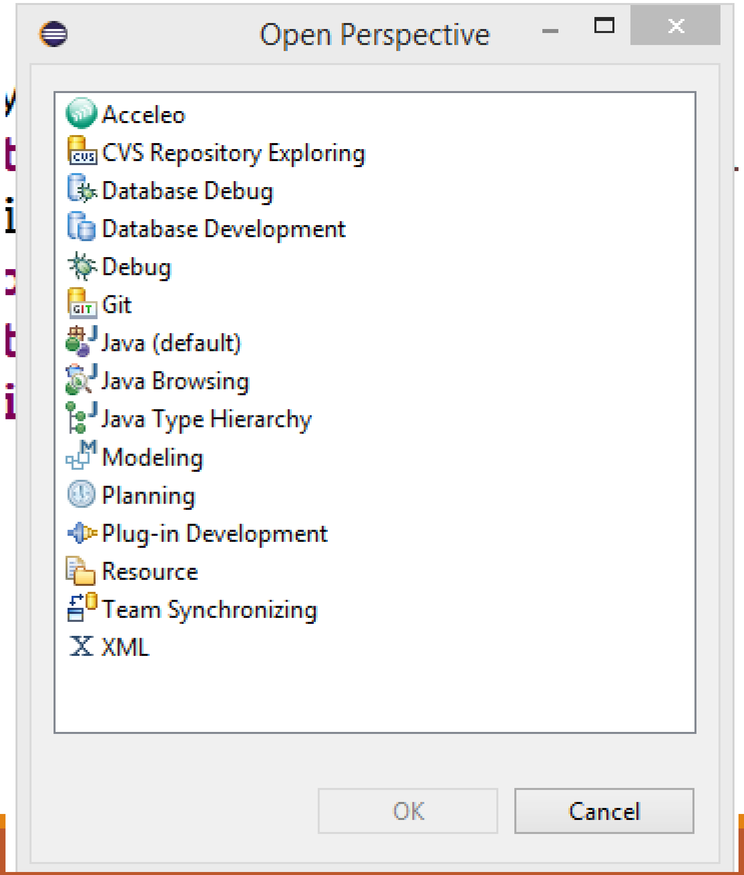


Specify a Driver Template and Definition Name

Unable to locate JAR/zip in file system as specified by the driver definition: derby.jar.

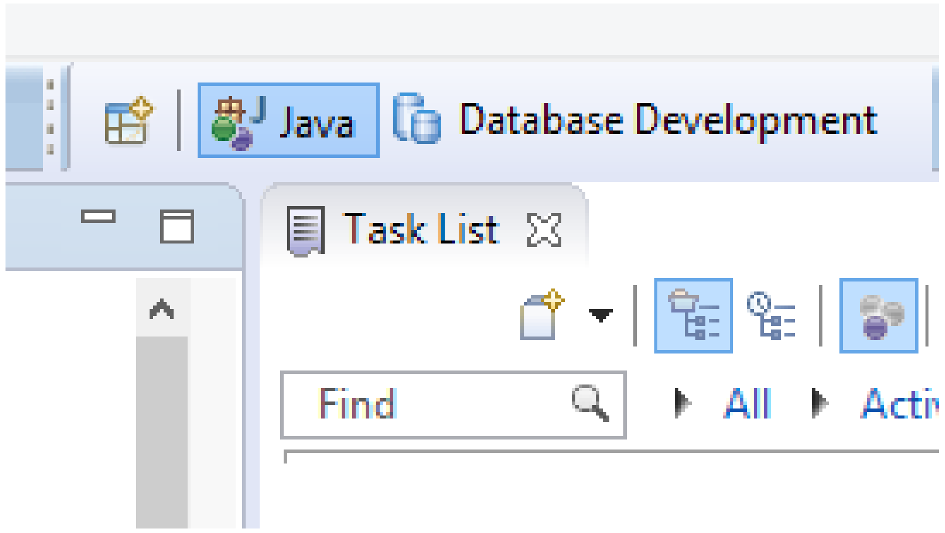
Open Database Perspective

Open New Perspective

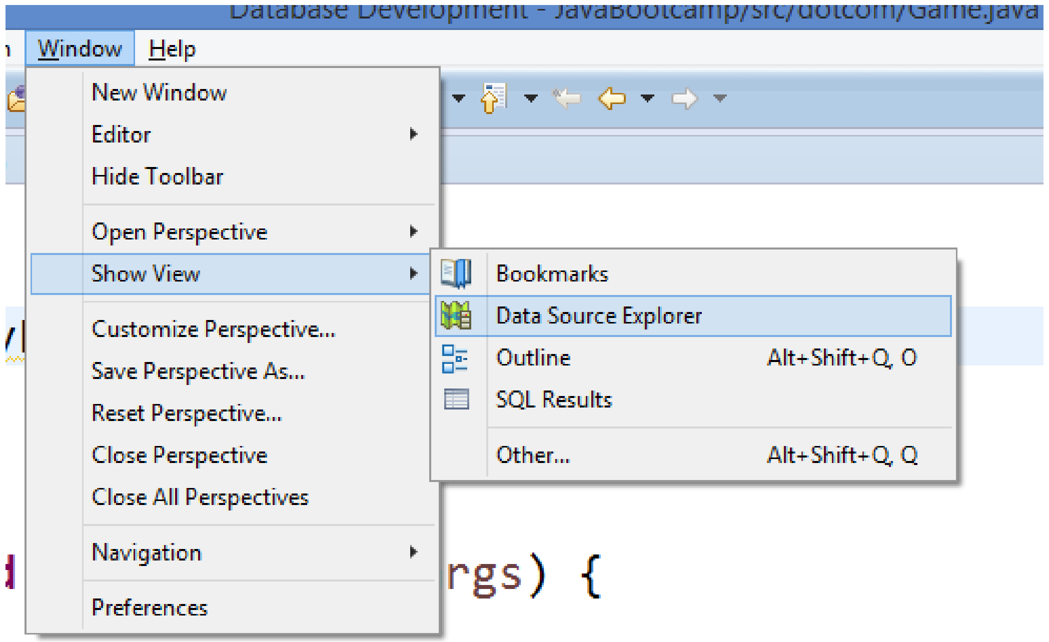


Open Perspective

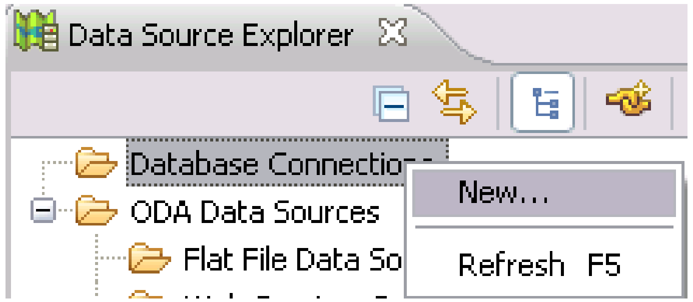
- Acceleo
- CVS Repository Exploring
- Database Debug
- Database Development
- Debug
- Git
- Java (default)
- Java Browsing
- Java Type Hierarchy
- Modeling
- Planning
- Plug-in Development
- Resource
- Team Synchronizing
- XML

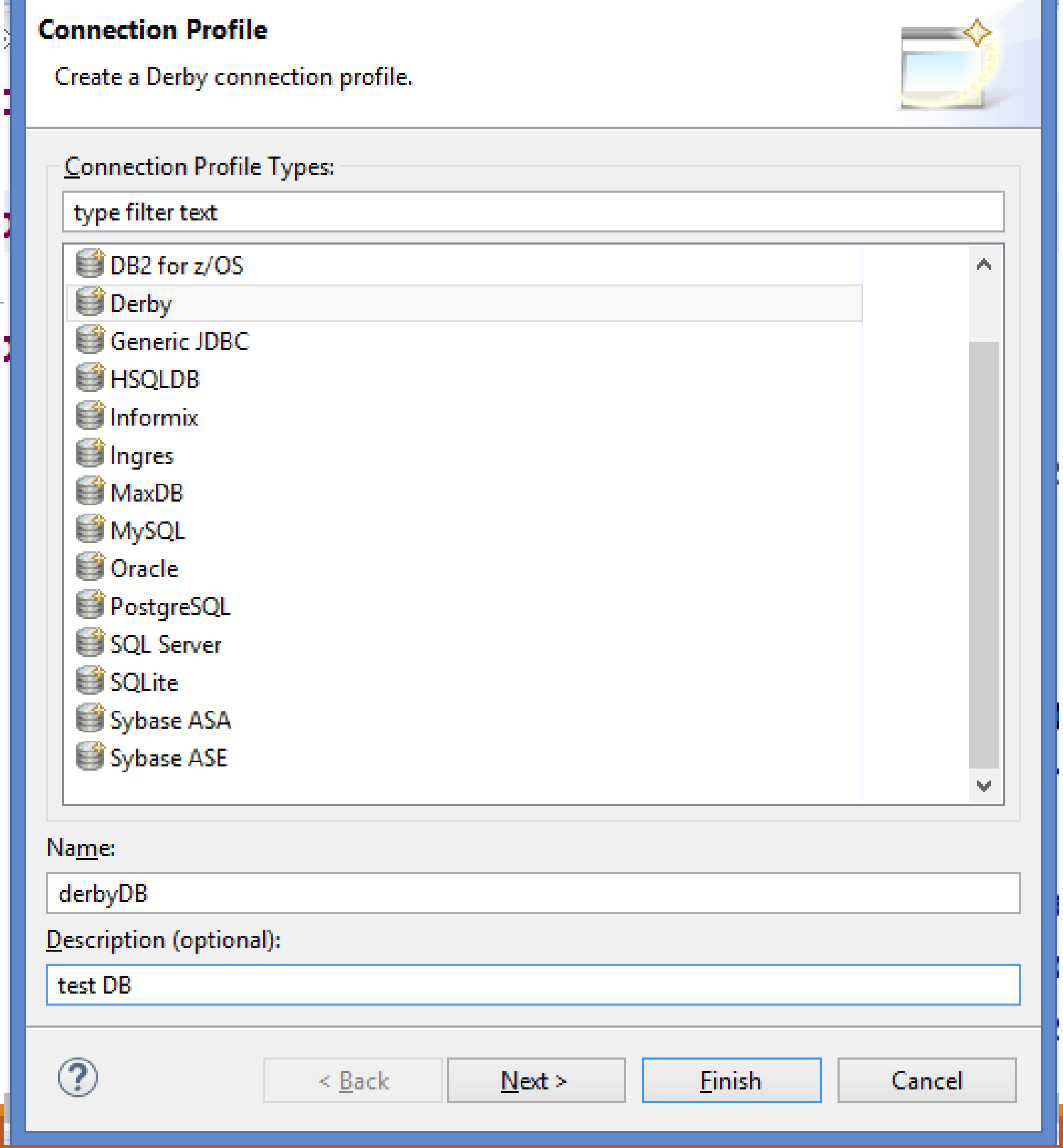


Open Database Explorer



New Connection




Connection Profile

Create a Derby connection profile.

Connection Profile Types:

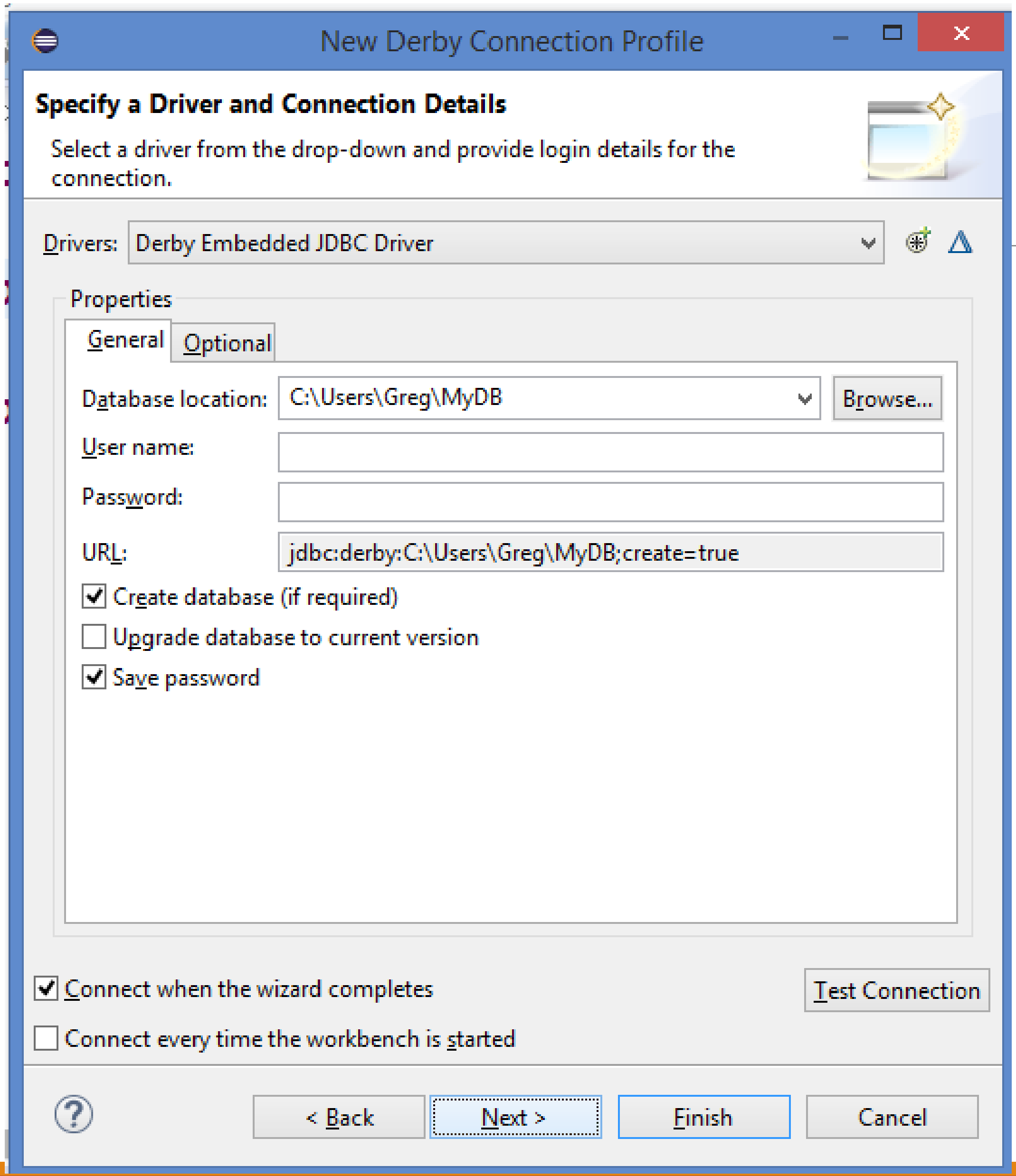
type filter text

- DB2 for z/OS
- Derby
- Generic JDBC
- HSQLDB
- Informix
- Ingres
- MaxDB
- MySQL
- Oracle
- PostgreSQL
- SQL Server
- SQLite
- Sybase ASA
- Sybase ASE

Name: derbyDB

Description (optional): test DB

New Derby Database



New Derby Connection Profile

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: Derby Embedded JDBC Driver

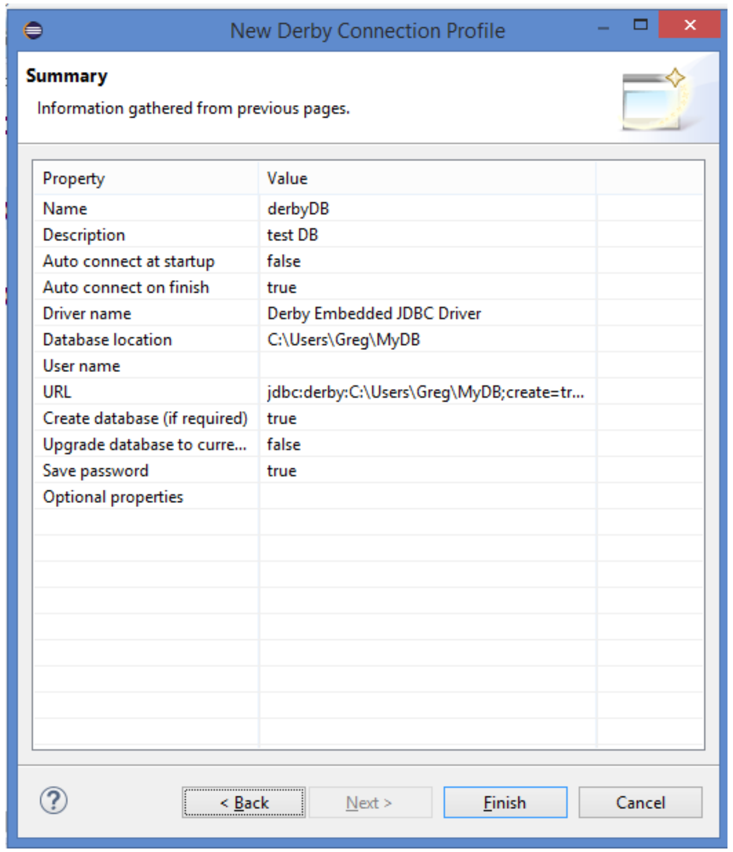
Properties

General | Optional

Database location:	C:\Users\Greg\MyDB
User name:	
Password:	
URL:	jdbc:derby:C:\Users\Greg\MyDB;create=true

- [x] Create database (if required)
- [ ] Upgrade database to current version
- [x] Save password

- [x] Connect when the wizard completes
- [ ] Connect every time the workbench is started



New Derby Connection Profile

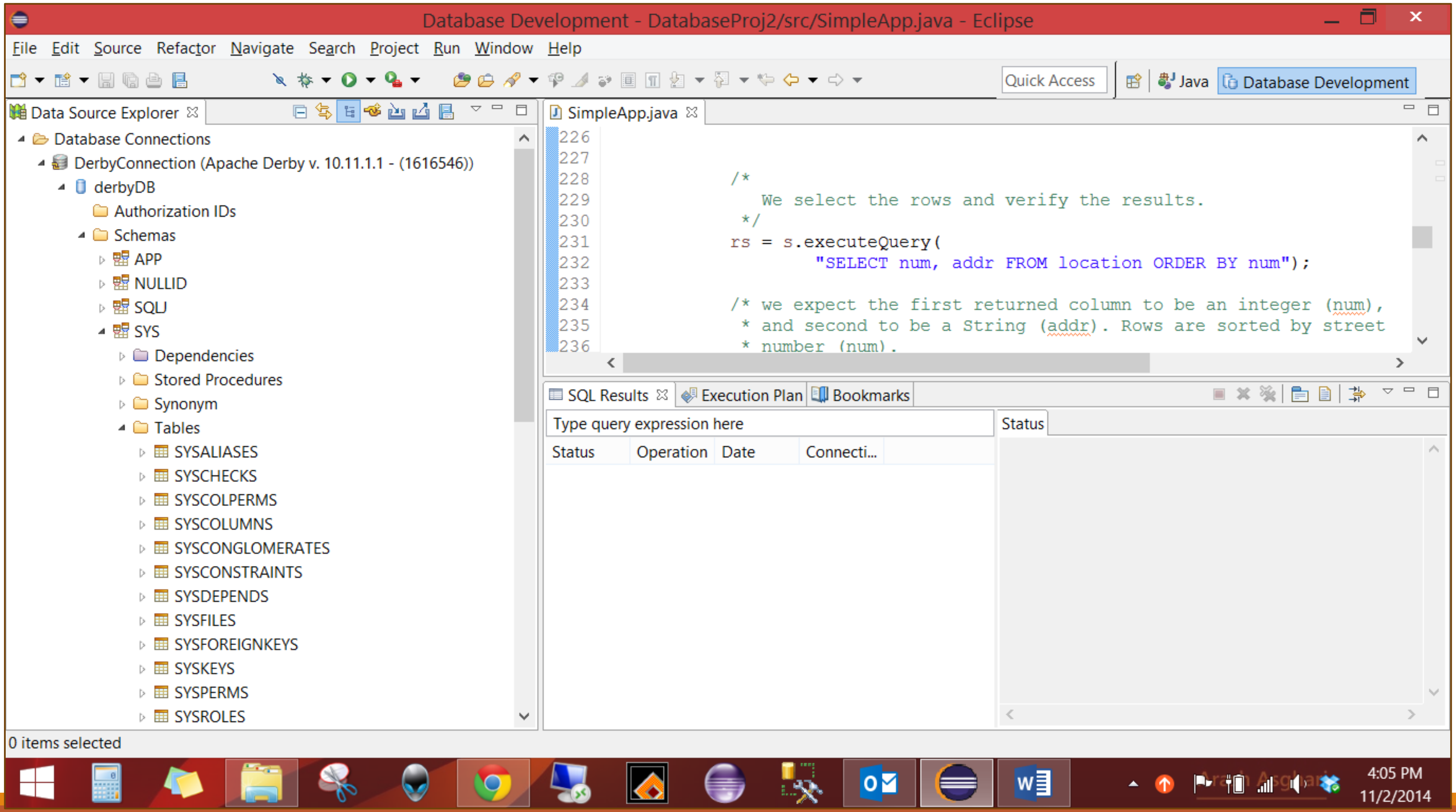
Summary

Information gathered from previous pages.

Property	Value
Name	derbyDB
Description	test DB
Auto connect at startup	false
Auto connect on finish	true
Driver name	Derby Embedded JDBC Driver
Database location	C:\Users\Greg\MyDB
User name	
URL	jdbc:derby:C:\Users\Greg\MyDB;create=tr...
Create database (if required)	true
Upgrade database to curre...	false
Save password	true
Optional properties	

Examining a database

To look at a database in Luna, we must first open the perspective Data Source Explorer. Then if not already present, we add our database into the database connections as shown in the graphic below:



Database Development - DatabaseProj2/src/SimpleApp.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Database Development
Data Source Explorer
SimpleApp.java
SQL Results
Execution Plan
Bookmarks

- Database Connections
  - DerbyConnection (Apache Derby v. 10.11.1.1 - (1616546))
    - derbyDB
      - Authorization IDs
      - Schemas
        - APP
        - NULLID
        - SQLJ
        - SYS
          - Dependencies
          - Stored Procedures
          - Synonym
          - Tables
            - SYSALIASES
            - SYSCHECKS
            - SYSCOLPERMS
            - SYSCOLUMNS
            - SYSCONGLOMERATES
            - SYSCONSTRAINTS
            - SYSDEPENDS
            - SYSFILES
            - SYSFOREIGNKEYS
            - SYSKEYS
            - SYSPERMS
            - SYSROLES

0 items selected

```
            /*
               We select the rows and verify the results.
             */
            rs = s.executeQuery(
                    "SELECT num, addr FROM location ORDER BY num");

            /* we expect the first returned column to be an integer (num),
             * and second to be a String (addr). Rows are sorted by street
             * number (num).
```

Type query expression here			
Status	Operation	Date	Connecti...

Status

4:05 PM
11/2/2014

Database drivers

A couple of terms to define:

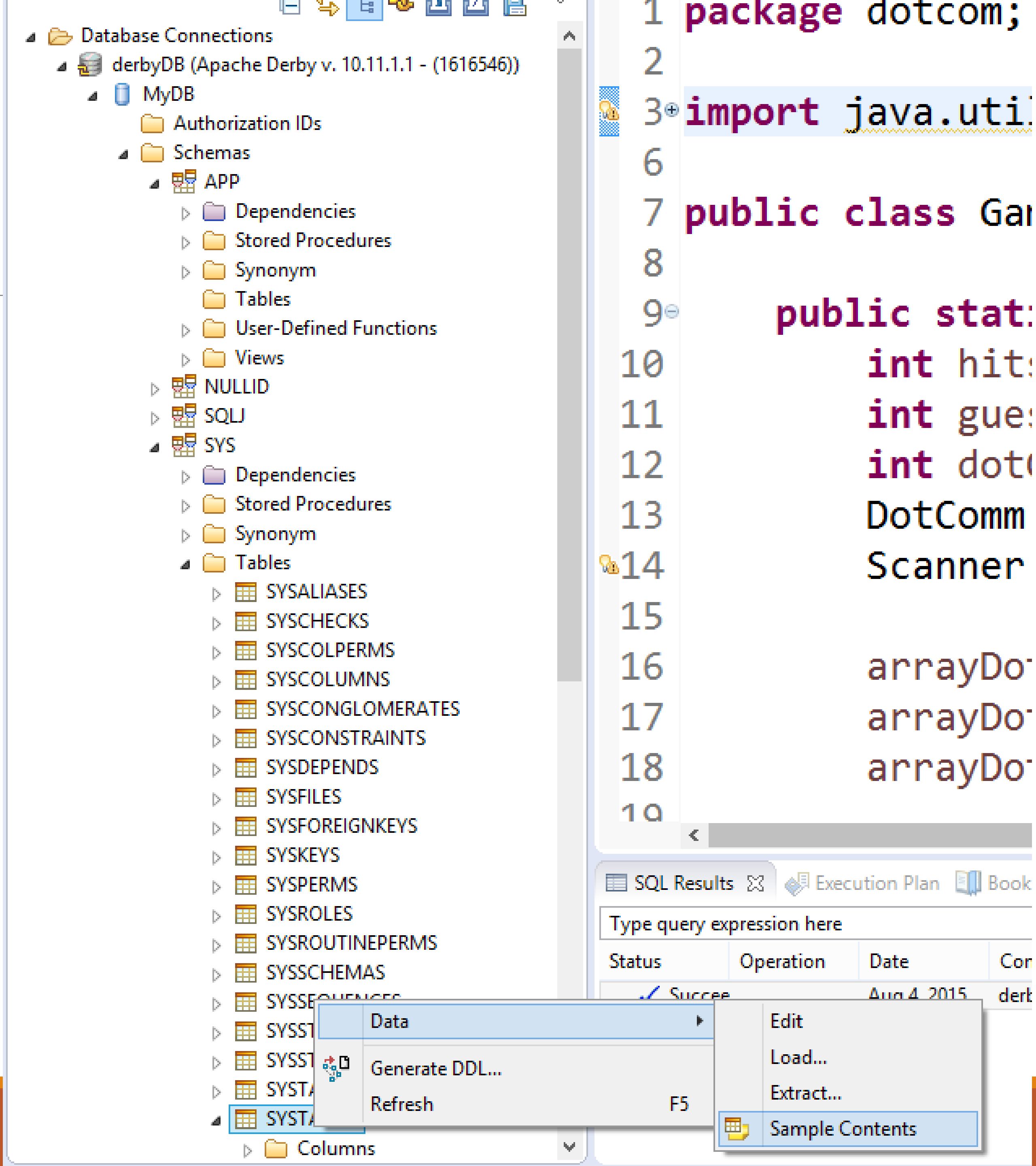
TABLE: A container for some related data such as customers, invoices, or items

ROW: Also known as a record. This is one unique instance of an element in a table. For example a record for customer John Smith would be one 1 of data that must uniquely identify John Smith and only John Smith.

COLUMN: Also known as a field. This is a single element in a table of data such as a Customer Id or Product Number.

Database Drivers

With these terms in mind, when we want to see the contents of a table such as SYSTABLES as shown in the above graphic, we can use Luna to do this as well as write a query against the table. In Luna, we right click on the table (SYSTABLES) -> Data -> Sample Contents. This is shown below.



Database Connections
- derbyDB (Apache Derby v. 10.11.1.1 - (1616546))
  - MyDB
    - Authorization IDs
    - Schemas
      - APP
        - Dependencies
        - Stored Procedures
        - Synonym
        - Tables
        - User-Defined Functions
        - Views
      - NULLID
      - SQLJ
      - SYS
        - Dependencies
        - Stored Procedures
        - Synonym
        - Tables
          - SYSALIASES
          - SYSCHECKS
          - SYSCOLPERMS
          - SYSCOLUMNS
          - SYSCONGLOMERATES
          - SYSCONSTRAINTS
          - SYSDEPENDS
          - SYSFILES
          - SYSFOREIGNKEYS
          - SYSKEYS
          - SYSPERMS
          - SYSROLES
          - SYSROUTINEPERMS
          - SYSSCHEMAS
          - SYSSE
          - SYSST
          - SYSST
          - SYSTA
          - SYSTA
            - Columns

```
package dotcom;

import java.uti

public class Ga

    public stat
        int hit
        int gue
        int dot
        DotComm
        Scanner

        arrayDo
        arrayDo
        arrayDo
```

Status	Operation	Date	Cor
✓ Succee		Aug 4, 2015	derb

Database Drivers

By using this method of seeing data, Luna is executing a SQL statement on our behalf which is basically the following:

Select * from SYS.SYSTABLES;

This statement selects all fields (*****) from the table called **SYS.SYSTABLES**. SYS is the database name and SYSTABLES is the table name.

Exercise 1: SysTableReporter.java P. 290-4

In this example we pull data from the systables table. Make sure the database is stopped since only one connection at a time can be used in embedded mode. Not doing so will cause a connection exception.

Writing records from a database

In SysTableReporter, we retrieved records from the database using SQL prepared as a string. This is a common method to call use SQL. The java.sql package supports the prepared statement as well. A **PREPARED STATEMENT** is represented by the **PreparedStatement** class is a pre-compiled statement. This enables the statement to return data more quickly and is often the preferred method in programming. We create the prepared statement by calling the connection's **preparedStatement(String)** method with the string that indicates the structure of the SQL statement. To indicate the structure, we write the SQL statement in which parameters have been replaced with question marks as shown below to serve as a sort of placeholder.

PreparedStatement ps = cc.prepareStatement("select * from SYS.TABLES where (TABLETYPE = ?)" ordered by TABLENAME");

Writing records from a database

When we execute, we pass the data we need to the placeholders. We can do this with the **setString(int, String)** method: ps.setString(1, “abcde”); -- where the first argument indicates the placeholder’s position and the is data to put in the statement. We have many methods available to us to set the variety of data types we that have such as integer, bytes, dates and so on. For example, the following statement puts a null value in the fifth position in a prepared statement called ps: ps.setNull(5, Types.Char);

Exercise 1: QuoteData.java, P. 294-6

Yahoo! offers a downloadable spreadsheet link on its home page for each ticker symbol. This link can be found at http://quote.yahoo.com/q?s=fb&d=v1 Below the price chart, there is a download data link. The file has the data stored in a single line of data of the latest close information. The data is in order by ticker symbol. It contains the symbol, closing price, date, time, price change since last close, daily low, daily high, daily open and the volume.

The application we are about to develop, QuoteData.java, uses each of these fields except one, the time, to load our stock data.

Exercise 1: QuoteData.java

The following actions take place:

A stock ticker symbol is used as a command-line argument

A QuoteData object is created with the ticker symbol as an instance variable called ticker

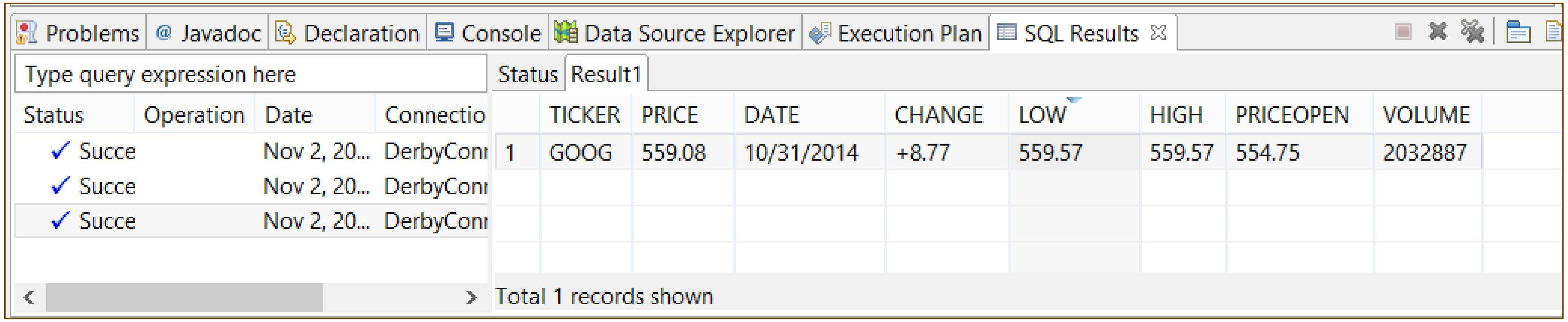
The object’s retrieveQuote() method is called to download the stock data from Yahoo! and returns it as a String

The object’s storeQuote() method is called with that String as an argument. It saves the data to a database.

Before we can begin, we must create a table in our database to hold the data. Please run the AddATable.java file in your student files to create the STOCKS table under the derbyDB for the USER1 schema.

Once we have the STOCKS table, code the following module and test it accordingly.

output



Problems | Javadoc | Declaration | Console | Data Source Explorer | Execution Plan | SQL Results

Type query expression here

Status	Operation	Date	Connectio
✓ Succe		Nov 2, 20...	DerbyConn
✓ Succe		Nov 2, 20...	DerbyConn
✓ Succe		Nov 2, 20...	DerbyConn

Status | Result1

	TICKER	PRICE	DATE	CHANGE	LOW	HIGH	PRICEOPEN	VOLUME
1	GOOG	559.08	10/31/2014	+8.77	559.57	559.57	554.75	2032887

Total 1 records shown

Let’s examine the code

This application stores data but does not produce output per se. We can use the extract feature in Luna to see that GOOG was entered in the database table STOCKS.

Lines 13 – 34: Download the Yahoo! data and save it as a string

Lines 36 – 69: Use SQL to store the data to the database. We use StringTokenizer to split the data into its tokens on the comma character that is between each token. We then store this in a String array with nine elements. The array contains the same elements as the table we created.

Lines 42 – 46: Connect to the database

Lines 49 – 53: Prepare our statements – an insert statement is used to store the data

Lines 54 – 61: Put the elements of the String array into the prepared statement using a series of setString() method. They must be put in the statement in the order of the database columns

Line 62: Execute the statements after the prepared statement has been filled using the executeUpdate() – it will either add it to the database or throw a SQL error.

Line 40: Strip any excess quotes from the data using the stripQutoes() private method

Moving through a result set

Since we used prepared statements we have the following methods available to us to help navigate the set. This is because the result set's policies have been specified as arguments to the connection's **createStatement()** and **prepareStatement()** methods. The **ResultSet** class includes many class variables to give us more flexibility when working with sets.

- afterLast() – go to the place immediately after the last record

- beforeFirst() – go to the place immediately before the first record

- first() – go to the first record

- last() – go to the last record

- previous() – go to the previous record in the set

Summary

We've learned to read and write database records using classes that work with the most popular relational database products. The only difference is that the database driver must be specific to the product we are using.

We also received a very brief view into database technologies and how a program communicates with it to retrieve and store data.

Independent exercises, P. 297

A. Modify the SysTableReporter application to pull fields from another table in SYS. You will have to make modifications to the connection strings and the drivers for this application. The Access database is in the student files.

B. Write an application that reads and displays records from Yahoo! Stock quote database.