# JDBC

The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a [Relational Database.](http://docs.oracle.com/javase/tutorial/jdbc/overview/#relational)

JDBC helps you to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to your query

The following simple code fragment gives a simple example of these three steps:

public void connectToAndQueryDatabase(String username, String password) {

Connection con = DriverManager.getConnection(

"jdbc:myDriver:myDatabase",

username,

password);

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

while (rs.next()) {

int x = rs.getInt("a");

String s = rs.getString("b");

float f = rs.getFloat("c");

}

}

This short code fragment instantiates a DriverManager object to connect to a database driver and log into the database, instantiates a Statement object that carries your SQL language query to the database; instantiates a ResultSet object that retrieves the results of your query, and executes a simple while loop, which retrieves and displays those results.

**JDBC Product Components**

JDBC includes four components:

**The JDBC API** —  The JDBC™ API provides programmatic access to relational data from the Java™ programming language. Using the JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to an underlying data source. The JDBC API can also interact with multiple data sources in a distributed, heterogeneous environment.

**JDBC Driver Manager** —  The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManagerhas traditionally been the backbone of the JDBC architecture. It is quite small and simple.

**JDBC Architecture**

**Two-tier and Three-tier Processing Models**

The JDBC API supports both two-tier and three-tier processing models for database access.

In the two-tier model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user.

**Processing SQL Statements with JDBC**

In general, to process any SQL statement with JDBC, you follow these steps:

1. [Establishing a connection.](http://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html#establishing_connections)
2. [Create a statement.](http://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html#creating_statements)
3. [Execute the query.](http://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html#executing_queries)
4. [Process the ResultSet object.](http://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html#processing_resultset_objects)
5. [Close the connection.](http://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html#closing_connections)

## Establishing Connections

First, establish a connection with the data source you want to use.  This connection is represented by a Connection object.

Typically, a JDBC application connects to a target data source using one of two classes:

* DriverManager: This fully implemented class connects an application to a data source, which is specified by a database URL. When this class first attempts to establish a connection, it automatically loads any JDBC 4.0 drivers found within the class path. Note that your application must manually load any JDBC drivers prior to version 4.0.
* DataSource: This interface is preferred over DriverManager because it allows details about the underlying data source to be transparent to your application. A DataSource object's properties are set so that it represents a particular data source.

**Using the DriverManager Class**

Connecting to your DBMS with the DriverManager class involves calling the method DriverManager.getConnection.

The method DriverManager.getConnection establishes a database connection. This method requires a database URL, which varies depending on your DBMS.

This method specifies the user name and password required to access the DBMS with a Properties object.

*In previous versions of JDBC, to obtain a connection, you first had to initialize your JDBC driver by calling the method Class.forName. This methods required an object of type java.sql.Driver. Each JDBC driver contains one or more classes that implements the interface java.sql.Driver. The drivers for Java DB are org.apache.derby.jdbc.EmbeddedDriver and org.apache.derby.jdbc.ClientDriver, and the one for MySQL Connector/J is com.mysql.jdbc.Driver. See the documentation of your DBMS driver to obtain the name of the class that implements the interfacejava.sql.Driver.*

The method returns a Connection object, which represents a connection with the DBMS or a specific database. Query the database through this object.

**Specifying Database Connection URLs**

A database connection URL is a string that your DBMS JDBC driver uses to connect to a database. It can contain information such as where to search for the database, the name of the database to connect to, and configuration properties. The exact syntax of a database connection URL is specified by your DBMS.

**Retrieving and Modifying Values from Result Sets**

A ResultSet object is a table of data representing a database result set, which is usually generated by executing a statement that queries the database.

**ResultSet Interface**

The ResultSet interface provides methods for retrieving and manipulating the results of executed queries, and ResultSet objects can have different functionality and characteristics. These characteristics are type, concurrency, and cursor *holdability*.

**Retrieving Column Values from Rows**

The ResultSet interface declares getter methods (for example, getBoolean and getLong) for retrieving column values from the current row. You can retrieve values using either the index number of the column or the alias or name of the column. The column index is usually more efficient. Columns are numbered from 1. For maximum portability, result set columns within each row should be read in left-to-right order, and each column should be read only once.