**JDBC Overview**

http://www.oracle.com/technetwork/java/overview-141217.html

**Contents**

|  |  |
| --- | --- |
| - | [JDBC API Overview](http://www.oracle.com/technetwork/java/overview-141217.html#1) |
| - | [JDBC Architecture](http://www.oracle.com/technetwork/java/overview-141217.html#2) |
| - | [Partnering for Progress](http://www.oracle.com/technetwork/java/overview-141217.html#3) |
| - | [Industry Momentum](http://www.oracle.com/technetwork/java/overview-141217.html#4) |
| - | [Advantages of JDBC Technology](http://www.oracle.com/technetwork/java/overview-141217.html#5) |
| - | [Key Features](http://www.oracle.com/technetwork/java/overview-141217.html#6) |
| - | [Related Documents](http://www.oracle.com/technetwork/java/overview-141217.html#7) |

The JDBC API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC API provides a call-level API for SQL-based database access. JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data.

**JDBC API Overview**

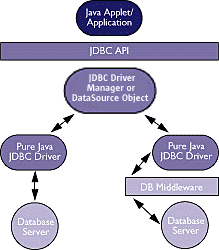
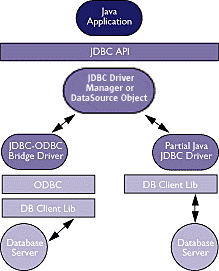
|  |
| --- |
|  |

The JDBC API makes it possible to do three things:

* Establish a connection with a database or access any tabular data source
* Send SQL statements
* Process the results

**JDBC Architecture**

|  |
| --- |
|  |

The JDBC API contains two major sets of interfaces: the first is the JDBC API for application writers, and the second is the lower-level JDBC driver API for driver writers. JDBC technology drivers fit into one of four categories. Applications and applets can access databases via the JDBC API using pure Java JDBC technology-based drivers, as shown in this figure:   
  
  
  
*Left side, Type 4: Direct-to-Database Pure Java Driver*   
This style of driver converts JDBC calls into the network protocol used directly by DBMSs, allowing a direct call from the client machine to the DBMS server and providing a practical solution for intranet access.   
  
*Right side, Type 3: Pure Java Driver for Database Middleware*   
This style of driver translates JDBC calls into the middleware vendor's protocol, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases.   
  
The graphic below illustrates JDBC connectivity using ODBC drivers and existing database client libraries.   
  
  
  
*Left side, Type 1: JDBC-ODBC Bridge plus ODBC Driver*   
This combination provides JDBC access via ODBC drivers. ODBC binary code -- and in many cases, database client code -- must be loaded on each client machine that uses a JDBC-ODBC Bridge. Sun provides a JDBC-ODBC Bridge driver, which is appropriate for experimental use and for situations in which no other driver is available.   
  
*Right side, Type 2: A native API partly Java technology-enabled driver*   
This type of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.   
  
For comparison of driver types, please see the article published in Computerworld.

**Partnering for Progress**

|  |
| --- |
|  |

Sun worked with an array of companies in the industry to create and rapidly establish the JDBC API as the industry-standard, open interface for Java applications to access databases.

**Industry Momentum**

|  |
| --- |
|  |

Leading database, middleware and tool vendors have been building support for JDBC technology into many new products. This ensures that customers can build portable Java applications while choosing from a wide range of competitive products for the solution best suited to their needs. See the [Industry Support](http://www.oracle.com/technetwork/java/index-136695.html) page for a list of companies that are shipping products with support for JDBC technology.

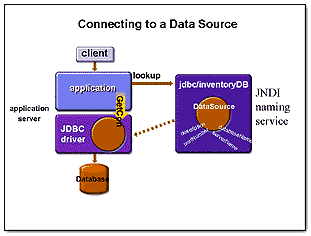
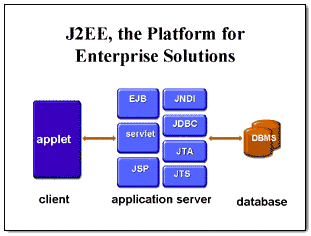
**Advantages of JDBC Technology**

|  |
| --- |
|  |

***Leverage Existing Enterprise Data***   
With JDBC technology, businesses are not locked in any proprietary architecture, and can continue to use their installed databases and access information easily -- even if it is stored on different database management systems.   
  
***Simplified Enterprise Development***   
The combination of the Java API and the JDBC API makes application development easy and economical. JDBC hides the complexity of many data access tasks, doing most of the "heavy lifting"for the programmer behind the scenes. The JDBC API is simple to learn, easy to deploy, and inexpensive to maintain.   
  
***Zero Configuration for Network Computers***   
With the JDBC API, no configuration is required on the client side. With a driver written in the Java programming language, all the information needed to make a connection is completely defined by the JDBC URL or by a DataSource object registered with a Java Naming and Directory Interface (JNDI) naming service. Zero configuration for clients supports the network computing paradigm and centralizes software maintenance.

**Key Features**

|  |
| --- |
|  |

***Full Access to Metadata***   
The JDBC API provides metadata access that enables the development of sophisticated applications that need to understand the underlying facilities and capabilities of a specific database connection.   
  
***No Installation***   
A pure JDBC technology-based driver does not require special installation; it is automatically downloaded as part of the applet that makes the JDBC calls.   
  
***Database Connection Identified by URL***   
JDBC technology exploits the advantages of Internet-standard URLs to identify database connections. The JDBC API includes an even better way to identify and connect to a data source, using a DataSource object, that makes code even more portable and easier to maintain.   
  
  
  
In addition to this important advantage, DataSource objects can provide connection pooling and distributed transactions, essential for enterprise database computing. This functionality is provided transparently to the programmer.   
  
***Included in the Java Platform***   
As a core part of the Java 2 Platform, the JDBC API is available anywhere that the platform is. This means that your applications can truly write database applications once and access data anywhere. The JDBC API is included in both the Java 2 Platform, Standard Edition (J2SE) and the Java 2 Platform, Enterprise Edition (J2EE), providing server-side functionality for industrial strength scalability.   
  
An example of a J2EE based architecture that includes a JDBC implementation:   
  
  
  
***Requirements***

* Software: The Java 2 Platform (either the Java 2 SDK, Standard Edition, or the Java 2 SDK, Enterprise Edition), an SQL database, and a JDBC technology-based driver for that database.
* Hardware: Same as for the Java 2 Platform.

# JDBC – Introduction

# http://www.tutorialspoint.com/jdbc/jdbc-introduction.htm

## What is JDBC?

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

* Making a connection to a database.
* Creating SQL or MySQL statements.
* Executing SQL or MySQL queries in the database.
* Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

* Java Applications
* Java Applets
* Java Servlets
* Java ServerPages (JSPs)
* Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

## Pre-Requisite

Before moving further, you need to have a good understanding of the following two subjects −

* [Core JAVA Programming](http://www.tutorialspoint.com/java/index.htm)
* [SQL or MySQL Database](http://www.tutorialspoint.com/mysql/index.htm)

## JDBC Architecture

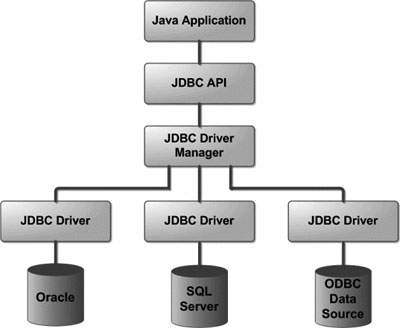
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

* **JDBC API:** This provides the application-to-JDBC Manager connection.
* **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



## Common JDBC Components

The JDBC API provides the following interfaces and classes −

* **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
* **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
* **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
* **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
* **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
* **SQLException:** This class handles any errors that occur in a database application.

## The JDBC 4.0 Packages

The java.sql and javax.sql are the primary packages for JDBC 4.0. This is the latest JDBC version at the time of writing this tutorial. It offers the main classes for interacting with your data sources.

The new features in these packages include changes in the following areas −

* Automatic database driver loading.
* Exception handling improvements.
* Enhanced BLOB/CLOB functionality.
* Connection and statement interface enhancements.
* National character set support.
* SQL ROWID access.
* SQL 2003 XML data type support.
* Annotations.