JDBC [Java Database Connectivity]

JDBC is an API (Application programming Interface) that helps a programmer to write java program to connect to a database, retrieve the data from the database and perform various operations on the data in the java program.
The java database connectivity application programming interface is a set of specifications that defines how a java program can communicate with the database.
The API contains a set of classes & Interfaces to enable programmers to communicate with a database using java.
The JDBC API makes it possible to do three things:

i. Establish a connection with a data source.
ii. Send queries and update statements to the data source.
iii. iii. Process the results.

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 −

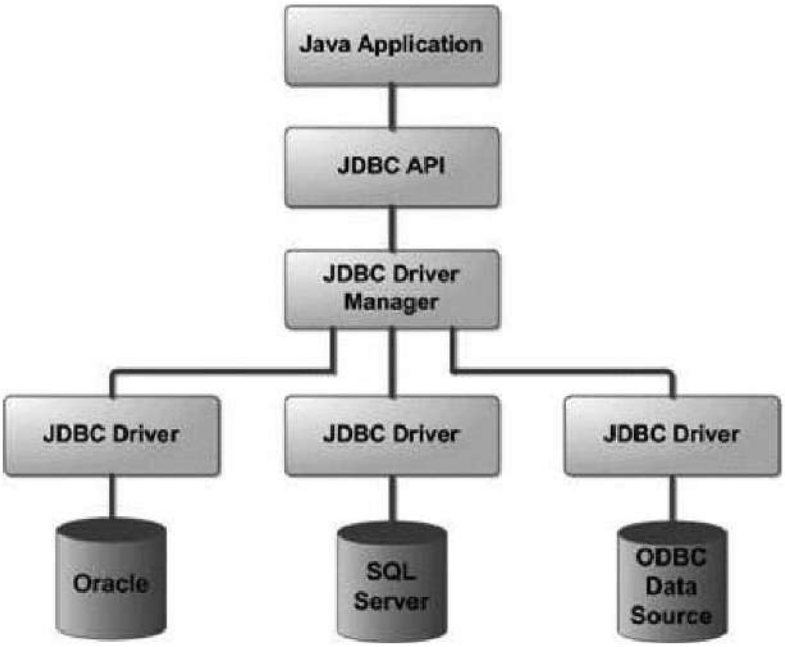
- JDBC API − This provides the application-to-JDBC Manager connection.
- JDBC Driver API − This supports the JDBC Manager-to-Driver Connection.

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

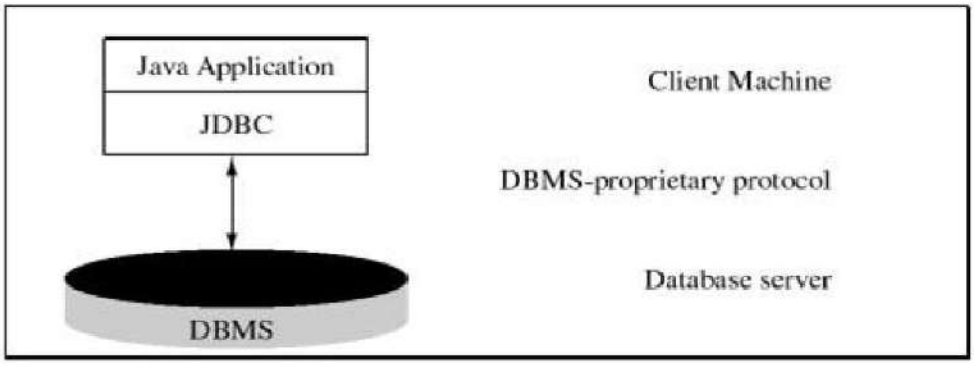
The JDBC driver manager ensures that the correct driver is used to access each data source.

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 −



Two-tier Architecture

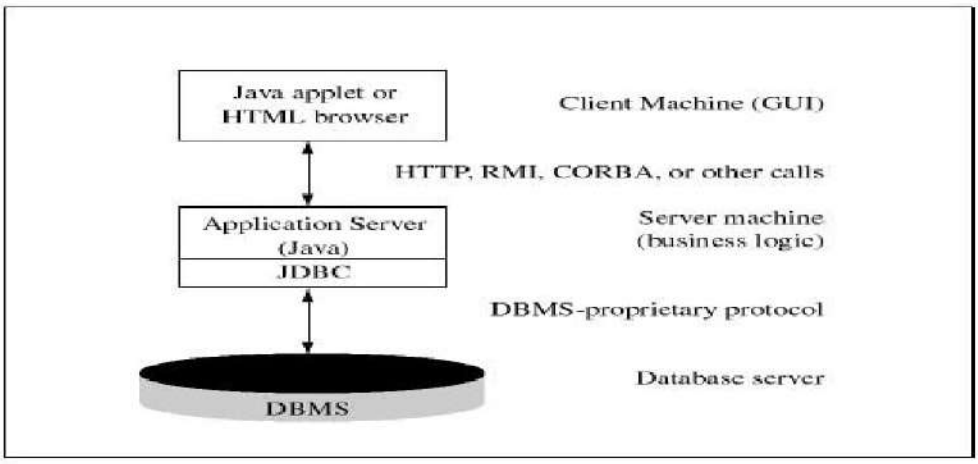


In the two-tier model, a Java applet or application talks directly to the database.
This requires a JDBC driver that can communicate with the particular database management system being accessed.
A user's SQL statements are delivered to the database, and the results of those statements are sent back to the user.
The database may be located on another machine to which the user is connected via a network.
This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the database as the server.
The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

Advantages
1) It is easy to implement.
2) It maintains a persistent connection between the client and the database thereby eliminating overhead associated with opening and closing connections.
3) It is faster than three-tier system.

Disadvantages
1) Native libraries must be loaded on each client machine.
2) Applets can only open up connections to the server from which they were downloaded.
3) This requires that web server and database server should be present on the same machine.

Three-tier Architecture



In the three-tier model, commands are sent to a "middle tier" of services, which then send SQL statements to the database.
The database processes the SQL statements and sends the results back to the middle tier, which then sends them to the user.
The three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data.
Another advantage is that when there is a middle tier, the user can employ an easy-to-use higher-level API which is translated by the middle tier into the appropriate low-level calls.
In many cases the three-tier architecture can provide performance advantages.

Advantages

i) Clients do not need to have native libraries loaded locally,
ii) Drivers can be managed centrally,
iii) Database server is not directly visible to the internet.

Disadvantages

i) The client does not maintain a persistent database connection.
ii) A separate proxy server may be required.
iii) The communication between the clients and the server is slower because the client calls must be translated into network protocol and then they must be translated into specific DB calls.

Common JDBC Components

The JDBC API provides the following interfaces and classes

1) **DriverManager**
   This class manages a list of database drivers. It matches connection requests from the java application with the proper database driver using communication sub protocol.

2) **Driver**
   This interface handles the communications with the database server. You can interact directly with driver objects rarely but instead you can use DriverManager objects, which manages objects of this type.

3) **Connection**
   This interface handles all methods for connecting to a database. The connection object represents communication context i.e. all communication with database is done through the connection object only.

4) **Statement**
   Objects created from this interface are used to submit the SQL statements to the database. Some derived interfaces accept the parameters in addition to executing stored procedures.

5) **ResultSet**
   These objects hold data retrieved from a database after you execute an SQL query using Statement object. It acts as an iterator which allows you to move through its data.

6) **SQLException**
   This class handles any errors that occur in a database application.

JDBC Drivers

JDBC drivers are set of classes that enable the JAVA application/program to communicate with database.
It also ensure that the request made by the application presented to the database in a language understood by the database.
JDBC driver does the following tasks.

1) The JDBC driver receives the request from the client.
2) It converts the request into the format understandable by the database.
3) It then submits the requests to the database.
4) The JDBC driver receives the response from the database.
5) It translates the response back to JAVA data format.
6) It then submits the response to the client application.

There are 4 types of JDBC drivers:

1) Type 1: JDBC-ODBC Bridge driver
2) Type 2: Native- API partly Java driver
3) Type 3: JDBC- Net pure Java Driver
4) Type 4: Native-protocol pure Java Driver

1) **Type 1: JDBC-ODBC Bridge driver**
   - The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.
   - The JDBC-ODBC bridge driver receives any JDBC call and sends them to ODBC driver.
   - ODBC driver understands these calls and communicates with the database library provided by the vendor so ODBC driver & the vendor database library must be present on client machine.
   - In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine.

Advantages of type1 driver

1) It is easy to use.
2) The JDBC-ODBC Bridge driver allows access to all the databases since ODBC driver are already available on the client machine..

Disadvantages of type1 driver

1) This driver is not fully written in JAVA hence it is not portable.
2) The performance of this driver is less because the JDBC call goes through the bridge driver to the ODBC driver then to the database & this applies in reverse manner also.
3) The ODBC driver needs to be installed on the client machine.

2) **Type 2: Native- API partly Java driver**
   - Native-API partly Java driver enables Java programs to communicate with the database using database-specific APIs which are normally written in C or C++.
   - The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API.
   - These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor specific driver must be installed on each client machine. If we change the database we have to change the native API.

Advantages

1) Typr-2 driver offers better performance offers better performance the type-1 driver.
2) Type-2 driver uses database specific API's (Nativ API's) hence communication is faster.

Disadvantages

1) This driver is not fully written in JAVA hence it is not portable.
2) The native driver needs to be installed on the client machine.
3) The vendor client library needs to be installed on the client machine.
4) This diver is not suitable for web.
5) Type-2 driver have slower performance than Type-3 and Type-4 drivers.

3) **Type 3: JDBC- Net pure Java Driver**
   - This driver translates JDBC calls into a database independent net protocol, which is then translated to a DBMS protocol by a net server.
   - The Network Protocol driver uses middleware (i.e. application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.
   - This kind of driver is flexible as it requires no code installed on the client machine and a single driver can actually provide access to multiple databases.

Advantages

1) This driver is fully written in JAVA hence it is portable.
2) This diver is suitable for web.
3) This driver is server based, so there is no need for any vendor database library to be present on client machine.
4) The client driver need not be changed for a new database.

Disadvantages

1) It requires a server application to be installed and maintained.
2) Accessing of data may take longer time, since the data comes from the backend server.

4) **Type 4: Native-protocol pure Java Driver**
   - This driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.
   - This is the highest performance driver available for the database and is usually provided by the vendor.
   - This driver is extremely flexible as you don't need to install any special software on the client or server.
   - These drivers can be downloaded dynamically.

Advantages

1) This driver is fully written in JAVA hence it is portable.
2) This diver is suitable for web.
3) The numbers of translations are very less.
4) This driver has better performance than Type-1 and Type-2 driver also there is no need to install any special software on the client or server.

Disadvantages

1) In Type-4 driver, user needs different drivers for each database.
   Eg:- to communicate with ORACLE server we need ORACLE driver.

Basic JDBC Steps

All JDBC programs follow some basic steps to communicate with the database.

1) Load the JDBC driver
2) Establish the Connection with database
3) Create a Statement Object
4) Execute a query
5) Process the data returned by database
6) Close the connection

1) Load the JDBC driver

To communicate with any database we have to use a driver.
Before establishing the actual connection with database JDBC driver must be loaded.

There are two ways to load and register the driver.

a) Using Class.forName() method

- Class.forName() is a static method of the class "Class".
- This method loads the specified driver class.
- When the getConnection method is used, the DriverManager selects the appropriate driver from the loaded drivers.

- Syntax

```
Class.forName("driverName");
```

- Example

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Class.forName("org.gjt.mm.mysql.Driver");
Class.forName("org.postgresql.Driver");
```

b) Explicit registration using registerDriver method

- The registerDriver is a static method of the DriverManager class. It is used to explicitly register the newly loaded driver.

- Syntax

```
DriverManager.registerDriver(driver);
```

- Example

```
Driver myDriver = (Driver) Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
DriverManager.registerDriver(myDriver);
```

2) Establish the Connection with database

- A connection represents a session with a specific database.
- Within the context of a connection, SQL statements are executed and results are returned.
- There can be multiple connections to a database.
- A connection also provides information about the database, tables and fields i.e. "metadata".
- To establish a connection you can use DriverManager.getConnection() method. This method takes three arguments: a JDBC URL, a database username and a password. It returns a connection object and throws java.sql.SQLException.

- Syntax

```
DriverManager.getConnection("url", "user", "password");
```

- Example

```
Connection con=DriverManager.getConnection(jdbc:odbc:DB");
```

3) Create a Statement Object

Once connection is created, you can use it to execute SQL statements.

This is done by using statement object.

There are three kinds of statements in JDBC.

Method	Description
Statement	• A Statement represents a general SQL statement without parameters. • The method createStatement() creates a Statement object.
PreparedStatement	• A PreparedStatement represents a precompiled SQL statement, with or without parameters. • The method prepareStatement(String sql) creates a PreparedStatement object.
CallableStatement	• CallableStatement objects are used to execute SQL stored procedures. • The method prepareCall(String sql) creates a CallableStatement object.

4) Execute a Query

The following methods of the statement interface are used to execute the SQL statements.

Method	Description
ResultSet executeQuery(String sql)	It is used for statements that return an output result. Example:- SELECT
int executeUpdate(String sql)	It is used for the statements which update the database, that is, that do not return an output. The executeUpdate returns an integer value specifying the number of rows affected by query. Example:- UPDATE, DELETE, CREATE TABLE, DROP TABLE and ALTER TABLE
boolean execute(String sql)	It is used when there may be multiple results returned. Used when we don't know whether an SQL statement is going to return the results. If there is a result it can be obtained from the getResultSet() method. Example :- String query = "drop table if exists student"; boolean ans = stmt.execute(query);

5) Process the Result (data returned by database)

A ResultSet object contains the result of an executing SQL query.
The data is stored in a ResultSet in a tabular manner with column heading & the corresponding values returned by a query.

To access these values following methods are used
Using Column name:- getXXX(String ColumnName)
Using Column Number:- getXXX(String ColumnNumber)

6) Close the Connection

After all the tasks done it is important to close the connection. Before closing the connection close the ResultSet and the Statement object. The close() method is used to close the objects.

Examples:- stmt.close();
rs.close();
con.close()

ResultSet (Scrollable and Updatable)

The query we fire on database will return some result if there is no problem in the query.
The results are stored in ResultSet in java.
The default type of ResultSet is Forward-only and Read-only which means that we can traverse only in one direction.
We can also create a ResultSet in following manner
Scrollable: It indicates how the cursor moves in ResultSet.
Updatable: It indicates the changes made to the underlying tables by other users are shown in ResultSet or not.

Syntax:
Statement stmt = con.createStatement(int scrollType, int concurrencyType);

ResultSet Scroll Types
ResultSet.TYPE_FORWARD_ONLY
ResultSet.TYPE_SCROLL_SENSITIVE
ResultSet.TYPE_SCROLL_INSENSITIVE

ResultSet Concurrency
ResultSet.CONCUR_READ_ONLY
ResultSet.CONCUR_UPDATABLE

Scroll Type	
TYPE_FORWARD_ONLY	The result set is not scrollable.
TYPE_SCROLL_INSENSITIVE	The result set is scrollable but not sensitive to database changes.
TYPE_SCROLL_SENSITIVE	The result set is scrollable and sensitive to database changes.
Concurrency Type	
CONCUR_READ_ONLY	The result set cannot be used to update the database.
CONCUR_UPDATABLE	The result set can be used to update the database.

ResultSet Interface

The ResultSet interface provides methods for retrieving and manipulating the results of executed queries.

Method	Description
void beforeFirst()	Moves the cursor to the front of this ResultSet object, just before the first row.
void afterLast()	Moves the cursor to the end of this ResultSet object, just after the last row.
boolean first()	Moves the cursor to the first row in this ResultSet object.
boolean last()	Moves the cursor to the last row in this ResultSet object.
boolean next()	Moves the cursor forward one row from its current position.
boolean previous()	Moves the cursor to the previous row in this ResultSet object.
boolean absolute(int row)	Moves the cursor to the given row number in this ResultSet object.
boolean relative(int rows)	Moves the cursor a relative number of rows, either positive or negative.
int getRow()	Retrieves the current row number
boolean isFirst()	Retrieves whether the cursor is on the first row of this ResultSet object.
boolean isLast()	Retrieves whether the cursor is on the last row of this ResultSet object.
boolean isBeforeFirst()	Retrieves whether the cursor is before the first row in this ResultSet object.
boolean isAfterLast()	Retrieves whether the cursor is after the last row in this ResultSet object.
void refreshRow()	Refreshes the current row with its most recent value in the database.
void insertRow()	Inserts the contents of the insert row into this ResultSet object and into the database.
void deleteRow()	Deletes the current row from this ResultSet object and from the underlying database.
void updateRow()	Updates the underlying database with the new contents of the current row of this ResultSet object.
getXXX(String columnName)	Retrieves the value of the designated column in the current row as a corresponding type in the Java programming language. XXX represents a type: Int, String, Float, Short, Long, Time etc.
updateXXX(int columnNumber, XXX value)	Updates the value of the designated column in the current row as a corresponding type in the Java programming language. XXX represents a type: Int, String, Float, Short, Long, Time etc.
void close()	Releases this ResultSet object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.

Metadata

Metadata means the data about database data or the information about tables, views, column types, column names, result set and stored procedures.

Uses of Metadata

JDBC Metadata API is used to retrieve the following information about the database.

- Database schema and catalog information
- Database user tables, views, stored procedures
- Table, view and column privileges
- Information about primary key and foreign key of a table

JDBC provides four important interfaces to retrieve the metadata

Metadata	Description
DatabaseMetaData	This interface provides information about the database as a whole such as table name, indexes and version and product name.
ResultSetMetaData	It provides information about the ResultSet object returned from a databse query. It is used to find out number of columns, its name, type, length, table name , whether a column is readable/searchable or writable and so on.
ParameterMetaData	It is used to get information about the types and properties of the parameters in the PreparedStatement object such as getting parameter count, it's types, designated precision specified by the column size.
RowSetMetaData	It provides the information about the columns in a RowSet object and can be used to find out number of columns contains in a rowset or the type of data in each column.

DatabaseMetaData

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

```
DatabaseMetaData dbmd=con.getMetaData();
```

Methods	Description
String getURL()	Retrieves the URL for this DBMS.
String getUserName()	Retrieves the user name as known to this database.
String getDatabaseProductName()	Retrieves the name of this database product.
String getDatabaseProductVersion()	Retrieves the version number of this database product.
int getDatabaseMajorVersion()	Retrieves the major version number of the underlying database.
int getDatabaseMinorVersion()	Retrieves the minor version number of the underlying database.
String getDriverName()	Retrieves the name of this JDBC driver.
String getDriverVersion()	Retrieves the version number of this JDBC driver as a String.
int getDriverMajorVersion()	Retrieves this JDBC driver's major version number.
int getDriverMinorVersion()	Retrieves this JDBC driver's minor version number.
ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	Retrieves a description of the tables available in the given catalog.

DatabaseMetaData Program

```
import java.io.*;
import java.sql.*;
class DatabaseMetaDataDemo
{
	public static void main(String args[])throws Exception
	{
		Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
		Connection con = DriverManager.getConnection("Jdbc:Odbc:DB");
		DatabaseMetaData ob = con.getMetaData();
		System.out.println("Database Name = "+ob.getDatabaseProductName());
		System.out.println("Database Version = "+ob.getDatabaseProductVersion());
		System.out.println("Database Driver Name = "+ob.getDriverName());
		System.out.println("Database URL = "+ob.getURL());
		System.out.println("Database Major Version = "+ob.getDriverMajorVersion());
		System.out.println("Database Manor Version = "+ob.getDriverMinorVersion());
		System.out.println("\nList of All Tables");
		String t[] = {"TABLE"};
		ResultSet rs = ob.getTables(null, null, null, t);
		while(rs.next())
		{
			System.out.println(rs.getString("TABLE_NAME"));
		}
		con.close();
	}
}
/* OUTPUT
C:\Program Files\Java\jdk1.5.0\bin>javac DatabaseMetaDataDemo.java
C:\Program Files\Java\jdk1.5.0\bin>java DatabaseMetaDataDemo

Database Name = ACCESS
Database Version = 12.00.0000
Database Driver Name = JDBC-ODBC Bridge (ACEODBC.DLL)
Database URL = Jdbc:Odbc:DB
Database Major Version = 2
Database Manor Version = 1

List of All Tables
Book
Employee
Student
*/
```

ResultSetMetaData

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

```
ResultSetMetaData rsmd=rs.getMetaData();
```

Methods	Description
String getTableName(int column)	Gets the designated column's table name.
int getColumnCount()	Returns the number of columns in this ResultSet object.
String getColumnLabel(int column)	Gets the designated column's suggested title for use in printouts and displays.
String getColumnName(int column)	Get the designated column's name.
int getColumnDisplaySize(int column)	Indicates the designated column's normal maximum width in characters.
int getColumnType(int column)	Retrieves the designated column's SQL type.
String getColumnTypeName(int column)	Retrieves the designated column's database-specific type name.
int getPrecision(int column)	Get the designated column's specified column size.
boolean isAutoIncrement(int column)	Indicates whether the designated column is automatically numbered.
boolean isCurrency(int column)	Indicates whether the designated column is a cash value.
boolean isReadOnly(int column)	Indicates whether the designated column is definitely not writable.
boolean isWritable(int column)	Indicates whether it is possible for a write on the designated column to succeed.
boolean isSearchable(int column)	Indicates whether the designated column can be used in a where clause.
boolean isSigned(int column)	Indicates whether values in the designated column are signed numbers.
int isNullable(int column)	Indicates the nullability of values in the designated column.

ResultSetMetaData Program

```
import java.io.*;
import java.sql.*;
class ResultSetMetaDataDemo
{
	public static void main(String args[])throws Exception
	{
		Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
		Connection con = DriverManager.getConnection("Jdbc:Odbc:DB");
		Statement st = con.createStatement();
		ResultSet rs = st.executeQuery("select * from student");
		ResultSetMetaData ob = rs.getMetaData();
		int n=ob.getColumnCount();
		System.out.println("Total Column Count = "+n);
		for(int i=1 ; i<=n ; i++)
		{
			System.out.println("\nCulumn Number = "+i);
			System.out.println("Column Name = "+ob.getColumnName(i));
			System.out.println("Column Data Type = "+ob.getColumnTypeName(i));
			System.out.println("Column Size = "+ob.getColumnDisplaySize(i));
			System.out.println("Column Precision = "+ob.getPrecision(i));
			System.out.println("Is Currency = "+ob.isCurrency(i));
			System.out.println("Is Read Only = "+ob.isReadOnly(i));
			System.out.println("Is Writable = "+ob.isWritable(i));
			System.out.println("Is Searchable = "+ob.isSearchable(i));
			System.out.println("Is Signed = "+ob.isSigned(i));
		}
		con.close();
	}
}
/* OUTPUT
C:\Program Files\Java\jdk1.5.0\bin>javac ResultSetMetaDataDemo.java
C:\Program Files\Java\jdk1.5.0\bin>java ResultSetMetaDataDemo
Total Column Count = 3
```

Column Number = 1 Column Name = sno Column Data Type = INTEGER Column Size = 11 Column Precision = 10 Is Currency = false Is Read Only = false Is Writable = false Is Searchable = true Is Signed = true	Column Number = 2 Column Name = sname Column Data Type = VARCHAR Column Size = 255 Column Precision = 255 Is Currency = false Is Read Only = false Is Writable = false Is Searchable = true Is Signed = false	Column Number = 3 Column Name = smark Column Data Type = INTEGER Column Size = 11 Column Precision = 10 Is Currency = false Is Read Only = false Is Writable = false Is Searchable = true Is Signed = true