Web Page designing

Frames

- division of window either horizontally or verdically.
- enables users to divide page into number of rectangular regions or windows of various sizes
- A page can have more than one frames.
- Frames look similar to tables visually but they are much powerful.
- A frame divides the page ~~as that~~ each of the frames is a website in itself, i.e; they can access different URLs and each of these can be updated independently.
- Frames could be applied for following on the web page:
  ① display the log or a stationary information in one fixed portion of the page.
  ② For table of contents in a page where people can just click and move around the website without having to move constantly to the contents page.

Definition of Frameset

- FRAMESET is the HTML tag used to create a framed page. This tag replaces body tag completely to create frame style pages.

```
<HTML>
  <HEAD>   <TITLE> ... </TITLE> </HEAD>
  <FRAMESET  ROWS = ——  OR  COLS = —>
                rows are rows & cols are columns like 3,5.
  </FRAMESET>
</HTML>
```

- The frame tag can have 2 attributes 'rows' & 'cols'.

  Rows: specify no. of horizontal windows or frames & value in the rows attribute specify of height of frames in frameset.

  Columns: specify no. of vertical windows or frame and value in the cols attribute specify width of the frames in frameset.

The frame rows & cols attributes can have 3 units: values in

① in pixels
② in % of ~~sou~~ parent frame
③ As *

① in pixels

```
<FRAMESET COLS = 200, 400>
    ....
</FRAMESET>
```

⇒ Frame (parent) window will have 2 frames where 1st frame's width is 200 px & 2nd frame's width is 400 px.

② as percentage of parent frame

```
<FRAMESET ROWS = 20%, 80%>
    ....
</FRAMESET>
```

⇒ Frame (parent) window will have ~~2 for~~ 2 frames & 2 rows. Height of first frame is 20% of the parent frame & 2nd frame's height is 20% of (main) parent frame.

③ As * :

```
<FRAMESET ROWS = 100, 40% *>
    ....
</FRAMESET>
```

Here, frame (main) window will have 3 ~~colum~~ rows (frames):

1st one's height = 100px
2nd one's height = 40%
3rd one's height = rest

<FRAME> Tag

- defined within frameset container.
- used to specify what actually appears within a particular frame.

Syntax:

```
<FRAMESET ROWS/COLS = ...., ....>
    <FRAME SRC = "VALUE"
           NAME = "VALUE"
           SCROLLING = "VALUE"
           MARGINWIDTH = "VALUE"
```

MARGINHEIGHT = "Value">

</IFRAME>

</FRAMESET>

SRC → determines name of document page that is to be ~~used~~ displayed in the frame by Window.URL.

Name → gives name of window (frame). Value can be string.

~~Title Bar~~

~~Blank~~

Scrolling → specify if window (frame) can have scrollbars or not, by setting the value as Yes | No | Auto.

Margin Width → sets the left & right margins in frame in pixels.

margin height → set top & bottom margin in the frame

The code for display A1.html & A2.html in 2 frames on a html framepage:

```
<Html>
  <head> <title> Frame Doc </title> </head>
  <Frameset  Rows = 50%, 50%>
    <Frame> src = "A1.html" name = "firstframe">
    <Frame> src = "A2.html" name = "second frame">
  </Frameset>
</Html>
```

A1.html

```
<Html>
  <head> <title>
  <body bgcolor = purple>
    Frame 1
  </body>
</Html>
```

A2.html

```
<Html>
  <body bgcolor = yellow>
    Frame 2
  </body>
</Html>
```

Nested Frames

↳ Frames within another frame.

frameset works only with either rows or cols attributes.

To have both attributes together, Frameset tag has to be nested.

* <Frameset rows=..>
  <Frame src="--"></Frame> → one row
  <Frameset cols=--->
    <Frame src="--"> → 2 columns
    <Frame src="--">
  </Frame></Frame>
</Frameset>

eg

```
<html>
  <head><title> Nested frames </title></head>
  <frameset rows=200, 500, *>
    <frame src="A1.html" name="FFrame">
    <frame src="A2.html" name="SFrame">
    <Frameset cols = 40%, 50%, *>
      <frame src="index.html" name="FCFrame">
      <frame src="welcome.html" name="SCFrame">
    </Frame></Frame>
  </Frameset>
</Frameset>
```

Creating Linked Frames

↳ Creating hyperlink in one frame to another or from some other website.

eg to link A1.html to netscape.com's homepage:

```
<html>
  <body bgcolor = purple>
    Frame 1
    <a href=http://www.netscape.com> Netscape HP </a>
    MARGINWIDTH= value
```

</body>
</html>

XML

- acronym for Extensible Markup language.
- derived from SGML
- markup language. → case sensitive
- XML document is written using markup language to contain structured info.
  → markup makes it easy to identify structures within a document.
- Structure content in an XML document may contain words, pictures and a description of the relationship of the content within.

XML is different from Html.

- In Html, tag semantics and tag sets are fixed & rigidly defined.

- XML is a way of marking up data; adding metadata & separating structure from formatting & style.
- With XML info about documents & pieces of documents can be stored.

XML schema

- XML ~~schema~~ based alternative to DTD
- describes structure of XML document
- XML schema language is also referred as XML schema Definition (XSD)
- XML schema defines:
  1. elements that can appear in a document
  2. attributes that can appear in a document
  3. which elements are child elements.
  4. order & no. of child elements.
  5. whether an element is empty or can include text
  6. datatypes for elements & attributes
  7. default and fixed values for elements & attributes.

XML over DTDs:

Because XML schemas are

1. extensible ~~for~~ to future additions
2. richer & more powerful than DTDs
3. written in XML
4. support datatype
5. support namespaces
6. ~~xxx~~

XML schemas secure data ~~xxx~~ communication

When sending data from a sender to receiver, it is essential that both have same expectation about the content.

With XML schemas, sender can describe the data in a way that receiver will understand

eg: 03-8-2009 may be considered as 3rd August in one country & 8th March in other

but XML element with datatype like this:

<date type="date">2009-03-08</date>

ensures mutual understanding of the content, data datatype requires format of "YYYY-MM-DD"

XML schemas are extensible

because written in XML.

With an extensible schema, one can:

1. Reuse schema into other schemas
2. create own Datatypes derived from standard datatypes.
3. Reference multiple schemas in same document

Simple XML document

note.xml

```
<?xml version="1.0"?>
<note>
    <to> Tove </to>
    <from> Jani </from>
    <heading> Reminder </heading>
    <body> Don't sleep </body>
</note>
```

DTD file

note.dtd

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

4 children → to.
from
heading
body.

XML Schema

note.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="URL/XMLSchema"
           targetNamespace="URL"
           xmlns="URL"
           elementFormDefault="qualified">

<xs:element name="note">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="to" type="xs:string"/>
            <xs:element name="from" type="xs:string"/>
            <xs:element name="heading" type="xs:string"/>
```

```
<xs: element name = "body type = "xs: string" />
</xs: sequence>
</xs: complexType>
</xs : element>
</xs:schema>
```

It is complex type as it contains other elements, to, from, heading, body while to, from, heading & body are simple types, as they do not contain other elements.

Reference to a DTD from XML document

```
<? xml version = "1.0" ?>
<! DOCTYPE note SYSTEM "URL/note.dtd">
<note>
    <to> </to>
    <from> </from>
    <heading> </heading>
    <body> </body>
</note>
```

Reference to XML schema from XML document

```
<?xml version "1.0" ?>
<note xmlns = "URL"
      xmlns : xsi = "URL"
      xsi : schema Location = "URL note.xsd">
    <to> </to>
    <from> </from>
    <heading> </heading>
    <body> </body>
</note>
```

Document Type definitions

- describes a model of structure of content of XML document.
- This model says, what must be present, which one are optional, what are their attribute are & how they can be structured with relation to each-other.
- defines / declares only the structure of tags.
- it is an XML description of content model of a type of documents.
- is a statement in an XML file that identifies the DTD that belongs to the document.

syntax: `<!DOCTYPE DTD.name [internal subset]>`

↑ content of internal DTD subset; part of DTD that stays in XML document itself.

Developing DTD from XML code

xmldoc:

```
<? xml version = "1.0" ?>
<page>                                  → root
    <head> <title> HP </title> </head>  → element1, → subelement
    <body>                              → element2
        welcome
        <para> Hey </para>              → subelement1
    </body>
</page>
```

DTD:

```
<! DOCTYPE page [                       → root
    <! ELEMENT page (head, body)>
    <! ELEMENT head (title)>
    <! ELEMENT body (para)>
```

against grammatical rules of apt - XML language.

Two types of DTDs

Internal DTD

syntax:

<!DOCTYPE element DTD-identifier
[
declaration 1
declaration 2
...
]

(root name → element; path to root → DTD-identifier)

⇒ <!DOCTYPE → delimiter

⇒ element tells parser to parse the document from the specified root element.

⇒ DTD identifier is an identifier for the DTD which may be the path to a file on system or URL to a file on the internet.
If DTD points to external path, called external subset

⇒ [ ] enclose an optional list of entity declarations called Internal subset.

① Internal DTDs

if elements are declared within the XML files.
To refer to an internal DTD, standalone attribute in XML declaration must be set to 'Yes'
→ declaration works independent of an external source.

External DTD

syntax:

<!DOCTYPE root-element [ele-declaration]>
↓
element of root

eg:

```
<?xml version="1.0"?>
<!DOCTYPE address [
  <!ELEMENT address (name, phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<address>
  <name>Aarti</name>
  <phone>09xxxxxx76</phone>
</address>
```

PCDATA means parser-able text-data.

② External DTDs

if elements are declared outside XML file.
standalone attribute = No
⇒ info from an external source.

syntax:
<!DOCTYPE rootelement SYSTEM "filename">

eg: P-7.0..

eg <?xml version="1.0" ?>

<!DOCTYPE addres SYSTEM "addres.dtd" >

<addres>

<name></name>

<phone></phone>

</addres>

in addres.dtd :

<!ELEMENT addres (name, phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT phone (#PCDATA)>

<

SAX

Simple API for XML is an event based parser for XML documents.

Unlike a DOM parser, SAX parser creates no parse trees

SAX is a streamy interface for XML, which means that applications using SAX receive event notifications about the XML document being processed. an element & attribute at a time in sequential order starting at the top of the document & ending with the closing of the root element.

- reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XML document.
- tokens are processed in the same order as they appear in document
- Application program provides an event handler that must be registered with the parser.
- As the tokens are identified, callback methods in the handler are invoked with the relevant information

Viewing XML in I.E (browser)

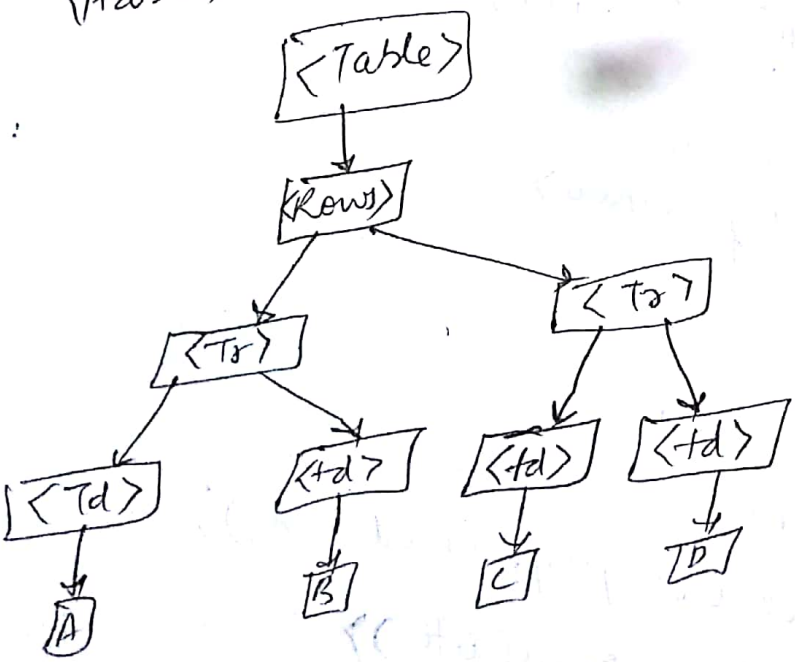
① Using object models

- Document Object Model (DOM) is a progg interface (API) for HTML & XML documents.
  - defines logical structure of documents & the way a document is accessed & manipulated.
  - makes tree structure view for an XML document.

Html document

```
<table
    <rows>
        <tr>
            <td> A </td>
            <td> B </td>
        </tr>
        <tr>
            <td> C </td>
            <td> D </td>
        </tr>
    </rows>
</table>
```

DOM :



DHTML

- Dynamic html
- totally different from HTML
- browsers which support DHTML are some of the version of Netscape Navigator & IE of version > 4.0.
  - DHTML is based on properties of HTML, JS, CSS & DOM (access individual elements of a document) which helps in making dynamic content.
- ~~combined~~ make use of Dynamic object model to make changes in setting & also in properties & methods.
- also makes use of scripting & it is ~~so~~ also part of earlier computing trends.
- allows different scripting languages in a webpage to change their variables, which enhance the effects, looks & many other functions after the whole page have been fully loaded or under a view process., or otherwise static HTML pages, on the same.

But there is no thing that as dynamic in DHTML, there is only the enclosing of different technologies like CSS, HTML, JS DOM & different sets of static languages which makes it as dynamic.

- used to create interative & animated webpages that are generated in real time., also known as dynamic webpages so that when such a page is ~~create~~ accessed, the code within the page is analyzed on the web server & the resulting HTML is sent to clients web browser.

HTML : client-side markup language
: structuring

CSS : styling & designing
: CSS rules ~~change~~ modify for DHTML at different levels by JS with event handlers which adds a significant amount of dynamism with very little code.

JS : client side scripting language

: have cookies to determine user needs.

DOM : act as weakest links in it

: only defect in it is that most of the browsers doesn't not support DOM.

: way to manipulate static content.

14.1. DATABASE PROGRAMMING USING JDBC

Features of JDBC API (UPTU 2006)

JDBC is an API specification developed by Sun Microsystems that defines a uniform interface for accessing various relational databases. JDBC is a core part of the Java platform and is included in the standard JDK distribution.

The primary function of the JDBC API is to provide a means for the developer to issue SQL statements and process the results in a consistent, database-independent manner. JDBC provides rich, object-oriented access to databases by defining classes and interfaces that represent objects such as:

1. Database connections
2. SQL statements
3. Result Set
4. Database metadata
5. Prepared statements
6. Binary Large Objects (BLOBs)
7. Character Large Objects (CLOBs)
8. Callable statements
9. Database drivers
10. Driver manager.

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. The location of the driver manager with respect to the JDBC drivers and the servlet is shown in Fig. 14.1.

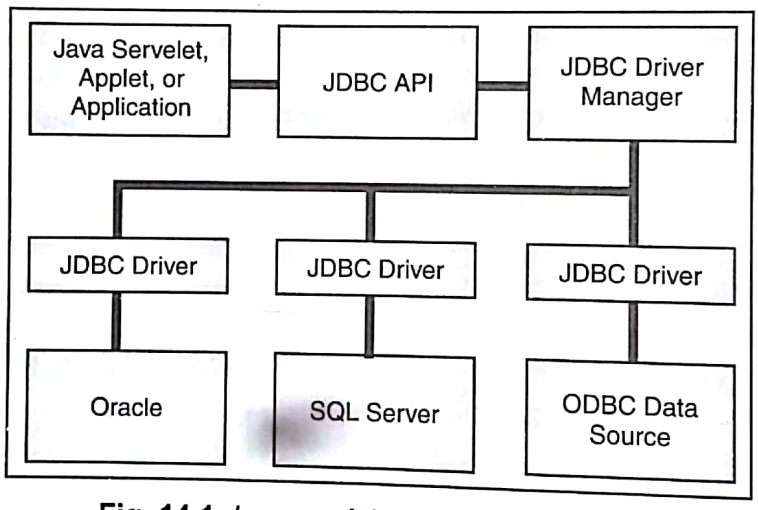


Fig. 14.1. *Layers of the JDBC Architecture*

A JDBC driver translates standard JDBC calls into a network or database protocol or into a database library API call that facilitates communication with the database. This translation layer provides JDBC applications with database independence. If the back-end database changes, only the JDBC driver need be replaced with few code modifications required. There are four distinct types of JDBC drivers.

JDBC Versions

JDBC 2.0 API is the latest version of JDBC API available in the java.sql package. The previous version focused primarily on basic database programming services such as creating connections, executing statements and prepared statements, running batch queries, etc. However, the current API supports batch updates, scrollable resultsets, transaction isolation, and the new SQL:1999 data types such as BLOB and CLOB in addition to the SQL2 data types.

JDBC 2.0 Optional Package API is available in the javax.sql package and is distributed with the enterprise edition of Java 2, that is J2EE. The optional package addresses Java Naming and Directory Interface (JNDI)-based data sources for managing connections, connection pooling, distributed transactions and rowsets.

So, you may be asking yourself, "Which is the right type of driver for your application?" Well, that depends on the requirements of your particular project. If you do not have the opportunity or inclination to install and configure software on each client, you can rule out Type 1 and Type 2 drivers.

However, if the cost of Type 3 or Type 4 drivers is prohibitive, Type 1 and type 2 drivers may become more attractive because they are usually available free of charge. Price aside, the debate will often boil down to whether to use Type 3 or Type 4 driver for a particular application. In this case, you may need to weigh the benefits of flexibility and interoperability against performance. Type 3 drivers offer your application the ability to transparently access different types of databases, while Type 4 drivers usually exhibit better performance and, like Type 1 and Type 2 drivers, may be available free of charge from the database manufacturer.

Java Database Connectivity (JDBC) provides a database-programming API for Java programs. Some of the features of JDBC API are as follows :

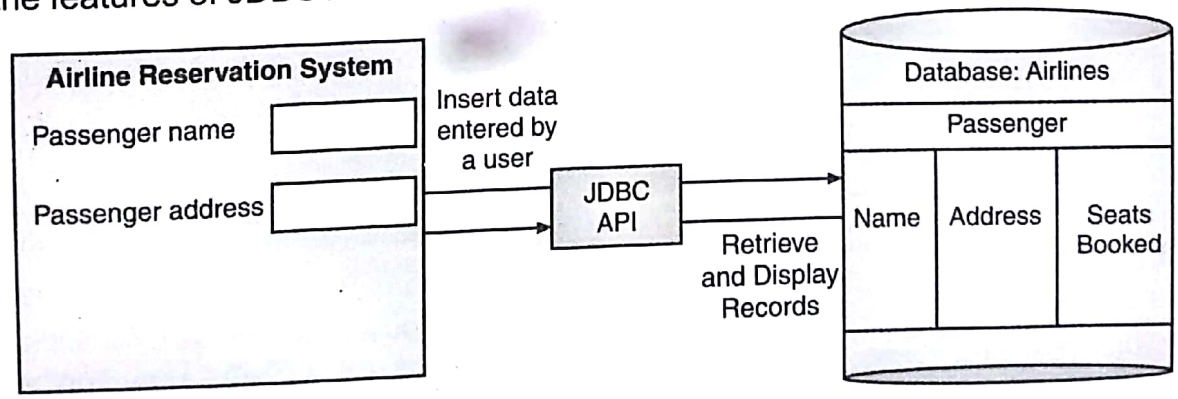


Fig. 14.2.

(a) Contains a set of classes and interfaces that are used to connect to a database built using any DBMS/RDBMS, submit SQL queries to a database and retrieve and process the results of SQL queries.

(b) Is a low-level interface in which SQL select and update statements are called directly from within Java programs.

The JDBC 3.0 API is the latest update of the JDBC API. It contains many features, including scrollable result sets and the SQL:1999 data types.

(c) Can be used with both two-tier and three-tier database architectures. In two-tier architecture, a Java program invokes the method of JDBC API, which in turn communicates with the database server. In three-tier architecture a Java applet or a HTML form submits SQL queries to a middle tier server. The middle tier server in turn uses JDBC API to communicate with the database server.

Points to Remember

1. High-level interfaces are a set of user-friendly and easily understandable APIs. In the case of high level interfaces, a Java program uses the classes and objects of the high-level interface.
2. The high-level interface in turn invokes the methods of the JDBC API. The JDBC API in turn communicates with the database. An example of high-level interface is Embedded SQL for Java.

14.2. JDBC DRIVERS

(UPTU 2005)

There are several types of JDBC drivers available. They are :

1. JDBC-ODBC bridge driver
2. Native API partly java driver
3. JDBC Net pure java driver
4. Native protocol pure java driver.

14.2.1 JDBC-ODBC Bridge Driver (Type 1 Driver)

These drivers are the bridge drivers such as JDBC-ODBC bridge. These drivers rely on an intermediary such as ODBC to transfer the SQL calls to the database. Bridge drivers often rely on native code, although the JDBC-ODBC library Native code is part of the Java-2 virtual machine.

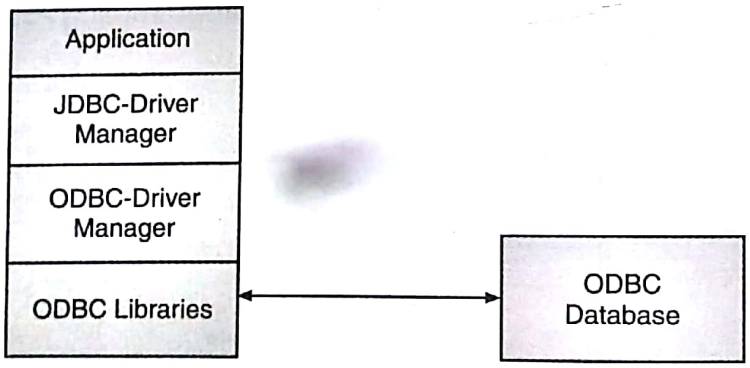


Fig. 14.3. *JDBC-ODBC Bridge Driver*

14.2.2 Native API Partly Java Driver (Type 2 Drivers)

A native API is partly a java driver, uses native C language library calls to translate JDBC to native client library. These drivers are available for oracle, Sybase, DB2 and other client library based RDBMS. Although Type 2 drivers are faster than Type 1 drivers, Type 2 drivers are faster than Type 1 drivers, Type 2 drivers use native code and require additional permissions to work in an applet. A type 2 driver might need client side data base code to connect over the network.

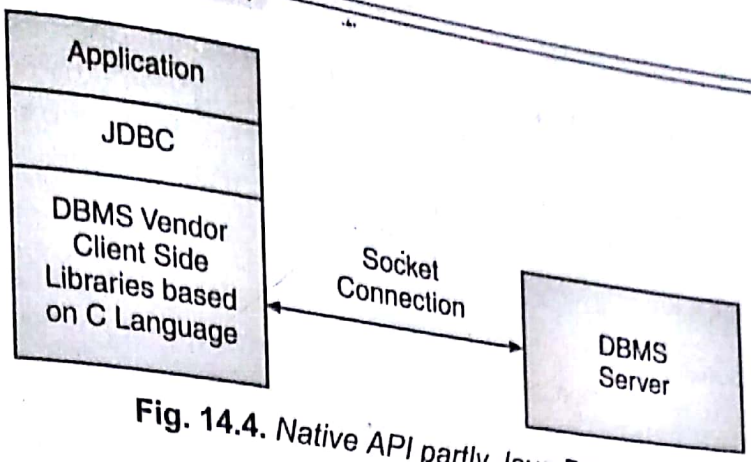


Fig. 14.4. *Native API partly Java Drivers*

14.2.3 JDBC - Net Pure Java Driver (Type 3 Drivers)

JDBC - Net pure java driver consists of JDBC and DBMS independent protocol driver. Here the calls are translated and sent to middle tier server through the socket. The middle tier contact the databases. It is simple and most flexible. Type 3 drivers call the database API on the server. The JDBC requests from the client are first provided to the JDBC Driver on the server to run. Type 3 and 4 drivers can be used by thin (Java based) clients as they need no native code.

14.2.4 Native Protocol Pure Java Driver (Type 4 Drivers)

A native protocol java driver contain JDBC calls that are converted directly to the network protocol used by the DBMS server. The highest level of driver implements the database network API in the Java language. Type 4 drivers can also be used on thin clients as they also have no native code. The network protocol is defined by the vendor and is typically proprietary, the driver usually comes only from the database vendor.

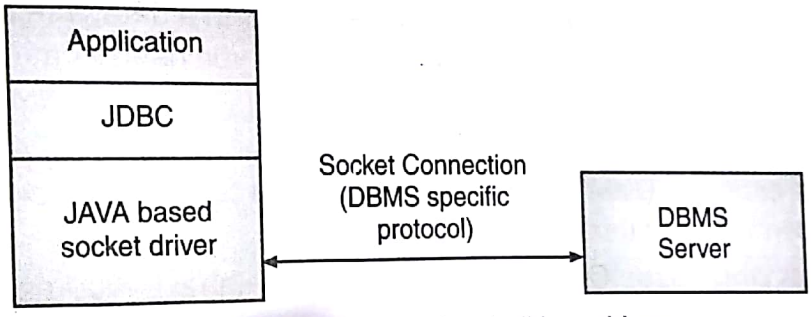


Fig. 14.5. *Native protocol all java driver.*

14.3. JDBC APPLICATION ARCHITECTURE

JDBC application architecture is shown in Fig. 14.6. In this architecture there are five modules, here module1 is the java application program or java applet. As we know that the java applet is the small component program. Module 2 is the Java Database Connectivity (JDBC) driver manager, which handles the request from module1 and takes the help of Java Database Connectivity (JDBC) bridge driver, which removes the function gap between the module 2 and module 4 is the driver module related to particular database, either specific to MS-Access driver or specific to MS-SQL database, which are decided according to module 5 *i.e.*, if data base used MSAccess then the Access driver comes into picture, or if database is in MS-SQL then ms-SQL driver comes into picture.

There are different sequences of steps required to access the database which are described in section 14.4.

Use of the JDBC-ODBC bridge from an untrusted applet running in a browser, such as Netscape Navigator, isn't allowed. The JDBC-ODBC bridge doesn't allow untrusted code to call it for security reasons.

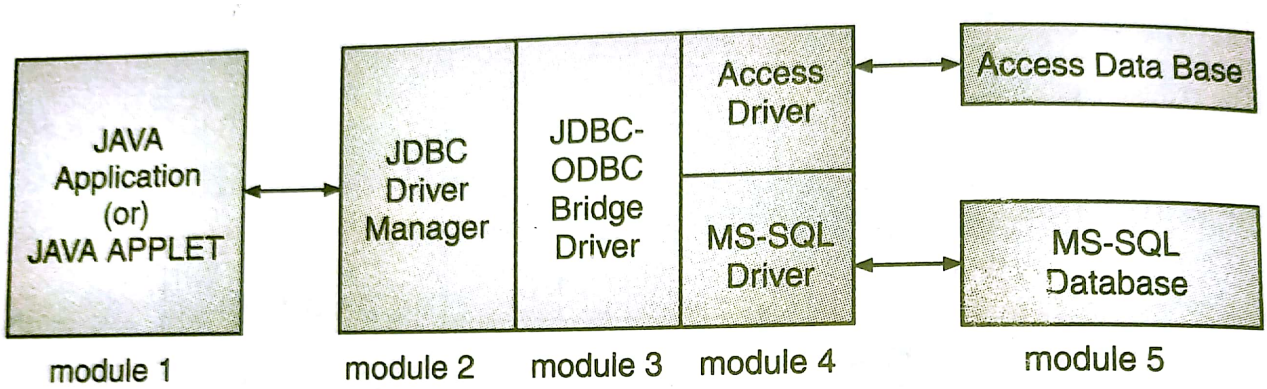


Fig. 14.6. *JDBC application architecture.*