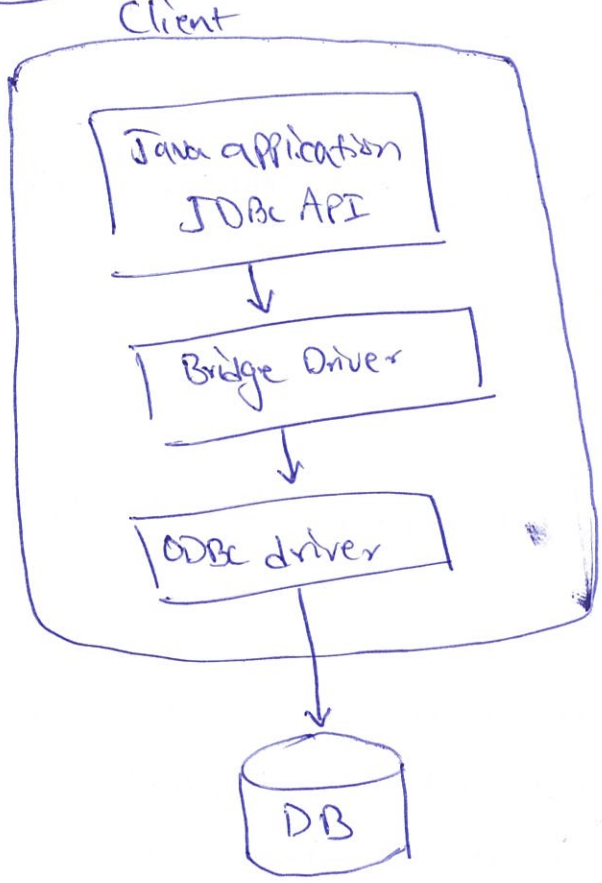
30/8/2014

Java DataBase Integration with JDBC.

→ There a different type of JDBC Drivers. They are.

Type 1: JDBC-ODBC bridge (ODBC → Open Database Connectivity protocol)



→ The communication to a DB is done through bridge and ODBC drivers.

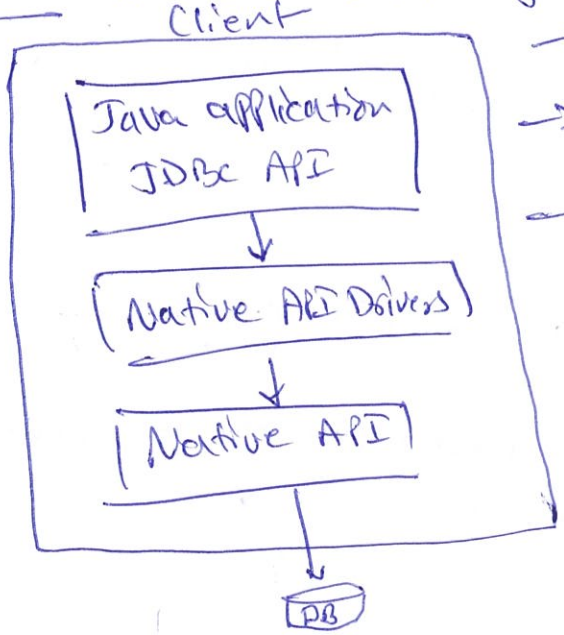
→ The Pros of this is that it (JDBC) can talk to any type of DB.

→ Cons not portable and not 100% Java.

→ Drivers must be on the same Comp. as application

→ the ODBC drivers must be matched the DB version.

Type 2: Native API / Partly Java.

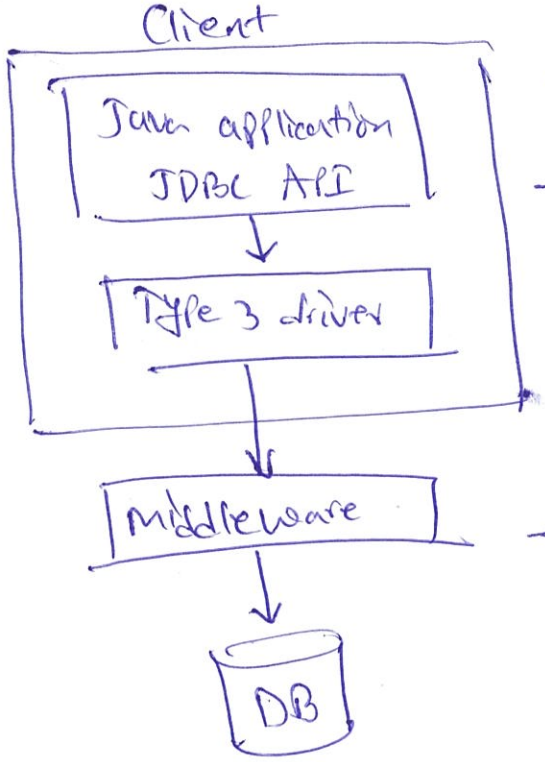


→ Comm. Through OS specific API

→ Pros: Better performance then JDBC/ODBC

→ Cons: same as JDBC/ODBC.

Type 3: an Java/network Protocol driver



→ Communication through network and middle-ware server

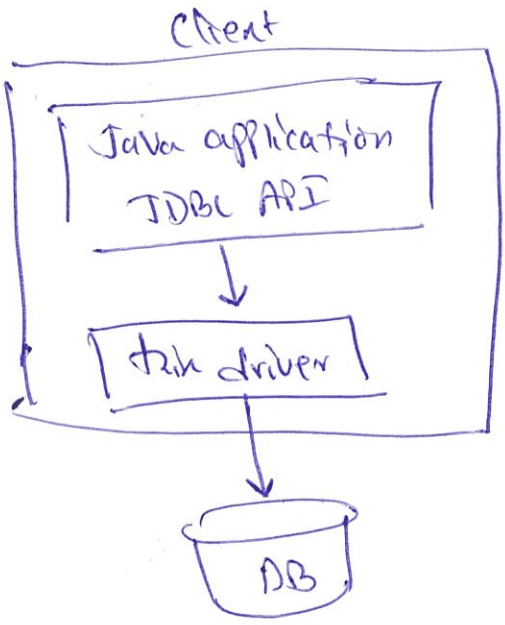
→ Pros:

- driver is server based and vendor specific libraries not required on the client.
- The Client driver can be very small.

→ cons.

- another server to install and maintain.

Type 4: 100% Java thin driver



→ comm. directly from Application to DB.

→ Pros:

- the driver is 100% Java no additional layers to install or maintain.
- the driver is installed with the app. So maintenance is simplified.

→ Cons:

- You need a different driver for each database s/w.

Types of DB App used in this course:

→ I will be using two DBMS Applications in this course

- mysql:
  - → which is a web based (or locally) installed RDBMS server.

~~HyperSQL~~

- HyperSQL (HSQLDB)

→ is a 100% Java in-memory ~~based~~ DB used with local file, with server running with same Java process as App.

Connecting to mysql DB:

→ Please see "up and Running with Java application" to setup and import an sql file.

→ The sql file you need to import in this is in "DB" folder

↓

"LIB"

↓

"mysql-connector-java-5.1.32-bin.jar"

→ Create a new Java Project in Eclipse. "ConnectMyDB" copy this file from the folder.

→ now go to eclipse.

right Click on the created Java project "ConnectMyDB"

new → folder → give it a name "Lib".

→ right Click on "lib" folder and click on "paste". You should now see the "Jar" file copied to "lib" folder.

→ now Right Click on the copied "jar" file

Build Path → Add to build path ↵

which will allow you to uses the methods from that library.

→ See "1_ConnectMySqlDB."

Connecting to hsqldb

→ follow the same procedure for copying the HSQLDB Driver from

DB → Lib → ~~hsqldb.jar~~ HSQLDB → hsqldb.jar

→ You don't need a SQL server for this rather HSQLDB uses a text file to access and store DB. It uses script and properties file.

→ You can find it at (DB files)

DB → Lib → HSQLDB → exploreCalifornia.properties, exploreCalifornia.script. } copy them

→ Now same as creating a "lib" folder in Eclipse create a new folder "data" and now copy those two file and paste them in this "data" folder

→ There are few changes that needs to be done here; open the "exploreCalifornia.properties" by double clicking on it in Eclipse.

→ This should open a new text file. Search for

CREAT USER SA PASSWORD " "

↳ System Administrator

→ Copy the above statement and lets make it sure it coincides with our main.java class. and paste it below it. edit it to.

CREAT USER "root" PASSWORD "root"

→ Now we have to grant the permissions to it. Search for

GRANT DBA TO SA.

↳ Database administration

->copy this and paste it in the next line and edit it into.

GRANT DBA TO "root"

->save the file.

->in the "main.java" you will have to change only one thing.

in CONN_STRING change it to.

"jdbc:hsqldb:data/explorecalifornia.".

↳ the folder you have created

-> see "2_ConnectMyHSQLDB."

Static SQL Statement

-> see "3_StaticSQL"

->I have include two java files in this eg.

HsqldbMain.java

MysqlMain.java.

-> Both contain a method of count how many number of rows available in a particular table, here I have taken "states" table into consideration

-> Note: you must understand that the count in DB start with "1" where as in Arrays it starts with "0"

- Also MySQL data is a scrollable data, that means we can point to any where and retrieve the data where as HSQLDB is not, if the point starts from top it must go down and cannot come up.

Connecting to multiple database:

→ See "4_MultipleConnection."

→ This example shows how you can connect to different DBMS with a single file using Switch and Enumeration.

Exception handeling in JDBC:

→ when ever an error occurs in JDBC it throws an "SQLException" Class.

→ SQLException is an Extension of Exception class. and inherits all the tools that are part of Exception class such as

- getMessage
- getLocalizedMessage
- getStackTrace

etc...

→ we have in this example use 3 such exception methods.

- getMessage ⇒ gets the message as to what went wrong.
- getErrorCode ⇒ gets the error code of what went wrong.
- getSQLState ⇒ gets the state code of the error.

→ Note: all these error depend what type of DBMS you are using.

→ See "5_JDBCErrorHandeling"

→ All the information for the error can be obtained from MySQL and HSQLD web site.

Closing an Connection in Java 7 and above

→ Prior to Java 6 and below, all the connections has to be closed manually using finally block in that method and the using close() method.

→ but in Java 7 and above; there has been a new technique in which all the statements such as (methods)

Connection
Statement
ResultSet
} (1)

has AutoCloseable interface, which does not need to be closed manually, to make sure this works we have to use this in some thing called as try(with){

```
---
} catch (SQLException e){
---- ↳ This exception is only for SQL.
}
```

→ Instead of having "null" to all the methods; these can be initialized inside the "with" method i.e.

```
try (
    Connection conn = ------;
    Statement stmt = conn. ------;
    ResultSet rs = stmt. ------;
){
---
} catch (SQLException e){---}
```

→ See "6_Java7Closing".

Rows looping:

→ We use two methods to get an output from the DB table

- While loop
- ResultSet

→ The condition goes like this

```
While (ResultSet.next()) {

        Execute the code---

}
```

→ The method next(); says that if there is a next row which is not empty give the output which is stored in resultSet.

→ see Tours.Java class in "7_RowsLooping".

31/8/14

Scrollable Result Set:

→ by default ResultSet is not Scrollable; that means you cannot move or fetch data from where ever you want but can looping from top to bottom.

→ You can always implement methods to override Resultsets orginal features.

→ following are few methods where you can choose your cursor position.

- methods that move the cursor to specific row:

  * rs.beforeFirst();
  * rs.first();
  * rs.last();
  * rs.afterLast();
  * rs.absolute(int row);

- boolean methods that check the cursor position.

  * rs.isBeforeFist();
  * rs.isFist();
  * rs.isLast();
  * rs.isBeforeLast();

  } gives out boolean i.e. (true/false)

→ See "8_MovingCursorInDB".

Limiting the rows for each table.

→ There are two ways of limiting rows ~~ones~~

- Pure JDBC way.
- SQL way.

→ JDBC is a complex Java SQL code we can limit the output of certain DB in non-arbitrary way by using this method.

setMaxRows(int);

- The Problem with this is that if there are 500 rows in a DB table all these rows are called; but while execution this method only displays

the given input. This will take over the bandwidth So to overcome this we use SQL statements. "LIMIT int, int"

→ "LIMIT" key word would limit the no. of output with the given input eg:

"LIMIT 10, 5"

→ This will leave the first 10 rows and display the next 5 rows leaving anything after that.

Using Prepared statements to give Parameters to SQL Statements

→ what if we need to give a condition to an squal from the user So that he/she could see what they want.

→ "PreparedStatement" is an class in Java.sql.* class where you can set value to an Sql statement using get___(_,_) method.

eg: Doble maxPrice;

PreparedStatement stmt = Conn.PreparedStatement(" SELECT Price FROM tours WHERE price <= ? ")

Then ② ↓ 10.

stmt.setDouble(1, maxPrice);

① If ↓ 10

→ See "10_PreparedStatements"

1/9/2014

Stored Procedures:

→ A stored procedure is like a function but in the data base; a typical function in Java is like

```
add (int a, int b) {
    c=a+b;
    return c;
}
```

→ In the above example we do ~~the same~~ given in two integer value by calling its function like

```
add(1,2);
```

→ This will return '3', like wise there are methods in Data bases where you can do this function type rather than giving a big string.

→ These function can then be called via JDBC connection.

→ see "1_StoredProcedures"

→ You will be using the following syntax.

for sql syntax in Java.

SQL = "{call ____(?)}"

and for the method (callableStatement stmt = Conn.prepared(all(SQL);

→ If there are more than two out puts / inputs with different data types see "12_Stored Procedures Out"

Generic Getters in Java 7 and above

→ Instead of explicitly giving the type of get value that is

getInt, getString

Java 7 has a new feature called getObject

Syntax

getObject(_____ , _____ .class);

↑ Variable of the table name

↑ Data type

→ (see "13_GenericGettersJava7"

Managing Data bases using Java Bean.

Setting up Java bean:

→ We use Java bean to update and modify a data in DB.

→ I have created a project called "14_settingUpJavaBean" in that under src → com.gollahalli.main.bean → Admin.javaBean ↵

↑ have a look at it.

→ A Java bean usese setter and getter methods, which can take in and give out data when asked.

→ This exercise does not run but have a look at it.

Retrieving a Java bean:

-> see "15_RetrieveBean".

Inserting SQL

-> see "16_InsertSQL".

Update SQL

-> see "17_UpdateSQL"

Delete SQL

-> see "18_DeleteSQL"

Updatable ResultSet:

->This is an another version to update an entry in SQL DB.

-> see "19_UpdatableResultSet".

Transactions:

-> A Transaction is an option type of how to manipulate your DB read/write option.

Metadata:

-> A metadata class can be used If you want to ~~to~~ know the structure of the tables.

→ see "22_MetaData" & "23_ColumnsAndTables".