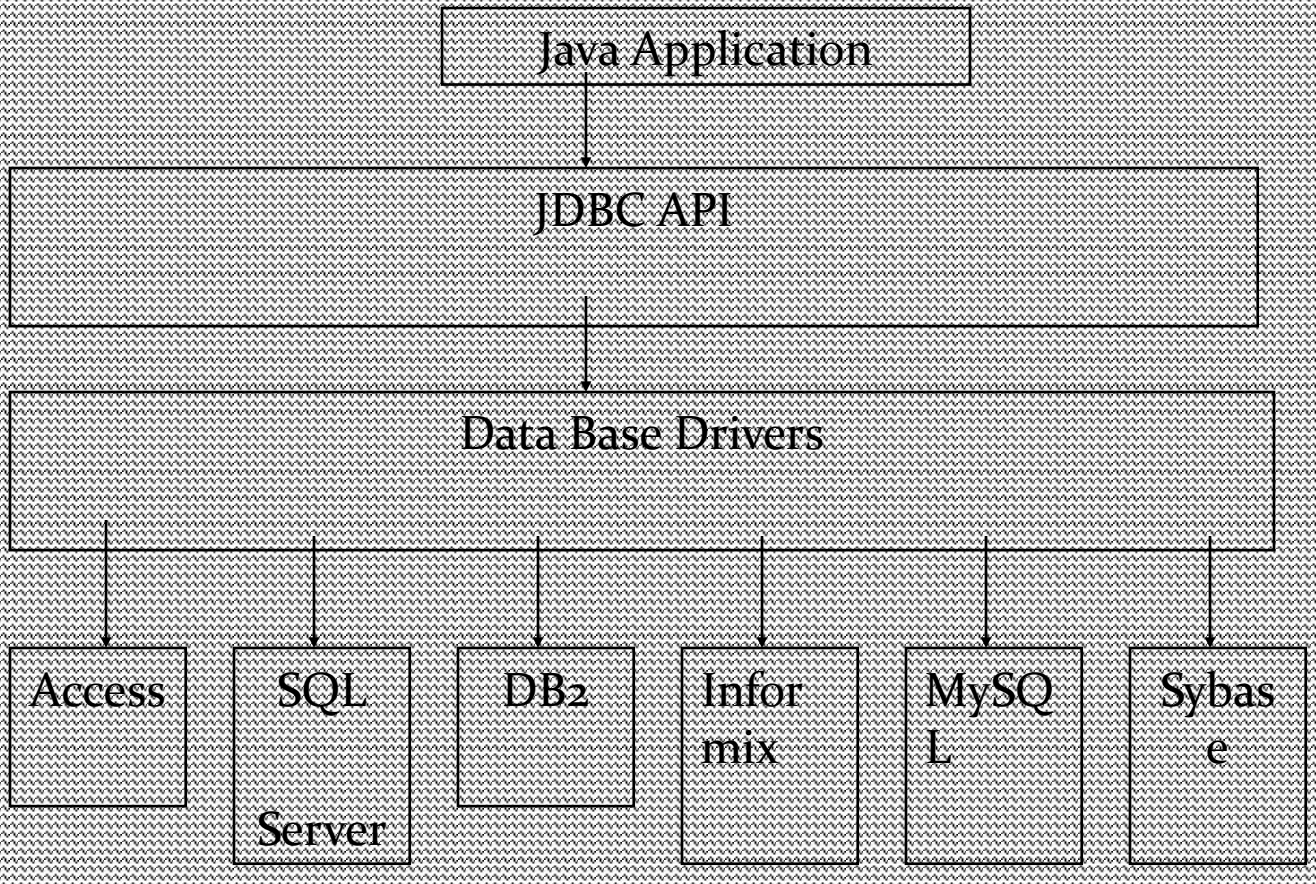
JDBC

JDBC

- JDBC is a Sun trademark
  - It is often taken to stand for Java Database Connectivity
- Java is very standardized, but there are many versions of SQL
- JDBC is a means of accessing SQL databases from Java
  - JDBC is a standardized API for use by Java programs
  - JDBC is also a specification for how third-party vendors should write database drivers to access specific SQL versions

JDBC Architecture



Database Drivers

- Think of a database as just another device connected to your computer
- like other devices it has a driver program to relieves you of having to do low level programming to use the database
- the driver provides you with a high level api to the database

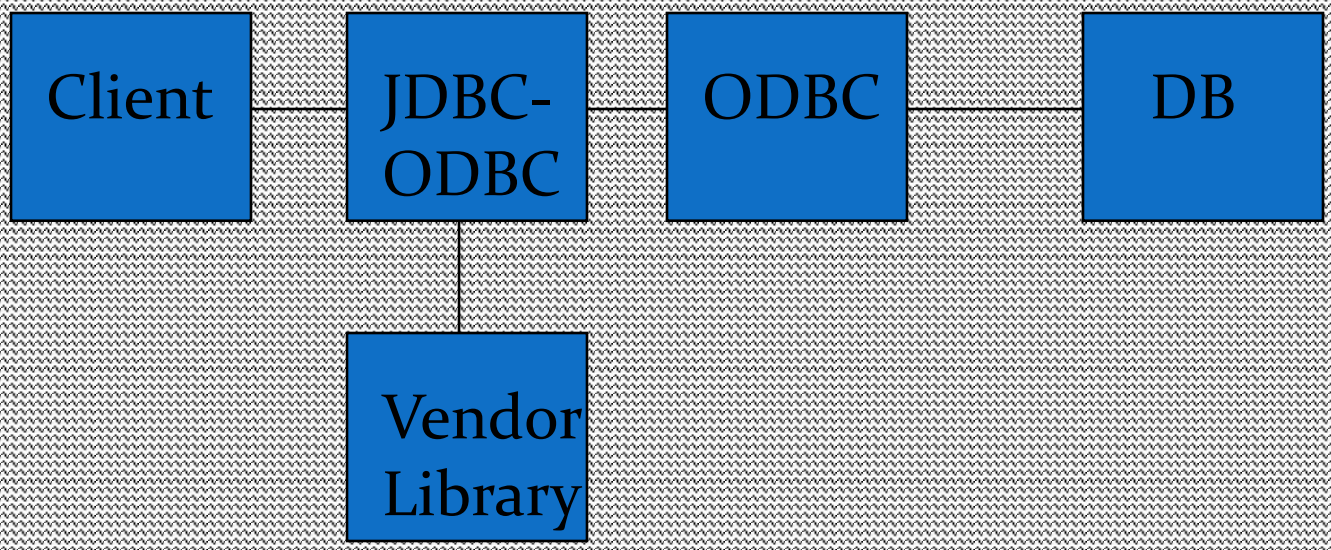
JDBC Driver Types

- Type 1
  - JDBC-ODBC Bridge
- Type 2
  - Native API, partially java
- Type 3
  - JDBC Network Driver, partially java
- Type 4
  - 100% Java

Type 1 Drivers

- Translate JDBC into ODBC and use Windows ODBC built in drivers
- ODBC must be set up on every client
  - driver must be physically on each machine for both java applications and applets
  - for server side servlets ODBC must be set up on web server
- driver sun.jdbc.odbc.JdbcOdbc provided by JavaSoft with JDK

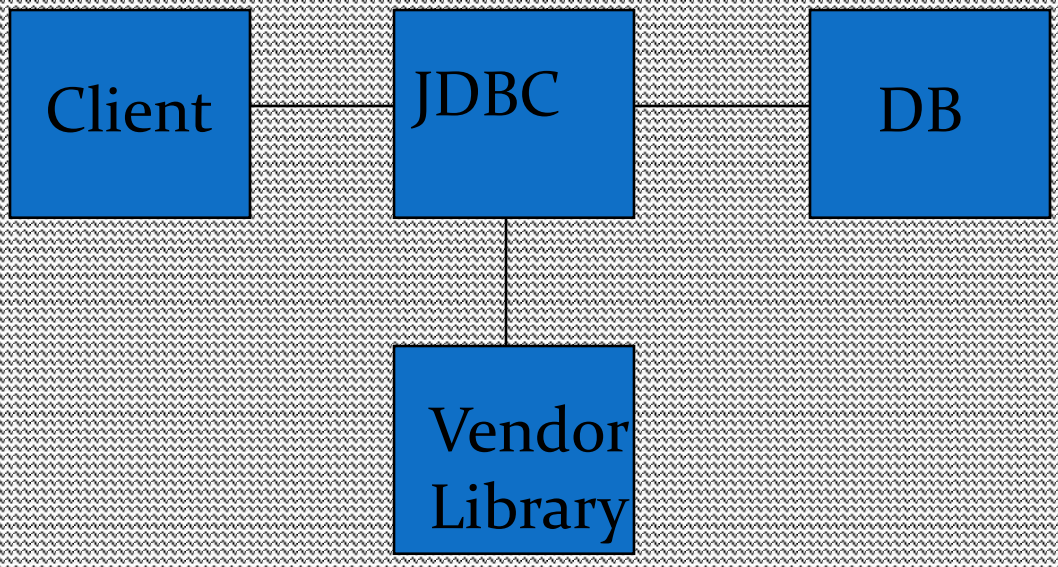
Type 1 Driver (cont.)



Type 2 Drivers

- Converts JDBC to data base vendors native SQL calls
- like Type 1 drivers; requires installation of binaries on each client

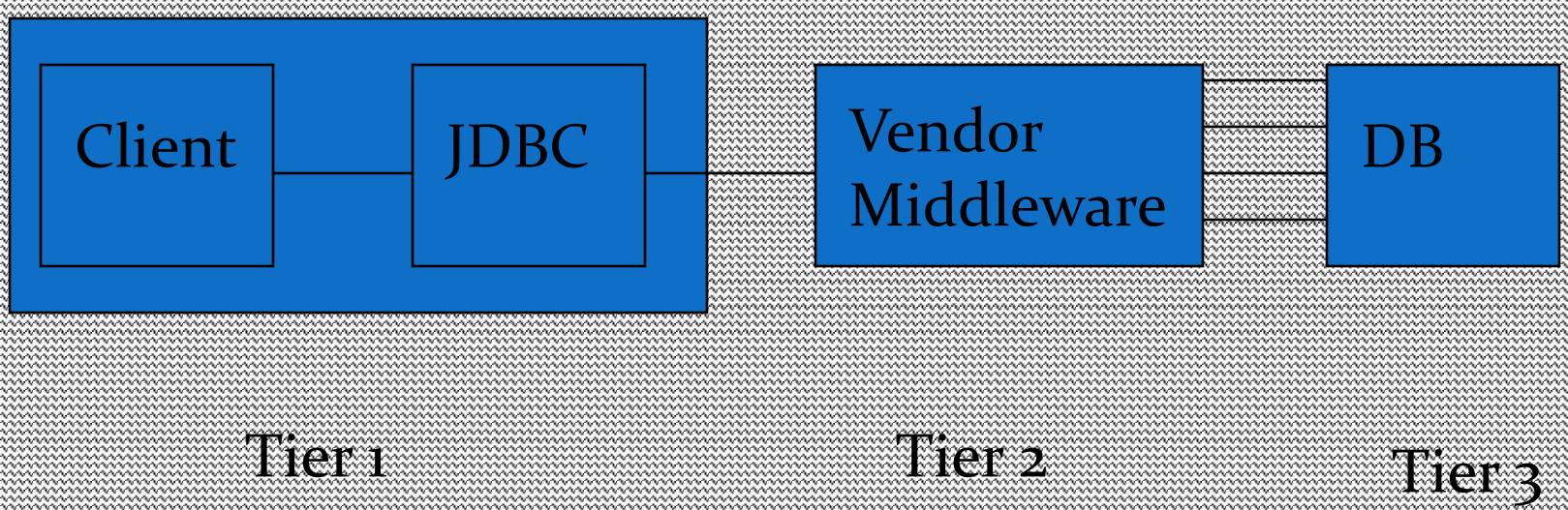
Type 2 Drivers (cont.)



Type 3 Drivers

- Translates JDBC to a DBMS independent network protocol
- Typically talks directly with a middleware product which in turn talks to the RDBMS
  - Jaguar, DBAnywhere, SequeLink
- Most flexible driver type
- all java

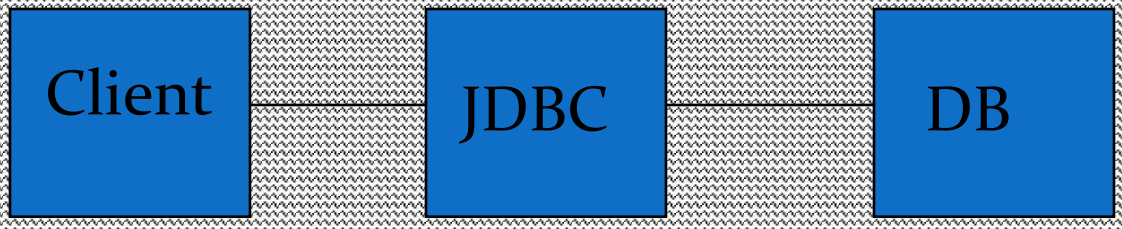
Type 3 Drivers (cont.)



Type 4 Drivers

- Converts JDBC directly to native API used by the RDBMS
- compiles into the application , applet or servlet; doesn't require anything to be installed on client machine, except JVM
- handiest driver type

Type 4 Drivers (cont.)



JDBC APIs

java.sql

- JDBC is implemented via classes in the java.sql package

Loading the Driver -DriverManager

- DriverManager tries all the drivers
- Uses the first one that works
- When a driver class is first loaded, it registers itself with the DriverManager
- Therefore, to register a driver, just load it!

Registering a Driver

- statically load driver

```
Class.forName("foo.bar.MyDriver");
Connection c=
  DriverManager.getConnection(Database
  URL);
```

- or use the `jdbc.drivers` system property

JDBC Object Classes

- DriverManager
  - Loads, chooses drivers
- Driver
  - connects to actual database
- Connection
  - a series of SQL statements to and from the DB
- Statement
  - a single SQL statement
- ResultSet
  - the records returned from a Statement

JDBC URLs

`jdbc:`*`subprotocol`*`:`*`source`*

- each driver has its own subprotocol
- each subprotocol has its own syntax for the source

`jdbc:odbc:`*`DataSource`*

- e.g. `jdbc:odbc:Northwind`

`jdbc:msql://`*`host[:port]/database`*

- e.g.
  `jdbc:msql://foo.nowhere.com:4333/accounting`

DriverManager

```
Connection getConnection
    (String url, String user, String
    password)
```

- Connects to given JDBC URL with given user name and password
- Throws java.sql.SQLException
- returns a Connection object

Connection

- A Connection represents a session with a specific database.
- Within the context of a Connection, SQL statements are executed and results are returned.
- Can have multiple connections to a database
  - Some drivers don't support serialized connections
  - Fortunately, most do (now)
- Also provides "metadata" -- information about the database, tables, and fields
- Also methods to deal with transactions

Obtaining a Connection

```
String url   = "jdbc:odbc:Northwind";
try {
   Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection con = DriverManager.getConnection(url);
}
catch (ClassNotFoundException e)
   { e.printStackTrace(); }
catch (SQLException e)
   { e.printStackTrace(); }
```

Connection Methods

`Statement createStatement()`

- returns a new Statement object

`PreparedStatement prepareStatement(String sql)`

`Creates a precompiled queries for submissions to database.`

- returns a new PreparedStatement object

`CallableStatement prepareCall(String sql)`

- Access stored procedures in database.
- returns a new CallableStatement object

- Close()
- isClosed()

Statement

- A Statement object is used for executing a static SQL statement and obtaining the results produced by it.

Statement Methods

`ResultSet executeQuery(String)`

- Execute a SQL statement that returns a single ResultSet.

`int executeUpdate(String)`

- Execute a SQL INSERT, UPDATE or DELETE statement. Returns the number of rows changed.

`boolean execute(String)`

- Execute a SQL statement that may return multiple results.

- Why all these different kinds of queries? Optimization.

ResultSet

- A ResultSet provides access to a table of data generated by executing a Statement.
- Only one ResultSet per Statement can be open at once.
- The table rows are retrieved in sequence.
- A ResultSet maintains a cursor pointing to its current row of data.
- The 'next' method moves the cursor to the next row.
  - you can't rewind

ResultSet Methods

- boolean next()
  - activates the next row
  - the first call to next() activates the first row
  - returns false if there are no more rows
- void close()
  - disposes of the ResultSet
  - allows you to re-use the Statement that created it
  - automatically called by most Statement methods

ResultSet Methods

- *Type* get*Type*(int columnIndex)
  - returns the given field as the given type
  - fields indexed starting at 1 (not 0)
- *Type* get*Type*(String columnName)
  - same, but uses name of field
  - less efficient
- int findColumn(String columnName)
  - looks up column index given column name

ResultSet Methods

- String getString(int columnIndex)
- boolean getBoolean(int columnIndex)
- byte getByte(int columnIndex)
- short getShort(int columnIndex)
- int getInt(int columnIndex)
- long getLong(int columnIndex)
- float getFloat(int columnIndex)
- double getDouble(int columnIndex)
- Date getDate(int columnIndex)
- Time getTime(int columnIndex)
- Timestamp getTimestamp(int columnIndex)

ResultSet Methods

- String getString(String columnName)
- boolean getBoolean(String columnName)
- byte getByte(String columnName)
- short getShort(String columnName)
- int getInt(String columnName)
- long getLong(String columnName)
- float getFloat(String columnName)
- double getDouble(String columnName)
- Date getDate(String columnName)
- Time getTime(String columnName)
- Timestamp getTimestamp(String columnName)

isNull

- In SQL, NULL means the field is empty
- Not the same as o or “”
- In JDBC, you must explicitly ask if a field is null by calling ResultSet.isNull(column)

Sample Database

Employee ID	Last Name	First Name
1	Davolio	Nancy
2	Fuller	Andrew
3	Leverling	Janet
4	Peacock	Margaret
5	Buchanan	Steven

SELECT Example

```
Connection con =
DriverManager.getConnection(url, "alex",
"8675309");
Statement st = con.createStatement();
ResultSet results =
st.executeQuery("SELECT EmployeeID,
LastName, FirstName FROM Employees");
```

SELECT Example (Cont.)

```
while (results.next()) {
  int id = results.getInt(1);
  String last = results.getString(2);
  String first = results.getString(3);
  System.out.println("" + id + ": " +
first + " " + last);
}
st.close();
con.close();
```

Mapping Java Types to SQL Types

SQL type	Java Type
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Database Time

- Times in SQL are notoriously unstandard
- Java defines three classes to help
- java.sql.Date
  - year, month, day
- java.sql.Time
  - hours, minutes, seconds
- java.sql.Timestamp
  - year, month, day, hours, minutes, seconds, nanoseconds
  - usually use this one

Modifying the Database

- use executeUpdate if the SQL contains “INSERT” or “UPDATE”
- executeUpdate returns the number of rows modified
- executeUpdate also used for “CREATE TABLE” etc. (DDL)

Transaction Management

- Transactions are not explicitly opened and closed
- Instead, the connection has a state called *AutoCommit* mode
- if *AutoCommit* is true, then every statement is automatically committed
- default case: true

setAutoCommit

Connection.setAutoCommit(boolean)

- if *AutoCommit* is false, then every statement is added to an ongoing transaction
- you must explicitly commit or rollback the transaction using Connection.commit() and Connection.rollback()

Optimized Statements

- Prepared Statements
  - SQL calls you make again and again
  - allows driver to optimize (compile) queries
  - created with Connection.prepareStatement()
- Stored Procedures
  - written in DB-specific language
  - stored inside database
  - accesed with Connection.prepareCall()

Metadata

- Connection:
  - DatabaseMetaData getMetaData()
- ResultSet:
  - ResultSetMetaData getMetaData()

Connecting to the server

- First, make sure the MySQL server is running
- In your program,
  - import java.sql.Connection; // not com.mysql.jdbc.Connection
    import java.sql.DriverManager;
    import java.sql.SQLException;
  - Register the JDBC driver,
    Class.forName("com.mysql.jdbc.Driver").newInstance();
  - Invoke the getConnection() method,
    Connection con =
    DriverManager.getConnection("jdbc:mysql:///myDB",
    myUserName,
    myPassword);
  - or getConnection("jdbc:mysql:///myDB?user=dave&password=xxx")

A complete program

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JdbcExample1 {

    public static void main(String args[]) {
        Connection con = null;
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            con = DriverManager.getConnection("jdbc:mysql:///test", "root", "rootpswd");
            if (!con.isClosed())
                System.out.println("Successfully connected to MySQL server...");
        } catch(Exception e) {
            System.err.println("Exception: " + e.getMessage());
        } finally {
            try {
                if (con != null)
                    con.close();
            } catch(SQLException e) {}
        }
    }
}
```

Using the Connection object

- public Statement createStatement()
  throws SQLException
  - Creates a Statement object for sending SQL statements to the database. SQL statements without parameters are normally executed using Statement objects.
  - The Statement object may be reused for many statements
- public PreparedStatement prepareStatement(String sql)
  throws SQLException
  - Creates a PreparedStatement object for sending parameterized SQL statements to the database.
  - A SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

Issuing queries

- The following are methods on the Statement object:
  - int executeUpdate() -- for issuing queries that modify the database and return no result set
    - Use for DROP TABLE, CREATE TABLE, and INSERT
    - Returns the number of rows in the resultant table
  - ResultSet executeQuery() -- for queries that do return a result set.
    - Returns results as a ResultSet object

Creating a table

- ```
  CREATE TABLE animal (
      id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
      PRIMARY     KEY (id),
      name        CHAR(40),
      category    CHAR(40)
  )
  ```

- ```
  Statement s = conn.createStatement ();
  s.executeUpdate ("DROP TABLE IF EXISTS animal");
  s.executeUpdate (
          "CREATE TABLE animal ("
        + "id INT UNSIGNED NOT NULL AUTO_INCREMENT,"
        + "PRIMARY KEY (id),“
        + "name CHAR(40), category CHAR(40))");
  ```

Populating the table

```
int count;
count = s.executeUpdate (
            "INSERT INTO animal (name, category)"
            + " VALUES"
            + "('snake', 'reptile'),"
            + "('frog', 'amphibian'),"
            + "('tuna', 'fish'),"
            + "('racoon', 'mammal')");
  s.close ();
  System.out.println (count +
                        " rows were inserted");
```

ResultSet

- executeQuery() returns a ResultSet
  - ResultSet has a very large number of get*XXX* methods, such as
    - public String getString(String *columnName*)
    - public String getString(int *columnIndex*)
  - Results are returned from the current row
  - You can iterate over the rows:
    - public boolean next()
- ResultSet objects, like Statement objects, should be closed when you are done with them

Example, continued

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT id, name, category " +
                "FROM animal");
ResultSet rs = s.getResultSet ();
int count = 0;

    // Loop (next slide) goes here

rs.close ();
s.close ();
System.out.println (count + " rows were retrieved");
```

Example, continued

```
while (rs.next ()) {
    int idVal = rs.getInt ("id");
    String nameVal = rs.getString ("name");
    String catVal = rs.getString ("category");
    System.out.println (
            "id = " + idVal
            + ", name = " + nameVal
            + ", category = " + catVal);
    ++count;
}
```

Prepared statements

- Prepared statements are precompiled, hence much more efficient to use
  - PreparedStatement s;
    s = conn.prepareStatement (
    "INSERT INTO animal (name, category VALUES(?,?)");
    s.setString (1, nameVal);
    s.setString (2, catVal);
    int count = s.executeUpdate ();
    s.close ();
    System.out.println (count + " rows were inserted");

Error handling

```
try {
    Statement s = conn.createStatement ();
    s.executeQuery ("XYZ"); // issue invalid query
    s.close ();
}
catch (SQLException e) {
    System.err.println ("Error message: "
                        + e.getMessage ());
    System.err.println ("Error number: "
                        + e.getErrorCode ());
}
```

For Mysql.... DB connection..

```
Class.forName("com.mysql.jdbc.Driver");

Connection con =
DriverMagenager.getConnection(
"jdbc:mysql://localhost:3306/DBName",
"root", "root");

Statement stmt = con.createStatement();
```