JDBC
JAVA DATABASE CONNECTIVITY

NAMIT KHANDUJA

ASSTT. PROF.

DEPT. OF CS&E, FET, GKV.

OUTLINE

- JDBC
- JDBC PRODUCT COMPONENTS
- JDBC ARCHITECTURE
- DRIVERS
- STEPS
- EXAMPLE
- REFERENCES

WHAT IS JDBC?

- JDBC – JAVA DATABASE CONNECTIVITY
- THE JDBC API IS A JAVA API THAT CAN ACCESS ANY KIND OF TABULAR DATA, ESPECIALLY DATA STORED IN A RELATIONAL DATABASE.
- IT IS A STANDARD JAVA API FOR DATABASE-INDEPENDENT CONNECTIVITY BETWEEN THE JAVA PROGRAMMING LANGUAGE AND DATABASES.

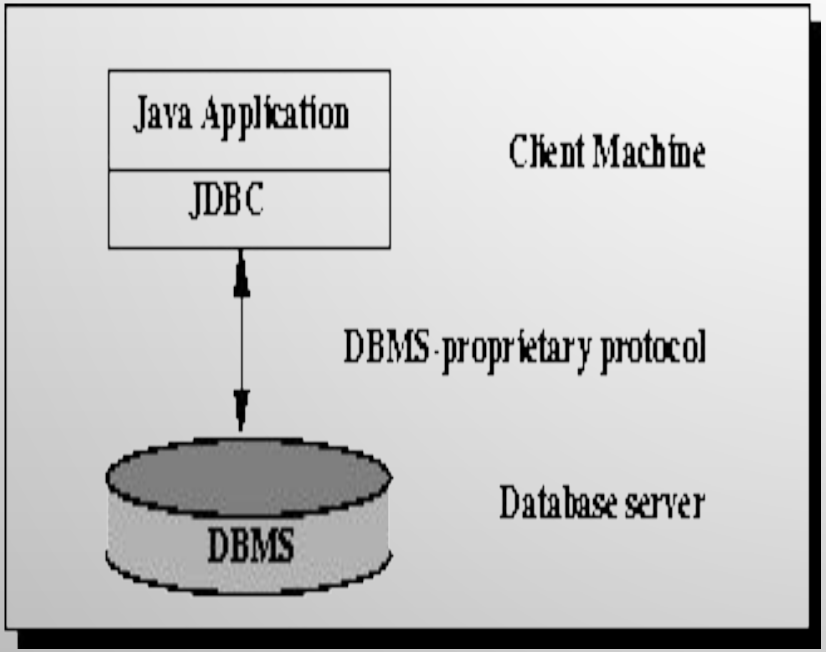
JDBC PRODUCT COMPONENTS

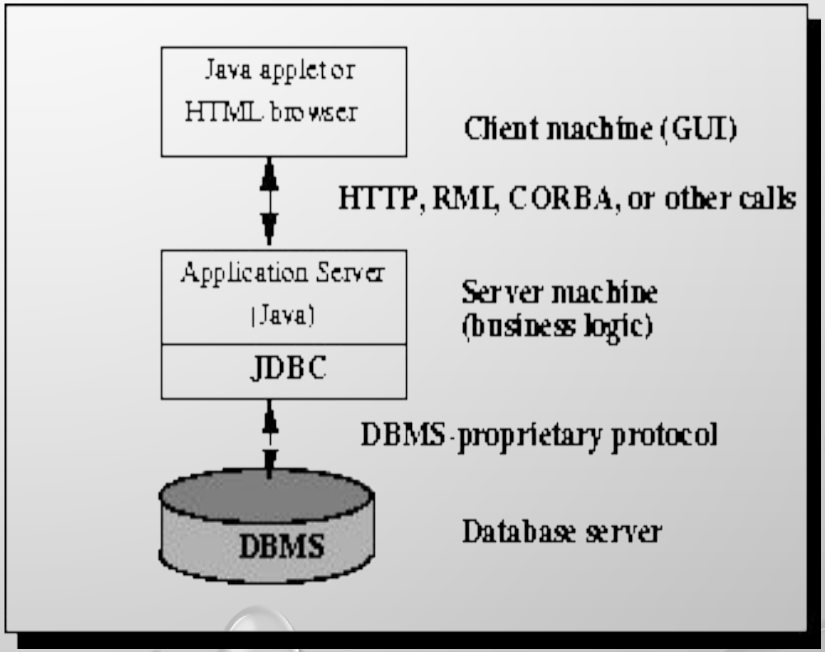
- **THE JDBC API** — THE JDBC™ API PROVIDES PROGRAMMATIC ACCESS TO RELATIONAL DATA FROM THE JAVA™ PROGRAMMING LANGUAGE. USING THE JDBC API, APPLICATIONS CAN EXECUTE SQL STATEMENTS, RETRIEVE RESULTS, AND PROPAGATE CHANGES BACK TO AN UNDERLYING DATA SOURCE. THE JDBC API CAN ALSO INTERACT WITH MULTIPLE DATA SOURCES IN A DISTRIBUTED, HETEROGENEOUS ENVIRONMENT.
- **JDBC DRIVER MANAGER** — THE JDBC DRIVERMANAGER CLASS DEFINES OBJECTS WHICH CAN CONNECT JAVA APPLICATIONS TO A JDBC DRIVER. DRIVERMANAGER HAS TRADITIONALLY BEEN THE BACKBONE OF THE JDBC ARCHITECTURE. IT IS QUITE SMALL AND SIMPLE.
- **JDBC TEST SUITE** — THE JDBC DRIVER TEST SUITE HELPS YOU TO DETERMINE THAT JDBC DRIVERS WILL RUN YOUR PROGRAM. THESE TESTS ARE NOT COMPREHENSIVE OR EXHAUSTIVE, BUT THEY DO EXERCISE MANY OF THE IMPORTANT FEATURES IN THE JDBC API.
- **JDBC-ODBC BRIDGE** — THE JAVA SOFTWARE BRIDGE PROVIDES JDBC ACCESS VIA ODBC DRIVERS. NOTE THAT YOU NEED TO LOAD ODBC BINARY CODE ONTO EACH CLIENT MACHINE THAT USES THIS DRIVER. AS A RESULT, THE ODBC DRIVER IS MOST APPROPRIATE ON A CORPORATE NETWORK WHERE CLIENT INSTALLATIONS ARE NOT A MAJOR PROBLEM, OR FOR APPLICATION SERVER CODE WRITTEN IN JAVA IN A THREE-TIER ARCHITECTURE. (EXCLUDED IN JAVA 1.8)

JDBC ARCHITECTURE

- THE JDBC API SUPPORTS BOTH TWO-TIER AND THREE-TIER PROCESSING MODELS FOR DATABASE ACCESS.

TWO TIER THREE TIER




Java applet or HTML browser
Client machine (GUI)
HTTP, RMI, CORBA, or other calls
Application Server (Java)
JDBC
Server machine (business logic)
DBMS-proprietary protocol
DBMS
Database server

JDBC DRIVERS

JDBC API DOES NOT DIRECTLY COMMUNICATE WITH THE DATABASE. IT USES **JDBC DRIVER** OF THE DATABASE TO INTERACT WITH THE DATABASE. **JDBC DRIVER** IS A SOFTWARE COMPONENT PROVIDED ALONG WITH THE DATABASE WHICH IS REQUIRED BY THE JDBC API TO INTERACT WITH THE DATABASE. EACH DATABASE WILL HAVE ITS OWN JDBC DRIVER.

TYPES OF JDBC DRIVERS

1) TYPE 1 JDBC DRIVER / JDBC-ODBC BRIDGE DRIVER

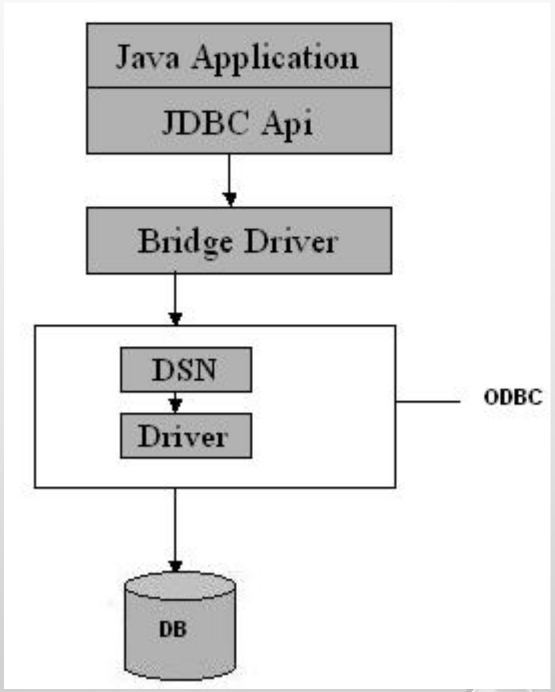
2) TYPE 2 JDBC DRIVER / NATIVE API DRIVER

3) TYPE 3 JDBC DRIVER / NETWORK PROTOCOL DRIVER

4) TYPE 4 JDBC DRIVER / NATIVE PROTOCOL DRIVER /THIN DRIVER

TYPE 1
JDBC-ODBC BRIDGE DRIVER

- PROVIDE THE BRIDGE BETWEEN JDBC AND ODBC API AND HENCE THE NAME **'JDBC-ODBC BRIDGE DRIVERS'.** THIS TYPE OF DRIVERS TRANSLATE ALL JDBC CALLS INTO ODBC CALLS AND SENDS THEM TO ODBC DRIVER WHICH INTERACTS WITH THE DATABASE.



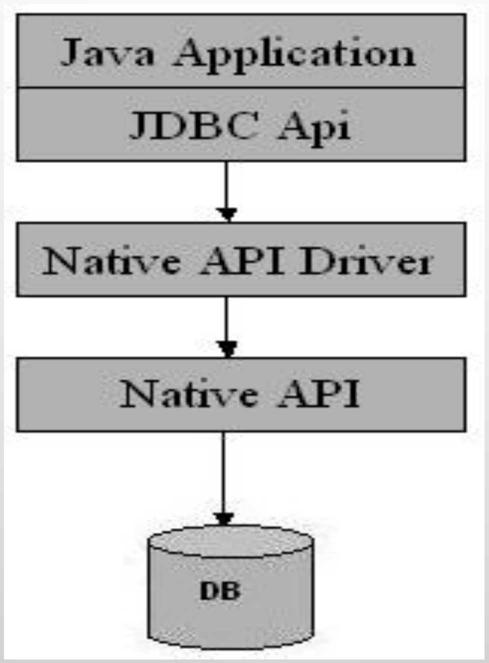
CONTD.

- **ADVANTAGE**
- THE JDBC-ODBC BRIDGE ALLOWS ACCESS TO ALMOST ANY DATABASE, SINCE THE DATABASE'S ODBC DRIVERS ARE ALREADY AVAILABLE.
- **DISADVANTAGES**
- 1. SINCE THE BRIDGE DRIVER IS NOT WRITTEN FULLY IN JAVA, TYPE 1 DRIVERS ARE NOT PORTABLE.
  2. A PERFORMANCE ISSUE IS SEEN AS A JDBC CALL GOES THROUGH THE BRIDGE TO THE ODBC DRIVER, THEN TO THE DATABASE, AND THIS APPLIES EVEN IN THE REVERSE PROCESS. THEY ARE THE SLOWEST OF ALL DRIVER TYPES.
  3. THE CLIENT SYSTEM REQUIRES THE ODBC INSTALLATION TO USE THE DRIVER..

4.IT IS TIME CONSUMING AND RAISES THE PERFORMANCE ISSUES.

TYPE 2
NATIVE API DRIVER

- TRANSLATES ALL JDBC METHOD CALLS INTO DATABASE SPECIFIC CALLS USING NATIVE API OF THE DATABASE.



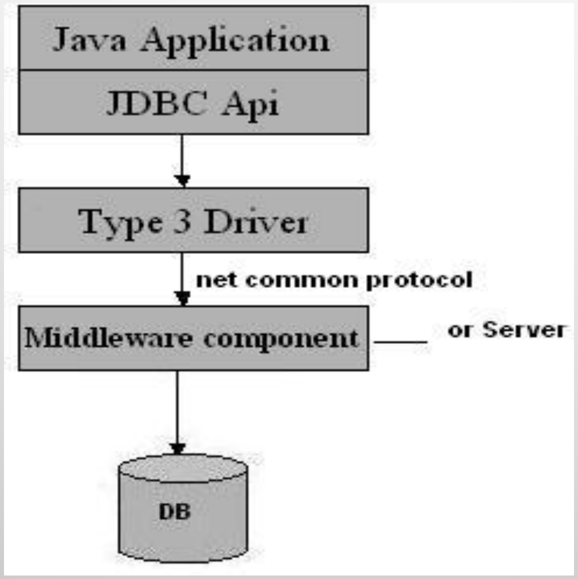
CONTD

- **ADVANTAGE**
- TYPICALLY OFFER BETTER PERFORMANCE THAN THE JDBC-ODBC BRIDGE AS THE LAYERS OF COMMUNICATION (TIERS) ARE LESS
- **<u>DISADVANTAGE</u>**

1. NATIVE API MUST BE INSTALLED IN THE CLIENT SYSTEM AND HENCE TYPE 2 DRIVERS CANNOT BE USED FOR THE INTERNET.
2. IT'S NOT WRITTEN IN JAVA LANGUAGE WHICH FORMS A PORTABILITY ISSUE.
3. PLANTFORM DEPENDENT
4. MOSTLY OBSOLETE NOW AND USUALLY NOT THREAD SAFE.

TYPE 3
NETWORK PROTOCOL DRIVER

- MAKE USE OF **MIDDLE WARE** OR **APPLICATION SERVER** THAT TRANSLATES ALL JDBC CALLS INTO DATABASE SPECIFIC CALLS



CONTD.

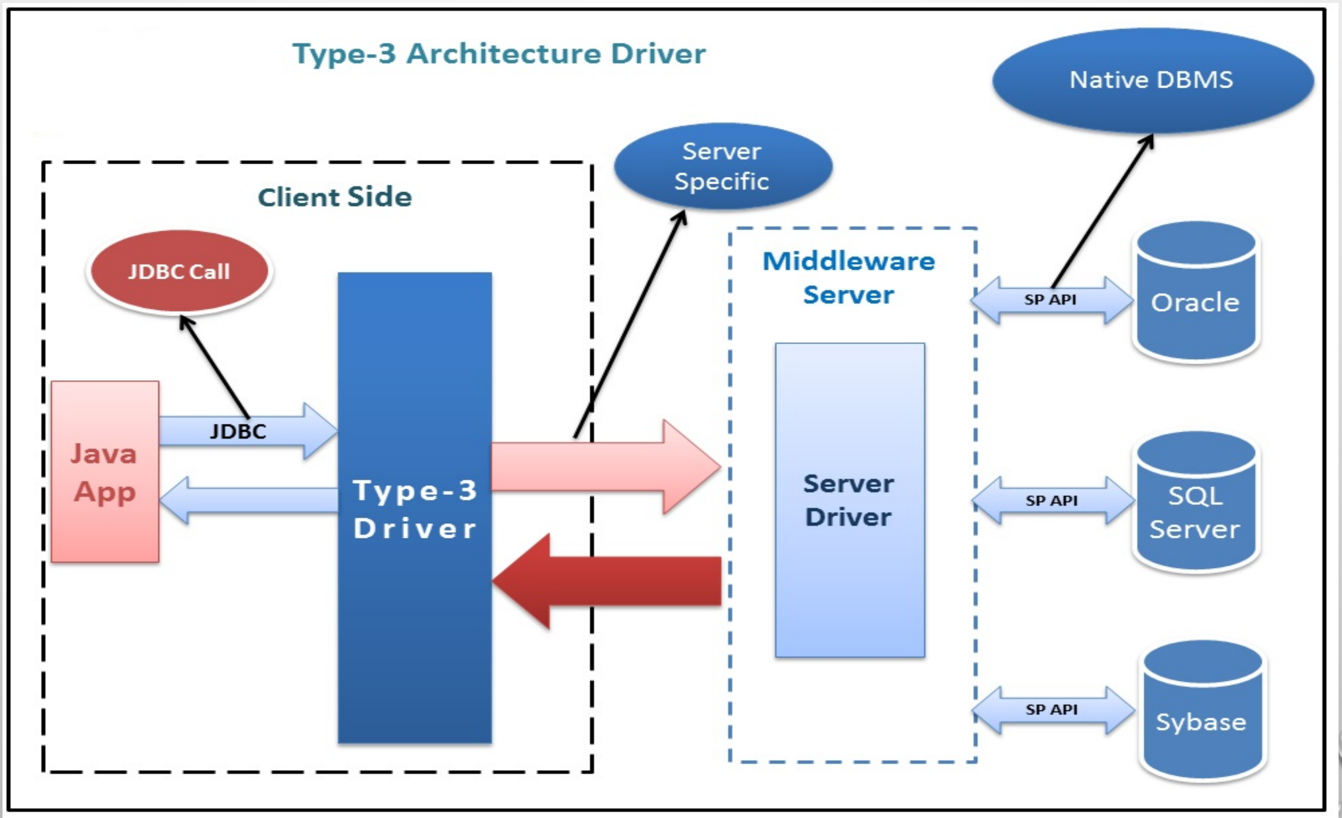
- TYPE 3 DRIVER USES THE MIDDLE TIER(APPLICATION SERVER) BETWEEN THE CALLING PROGRAM AND THE DATABASE AND THIS MIDDLE TIER CONVERTS JDBC METHOD CALLS INTO THE VENDOR SPECIFIC DATABASE PROTOCOL AND THE SAME DRIVER CAN BE USED FOR MULTIPLE DATABASES ALSO SO IT'S ALSO KNOWN AS A NETWORK-PROTOCOL DRIVER AS WELL AS A JAVA DRIVER FOR DATABASE MIDDLEWARE.

ADVANTAGE

- MAIN ADVANTAGE OF THIS DRIVER IS THAT IT IS ENTIRELY WRITTEN IN JAVA LANGUAGE. SO NO PORTABILITY ISSUES.
- (1) THERE IS NO NEED FOR THE VENDOR DATABASE LIBRARY ON THE CLIENT MACHINE BECAUSE THE MIDDLEWARE IS DATABASE INDEPENDENT AND IT COMMUNICATES WITH CLIENT.
- (2) TYPE 3 DRIVER CAN BE USED IN ANY WEB APPLICATION AS WELL AS ON INTERNET ALSO BECAUSE THERE IS NO ANY SOFTWARE REQUIRE AT CLIENT SIDE.
- (3) A SINGLE DRIVER CAN HANDLE ANY DATABASE AT CLIENT SIDE SO THERE IS NO NEED A SEPARATE DRIVER FOR EACH DATABASE.
- (4) THE MIDDLEWARE SERVER CAN ALSO PROVIDE THE TYPICAL SERVICES SUCH AS CONNECTIONS, AUDITING, LOAD BALANCING, LOGGING ETC.

DISADVANTAGE

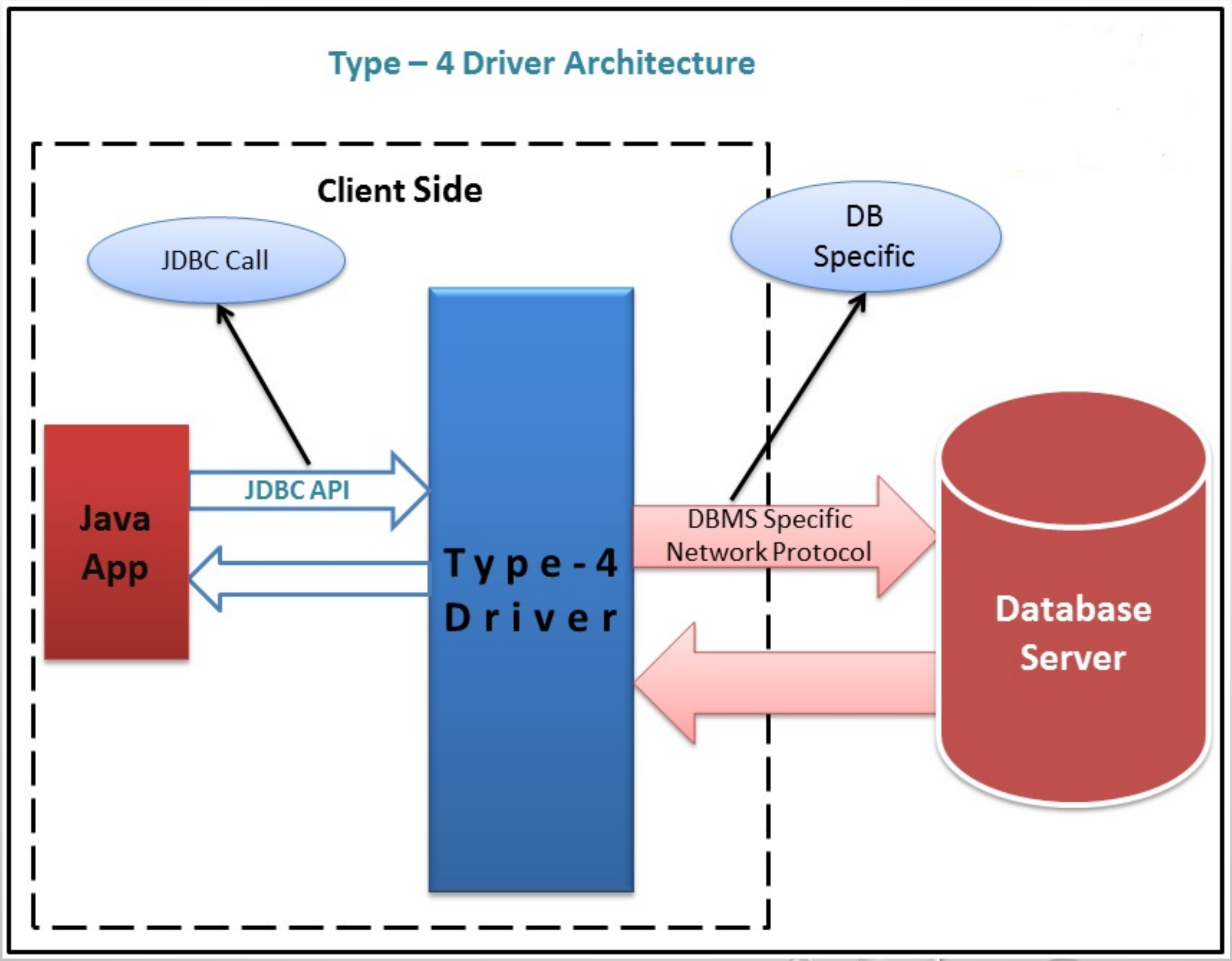
- (1) AN EXTRA LAYER ADDED, MAY BE TIME CONSUMING.
- (2) AT THE MIDDLEWARE DEVELOP THE DATABASE SPECIFIC CODING, MAY BE INCREASE COMPLEXITY.



TYPE 4 NATIVE PROTOCOL DRIVERS/ALL JAVA DRIVER/ PURE JAVA DRIVER

- TYPE 4 DRIVER CONVERTS JDBC METHOD CALLS DIRECTLY INTO THE VENDOR SPECIFIC DATABASE PROTOCOL AND IN BETWEEN DO NOT NEED TO BE CONVERTED ANY OTHER FORMATTED SYSTEM SO THIS IS THE FASTEST WAY TO COMMUNICATE QUIRES TO DBMS AND IT IS COMPLETELY WRITTEN IN JAVA

CONTD.


Type – 4 Driver Architecture
Client Side
JDBC Call
DB Specific
JDBC API
Java App
Type-4 Driver
DBMS Specific Network Protocol
Database Server

CONTD.

- **ADVANTAGE:**
- (1) IT'S A 100% PURE JAVA DRIVER SO IT'S A PLATFORM INDEPENDENCE.
- (2) NO TRANSLATION OR MIDDLEWARE LAYERS ARE USED SO CONSIDER AS A FASTER THAN OTHER DRIVERS.
- (3) THE ALL PROCESS OF THE APPLICATION-TO-DATABASE CONNECTION CAN MANAGE BY JVM SO THE DEBUGGING IS ALSO MANAGED EASILY.
- **DISADVANTAGE:**
- (1)THERE IS A SEPARATE DRIVER NEEDED FOR EACH DATABASE AT THE CLIENT SIDE.
- (2) DRIVERS ARE DATABASE DEPENDENT, AS DIFFERENT DATABASE VENDORS USE DIFFERENT NETWORK PROTOCOLS.

BASIC STEPS

- 1.ESTABLISH A **CONNECTION**
- 2.CREATE JDBC **STATEMENTS**
- 3.EXECUTE **SQL** STATEMENTS
- 4.GET **RESULTSET**
- 5.**CLOSE** CONNECTIONS

ESTABLISH A CONNECTION

- **IMPORT SQL PACKAGE**
- **LOAD THE DRIVER**
  - CLASS.FORNAME("COM.MYSQL.JDBC.DRIVER");
  - //DYNAMICALLY LOADS A DRIVER CLASS
- **ESTABLISH A CONNECTION**

CONNECTION CONN = DRIVERMANAGER.GETCONNECTION(MYSQL: "JDBC:MYSQL://HOSTNAME:PORT/DATABASENAME", "USERNAME", "PASSWORD“);

//ESTABLISHES CONNECTION TO DATABASE BY OBTAINING A *CONNECTION* OBJECT

CREATE JDBC STATEMENT

- STATEMENT STMT = CON.CREATESTATEMENT() ;
- CREATES A STATEMENT OBJECT FOR SENDING SQL STATEMENTS TO THE DATABASE

EXECUTE SQL STATEMENT

- STRING CREATESTUDENT = "CREATE TABLE STUDENT " +

  "(ROLLNO INTEGER NOT NULL, NAME VARCHAR(32), " + "MARKS INTEGER)";

  STMT.**EXECUTEUPDATE**(CREATESTUDENT);

  //WHAT DOES THIS STATEMENT DO?

- STRING INSERTSTUDENT = "INSERT INTO STUDENT VALUES“ + "(123456789,ABC,100)";

  STMT.**EXECUTEUPDATE**(INSERTSTUDENT);

GET RESULTSET

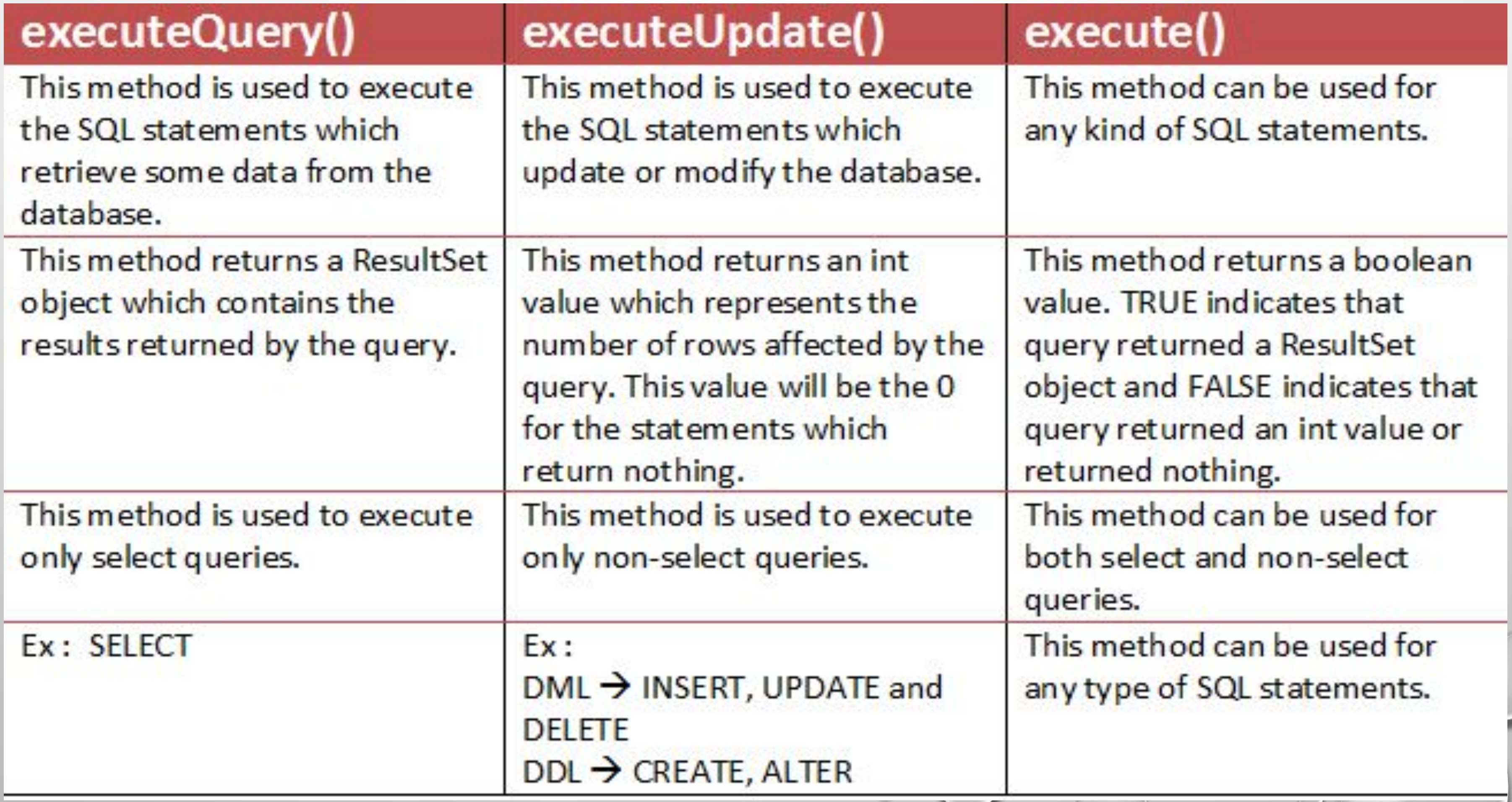
```
STRING QUERYSTUDENT = "SELECT * FROM STUDENT";

RESULTSET RS = STMT.EXECUTEQUERY(QUERYSTUDENT);

WHILE (RS.NEXT()) {
      INT SSN = RS.GETINT(“ROLLNO");
      STRING NAME = RS.GETSTRING("NAME");
      INT MARKS = RS.GETINT("MARKS");
}
```

CLOSE CONNECTION

- STMT.CLOSE();
- CON.CLOSE();

Statement	PreparedStatement	CallableStatement
It is used to execute normal SQL queries.	It is used to execute parameterized or dynamic SQL queries.	It is used to call the stored procedures.
It is preferred when a particular SQL query is to be executed only once.	It is preferred when a particular query is to be executed multiple times.	It is preferred when the stored procedures are to be executed.
You cannot pass the parameters to SQL query using this interface.	You can pass the parameters to SQL query at run time using this interface.	You can pass 3 types of parameters using this interface. They are – IN, OUT and IN OUT.
This interface is mainly used for DDL statements like CREATE, ALTER, DROP etc.	It is used for any kind of SQL queries which are to be executed multiple times.	It is used to execute stored procedures and functions.
The performance of this interface is very low.	The performance of this interface is better than the Statement interface (when used for multiple execution of same query).	The performance of this interface is high.

EXECUTEQUERY() VS EXECUTEUPDATE() VS EXECUTE()



executeQuery()	executeUpdate()	execute()
This method is used to execute the SQL statements which retrieve some data from the database.	This method is used to execute the SQL statements which update or modify the database.	This method can be used for any kind of SQL statements.
This method returns a ResultSet object which contains the results returned by the query.	This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing.	This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing.
This method is used to execute only select queries.	This method is used to execute only non-select queries.	This method can be used for both select and non-select queries.
Ex : SELECT	Ex : DML → INSERT, UPDATE and DELETE DDL → CREATE, ALTER	This method can be used for any type of SQL statements.

EXAMPLE 1

IMPORT JAVA.SQL.DRIVERMANAGER;

IMPORT JAVA.SQL.CONNECTION;

IMPORT JAVA.SQL.SQLEXCEPTION;

PUBLIC CLASS JDBCEXAMPLE {

PUBLIC STATIC VOID MAIN(STRING[] ARGV) {

- SYSTEM.OUT.PRINTLN("-------- MYSQL JDBC CONNECTION TESTING ------------");
- TRY {
- CLASS.FORNAME("COM.MYSQL.JDBC.DRIVER");
- } CATCH (CLASSNOTFOUNDEXCEPTION E)
- { SYSTEM.OUT.PRINTLN("WHERE IS YOUR MYSQL JDBC DRIVER?"); E.PRINTSTACKTRACE(); RETURN; }
- SYSTEM.OUT.PRINTLN("MYSQL JDBC DRIVER REGISTERED!");
- CONNECTION CONNECTION = NULL;
- TRY { CONNECTION = DRIVERMANAGER.GETCONNECTION("JDBC:MYSQL://LOCALHOST:3306/DBNAME","ROOT", "PASSWORD"); } CATCH (SQLEXCEPTION E)
- { SYSTEM.OUT.PRINTLN("CONNECTION FAILED! CHECK OUTPUT CONSOLE"); E.PRINTSTACKTRACE(); RETURN; }
- IF (CONNECTION != NULL) { SYSTEM.OUT.PRINTLN("YOU MADE IT, TAKE CONTROL YOUR DATABASE NOW!"); } ELSE { SYSTEM.OUT.PRINTLN("FAILED TO MAKE CONNECTION!"); } } }

INET ADDRESS

- THE **INETADDRESS** CLASS HAS NO VISIBLE CONSTRUCTORS. TO CREATE AN **INETADDRESS** OBJECT, YOU HAVE TO USE ONE OF THE AVAILABLE FACTORY METHODS. *FACTORY METHODS* ARE MERELY A CONVENTION WHEREBY STATIC METHODS IN A CLASS RETURN AN INSTANCE OF THAT CLASS. THIS IS DONE IN LIEU OF OVERLOADING A CONSTRUCTOR WITH VARIOUS PARAMETER LISTS WHEN HAVING UNIQUE METHOD NAMES MAKES THE RESULTS MUCH CLEARER. IN THE CASE OF **INETADDRESS**, THE THREE METHODS **GETLOCALHOST()**, **GETBYNAME()**, AND **GETALLBYNAME()** CAN BE USED TO CREATE INSTANCES OF **INETADDRESS**.

- THESE METHODS ARE SHOWN HERE:
- STATIC INETADDRESS GETLOCALHOST( )
  THROWS UNKNOWNHOSTEXCEPTION
  STATIC INETADDRESS GETBYNAME(STRING *HOSTNAME*)
  THROWS UNKNOWNHOSTEXCEPTION
  STATIC INETADDRESS[ ] GETALLBYNAME(STRING *HOSTNAME*)
  THROWS UNKNOWNHOSTEXCEPTION
- THE **GETLOCALHOST( )** METHOD SIMPLY RETURNS
  THE **INETADDRESS** OBJECT THAT REPRESENTS THE LOCAL HOST.
  THE **GETBYNAME( )** METHOD RETURNS AN **INETADDRESS** FOR A HOST NAME PASSED TO IT. IF THESE METHODS ARE UNABLE TO RESOLVE THE HOST NAME, THEY THROW AN **UNKNOWNHOSTEXCEPTION**.
- ON THE INTERNET, IT IS COMMON FOR A SINGLE NAME TO BE USED TO REPRESENT SEVERAL MACHINES. IN THE WORLD OF WEB SERVERS, THIS IS ONE WAY TO PROVIDE SOME DEGREE OF SCALING.
  THE **GETALLBYNAME( )** FACTORY METHOD RETURNS AN ARRAY
  OF **INETADDRESS**ES THAT REPRESENT ALL OF THE ADDRESSES THAT A PARTICULAR NAME RESOLVES TO. IT WILL ALSO THROW AN**UNKNOWNHOSTEXCEPTION** IF IT CAN'T RESOLVE THE NAME TO AT LEAST ONE ADDRESS.

- THE FOLLOWING EXAMPLE PRINTS THE ADDRESSES AND NAMES OF THE LOCAL MACHINE AND TWO WELL-KNOWN INTERNET WEB SITES:

-
  ```
  // DEMONSTRATE INETADDRESS.
  IMPORT JAVA.NET.*;
  CLASS INETADDRESSTEST
  {
  PUBLIC STATIC VOID MAIN(STRING ARGS[]) THROWS
  UNKNOWNHOSTEXCEPTION {
  INETADDRESS ADDRESS = INETADDRESS.GETLOCALHOST();
  SYSTEM.OUT.PRINTLN(ADDRESS);
  ADDRESS = INETADDRESS.GETBYNAME("STARWAVE.COM");
  SYSTEM.OUT.PRINTLN(ADDRESS);
  INETADDRESS SW[] = INETADDRESS.GETALLBYNAME("WWW.NBA.COM");
  FOR (INT I=0; I<SW.LENGTH; I++)
  SYSTEM.OUT.PRINTLN(SW[I]);
  }
  }
  ```

- HERE IS THE OUTPUT PRODUCED BY THIS PROGRAM. (OF COURSE, THE OUTPUT YOU SEE WILL BE SLIGHTLY DIFFERENT.)

-
  ```
  DEFAULT/206.148.209.138
  STARWAVE.COM/204.202.129.90
  WWW.NBA.COM/204.202.130.223
  ```

REFERENCES

- HTTPS://SITES.GOOGLE.COM/SITE/ENTERPRISECOMPUTINGWITHJAVA/HOME
- HTTP://JAVA2ALL.COM/JAVA-TUTORIAL/JDBC-TUTORIAL/JDBC-DRIVER-TYPES/
- HTTP://JAVAHUNGRY.BLOGSPOT.COM/2013/12/JDBC-DRIVER-TYPES-JAVA-EXAMPLE-CODE.HTML
- HTTP://WWW.JDBC-TUTORIAL.COM/JDBC-DRIVER-TYPES.HTM
- HTTP://JAVACONCEPTOFTHEDAY.COM/TYPES-OF-JDBC-DRIVERS/
- HTTPS://WWW.TUTORIALSPOINT.COM/JDBC/JDBC-DRIVER-TYPES.HTM
- HTTP://WWW.MKYONG.COM/TUTORIALS/JDBC-TUTORIALS/
- JAVACONCEPTOFTHEDAY.COM/DIFFERENCE-BETWEEN-EXECUTEQUERY-EXECUTEUPDATE-EXECUTE-IN-JDBC/
- HTTPS://WWW.GEEKSFORGEEKS.ORG/TAG/JAVA-NETWORKING/