CSC584 Enterprise Programming

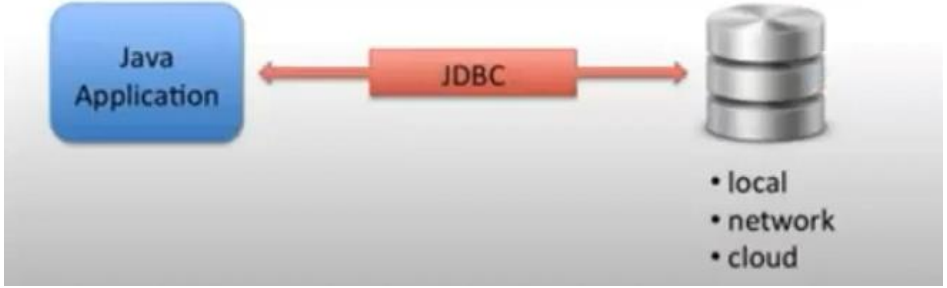
CHAPTER 5 – JAVA DATABASE CONNECTIVITY

Chapter 5 Outline

- Overview of java database programming
- Define JDBC API
- Describe various types of JDBC
- Identify JDBC product
- Describe the 2 tier server client model
- Setup JDBC connection to a database with JSP and Servlet
- Create and Execute SQL statement
- Describe ResultSet Object

What is JDBC?

- Allow Java applications to connect to relational database



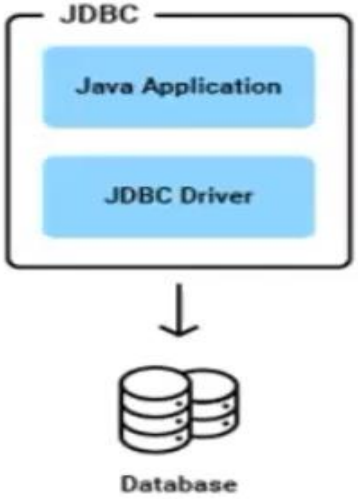
- JDBC supports a large number of databases

What is JDBC?

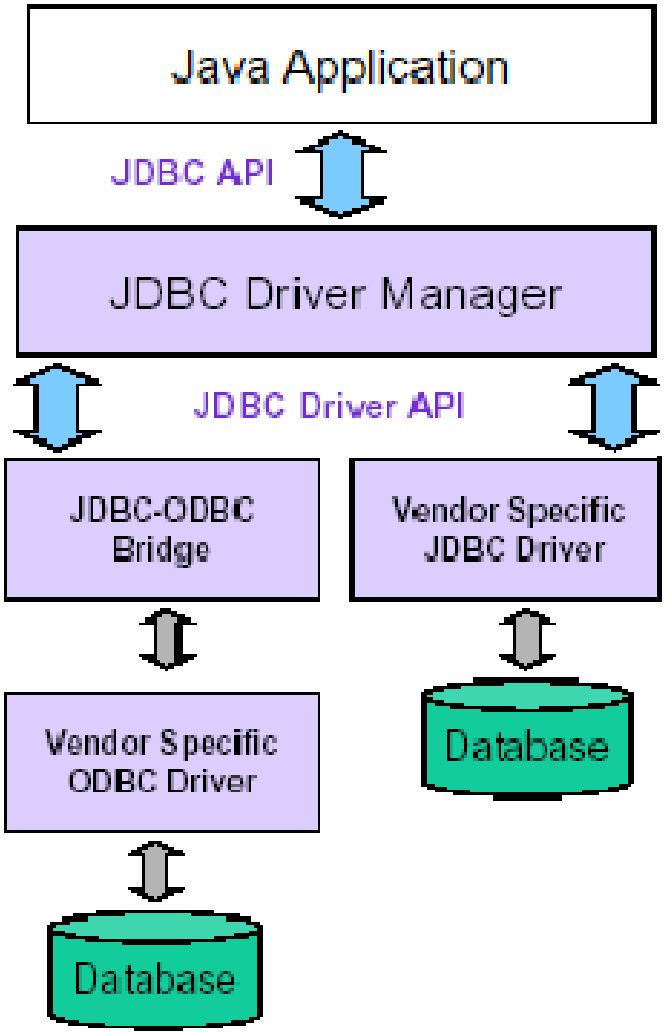
- "An API that lets you access virtually any tabular data source from the Java programming language"
- JDBC Data Access API
- What's an API?
  - See J2SE documentation
- What's a tabular data source?
  - "… access virtually any data source, from relational databases to spreadsheets and flat files."

JDBC API

- Features of JDBC API
  - Provide portable access to various databases
    - No need to develop code for different databases
- Can build your own custom SQL statements
  - Select, Insert, update, delete
  - Complex SQL queries – join, union



General Architecture



SQL

- Structured Query Language, pronounced S-Q-L, or Sequel
- To access or write applications for database systems, you need to use the Structured Query Language (SQL).
- SQL is the universal language for accessing relational database systems.
- Application programs may allow users to access database without directly using SQL, but these applications themselves must use SQL to access the database.

Examples of simple SQL statements

DDL
- Create table
- Drop table
- Describe table

DML
- Select
- Insert
- Delete
- Update

- Creating a basic table involves naming the table and defining its columns and each column's data type.

```
create table Course (
  courseId char(5),
  subjectId char(4) not null,
  courseNumber integer,
  title varchar(50) not null,
  numOfCredits integer,
  primary key (courseId)
);
```

```
create table Student (
  ssn char(9),
  firstName varchar(25),
  mi char(1),
  lastName varchar(25),
  birthDate date,
  street varchar(25),
  phone char(11),
  zipCode char(5),
  deptId char(4),
  primary key (ssn)
);
```

Examples of simple SQL statements

Create table
Drop table
Describe table
Select
Insert
Delete
Update

- remove a table definition.

```
drop table Enrollment;
drop table Course;
drop table Student;
```

Examples of simple SQL statements

Create table
Drop table
Describe table
Select
Insert
Delete
Update

- display the structure of table.

```
describe Course;
```

Examples of simple SQL statements

Create table
Drop table
Describe table
Select
Insert
Delete
Update

- fetch the data from a database table which returns this data in the form of a result table.

```
select firstName, mi, lastName
from Student
where deptId = 'CS';

select firstName, mi, lastName
from Student
where deptId = 'CS' and zipCode = '31411';

select *
from Student
where deptId = 'CS' and zipCode = '31411';
```

Examples of simple SQL statements

Create table
Drop table
Describe table
Select
Insert
Delete
Update

- add new rows of data to a table in the database.

```
insert into Course (courseId, subjectId, courseNumber, title)
values ('11113', 'CSCI', '3720', 'Database Systems', 3);
```

Examples of simple SQL statements

Create table
Drop table
Describe table
Select
Insert
Delete
Update

- delete the existing records from a table.

```
delete Course
where title = 'Database System';
```

Examples of simple SQL statements

Create table
Drop table
Describe table
Select
Insert
Delete
Update

- modify the existing records in a table.

```
update Course
set numOfCredits = 4
where title = 'Database Systems';
```

Basic steps to use a database in Java

1. Establish a **connection**
2. Create JDBC **Statements**
3. Execute **SQL** Statements
4. GET **ResultSet**
5. **Close** connections

Basic steps to use a database in Java

1. Establish a **connection**

import java.sql.*;

Load the vendor specific driver

- Dynamically loads a driver class, for Oracle database

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Make the connection

- Establishes connection to database by obtaining a *Connection* object

```
Connection con = DriverManager.getConnection( "jdbc:oracle:thin:@oracle-prod:1521:OPROD", username, passwd);
```

Basic steps to use a database in Java

2. Create JDBC **Statements**

- Creates a Statement object for sending SQL statements to the database

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

Basic steps to use a database in Java

3. Execute **SQL** Statements

```
String createLehigh = “create table Lehigh " +
"(SSN Integer not null, Name VARCHAR(32), " + "Marks Integer)";

stmt.executeUpdate(createLehigh);
//What does this statement do?

String insertLehigh = "insert into Lehigh values" +"(123456789,'abc',100)";

stmt.executeUpdate(insertLehigh);
```

Basic steps to use a database in Java

4. GET **ResultSet**

```
String queryLehigh= "select * from Lehigh";

ResultSet rs= Stmt.executeQuery(queryLehigh);
//What does this statement do?

while (rs.next()) {
   int ssn= rs.getInt("SSN");
   String name = rs.getString("NAME");
   int marks = rs.getInt("MARKS");
}
```

Basic steps to use a database in Java

5. **Close** connections

```
stmt.close();
con.close();
```

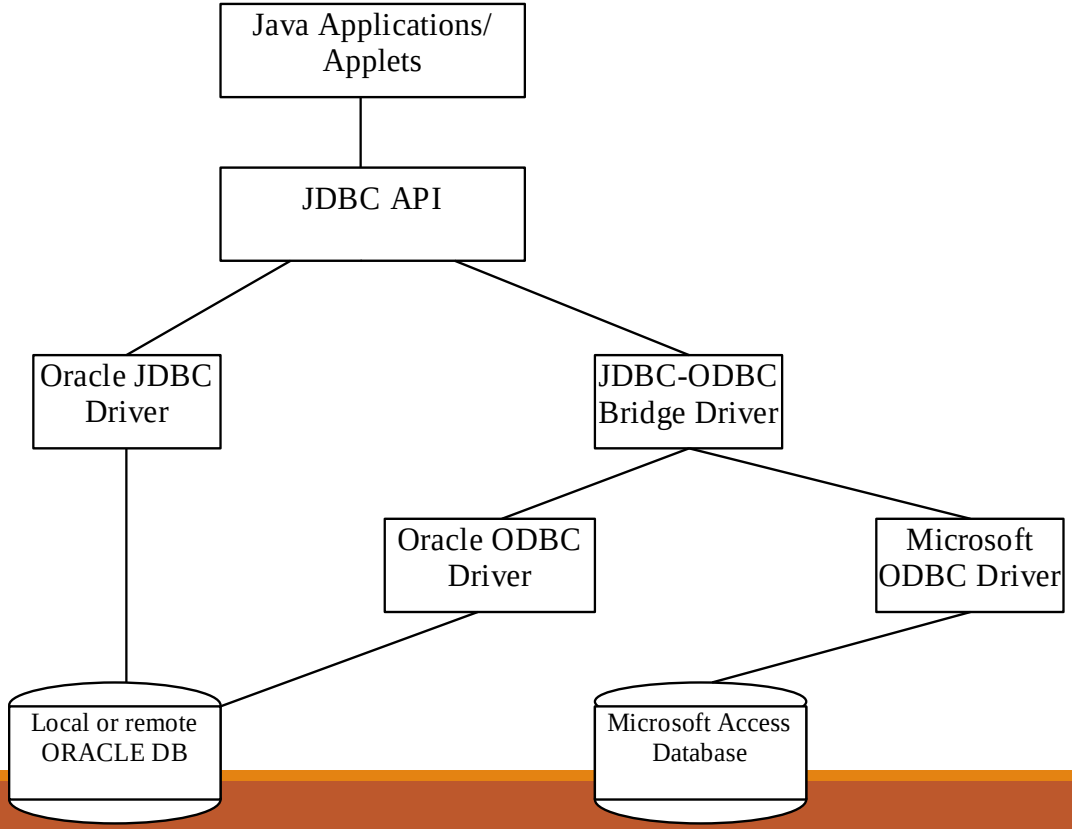
Why Java for Database Programming?

- First, Java is platform independent. You can develop platform-independent database applications using SQL and Java for any relational database systems.
- Second, the support for accessing database systems from Java is built into Java API, so you can create database applications using all Java code with a common interface.
- Third, Java is taught in almost every university either as the first programming language or as the second programming language.

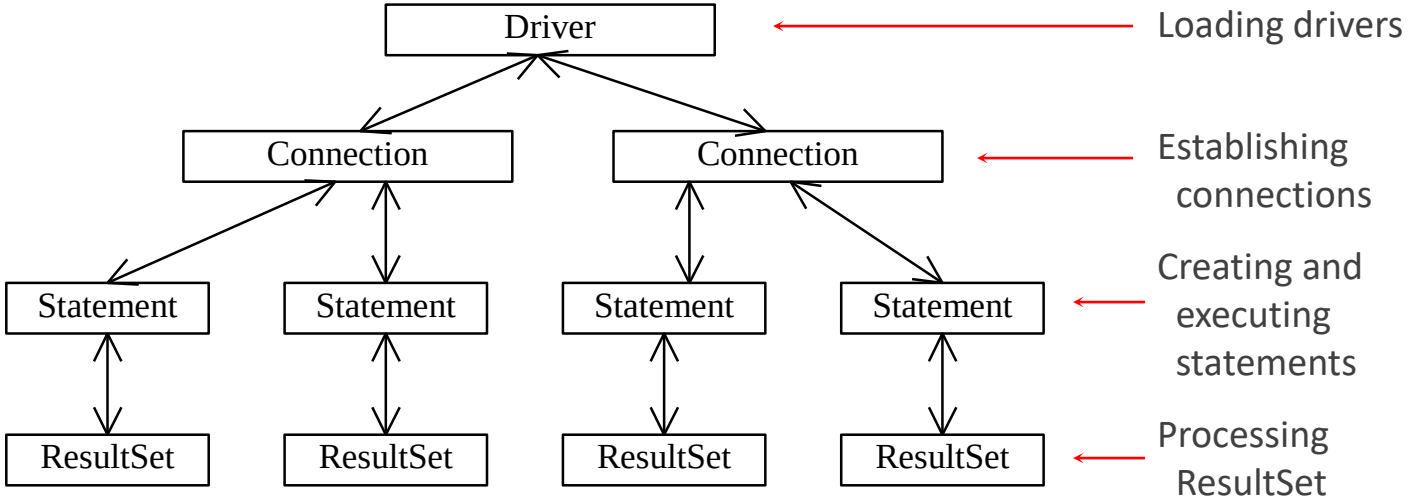
Database Applications Using Java

- GUI
- Client/Server
- Server-Side programming

The Architecture of JDBC



The JDBC Interfaces



Developing JDBC Programs

Loading driver

Establishing connections

Creating and executing statements

Processing ResultSet

Close connection

Statement to load a driver:

```
Class.forName("JDBC Driver Class");
```

A driver is a class. For example:

Database	Driver Class	Source
Access	sun.jdbc.odbc.jdbcOdbcDriver	Already in JDK
MySQL	com.mysql.jdbc.Driver	Website
Oracle	oracle.jdbc.driver.OracleDriver	Website
Java DB	org.apache.derby.jdbc.ClientDriver	Already in JDK

The JDBC-ODBC driver for Access is bundled in JDK.

MySQL driver class is in mysqljdbc.jar

Oracle driver class is in classes12.jar

Example:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

To use the MySQL and Oracle drivers, you have to add mysqlconnector.jar and ojdbc6.jar in the classpath

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Close connection

Connection conn = DriverManager.getConnection(database URL);

Database	URL Pattern
Access	jdbc:odbc:dataSource
MySQL	jdbc:mysql://hostname/dbname
Oracle	jdbc:oracle:thin:@hostname:port#:oracleDBSID
Java DB	jdbc:derby://hostname:port#/dbname

Examples:

For Access:

```
Connection conn = DriverManager.getConnection("jdbc:odbc:ExampleMDBDataSource");
```

For MySQL:

```
Connection conn = DriverManager.getConnection ("jdbc:mysql://localhost/test");
```

For Oracle:

```
Connection conn =
    DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",  "hr", "hr");
```

For Java DB:

```
Connection conn = DriverManager.getConnection ("jdbc:derby://localhost:1527/test");
```

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Close connection

Creating statement:

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

Executing statement (for update, delete, insert, DDL):

```
stmt.executeUpdate
   ("create table Temp (col1 char(5), col2 char(5))");
```

Executing statement (for select):

```
// Select the columns from the Student table
ResultSet rs = stmt.executeQuery
  ("select firstName, mi, lastName from Student where lastName "
   + " = 'Smith'");
```

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Close connection

Executing statement (for select):

```
// Select the columns from the Student table
ResultSet rs = stmt.executeQuery
  ("select firstName, mi, lastName from Student where lastName "
    + " = 'Smith'");
```

Processing ResultSet (for select):

```
// Iterate through the result and print the student names
while (rs.next())
  System.out.println(rs.getString(1) + " " + rs.getString(2)
    + ". " + rs.getString(3));
```

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Close connection

```
// Close the connection
stmt.close();
conn.close()
```

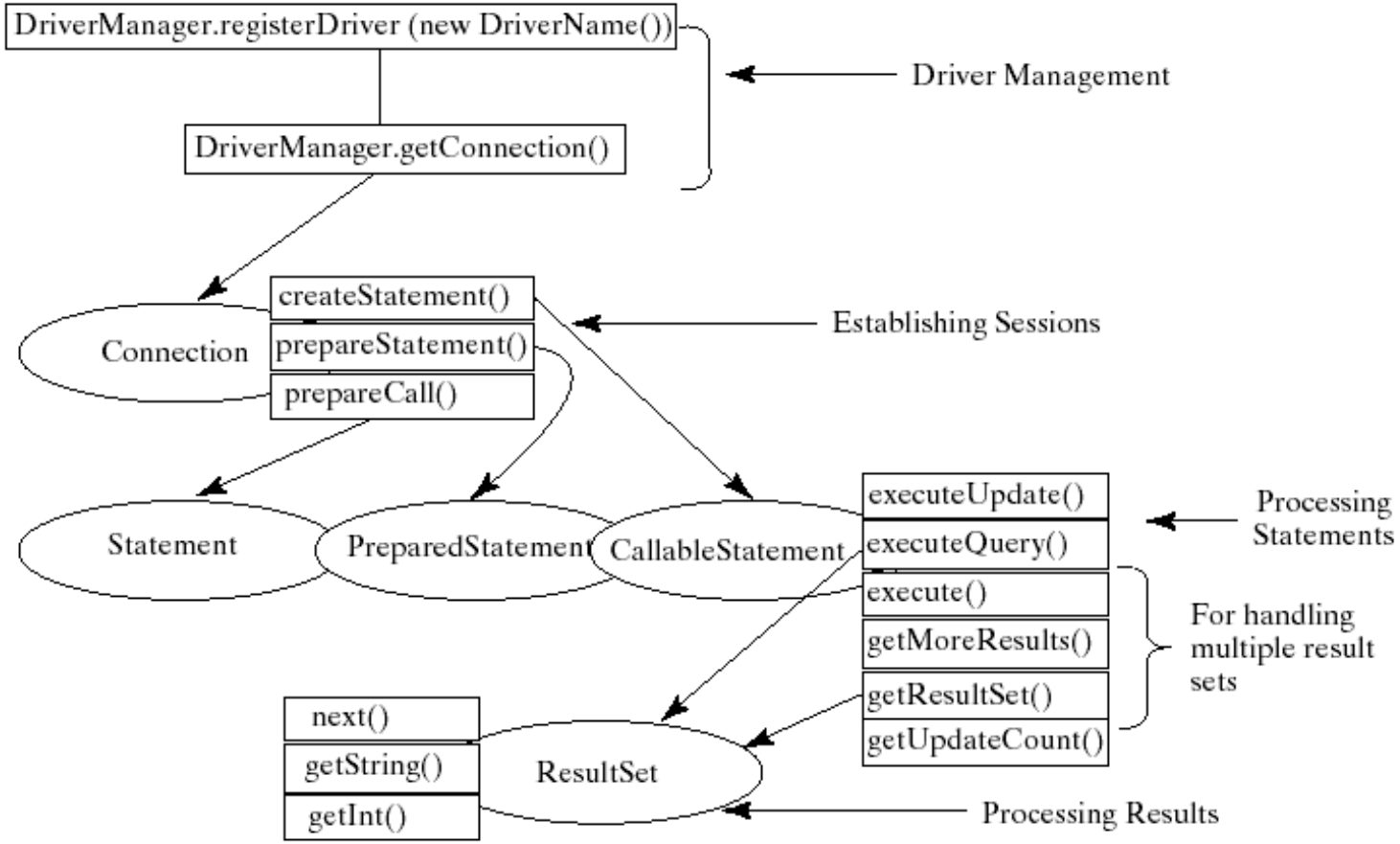
Simple JDBC Example

```
import java.sql.*;
public class SimpleJdbc {
    public static void main(String[] args)
      throws SQLException, ClassNotFoundException {
    // Load the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded");

    // Establish a connection
    Connection conn = DriverManager.getConnection
        ("jdbc:mysql://localhost/cs","root","");
    System.out.println("Database connected");

    // Create a statement
    Statement stmt = conn.createStatement();

    // Execute a statement"select std_id, std_name from Student"
    ResultSet rs = stmt.executeQuery
      ("select std_id, std_name from Student");

    // Iterate through the result and print the student id, names
    while (rs.next()){
        System.out.println(rs.getString(1) + "\t " +
          rs.getString(2));
    }
    // Close the connection
    conn.close();
  }
}
```

Processing Statements

- Once a connection to a particular database is established, it can be used to send SQL statements from your program to the database.
- JDBC provides the Statement, PreparedStatement, and CallableStatement interfaces to facilitate sending statements to a database for execution and receiving execution results from the database.

Processing Statements Diagram



The execute, executeQuery, and executeUpdate Methods

- The methods for executing SQL statements are execute, executeQuery, and executeUpdate, each of which accepts a string containing a SQL statement as an argument.
- This string is passed to the database for execution.
- The execute method should be used if the execution produces multiple result sets, multiple update counts, or a combination of result sets and update counts.
- The executeQuery method should be used if the execution produces a single result set, such as the SQL select statement.
- The executeUpdate method should be used if the statement results in a single update count or no update count, such as a SQL INSERT, DELETE, UPDATE, or DDL statement.

PreparedStatement

- The PreparedStatement interface is designed to execute dynamic SQL statements and SQL-stored procedures with IN parameters. These SQL statements and stored procedures are precompiled for efficient use when repeatedly executed.

```
Statement pstmt = connection.prepareStatement
  ("insert into Student (firstName, mi, lastName) +
    values (?, ?, ?)");
```

Handling Errors with Exceptions

- Programs should recover and leave the database in a consistent state.
- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements
- How might a finally {…} block be helpful here?
- E.g. you could rollback your transaction in a catch {…} block or close database connection and free database related resources in finally {…} block

Sample Program

```
import java.sql.*;
class TestJDBC {
   public static void main(String[] args) {
      try {
         // Load the JDBC driver
         Class.forName("com.mysql.jdbc.Driver");//dynamic loading of driver
         System.out.println("Driver loaded");

         // Establish a connection
         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/cs","root","");
         Statement s = conn.createStatement();
         s.execute("create table TEST12345 ( firstcolumn integer )");
         s.execute("insert into TEST12345 values(1)");
         s.execute("select firstcolumn from TEST12345");

         ResultSet rs = s.getResultSet();
         if (rs != null) // if rs == null, then there is no ResultSet to view
         while ( rs.next() ) // this will step through our data row by row
         {   /* the next line will get the first column in our current row's ResultSet
                as a String (getString(columnNumber)) and output it to the screen */
            System.out.println("Data from column_name: " + rs.getString(1));
         }
         s.close(); // close Statement to let the database know we're done with it
         conn.close(); //close connection
      }
      catch (Exception err) {
         System.out.println("ERROR: " + err);
      }
   }
}
```

Mapping types JDBC -Java

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT DOUBLE	double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC Type	Java Type
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	Clob*
BLOB	Blob*
ARRAY	Array*
DISTINCT	mapping of underlying type
STRUCT	Struct*
REF	Ref*
JAVA_OBJECT	underlying Java class

*SQL3 data type supported in JDBC 2.0

Retrieving Database Metadata

- Database metadata is the information that describes database itself.
- JDBC provides the DatabaseMetaData interface for obtaining database wide information and the ResultSetMetaData interface for obtaining the information on the specific ResultSet.

DatabaseMetadata, cont.

- The DatabaseMetaData interface provides more than 100 methods for getting database metadata concerning the database as a whole.

- These methods can be divided into three groups: for retrieving general information, for finding database capabilities, and for getting object descriptions.

General Information

- The general information includes the URL, username, product name, product version, driver name, driver version, available functions, available data types and so on.

Obtaining Object Descriptions

- The examples of the database objects are tables, views, and procedures.

Examples

```
DatabaseMetaData dbMetaData = conn.getMetaData();

System.out.println("database URL: " +
   dbMetaData.getURL());
System.out.println("database username: " +
  dbMetaData.getUserName());
System.out.println("database product name: " +
  dbMetaData.getDatabaseProductName());
System.out.println("database product version: " +
  dbMetaData.getDatabaseProductVersion());
System.out.println("JDBC driver name: " +
  dbMetaData.getDriverName());
System.out.println("JDBC driver version: " +
  dbMetaData.getDriverVersion());
System.out.println("JDBC driver major version: " +
  new Integer(dbMetaData.getDriverMajorVersion()));
System.out.println("JDBC driver minor version: " +
  new Integer(dbMetaData.getDriverMinorVersion()));
System.out.println("Max number of connections: " +
  new Integer(dbMetaData.getMaxConnections()));
System.out.println("MaxTableNameLentgh: " +
  new Integer(dbMetaData.getMaxTableNameLength()));
System.out.println("MaxColumnsInTable: " +
  new Integer(dbMetaData.getMaxColumnsInTable()));
conn.close();
```

Sample Run

```
database URL: jdbc:mysql://localhost/cs
database username: root@localhost
database product name: MySQL
database product version: 5.7.33
JDBC driver name: MySQL-AB JDBC Driver
JDBC driver version: mysql-connector-java-5.1.23 ( Revision: ${bzr.revision-id} )
JDBC driver major version: 5
JDBC driver minor version: 1
Max number of connections: 0
MaxTableNameLentgh: 64
MaxColumnsInTable: 512
```

JDBC references

JDBC Data Access API –JDBC Technology Homepage

- http://java.sun.com/products/jdbc/index.html

JDBC Database Access –The Java Tutorial

- http://java.sun.com/docs/books/tutorial/jdbc/index.html

JDBC Documentation

- http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/index.html

java.sql package

- http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

JDBC Technology Guide: Getting Started

- http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/GettingStarted TOC.fm.html

JDBC Tutorial

https://www.javatpoint.com/java-jdbc