# Introduction to JDBC Programming

JAVA DATABASE CONNECTIVITY

# What is a Database?

- Structured set of data
- Data is stored in the form of tables.
- Example:

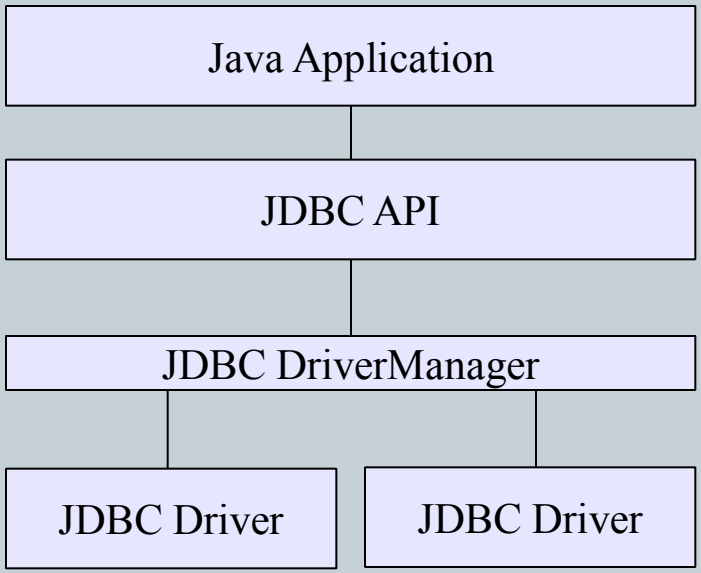
| ID | AGE | FIRST | LAST |
|---|---|---|---|
| 100 | 18 | Zara | Ali |
| 101 | 25 | Mahnaz | Fatma |
| 102 | 30 | Zaid | Khan |
| 103 | 28 | Sumit | Mittal |

# What is JDBC?

- JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- i.e Java API for connecting programs written in java to the data in relational databases.
- JDBC classes and interfaces are in java.sql package.
- JDBC API uses jdbc drivers to connect to the database.

# JDBC Architecture

- With JDBC, the Application programmer uses the JDBC API



# DBMS Commands

- To create a Database:
  - CREATE DATABASE *database_name*
    - Where *database_name* is specified by user.
- To select an existing database:
  - USE *database_name*
- To create a new table:
  - CREATE TABLE table_name( column1 datatype, column2 datatype, column3 datatype, ..... columnN datatype, PRIMARY KEY( one or more columns ) );
    - CREATE TABLE REGISTRATIO(id INTEGER , name VARCHAR(255), age INTEGER, PRIMARY KEY(id ))

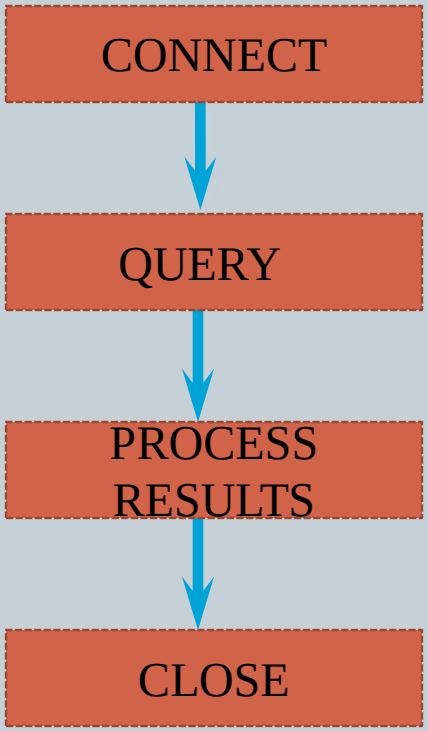
# Data Types

CHAR
VARCHAR
LONGVARCHAR
NUMERIC
DECIMAL
BIT
TINYINT
SMALLINT
INTEGER
BIGINT
REAL
FLOAT
DOUBLE
BINARY
VARBINARY
DATE
TIME
TIMESTAMP

# DBMS Commands

- To insert new rows of data to a table in the database:
  - INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)VALUES (value1, value2, value3,...valueN);
    - INSERT INTO  registration (ID,NAME,AGE) VALUES (1, 'Priya', 25);
- To fetch the data from a database table:
  - SELECT column1, column2, columnN FROM table_name;
    - SELECT * FROM table_name; //to fecth all the columns.
    - SELECT id, name FROM registration;

# Overview of Querying a Database with JDBC



# Tasks Associated with JDBC

- There are 4 major tasks:

1) Making a connection to a database
2) Creating SQL or MySQL statements
3) Executing that SQL or MySQL queries in the database
4) Viewing & Modifying the resulting records

# Steps involved to create a JDBC Application

1) Import the Packages.
2) Register the JDBC Driver
3) Open a Connection
4) Create a Statement Object
5) Execute a Query
6) Extract the data from the Result set
7) Clean up the Environment

# Step1: Import the packages

- To include the packages containing the JDBC classes needed for database programming

- import java.sql.*;  //required package

# Step2:Register the JDBC Driver

- JDBC driver is a software component that enables the java applications to interact with the database.
- When a JDBC Driver class is loaded, it must create an instance of itself and register that instance with the JDBC DriverManager.
- The forName() method of *Class* class is used to register the driver class.
- This method is used to dynamically load the driver class.

# Step2:Register the JDBC Driver

- Class.forName("com.mysql.jdbc.Driver");
  - This method will return the class object associated with mysql driver class or interface.
- Exception
  - **LinkageError** -- if the linkage fails.
  - **ExceptionInInitializerError** -- if the initialization provoked by this method fails.
  - **ClassNotFoundException** -- if the class cannot be located.

# Step3: To open a Connection(Connection Interface)

- A Connection is the session between java application and database.
- The getConnection() method of DriverManager class is used to establish connection with the database.
- Syntax of getConnection() method:
  - public static Connection getConnection(String url) throws SQLException
  - public static Connection getConnection(String url,String name,String password) throws SQLException

# Step3: To open a Connection

- This method establishes a database connection.
- It accepts a parameter specifying database URL, which varies depending upon DBMS being used, username, password.
- DriverManager.getConnection(DB_URL,USER,PASS);
- Example for MySql Driver:
  - MySQL: jdbc:mysql://localhost:3306/, where localhost is the name of the server hosting your database, and 3306 is the port number
- In the database URL, it is require to specify the name of an existing database to which connection is need to be made.

# Step3: To open a Connection

- To create a Connection object, which represents a physical connection with the database as follows:

```
static final String USER = "username";
static final String PASS = "password";
System.out.println("Connecting to database...");
conn =
   DriverManager.getConnection(DB_URL,USER,PASS);
```

# Step4: Create a Statement Object

- The object of Connection can be used to get the object of Statement and PreparedStatement.
- The createStatement() method of Connection interface is used to create statement.
- Syntax of createStatement() method:
  - public Statement createStatement()throws SQLException
- Example to create the statement object:
  - Statement stmt=conn.createStatement(); //where conn has already been defined.
- The object of statement is responsible to execute queries with the database.

# Step5:Execute a Query (Statement interface)

- The *Statement interface* provides methods to execute queries with the database.
- The *executeQuery()* method of Statement interface is used to execute queries to the database.
- This method returns the object of *ResultSet* that can be used to get all the records of a table.
- Syntax of *executeQuery()* method:
  - public ResultSet executeQuery(String sql) throws SQLException

# Step5:Execute a Query

- Using an object of type Statement or PreparedStatement for building and submitting an SQL statement to the database as follows:

```
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);
```

# Execute a query

- public int executeUpdate()
  - executes the query. It is used for create, drop, insert, update, delete etc.
- public ResultSet executeQuery()
  - executes the select query. It returns an instance of ResultSet.

# PreparedStatement Interface

- The PreparedStatement interface is a subinterface of Statement.
- It is used to execute parameterized query.
- An Example of Parameterized Query:
  - String sql="insert into emp values(?,?,?)";
    - parameter (?) is passed for the values. Its value will be set by calling the setter methods of PreparedStatement.
- Syntax to get instance of PreparedStatement:
  - public PreparedStatement prepareStatement(String query)throws SQLException{}

# PreparedStatement Interface

- Important methods of PreparedStatement interface:
  - public void setInt(int paramIndex, int value)
    - sets the integer value to the given parameter index.
  - public void setString(int paramIndex, String value)
    - sets the String value to the given parameter index.
  - public void setFloat(int paramIndex, float value)
    - sets the float value to the given parameter index.
  - public void setDouble(int paramIndex, double value)
    - sets the double value to the given parameter index.

# Example of PreparedStatement Interface to insert a record

| EMP(Name of Table) | |
|---|---|
| Identification No | Name |

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

Connection con=DriverManager.getConnection(" jdbc:mysql://localhost:3306/",
“root",“12345");

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();
System.out.println(i+" records inserted");
```

# Example of PreparedStatement Interface to update a record

| EMP(Name of Table) | |
|---|---|
| Identification No | Name |

```
PreparedStatement stmt;
stmt=con.prepareStatement("update emp set name=? where id=?");

stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,101);

int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

# ResultSet Interface

- The object of ResultSet maintains a cursor pointing to a particular row of data.
- Initially, cursor points to the first row.
- By default, ResultSet object can be moved forward only.
- Commonly used methods of ResultSet Interface:
  - public boolean next()
    - used to move the cursor to the one row next from the current position.
  - public boolean first()
    - is used to move the cursor to the first row in result set object.

# ResultSet Interface

- public boolean last()
  - used to move the cursor to the last row in result set object.
- public int getInt(int columnIndex)
  - used to return the data of specified column index of the current row as int.
- public int getInt(String columnName)
  - is used to return the data of specified column name of the current row as int.
- public String getString(int columnIndex)
  - used to return the data of specified column index of the current row as String.
- public String getString(String columnName)
  - is used to return the data of specified column name of the current row as String.
- public boolean absolute(int row)
  - used to move the cursor to the specified row number in the ResultSet object.

# Step6:Extract data from result set

- The appropriate ResultSet.getXXX() method can be used to retrieve the data fetched from the database using result set as follows:
- while(rs.next()){

```
//Retrieve by column name
int id = rs.getInt("id");
int age = rs.getInt("age");
String first = rs.getString("first");
String last = rs.getString("last");
```

# Step7:Clean up the environment

- By closing connection object statement and *ResultSet* will be closed automatically.
- The close() method of Connection interface is used to close the connection.
- Syntax of close() method:
  - public void close()throws SQLException
- Close all database resources

```
rs.close(); //closing expilicitly
stmt.close();
conn.close();
```

# JDBC Example program

```
import java.sql.*;
public class JDBCExample {

    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost:3306/";

    // Database credentials
    static final String USER = "root";
    static final String PASS = "GOOGLE";
    public static void main(String[] args)
    {
            Connection conn = null;
            Statement stmt = null;
            try{
                    //STEP 2: Register JDBC driver
                    Class.forName("com.mysql.jdbc.Driver").newInstance();
```

# Contd...

```
//STEP 3: Open a connection
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL+"recommender", USER, PASS);
if(conn!= null){
            System.out.println("Got Connection.");}
else{
            System.out.println("Could not Get Connection");}

//STEP 4: Execute a query
System.out.println("Creating database...");
stmt = conn.createStatement();
String sql;
sql= "CREATE DATABASE STUDENTS";
stmt.executeUpdate(sql);
System.out.println("Database created successfully...");
sql = "USE STUDENTS";
stmt.executeUpdate(sql);
System.out.println("Connected to Database ");
```

# Contd..

```
sql = "CREATE TABLE REGISTRATION " + "(id INTEGER not NULL, “ + "
first VARCHAR(255), " + "        last VARCHAR(255), “  + " age INTEGER, " +
" PRIMARY KEY ( id ))";
stmt.executeUpdate(sql);
System.out.println("Created table in given database...");
sql = "INSERT INTO Registration " + "VALUES (100, 'Zara', 'Ali', 18)";
stmt.executeUpdate(sql);
sql = "INSERT INTO Registration " + "VALUES (101, 'Mahnaz', 'Fatma', 25)";
stmt.executeUpdate(sql);
sql = "SELECT id, first, last, age FROM Registration";
ResultSet rs = stmt.executeQuery(sql);
//STEP 5: Extract data from result set
while(rs.next()){
            int id = rs.getInt("id");
            int age = rs.getInt("age");
            String first = rs.getString("first");
            String last = rs.getString("last"); //Display values
 }
```

# Contd...

```
            rs.close();
}
catch(SQLException se)
{
    //Handle errors for JDBC
        se.printStackTrace();
    }
```

# Output of the program

- Connecting to database...
- Got Connection.
- Creating database...
- Database created successfully...
- Connected to Database
- Created table in given database...
- Inserted records into the table...
- Created table look like

| ID | AGE | FIRST | LAST |
|---|---|---|---|
| 100 | 18 | Zara | Ali |
| 101 | 25 | Mahnaz | Fatma |
| 102 | 30 | Zaid | Khan |
| 103 | 28 | Sumit | Mittal |

# How to run the program?

- Steps to be followed:
- Project -> Properties ->Libraries
- Add jar files -> select the JDBC Driver -> Open
- Confirm that jar file is added to the file and click OK.
- Run File.

# SQLException Methods

- A SQLException can occur both in the driver and the database. When such an exception occurs, an object of type SQLException will be passed to the catch clause.

| Method | Description |
|---|---|
| getErrorCode() | Get the error number associated with the exception. |
| getMessage() | Gets the JDBC driver's error message for an error handled by the driver or gets the Oracle error number and message for a database error. |
| printStackTrace( ) | Prints the current exception, and its backtrace to a standard error stream. |
| | |

# Retrieving the Error Information

- Example for getMessage() and getErrorCode():

```
catch(SQLException e)
{
 System.out.println("exception: " +e.getErrorCode+” :”+ e.getMessage());
}
```

**OUTPUT:**
Exception 1024: Invalid column type

# Printing the Stack Trace

- To illustrate how the JDBC drivers handle errors, assume the following code uses an incorrect column index:

```
// Iterate through the result and print the employee names of the code
try {
while (rset.next ())
 System.out.println (rset.getString (5)); // incorrect column index
}
catch(SQLException e) { e.printStackTrace (); }
```

**OUTPUT:**

```
java.sql.SQLException: Invalid column index
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:112)
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:146)
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:208)
at
oracle.jdbc.driver.OracleResultSetImpl.getDate(OracleResultSetImpl.java:1556)
at Employee.main(Employee.java:41)
```

# JDBC Data Types:

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

# Assignment

Create a database named as “student” , then create a table “admission”. The attributes of admission table are <id, int>,< Name, string> , <age, int>, <Address, String> where primary key would be id attribute.

Insert the following data into this table:

| Id | Name | Age | Address |
|---|---|---|---|
| 1 | V | 20 | XYZ |
| 2 | W | 18 | XYZ |
| 3 | X | 19 | XYZ |
| 4 | Y | 22 | XYZ |
| 5 | Z | 21 | XYZ |