Unit-2

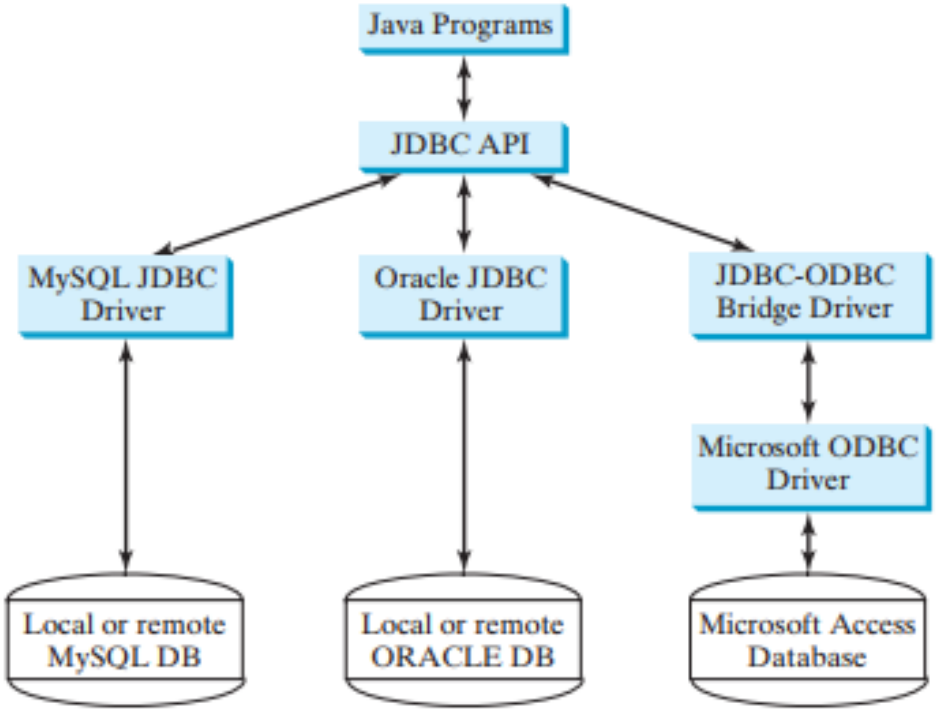
Database Programming

Asst. Prof. Roshan Tandukar

JDBC(Java Database Connectivity)

- The Java API for developing Java database applications is called JDBC.
- JDBC is the trademarked name of a Java API that supports Java programs that access relational databases.
- JDBC provides Java programmers with a uniform interface for accessing and manipulating a wide range of relational databases.
- Using the JDBC API, applications written in the Java programming language can execute SQL statements, retrieve results, present data in a user friendly interface, and propagate changes back to the database.
- The JDBC API can also be used to interact with multiple data sources in a distributed, heterogeneous environment.

JDBC(Java Database Connectivity)



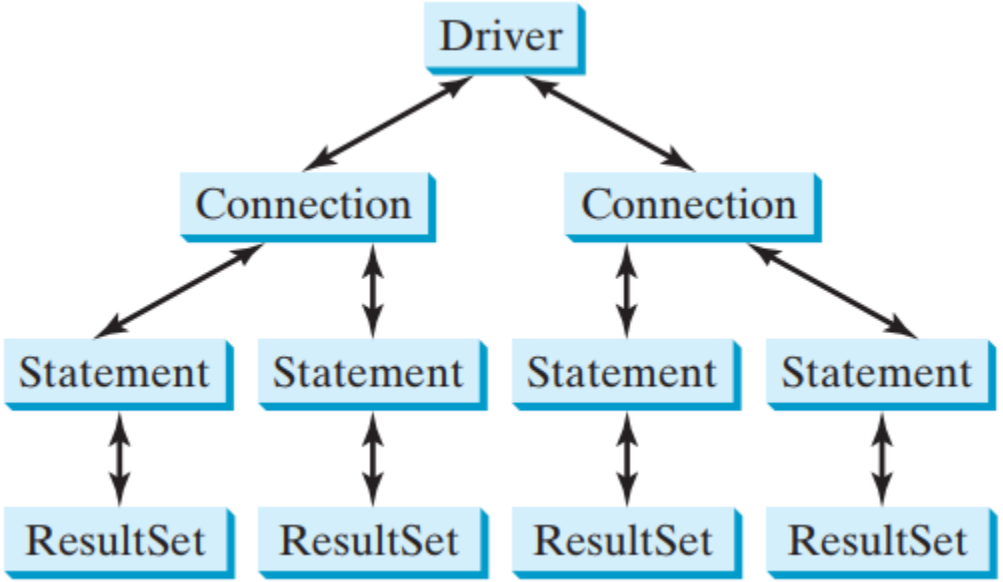
Relationships between Java programs, JDBC API, JDBC drivers, and relational databases

Developing Database Applications Using JDBC

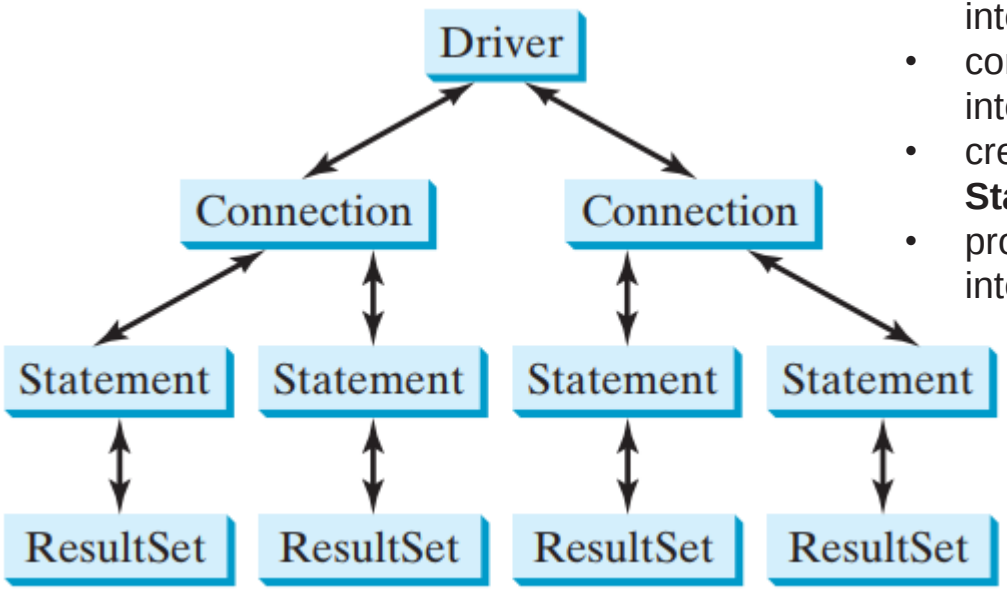
- The JDBC API is a Java application program interface to generic SQL databases that enables Java developers to develop DBMS-independent Java applications using a uniform interface.
- The JDBC API consists of classes and interfaces for establishing connections with databases, sending SQL statements to databases, processing the results of SQL statements, and obtaining database metadata.
- Four key interfaces are needed to develop any database application using Java: **Driver**, **Connection**, **Statement**, and **ResultSet**.
- These interfaces define a framework for generic SQL database access.
- The JDBC API defines these interfaces, and the JDBC driver vendors provide the implementation for the interfaces. Programmers use these interfaces.

Developing Database Applications Using JDBC

- JDBC classes enable Java programs to connect to the database, send SQL statements, and process results.



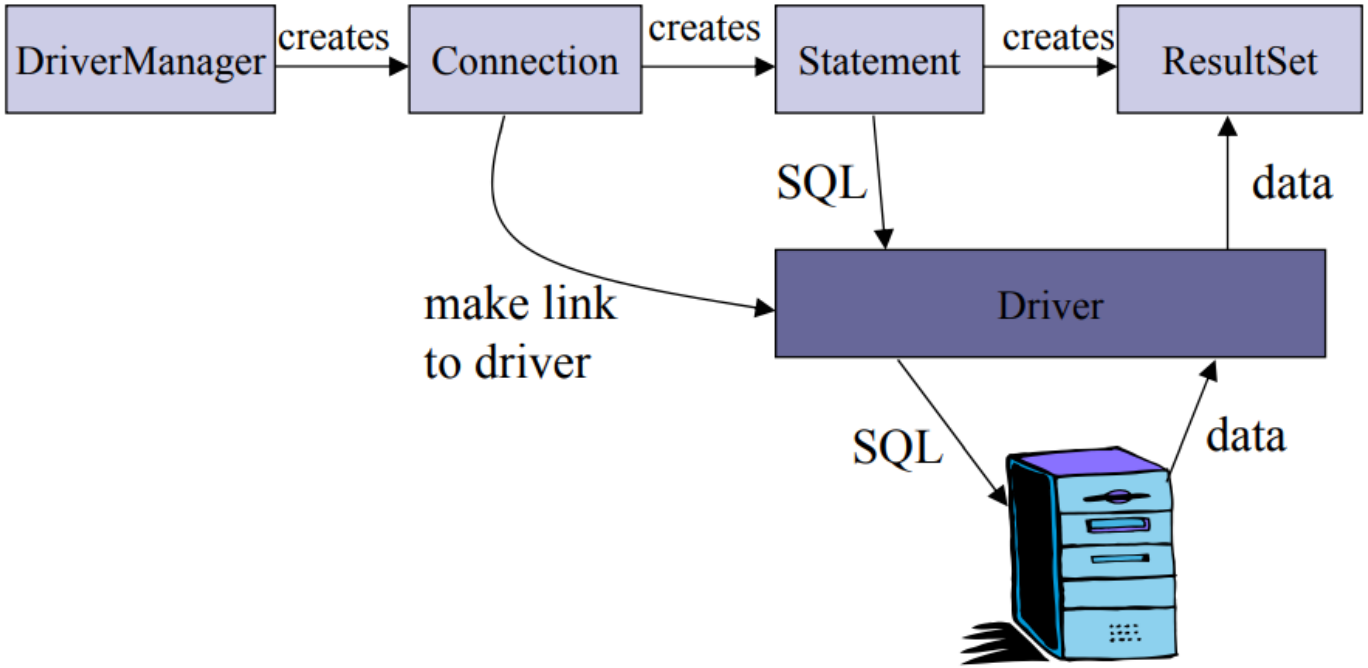
Developing Database Applications Using JDBC



A JDBC application

- loads an appropriate driver using the **Driver** interface,
- connects to the database using the **Connection** interface,
- creates and executes SQL statements using the **Statement** interface,and
- processes the result using the **ResultSet** interface if the statements return results.

Developing Database Applications Using JDBC



JDBC Configuration

JDBC configuration involves:

- Setting up a JDBC driver,
- Defining a JDBC connection, and
- Registering the JDBC resources used by the application.

Steps for connectivity between Java program and database

1. Loading the Driver

- To begin with, you first need load the driver or register it before using it in the program .
- A driver is a concrete class that implements the **java.sql.Driver** interface.
- If your program accesses several different databases, all their respective drivers must be loaded.
- Registration is to be done once in your program. You can register a driver in one of two ways mentioned below :

I. **Class.forName()**

II. **DriverManager.registerDriver()**

Note: *However, that is not required since JDBC 4.0 (JDK 6.0) because the driver manager can detect and load the driver class automatically as long as a suitable JDBC driver present in the classpath.*

Steps for connectivity between Java program and database

1. **Class.forName()**

- Here we load the driver's class file into memory at the runtime.
- And there is no need of using new or creation of object .
- The following example uses Class.forName() to load the Oracle driver –

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

2. DriverManager.registerDriver()

- DriverManager is a Java inbuilt class with a static member register.
- Here we call the constructor of the driver class at compile time .
- The following example uses DriverManager.registerDriver()to register the Oracle driver:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())
```

Steps for connectivity between Java program and database

2. **Create the connections**
   After loading the driver, establish connections using the static method **getConnection(databaseURL)** in the **DriverManager** class, as follows:

```
Connection connection = DriverManager.getConnection(databaseURL);
```

where **databaseURL** is the unique identifier of the database on the Internet.

OR,

```
Connection con = DriverManager.getConnection(url, user, password)
```

Where, user – username from which your sql command prompt can be accessed.

password – password from which your sql command prompt can be accessed.

Steps for connectivity between Java program and database

2. Create the connections

RDBMS	Database URL format
MySQL	`jdbc:mysql://`*hostname*`:`*portNumber*`/`*databaseName*
ORACLE	`jdbc:oracle:thin:@`*hostname*`:`*portNumber*`:`*databaseName*
DB2	`jdbc:db2:`*hostname*`:`*portNumber*`/`*databaseName*
PostgreSQL	`jdbc:postgresql://`*hostname*`:`*portNumber*`/`*databaseName*
Java DB/Apache Derby	`jdbc:derby:`*dataBaseName* (embedded) `jdbc:derby://`*hostname*`:`*portNumber*`/`*databaseName* (network)
Microsoft SQL Server	`jdbc:sqlserver://`*hostname*`:`*portNumber*`;databaseName=`*dataBaseName*
Sybase	`jdbc:sybase:Tds:`*hostname*`:`*portNumber*`/`*databaseName*

Popular JDBC database URL formats

Steps for connectivity between Java program and database

3. **Create a statement**

- Once a Connection object is created, you can create statements for executing SQL statements as follows:

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

- The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.

Steps for connectivity between Java program and database

4. **Execute the query**

- Now comes the most important part i.e executing the query.
- Query here is an SQL Query and we can have multiple types of queries.
- Some of them are as follows:
    - Query for updating / inserting table in a database.
    - Query for retrieving data .

Steps for connectivity between Java program and database

- SQL data definition language (DDL) and update statements can be executed using **executeUpdate(String sql).**
- An SQL query statement can be executed using **executeQuery(String sql).**
- The result of the query is returned in **ResultSet**.
- For example, the following code executes the SQL statement create table Student (Rollno int, Name char(25)):

```
String sql="create table Student (Rollno int, Name char(25))"
st.executeUpdate(sql);
```

Steps for connectivity between Java program and database

- SQL data definition language (DDL) and update statements can be executed using **executeUpdate(String sql).**
- An SQL query statement can be executed using **executeQuery(String sql).**
- The result of the query is returned in **ResultSet**.
- To execute the SQL query select firstName, mi, lastName from Student where lastName = 'Smith':

```
String sql="select firstName, mi, lastName from Student where lastName = 'Smith'";

ResultSet resultSet = statement.executeQuery(sql);
```

Steps for connectivity between Java program and database

4. Processing **ResultSet**.

- The **ResultSet** maintains a table whose current row can be retrieved.
- The initial row position is **null**.
- You can use the **next** method to move to the next row and the various get methods to retrieve values from a current row.
- For example, the following code displays all the results from the preceding SQL query.

Steps for connectivity between Java program and database

4. Processing **ResultSet**.

- For example, the following code displays all the results from the preceding SQL query.

```
while (resultSet.next())
        System.out.println(resultSet.getString(1) + " " + resultSet.getString(2) + " " +
        resultSet.getString(3));
```

- The getString(1), getString(2), and getString(3) methods retrieve the column values for firstName, mi, and lastName, respectively.
- Alternatively, you can use getString("firstName"), getString("mi"), and getString("lastName") to retrieve the same three column values.
- The first execution of the next() method sets the current row to the first row in the result set, and subsequent invocations of the next() method set the current row to the second row, third row, and so on, to the last row.

Steps for connectivity between Java program and database

5. **Close the connections**

- So finally we have sent the data to the specified location and now we are at the verge of completion of our task .
- By closing connection, objects of Statement and ResultSet will be closed automatically.
- The close() method of Connection interface is used to close the connection.

Example:

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

Table creation using JDBC in MS SQL Server

```
import java.sql.*;
public class CreateProductTable{
  public static void main(String[ ] args){
     try{
       Class.forName( "com.microsoft.sqlserver.jdbc.SQLServerDriver" );
       String DB_URL=
       "jdbc:sqlserver://<SQLServername>\\SQLEXPRESS:1433;databaseName=BCA;integratedSecurity=true;";

       Connection con = DriverManager.getConnection( DB_URL );
       System.out.println("Database connected...");
       Statement statement = con.createStatement();
       String createProductTable = "CREATE TABLE PRODUCT1 (ID VARCHAR(10),NAME VARCHAR(30),PRICE
FLOAT,PRIMARY KEY(ID));";
       statement.executeUpdate( createProductTable );
     }
     catch( Exception e ){
       e.printStackTrace();
     }
  }
}
```

Data Insertion in a Table using JDBC in MS SQL Server

```
import java.sql.*;
public class InsertProduct{
  public static void main(String[ ] args){
    try{
        Class.forName( "com.microsoft.sqlserver.jdbc.SQLServerDriver" );
        String DB_URL=
        "jdbc:sqlserver://<SQLServername>\\SQLEXPRESS:1433;databaseName=BCA;integratedSecurity=true;";

       Connection con = DriverManager.getConnection( DB_URL);
        System.out.println("Database connected...");
        Statement statement = con.createStatement();
        String createProductTable = "INSERT INTO PRODUCT1 VALUES('A101','Shirt',450);";
        statement.executeUpdate( createProductTable );
     }
    catch( Exception e ){
       e.printStackTrace();
    }
  }
}
```

SELECT Query in a table using JDBC in MS SQL Server

```
import java.sql.*;
public class ViewProduct{
public static void main(String[ ] args){
    try{
        Class.forName( "com.microsoft.sqlserver.jdbc.SQLServerDriver" );
         String DB_URL=
            "jdbc:sqlserver://<SQLServername>\\SQLEXPRESS:1433;databaseName=BCA;integratedSecurity=true;";
        Connection con = DriverManager.getConnection( DB_URL);
        Statement statement = con.createStatement();
        String sql = "SELECT * FROM PRODUCT1;";
         ResultSet rs=statement.executeQuery(sql);
        String id, name, price;
        while(rs.next()){
           id =rs.getString("ID");
           name=rs.getString("NAME");
           price =rs.getString("PRICE");
           System.out.println(id+"\t"+name+"\t"+price);
        }
    } catch( Exception e ){        e.printStackTrace();  }
  }
}
```

ResultSet Object

- ResultSet interface is defined in the java.sql package as java.sql.ResultSet
- It encapsulates the results of a SQL query(Stores the results of a SQL query)
- A ResultSet object is similar to a 'table' of answers, which can be examined by moving a 'pointer' (cursor)

<u>Accessing a ResultSet</u>

Cursor operations: first(), last(), next(), previous(), etc.

cursor

23	John
5	Mark
17	Paul
98	Peter

Example:

```
while( rs.next() ) {
    // process the row;
}
```

Accessing a ResultSet

- The ResultSet class contains many methods for accessing the value of a column of the current row can use the column name or position
  eg. get the value in the lastName column:
  rs.getString("lastName")
  OR rs.getString(2)

- As the values obtained through the ResultSet are SQL data, these must be converted to Java types/objects.
- There are many methods for accessing/converting the data:
  getString(), getDate(), getInt(), getFloat(), getObject()

Meta Data

- Meta data is the information about the database: e.g. the number of columns, the types of the columns
- It is the schema information

ID	Name	Course	Mark
007	James Bond	Shooting	99
008	Aj. Andrew	Kung Fu	1

meta data ←

- The numbers, types, and properties of a ResultSet object's columns are provided by the ResulSetMetaData object returned by the ResultSet.getMetaData method
- The getMetaData() method can be used on a ResultSet object to create its meta data object.

```
ResultSetMetaData md = rs.getMetaData();
```

Meta Data

```
ResultSetMetaData metaData=rs.getMetaData();
        int numCol=metaData.getColumnCount();

        for(int i=1;i<=numCol;i++)
           System.out.print(metaData.getColumnName(i)+"\t");
        System.out.println();
```

```
import java.sql.*;
public class ViewProduct{
public static void main(String[ ] args){
        Class.forName( "com.microsoft.sqlserver.jdbc.SQLServerDriver" );
         String DB_URL=
"jdbc:sqlserver://<SQLServername>\\SQLEXPRESS:1433;databaseName=BCA;integratedSecurity=true;";
        Connection con = DriverManager.getConnection( DB_URL);
        Statement statement = con.createStatement();
        String sql = "SELECT * FROM PRODUCT1;";
        ResultSet rs=statement.executeQuery(sql);

        ResultSetMetaData metaData=rs.getMetaData();
        int numCol=metaData.getColumnCount();
        for(int i=1;i<=numCol;i++)
                System.out.print(metaData.getColumnName(i)+"\t");
        System.out.println();

        while(rs.next()){
                System.out.println(rs.getString(1) + " " + rs.getString(2) + " " +rs.getString(3));
        }
}
```

Result set

- A default ResultSet object is not updatable and has a cursor that moves forward only.
- Thus, you can iterate through it only once and only from the first row to the last row.
- It is possible to produce ResultSet objects that are scrollable and/or updatable.

Result set types

1. **Forward-only**

- Identified by java.sql.ResultSet.TYPE_FORWARD_ONLY
- It allow you to move forward, but not backward, through the data
- The application can move forward using the next() method
- In general, most of the applications work with forward-only result sets.

2. **Scroll-insensitive**

- Identified by java.sql.ResultSet.TYPE_SCROLL_INSENSITIVE
- It ignores changes that are made while it is open
- It provides a static view of the underlying data it contains.
- The membership, order, and column values of rows are fixed when the result set is created.

result set types

3. Scroll sensitive

- Identified by java.sql.ResultSet.TYPE_SCROLL_SENSITIVE
- It provides a dynamic view of the underlying data, reflecting changes that are made while it is open
- The membership and ordering of rows in the result set may be fixed, depending on how it is implemented.

result set types

```
Statement stmt = conn.createStatement(
                        ResultSet.TYPE_SCROLL_INSENSITIVE,
                        ResultSet.CONCUR_UPDATABLE);

ResultSet rs = stmt.executeQuery("SELECT * FROM TABLE");
        // rs will be scrollable, will not show changes made by others and will be
        updatable
```

Setting a ResultSet Type

Method	Description
Statement createStatement()	Creates a Statement object for sending SQL statements to the database. Result sets created using the returned Statement object will by default be type TYPE_FORWARD_ONLY and have a concurrency level of CONCUR_READ_ONLY.
Statement createStatement (int resultSetType, int resultSetConcurrency)	Creates a Statement object that will generate ResultSet objects with the given type and concurrency
Statement createStatement (int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a Statement object that will generate ResultSet objects with the given type, concurrency, and holdability

Setting a ResultSet Type

- In the similar way like Statement object, a CallableStatement object can be used for calling database stored procedures and a PreparedStatement object can be used for sending parameterized SQL statements to the database.

```
CallableStatement prepareCall(String sql,
int resultSetType,
int resultSetConcurrency,
int resultSetHoldability)

PreparedStatement prepareStatement(String sql,
int resultSetType,
int resultSetConcurrency,
int resultSetHoldability)
```

Move the Cursor in a Scrollable ResultSet

Method	Semantics
`first()`	Moves to the first record
`last()`	Moves to the last record
`next()`	Moves to the next record
`previous()`	Moves to the previous record
`beforeFirst()`	Moves to immediately before the first record
`afterLast()`	Moves to immediately after the last record
`absolute(int)`	Moves to an absolute row number, and takes a positive or negative argument
`relative(int)`	Moves backward or forward a specified number of rows, and takes a positive or negative argument

cursor

23	John
5	Mark
17	Paul
98	Peter

ResultSet Concurrency

- Concurrency mode of a resultset refers to the ability to modify the data returned by a result set
- Result sets have one of two concurrency types.

1. **Read-only**

- Used, if application does not need to modify data
- Specifying java.sql.ResultSet.CONCUR_READ_ONLY for the concurrency mode parameter will cause the statement to create result sets that are read-only

2. **Updatable**

- Used to allow the application to make changes to data in the result set
- Specifying java.sql.ResultSet.CONCUR_UPDATABLE for the concurrency mode parameter creates updatable result set

ResultSet Concurrency

```
stmt = conn.createStatement(
            ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
rs = stmt.executeQuery("SELECT * FROM my_table");
rs.first();                 // Move cursor to the row to update
rs.updateString("column_1", "new data"); // Update the value of column column_1
on that row
rs.updateRow(); // Update the row; if autocommit is enabled, update is committed
```

ResultSet Holdability

- JDBC 3.0 adds support for specifying result set (or cursor) holdability
- Result set holdability is the ability to specify whether cursors (or a result set such as java.sql.ResultSet) should be held open or closed at the end of a transaction
- A holdable cursor, or result set, is one that does not automatically close when the transaction that contains the cursor is committed.
- You may improve database performance by including the ResultSet holdability
- If ResultSet objects are closed when a commit operation is implicitly or explicitly called, this can also improve performance

ResultSet Holdability

1. **java.sql.ResultSet.HOLD_CURSORS_OVER_COMMIT**

- The constant indicating that ResultSet objects should not be closed when the method Connection.commit is called
- ResultSet objects (cursors) are not closed; they are held open when the method commit is called.

2. **java.sql.ResultSet.CLOSE_CURSORS_AT_COMMIT**

- The constant indicating that ResultSet objects should be closed when the method Connection.commit is called
- ResultSet objects (cursors) are closed when the method commit is called
- Closing cursors at commit can result in better performance for some applications

RowSet interface

- **RowSet interface** configures the database connection and prepares query statements automatically
- It provides several set methods that allow you to specify the properties needed to establish a connection (such as the database URL, user name and password of the database) and create a Statement (such as a query)
- It also provides several get methods that return these properties

RowSet interface

- There are two types of RowSet objects: connected and disconnected.

1. **Connected RowSets**

- A **connected RowSet** object connects to the database once and remains connected while the object is in use

2. **Disconnected RowSets**

- A **disconnected RowSet** object connects to the database, executes a query to retrieve the data from the database and then closes the connection
- A program may change the data in a disconnected RowSet while it's disconnected.
- Modified data can be updated in the database after a disconnected RowSet reestablishes the connection with the database.

RowSet interface

RowSets are classified into five categories based on how they are implemented which are listed namely as below:

1. JdbcRowSet: A JDBCRowSet is a RowSet that wraps around a ResultSet object.
2. CachedRowSet: A CachedRowSet is a RowSet in which the rows are cached and the RowSet is disconnected(i.e no active connection to the database)
3. WebRowSet: A WebRowSet is an extension to CachedRowSet.
4. FilteredRowSet: A FilteredRowSet is an extension to WebRowSet that provides programmatic support for filtering its content.
5. JoinRowSet: A JoinRowSet is an extension to WebRowSet that consists of related data from different RowSets.

Example

```
import java.sql.*;
import javax.sql.rowset.*;
public class RowSetExample{
   public static void main(String[ ] args){
            Class.forName( "com.microsoft.sqlserver.jdbc.SQLServerDriver" );
            String DB_URL="jdbc:sqlserver://<ServerName>\\SQLEXPRESS2;
            databaseName=BCA;integratedSecurity=true;";
            JdbcRowSet rs=RowSetProvider.newFactory().createJdbcRowSet();
            rs.setUrl(DB_URL);
            rs.setCommand("select * from PRODUCT;");
            rs.execute();
             while(rs.next()){
                 System.out.println(rs.getString(1)+"]t"+rs.getString(2)+"\t"+rs.getString(3));
            rs.close();;
   }
}
```

Steps for connectivity between Java program and database

```
import java.sql.*;
public class ViewProduct{
  static final String
DB_URL="jdbc:sqlserver://COOLDUDE\\SQLEXPRESS2:1433;databaseName=BCA;int
egratedSecurity=true;";

  public static void main(String[ ] args){
    // static final String
DB_URL="jdbc:sqlserver://COOLDUDE\\SQLEXPRESS2:1433;databaseName=BCA;int
egratedSecurity=true;";

    try{
```