CHAPTER 1 : JAVA DATABASE CONNECTIVITY

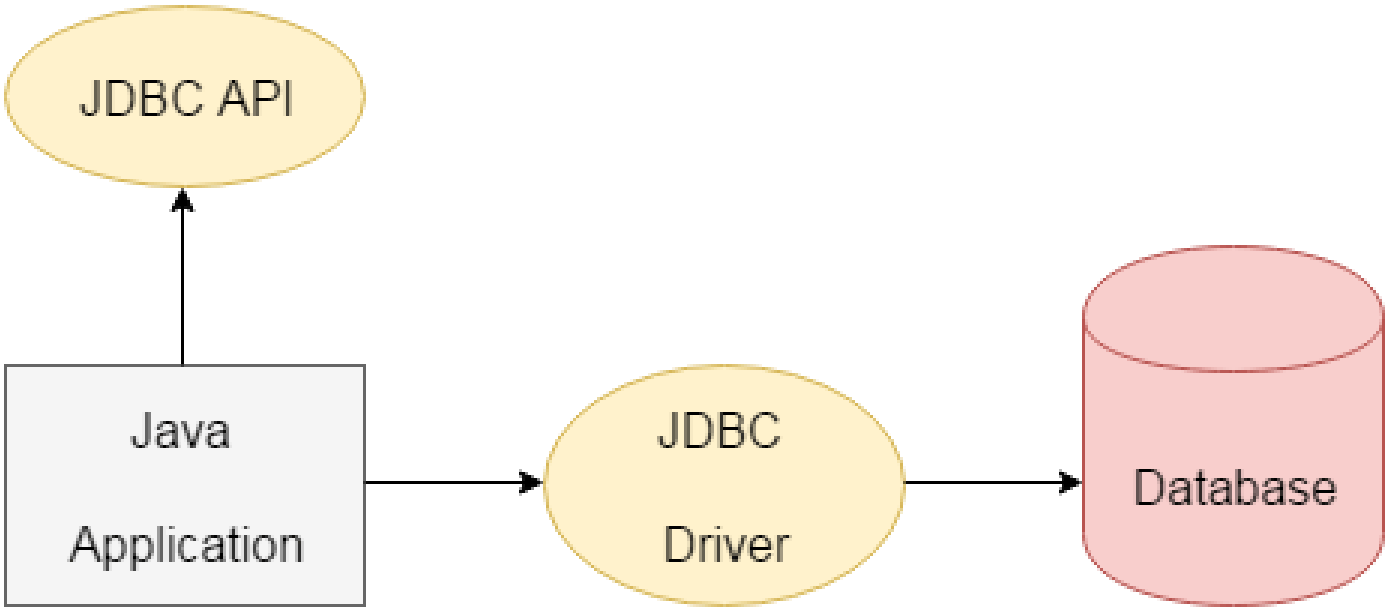
- Introduction to JDBC
- JDBC Architecture
- Common JDBC Components
- JDBC Driver
- Java Database Connectivity

Prepared By:
Prof. Jigar Bhawsar

Introduction to JDBC

- stands for Java Database Connectivity
- to connect and execute the query with the database
- uses JDBC drivers to connect with the database
- There are four types of JDBC drivers:
  - ✔ JDBC-ODBC Bridge Driver,
  - ✔ Native Driver,
  - ✔ Network Protocol Driver, and
  - ✔ Thin Driver

Continue…



Continue..

- The **java.sql** package contains classes and interfaces for JDBC API
- A list of popular *interfaces* of JDBC API are given below:
  - ✔ Driver interface
  - ✔ Connection interface
  - ✔ Statement interface
  - ✔ PreparedStatement interface
  - ✔ CallableStatement interface
  - ✔ ResultSet interface
  - ✔ ResultSetMetaData interface
  - ✔ DatabaseMetaData interface
  - ✔ RowSet interface

A list of popular *classes* of JDBC API are given below:

- ✔ DriverManager class
- ✔ Blob class
- ✔ Clob class
- ✔ Types class

Why Should We Use JDBC

- Before JDBC, ODBC API was the database API to connect and execute the query with the database
- ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured)
- That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language)

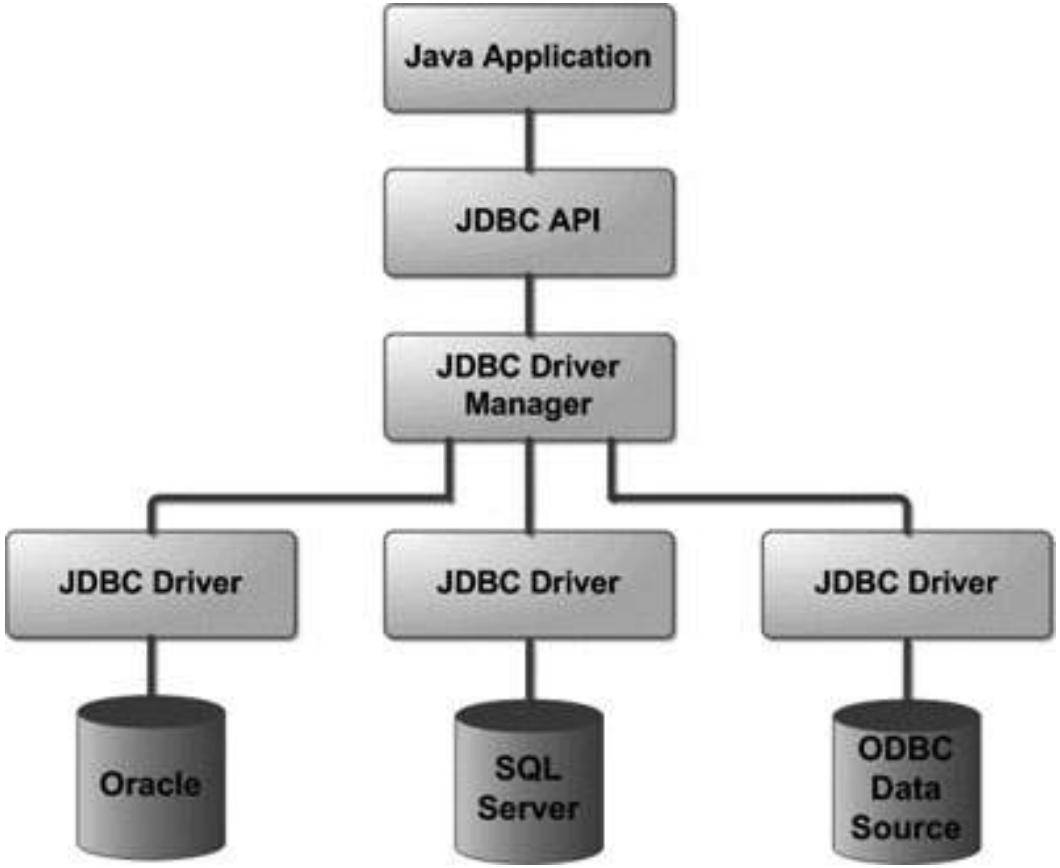
Continue…

- We can use JDBC API to handle database using Java program and can perform the following activities:
  1. Connect to the database
  2. Execute queries and update statements to the database
  3. Retrieve the result received from the database.

JDBC Architecture

- JDBC API supports both two-tier and three-tier processing models
- JDBC Architecture consists of two layers −
  - ✔ **JDBC API:** This provides the application-to-JDBC Manager connection.
  - ✔ **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
- uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- JDBC driver manager ensures that the correct driver is used to access each data source.
- The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Continue…


Java Application
JDBC API
JDBC Driver Manager
JDBC Driver
JDBC Driver
JDBC Driver
Oracle
SQL Server
ODBC Data Source

Common JDBC Components

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

Continue..

- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

JDBC Driver

- software component that enables java application to interact with the database
- There are 4 types of JDBC drivers:
  - ✔ JDBC-ODBC bridge driver
  - ✔ Native-API driver (partially java driver)
  - ✔ Network Protocol driver (fully java driver)
  - ✔ Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

- uses ODBC driver to connect to the database
- converts JDBC method calls into the ODBC function calls

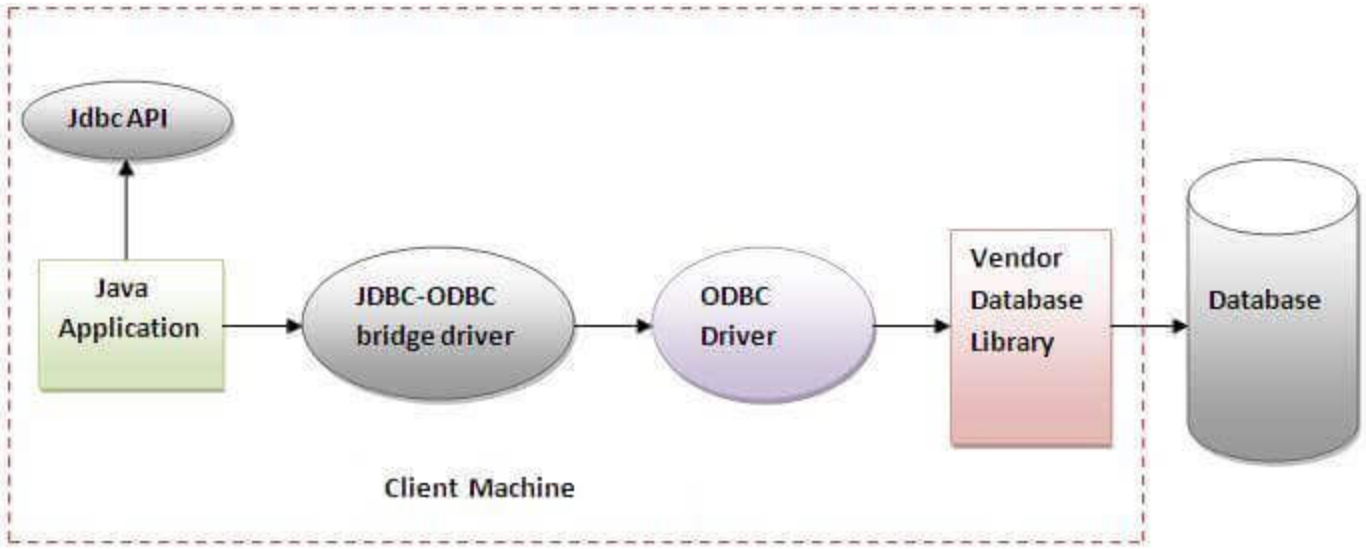


Figure- JDBC-ODBC Bridge Driver

Continue..

- **Advantages:**
  - ✔ easy to use.
  - ✔ can be easily connected to any database.

- **Disadvantages:**
  - ✔ Performance degraded because JDBC method call is converted into the ODBC function calls.
  - ✔ The ODBC driver needs to be installed on the client machine.

2) Native-API driver

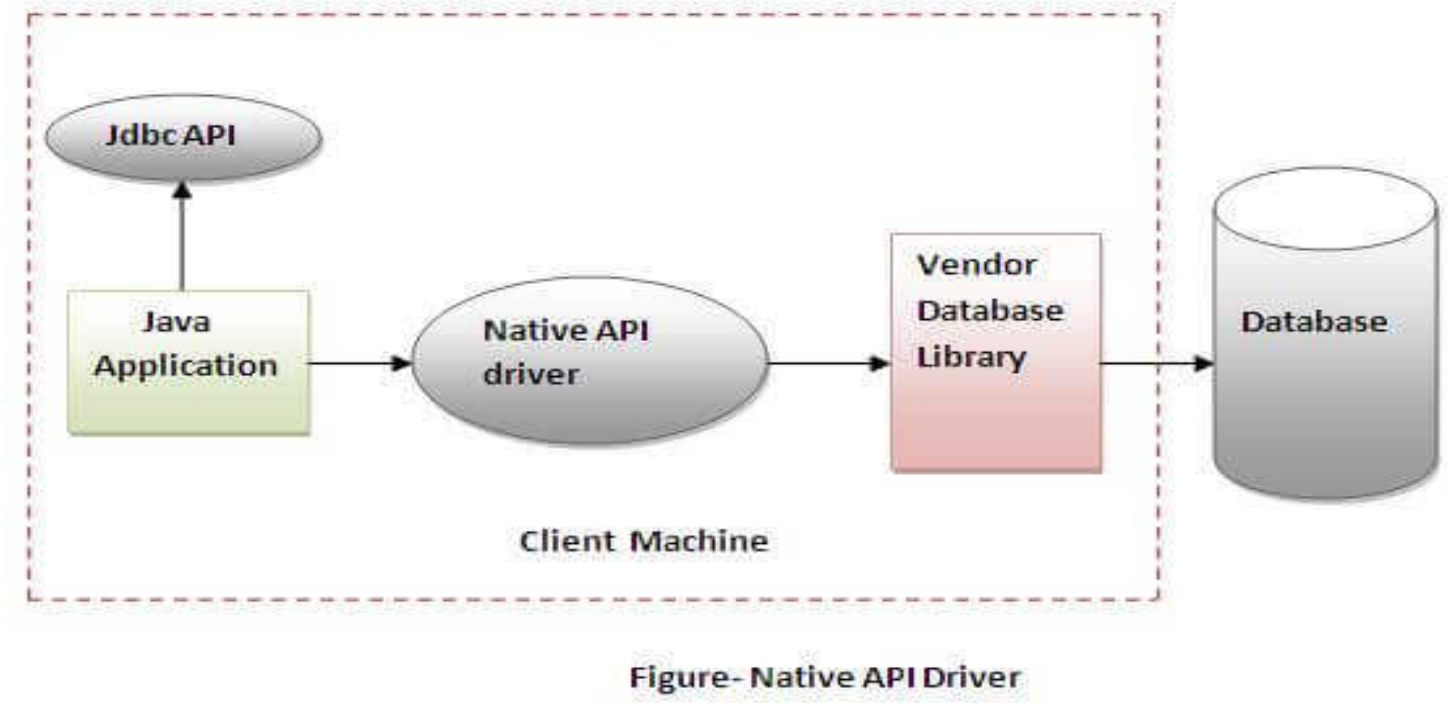


Figure- Native API Driver

Native API driver uses the client-side libraries of the database The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java

- Advantage:
  - ✔ performance upgraded than JDBC-ODBC bridge driver.
- Disadvantage:
  - ✔ The Native driver needs to be installed on the each client machine.
  - ✔ The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

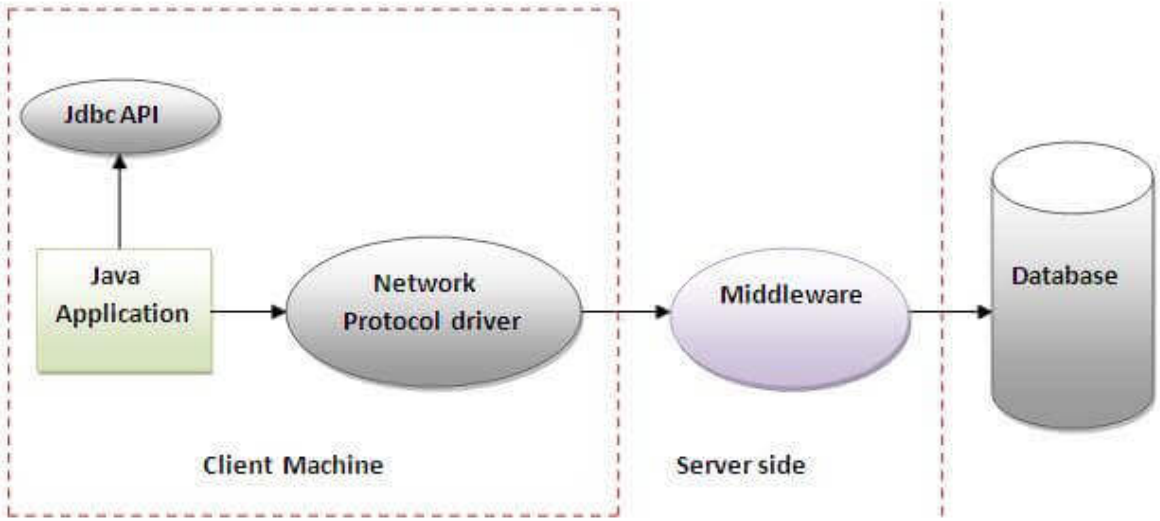


Figure- Network Protocol Driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

Continue..

- Advantage:
  - ✔ No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Disadvantages:
  - ✔ Network support is required on client machine.
  - ✔ Requires database-specific coding to be done in the middle tier.
  - ✔ Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

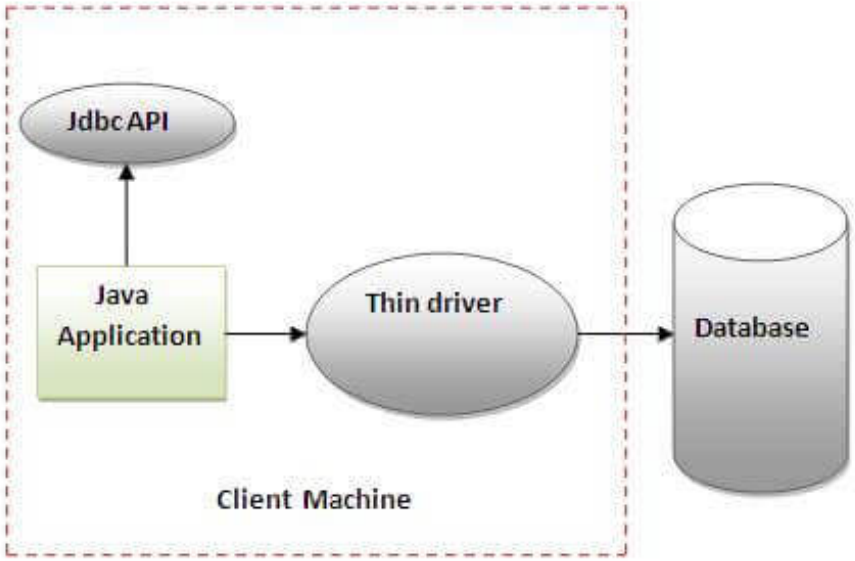


Figure- Thin Driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Continue..

- Advantage:
  - ✔ Better performance than all other drivers.
  - ✔ No software is required at client side or server side.
- Disadvantage:
  - ✔ Drivers depend on the Database.

Java Database Connectivity

1) Register the driver class

✔ The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax

public static void forName(String className)**throws** ClassNotFoun dException

Example

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

2) Create the connection object

- **getConnection()** method of DriverManager class is used to establish connection with the database

Syntax

1) **public static** Connection getConnection(String url)**throws** SQLException

2)**public static** Connection getConnection(String url,String name, String password) **throws** SQLException

Example

```
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

3) Create the Statement object

- ✔ The createStatement() method of Connection interface is used to create statement
- ✔ responsible to execute queries with the database.

Syntax

public Statement createStatement()**throws** SQLException

Example

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

4) Execute the query

- ✔ 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

public ResultSet executeQuery(String sql)**throws** SQLException

Example

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
    }
```

5) Close the connection object

- ✔ used to close the connection.

Syntax

public void close()**throws** SQLException

Example

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

Resultset metadata

- The metadata means data about data i.e. we can get further information from the data.
- If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object

ResultSet Metadata Interface

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index.

Precompiled statements

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

- String sql="insert into emp values(?,?,?)";
- Why use PreparedStatement?
- **Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

- **How to get the instance of PreparedStatement?**
- The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

- **public** PreparedStatement prepareStatement(String quer y)**throws** SQLException{}

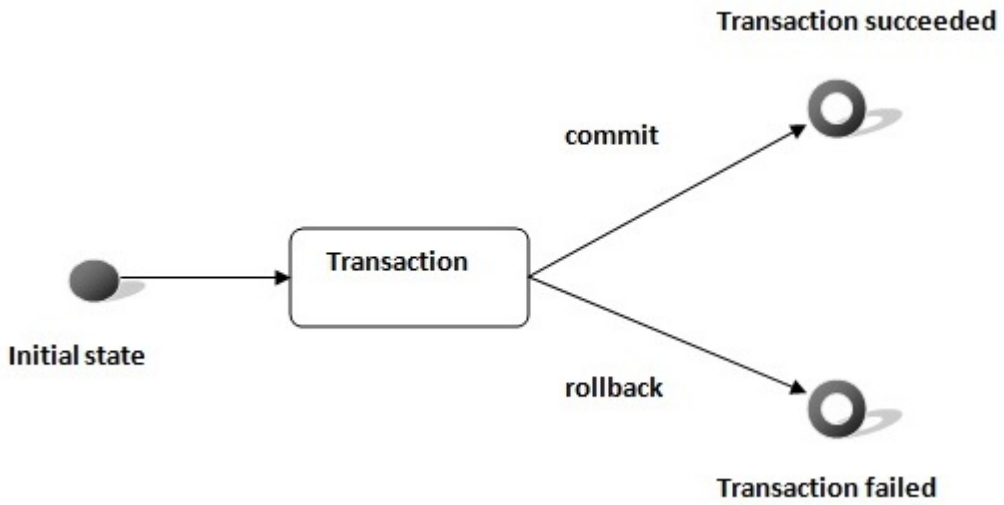
Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
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.

Performing CRUD operations through JDBC API

- Create
- Retrieve
- Update
- Delete

JDBC transactions

- Transaction represents **a single unit of work**.
- The ACID properties describes the transaction management well.
- ACID stands for Atomicity, Consistency, isolation and durability.


Transaction succeeded
commit
Transaction
Initial state
rollback
Transaction failed

Method	Description
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.