

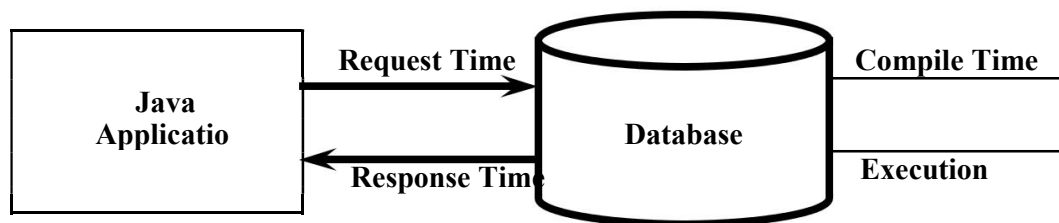
PreparedStatement

(I)

Need of PreparedStatement:

In the case of normal Statement, whenever we are executing SQL Query, every time compilation and execution will be happened at database side.

```
Statement st = con.createStatement();  
ResultSet rs = st.executeQuery ("select * from employees");
```



Total Time per Query = Req.T+C.T+E.T+Resp.T
1 ms + 1 ms + 1 ms + 1 ms = 4ms
per 1000 Queries = 4 * 1000ms = 4000ms

Sometimes in our application, we required to execute same query multiple times with same or different input values.

Eg1:

In Bus Ticking application, it is common requirement to list out all possible trains between 2 places

```
select * from bus where source='XXX' and destination='YYY';
```

Query is same but source and destination places may be different. This query is required to execute lakhs of times per day.

Eg2:

In BookMyShow application, it is very common requirement to display theatre names where a particular movie running/playing in a particular city

```
select * from theatres where city='XXX' and movie='YYY';
```

In this case this query is required to execute lakhs of times per day. May be with different movie names and different locations.

For the above requirements if we use Statement object, then the query is required to compile and execute every time, which creates performance problems.

To overcome this problem, we should go for PreparedStatement.

The main advantage of PreparedStatement is the query will be compiled only once even though we are executing multiple times, so that overall performance of the application will be improved.

We can create PreparedStatement by using prepareStatement() method of Connection interface.

```
public PreparedStatement prepareStatement(String sqlQuery) throws SQLException
```

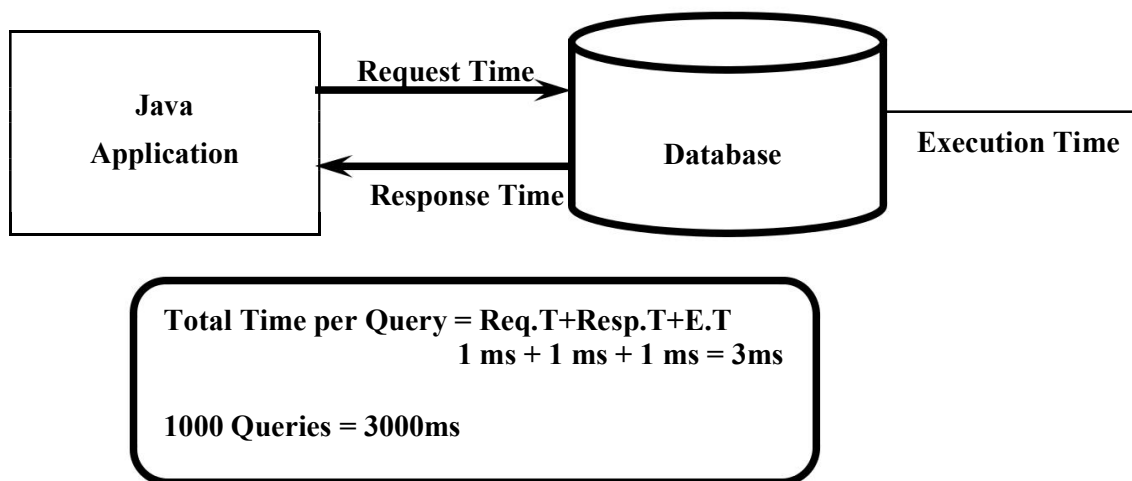
Eg: `PreparedStatement pst=con.prepareStatement(sqlQuery);`

At this line,sqlQuery will send to the database. Database engine will compile that query and stores in the database.

That pre compiled query will be returned to the java application in the form of PreparedStatement object.

Hence PreparedStatement represents "pre compiled sql query".

Whenever we call execute methods,database engine won't compile query once again and it will directly execute that query,so that overall performance will be improved.



Steps to develop JDBC Application by using PreparedStatement

Prepare SQLQuery either with parameters or without parameters. Eg: insert into employees

values(100,'adam',1000,'hyd');

insert into employees values(?, ?, ?, ?);

↓
Positional Parameter OR Place Holder OR IN Parameter

Create PreparedStatement object with our sql query.
PreparedStatement pst =
con.prepareStatement(sqlQuery);

At this line only query will be compiled.

3.If the query is parameterized query then we have to set input values to these parameters by using corresponding setter methods.

We have to consider these positional parameters from left to right and these are 1 index based. i.e index of first positional parameter is 1 but not zero.

```
pst.setInt(1,100);  
pst.setString(2,"adam");  
pst.setDouble(3,1000);  
pst.setString(4,"Hyd");
```

Note:

Before executing the query, for every positional parameter we have to provide input values otherwise we will get SQLException

Execute SQL Query:

PreparedStatement is the child interface of Statement and hence all methods of Statement interface are by default available to the PreparedStatement. Hence we can use same methods to execute sql query.

```
executeQuery()  
executeUpdate()  
execute()
```

Note:

We can execute same parameterized query multiple times with different sets of input values. In this case query will be compiled only once and we can execute multiple times.

Note:

We can use ? only in the place of input values and we cannot use in the place of sql keywords, table names and column names.

Static Query vs Dynamic Query:

The sql query without positional parameter(?) is called static query.

Eg: delete from employees where ename='adam'

The sql query with positional parameter(?) is called dynamic query.

Eg: select * from employees where esal>?

Program-1 to Demonstrate PreparedStatement:

```
import java.sql.*;
public class PreparedStatementDemo1
{
    public static void main(String[] args) throws Exception
    {
        String driver="com.mysql.jdbc.Driver";
        Class.forName(driver);
        String jdbc_url="jdbc:mysql://localhost:3306";
        String user="root";
        String pwd="root";
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        String sqlQuery = "delete from employees where ename=?";
13)      PreparedStatement pst = con.prepareStatement(sqlQuery);
        pst.setString(1,"Mallika");
        int updateCount=pst.executeUpdate();
        Root.out.println("The number of rows deleted :"+updateCount);

        Root.out.println("Reusing PreparedStatement to delete one more record...");
        pst.setString(1,"Adam");
        int updateCount1=pst.executeUpdate();
        Root.out.println("The number of rows deleted :"+updateCount1);
        con.close();
    }
}
```

```
}  
}
```

Program-2 to Demonstrate PreparedStatement:

```
import java.sql.*;  
import java.util.*;  
public class PreparedStatementDemo2  
{  
    public static void main(String[] args) throws Exception  
    {  
        String driver="com.mysql.jdbc.Driver";  
        String jdbc_url="jdbc:mysql://localhost:3306";  
        String user="root";  
        String pwd="root";  
        Class.forName(driver);  
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);  
        String sqlQuery="insert into employees values(?,?,?,?)";  
        PreparedStatement pst = con.prepareStatement(sqlQuery);  
15) Scanner sc = new Scanner(System.in);  
        while(true)  
        {  
            Root.out.println("Employee Number:");  
            int eno=sc.nextInt();  
            Root.out.println("Employee Name:");  
            String ename=sc.next();  
            Root.out.println("Employee Sal:");  
            double esal=sc.nextDouble();  
            Root.out.println("Employee Address:");  
            String eaddr=sc.next();  
            pst.setInt(1,eno);  
            pst.setString(2,ename);  
            pst.setDouble(3,esal);  
            pst.setString(4,eaddr);  
            pst.executeUpdate();  
            Root.out.println("Record Inserted Successfully");  
            Root.out.println("Do U want to Insert one more record[Yes/No]:");  
            String option = sc.next();  
            if(option.equalsIgnoreCase("No"))  
            {  
                break;  
            }  
        }  
        con.close();  
    }  
}
```

Advantages of PreparedStatement:

Performance will be improved when compared with simple Statement b'z query will be compiled only once.

Network traffic will be reduced between java application and database b'z we are not required to send query every time to the database.

We are not required to provide input values at the beginning and we can provide dynamically so that we can execute same query multiple times with different sets of values.

It allows to provide input values in java style and we are not required to convert into database specific format.

Best suitable to insert Date values

Best Suitable to insert Large Objects (CLOB,BLOB)

It prevents SQL Injection Attack.

Limitation of PreparedStatement:

We can use PreparedStatement for only one sql query , but we can use simple Statement to work with any number of queries .

Eg: Statement st = con.createStatement();
st.executeUpdate("insert into ...");
st.executeUpdate("update employees...");
st.executeUpdate("delete...");

Here We Are Using One Statement Object To Execute 3 Queries

```
PreparedStatement pst = con.prepareStatement("insert into employees..");
```

Here PreparedStatement object is associated with only insert query.

Note: Simple Statement can be used only for static queries where as PreparedStatement can be used for both static and dynamic queries.



Differences between Statement And PreparedStatement

Statement	PreparedStatement
1) At the time of creating Statement Object, we are not required to provide any Query. Statement st = con.createStatement(); Hence Statement Object is not associated with any Query and we can use for multiple Queries.	1) At the time of creating PreparedStatement, we have to provide SQL Query compulsory and will send to the Database and will be compiled. PS pst = con.prepareStatement(query); Hence PS is associated with only one Query.
2) Whenever we are using execute Method, every time Query will be compiled and executed.	2) Whenever we are using execute Method, Query won't be compiled just will be executed.
3) Statement Object can work only for Static Queries.	3) PS Object can work for both Static and Dynamic Queries.
4) Relatively Performance is Low.	4) Relatively Performance is High.
5) Best choice if we want to work with multiple Queries.	5) Best choice if we want to work with only one Query but required to execute multiple times.
6) Inserting Date and Large Objects (CLOB and BLOB) is difficult.	6) Inserting Date and Large Objects (CLOB and BLOB) is easy.

Stored Procedures and CallableStatement

In our programming if any code repeatedly required, then we can define that code inside a method and we can call that method multiple times based on our requirement.

Hence method is the best reusable component in our programming.

Similarly in the database programming, if any group of sql statements is repeatedly required then we can define those sql statements in a single group and we can call that group repeatedly based on our requirement.

This group of sql statements that perform a particular task is nothing but Stored Procedure. Hence stored procedure is the best reusable component at database level.

Hence Stored Procedure is a group of sql statements that performs a particular task.

These procedures stored in database permanently for future purpose and hence the name stored procedure.

Usually stored procedures are created by Database Admin (DBA).

Every database has its own language to create Stored Procedures.

Oracle has → PL/SQL

MySQL has → Stored Procedure Language

Microsoft SQL Server has → Transact SQL(TSQL)

Similar to methods stored procedure has its own parameters. Stored Procedure has 3 Types of parameters.

IN parameters(to provide input values)

OUT parameters(to collect output values)

INOUT parameters(to provide input and to collect output)

Eg 1 :

Z:=X+Y;

X,Y are IN parameters and Z is OUT parameter

Eg 2:

X:=X+X;

X is INOUT parameter

Syntax for creating Stored Procedure (mysql):

```
create or replace procedure procedure1(X IN number, Y IN number,Z OUT number) as
BEGIN
  z:=x+y;
END;
```

Note:

SQL and PL/SQL are not case-sensitive languages. We can use lower case and upper case also.

After writing Stored Procedure, we have to compile for this we required to use "/" (forward slash)

/ → For compilation

while compiling if any errors occurs,then we can check these errors by using the following command

```
SQL> show errors;
```

Once we created Stored Procedure and compiled successfully,we have to register OUT parameter to hold result of stored procedure.

```
SQL> variable sum number; (declaring a variable)
```

We can execute with execute command as follows

```
SQL> execute procedure1(10,20,:sum);
```

```
SQL> print sum;
```

Eg 2:

```
create or replace procedure procedure1(X IN number,Y OUT number) as
BEGIN
  Y:= x*x;
END;
/
```

```
SQL> variable square number;
```

```
SQL> execute procedure1(10,:square);
```

```
SQL> print square;
```

SQUARE

100

Eg3: Procedure To Print Employee Salary Based On Given Employee Number.

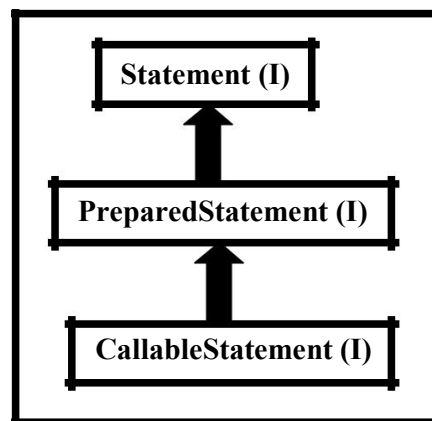
```
create or replace procedure procedure2(eno1 IN number,esal1 OUT number) as
BEGIN
select esal into esal1 from employees where eno=eno1;
END;
/
```

```
SQL>variable salary number;
SQL>execute procedure2(100,;salary);
SQL>print salary;
```

Java Code for calling Stored Procedures:

If we want to call stored procedure from java application, then we should go for CallableStatement.

CallableStatement is an interface present in java.sql package and it is the child interface of PreparedStatement.

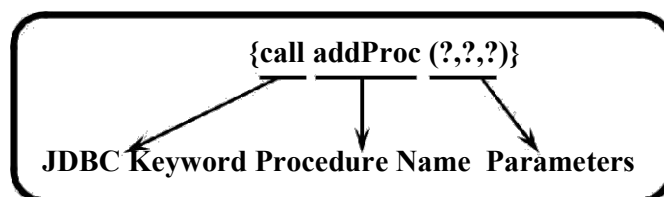


Driver software vendor is responsible to provide implementation for CallableStatement interface.

We can create CallableStatement object by using following method of Connection interface.

```
public CallableStatement prepareCall(String procedure_call) throws SQLException
```

Eg: CallableStatement cst=con.prepareCall("{call addProc(?,?,?)}");



Whenever JVM encounters this line, JVM will send call to database. Database engine will check whether the specified procedure is already available or not. If it is available then it returns CallableStatement object representing that procedure.

Mapping Java Types to database Types by using JDBC Types:

Java related data types and database related data types are not same. Some mechanism must be required to convert java types to database types and database types to java types. This mechanism is nothing but "JDBC Types", which are also known as "Bridge Types".

Java Data Type	JDBC Data Type	Oracle Data Type
int	Types.INTEGER	number
float	Types.FLOAT	number
String	Types.VARCHAR	varchar2
java.sql.Date	Types.DATE	date
:	:	:
:	:	:
:	:	:

Note: JDBC data types are defined as constants in "java.sql.Types" class.

Process to call Stored Procedure from java application by using CallableStatement:

Make sure Stored procedure available in the database

```



create or replace procedure addProc(num1 IN number,num2 IN number,num3 OUT number)
as
    BEGIN
        num3 :=num1+num2;
    END;
/

```

Create a CallableStatement with the procedure call.

```
CallableStatement cst = con.prepareCall("{call addProc(?,?,?)}");
```

3. Provide values for every IN parameter by using corresponding setter methods.

<pre>cst.setInt(1, 100); cst.setInt(2, 200);</pre>
<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  <p>index</p> </div> <div style="text-align: center;">  <p>value</p> </div> </div>

4. Register every OUT parameter with JDBC Types.

If stored procedure has OUT parameter then to hold that output value we should register every OUT parameter by using the following method.

```
public void registerOutParameter (int index, int jdbcType)
```

Eg: `cst.registerOutParameter(3,Types.INTEGER);`

Note:

Before executing procedure call, all input parameters should set with values and every OUT parameter we have to register with jdbc type.

execute procedure
call `cst.execute();`

Get the result from OUT parameter by using the corresponding getXxx() method.

Eg: `int result=cst.getInt(3);`

Stored Procedures App1: JDBC Program to call StoredProcedure which can take two input numbers and produces the result.

Stored Procedure:

```
create or replace procedure addProc(num1 IN number,num2 IN number,num3 OUT
numbe r) as
BEGIN
    num3 :=num1+num2;
END;
/
```

StoredProceduresDemo1.java

```
import java.sql.*;
class StoredProceduresDemo1
{
    public static void main(String[] args) throws Exception
    {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306
", "root", "adam");
        CallableStatement cst=con.prepareCall("{call addProc(?,?,?)}");
        cst.setInt(1,100);
        cst.setInt(2,200);
        cst.registerOutParameter(3,Types.INTEGER);
        cst.execute();
        Root.out.println("Result."+cst.getInt(3));
        con.close();
    }
}
```

```
}  
}
```

Stored Procedures App2: JDBC Program to call StoredProcedure which can take employee number as input and provides corresponding salary.

Stored Procedure:

```
create or replace procedure getSal(id IN number,sal OUT number) as  
BEGIN  
    select esal into sal from employees where eno=id;  
END;  
/
```

StoredProceduresDemo2.java

```
import java.sql.*;  
class StoredProceduresDemo2  
{  
    public static void main(String[] args) throws Exception  
    {  
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",  
            "root","adam");  
        CallableStatement cst=con.prepareCall("{call getSal(?,?)}");  
        cst.setInt(1,100);  
        cst.registerOutParameter(2,Types.FLOAT);  
        cst.execute();  
        Root.out.println("Salary ..." +cst.getFloat(2));  
        con.close();  
    }  
}
```

Statement vs PreparedStatement vs CallableStatement:

We can use normal Statement to execute multiple queries.
`st.executeQuery(query1)`
`st.executeQuery(query2)` `st.executeUpdate(query2)`

i.e if we want to work with multiple queries then we should go for Statement object.

If we want to work with only one query, but should be executed multiple times then we should go for PreparedStatement.

If we want to work with stored procedures and functions then we should go for CallableStatement.

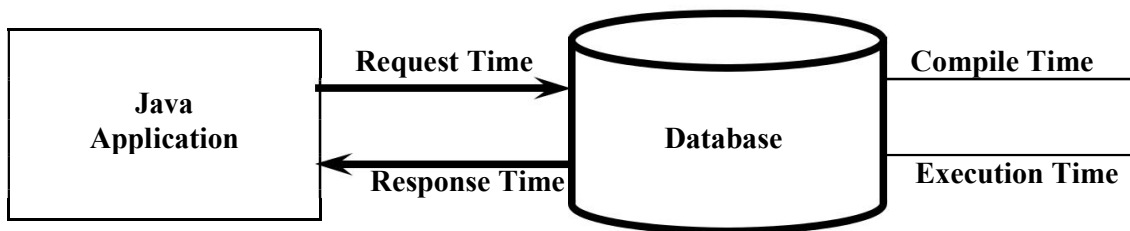


Batch Updates

Need of Batch Updates:

When we submit multiple SQL Queries to the database one by one then lot of time will be wasted in request and response.

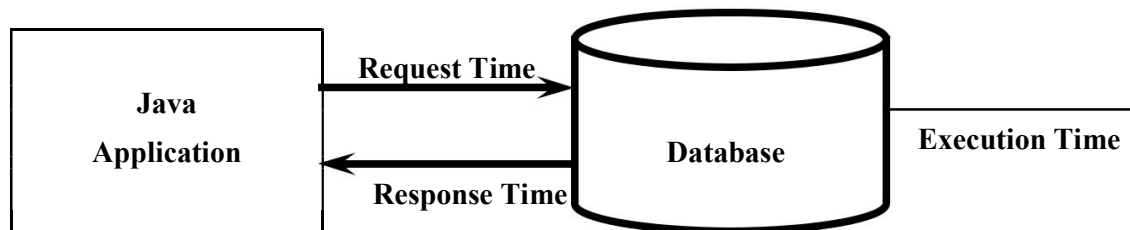
In the case of simple Statement:



$$\text{Total Time per Query} = \text{Req.T} + \text{C.T} + \text{E.T} + \text{Resp.T}$$
$$1 \text{ ms} + 1 \text{ ms} + 1 \text{ ms} + 1 \text{ ms} = 4 \text{ ms}$$

$$\text{per 1000 Queries} = 4 * 1000 \text{ ms} = 4000 \text{ ms}$$

In the case of PreparedStatement:

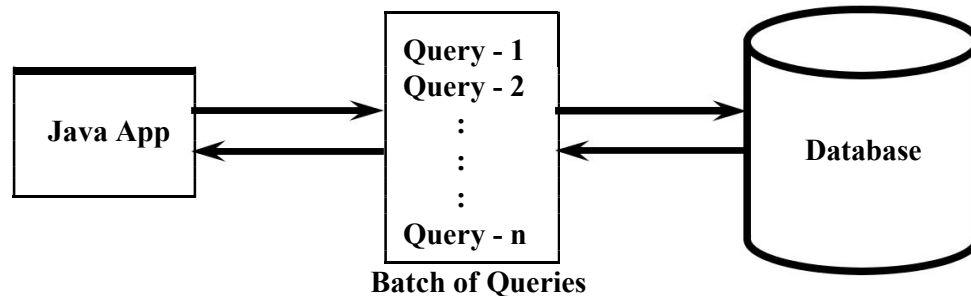


$$\text{Total Time per Query} = \text{Req.T} + \text{Resp.T} + \text{E.T}$$
$$1 \text{ ms} + 1 \text{ ms} + 1 \text{ ms} = 3 \text{ ms}$$

$$1000 \text{ Queries} = 3000 \text{ ms}$$

In the above 2 cases, we are trying to submit 1000 queries to the database one by one. For submitting 1000 queries we need to communicate with the database 1000 times. It increases network traffic between Java application and database and even creates performance problems also.

To overcome these problems, we should go for Batch updates. We can group all related SQL Queries into a single batch and we can send that batch at a time to the database.



With Simple Statement Batch Updates:

Per 1000 Queries = Req.Time+1000*C.T+1000*E.T+Resp.Time
 $1\text{ms} + 1000 * 1\text{ms} + 1000 * 1\text{ms} + 1\text{ms}$
 2002ms

With PreparedStatement Batch Updates:

Per 1000 Queries = Req.Time+1000*E.T+Resp.Time
 $1\text{ms} + 1000 * 1\text{ms} + 1\text{ms}$
 1002ms

Hence the main advantages of Batch updates are

- We can reduce network traffic
- We can improve performance.

We can implement batch updates by using the following two methods

```
public void addBatch(String
sqlQuery) To add query to batch
```

```
int[] executeBatch()
to execute a batch of sql queries
```

We can implement batch updates either by simple Statement or by PreparedStatement

Program to Demonstrate Batch Updates with Simple Statement

```
import java.sql.*;
public class BatchUpdatesDemo1
{
    public static void main(String[] args) throws Exception
    {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",
            "root","root");
        Statement st = con.createStatement();
        //st.addBatch("select * from employees");
        st.addBatch("insert into employees values(600,adam,6000,'Chennai')");
        st.addBatch("update employees set esal=esal+1000 where esal<4000");
        st.addBatch("delete from employees where esal>5000");
        int[] count=st.executeBatch();
        int updateCount=0;
        for(int x: count)
        {
            updateCount=updateCount+x;
        }
        Root.out.println("The number of rows updated :"+updateCount);
        con.close();
    }
}
```

Program to Demonstrate Batch Updates with PreparedStatement

```
import java.sql.*;
import java.util.*;
public class BatchUpdatesDemo2
{
    public static void main(String[] args) throws Exception
    {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",
            "root","root");
        PreparedStatement pst = con.prepareStatement("insert into employees
            values(?,?,?,?)");
        Scanner sc = new Scanner(Root.in);
        while(true)
        {
            Root.out.println("Employee Number:");
            int eno=sc.nextInt();
            Root.out.println("Employee Name:");
            String ename=sc.next();
            Root.out.println("Employee Sal:");
            double esal=sc.nextDouble();
            Root.out.println("Employee Address:");
            String eaddr=sc.next();
        }
    }
}
```

```

        pst.setInt(1,eno);
        pst.setString(2,ename);
        pst.setDouble(3,esal);
        pst.setString(4,eaddr);
        pst.addBatch();
        Root.out.println("Do U want to Insert one more record[Yes/No]:");
        String option = sc.next();
        if(option.equalsIgnoreCase("No"))
        {
            break;
        }
    }
    pst.executeBatch();
    Root.out.println("Records inserted Successfully");
    con.close();
}
}

```

Advantages of Batch Updates:

Network traffic will be reduced
Performance will be improved

Limitations of Batch updates:

We can use Batch Updates concept only for non-select queries. If we are trying to use for select queries then we will get RE saying BatchUpdateException.

In batch if one sql query execution fails then remaining sql queries wont be executed.

In JDBC How Many Execute Methods Are Available?

In total there are 4 methods are available

executeQuery() → For select queries

executeUpdate() → For non-select queries(insert|delete|update)

execute()

For both select and non-select queries
For calling Stored Procedures

executeBatch()→ For Batch Updates



Properties

In Java Program if anything which changes frequently(like jdbc url, username, pwd etc)is not recommended to hard code in our program.

The problem in this approach is if there is any change in java program,to reflect that change we have to recompile,rebuild and redeploy total application and even some times server restart also required,which creates a big business impact to the client.

To overcome this problem, we should go for Properties file. The variable things we have to configure in Properties file and we have to read these properties from java program.

The main advantage of this approach is if there is any change in Properties file and to reflect that change just redeployment is enough, which won't create any business impact to the client.

Program to Demonstrate use of Properties file:

db.properties:

```
url= jdbc:mysql://localhost:3306
user= root
pwd= root
```

JdbcPropertiesDemo.java:

```
import java.sql.*;
import java.util.*;
import java.io.*;
class JdbcPropertiesDemo
{
    public static void main(String[] args) throws Exception
    {
        Properties p = new Properties();
        FileInputStream fis = new FileInputStream("db.properties");
        p.load(fis); // to load all properties from properties file into java Properties object
        String url=p.getProperty("url");
        String user=p.getProperty("user");
        String pwd=p.getProperty("pwd");
        Connection con=DriverManager.getConnection(url,user,pwd);
        Statement st =con.createStatement();
        ResultSet rs=st.executeQuery("select * from employees");
        Root.out.println("ENO\tENAME\tESAL\tEADDR");
    }
}
```

```

Root.out.println("-----");
while(rs.next())
{
Root.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
}
con.close();
}
}

```

If we change the properties in properties file for mysql database then the program will fetch data from mysql database.

db.properties:

```

url= jdbc:com.mysql.jdbc.Driver/adamdb
user= root
pwd= root

```

Program to Demonstrate use of Properties file:

db1.properties:

```

user=root
password=root

```

JdbcPropertiesDemo1.java:

```

import java.sql.*;
import java.util.*;
import java.io.*;
class JdbcPropertiesDemo1 {
public static void main(String[] args) throws Exception
{
Properties p = new Properties();
FileInputStream fis = new FileInputStream("db1.properties");
p.load(fis); // to load all properties from properties file into java Properties object
Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306"
,p);
Statement st =con.createStatement();
ResultSet rs=st.executeQuery("select * from employees");
Root.out.println("ENO\tENAME\tESAL\tEADDR");
Root.out.println("-----");
while(rs.next())
{
Root.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
}
}
}

```

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

How many getConnection() methods are available in DriverManager class.

```
Connection con=DM.getConnection(url,user,pwd);  
Connection con=DM.getConnection(url,Properties);  
Connection con=DM.getConnection(url);
```

Eg:

Connection

```
con=DM.getConnection("jdbc:com.mysql.jdbc.Driver/adamdb?user=root&password=root");
```

Eg:

```
Connection con=DriverManager.getConnection("jdbc:oracle:thin:root/root@localhost:1521:XE");
```

Transaction Management in JDBC

Process of combining all related operations into a single unit and executing on the rule "either all or none", is called transaction management.

Hence transaction is a single unit of work and it will work on the rule "either all or none".

Case-1: Funds Transfer

debit funds from sender's account
credit funds into receiver's account

All operations should be performed as a single unit only. If debit from sender's account completed and credit into receiver's account fails then there may be a chance of data inconsistency problems.

Case-2: Movie Ticket Reservation

Verify the status
Reserve the tickets
Payment
issue tickets.

All operations should be performed as a single unit only. If some operations success and some operations fails then there may be data inconsistency problems.

Transaction Properties:

Every Transaction should follow the following four ACID properties.

1. A → Atomicity

Either all operations should be done or None.

2. C → Consistency(Reliable Data)

It ensures bringing database from one consistent state to another consistent state.

3. I → isolation (Sepatation)

Ensures that transaction is isolated from other transactions

4. D → Durability

It means once transaction committed, then the results are permanent even in the case of root restarts, errors etc.



Types of Transactions:

There are two types of Transactions

- Local Transactions
- Global Transactions

1. Local Transactions:

All operations in a transaction are executed over same database.

Eg: Funds transfer from one account to another account where both accounts in the same bank.

2. Global Transactions:

All operations in a transaction are expected over different databases.

Eg: Funds Transfer from one account to another account and accounts are related to different banks.

Note:

JDBC can provide support only for local transactions.

If we want global transactions then we have to go for EJB or Spring framework.

Process of Transaction Management in JDBC:

1. Disable auto commit mode of JDBC

By default auto commit mode is enabled. i.e after executing every sql query, the changes will be committed automatically in the database.

We can disable auto commit mode as follows

```
con.setAutoCommit(false);
```

If all operations completed then we can commit the transaction by using the following method. `con.commit();`

If any sql query fails then we have to rollback operations which are already completed by using `rollback()` method.

```
con.rollback();
```

Program: To demonstrate Transactions

```
create table accounts(name varchar2(10),balance number);

insert into accounts values('adam',100000);
insert into accounts values('sss',10000);
```

TransactionDemo1.java

```
import java.sql.*;
import java.util.*;
public class TransactionDemo1
{
    public static void main(String[] args) throws Exception
    {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",
            "root","root");
        Statement st = con.createStatement();
        Root.out.println("Data before Transaction");
        Root.out.println("-----");
        ResultSet rs = st.executeQuery("select * from accounts");
        while(rs.next())
        {
            Root.out.println(rs.getString(1)+"..." +rs.getInt(2));
        }
        Root.out.println("Transaction begins...");
        con.setAutoCommit(false);
        st.executeUpdate("update accounts set balance=balance-
            10000 where name='adam'");
        st.executeUpdate("update accounts set balance=balance+10000 where name='sunny'
            ");
        Root.out.println("Can you please confirm this transaction of 10000....[Yes|No]");
        Scanner sc = new Scanner(Root.in);
        String option = sc.next();
        if(option.equalsIgnoreCase("yes"))
        {
            con.commit();
            Root.out.println("Transaction Committed");
        }
        else
        {
            con.rollback();
            Root.out.println("Transaction Rolled Back");
        }
        Root.out.println("Data After Transaction");
        Root.out.println("-----");
        ResultSet rs1 = st.executeQuery("select * from accounts");
```



```

while(rs1.next())
{
    Root.out.println(rs1.getString(1)+"..." +rs1.getInt(2));
}
con.close();
}
}

```

Savepoint(I):

Savepoint is an interface present in java.sql package.

Introduced in JDBC 3.0 Version.

Driver Software Vendor is responsible to provide implementation.

Savepoint concept is applicable only in Transactions.

Within a transaction if we want to rollback a particular group of operations based on some condition then we should go for Savepoint.

We can set Savepoint by using *setSavepoint()* method of Connection interface.

```
Savepoint sp = con.setSavepoint();
```

To perform rollback operation for a particular group of operations wrt Savepoint, we can use rollback() method as follows.

```
con.rollback(sp);
```

We can release or delete Savepoint by using release Savepoint() method of Connection interface.

```
con.releaseSavepoint(sp);
```

Case Study:

```

con.setAutoCommit(false);
Operation-1;
Operation-2;
Operation-3;
Savepoint sp = con.setSavepoint();
Operation-4;
Operation-5;
if(balance<10000)
{
    con.rollback(sp);
}
else
{
    con.releaseSavepoint(sp);
}

```



```
}  
operation-6;  
con.commit();
```

At line-1 if balance < 10000 then operations 4 and 5 will be Rollback, otherwise all operations will be performed normally.

Program to Demonstrate Savepoint:

```
create table politicians(name varchar2(10),party varchar2(10));
```

```
import java.sql.*;  
public class SavePointDemo1  
{  
    public static void main(String[] args) throws Exception  
    {  
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",  
            "root","root");  
        Statement st = con.createStatement();  
        con.setAutoCommit(false);  
        st.executeUpdate("insert into politicians values ('kcr','t1')");  
        st.executeUpdate("insert into politicians values ('babu','t2')");  
        Savepoint sp = con.setSavepoint();  
        st.executeUpdate("insert into politicians values ('siddu','t3')");  
        Root.out.println("oops ..wrong entry just rollback");  
        //con.rollback(sp);  
        con.releaseSavepoint(sp);  
        //Root.out.println("Operations are roll back from Savepoint");  
        con.commit();  
        con.close();  
    }  
}
```

Note:

Some drivers won't provide support for Savepoint. Type-1 Driver won't provide support, but Type-4 Driver can provide support.

Type-4 Driver of Oracle provide support only for *setSavepoint()* and *rollback()* methods but not for *releaseSavepoint()* method.

Transaction Concurrency Problems:

Whenever multiple transactions are executing concurrently then there may be a chance of transaction concurrency problems.

The following are the most commonly occurred concurrency problems.



MetaData

Metadata means data about data. I.e. Metadata provides extra information about our original data.

Eg:

Metadata about database is nothing but database product name, database version etc..

Metadata about ResultSet means no of columns, each column name, column type etc..

JDBC provides support for 3 Types of Metadata

DatabaseMetaData
ResultSetMetaData
ParameterMetaData

1. DatabaseMetaData

It is an interface present in java.sql package.

Driver Software vendor is responsible to provide implementation.

We can use DatabaseMetaData to get information about our database like database product name, driver name, version, number of tables etc..

We can also use DatabaseMetaData to check whether a particular feature is supported by DB or not like stored procedures, full joins etc..

We can get DatabaseMetaData object by using getMetaData() method of Connection interface.

```
public DatabaseMetaData getMetaData();
```

Eg: DatabaseMetaData dbmd=con.getMetaData();

Once we got DatabaseMetaData object we can call several methods on that object

like getDatabaseProductName()
getDatabaseProductVersion()
getMaxColumnsInTable()
supportsStoredProcedures()
etc...



App1: Program to display Database meta information by using DataBaseMetaData

```
import java.sql.*;
class DatabaseMetaDataDemo1
{
    public static void main(String[] args) throws Exception
    {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",
            "root","adam");
        DatabaseMetaData dbmd=con.getMetaData();
        Root.out.println("Database Product Name:"+dbmd.getDatabaseProductName());
        Root.out.println("DatabaseProductVersion:"+dbmd.getDatabaseProductVersion());

        Root.out.println("DatabaseMajorVersion:"+dbmd.getDatabaseMajorVersion());
        Root.out.println("DatabaseMinorVersion:"+dbmd.getDatabaseMinorVersion());
        Root.out.println("JDBCMinorVersion:"+dbmd.getJDBCMinorVersion());
        Root.out.println("JDBCMajorVersion:"+dbmd.getJDBCMajorVersion());
        Root.out.println("DriverName:"+dbmd.getDriverName());
        Root.out.println("DriverVersion:"+dbmd.getDriverVersion());
        Root.out.println("URL:"+dbmd.getURL());
        Root.out.println("UserName:"+dbmd.getUserName());
        Root.out.println("MaxColumnsInTable:"+dbmd.getMaxColumnsInTable());
        Root.out.println("MaxRowSize:"+dbmd.getMaxRowSize());
        Root.out.println("MaxStatementLength:"+dbmd.getMaxStatementLength());
        Root.out.println("MaxTablesInSelect"+dbmd.getMaxTablesInSelect());
        Root.out.println("MaxTableNameLength:"+dbmd.getMaxTableNameLength());
        Root.out.println("SQLKeywords:"+dbmd.getSQLKeywords());
        Root.out.println("NumericFunctions:"+dbmd.getNumericFunctions());
        Root.out.println("StringFunctions:"+dbmd.getStringFunctions());
        Root.out.println("RootFunctions:"+dbmd.getRootFunctions());
        Root.out.println("supportsFullOuterJoins:"+dbmd.supportsFullOuterJoins());
        Root.out.println("supportsStoredProcedures:"+dbmd.supportsStoredProcedures());

        con.close();
    }
}
```

App2: Program to display Table Names present in Database by using DataBaseMetaData

```
import java.sql.*;
import java.util.*;
class DatabaseMetaDataDemo2
{
    public static void main(String[] args) throws Exception
    {
        int count=0;
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",
            "root","adam");
```

```

DatabaseMetaData dbmd=con.getMetaData();
String catalog=null;
String schemaPattern=null;
String tableNamePattern=null;
String[] types=null;
ResultSet rs = dbmd.getTables(catalog,schemaPattern,tableNamePattern,types);
//the parameters can help limit the number of tables that are returned in the ResultSe
t
//the ResultSet contains 10 columns and 3rd column represent table names.
while(rs.next())
{
count++;
Root.out.println(rs.getString(3));
}
Root.out.println("The number of tables:"+count);
con.close();
}
}

```

Note: Some driver softwares may not capture complete information. In that case we will get default values like zero.

Eg: getMaxRowSize() → 0

ResultSetMetaData:

It is an interface present in java.sql package.

Driver software vendor is responsible to provide implementation.

It provides information about database table represented by ResultSet object.

Useful to get number of columns, column types etc..

We can get ResultSetMetaData object by using getMetaData() method of ResultSet interface.

```
public ResultSetMetaData getMetaData()
```

Eg: ResultSetMetaData rsmd=rs.getMetaData();

Once we got ResultSetMetaData object, we can call the following methods on that object

like getColumnCount()

getColumnName()

getColumnType()

etc...



App3: Program to display Columns meta information by using ResultMetaData

```
import java.sql.*;
class ResultSetMetaDataDemo
{
    public static void main(String[] args) throws Exception
    {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",
            "root","adam");
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select * from employees");
        ResultSetMetaData rsmd=rs.getMetaData();
        int count=rsmd.getColumnCount();
        for(int i=1;i<= count;i++)
        {
            Root.out.println("Column Number:"+i);
            Root.out.println("Column Name:"+rsmd.getColumnName(i));
            Root.out.println("Column Type:"+rsmd.getColumnType(i));
            Root.out.println("Column Size:"+rsmd.getColumnDisplaySize(i));
            Root.out.println("-----");
        }
        con.close();
    }
}
```

App4: Program to display Table Data including Column Names by using ResultMetaData

```
import java.sql.*;
class ResultSetMetaDataDemo1 {
    public static void main(String[] args) throws Exception {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",
            "root","root");
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select * from movies");
        ResultSetMetaData rsmd=rs.getMetaData();
        String col1=rsmd.getColumnName(1);
        String col2=rsmd.getColumnName(2);
        String col3=rsmd.getColumnName(3);
        String col4=rsmd.getColumnName(4);
        Root.out.println(col1+"\t"+col2+"\t"+col3+"\t"+col4);
        Root.out.println("-----");
        while(rs.next())
        {
            Root.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.getString(4));
        }
    }
}
```

ParameterMetaData (I):

It is an interface and present in java.sql package.

Driver Software vendor is responsible to provide implementation.

In General we can use positional parameters(?) while creating PreparedStatement object.

```
PreparedStatement pst=con.prepareStatement("insert into employees values(?,?,?,?)");
```

We can use ParameterMetaData to get information about positional parameters like parameter count, parameter mode, and parameter type etc..

We can get ParameterMetaData object by using getParameterMetaData() method of PreparedStatement interface.

```
ParameterMetaData pmd=pst.getParameterMetaData();
```

Once we got ParameterMetaData object, we can call several methods on that object like

1. **int getParameterCount()**
2. **int getParameterMode(int param)**
3. **int getParameterType(int param)**
 String getParameterTypeName(int
param) etc..

Important Methods of ParameterMetaData:

1. int getParameterCount()

Retrieves the number of parameters in the PreparedStatement object for which this ParameterMetaData object contains information.

2.int getParameterMode(int param)

Retrieves the designated parameter's mode.

3. int getParameterType(int param)

Retrieves the designated parameter's SQL type.

4. String getParameterTypeName(int param)

Retrieves the designated parameter's database-specific type name.

5. int getPrecision(int param)

Retrieves the designated parameter's specified column size.

6. int getScale(int param)

Retrieves the designated parameter's number of digits to right of the decimal point.

7. int isNullable(int param)

Retrieves whether null values are allowed in the designated parameter.

8. boolean isSigned(int param)

Retrieves whether values for the designated parameter can be signed numbers.

App14: Program to display Parameter meta information by using ParameterMetaData

```
import java.sql.*;
class ParameterMetaDataDemo
{
    public static void main(String[] args) throws Exception
    {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306",
            "root","adam");
        PreparedStatement pst = con.prepareStatement("insert into employees values(?,?,?,?)");
        ParameterMetaData pmd=pst.getParameterMetaData();
        int count=pmd.getParameterCount();
        for(int i=1;i<= count;i++)
        {
            Root.out.println("Parameter Number:"+i);
            Root.out.println("Parameter Mode:"+pmd.getParameterMode(i));
            Root.out.println("Parameter Type:"+pmd.getParameterType(i));
            Root.out.println("Parameter Precision:"+pmd.getPrecision(i));
            Root.out.println("Parameter Scale:"+pmd.getScale(i));
            Root.out.println("Parameter isSigned:"+pmd.isSigned(i));
            Root.out.println("Parameter isNullable:"+pmd.isNullable(i));
            Root.out.println("-----");
        }
        con.close();
    }
}
```

Note: Most of the drivers won't provide support for ParameterMetaData.



Q1. What is Driver and how many types of drivers are there in JDBC?

The Main Purpose of JDBC Driver is to convert Java (JDBC) calls into Database specific calls and Database specific calls into Java calls. i.e. It acts as a Translator.

There are 4 Types of JDBC Drivers are available

Type-1 Driver (JDBC-ODBC Bridge Driver OR Bridge Driver)

Type-2 Driver (Native API-Partly Java Driver OR Native Driver)

Type-3 Driver (All Java Net Protocol Driver OR Network Protocol Driver OR Middleware Driver)

Type-4 Driver (All Java Native Protocol Driver OR Pure Java Driver OR Thin Driver)

Q2. Explain Differences between executeQuery(), executeUpdate() and execute() methods?

We can use execute Methods to execute SQL Queries.
There are 4 execute Methods in JDBC.

1. executeQuery():

can be used for Select Queries

2. executeUpdate():

Can be used for Non-Select Queries (Insert|Delete|Update)

3. execute()

Can be used for both Select and Non-Select Queries
It can also be used to call Stored Procedures.

4. executeBatch()

Can be used to execute Batch Updates

executeQuery() vs executeUpdate() vs execute():

If we know the Type of Query at the beginning and it is always Select Query then we should use executeQuery() Method.

If we know the Type of Query at the beginning and it is always Non-Select Query then we should use executeUpdate() Method.



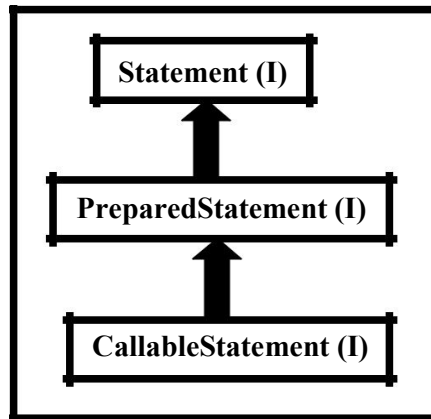
If we don't know the Type of SQL Query at the beginning and it is available dynamically at Runtime (may be from Properties File OR from Command Prompt etc) then we should go for execute() Method.

Q3. What is Statement and How many Types of Statements are available?

To send SQL Query to the Database and to bring Results from Database some Vehicle must be required. This Vehicle is nothing but Statement Object.

Hence, by using Statement Object we can send our SQL Query to the Database and we can get Results from Database.

There are 3 Types of Statements



1. Statement:

If we want to execute multiple Queries then we can use Statement Object. Every time Query will be compiled and executed. Hence relatively performance is low.

2. PreparedStatement:

If we want to execute same Query multiple times then we should go for PreparedStatement. Here Query will be compiled only once even though we executed multiple times. Hence relatively performance is high.

PreparedStatement is always associated with precompiled SQL Queries.

3. CallableStatement:

We can use CallableStatement to call Stored Procedures and Functions from the Database.

Q4. Explain differences between Statement and PreparedStatement?

Differences Between Statement And PreparedStatement

Statement	PreparedStatement
1) At the time of creating Statement Object, we are not required to provide any Query. Statement st = con.createStatement(); Hence Statement Object is not associated with any Query and we can use for multiple Queries.	1) At the time of creating PreparedStatement, we have to provide SQL Query compulsory and will send to the Database and will be compiled. PS pst = con.prepareStatement(query); Hence PS is associated with only one Query.
2) Whenever we are using execute Method, every time Query will be compiled and executed.	2) Whenever we are using execute Method, Query won't be compiled just will be executed.
3) Statement Object can work only for Static Queries.	3) PS Object can work for both Static and Dynamic Queries.
4) Relatively Performance is Low.	4) Relatively Performance is High.
5) Best choice if we want to work with multiple Queries.	5) Best choice if we want to work with only one Query but required to execute multiple times.
6) Inserting Date and Large Objects (CLOB and BLOB) is difficult.	6) Inserting Date and Large Objects (CLOB and BLOB) is easy.

Q5. Explain Steps to develop JDBC Application?

Load and Register Driver
Establish Connection b/w Java Application and Database
Create Statement Object
Send and Execute SQL Query
Process Results from ResultSet
Close Connection

Q6. Explain main Important components of JDBC?

The Main Important Components of JDBC are:

Driver
DriverManager
Connection
Statement
ResultSet

1.Driver(Translator):

To convert Java Specific calls into Database specific calls and Database specific calls into Java calls.

2. DriverManager:

DriverManager is a Java class present in *java.sql* Package.
It is responsible to manage all Database Drivers available in our Root.

DriverManager is responsible to register and unregister Database Drivers.

```
DriverManager.registerDriver(driver);  
DriverManager.unregisterDriver(driver);
```

DriverManager is responsible to establish Connection to the Database with the help of Driver Software.

```
Connection con=DriverManager.getConnection(jdbcurl,username,pwd);
```

3. Connection (Road):

By using Connection, Java Application can communicate with Database.

4. Statement (Vehicle):

By using Statement Object we can send our SQL Query to the Database and we can get Results from Database.

To send SQL Query to the Database and to bring Results from Database some Vehicle must be required. This Vehicle is nothing but Statement Object.

Hence, by using Statement Object we can send our SQL Query to the Database and we can get Results from Database.

There are 3 types of Statements

- 1.Statement
- 2.PreparedStatement
- 3.CallableStatement

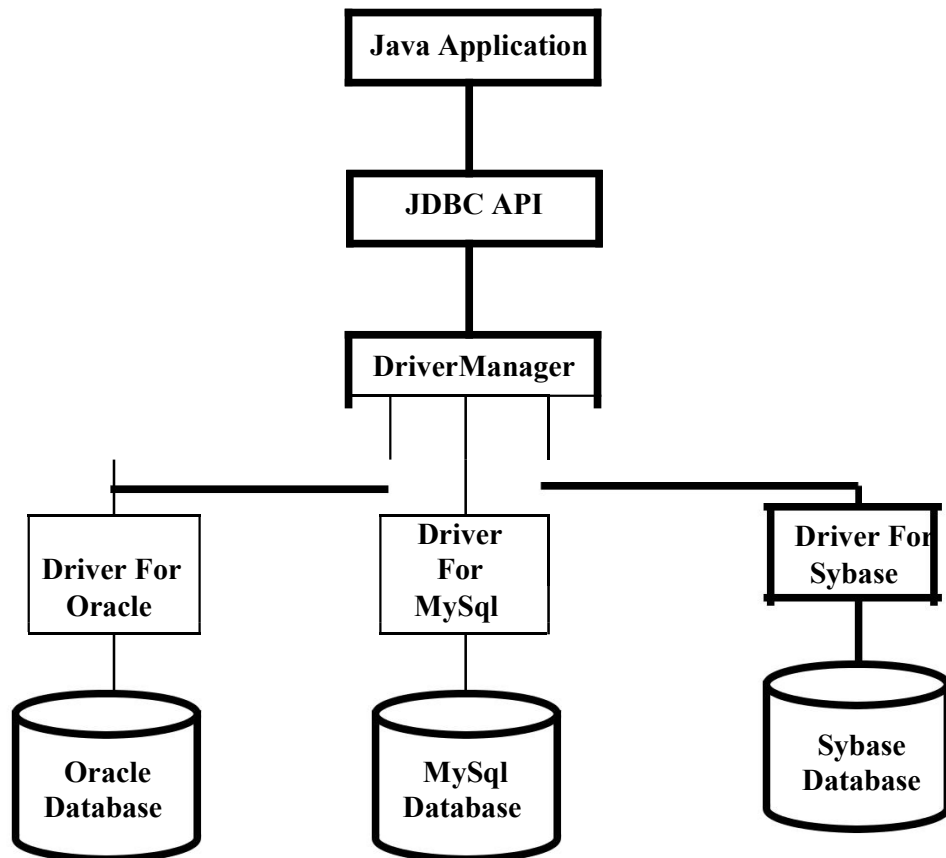
5. ResultSet:

Whenever we are executing Select Query, Database engine will provide Result in the form of ResultSet. Hence ResultSet holds Results of SQL Query. By using ResultSet we can access the Results.



Q7. Explain JDBC Architecture?

JDBC Architecture



JDBC API provides DriverManager to our Java Application.

Java Application can communicate with any Database with the help of DriverManager and Database specific Driver.

DriverManager:

It is the Key Component in JDBC Architecture.

DriverManager is a Java Class present in java.sql Package.

It is responsible to manage all Database Drivers available in our Root.

DriverManager is responsible to register and unregister Database Drivers. `DriverManager.registerDriver(Driver);`
`DriverManager.unregisterDriver(Driver);`

DriverManager is responsible to establish Connection to the Database with the help of Driver Software.

```
Connection con = DriverManager.getConnection (jdbcurl, username, pwd);
```

Database Driver:

It is the very Important Component of JDBC Architecture.

Without Driver Software we cannot Touch Database.

It acts as Bridge between Java Application and Database.

It is responsible to convert Java calls into Database specific calls and Database specific calls into Java Calls.

Q8. Explain about BLOB and CLOB?

Sometimes as the Part of programming Requirement, we have to Insert and Retrieve Large Files like Images, Video Files, Audio Files, Resume etc wrt Database.

Eg:

Upload Image in Matrimonial Web Sites

Upload Resume in Job related Web Sites

To Store and Retrieve Large Information we should go for Large Objects (LOBs).

There are 2 Types of Large Objects.

Binary Large Object (BLOB)

Character Large Object (CLOB)

1. Binary Large Object (BLOB):

A BLOB is a Collection of Binary Data stored as a Single Entity in the Database.

BLOB Type Objects can be Images, Video Files, Audio Files etc..

BLOB Data Type can store Maximum of "4GB" Binary Data.

2. Character Large Objects (CLOB):

A CLOB is a Collection of Character Data stored as a Single Entity in the Database.

CLOB can be used to Store Large Text Documents (May Plain Text OR XML Documents)

CLOB Type can store Maximum of 4 GB Data.

Eg: hydhistory.txt

Q9. Explain about Batch Updates?

When we Submit Multiple SQL Queries to the Database one by one then lot of time will be wasted in Request and Response.

For Example our Requirement is to execute 1000 Queries. If we are trying to submit 1000 Queries to the Database one by one then we need to communicate with the Database 1000 times. It increases Network Traffic between Java Application and Database and even creates Performance Problems also.

To overcome these Problems, we should go for Batch Updates. We can Group all related SQL Queries into a Single Batch and we can send that Batch at a time to the Database.

Sample Code:

```
st.addBatch(sqlQuery-1);
st.addBatch(sqlQuery-2);
st.addBatch(sqlQuery-3);
st.addBatch(sqlQuery-4);
st.addBatch(sqlQuery-5);
st.addBatch(sqlQuery-6);
...
st.addBatch(sqlQuery-1000);
st.executeBatch();
```

