Java PreparedStatement IN clause alternatives

APRIL 2, 2018 BY PANKAJ — 8 COMMENTS

If you are using JDBC API to run queries on database, you should know that PreparedStatement is the better choice than Statement. However since JDBC API allows only one literal for one "?" parameter, PreparedStatement doesn't work for IN clause queries.

PreparedStatement IN clause

So if we need to execute a database query with IN clause, we need to look for some alternative approach. The aim of this post is to analyze different approaches and you can choose the one that suits your requirements.

1. Execute Single Queries
2. Using Stored Procedure
3. Creating PreparedStatement Query dynamically
4. Using NULL in PreparedStatement Query

Let's look at these approaches one by one. But before that let's create a utility program for database connection reading configurations from property file.

db.properties

```
#mysql DB properties
DB_DRIVER_CLASS=com.mysql.jdbc.Driver
DB_URL=jdbc:mysql://localhost:3306/UserDB
DB_USERNAME=pankaj
DB_PASSWORD=pankaj123

#Oracle DB Properties
#DB_DRIVER_CLASS=oracle.jdbc.driver.OracleDriver
#DB_URL=jdbc:oracle:thin:@localhost:1521:orcl
#DB_USERNAME=hr
#DB_PASSWORD=oracle
```

```
package com.journaldev.jdbc.preparedstatement.in;

import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class DBConnection {

	public static Connection getConnection() {
		Properties props = new Properties();
		FileInputStream fis = null;
		Connection con = null;
```

```
		try {
			fis = new FileInputStream("db.properties");
			props.load(fis);

			// load the Driver Class
			Class.forName(props.getProperty("DB_DRIVER_CLASS"));
```

Make sure you have JDBC jars in the build path of the project.

Now let's look at the different approaches and their analysis.

Execute Single Queries

This is the simplest approach. We can get the input and execute single PreparedStatement query multiple times. A sample program with this approach will look like below.

JDBCPreparedStatementSingle.java

```
package com.journaldev.jdbc.preparedstatement.in;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JDBCPreparedStatementSingle {

	private static final String QUERY = "select empid, name from Employee where
empid = ?";

	public static void printData(int[] ids){
		Connection con = DBConnection.getConnection();
		PreparedStatement ps = null;
		ResultSet rs = null;
		try {
			ps = con.prepareStatement(QUERY);

			for(int empid : ids){
				ps.setInt(1, empid);
				rs = ps.executeQuery();
```

The approach is simple but it's very slow because if there are 100 parameters then it will make 100 database calls. This will result in 100 ResultSet objects that will overload the system and it will also cause performance hit. So this approach is not recommended.

Using Stored Procedure

We can write a stored procedure and send the input data to the stored procedure. Then we can execute queries one by one in the stored procedure and get the results. This approach gives fastest performance but as we all know that Stored Procedures are database specific. So if our application deals with multiple types of databases such as Oracle, MySQL then it will become hard to maintain. We should use this approach only when we are working on single type of database and there is no plan to change the database server. Since writing stored procedure is out of scope of this tutorial, I will not demonstrate how to use it.

Creating PreparedStatement Query dynamically

This approach involves writing logic to create the PreparedStatement query dynamically based on the size of the elements in IN clause. A simple example showing how to use it will look like below code.

JDBCPreparedStatementDynamic.java

```
package com.journaldev.jdbc.preparedstatement.in;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JDBCPreparedStatementDynamic {

	public static void printData(int[] ids){

		String query = createQuery(ids.length);

		System.out.println("Query="+query);
		Connection con = DBConnection.getConnection();
		PreparedStatement ps = null;
		ResultSet rs = null;
		try {
			ps = con.prepareStatement(query);

			for(int i = 1; i <=ids.length; i++){
				ps.setInt(i, ids[i-1]);
```

Notice that the query is created dynamically and it will run perfectly. There will be only one database call and the performance will be good. However if the size of user input varies a lot, we won't get the PreparedStatement benefit of caching and reusing the execution plan. If you are not worried about PreparedStatement caching and there are not many queries with IN clause, then it seems to be the way to go.

Using NULL in PreparedStatement Query

If you really want to utilize the PreparedStatement caching feature, then another approach is to use NULL in PreparedStatement parameters. Suppose that the maximum allowed parameters in the query is 10, then we can write our logic like below.

JDBCPreparedStatementNULL.java

```
package com.journaldev.jdbc.preparedstatement.in;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JDBCPreparedStatementNULL {

	private static final String QUERY = "select empid, name from Employee where
empid in ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
	private static final int PARAM_SIZE = 10;
	public static void printData(int[] ids){

		if(ids.length > PARAM_SIZE){
			System.out.println("Maximum input size supported is
"+PARAM_SIZE);
			//in real life, we can write logic to execute in batches, for
simplicity I am returning
			return;
		}
		Connection con = DBConnection.getConnection();
```

Notice that above program is using same PreparedStatement query for executing IN clause statement and will get the benefit of query caching and executing plan. For simplicity, I am just returning if the number of input parameters is greater than the parameters size in the query but we can easily extend it to execute in batches to allow any number of inputs.

Now let's write a simple test program to check the output. For my test program, I am using Employee table created in JDBC DataSource example.

Our test program code is;

JDBCPreparedStatementINTest.java

```
package com.journaldev.jdbc.preparedstatement.in;

public class JDBCPreparedStatementINTest {

	private static int[] ids = {1,2,3,4,5,6,7,8,9,10};

	public static void main(String[] args) {

		JDBCPreparedStatementSingle.printData(ids);

		System.out.println("*********");

		JDBCPreparedStatementDynamic.printData(ids);

		System.out.println("*********");

		JDBCPreparedStatementNULL.printData(new int[]{1,2,3,4,5});
	}

}
```

When we execute it with some test data in Employee table, we get below output.

```
Employee ID=1, Name=Pankaj
Employee ID=2, Name=David
Employee ID=3, Name=Ram
Employee ID=4, Name=Leela
Employee ID=5, Name=Lisa
Employee ID=6, Name=Saurabh
Employee ID=7, Name=Mani
Employee ID=8, Name=Avinash
Employee ID=9, Name=Vijay
*********
Query=select empid, name from Employee where empid in ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Employee ID=1, Name=Pankaj
Employee ID=2, Name=David
Employee ID=3, Name=Ram
Employee ID=4, Name=Leela
```

```
Employee ID=5, Name=Lisa
Employee ID=6, Name=Saurabh
Employee ID=7, Name=Mani
Employee ID=8, Name=Avinash
Employee ID=9, Name=Vijay
*********
Employee ID=1, Name=Pankaj
```

That's all for the different options we have to use PreparedStatement for IN clause in queries. You can use any one of these based on your project requirements.

« PREVIOUS
Tomcat DataSource JNDI Example in Java

NEXT »
JDBC Interview Questions and Answers

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DATABASE, JAVA

Comments

avalsa says
FEBRUARY 1, 2018 AT 3:12 AM
Thanks, but what if db field is nullable, and in query like select * from User where extraInfo in('a', 'b', NULL, NULL). Imagine that extraInfo is char NULL. How would you deal with it?
Reply

Alex says
MARCH 3, 2018 AT 12:05 PM
just add extraInfo is not null at the end of the query.
Reply

Pravin says
MAY 18, 2017 AT 12:15 AM
Nice article...!!!
But one concern here is, if I'm having different data types when setting to the Prepared Statement object (JDBCPreparedStatementDynamic.java)::
ps = con.prepareStatement(query);
for(int i = 1; i Hence _May/Must_ this approch is not worthy for different type of data types of data in database.
Thanks
Reply

Shruti says
MARCH 30, 2015 AT 10:20 PM
This code is awsome , it worked like a cham, Thank you, u saved my many hours □
Reply

Pallavi K says
JANUARY 20, 2015 AT 11:06 PM

Just what I needed !!! Thanks!!

Reply

vivek mishra says

MAY 9, 2014 AT 2:49 PM

Thanks Pankaj for this blog. I was also thinking in same line and your blog has helped me to decide on final design I need to follow

Thanks

Vivek Mishra

Reply

Sachin says

JULY 9, 2014 AT 1:58 AM

Sir, Really very good ……..U r Rock Sir…….

Reply

Rishi Raj says

FEBRUARY 3, 2014 AT 9:45 AM

Thanks, Pankaj, for presenting these methods.

One query here:

I was expecting data for 10 employee records to be returned on running JDBCPreparedStatementINTest, but the returned data are only for emp IDs- 1 to 9.

Please check if my expectation is wrong or there is something wrong with the output.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

CORE JAVA TUTORIAL

Java 10 Tutorials Java 9 Tutorials Java 8 Tutorials Java 7 Tutorials Core Java Basics OOPS Concepts Data Types and Operators String Manipulation Java Arrays Annotation and Enum Java Collections Java IO Operations Java Exception Handling MultiThreading and Concurrency

Regular Expressions Advanced Java Concepts

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial