Java JDBC Transaction Management and Savepoint

APRIL 2, 2018 BY PANKAJ — 12 COMMENTS

Transaction Management in java is required when we are dealing with relational databases. We use JDBC API for database operations and today we will learn how to use JDBC transaction management. In the **JDBC Tutorial** we learned how we can use JDBC API for database connectivity and execute SQL queries. We also looked at the different kind of drivers and how we can write loosely couple JDBC programs that helps us in switching from one database server to another easily.

Transaction Management in Java JDBC

This tutorial is aimed to provide details about **JDBC Transaction Management** and using **JDBC Savepoint** for partial rollback.

By default when we create a database connection, it runs in **auto-commit** mode. It means that whenever we execute a query and it's completed, the commit is fired automatically. So every SQL query we fire is a transaction and if we are running some DML or DDL queries, the changes are getting saved into database after every SQL statement finishes.

Sometimes we want a group of SQL queries to be part of a transaction so that we can commit them when all the queries runs fine. If we get any exception, we have a choice of rollback all the queries executed as part of the transaction.

Let's understand with a simple example where we want to utilize JDBC transaction management support for data integrity. Let's say we have UserDB database and Employee information is saved into two tables. For

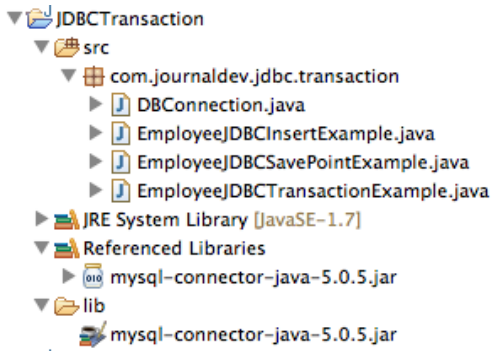
my example, I am using MySQL database but it will run fine on other relational databases as well such as Oracle and PostgreSQL.

The tables store employee information with address details in tables, DDL scripts of these tables are like below.

```
CREATE TABLE `Employee` (
  `empId` int(11) unsigned NOT NULL,
  `name` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`empId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `Address` (
  `empId` int(11) unsigned NOT NULL,
  `address` varchar(20) DEFAULT NULL,
  `city` varchar(5) DEFAULT NULL,
  `country` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`empId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Our final project looks like below image, we will look into each of the classes one by one.



As you can see that I have MySQL JDBC jar in the project build path, so that we can connect to the MySQL database.

DBConnection.java

```
package com.journaldev.jdbc.transaction;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
public class DBConnection {

	public final static String DB_DRIVER_CLASS = "com.mysql.jdbc.Driver";
	public final static String DB_URL = "jdbc:mysql://localhost:3306/UserDB";
	public final static String DB_USERNAME = "pankaj";
	public final static String DB_PASSWORD = "pankaj123";

	public static Connection getConnection() throws ClassNotFoundException,
SQLException {

		Connection con = null;

		// load the Driver Class
		Class.forName(DB_DRIVER_CLASS);

		// create the connection now
```

DBConnection is the class where we are creating MySQL database connection to be used by other classes.

EmployeeJDBCInsertExample.java

```
package com.journaldev.jdbc.transaction;

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

public class EmployeeJDBCInsertExample {

	public static final String INSERT_EMPLOYEE_QUERY = "insert into Employee
(empId, name) values (?,?)";

	public static final String INSERT_ADDRESS_QUERY = "insert into Address
(empId, address, city, country) values (?,?,?,?)";

	public static void main(String[] args) {

		Connection con = null;
		try {
			con = DBConnection.getConnection();
```

```
                insertEmployeeData(con, 1, "Pankaj");
```

This is a simple JDBC program where we are inserting user provided values in both Employee and Address tables created above. Now when we will run this program, we will get following output.

```
DB Connection created successfully
Employee Data inserted successfully for ID=1
com.mysql.jdbc.MysqlDataTruncation: Data truncation: Data too long for column 'city'
at row 1
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2939)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)
        at
com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1268)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1541)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1455)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1440)
        at
com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.insertAddressData(EmployeeJDB

        at
com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.main(EmployeeJDBCInsertExampl
```

As you can see that SQLException is raised when we are trying to insert data into Address table because the value is bigger than the size of the column.

If you will look at the content of the Employee and Address tables, you will notice that data is present in Employee table but not in Address table. This becomes a serious problem because only part of the data is inserted properly and if we run the program again, it will try to insert into Employee table again and throw below exception.

```
com.mysql.jdbc.exceptions.MySQLIntegrityConstraintViolationException: Duplicate entry
'1' for key 'PRIMARY'
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:931)
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2941)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
```

```
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)
        at
com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1268)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1541)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1455)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1440)
        at
com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.insertEmployeeData(EmployeeJD

        at
com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.main(EmployeeJDBCInsertExampl
```

So there is no way we can save the data into Address table now for the Employee. So this program leads to data integrity issues and that's why we need transaction management to insert into both the tables successfully or rollback everything if any exception arises.

JDBC Transaction Management

JDBC API provide method setAutoCommit() through which we can disable the auto commit feature of the connection. We should disable auto commit only when it's required because the transaction will not be committed unless we call the commit() method on connection. Database servers uses table locks to achieve transaction management and it's resource intensive process. So we should commit the transaction as soon as we are done with it. Let's write another program where we will use JDBC transaction management feature to make sure data integrity is not violated.

EmployeeJDBCTransactionExample.java

```
package com.journaldev.jdbc.transaction;

import java.sql.Connection;
import java.sql.SQLException;

public class EmployeeJDBCTransactionExample {

        public static void main(String[] args) {

                Connection con = null;
                try {
                        con = DBConnection.getConnection();
```

```
                        //set auto commit to false
                        con.setAutoCommit(false);

                        EmployeeJDBCInsertExample.insertEmployeeData(con, 1,
"Pankaj");

                        EmployeeJDBCInsertExample.insertAddressData(con, 1, "Albany
Dr", "San Jose", "USA");
```

Please make sure you remove the earlier inserted data before running this program. When you will run this program, you will get following output.

```
DB Connection created successfully
Employee Data inserted successfully for ID=1
com.mysql.jdbc.MysqlDataTruncation: Data truncation: Data too long for column 'city'
at row 1
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2939)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)
        at
com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1268)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1541)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1455)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1440)
        at
com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.insertAddressData(EmployeeJDB

        at
com.journaldev.jdbc.transaction.EmployeeJDBCTransactionExample.main(EmployeeJDBCTransac
```

The output is similar to previous program but if you will look into the database tables, you will notice that data is not inserted into Employee table. Now we can change the city value so that it can fit in the column and rerun the program to insert data into both the tables. Notice that connection is committed only when both the inserts executed fine and if any of them throws exception, we are rolling back complete transaction.

JDBC Savepoint

Sometimes a transaction can be group of multiple statements and we would like to rollback to a particular point in the transaction. JDBC Savepoint helps us in creating checkpoints in a transaction and we can rollback to that particular checkpoint. Any savepoint created for a transaction is automatically released and become invalid when the transaction is committed, or when the entire transaction is rolled back. Rolling a transaction back to a savepoint automatically releases and makes invalid any other savepoints that were created after the savepoint in question.

Let's say we have a Logs table where we want to log the messages that employee information is saved successfully. But since it's just for logging, if there are any exceptions while inserting into Logs table, we don't want to rollback the entire transaction. Let's see how we can achieve this with JDBC savepoint.

```
CREATE TABLE `Logs` (
  `id` int(3) unsigned NOT NULL AUTO_INCREMENT,
  `message` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

EmployeeJDBCSavePointExample.java

```
package com.journaldev.jdbc.transaction;

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

public class EmployeeJDBCSavePointExample {

        public static final String INSERT_LOGS_QUERY = "insert into Logs (message)
values (?)";

        public static void main(String[] args) {

                Connection con = null;
                Savepoint savepoint = null;
                try {
                        con = DBConnection.getConnection();

                        // set auto commit to false
                        con.setAutoCommit(false);
```

The program is very simple to understand. As you can see that I am creating the savepoint after data is inserted successfully into Employee and Address tables. If SQLException arises and savepoint is null, it means that exception is raised while executing insert queries for either Employee or Address table and hence I am rolling back complete transaction.

If savepoint is not null, it means that SQLException is coming in inserting data into Logs table, so I am rolling back transaction only to the savepoint and committing it.

If you will run above program, you will see below output.

```
DB Connection created successfully
Employee Data inserted successfully for ID=2
Address Data inserted successfully for ID=2
com.mysql.jdbc.MysqlDataTruncation: Data truncation: Data too long for column
'message' at row 1
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2939)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)
        at
com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1268)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1541)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1455)
        at
com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1440)
        at
com.journaldev.jdbc.transaction.EmployeeJDBCSavePointExample.insertLogData(EmployeeJDBC


        at
```

If you will check database tables, you will notice that the data is inserted successfully in Employee and Address tables. Note that we could have achieved this easily by committing the transaction when data is inserted successfully in Employee and Address tables and used another transaction for inserting into logs table. This is just an example to show the usage of JDBC savepoint in java programs.

Download JDBC Transaction Management Example Project

Download project from above link and play around with it, try to use multiple savepoints and JDBC transactions API to learn more about it.

« PREVIOUS
JDBC Example – MySQL, Oracle

NEXT »
JDBC Statement vs PreparedStatement – SQL Injection Example

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

javed says
JUNE 10, 2018 AT 3:49 AM

very good explanation with simple example.
Reply

Srinivas says
MAY 27, 2018 AT 9:55 PM
I tried similar example, in which my first query executes succesfully and second query throws an exception but the data inserted into table1 didnt roll back. I dont konw where I have a mistake. Can someone please help me. I have followed the same steps as above but with different tables and java classes
Reply

Nissi says
JANUARY 11, 2018 AT 6:51 AM
Super, Really great explanation!
Reply

pralad says
MAY 15, 2017 AT 12:03 AM
I saw that in c# the transaction scope manage by itself without comitting or setting g rollback (May be i am wrong , not a c# guy). I want to do same in java , is there a way? or i needed to commit every time.
Reply

sandeep says
JULY 20, 2015 AT 3:28 AM
Good article. Easy to understand.
Reply

shadab says
APRIL 11, 2015 AT 10:25 AM
it should always be a practice to revert
con.setAutoCommit(true); in the finally block.
which is missed out in the example above.
Reply

mayank says
FEBRUARY 11, 2014 AT 12:26 PM
this is inserting data properly but it is showing error on console that no operation allowed after connection has been closed
WHY ? can u suggest me something regarding this
Reply

Pankaj says
FEBRUARY 11, 2014 AT 4:59 PM
Can you tell me which program and which line number is giving warning message?
Reply

mayank says
FEBRUARY 12, 2014 AT 5:31 AM
hello sir
i have my program in which i have to insert data into more than 1 table in that case first of all i made auto commit false then after executing queries i call commit and in catch statement i call rollback, for both lines where i call commit and roll back it is showing (No operations allowed after connection closed) in starting i used connection.getConnection to get connection from mySql
Reply

mayank says
FEBRUARY 12, 2014 AT 5:41 AM
sir i come to know the problem what coming, it is showing warning because i have some methods before this method in which i have close connection so it is showing warning but in starting of this method i call connection.getConnection even then it is showing this warning
Reply

Siddu says
JANUARY 24, 2014 AT 6:34 AM
How to specify more than one savepoint in a one transction & how to rollback to particular savepoint. Could you explain please.

Reply

subbareddy says

JANUARY 10, 2014 AT 9:44 AM

Good effort on jdbc……thnks

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...

- 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

Java Exceptions

Java Source Code

Java Application Server

Java Documentation

Access JDBC Driver

Java Swing

Java Runtime Environment

Java Applets