7. Working with a CallableStatement

Contents

[1.Introduction 1](#_Toc21361)

[2. Demonstration - Callable Statements 2](#_Toc13623)

[3. In Parameters 3](#_Toc4228)

[4. Demonstration - In Parameters 4](#_Toc26603)

[5. Out Parameters 5](#_Toc9281)

[6. Demonstration - Out Parameters 6](#_Toc27730)

[7. In/Out Parameters 7](#_Toc3796)

[8. Demonstration - In/Out Parameters 7](#_Toc8542)

[9. JDBC Stored Procedures with Exception Handling 9](#_Toc31997)

[MySQL error handling examples 10](#_Toc27678)

[10. Summary 13](#_Toc9647)

# 1.Introduction

=>slides: Pg. 1

Welcome back to the Database Applications with JDBC in Java SE Applications course. This is the Working with a CallableStatement modules.

=>slides: Pg. 2

So what's in this module? So databases can have stored procedures and we need a way of executing those stored procedures, and the way we do that in JDBC is to use a CallableStatement rather than a PreparedStatement. So we'll see how we can use CallableStatements to execute the stored procedure inside a database. We'll show how to pass data to that CallableStatement so stored procedures can have input parameters, and the CallableStatement needs to be able to pass those parameters through to the stored procedure, and the stored procedures return data, and they can return data as a ResultSet, and we'll see how to do that, but they can also have OUT parameters. So we'll see how to retrieve data from the CallableStatement as well.

=>slides: Pg. 3

So for this module, we've created four stored procedures that are in our database. One will get a list of all the acts in the database, one creates a report of which gigs are running when, one will tell us the total sales that we've achieved for all the gigs, and one will try and raise the ticket price of a given gig. And between them, these allow us to show IN parameters, OUT parameters, and IN/OUT parameters. I'm not going to go into the details of the stored procedure code, but this is provided in the demo code for this course. Okay, so given that we have these stored procedures, how do we call them from JDBC?

=>slides: Pg. 4

So what is the syntax for CallableStatement? So when we create the CallableStatement, we pass a syntax which looks something like this. So, it'll be a string and in that string we have braces, and inside the braces we say call and then the name of the stored procedure. So in our case, this procedure\_name would be the name of one of the stored procedures in the database, one of the four that we've created. =>slides: Pg. 5

So this is one of the stored procedures. We've called it GetActs, and notice this select the name of the act and the recordlabel from the act from the acts database, and there's a filter to say where the recordlabel IS NOT null. So it gets all the acts that have been signed to a record label.

=>slides: Pg. 6

So to call this, we'd write code something like this. The sql has a syntax for CallableStatement, so, braces, and inside those braces, call, and then the name of the stored procedure, which is GetActs. Now in this case, GetActs has no IN parameters and no OUT parameters. So essentially it creates a ResultSet that we can then read. We prepare the call. So rather than creating a PreparedStatement, we create a CallableStatement by calling prepareCall and we pass it the sql. We call executeQuery against that call, so in the same way we'd execute a PreparedStatement, and we get back a ResultSet. And again, in the same way we would with a PreparedStatement, we iterate over the ResultSet and get the values, and then use them. Okay, so let's take a look at the code that can do this.

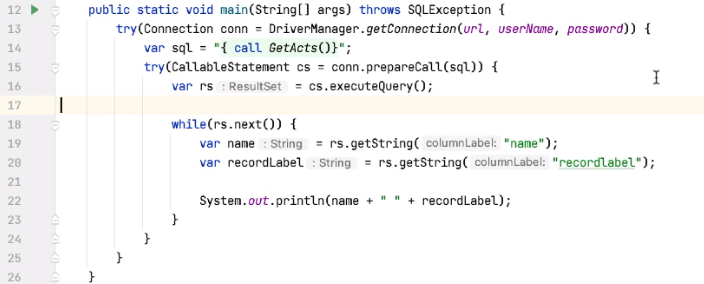
# Demonstration - Callable Statements

=>slides: Pg. 7

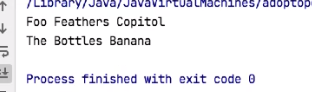




Okay, so here we are again. There are try resources creating the connection. So inside here, the first thing we do again is to create our SQL.



So let's say var sql =. So remember the syntax for this is to use braces, and then say call, and then the name of the stored procedure that we're going to call, and ours is called GetActs. Once I have the SQL, I can then prepare the called procedure. And again, we can wrap this in a try‑with‑resources block. So we can say try, call up a statement, cs = connection.prepareCall, and we pass prepareCall the SQL from above. Inside here, we can say rs = cs.executeQuery. And then like this, the rest of the code is the same as for a PreparedStatement. So we'd say while rs.next, var name = getString name, and var recordLabel = getString recordlabel. And then we can make use of this data.



So if I run this, sure enough, we get the acts with the recordLabel.



So let me just put a hyphen here just to make this slightly clearer, and run it again. So Foo Feathers is assigned to Copitol, and The Bottles is assigned to Banana.

# In Parameters

=>slides: Pg. 8

So CallableStatements wouldn't be much use if we couldn't pass parameters into them, and there are three types of parameters, IN, OUT, and INOUT. So first of all, we'll take a look at IN parameters. So we've seen that we use CallableStatements to call stored procedures, and then we can set IN parameters on this, and they're just like setting parameters on a PreparedStatement.

=>slides: Pg. 9

So our second called procedure looks something like this. It's called GigReport, and it takes two IN parameters, and they're both dates, and we're looking for gigs between certain dates. So how do we set those parameters?

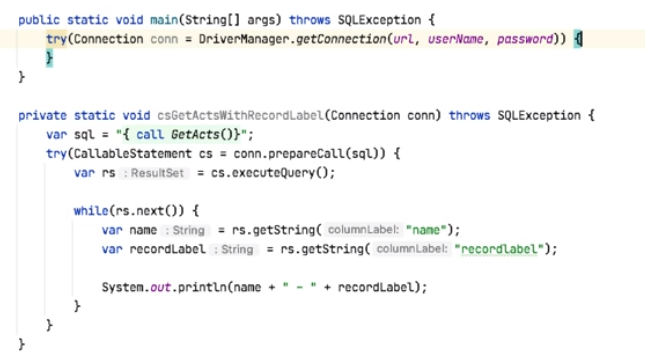
=>slides: Pg. 10

Again, we have the SQL, so again, notice we have the braces. We say call, we give it the name of the stored procedure, and then we use question marks where the parameters would be. We create the CallableStatement by calling prepareCall, and then we set the two parameters, and here we're using different setters, we're calling setDate to set these values, so we set the startdate and the enddate, then we execute the query, and then we use the data from the columns in the stored procedure. So let's go and see how we use that.

# Demonstration - In Parameters

=>slides: Pg. 11

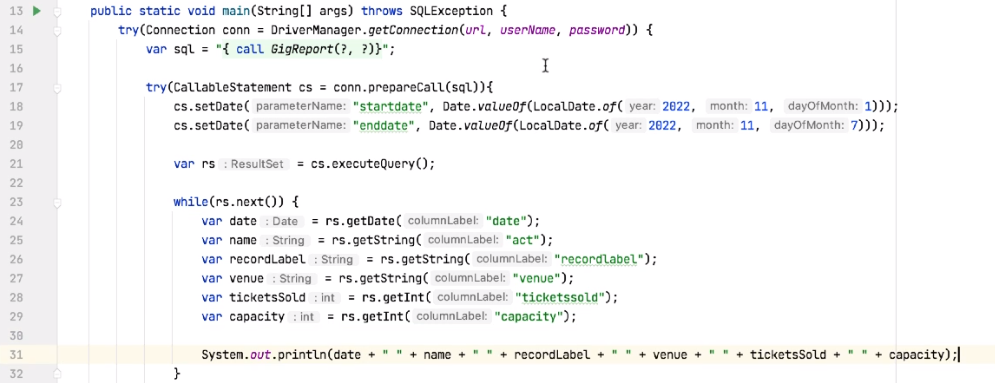
So how do we do this?



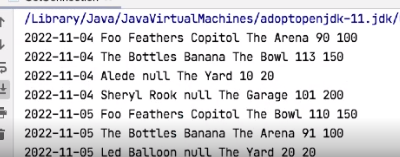
So here I'm starting with a blank slate again. I've moved the previous code out into its own method. So to call a stored procedure with parameters, we first define the sql.

So, again, the braces, and then we say, call, the name of the stored procedure, which in our case is GigReport, and then question marks for each of the parameters we're going to pass to that stored procedure. We then create the CallableStatement. So, again, inside a try block, let's say CallableStatement cs = conn.prepareCall, passing it the sql. Now inside here I want to set these two parameters and they're both dates. You can do cs.setDate, and again, we can do this by index or by name. We're going to use the names in this case.





And if I look at the stored procedure, this is in the initial\_tables.sql, we can see that the parameters are called startdate and enddate. So here we can say startdate, we use LocalDate to create the date, and then we convert that into a Date object. And the code looks like this, we say Date.valueOf, LocalDate.of, and let's use 2022, 11, 1 as the startdate. We can duplicate that line, and then for the enddate let's set November the seventh, so we're looking for dates between the 1st of November and the 7th of November. Once we have that, we can execute the query, and then we can add our while loop. And inside here, we can extract the values. We'll get the date column, the name of the act, the name of the record label, the venue, the tickets that have been sold, and the capacity for this gig, and then we can print that out. Okay, so if we run this, we should get a report of all the gigs between the 1st of November and the 7th of November, and print out all the information associated with those gigs.



So let's give that a go, and sure enough, there's the report. So we have a gig, for example, on the 4th of November with the Foo Feathers, who have been signed to Copitol, it's at The Arena, we've sold 90 tickets, and the capacity is 100, and we can see that report for all those entries in the database.

# Out Parameters

=>slides: Pg. 12

So stored procedures can also return values, and we specify this by using what are known as OUT parameters. So to get a value back from a stored procedure, we use syntax which looks something like this, so ?=. So our SQL would look like this, so ?= call, and then the name of the stored procedure, and there may be other parameters that we need to use. So be aware that this syntax is optional. Not all JDBC drivers support this, and, in fact, the MySQL driver that we're using here doesn't support this, but you will see this syntax used. However, whether this syntax is used or not, we have to register the out parameters before we make the call to let the driver know that we are expecting data back, and the type of data that we're expecting. So let's see an example.

=>slides: Pg. 13

So our stored procedure with an OUT parameter looks like this. It's going to return a simple value that's the total sales that we've made. So notice that when we create the stored procedure, we give it an OUT parameter, call sales, and it's a decimal type, and inside the stored procedure we calculate the total value and we select it into the OUT parameter. So our stored procedure has a single OUT parameter here. Stored procedures can have multiple out parameters.

=>slides: Pg. 14

So how do we use this? So, again, we create the SQL. Now, in our case we're not using the ?= syntax as the driver doesn't support it, but you will see that syntax. So we're saying call GetTotalSales, and in the same way that we define an IN parameter as a question mark, we define an OUT parameter as a question mark, and a stored procedure can be a mix of IN and OUT parameters, as we'll see a little later. So we prepare the call, and then we register any OUT parameters. So here the parameter in the first position is an OUT parameter. Again, note that it's 1 based, not 0 based. And we tell the driver what type of parameter it is, in this case it's a decimal type. We then call execute to execute the stored procedure, so not executeQuery. And then to get the data from the OUT parameter, we call the get method on the CallableStatement. So in this case, we want to get the Double value, so we call cs.getDouble, and it's the parameter at position number 1 that we're getting the value from. Okay, so let's go and see this in code.

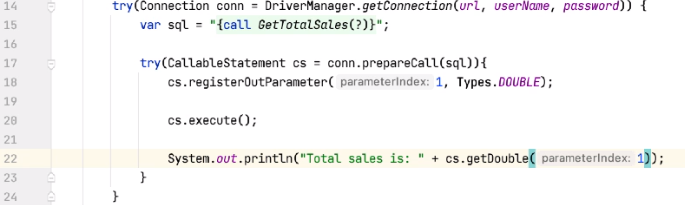
# Demonstration - Out Parameters

=>slides: Pg. 15

So again, we start with the SQL.



So my SQL will look like braces, and then call the name of the stored procedure, which is GetTotalSales, and then the placeholder for the out parameter. Now remember, the alternative syntax to this is to say ?=. And if you can use a syntax, you probably should, as it just shows that we have out parameters.



But the driver I'm using doesn't support this, so we'll just say call GetTotalSales. Once we have that, we need to create the callable statement, so we can say CallableStatement cs = conn.prepareCall, passing it the sql. And then we have to register any out parameters that we have. So we say cs.registerOutParameter, it's position number 1, and then the type is going to be Types.DOUBLE. So now I can just execute the stored procedure and print out the returned value. But the returned value, in this case, is the out parameter from the stored procedure. So let's say something like Total sales is, and then say cs.getDouble. And there's one out parameter, so it's the parameter at index number 1.



And if you run this code now, then the total value of the sales is 19,337.05. So that's the total sales of all gigs in the database.

# In/Out Parameters

=>slides: Pg. 16

So, what about INOUT parameters? So we can have a stored procedure, which has a mixture of IN parameters and OUT parameters. And in this case, we'd use a question mark for each parameter. If we have IN parameters, we set the value for the IN parameter. If we have INOUT parameters, we set a value for the input to that parameter. And in the same way that we'd register any OUT parameters, we'd also register any INOUT parameters as well.

=>slides: Pg. 17

So the stored procedure we're going to use for this is called SetNewPrice. What this tries to do is to update the sales price for a gig. But we set a maximum price we want to update the price to. So we try and set the price, and if the price is higher than the maximum, we leave the price unset, but we change the maxPrice parameter on the way out to be the new price. So it looks like this.

=>slides: Pg. 18

It takes the id of the gig we want to change, it takes an IN parameter, which is the amount we'd like to update the gig by, and this is a percentage, we might want to increase the price by 10%, and then it takes a maxPrice and that's an INOUT parameter. On the way in, this is the maximum price that we'll accept. So we try and update the price. If the proposed price is less than the maximum price, then we update the price of the tickets inside the table and we set the maximum price on the way out to this new proposedprice. If the proposedprice is greater than or equal to the maximum price, we don't update the table, and we set the maximum price on the way out to be the price that's currently in the table.

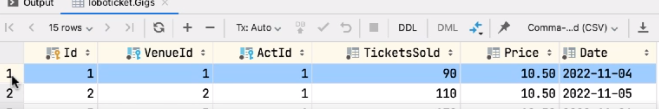
=>slides: Pg. 19

So we call it something like this. We define the SQL, and then we have three question marks, two for the two IN parameters and one for the INOUT parameter. We prepare the call, we set the value for the first parameter, which is gig id 1, we set the value for the second parameter, which is a percentage, so that's 0.1, and then we set the value for the INOUT parameter. This is the maximum price that we'll accept. We then register this third parameter as an OUT parameter. So we set the value for the way in, and we tell the driver what to expect on the way out. We then execute the query, and then in the same way as we did for the out parameters, we can then get the data from this INOUT parameter. So let's write the code and see how this works.

# Demonstration - In/Out Parameters

=>slides: Pg. 20

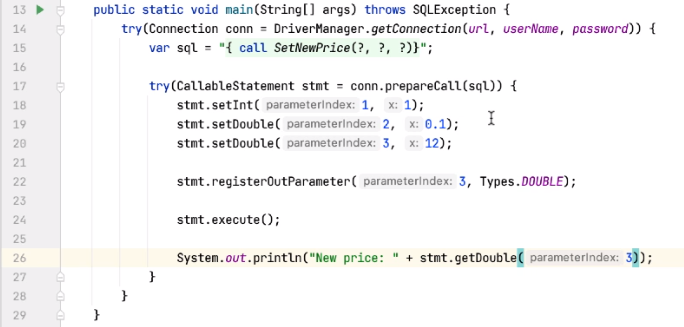


So remember, we're going to use a stored procedure that's going to attempt to raise the price of a gig, 

and the gig we're going to use is with the Id of 1.



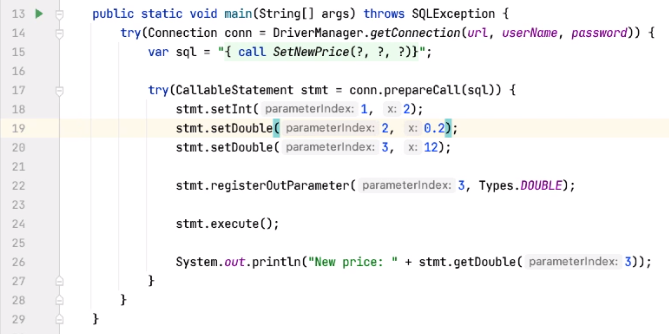
And if we notice, its current price is 10.50. So if we try and raise that by, for example, 10%, then that will raise it by 1.05 and give us a gig price of 11.55. So let's see what happens when we try and do that.



So, again, the usual process, we define the sql, again the braces, and then call the stored procedures called SetNewPrice, and it has three parameters, so there's three question marks. We create our try, pass in the sql, and then set these three values. So stmt.setInt for the first one, and remember, this is the gig Id, so gig Id 1; cs.setDouble for the second one, and this is the amount we want to raise the price by, so 10%; and then for the third one, let's set a maximum price, and again, that's a Double, and first of all, let's set a max price of 12 and see what we get back. So, remember, this third parameter is an In/Out parameter. So we've set the value for the InParameter, we now need to register it as an OutParameter. So let's say stmt.registerOutParameter, and it's parameter 3 in this case, and the types will be DOUBLE. So I can now execute the query, so I can say stmt.execute, and then I can get this value. So we say System.out.println ("New Price"), and then to get the value we call stmt.getDouble and it's parameterIndex 3. So you're raising this by 10%, so that should be within the range. So we've set the maximum price of 12, and the new price should be less than 12, so we should see that new price. 

So let's try that, and sure enough, it comes back with a new price of 11.55.

So what about gig number 2? So gig number 2 has the same price, it's 10.5.



So let's say for gig number 2 we try and raise that by 20%, and we set the maximum at 12 still. So 20% should take us over the maximum value, so let's see what we get back then. So now when we run this, we should get back the original price, not the new price, as the proposed price is higher than the maximum that we've set.



So let's run the code, and sure enough, we get back a new price of 10.5. So here we've reset the OutParameter, and we've reset it to a value that was stored in the database, not the calculated values we did the first time we made this call.

# JDBC Stored Procedures with Exception Handling

=>slides: Pg. 21

To declare a handler, you use the  DECLARE HANDLER statement as follows:

DECLARE action HANDLER FOR condition\_value statement;

Code language: SQL (Structured Query Language) (sql)

If a condition whose value matches the  condition\_value , MySQL will execute the statement and continue or exit the current code block based on the action .

The action accepts one of the following values:

* CONTINUE :  the execution of the enclosing code block ( BEGIN … END ) continues.
* EXIT : the execution of the enclosing code block, where the handler is declared, terminates.

The  condition\_value specifies a particular condition or a class of conditions that activate the handler. The  condition\_value accepts one of the following values:

* A MySQL error code.
* A standard SQLSTATE value. Or it can be an SQLWARNING , NOTFOUND or SQLEXCEPTION condition, which is shorthand for the class of SQLSTATE values. The NOTFOUND condition is used for a [cursor](https://www.mysqltutorial.org/mysql-cursor/" \o "MySQL Cursor)or  SELECT INTO variable\_list statement.
* A named condition associated with either a MySQL error code or SQLSTATE value.

The statement could be a simple statement or a compound statement enclosing by the BEGIN and END keywords.

## MySQL error handling examples

Let’s take some examples of declaring handlers.

The following handler set the value of the  hasError variable to 1 and continue the execution if an SQLEXCEPTION occurs

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET hasError = 1;

Code language: SQL (Structured Query Language) (sql)

The following handler rolls back the previous operations, issues an error message, and exit the current code block in case an error occurs. If you declare it inside the BEGIN END block of a stored procedure, it will terminate the stored procedure immediately.

DECLARE EXIT HANDLER FOR SQLEXCEPTIONBEGIN

ROLLBACK;

SELECT 'An error has occurred, operation rollbacked and the stored procedure was terminated';END;

Code language: SQL (Structured Query Language) (sql)

The following handler sets the value of the  RowNotFound variable to 1 and continues execution if there is no more row to fetch in case of a [cursor](https://www.mysqltutorial.org/mysql-cursor/" \o "MySQL Cursor)or SELECT INTO statement:

DECLARE CONTINUE HANDLER FOR NOT FOUND SET RowNotFound = 1;

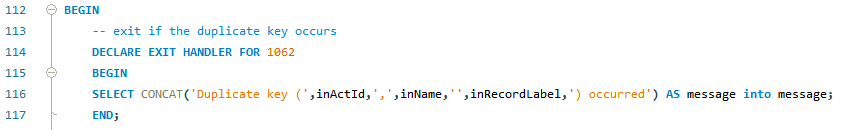
Code language: SQL (Structured Query Language) (sql)

If a duplicate key error occurs, the following handler issues an error message and continues execution.

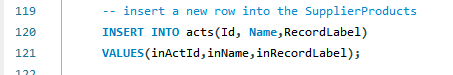
DECLARE CONTINUE HANDLER FOR 1062SELECT 'Error, duplicate key occurred';

**How it works.**

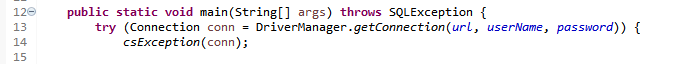
The following exit handler terminates the stored procedure whenever a duplicate key occurs (with code 1062). In addition, it returns an error message.

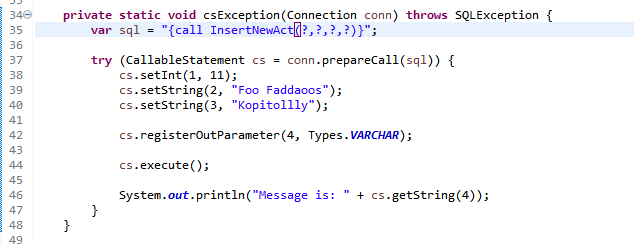


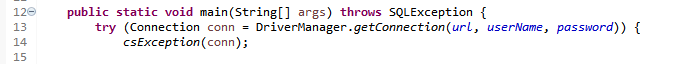
This statement [inserts a row](https://www.mysqltutorial.org/mysql-insert-statement.aspx) into the Acts table. If a duplicate key occurs, the code in the handler section will execute.



Third, call the InsertNewAct() to insert some rows into the Acts table:



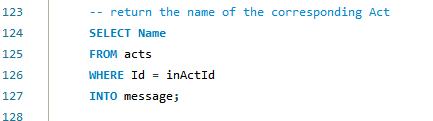
Fourth, attempt to insert a row whose values already exist in the Acts table:



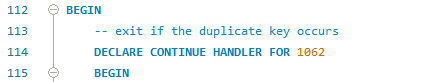
Here is the error message:



Because the handler is an EXIT handler, the last statement does not execute:

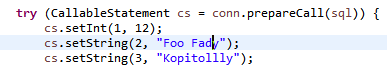


If  you change the EXIT in the handler declaration to CONTINUE , you will also get the number of products provided by the supplier:



Finally, call the stored procedure again to see the effect of the CONTINUE handler:





Here is the output:



# Demonstration: JDBC Stored Procedures with Exception Handling

=>slides: Pg. 22

# Summary

=>slides: Pg. 23

Okay, so in this module, we've seen that we can use a CallableStatement to execute stored procedures. And these have a particular syntax, so we say, brace, call, then the name of the stored procedure, and then a closing brace. They can have IN, OUT, and IN/OUT parameters. We set the IN parameters by name or by column, column numbers are 1 based, the same as they are for PreparedStatement, and then we register the OUT parameters, and we register the type and the position of the OUT parameters. So that's the end of this course. I hope you've enjoyed it, and I hope this helps you pass the exam.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*