就像Connection对象创建Statement和PreparedStatement对象一样，它可使用同样的方式创建CallableStatement对象，该对象将用于执行对数据库存储过程的调用。

创建CallableStatement对象

假设需要执行以下Oracle存储过程 -

```
CREATE OR REPLACE PROCEDURE getEmpName
   (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END;
SQL
```

注意：上面的存储过程是为Oracle编写的，但是如果使用MySQL数据库，参考以下代码为MySQL编写相同的存储过程，如下在EMP数据库中创建它 -

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
   (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END $$

DELIMITER ;
SQL
```

存在三种类型的参数：IN，OUT和INOUT。 PreparedStatement对象只使用IN参数。 CallableStatement对象可以使用上面三种类型参数。

以下是上面三种类型参数的定义 -

参数	描述
IN	创建SQL语句时其参数值是未知的。 使用setXXX()方法将值绑定到IN参数。
OUT	由SQL语句返回的参数值。可以使用getXXX()方法从OUT参数中检索值。
INOUT	提供输入和输出值的参数。使用setXXX()方法绑定变量并使用getXXX()方法检索值。

以下代码片段显示了如何使用Connection.prepareCall()方法根据上述存储过程来实例化一个CallableStatement对象 -

```
CallableStatement cstmt = null;
try {
```

```
   String str = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

String变量str表示存储过程，带有参数占位符。

使用CallableStatement对象与使用PreparedStatement对象很像。 在执行语句之前，必须将值绑定到所有参数，否则将收到一个SQLException异常。

如果有IN参数，只需遵循适用于PreparedStatement对象的相同规则和技术; 使用与绑定的Java数据类型相对应的setXXX()方法。

使用OUT和INOUT参数时，必须使用一个额外的CallableStatement对象方法registerOutParameter()。 registerOutParameter()方法将JDBC数据类型绑定到存储过程预期返回的数据类型。

当调用存储过程后，可以使用适当的getXXX()方法从OUT参数中检索该值。 此方法将检索到的SQL类型的值转换为Java数据类型。

关闭CallableStatement对象

就像关闭其他Statement对象一样，由于同样的原因(节省资源)，还应该关闭CallableStatement对象。

简单的调用close()方法将执行关闭工作。 如果先关闭Connection对象，它也会关闭CallableStatement对象。 但是，应该始终显式关闭CallableStatement对象，以确保正确的顺序清理。

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```

JDBC SQL转义语法

通过使用标准JDBC方法和属性，转义语法使您能够灵活地使用不可用的数据库特定功能。

一般SQL转义语法格式如下 -

{keyword 'parameters'}
以下是以下转义序列，在执行JDBC编程时非常有用 -
d, t, ts关键字
它们用于帮助确定日期，时间和时间戳文字。没有哪两个DBMS表示时间和日期的方式相同。该转义语法告诉驱动程序以目标数据库的格式呈现日期或时间。 例如 -
{d 'yyyy-mm-dd'}
yyyy=年份，mm=月份; dd=日期。 使用这种语法{d'2019-09-03'}表示的是2019年3月9日。
这是一个简单的示例，显示如何将日期插入表中 -
```
//Create a Statement object
stmt = conn.createStatement();
//Insert data ==> ID, First Name, Last Name, DOB
String sql="INSERT INTO STUDENTS VALUES" +
            "(100,'Kobe','Bryant', {d '2002-12-16'})";

stmt.executeUpdate(sql);
```
Java
同样，还可以使用以下两种语法：t或ts -
{t 'hh:mm:ss'}
这里，hh=小时，mm =分钟， ss =秒。 使用这种语法{t '13:30:29'}是1:30:29 PM。
{ts 'yyyy-mm-dd hh:mm:ss'}
这里“d”和“t”是上述两种语法的组合语法来表示时间戳。
escape关键字
escape关键字标识LIKE子句中使用转义字符。 使用SQL通配符%(与0个或多个字符匹配)时很有用。 例如 -
```
String sql = "SELECT symbol FROM MathSymbols
              WHERE symbol LIKE '\%' {escape '\'}";
stmt.execute(sql);
```
Java
如果使用反斜杠字符(\)作为转义字符，则还必须在Java字符串文字中使用两个反斜杠字符，因为反斜杠也是Java转义字符。
fn 关键字
这个关键字表示DBMS中使用的标量函数。 例如，可以使用SQL函数长度来获取字符串的长度 -
{fn length('Hello World')}
Java
上面语句返回结果值为：11，也就是字符串’Hello World‘的长度。
call 关键字
此关键字用于调用存储过程。 例如，对于需要IN参数的存储过程，请使用以下语法 -
{call my_procedure(?)};
Java
对于需要IN参数并返回OUT参数的存储过程，请使用以下语法 -

{? = call my_procedure(?)};

Java

oj关键字

此关键字用于表示外部连接。 语法如下 -

{oj outer-join}

Java

这里，*outer-join = table {LEFT/RIGHT/FULL} OUTERJOIN {table | outer-join}* 搜索条件。 例如 -

```
String sql = "SELECT Employees
              FROM {oj ThisTable RIGHT
              OUTER JOIN ThatTable on id = '100'}";
stmt.execute(sql);
```