

# JavaScript SQL Adapter

## Overview

An IBM MobileFirst Foundation SQL adapter is designed to communicate with any SQL data source. You can use plain SQL queries or stored procedures.

To connect to a database, JavaScript code needs a JDBC connector driver for the specific database type. You must download the JDBC connector driver for the specific database type separately and add it as a dependency in your project. For more information on how to add a dependency, see the Dependencies section in the Creating Java and JavaScript Adapters (../creating-adapters/#dependencies) tutorial.

In this tutorial and in the accompanying sample, you learn how to use a MobileFirst adapter to connect to a MySQL database.

**Prerequisite:** Make sure to read the JavaScript Adapters (../) tutorial first.

## The XML File

The XML file contains settings and metadata.

1. In the adapter XML file, declare the following parameters:

- JDBC Driver Class
- Database URL
- Username
- Password

```
<?xml version="1.0" encoding="UTF-8"?>
<mfp:adapter name="JavaScriptSQL"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:mfp="http://www.ibm.com/mfp/integration"
  xmlns:sql="http://www.ibm.com/mfp/integration/sql">

  <displayName>JavaScriptSQL</displayName>
  <description>JavaScriptSQL</description>
  <connectivity>
    <connectionPolicy xsi:type="sql:SQLConnectionPolicy">
      <dataSourceDefinition>
        <driverClass>com.mysql.jdbc.Driver</driverClass>
        <url>jdbc:mysql://localhost:3306/mobilefirst_training</url>
        <user>mobilefirst</user>
        <password>mobilefirst</password>
      </dataSourceDefinition>
    </connectionPolicy>
  </connectivity>
</mfp:adapter>
```

2. Declare a procedure in the adapter XML file.

```
<procedure name="getAccountTransactions1"/>
```

# JavaScript implementation

The adapter JavaScript file is used to implement the procedure logic.

There are two ways of running SQL statements:

- SQL statement query
- SQL stored procedure

## SQL statement query

1. Assign your SQL query to a variable. This must always be done outside the function scope.
2. Add parameters, if necessary.
3. Use the `MFP.Server.invokeSQLStatement` method to call prepared queries.
4. Return the result to the application or to another procedure.

```
// 1. Assign your SQL query to a variable (outside the function scope)
// 2. Add parameters, if necessary
var getAccountsTransactionsStatement = "SELECT transactionId, fromAccount, toAccount, transactionDate, transactionAmount, transactionType " +
"FROM accounttransactions " +
"WHERE accounttransactions.fromAccount = ? OR accounttransactions.toAccount = ? " +
"ORDER BY transactionDate DESC " +
"LIMIT 20;";

// Invoke prepared SQL query and return invocation result
function getAccountTransactions1(accountId){
    // 3. Use the `MFP.Server.invokeSQLStatement` method to call prepared queries
    // 4. Return the result to the application or to another procedure.
    return MFP.Server.invokeSQLStatement({
        preparedStatement : getAccountsTransactionsStatement,
        parameters : [accountId, accountId]
    });
}
```

## SQL stored procedure

To run a SQL stored procedure, use the `MFP.Server.invokeSQLStoredProcedure` method. Specify a SQL stored procedure name as an invocation parameter.

```
// Invoke stored SQL procedure and return invocation result
function getAccountTransactions2(accountId){
    // To run a SQL stored procedure, use the `MFP.Server.invokeSQLStoredProcedure` method
    return MFP.Server.invokeSQLStoredProcedure({
        procedure : "getAccountTransactions",
        parameters : [accountId]
    });
}
```

## Using multiple parameters

When using either single or multiple parameters in an SQL query make sure to accept the variables in the function and pass them to the `invokeSQLStatement` or `invokeSQLStoredProcedure` parameters in an **array**.

```

var getAccountsTransactionsStatement = "SELECT transactionId, fromAccount, toAccount, transactionDate, transactionAmount, transactionType " +
  "FROM accounttransactions " +
  "WHERE accounttransactions.fromAccount = ? AND accounttransactions.toAccount = ? " +
  "ORDER BY transactionDate DESC " +
  "LIMIT 20;";

//Invoke prepared SQL query and return invocation result
function getAccountTransactions1(fromAccount, toAccount){
  return MFP.Server.invokeSQLStatement({
    preparedStatement : getAccountsTransactionsStatement,
    parameters : [fromAccount, toAccount]
  });
}

```

## Invocation Results

The result is retrieved as a JSON object:

```

{
  "isSuccessful": true,
  "resultSet": [{
    "fromAccount": "12345",
    "toAccount": "54321",
    "transactionAmount": 180.00,
    "transactionDate": "2009-03-11T11:08:39.000Z",
    "transactionId": "W06091500863",
    "transactionType": "Funds Transfer"
  }, {
    "fromAccount": "12345",
    "toAccount": null,
    "transactionAmount": 130.00,
    "transactionDate": "2009-03-07T11:09:39.000Z",
    "transactionId": "W214122V5337",
    "transactionType": "ATM Withdrawal"
  }]
}

```

- The `isSuccessful` property defines whether the invocation was successful.
- The `resultSet` object is an array of returned records.
  - To access the `resultSet` object on the client-side: `result.invocationResult.resultSet`
  - To access the `resultSet` object on the server-side: `result.ResultSet`

## Sample adapter

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/Adapters>) the Adapters Maven project.

The Adapters Maven project includes the **JavaScriptSQL** adapter described above. Also included is an SQL script in the **Utils** folder.

## Sample usage

- Run the .sql script in your SQL database.

- Make sure that the `mobilefirst@%` user has all access permissions assigned.
- Use either Maven, MobileFirst CLI or your IDE of choice to build and deploy the JavaScriptSQL adapter (`../../creating-adapters/`).
- To test or debug an adapter, see the testing and debugging adapters (`../../testing-and-debugging-adapters`) tutorial.

When testing, the account value should be passed in an array: `["12345"]`.