

## Tutorial: Aster Data Developer Express – Word Count Function

Ade\_1\_WordCountFunction

---

### Introduction:

Aster Data Developer Express (“ADE”) is an integrated development environment for creating, testing and deploying SQL-MapReduce applications in Java. The ADE is delivered through integration with the Eclipse Integrated Development Environment.

### Objectives:

This tutorial explains how to use Aster Data Developer Express (“ADE”) to build, test, and deploy a SQL-MapReduce application that performs the first part of a word count function, the map component, which splits strings into individual words.

### Prerequisites:

Prior to starting this tutorial you should download and install:

1. [Eclipse version 3.5.1](#)
2. [Aster Data Developer Express Plug-In](#)

### Sample Files:

None.

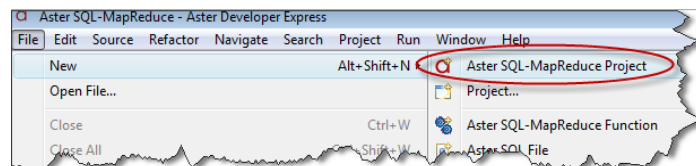
### Part 1: Create Your Aster Project in ADE

The ADE project serves two purposes:

- It's the container that holds your SQL-MapReduce code, related libraries, and tests.
- If you have an *n*Cluster implementation (not required for this tutorial) it provides nCluster connection details to connect to your database(s).

#### *To create your project:*

1. Start Eclipse and select the command File: New: Aster SQL-MapReduce Project.



2. The first screen of the New Project Wizard appears. Type a Project name (use letters, numerals, and hyphens only). Click **Finish**.
3. Expand the WordCount project in Eclipse's Package Explorer in the left pane. You will notice that all the libraries required for SQL-MapReduce development have been automatically created and placed there.

### Part 2: Create Your SQL-MapReduce Function in ADE

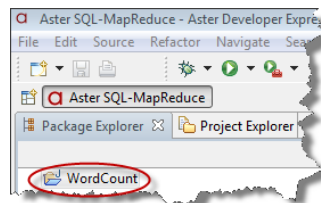
There are two main steps in creating a new function:

- Create the SQL-MapReduce Function Stub Code, below.
- Implement Your operateOn...() Method

#### *Create the SQL-MapReduce Function Stub Code*

To create the SQL-MapReduce function stub code:

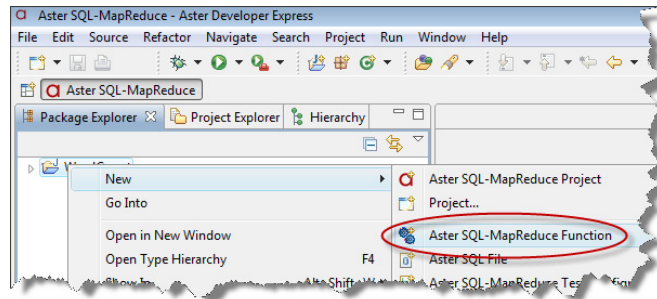
1. In Eclipse's Package Explorer, click on the name of your ADE project.



3. Choose **New: Aster SQL-MR Function**.

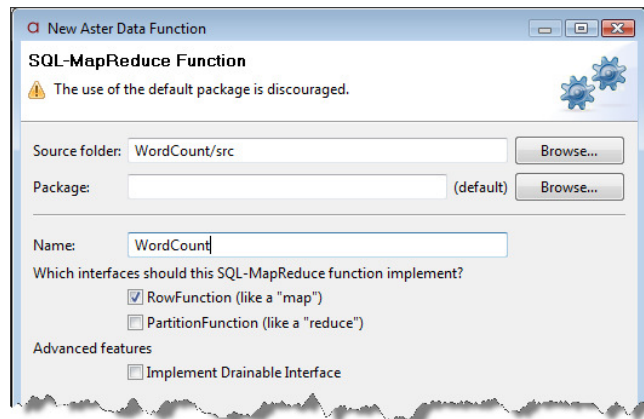
## Tutorial: Aster Data Developer Express – Word Count Function

Ade\_1\_WordCountFunction

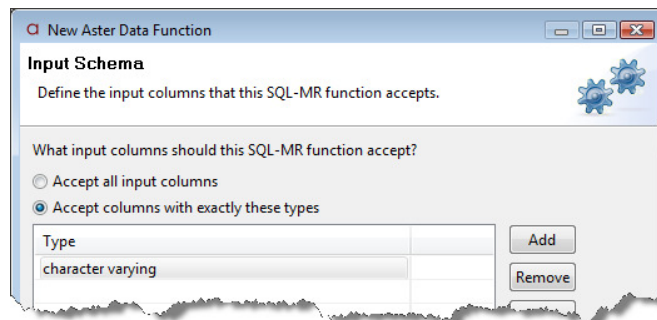


This launches the *ADE Function Wizard*, which will ask you a number of questions about what sort of function you want to create.

3. In the first window of the function wizard,
  - Type a **Name** for the function (we'll call our example, "WordCount").
  - Click **RowFunction** or **PartitionFunction** to specify the type of function. We are creating a Map Function for WordCount, which is also a RowFunction. Click the box next to **RowFunction**. See "[Introduction to SQL-MapReduce Functions](#)" for an explanation of row and partition functions.
  - Click **Next**.



4. In the second wizard window, *Input Schema*, you specify the types of columns your function can take as input.



With WordCount we want the ability to take in a line as a string for input. Choose **Accept columns with exactly these** types (you can add restrictions in your SQL-MapReduce Java code later). Then click **Add** to add a column. By default columns are

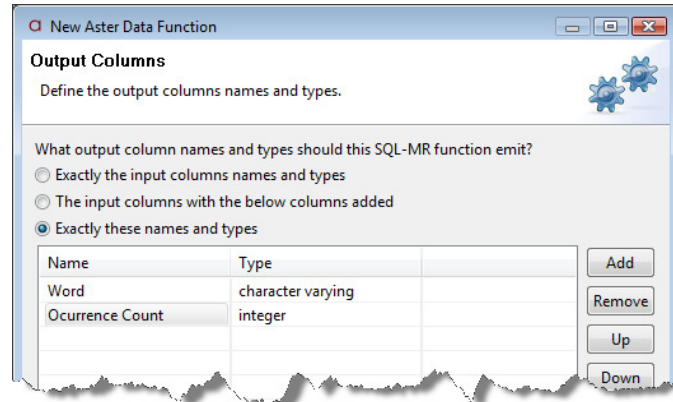
## Tutorial: Aster Data Developer Express – Word Count Function

Ade\_1\_WordCountFunction

---

assigned the integer datatype. So double click on integer to re-define the datatype by selecting from the drop-down list **character varying** (this is the SQL-equivalent of a string). Click **Next**.

5. In the third wizard screen, *Output Columns*, you choose your output columns and types.



You can choose to:

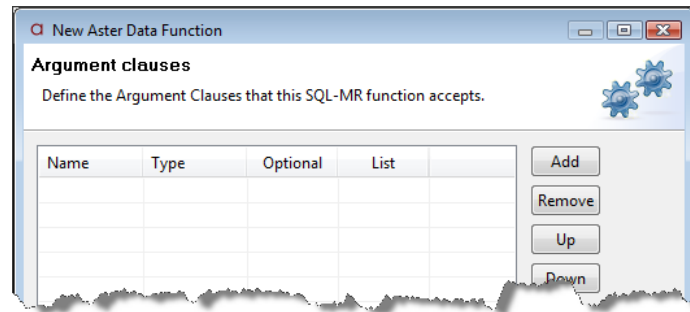
- return all the input columns, meaning the types and names of the output columns match those of the input (choose **Exactly the input columns' names and types**); or
- return all the input columns *plus additional columns*, (choose **The input columns with the below columns added**); or
- it can return a set of columns you specify (choose **Exactly these names and types**). If you choose this option, the output columns need not match the names or types of the input columns.

In this example, we'll choose the third option, because in the classic WordCount example, the Map function provides as output each word and its occurrence count as the integer "1". To do this we check the third checkbox, **Exactly these names and types**, and click the **Add** button. In the row that appears, click on the **Name** field and edit it. This column will hold the word that is being counted, so call it something like word. Then click on the **Type** field and choose the datatype. The word will be presented as a string so, change its type to be **Character varying**. You also need to add a second output column to receive the occurrence count of each word. To do this, again click the **Add** button. In the row that appears, click on the **Name** field and edit it. This column then corresponds to each word being output, so call it something like occurrence count.. Its default type is **Integer**, which is the correct type for the occurrence count.

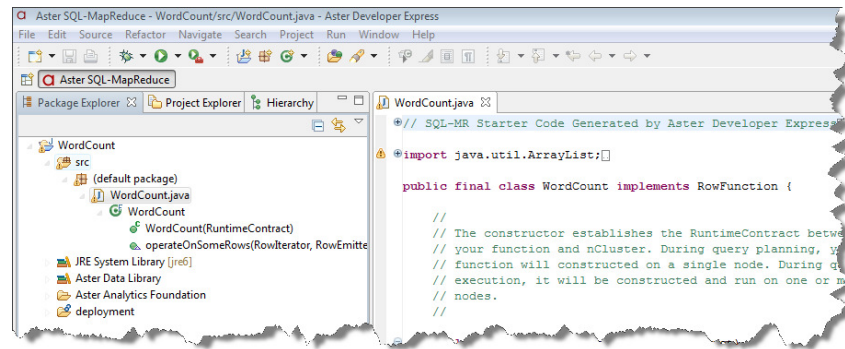
6. Click **Next**.
7. In the next wizard screen, *Argument clauses*, you can set up your function to accept arguments from the query that calls the function. These are typically parameter settings that you provide so that the user can specify how the function should operate, in the context of his or her query. Since the WordCount Map function requires no parameters for its execution, we leave this screen as is.

## Tutorial: Aster Data Developer Express – Word Count Function

Ade\_1\_WordCountFunction



8. Click **Finish**. ADE generates your stub code, which is visible in the Package Explorer.



9. In Eclipse, click the save icon or click **File: Save** to save your function stub code.

### ***Implement Your operateOn...() Method***

Your stub code contains two important methods:

- The constructor specifies mapping between input and output columns. In most cases you can use the constructor as-is.
- The row function (`operateOnSomeRows()`) or partition function (`operateOnPartition()`) is where you will implement the core logic of your SQL-MapReduce function. Your main task in building an SQL-MapReduce function is to implement this function, which is a row function in our example.

Implement the `operateOnSomeRows()` or `operateOnPartition()` function:

1. In Eclipse, open your SQL-MapReduce function source file for editing. In this example, the file is called `WordCount.java`.
2. In the row or partition function, find this while loop and implement your function there.

```
while (inputIterator.advanceToNextRow()) {}
```

Do the following:

- a. Delete the `ClientVisibleException`. This is provided as a hint that the implementation has not yet been done. Delete the following lines:

```
throw new ClientVisibleException(  
    "operateOnSomeRows() needs to be implemented!");
```

- b. Where you deleted the exception code, write the implementation of your function. For example, below is what our sample **word count row function** looks like.

## Tutorial: Aster Data Developer Express – Word Count Function

Ade\_1\_WordCountFunction

---

**CAUTION:** Do not attempt to cut and paste this code from the tutorial. You will need to type the java code directly into Eclipse.

```
public void operateOnSomeRows(RowIterator inputIterator,
                               RowEmitter outputEmitter) {
    /*
     * // This method will be called once for each set of rows.
     */

    String a;
    StringTokenizer tokenizer;
    while (inputIterator.advanceToNextRow()) {
        /*
         * // Construct output rows here using calls like outputEmitt-
         * ter.addInt(),
         * //   outputEmitter.addString(), or outputEmitt-
         * ter.addFromRow(). Once
         * //   finished building the row, call outputEmitt-
         * ter.emitRow().
         * //
         * //
         * // Add all the columns from the RowIterator to the RowE-
         * mitter.
         */

        a=inputIterator.getStringAt(0);
        tokenizer=new StringTokenizer(a);
        while(tokenizer.hasMoreTokens())
        {
            outputEmitter.addString(tokenizer.nextToken());
            outputEmitter.addShort((short)1);
            outputEmitter.emitRow();
        }
    }
}
```

## Part 3: Test Your SQL-MapReduce Function in ADE

ADE includes a feature called TestRunner that lets you create test file in Eclipse that can test your code locally, without an nCluster connection. There are three main steps in this section:

- Create Your Tests, below.
- Create Your Input Data
- Run Your Test

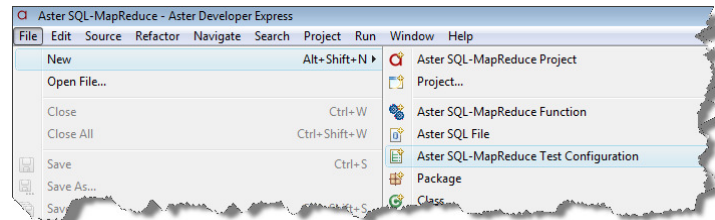
### *Create Your Tests*

Create your tests in TestRunner as follows:

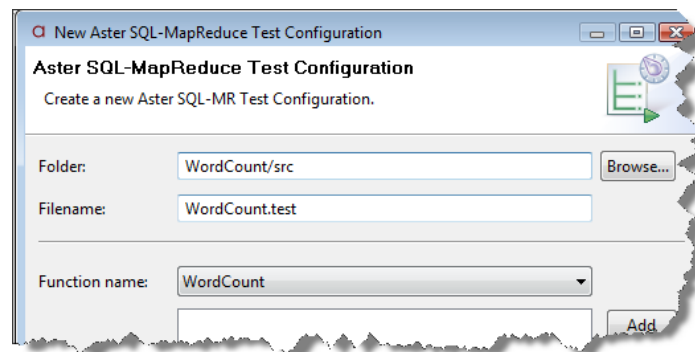
1. From the menu, select the menu command **File**. Then **New: Aster SQL-MapReduce Test Configuration**.

## Tutorial: Aster Data Developer Express – Word Count Function

Ade\_1\_WordCountFunction



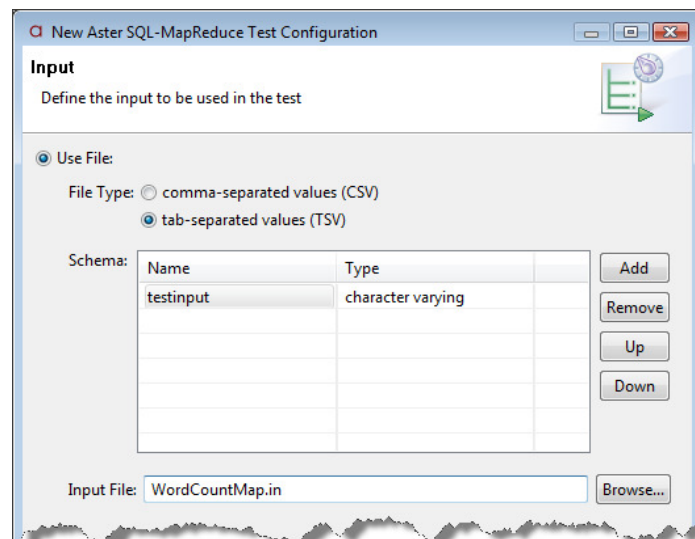
2. The New Test Configuration wizard appears. In the first window of the wizard, you can typically accept the default values. The **Folder** is where the test will be saved, the **Filename** is the name of the file with test data and settings, and the **Function name** is the name you gave your SQL-MapReduce function (not an individual method, but the class) in In the first window of the function wizard, .



If your function takes arguments, click the **Add** button next to the **Argument Clauses** list and add a name/value pair for each argument your test will pass to the function. The Word Count example does not take arguments.

Click **Next**.

3. In the second screen of the wizard, *Input*, we define the format and source of the input data the test will use. We can choose to use a static data file, or we can choose to consume the output of another test as our input.



## Tutorial: Aster Data Developer Express – Word Count Function

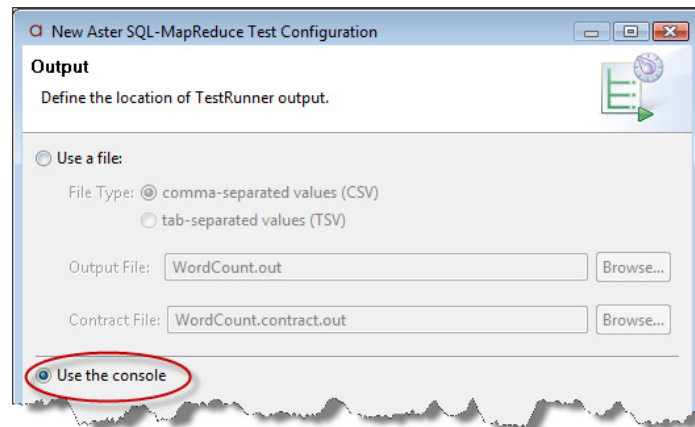
Ade\_1\_WordCountFunction

For this example, we'll use a static, tab separated data input file. Click **Use File**, and for **File Type**, choose **TSV**.

Our example function, word count, takes as input a line as a string column, we can set the schema to comprise of exactly 1 column of type **Character Varying**. Click **Add**

Then, provide the name of the data file, say WordCountMap.in. Click **Next**.

4. In the third screen of the wizard, *Output*, we specify the format and destination of the test output. You can direct the output to a file or to the Eclipse console.



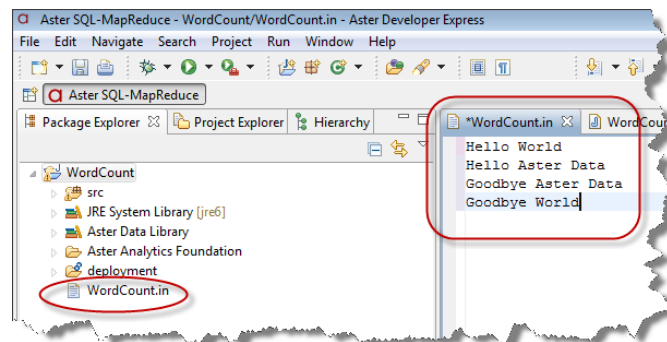
For this example, we'll send the results to the console.

5. Click **Finish**. ADE generates
  - the test file (its name ends in the `.test` suffix) and
  - if specified, the input data file (its name ends in the `.in` suffix).

These files are displayed in the Eclipse workspace.

### Create Your Input Data

If your test uses an input data file, you simply edit that file in Eclipse. In this example, find `WordCount.in` at the bottom of the Package Explorer tree, double-click it to edit it, type your test input values in the editing pane, and click **Save**.



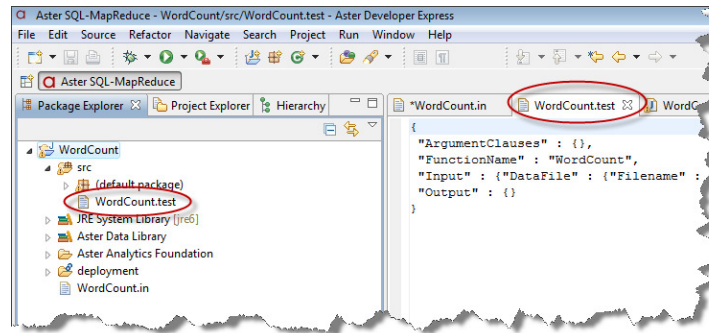
## Tutorial: Aster Data Developer Express – Word Count Function

Ade\_1\_WordCountFunction

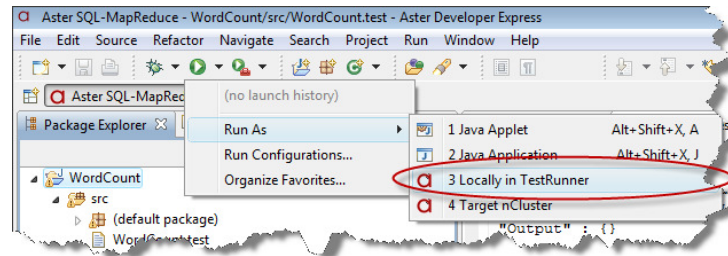
### Run Your Tests

To check your SQL-MapReduce function, run the test in Eclipse as follows.

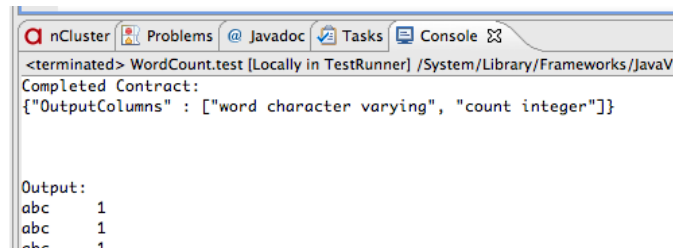
Find your test in Eclipse. In this example, we click on WordCount.test in the src part of the Package Explorer tree.



1. In the Run menu, choose **Run As: Locally in TestRunner**.



The results of the test appear in the Console near the bottom of the Eclipse window.



In this example, the `WordCount` function provides as output each word and its occurrence count as the integer “1”. This is the output expected for the Map function portion of a complete WordCount application. To complete the Word Count application, the Map function needs to be used in conjunction with a Reduce function that aggregates the result of the Map function.

Since the Reduce function required here is a simple aggregation, the power of SQL-MapReduce allows for the usage of standard SQL syntax to perform the Reduce function as a replacement for procedural java code. A SQL GROUP BY statement will suffice.

## Tutorial: Aster Data Developer Express – Word Count Function

Ade\_1\_WordCountFunction

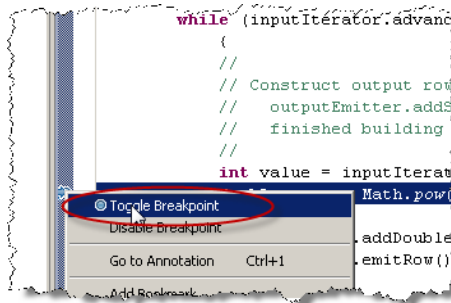
---

The syntax would look like:

```
SELECT word, SUM (occurrence) as frequency FROM WordCount (  
ON (select line from book_table)  
)T  
GROUP BY word
```

### Debugging

When testing your function locally with a `.test` file, you can use the debugging tools in Eclipse. For example, you can right-click in the margin to set a breakpoint



And then debug using your test file.

Running with breakpoints launches the Debug perspective in Eclipse, where you can step through your code to pinpoint errors.

### Summary

At the end of this tutorial the participants understand how to:

- Create your Aster project
- Create your SQL-MapReduce function
- Test your SQL-MapReduce function

### Questions & Clarifications

If you need additional information or have any feedback/comments, please let us know at [info@asterdata.com](mailto:info@asterdata.com).