
SlamData Documentation

Release 2.5

SlamData

July 12, 2016

1	Users Guide	3
1.1	Introduction	3
1.2	Launching SlamData	3
1.3	Key Concepts	3
1.4	Notebooks	4
1.5	Cell Types	4
1.6	Custom Styling	12
1.7	Importing Data	12
1.8	Exporting Data	13
2	SlamData Tutorial	17
2.1	Introduction	17
2.2	Setup	17
2.3	Create the Notebook	18
2.4	Explore the Data	18
2.5	Create a Static Form	19
2.6	Create a Dynamic form	20
2.7	Query Based on a Form	21
2.8	Chart the Results	21
2.9	Put It All Together	22
3	Securing SlamData Community Edition	25
3.1	Introduction	25
3.2	Assumptions	25
3.3	Architecture Overview	26
3.4	Setting Up The Backend	26
3.5	Installing Nginx	29
4	Troubleshooting FAQ	33
4.1	Configuration File Locations	33
4.2	Log File Locations	33
4.3	SlamData Won't Start	34
4.4	SlamData URL	34
4.5	How do I see which version I'm running?	34
4.6	SlamData on Amazon EC2 / Microsoft Azure, etc.	34
4.7	SlamData Notebook Won't Delete	34
5	Reference - SQL²	37

5.1	Introduction	37
5.2	Data Types	37
5.3	Clauses, Operators, and Functions	38
5.4	Syntax Changes in Version 2.5	38
5.5	Basic Selection	39
5.6	Filtering a Result Set	40
5.7	Numeric and String Operations	40
5.8	Dates and Times	41
5.9	Grouping	42
5.10	Nested Data and Arrays	43
5.11	Pagination and Sorting	44
5.12	Joining Collections	45
5.13	Conditionals and Nulls	46
5.14	Database Specific Notes	46
6	Reference - SlamDown	49
6.1	Introduction	49
6.2	Block Elements	49
6.3	Inline Elements	51
6.4	Form Elements	53
7	Reference - SlamData API	59
7.1	Introduction	59
7.2	Assumptions	59
7.3	Reference	59
8	Administration Guide - Community Edition	75
8.1	Prerequisites	75
8.2	Limitations	75
8.3	Installing SlamData	75
8.4	Launching SlamData	76
8.5	Connecting to a Database	77
8.6	Advanced Configuration	78
9	Indices and tables	85

Contents:



Users Guide

1.1 Introduction

This Users Guide can assist with daily usage of SlamData. For information on how to install and configure SlamData see the SlamData Administration Guide





1.2 Launching SlamData

Starting SlamData on a local system will automatically open a new browser window or tab with this URL:
`http://localhost:20223/slamdata/index.html`

If SlamData is not installed locally but instead is on a remote system it can be accessed with a similar URL:
`http://servername:20223/slamdata/index.html` where **servername** is the DNS name or IP address of the server.

1.3 Key Concepts

It is useful to understand the following key concepts when using SlamData.

Feature Name	Image	Description
Cluster		A cluster represents a database server
Folder		A folder represents a database
File		A file represents a table or collection
Notebook		A notebook contains a user's work

A Cluster (server) may contain 0, 1 or more Folders (databases).

A Folder (database) may contain 0, 1 or more Files (tables or collections).


Clicking on a Cluster, Folder, File or Notebook will display its contents.

1.4 Notebooks

Notebooks capture data workflows in a visual fashion that allows you to query data, transform and visualize it in the form of charts and reports.

Each stage in a Notebook's workflow is called a *cell* or *card*. Cells can rely on data from previous cells or exist independent of other cells.






A Notebook can be created in one of two ways:

1. By clicking on the Notebook  icon in the upper right of the SlamData UI.
2. Clicking on a collection and renaming the subsequently displayed Notebook something other than `Untitled Notebook`.

Whenever an existing Notebook is changed it is automatically saved and can be referred to in the future. For instance if a Notebook contains an exploration cell followed by a query cell, the query can be changed and executed and the Notebook is then automatically saved. This allows a user to work in a Notebook without fear of losing work or data.

1.5 Cell Types

To add a new cell to a Notebook click on the  icon and select one of the following cell types

Cell Type	Image	Description
Exploration		Browse data in a table or collection
Query		Leverage SQL2 for powerful queries
Search		Simple searching for non-technical users
SlamDown		Create static or interactive forms
API		Developers pass values into queries for dynamic results

1.5.1 Exploration Cell

An example exploration cell is shown below. Refer to the table below the image for the function of each icon surrounding the cell.

The screenshot displays the SlamData application interface. At the top, a dark header bar contains a back arrow (6), the 'SLAMDATA' logo, and the notebook title 'Untitled Notebook' (7). Below this is a navigation bar with 'Notebook', 'Insert', and 'Help' menus. A secondary bar features an 'Explore' button (5) and icons for undo (8), redo (9), and delete (10). A search bar (11) contains the text '/macbook/demo/olympics'. Below the search bar, a status bar shows 'Finished: took 0ms.' with refresh (12) and expand/collapse (13) icons. The main area displays a table with 9 columns: city, country, discipline, event, gender, sport, type, and year. The table lists 11 rows of Olympic data from Chamonix, France, in 1924. On the left, a sidebar (1) contains icons for help (2), search (3), and download (4). At the bottom, a pagination bar shows 'Page 1 of 232' and 'Per page: 10'.

city	country	discipline	event	gender	sport	type	year
Chamonix	FIN	Figure skating	pairs	X	Skating	Silver	1924
Chamonix	AUT	Figure skating	individual	W	Skating	Gold	1924
Chamonix	FRA	Biathlon	military patrol	M	Biathlon	Bronze	1924
Chamonix	FRA	Figure skating	pairs	X	Skating	Bronze	1924
Chamonix	AUT	Figure skating	individual	M	Skating	Silver	1924
Chamonix	GBR	Bobsleigh	four-man	M	Bobsleigh	Silver	1924
Chamonix	FIN	Speed skating	10000m	M	Skating	Silver	1924
Chamonix	FIN	Speed skating	10000m	M	Skating	Gold	1924
Chamonix	FIN	Speed skating	5000m	M	Skating	Gold	1924
Chamonix	GBR	Curling	curling	M	Curling	Gold	1924

Icon #	Purpose
1	Download the cell in CSV or JSON format
2	Create a graphical chart based on this cell's data
3	Simple search on this cell's data
4	Create a query on this cell's data
5	Execute or 'Play' the cell again
6	Go one level up or one level back
7	Double click to rename this Notebook
8	Hide the element which displays the path (useful when publishing Notebooks)
9	Delete this cell and all subsequent cells
10	Drop-down to select a different file path
11	Refresh the cell's data
12	Get HTML and JavaScript code to embed this cell in another web application
13	Displays the schema of the collection or table being viewed

Below is an example of what a nested schema would look like within the exploration cell. In this instance we have an array called `previous_addresses` with several documents, each containing fields `city`, `county`, `latitude`, `longitude`, `state` and `zip_code`.

previous_addresses					
city	county	latitude	longitude	state	zip_code
NEW ORLEANS	ORLEANS	30.032997	-89.882564	LA	70157
WEST ALTON	SAINT CHARLES	38.83275	-90.403416	MO	63386
OAKESDALE	WHITMAN	47.079658	-117.41146	WA	99158

The corresponding JSON would appear like this in the database:

```
...
"previous_addresses": [
  {
    "city": "NEW ORLEANS",
    "longitude": -89.882564,
    "county": "ORLEANS",
    "state": "LA",
    "latitude": 30.032997,
    "zip_code": 70157
  },
  {
    "city": "WEST ALTON",
```

```

    "longitude": -90.403416,
    "county": "SAINT CHARLES",
    "state": "MO",
    "latitude": 38.83275,
    "zip_code": 63386
  },
  {
    "city": "OAKESDALE",
    "longitude": -117.41146,
    "county": "WHITMAN",
    "state": "WA",
    "latitude": 47.079658,
    "zip_code": 99158
  }
  ...

```

1.5.2 SlamDown Cell

Reports and forms are created with a subset of Markdown called SlamDown. SlamDown allows a relatively non-technical user to create interactive forms, charts and reports without understanding HTML or other complicated markup.

For specific syntax see the SlamDown Reference Guide and the [Cheat Sheet](#).




Below is an image of both a SlamDown cell and it's rendering directly following it. As a reminder when you publish a Notebook you can include SlamDown cells, providing users with interactive forms that can directly affect a query and resulting report or chart.


1.5.3 Search Cell

The Search cell allows users to search through entire collections as well as previous search results resulting in a very refined data set. In other words a user can use a search cell to refine results and then use another search cell to refine those results even further; this process can continue until the appropriate results are found.

Default Search

1. Create a new Search cell:

- Click on the gray Search  icon on the left side of an existing exploration cell, or
- Click the Plus  icon and then select the Search  icon.

2. In the new Search cell, type in a search term and click the Play  icon beneath it.

In the example image below notice the term USA was searched for. Also note that the field name was not specified. By default **SlamData will search all fields in all documents**. For very large collections and tables, especially those without proper indexes assigned, this could take some time to complete; however this also provides a very powerful feature to find data that exists but the location is unknown.

Markdown

1 - # Age Report

2 - ## Select state from the drop down below

3

4 There are !`SELECT COUNT(*) AS ct FROM `/macbook/demo/patients` `` docs

5

6 zip = #_____ (78781)

7

8 st = {!`SELECT DISTINCT state FROM `/macbook/demo/patients` ORDER BY state`}

9

10 gender = (x) other () male () female

11

12 ```

13 SELECT * FROM `/macbook/demo`

14 ```

15

16 Run this query below: `SELECT * FROM

17 `/col` ``

18

▶

Finished: took 0ms.

👁️ ↺ ↔ ⏪

?

Age Report

Select state from the drop down below

There are 10000.0 docs

zip

st

gender ☐ other ☒ male ☐ female

SELECT * FROM `/macbook/demo`

Run this query below: SELECT * FROM /col

8

Chapter 1. Users Guide

Search								
<input type="text" value="/macbook/demo/olympics"/> <input type="text" value="USA"/> <input type="button" value="x"/> <input type="button" value="Q"/>								
<input type="button" value="▶"/> Finished: took 0ms. <input type="button" value="👁"/> <input type="button" value="✂"/> <input type="button" value="🔄"/> <input type="button" value="↔"/> <input type="button" value="◀"/>								
<input type="button" value="ⓘ"/> <input type="button" value="Q"/> <input type="button" value="🖼"/> <input type="button" value="⬇"/>	city	country	discipline	event	gender	sport	type	year
	Chamonix	USA	Speed skating	500m	M	Skating	Gold	1924
	Chamonix	USA	Ice Hockey	ice hockey	M	Ice Hockey	Silver	1924
	St. Moritz	USA	Bobsleigh	five-man	M	Bobsleigh	Silver	1928
	St. Moritz	USA	Skeleton	individual	M	Bobsleigh	Gold	1928
	St. Moritz	USA	Figure skating	individual	W	Skating	Bronze	1928
	Lake Placid	USA	Bobsleigh	four-man	M	Bobsleigh	Silver	1932
	St. Moritz	USA	Skeleton	individual	M	Bobsleigh	Silver	1928
	Lake Placid	USA	Bobsleigh	two-man	M	Bobsleigh	Bronze	1932
	Lake Placid	USA	Speed skating	1000m	M	Skating	Gold	1932

Fig. 1.1: Search and Results

Field Specific Search

To limit a search to a specific field prefix the search term with the field name, for example:

```
country:USA
```

Multiple Field Values

To limit a search with multiple fields list them in the search field. For example to find all women who won gold medals in a data set it may appear like this:

```
gender:W type:Gold
```

Mandatory Search

To search all documents that do **not** contain a value the value should be prefixed with the (-) symbol as follows:

```
-Skating
```

Numeric Searches

To search on fields containing numeric values use the following examples.

Range Search

Search for a field `year` whose value is between 1928 and 1932:

```
year:1928..1932
```

NOT Range Search

The opposite of the previous example, this searches for field `year` whose value is **not** between 1928 and 1932:

```
-year:1928..1932
```

Comparison Search

Search for a field `year` whose value is less than 1948. Below we use the < symbol for less than but the > can also be used for greater than:

```
year < 1948
```

Starts With Search

Search for a field `name` whose value starts with Jen:

```
name:Jen*
```

Nested Search

Search all documents which contain a `foo` field which contains a `bar` field which contains the text `baz`:

```
foo:bar:baz
```

Note: A concise set of search examples can also be found in the [SlamData CheatSheet](#)

1.5.4 Query Cell

The Query cell allows users to utilize SQL2 to directly query one or more collections or tables. This is the equivalent of a SQL command line console.

To create a query cell:

- From an empty Notebook click the Plus  icon then click the Query  icon

OR

- From an existing cell click the Query  icon to the left of the cell.


If the first option is selected the user will be presented with an empty Query cell. If the second option is selected the user will be presented with a Query cell that contains a default query, highlighted with colored syntax as shown below:



Fig. 1.2: Query Highlighted

The query can be manipulated in this alternate form but the highlighted text cannot be modified or removed. If the user prefers more control the first option above may be preferred. The Query cell also provides query completion at certain parts of your query as shown below:

The Query cell will also automatically highlight SQL2 keywords as shown the image above. The query itself can be written on a single line (which will not word wrap) or on multiple lines.

When a query is executed by clicking the Play  icon the cell beneath the query cell will show an icon indicating the query is running. When complete the query's results will display below the query.

Note: If a query takes longer than 30 seconds to execute SlamData considers it a timed out query and will result in an error.

For a complete review of SQL2 and example see the [SQL2 Reference Guide](#).

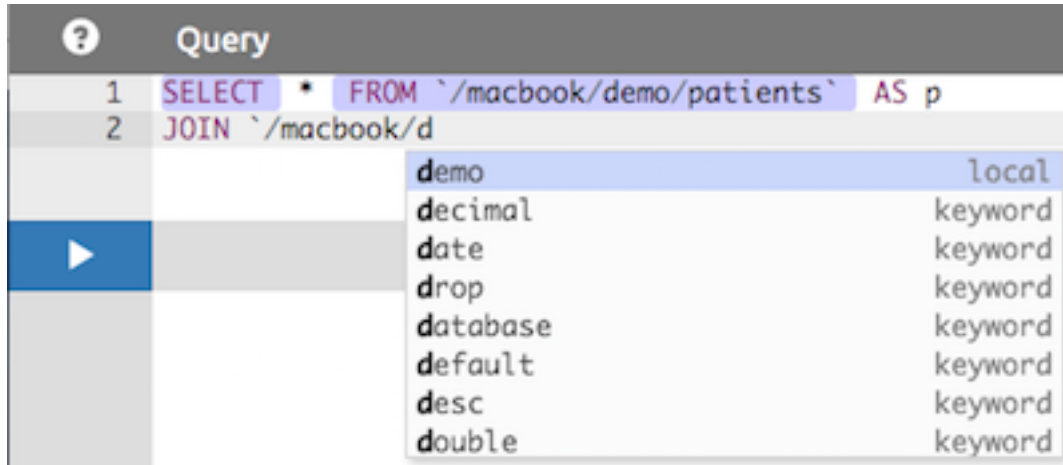


Fig. 1.3: Query Completion

1.6 Custom Styling

Users can add custom styles to notebooks by adding a query parameter to the URI. For example, to add a stylesheet located in `css/foo.css` to

```
http://slamdata.instance.com/notebook.html#/db/Folder/Notebook.slam/view
```

one should modify the route to

```
http://slamdata.instance.com/notebook.html?cssStyleSheets=css/foo.css#/db/Folder/Notebook.slam/view
```

The values of `cssStyleSheets` are decoded and then split by `,`, so to add two stylesheets one could use

- `cssStyleSheets=css/foo.css,http%3A%2F%2Ffoo.com%2Fstyles.css`
- `cssStyleSheets=css%2Ffoo.css,http%3A%2F%2Ffoo.com%2Fstyles.css`
- `cssStyleSheets=css%2Ffoo.css%2Chttp%3A%2F%2Ffoo.com%2Fstyles.css`

These URIs are checked and, if they are valid, corresponding `link` elements are added to the head

Here it would be

```
<link type="text/css" rel="stylesheet" href="css/foo.css">  
<link type="text/css" rel="stylesheet" href="http://foo.com/style.css">
```

1.7 Importing Data


SlamData allows users to import files in both **JSON** and **CSV** format.

JSON files may be formatted either as multiple single documents or within a JSON array.

Note: The first line of CSV files will be used as a *header* line creating the schema that the remaining rows will adhere to.

To upload a file into SlamData, follow these steps:

1. Navigate to the database where the data should be imported.




2. At the top of the page click the Upload File  icon.
3. Select a file from your file system. Large files may take a few moments to upload. After data has been imported a new collection will be created with the same name as the file.
4. A new Untitled Notebook will be created that displays the new collection's data.

1.8 Exporting Data

SlamData allows users to export refined result sets, collections and entire databases.

1.8.1 Result Sets

Once a result set has been refined either through query cells or search cells it may then be downloaded in **JSON** or **CSV** formats.

1. From an exploration cell or results set cell click the Download cell  icon to the left of the cell.
2. In the newly created Download cell select either the CSV  or JSON  icon on the left.
3. Select the appropriate options in the cell.
4. Click Download.

See the example image below of a query cell followed by the results cell.

Downloading the data from this **Results Cell** provides the following JSON export file:

```
[
  {
    "gender": "male",
    "name": "Tory Escobar",
    "addresses": [
      {
        "city": "OROFINO",
        "longitude": -116.184848,
        "county": "CLEARWATER",
        "state": "ID",
        "latitude": 46.4976,
        "zip_code": 83544
      },
      {
        "city": "ARRIBA",
        "longitude": -103.323143,
        "county": "LINCOLN",
        "state": "CO",
        "latitude": 39.316461,
        "zip_code": 80804
      },
      {
        "city": "OLGA",
        "longitude": -122.983742,
```

Query

```

1 SELECT
2   first_name || ' ' || last_name AS name,
3   previous_addresses AS addresses,
4   gender
5 FROM '/macbook/demo/patients'
6 WHERE
7   state = "TX" AND
8   city = "AUSTIN" AND
9   age > 50

```

Finished: took 0ms.

addresses

city	county	latitude	longitude	state	zip_code	gender	name
OROFINO	CLEARWATER	46.4976	-116.184848	ID	83544	male	Tory Escobar
ARRIBA	LINCOLN	39.316461	-103.323143	CO	80804		
OLGA	SAN JUAN	48.557824	-122.983742	WA	98279		
DELRAY BEACH	PALM BEACH	26.454218	-80.13473	FL	33484		
LYONS	LINN	44.749921	-122.594993	OR	97358		
CROSSROADS	LEA	32.690034	-103.209405	NM	88114	female	Damaris Savage
						female	See Harrison

Page 1 of 1 Per page: 10

```

    "county": "SAN JUAN",
    "state": "WA",
    "latitude": 48.557824,
    "zip_code": 98279
  },
  {
    "city": "DELRAY BEACH",
    "longitude": -80.13473,
    "county": "PALM BEACH",
    "state": "FL",
    "latitude": 26.454218,
    "zip_code": 33484
  },
  {
    "city": "LYONS",
    "longitude": -122.594993,
    "county": "LINN",
    "state": "OR",
    "latitude": 44.749921,
    "zip_code": 97358
  }
]
},
{
  "gender": "female",
  "name": "Damaris Savage",
  "addresses": [
    {
      "city": "CROSSROADS",
      "longitude": -103.209405,
      "county": "LEA",
      "state": "NM",
      "latitude": 32.690034,
      "zip_code": 88114
    }
  ]
},
{
  "gender": "female",
  "name": "See Harrison",
  "addresses": []
}
]

```

1.8.2 Collections and Tables


Collections and tables can also be exported in their entirety. When browsing a folder (database) within the SlamData UI simply hover over the file (table or collection) and notice the icons that appear to the right. Click on the Download

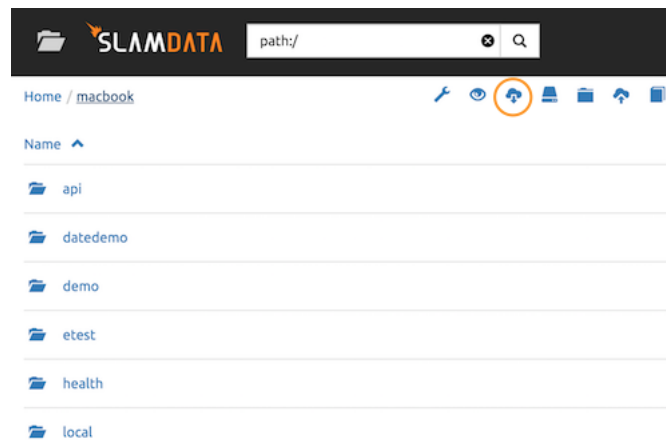


icon. See the image below with the highlighted icon.



1.8.3 Databases

Databases can be exported in their entirety as well. Simply ensure the appropriate database and its underlying collections are displayed in the UI and click on the Download  icon in the **top menu bar** as shown in the image below. A dialog will appear providing several options for the download and will result in a compress zip file containing all of the database's collections as separate files.



SlamData Tutorial


2.1 Introduction

This tutorial will step the user through many of the features in SlamData within the context of a Notebook. The tutorial starts with providing the data needed to perform the steps and continues through the end where a chart will be generated based on a dynamic user form.

2.2 Setup



The examples in the tutorial will reference the path `/local/demo/tutorial` which means a mount point of `/local` is created, which has a database called `demo` and a collection called `tutorial`.

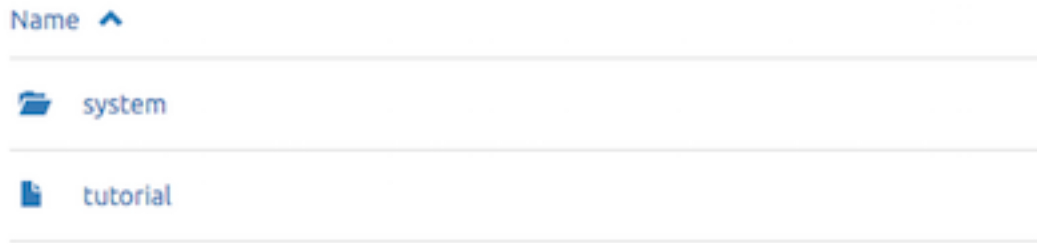
1. [Download](#) and save the tutorial file. This file contains information about which countries won medals in the Olympics as far back as 1924.
2. Start the SlamData application
3. Click on a server that has been configured. See the Connecting to a Database section of the Administration Guide if no servers exist.

4. Click the New Folder  icon in the upper right. A new folder called `Untitled Folder` now exists.
5. Rename the folder to `demo` by hovering the mouse over the `Untitled Folder` and then clicking on the rename icon that appears seen here:

 `Untitled Folder`



6. Click on the renamed folder to open it.
7. Click on the Upload  icon.
8. Locate and select the `tutorial` file in step 1. After a few moments a new Exploration cell will be displayed.
9. Click the Back  icon in the upper left to view the folder again.



Your folder display should appear as follows:

Depending on your server mount name the path to the newly created collection should be similar to `/local/demo/tutorial`.

2.3 Create the Notebook



From the `/local/demo` view, where the `tutorial` file can be seen:

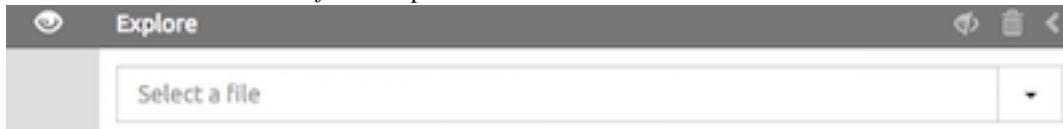
1. Create a Notebook  and rename it `Tutorial Notebook` by double-clicking the title.

Once the Notebook has been renamed it will be saved. At this point the Notebook can be shared in whatever form it currently is saved. Click on the Notebook menu in the top and select `Publish`. A new browser window will open displaying the contents of the Notebook. At this point nothing will be displayed as no cells have been added. The URL in the browser can be shared so others may see the results of the Notebook without modifying it.

2.4 Explore the Data

To get an idea of what the schema looks like and to become familiar with the data a new Exploration cell will now be created.

1. Click on the Plus  icon and then click on the Explore  icon.
2. Click on the *Select a file* drop-down menu and locate the `/local/demo/tutorial` file:



3. Click the Play  icon. Once clicked the exploration cell will appear as below:

Take a moment to view what the published Notebook looks like now. The exploration cell is displayed but the drop-down file selector is not. Again - publishing a Notebook allows others to view the results without modifying anything (unless desired - which will be covered shortly)


Finished: took 0ms. ↺ ↻ ↷

city	country	discipline	event	gender	sport	type	year
Chamonix	FIN	Figure skating	pairs	X	Skating	Silver	1924
Chamonix	AUT	Figure skating	individual	W	Skating	Gold	1924
Chamonix	FRA	Biathlon	military patrol	M	Biathlon	Bronze	1924
Chamonix	FRA	Figure skating	pairs	X	Skating	Bronze	1924
Chamonix	AUT	Figure skating	individual	M	Skating	Silver	1924
Chamonix	GBR	Bobsleigh	four-man	M	Bobsleigh	Silver	1924
Chamonix	FIN	Speed skating	10000m	M	Skating	Silver	1924
Chamonix	FIN	Speed skating	10000m	M	Skating	Gold	1924
Chamonix	FIN	Speed skating	5000m	M	Skating	Gold	1924
Chamonix	GBR	Curling	curling	M	Curling	Gold	1924


⏪ ⏩ Page of 232 ⏴ ⏵ Per page:

The schema shows several fields such as `city`, `country`, `event`, etc. and also has a numeric field `year`. Now that the schema has been viewed along with a sampling of data a SlamDown form will be created to show some of the reporting capabilities of SlamData.

2.5 Create a Static Form

1. Delete the exploration cell by clicking on the Trashcan  icon above the Explore cell.

2. Now create a new SlamDown  cell.

3. Enter the following text into the SlamDown form and then click the Play  icon.

```
# Olympics Report ![Olympics](https://raw.githubusercontent.com/damonLL/slamdemos/master/olympics)
There are !`SELECT COUNT(DISTINCT(country)) FROM `/local/demo/tutorial` `` unique countries in this
There are !`SELECT COUNT(DISTINCT(year)) FROM `/local/demo/tutorial` `` Olympiads in this data set.
There are !`SELECT COUNT(DISTINCT(sport)) FROM `/local/demo/tutorial` `` unique sports in this data
comprised of !`SELECT COUNT(DISTINCT(event)) FROM `/local/demo/tutorial` `` unique events.
```

- See the SlamDown rendering similar to the following image:

Again - take a moment to review the published version of this Notebook. And keep this important note in mind: *All of these queries and reports are executing against a live database. When the data underneath changes, this report will change the next time it is viewed.*

This section described how to create a static form and publish its results. There are many other SlamDown features you can read about in the SlamDown Reference.

Olympics Report

There are **45.0** unique countries inside this data set.

There are **20.0** Olympiads in this data set.

There are **7.0** unique sports in this data set comprised of **67.0** unique events.

The next section describes how to publish a dynamic form that allows user interaction.

2.6 Create a Dynamic form

This section describes how to create a dynamic form that allows users to interact with the data and eventually results in a chart generated from data the user has selected.

The existing Notebook can either be modified or a new Notebook can be created. This tutorial will modify the existing Notebook but users can choose either path.

1. Replace the Markdown code with the following code and click the Play  icon.

```
# Olympics Report ![Olympics](https://raw.githubusercontent.com/damonLL/slamdemofiles/master/olympic-
Years in this dataset range from
!``SELECT DISTINCT(year) FROM `/local/demo/tutorial` ORDER BY year ASC``
to
!``SELECT DISTINCT(year) FROM `/local/demo/tutorial` ORDER BY year DESC``

### Enter start year and end year for report

startyear = #____
endyear = #____
```

Note the pound or hashtag # symbol directly before the underscores _. This tells SlamData that the input field is for numbers. In SlamData input fields are String types by default. The data type is important as the query listed below would not work with String types against a numeric database field.

This results in the following rendered Markdown:

The rendered drop downs of `startyear` and `endyear` create variables that allow users to select data that is then used as query criteria in a query cell.

Olympics Report

Years in this dataset range from 1924.0 to 2006.0


Enter start year and end year for report


startyear

endyear

2.7 Query Based on a Form

In this section the variables defined in the previous section will be used to control the subsequent query.

1. Click on the gray Query  icon to the left of the SlamDown rendered cell. This is **important** as it allows the created variables to bind to the query cell.

2. Enter the following query into the Query cell and click the Play  icon.

```
SELECT
  COUNT(*) as cnt,
  country,
  type
FROM `/local/demo/tutorial`
WHERE
  year >= :startyear AND
  year <= :endyear
GROUP BY country, type
ORDER BY cnt DESC
```

It is important to group the results in the query above so the chart we create in the next section has valid data to work with.

2.8 Chart the Results

This section will display the results in the table from the previous section in chart form.

1. Next to the results cell, click the Visualize  icon to create a new visualization cell.



2. Select the Bar chart icon.

3. Select or enter the values below in the configuration fields:

Field	Value
Category	.country
Measure	.cnt
Series	.type
Height	400
Width	800
Axis Label Angle	45
Axis Font Size	12

This will result in the following chart:

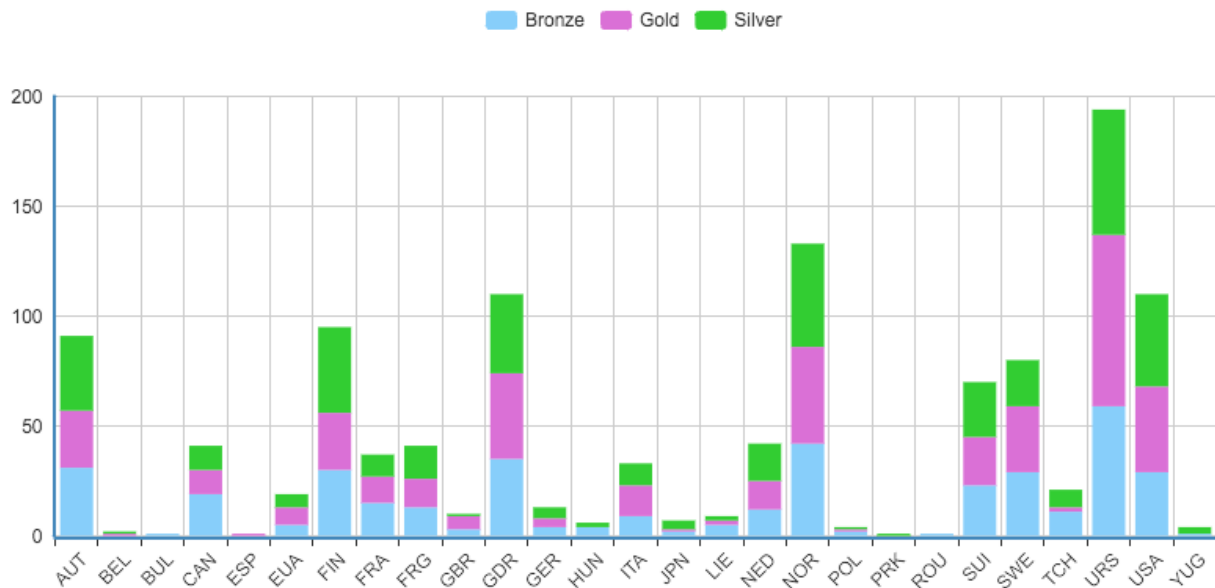



Fig. 2.1: Olympics Chart

The items in the legend at the top (Bronze, Silver, Gold) can be clicked to toggle viewing of that data series.

2.9 Put It All Together

You've learned how to create static and dynamic forms, add variables to queries and generate charts from user input. The only thing left is to publish the Notebook (or re-open the published link) and view how interactive the Notebook is. Because the SlamDown cell allows inputs this Notebook allows controlled input from users to create the charts.

Beyond publishing the Notebook as a whole, each cell can be published or embedded into another web application.

Simply click the Embed  icon and copy the contents (JavaScript and/or URL) and embed them directly into the HTML of the other web application. You can also view our YouTube video which demonstrates this functionality here.

Note: > To allow users to interact with data and have cells update other cells, the user must interact with the published Notebook, not the individual cells. When individual cells are embedded in applications the dynamic dependencies no longer work outside of the Notebook context.



Securing SlamData Community Edition

3.1 Introduction

This guide can assist with configuring NGINX (or other proxy service) and SlamData to limit access based on HTTP authentication and URL paths.

Note: SlamData Advanced includes LDAP/OAuth authentication, fine grained authorization and user auditing - all of the security you need for peace of mind in the enterprise.

SlamData is an all-in-one NoSQL visual analytics system. You can get started right away by simply downloading the software and running it on your local system, or install it on a dedicated server. There is no need to follow this guide if you do not need to limit access to SlamData. If you need to restrict access to the SlamData application to certain users, then read on!

Watching our [YouTube video](#) (~24 minutes) may help as well. The video is based off the instructions in this guide.

There are several methods of restricting access to SlamData. This guide focuses on configuring SlamData to run under Nginx providing basic http authorization, then forwarding requests to the backend either on the same host or a separate host.

This quick guide does not give detailed instructions on how to setup firewall rules or configure applications other than SlamData and Nginx.

3.2 Assumptions

For the example in this quick guide, we'll continue with the following assumptions:

- 192.168.138.220: IP address of MongoDB host, already running on port 27017
- 192.168.138.210: IP address of Backend host
- 192.168.138.200: IP address of Nginx/SlamData UI host
- If network security policy dictates, then network communication between hosts is restricted via ipchains or other firewall mechanism. This allows simpler Backend and Nginx configuration files.
- The reader has at least a basic understanding of JSON formatting and the Linux OS
- Running on Ubuntu 14.04 or similar. If using RedHat, use 'yum' rather than 'apt-get' for software management.

- You have version 2.2.1 or newer of SlamData source code.

If your IP addresses differ, change as appropriate.

3.3 Architecture Overview

As can be noted from the above diagram, the network communications path is straight forward:

1. A request comes into SlamData running under Nginx.
2. Nginx authenticates the user via standard http auth.
3. After appropriate authentication, Nginx allows the user into the SlamData application, which communicates directly to the Backend's web API.
4. The backend runs the appropriate tasks on the data source, MongoDB in this case, and returns the results back upstream, through Nginx to the client.

For increased security, this configuration assumes the reader has setup appropriate network or OS-level restrictions that will deny access to requests other than the system upstream from it. For instance, the MongoDB server should only allow requests to port 27017 from IP 192.168.138.210 (the Backend host). Similarly, the Backend should only allow requests to port 8080 from IP 192.168.138.200 (the SlamData / Nginx host). This setup is not covered in this guide and is left to the reader to implement.

3.4 Setting Up The Backend

The first step is to download the Backend and build the source on the system that will be hosting it.

3.4.1 Download and Build the Backend

```
$ git clone https://github.com/slamdata/quasar
$ cd quasar
$ ./sbt test
$ ./sbt 'project web' oneJar
$ ./sbt 'project core' oneJar
```

These commands will download the latest source code, run the test suite, and build both the Web and Core projects jar files.

3.4.2 Configure

Configure the Backend server to connect to MongoDB. Assuming there is a MongoDB databased called 'testdb' running on 192.168.138.220, your configuration file may be called quasar-config.json and look like this:

```
{
  "mountings": {
    "/local": {
      "mongodb": {
        "connectionUri": "mongodb://192.168.138.220/testdb"
      }
    }
  }
}
```

Restricting Access to SlamData via Nginx Proxy

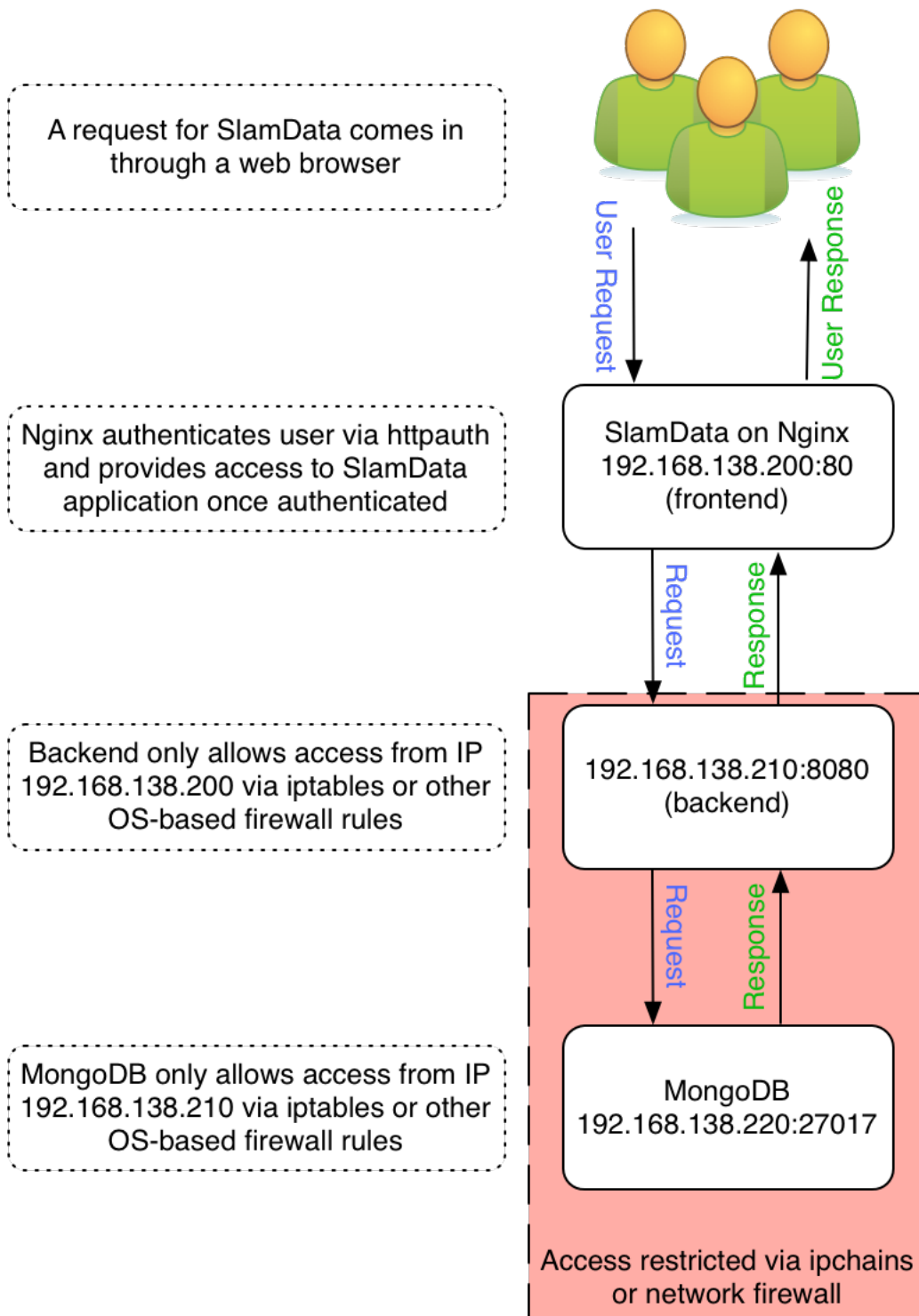


Fig. 3.1: Architecture Overview

```
}  
},  
"server": { "port": 8080 }  
}
```

For further details on the format of the configuration file, please see the [Configuration](#) documentation.

3.4.3 Testing the Backend

Start the Core jar file

From the Backend server, run the core jar file to verify you have connectivity and your mounting is correct:

```
java -jar ~/quasar/core/target/scala-2.11/core_2.11-2.2.1-SNAPSHOT-one-jar.jar ~/quasar-config.json
```

Once launched, a [REPL](#) console will appear representing a virtual file system where each MongoDB database is a directory, and each database directory contains one or more MongoDB collections.

Notice how the the OS-like file system commands and SQL commands are executed directly after the \$ prompt:

```
$ ls  
local@  
$ cd local  
$ ls  
local/  
testdb/  
$ cd testdb  
$ ls  
coll1  
$ select * from coll1;  
Mongo  
db.coll1.find();  
  
Query time: 0.0s  
name      | age   | gender | minor |  
-----|-----|-----|-----|  
Johnny    | 42.0  | male   | false |  
Jenny     | 27.0  | female | false |  
Deb       | 33.0  | female | false |  
Billy     | 15.0  | male   | true  |
```

Start the Web jar file

Once you have verified proper connectivity between the Backend and MongoDB, stop the Core jar file and now start the Web jar file with a slightly different syntax to point to the configuration file:

```
java -jar ~/quasar/web/target/scala-2.11/web_2.11-2.2.1-SNAPSHOT-one-jar.jar -c ~/quasar-config.json
```

Congratulations! You now have two of the three necessary systems up and running for this configuration.

3.4.4 Install Pre-requisites

You should now be working on the system that will be hosting Nginx and SlamData. If you do not already have npm or node installed there, do so first.

On Linux:

```
$ sudo apt-get install npm
$ sudo apt-get install nodejs-legacy
```

On Mac:

```
$ brew install npm
$ brew install nodejs
```

Once npm is installed, utilize it to install [Bower](#), [Gulp](#) and [PureScript](#):

```
$ npm install bower -g
$ npm install gulp -g
$ npm install purescript -g
```

3.4.5 Download and Build SlamData

Now that npm, node, bower, gulp and PureScript are installed, download the SlamData source and compile:

```
$ git clone https://github.com/slamdata/slamdata
$ cd slamdata
$ bower install
$ npm install
$ gulp
```

When gulp completes, you should have a full application under the ‘public’ directory:

```
[/Users/me/slamdata]$ ls public
css          fonts        img          index.html  js          notebook.html
```

It may be safest to copy this directory and place it under a directory with separate permissions for a more secure environment. Whatever directory you use will be considered your web root directory in future steps.

3.5 Installing Nginx

3.5.1 OS X

On OS X systems, consider using [HomeBrew](#) to install Nginx:

```
$ brew install nginx
```

Redhat / CentOS

On RedHat or CentOS systems:

```
$ sudo yum install nginx
```

Ubuntu / Debian

On Ubuntu or Debian systems:

```
$ sudo apt-get install nginx
```

3.5.2 Configuring Nginx

There are two main reasons we'll modify the Nginx configuration file in this guide:

1. To force user authentication, thus restricting access to known individuals
2. To redirect queries to the Backend engine on another host, thus limiting the access path to the Backend API to individuals authenticated with Nginx.

This example will use the 'default' Nginx site configuration. Nginx has many configuration files and is a versatile tool, please contact your Nginx application administrator if you have questions, or visit the Nginx web site.

If Nginx is already running, stop it:

```
sudo service nginx stop
```

3.5.3 Setting up Authentication

To allow http authentication, we'll need to create a file which stores the names and passwords of allowed individuals. To do this, use the apache2-utils package which provides those tools:

```
sudo apt-get install apache2-utils
```

Now create the htpasswd file we'll use to store the encrypted data:

```
sudo htpasswd -c /etc/nginx/.htpasswd exampleuser
```

Replace 'exampleuser' with a real username. You'll then be prompted for a password and verification password.

This creates the file /etc/nginx/.htpasswd which we will reference in the Nginx configuration file below.

3.5.4 Nginx Configuration File

Assuming the Nginx site configuration file is located at /etc/nginx/sites-available/default, replace the contents with the following code, making adjustments where necessary of course:

```
server {
    listen 80 default_server;
    listen [::]:80 default_server ipv6only=on;

    server_name your_server_name;

    client_max_body_size 1024M;

    location / {
        auth_basic "Restricted";
        auth_basic_user_file /etc/nginx/.htpasswd;
        root /slamdata/public;
        try_files $uri $uri/ @backend;
    }
}
```

```
location @backend {
    proxy_set_header X-Forwarder-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_pass http://192.168.138.210:8080;
}
```

Pay close attention to the ‘root’ directive above. Replace the value with the web root directory referenced earlier. The configuration above ensures the following important actions:

1. Nginx runs on port 80
2. Enables http authentication (lines beginning with ‘auth_basic’)
3. Nginx acts as a reverse proxy to the Backend service. This can either be on a remote host (as indicated above) or your the same host as Nginx.

Save the configuration and start or restart Nginx:

```
sudo service nginx restart
```

If your network firewall is setup properly, the Backend should be shielded from all requests except those coming directly from the Nginx host. Additionally all requests coming from the Nginx host should only originate from authenticated users via http authentication. You may test the SlamData / Nginx HTTP authorization by going to the URL:

`http://192.168.138.200`

You should be immediately prompted for a username and password. Upon successful authorization you should see the SlamData UI:

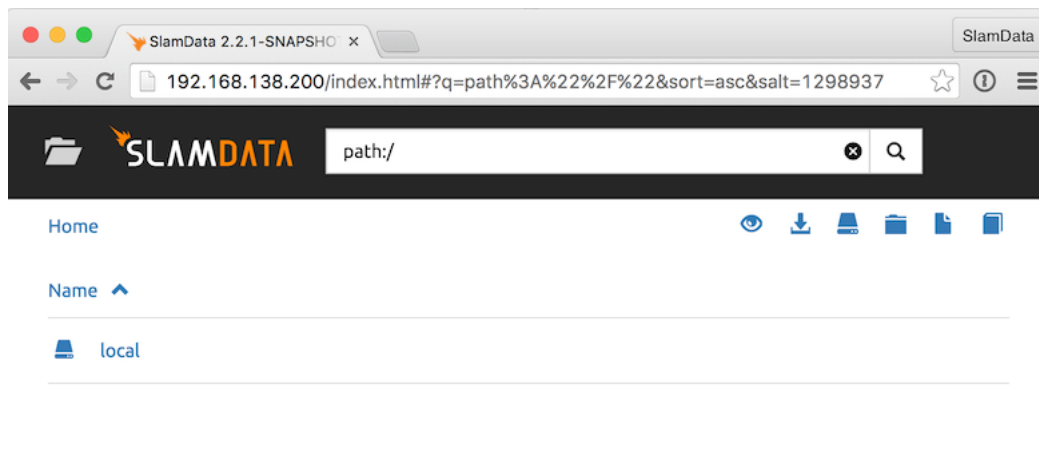


Fig. 3.2: After Login

Note that you are sending requests to Nginx (IP .200), which authenticates a username and then sends the request to the Backend (IP .210), then returns the results directly to the browser. The browser itself is not redirected as that would defeat the purpose of securing with Nginx.



Troubleshooting FAQ

4.1 Configuration File Locations

Upon initial launch, SlamData will not have a configuration file. Once a valid database mount has been configured, a file will be created and used to store those mount points. See below for the location of that configuration file. Unless specified on the command line, SlamData will look for its configuration in the following locations by default:

Windows:

```
%HOMEDIR%\AppData\Local\quasar\quasar-config.json
```

Mac OS X:

```
$HOME/Library/Application Support/quasar/quasar-config.json
```

Linux:

```
$HOME/.config/quasar/quasar-config.json
```

Note: Do not modify this file by hand unless you have first made a backup. If you modify this file while SlamData is running, your changes may be overwritten.

4.2 Log File Locations

SlamData has a single log file whose location depends upon the OS. Replace `version` below with the actual version number that is running.

Windows:

```
C:\Program Files (x86)\slamdata <version>\slamdata-<version>.log
```

Linux:

```
$HOME/slamdata<version>/slamdata-<version>.log
```

Mac OS X:

```
/Applications/SlamData <version>.app/Contents/java/app/slamdata-<version>.log
```

4.3 SlamData Won't Start

This is typically caused by an invalid database mount. Either the database is not currently available, or a previously configured mount is no longer valid. Locate the *configuration file* and if necessary modify the file by hand, or if you wish to erase all previously configured mounts, you may simply rename or delete the file and restart SlamData.

4.4 SlamData URL

When SlamData starts on Windows and Mac OS X, a browser window (or new tab) should open with the appropriate URL based on the port defined in the configuration file. Unless the port has been modified, the URL will likely be as follows, replacing servername with the actual host name:

```
http://servername:20223/slamdata/index.html
```

4.5 How do I see which version I'm running?

You can simply look at your title bar in your browser window. The title of the web page should include it.

4.6 SlamData on Amazon EC2 / Microsoft Azure, etc.

When running SlamData with a hosting provider, including Amazon EC2, the most common error encountered is a security policy misconfiguration. SlamData will need to connect to your database over the same port that a standard database client does.

Keep in mind that the database server and the SlamData server do not need to run on the same system, though there is no problem with that either. Follow this simple checklist to ensure network problems are minimized:

Verify the security policy for the database server is:

- accepting incoming connections from the SlamData server IP address
- accepting incoming connections on the correct port

If you are still unable to connect to your hosted database:

- Verify you can connect with a standard database client from any system.
 - Next connect with a standard database client from the same system SlamData is running on.
-

4.7 SlamData Notebook Won't Delete

When Notebooks in version 2.3.3 or earlier were renamed or deleted, sometimes the view entries associated were not updated in the configuration file.

In version 2.4 and newer, this is no longer an issue. It is a good idea to review your configuration file, removing any now-defunct entries relating to views associated with renamed or deleted notebooks.



Fig. 4.1: SlamData Logo

Reference - SQL²

5.1 Introduction

SQL2 is a subset of ANSI SQL, designed for queries into NoSQL databases.

SQL2 has support for every major SQL SELECT clause, such as AS, WHERE, JOIN, GROUP BY, HAVING, LIMIT, OFFSET, CROSS, etc. It also contains many standard SQL functions and operators. It follows PostgreSQL where SQL dialects diverge.

5.2 Data Types

The following data types are used by SQL2.

Note: > Some data types are not natively supported by all databases. Instead, they are emulated by SlamData, meaning that you can use them as if they were supported by the database.

MDB = Native MongoDB Support

XYZ = Native XYZ DB Support (example for future databases)

Type	Description	Examples	MDB	XYZ
Null	Indicates missing information.	null	Yes	???
Boolean	true or false	true, false	Yes	???
Integer	Whole numbers (no fractional component)	1, -2	Yes	???
Decimal	Decimal numbers (optional fractional components)	1.0, -2.19743	Yes	???
String	Text	"221B Baker Street"	Yes	???
DateTime	Date and time, in ISO8601 format	TIMESTAMP ("2004-10-19T10:23:54")	Yes	???
Time	Time in the format HH:MM:SS.	TIME ("10:23:54")	No	???
Date	Date in the format YYYY-MM-DD	DATE ("2004-10-19")	No	???
Interval	Time interval, in ISO8601 format	INTERVAL ("P3DT4H5M6S")	No	???
Object ID	Unique object identifier.	OID ("507f1f77bcf86cd799439011")	Yes	???
Ordered Set	Ordered list with no duplicates allowed	(1, 2, 3)	No	???
Array	Ordered list with duplicates allowed	[1, 2, 2]	Yes	???

5.3 Clauses, Operators, and Functions

The following clauses are supported:

Type	Clauses
Basic	SELECT, AS, FROM
Joins	LEFT OUTER JOIN, RIGHT OUTER JOIN, INNER JOIN, FULL JOIN, CROSS
Filtering	WHERE
Grouping	GROUP BY, HAVING
Conditional	CASE , WHEN, DEFAULT
Paging	LIMIT, OFFSET
Sorting	ORDER BY , DESC, ASC

The following operators are supported:

Type	Operators
Numeric	+, -, *, /, %
String	~, ~*, !~, !~*, LIKE,
Array	, [...]
Relational	=, >=, <=, <>, BETWEEN, IN, NOT IN
Boolean	AND, OR, NOT
Projection	foo.bar, foo[2], foo{*}, foo[*]
Date/Time	TIMESTAMP, DATE, INTERVAL, TIME
Identity	OID

Note: ~, ~*, !~, and !~* are regular expression operators. ~*, !~, and !~* are preliminary and may not work in the current release.

Also Note: The || operator for strings will concatenate two strings; for example, you can create a full name from a first and last name property: `c.firstName || ' ' || c.lastName`. The || operator for arrays will concatenate two arrays; for example, if `xy` is an array with two values, then `c.xy || [0]` will create an array with three values, where the third value is zero.

The following functions are supported:

Type	Functions
String	CONCAT, LOWER, UPPER, SUBSTRING, LENGTH, SEARCH
Date/Time	DATE_PART, TO_TIMESTAMP
Nulls	COALESCE
Arrays	ARRAY_LENGTH, FLATTEN_ARRAY
Objects	FLATTEN_OBJECT
Set-Level	DISTINCT, DISTINCT_BY
Aggregation	COUNT, SUM, MIN, MAX, AVG
Identity	SQUASH

5.4 Syntax Changes in Version 2.5

When SlamData 2.5 was released in early 2016 the SQL2 syntax changed slightly. Please make note of the following syntax to avoid problems in queries:

Query paths must be surrounded by backward-ticks ' on both ends of the path.

Example:

```
SELECT * FROM `/users`
```

When specifying a string in any context surround it by quotes " on both ends.

Example:

```
SELECT * FROM `/users` WHERE first_name = "Joe"
```

When specifying a single character, and not a full string, in any context, surround it by single ticks ' on both ends.

Example:

```
SELECT * FROM `/users` WHERE middle_initial = 'A'
```

When using functions such as DATE, DATETIME, TIMESTAMP, etc that require a value it must be surrounded by (and). Single character ' and string " quotes still apply.

Example:

```
SELECT * FROM `/users`  
WHERE  
    last_login > DATE("2016-01-01") AND  
    subscribed = 'Y'
```

5.5 Basic Selection

The SELECT statement returns a result set of records from one or more tables.

5.5.1 Select all values from a path

To select all values from a path, use the asterisk (*).

Example:

```
SELECT * FROM `/users`
```

5.5.2 Select specific fields from a path

To select specific fields from a path, use the field names, separated by commas.

Example:

```
SELECT name, age FROM `/users`
```

5.5.3 Give a path an alias to refer to in the query

Follow the path name with an AS and an alias name, and then you can use the alias name when specifying the fields. This is especially useful when you have data from more than one source.

Example:

```
SELECT c.name, c.age FROM `/users` AS c
```

5.6 Filtering a Result Set

You can filter a result set using the WHERE clause. The following operators are supported:

- Relational: `-`, `=`, `>=`, `<=`, `<>`, `BETWEEN`, `IN`, `NOT IN`
- Boolean: `AND`, `OR`, `NOT`

5.6.1 Filtering using a numeric value:

Example:

```
SELECT c.name FROM `/users` AS c WHERE c.age > 40
```

5.6.2 Filtering using a string value:

Example:

```
SELECT c.name FROM `/users` AS c WHERE c.name = "Sherlock Holmes"
```

5.6.3 Filtering using multiple Boolean predicates:

Example:

```
SELECT
  c.name FROM `/users` AS c
WHERE
  c.name = "Sherlock Holmes" AND
  c.street = "Baker Street"
```

5.7 Numeric and String Operations

You can use any of the operators or functions listed in the *Clauses, Operators, and Functions* section on numbers and strings. Some common string operators and functions include:

Operator or Function	Description
<code> </code>	Concatenates
<code>LOWER</code>	Converts to lowercase
<code>UPPER</code>	Converts to uppercase
<code>SUBSTRING</code>	Returns a substring
<code>LENGTH</code>	Returns length of string

Examples:

Using mathematical operations:

```
SELECT c.age + 2 * 1 / 4 % 2 FROM `/users` AS c
```

Concatenating strings:

```
SELECT c.firstName || ' ' || c.lastName AS name FROM `/users` AS c
```

Filtering by fuzzy string comparison using the `LIKE` operator:

```
SELECT * FROM `/users` AS c WHERE c.firstName LIKE "%Joan%"
```

Filtering by regular expression:

```
SELECT * FROM `/users` AS c WHERE c.firstName ~ "[sS]h+"
```

5.8 Dates and Times

Filter by dates and times using the `TIMESTAMP`, `TIME`, and `DATE` operators. The `DATEPART` operator can also be used to select part of a date, such as the day.

Note: Some databases will automatically convert strings into dates or date/times. SlamData does not perform this conversion, since the underlying database has no schema and no fixed type for any field. As a result, an expression like `WHERE ts > "2015-02-10"` compares string-valued `ts` fields with the string `"2015-02-10"` instead of a date comparison.

If you want to embed literal dates, timestamps, etc. into your SQL queries, you should use the time conversion operators, which accept a string and return value of the appropriate type. For example, the above snippet could be converted to `WHERE ts > DATE("2015-02-10")`, which looks for date-valued `ts` fields and compares them with the date `2015-02-10`.

NOTE for MongoDB Users:

If your MongoDB data does not use MongoDB's native date/time type, and instead, you store your timestamps as epoch milliseconds in a numeric value, then you should either compare numbers or use the `TO_TIMESTAMP` function.

5.8.1 Filter based on a timestamp (date and time)

Use the `TIMESTAMP` operator to convert a string into a date and time. The string should have the format `YYYY-MM-DDTHH:MM:SSZ`.

Example:

```
SELECT * FROM `/log/events` AS c WHERE c.ts > TIMESTAMP("2015-04-29T15:16:55Z")
```

5.8.2 Filter based on a time

Use the `TIME` operator to convert a string into a time. The string should have the format `HH:MM:SS`.

Example:

```
SELECT * FROM `/log/events` AS c WHERE c.ts > TIME("15:16:55")
```

5.8.3 Filter based on a date

Use the `DATE` operator to convert a string into a date. The string should have the format `YYYY-MM-DD`.

Example:

```
SELECT * FROM `/log/events` AS c WHERE c.ts > DATE("2015-04-29")
```

5.8.4 Filter based on part of a date

Use the `DATE_PART` function to select part of a date. `DATE_PART` has two arguments: a string that indicates what part of the date or time that you want and a timestamp field. Valid values for the first argument are century, day, decade, dow (day of week), doy (day of year), hour, isodoy, microseconds, millenium, milliseconds, minute, month, quarter, second, and year.

Example:

```
SELECT DATE_PART("day", c.ts) FROM `/log/events` AS c
```

5.8.5 Filter based on a Unix epoch

Use the `TO_TIMESTAMP` function to convert Unix epoch (milliseconds) to a timestamp.

Example:

```
SELECT * FROM `/log/events` AS c WHERE c.ts > TO_TIMESTAMP(1446335999)
```

5.9 Grouping

SQL2 allows you to group data by fields and by date parts.

5.9.1 Group based on a single field

Use `GROUP BY` to group results by a field.

Example:

```
SELECT
    c.age,
    COUNT(*) AS cnt
FROM `/users` AS c
GROUP BY c.age
```

5.9.2 Group based on multiple fields

You can group by multiple fields with a comma-separated list of fields after `GROUP BY`.

Example:

```
SELECT
    c.age,
    c.gender,
    COUNT(*) AS cnt
FROM `/users` AS c
GROUP BY c.age, c.gender
```

5.9.3 Group based on date part

Use the `DATE_PART` function to group by a part of a date, such as the month.

Example:

```
SELECT
    DATE_PART("day", c.ts) AS day,
    COUNT(*) AS cnt
FROM `/log/events` AS c
GROUP BY DATE_PART("day", c.ts)
```

5.9.4 Filter within a group

Filter results within a group by adding a `HAVING` clause followed by a Boolean predicate.

Example:

```
SELECT
    DATE_PART("day", c.ts) AS day,
    COUNT(*) AS cnt
FROM `/prod/purger/events` AS c
GROUP BY DATE_PART("day", c.ts)
HAVING c.gender = "female"
```

5.9.5 Double grouping

Perform double-grouping operations by putting operators inside other operators. The inside operator will be performed on each group created by the `GROUP BY` clause, and the outside operator will be performed on the results of the inside operator.

Example:

This query returns the average population of states. The outer aggregation function (`AVG`) operates on the results of the inner aggregation (`SUM`) and `GROUP BY` clause.

```
SELECT AVG(SUM(pop)) FROM `/population` GROUP BY state
```

5.10 Nested Data and Arrays

Unlike a relational database many NoSQL databases allow data to be nested (that is, data can be objects) and to contain arrays.

5.10.1 Nesting

Nesting is represented by levels separated by a period (.).

Example:

```
SELECT c.profile.address.street.number FROM `/users` AS c
```

5.10.2 Arrays

Array elements are represented by the array index in square brackets ([n]).

Example:

```
SELECT c.profile.allAddress[0].street.number FROM `/users` AS c
```

5.10.3 Flattening

You can extract all elements of an array or all field values simultaneously, essentially removing levels and flattening the data. Use the asterisk in square brackets ([*]) to extract all array elements.

Example:

```
SELECT c.profile.allAddresses[*] FROM `/users` AS c
```

Use the asterisk in curly brackets ({*}) to extract all field values.

Example:

```
SELECT c.profile.{*} FROM `/users` AS c
```

5.10.4 Filtering using arrays

You can filter using data in all array elements by using the asterisk in square brackets ([*]) in a WHERE clause.

Example:

```
SELECT DISTINCT * FROM `/users` AS c WHERE c.profile.allAddresses[*].street.number = "221B"
```

5.11 Pagination and Sorting

5.11.1 Pagination

Pagination is used to break large return results into smaller chunks. Use the LIMIT operator to set the number of results to be returned and the OFFSET operator to set the index at which the results should start.

Example (Limit results to 20 entries):

```
SELECT * FROM `/users` LIMIT 20
```

Example (Return the 100th to 119th entry):


```
SELECT * FROM `/users` OFFSET 100 LIMIT 20
```

5.11.2 Sorting

Use the `ORDER BY` clause to sort the results. You can specify one or more fields for sorting, and you can use operators in the `ORDER BY` arguments. Use `ASC` for ascending sorting and `DESC` for descending sorting.

Example (Sort users by ascending age):

```
SELECT * FROM `/users` ORDER BY age ASC
```

Example (Sort users by last digit in age, descending, and full name, ascending):

```
SELECT * FROM `/users`  
ORDER BY age % 10 DESC, firstName + lastName ASC
```

5.12 Joining Collections

Use the `JOIN` operator to join two or more different collections.

The `JOIN` operator is a powerful way to implement joins in non-relational databases such as MongoDB. There is no enforced limit to how many collections or tables can be joined in a query but common sense should prevail based on the size of collections.

Examples:

This example returns the names of employees and the names of the departments they belong to by matching up the employee department ID with the department's ID, where both IDs are ObjectID types.

```
SELECT  
    emp.name,  
    dept.name  
FROM `/employees` AS emp  
JOIN `/departments` AS dept ON dept._id = emp.departmentId
```

If one of the IDs is a string, then use the `OID` operator to convert it to an ID.

```
SELECT  
    emp.name,  
    dept.name  
FROM `/employees` AS emp  
JOIN `/departments` AS dept ON dept._id = OID(emp.departmentId)
```

5.12.1 Join Considerations

On `JOINS` with more than two collections or tables, the standard rule of thumb is to place the tables in order from smallest to largest. If the collections `a`, `b`, and `c` have 4, 8, and 16 documents respectively, then ordering `FROM `/a`, `/b`, `/c`` is most efficient with `WHERE a._id = b._id`.

If, however, the filter condition is `WHERE b._id = c._id` then the appropriate ordering would be `FROM `/b`, `/c`, `/a` WHERE b._id = c._id`. This is because without the filter `la bl = 32` which is less than `lb cl = 128`, but with the filter, `lb cl` is limited to the number of documents in `b`, which is 8 (and which is lower than the unconstrained `la bl`).

5.13 Conditionals and Nulls

5.13.1 Conditionals

Use the CASE expression to provide if-then-else logic to SQL2. The CASE syntax is:

```
SELECT (CASE <field>
  WHEN <value1> THEN <result1>
  WHEN <value2> THEN <result2>
  ...
  ELSE <elseResult>
END)
FROM `<path>`
```

Example:

The following example generates a code based on gender string values.

```
SELECT (CASE c.gender
  WHEN "male" THEN 1
  WHEN "female" THEN 2
  ELSE 3
END) AS genderCode
FROM `/users` AS c
```

5.13.2 Nulls

Use the COALESCE function to evaluate the arguments in order and return the current value of the first expression that initially does not evaluate to NULL.

Example:

This example returns a full name, if not null, but returns the first name if the full name is null.

```
SELECT COALESCE(c.fullName, c.firstName) AS name FROM `/users` AS c
```

5.14 Database Specific Notes

5.14.1 MongoDB

The `_id` Field

By default, the `_id` field will not appear in a result set. However, you can specify it by selecting the `_id` field. For example:

```
SELECT _id AS cust_id FROM `/users`
```

MongoDB has special rules about fields called `_id`. For example, they must remain unique, which means that some queries (such as `SELECT myarray[*] FROM foo`) will introduce duplicates that MongoDB won't allow. In addition, other queries change the value of `_id` (such as grouping). So SlamData manages `_id` and treats it as a special field.

Note: To filter on `_id`, you must first convert a string to an object ID, by using the `OID` function. For example:

```
SELECT * FROM `/foo` WHERE _id = OID("abc123")
```



Reference - SlamDown

This SlamDown Reference can assist with the proper formatting of SlamDown code to produce static and interactive forms within SlamData.

6.1 Introduction

SlamData contains its own markup language called SlamDown, which is useful for creating reports and forms. SlamDown is a subset of [CommonMark](#), a specification for a highly compatible implementation of [Markdown](#).

In addition, SlamDown also includes two extensions to CommonMark: *form fields* and *evaluated SQL2 queries*.

This reference contains the following sections:

6.2 Block Elements

The following SlamDown elements create blocks of content.

6.2.1 Horizontal Rules

Three dashes or more create a horizontal line. Put a blank line above and below the dashes.

```
Text here

---

More text here
```

results in:

Text here

More text here

6.2.2 Headers

Use hash marks (#) for *ATX headers*, with one hashmark for each level.

```
# Top level
## Second level
### Third level
```

results in a first, second, and third level heading:

Top Level
Second Level
Third Level

Fig. 6.1: Headers

6.2.3 Code Blocks

You can create blocks of code (that is, literal content in monospace font) in two ways:

Indented code blocks

Indent by four spaces.

Fenced code blocks

Start and end with three or more backtick (‘) characters.

```
```\nfor (int i =0; i < 10; i++)\n    sum += myArray[i];\n\\``
```

Both Indented Code Blocks and Fenced Code Blocks result in:

```
for (int i =0; i < 10; i++)\n sum += myArray[i];
```

## 6.2.4 Paragraphs

Paragraphs are separated by a blank line.

```
This is paragraph 1.\n\nThis is paragraph 2.
```

results in:

This is paragraph 1.

This is paragraph 2.

## 6.2.5 Block quotes

Start with a greater than sign (>) to create a block quote.

```
> This is a block quote.
```

results in:

This is a block quote.

## 6.2.6 Lists

Ordered lists start with numbers followed by periods. The actual numbers in the SlamDown do not matter. In the end, they will be displayed with ascending indices.

```
1. First item
2. Second item
3. Third item
```

results in:

1. First item
2. Second item
3. Third item

Unordered lists start with either asterisks (\*), dashes (-), or pluses (+). They are interchangeable.

```
* First item
* Second item
* Third item
```

results in:

- First item
- Second item
- Third item

## 6.3 Inline Elements

The following inline elements are supported in SlamDown. In addition to standard Markdown elements, there is also the ability to *evaluate a SQL query* and put the result into the content.

### 6.3.1 Emphasis and Strong Emphasis

Surround content with asterisks (\*) for emphasis and surround it with double asterisks (\*\*) for strong emphasis.

```
This is *important*. This is **more important**.
```

results in:

This is *important*. This is **more important**.

### 6.3.2 Links

Links contain the link title in square brackets ( [ ] ) and the link destination in parentheses ( ( ) ).

```
[SlamData] (http://slamdata.com)
```

results in:

[SlamData](http://slamdata.com)

If the link title and destination are the same, you can use an autolink, where the URI is contained in angle brackets (<>).

```
<http://slamdata.com>
```

results in:

<http://slamdata.com>

### 6.3.3 Images

Images start with an explanation mark ( ! ), followed by the image description in square brackets ( [ ] ) and the image URI in parentheses ( ( ) ).

```
![SlamData Logo] (https://media.licdn.com/media/p/6/005/088/002/039b9f8.png)
```

results in:

### 6.3.4 Inline code formatting

To add code formatting (literal content with monospace font) inline, put the content between backtick ( ` ) characters.

```
Start SQL statements with `SELECT * FROM`
```

results in:

Start SQL statements with `SELECT * FROM`

### 6.3.5 Evaluated SQL Query

SlamDown extends CommonMark by allowing you to evaluate a SQL query and insert the results into the rendered content. Start the query with an exclamation point and then contain the SQL query between double backtick ( ` ) characters.

Note: > Notice how the path to the query below has a space between the backtick that ends the path and the double backticks that end the query. This is a necessary space because three backticks in a row start a Fenced Code Block as stated above.

In the example below, if there are 20 documents in the `/col` file, then

```
There are !``SELECT COUNT(*) FROM `/col` `` documents inside the collection.
```

results in:

There are 20 documents inside the collection.



## 6.4 Form Elements

SlamDown contains a significant addition to CommonMark, which are form elements. This allows for the creation of interactive reports.

### 6.4.1 Text Field

Use one or more underscores (`_`) to create a text input field where a user can add text.

For example, this line creates an input field for a name. You can then refer to the user value with the string variable `name`.

```
name = _____
```

Optionally, you can pre-fill the input field with a default value by having it after the underscores in parentheses. This line creates an input field for spouse name with a default value of “none”. You can then refer to the user value with the string variable `spouse`.

```
spouse = _____ (none)
```

By default input fields are evaluated as String types. To enforce a numeric type prefix the underscores with the (`#`) symbol. For example:

```
year = #_____
```

### 6.4.2 Radio Buttons

Use parentheses followed by text to indicate radio buttons. A set of radio buttons has only one button selected at a time. Indicate which button is selected by putting an `x` in the parentheses.

For example, this line creates a set of radio buttons with the values “car”, “bus”, and “bike”, where “car” is marked as the default. The result is stored in the string variable named `commute` for later use.

```
commute = (x) car () bus () bike
```

This results in:

`commute`    ☒ `car`    ☐ `bus`    ☐ `bike`

Fig. 6.2: Radio Buttons

**Note:** Currently, the default value must be the first value.

### 6.4.3 Dynamic Radio Buttons

The results of a SQL<sup>2</sup> query can be used to dynamically populate a set of Radio Buttons. For example, this Markdown code creates a variable called `graphNode` that displays all of the sensors in a database, allowing a user to select which specific node to visualize in a chart:

```
graphNode =
(!`SELECT DISTINCT sensor FROM `/macbook/tsdemo/timeseries` ORDER BY sensor ASC LIMIT 1`)
(!`SELECT DISTINCT sensor FROM `/macbook/tsdemo/timeseries` ORDER BY sensor ASC`)
```

# Select which sensor to report on

graphNode ☒ S0 ☐ S1 ☐ S2 ☐ S3 ☐ S4

This results in:

**Note:** Currently, the default value must be the first value.

## 6.4.4 Checkboxes

Use square brackets followed by text to indicate checkboxes. In a set of checkboxes, each checkbox operates independently. Use an x in the square brackets to indicate that the checkbox should be checked by default. The string value returned will be an array of strings in square brackets.

For example, this line creates a set of checkboxes with the values “Android”, “iPhone”, and “Blackberry”. The result is stored in the string variable named `phones` for later use.

```
phones = [] Android [x] iPhone [x] Blackberry
```

If left as the default, the `phones` variable will have the value: `['iPhone', 'Blackberry']`.

This results in:

phones ☐ Android ☒ iPhone ☒ Blackberry

Fig. 6.3: Check Boxes

## 6.4.5 Dynamic Check Boxes

The results of a SQL<sup>2</sup> query can be used to populate Check Boxes. For example, this Markdown code creates a variable called `graphNodes` that displays all of the sensors in a database, allowing a user to select which nodes to visualize in a chart:

```
graphNodes =
[!``SELECT DISTINCT sensor FROM `/macbook/tsdemo/timeseries` ORDER BY sensor ASC LIMIT 1``]
!``SELECT DISTINCT sensor FROM `/macbook/tsdemo/timeseries` ORDER BY sensor ASC``
```

The first query tells SlamDown which entry is selected by default in the UI. The second query populates the remaining check boxes.

This results in:

## 6.4.6 Dropdown

Use a comma-separated list in curly brackets to indicate a dropdown element.

For example, this line creates a dropdown element with BOS, SFO, and NYC entries. The result is stored in the string variable named `city` for later use.

# Select which sensors to report on

**graphNodes** ☒ **S0** ☐ **S1** ☐ **S2** ☐ **S3** ☐ **S4**

```
city = {BOS, SFO, NYC}
```

Optionally, include a default value by listing it in parentheses at the end. In this line, NYC is set as the default.

```
city = {BOS, SFO, NYC} (NYC)
```

This results in:

city NYC ↕

## 6.4.7 Dynamic Dropdown

The results of a SQL<sup>2</sup> query can be used to populate a Drop Down element. For example, this Markdown code creates a variable called *industry* that displays all of the industry types listed in a stock market database, allowing a user to select which industry to visualize in a chart:

```
industry = {!``SELECT DISTINCT (Industry) FROM `/macbook/demo/stocks` ORDER BY Industry``}
```

This results in:

## 6.4.8 Dates and Times

Provide a date, time or both date & time selector for users by implementing the following syntax.

### Date

For example the following line creates a date selector element and stores the value in a variable called *start*:

```
start = __ - __ - ____ (04-19-2016)
```

### Time

The following lines creates a time selector element:

```
start = __ : __ (12:30 PM)
```

## Select Industry Below

industry

- ✓ Accident & Health Insurance
- Advertising Agencies
- Aerospace/Defense - Major Diversified
- Aerospace/Defense Products & Services
- Agricultural Chemicals
- Air Delivery & Freight Services
- Air Services, Other
- Aluminum
- Apparel Stores
- Appliances
- Application Software
- Asset Management
- Auto Dealerships
- Auto Manufacturers - Major
- Auto Parts

## Date & Time

The following line creates both a date and time selector element:

```
start = __ - __ - ____ __ : __ (06-06-2015 12:00 PM)
```

This results in:

**start**

04/19/2016, 08:30 PM

April 2016

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Fig. 6.4: Date and Time Selector





---

## Reference - SlamData API

---

### 7.1 Introduction

This API Reference provides detailed information for developers to use when interacting with SlamData via the API.

**Note:** Some features are only available in SlamData Advanced and are noted as such.

---

### 7.2 Assumptions

Throughout this document examples of URLs might be given. These examples assume that SlamData is running locally on IP 127.0.0.1 with hostname `localhost` and running on the default port 20223

---

### 7.3 Reference

#### 7.3.1 Executing a Small Query

Executes a SQL2 query where the results and computation are expected to be relatively small. Pagination and some filtering are supported.

##### URL

`/query/fs/{path}`

where `{path}` is an optional path to the data. If included, then all paths used in the query are relative to the `{path}` parameter, unless they begin with a `/`. A complete URL would appear as follows:

`http://127.0.0.1:20223/query/fs/{path}`

##### Method

GET

---

## Query parameters

Parameter	Description	Required	Default
q	The SQL query	Required	
offset	The starting index of the results to return	Optional	0
limit	The number of results to return   Optional	All results	
var	Specifies variables in the query. See Note below.	Optional	None

**Note:** The *var* parameter takes the format:

```
var.{name}={value}
```

where {name} is the name of the variable and {value} is the value of the variable. For example, the query might contain the variable *cutoff*:

```
SELECT * WHERE pop > :cutoff
```

Then the *cutoff* variable is assigned a value in the parameter *var.cutoff=1000*.

Failure to specify valid values for all variables used inside a query will result in an error. These values use the same syntax as the query itself; notably, strings should be surrounded by single quotes. Some acceptable values are 123, "CO", and DATE ("2015-07-06").

## Headers

Header	Description	Re-quired	Default
Accept	Specifies the format of the response body.	Optional	<i>application/json;mode=precise</i>
Accept-Encoding	Use the value <i>gzip</i> to compress the output.	Optional	No compression

The following values are supported for the *Accept* header:

Value	Description
None	“Human-Readable” results, one result per line. Note: not parseable as a single JSON object.
application/json	Nicely formatted JSON array
application/json;mode=precise	One result per line
text/csv	Comma-separated results. See Note below.

**Note:** The formatting of CSV output can be controlled with an extended media type with parameters for *columnDelimiter*, *quoteChar* and *escapeChar*. For example:

```
Accept: text/csv;columnDelimiter="\"&rowDelimiter=";""eChar="'"&escapeChar="\".
```

## Example

The following example returns data for the query:

```
SELECT * from `/data/SampleJSON` WHERE state="WA"
```



## Request

```
GET http://localhost:20223/query/fs?q=SELECT * from `/data/SampleJSON` WHERE state="WA"
```

**NOTE:** The query shown in the above request must be HTML encoded.

```
Headers: Accept: application/json
```

## Response

```
[
 {
 "city": "ALGONA",
 "state": "WA",
 "pop": 22846,
 "loc": [-122.270057, 47.316339]
 },
 {
 "city": "AUBURN",
 "state": "WA",
 "pop": 22846,
 "loc": [-122.206741, 47.30503]
 },
 ...]
```

---

## 7.3.2 Executing a Large Query

Executes a SQL<sup>2</sup> query where the results or computation are expected to be relatively large. The results are stored in an output path that is specified in the *Destination* header. Pagination and some filtering are supported.

### URL

```
/query/fs/{path}
```

where {path} is an optional path to the data. If included, then all paths used in the query and the output path are relative to the {path} parameter, unless they begin with a /.

### Method

POST

## Query parameters

Parameter	Description	Required	Default	Method
offset	The starting index of the results to return	0	Optional	GET
limit	The number of results to return	All results	GET	
var	Specifies variables in the query. See Note below.	Optional	None	GET and POST

**Note:** The *var* parameter takes the format:

```
var.{name}={value}
```

where {name} is the name of the variable and {value} is the value of the variable. For example, the query might contain the variable *cutoff*:

```
SELECT * WHERE pop > :cutoff
```

Then the *cutoff* variable is assigned a value in the parameter *var.cutoff=1000*.

Failure to specify valid values for all variables used inside a query will result in an error. These values use the same syntax as the query itself; notably, strings should be surrounded by single quotes. Some acceptable values are 123, "CO", and DATE ("2015-07-06").

## POST body

The POST body contains the query.

## Headers

Header	Description	Required
Destination	The URL of the output path, where the results of the query will become available if this API successfully completes.	Required

## Response

The following response elements are returned in JSON format:

Element	Description	Notes
out	Path to the query results	Element returned if request is successful
error	Error text	Element is returned if request is not successful
phases	An array containing a sequence of objects containing the result from each phase of the query compilation process	See note below

**Note:** Phase objects have three possible forms. All phase objects have a “name” element with the phase name.

A phase may contain a *tree* element with *type*, *label* and optional *children* elements, such as:

```
{..., "phases": [..., {
 "name": "Logical Plan",
 "tree": {
 "type": "LogicalPlan/Let",
 "label": "'tmp0",
 "children": [{
 "type": "LogicalPlan/Read",
 "label": "./zips"
 }, ...]
 }
}, ...]
}
```

Or a phase may contain a detail element with some kind of text, such as:

```
{...
 "phases": [..., {
 "name": "Mongo",
 "detail": "db.zips.aggregate([{ "
 $sort " : { "
 pop " : 1}]])"
 }
}]
}
```

If an error occurs, then the phase contains an `error` element with the error message. Typically the error phase is the last phase in the array. For example:

```
{...
 "phases": [..., {
 "name": "Physical Plan",
 "error": "Cannot compile..."
 }
}]
}
```

**Note:** The error is also included at the top level of the response, as described in the elements table.

## Example

The following example returns data for the query:

```
SELECT * from ``/data/SampleJSON`` WHERE state="WA"
```

It puts the result in a new file called `SampleResult` in the path `/data`.

## Request

```
POST http://localhost:20223/query/fs
```

```
POST body: SELECT * FROM ``/data/SampleJSON`` WHERE state="WA"
```

```
Headers: Destination: /data/sampleResults Accept: application/json
```

## Response

```
{
 "out": "/data/sampleResults",
 "phases": [{
 "name": "SQL AST",
 "tree": {
 "type": "AST/Select",
 "label": null,
 "children": [{
 "type": "AST/Proj",
 "label": null,
 "children": [{
 "type": "AST/Splice",
 "label": null
 }]
 }]
 }, ...]
 }, ...]
}
```

---

### 7.3.3 Compiling a Query

Compiles, but does not execute a SQL<sup>2</sup> query.

## URL

/compile/fs/{path}

where {path} is an optional path to the data. If included, then all paths used in the query are relative to the {path} parameter, unless they begin with a /.

The GET method has the SQL<sup>2</sup> query as a query parameter and the POST method has the SQL<sup>2</sup> query in the POST body.

## Method

GET or POST

## Query parameters

Parameter	Description	Required	Default	Method
q	The SQL query	Required		GET only
var	Specifies variables in the query. See Note below.	Optional	None	GET and POST

**Note:** The *var* parameter takes the format:

```
var.{name}={value}
```

where {name} is the name of the variable and {value} is the value of the variable. For example, the query might contain the variable `cutoff`:

```
SELECT * WHERE pop < :cutoff
```

Then the `cutoff` variable is assigned a value in the parameter `var.cutoff=1000`.

Failure to specify valid values for all variables used inside a query will result in an error. These values use the same syntax as the query itself; notably, strings should be surrounded by single quotes. Some acceptable values are 123, "CO", and DATE ("2015-07-06").

## POST body

For the POST request, use the SQL query as the POST body.

## Response

If the query compiles successfully, then the query plan is returned. If an error occurs, then JSON with an error element with message is returned, such as:

```
{
 "error": "operator ';' expected; ErrorToken(unclosed string literal)"
}
```

## Example

The following example returns the plan for the query:

```
SELECT * FROM `/data/SampleJSON` WHERE state="WA"
```

**NOTE:** The query shown in the above request must be HTML encoded.

## Request

```
GET http://localhost:20223/compile/fs?q=SELECT * FROM `/data/SampleJSON` WHERE state="WA"
```

## Response

```
Mongo db.getCollection("SampleJSON").aggregate(
 [{
 "match": {
 "state": "WA"
 }
 }, {
 "out": "tmp.gen_0"
 }], {
 "allowDiskUse": true
 });
db.tmp.gen_0.find();
```

### 7.3.4 Retrieving Data

Retrieves data from the specified path. Pagination is supported.

#### URL

```
/data/fs/{path}
```

where {path} is a path to the data to retrieve.

#### Method

GET

#### Headers

Header	Description	Re- quired	Default
Accept Accept- Encoding	Specifies the format of the response body. Use the value <i>gzip</i> to compress the output.	Optional Optional	<i>application/json;mode=precise</i> No compression

The following values are supported for the *Accept* header:

Value	Description
None	“Human-Readable” results, one result per line. Note: not parseable as a single JSON object.
application/json	Nicely formatted JSON array
application/json;mode=precise	[Precise JSON]{#precise-json}, one result per line
text/csv	Comma-separated results. See Note below.

#### Example

The following example returns data at the path */data/SampleJSON*. Data for the 10th and 11th items are returned.

#### Request

```
GET http://localhost:20223/data/fs/data/SampleJSON?offset=10&limit=2
```

```
Headers: Accept: application/json
```

#### Response

```
[{
 "city": "WESTOVER AFB",
 "state": "MA",
 "pop": 1764,
 "loc": [-72.558657, 42.196672]
}, {
```

```

 "city": "CUMMINGTON",
 "state": "MA",
 "pop": 1484,
 "loc": [-72.905767, 42.435296]
 }
]
```

**Note:** If you remove the *Accept* header, then you will receive *Precise JSON*, which is the default format. The response would then look like this:

```

{
 "city": "WESTOVER AFB",
 "state": "MA",
 "pop": 1764,
 "loc": [-72.558657, 42.196672]
}, {
 "city": "CUMMINGTON",
 "state": "MA",
 "pop": 1484,
 "loc": [-72.905767, 42.435296]
}
```

### 7.3.5 Replacing Data

Replaces data from the specified path.

Data is placed in the PUT body in the format of one JSON object per line. If successful, it replaces the existing data with the new data. If unsuccessful, it returns an error and the existing data is unchanged.

**Note:** An error will be returned if the path is a path to a view rather than a file.

#### URL

/data/fs/{path}

where {path} is a path to the data to replace.

#### Method

PUT

#### Headers

Header	Description
--------	-------------

#### PUT body

The data to replace, one JSON object per line.

Content-Type

Specifies the format of the PUT body. Set to *application/ldjson;mode=precise*

### Example

The following example replaces the data at the path `/data/SampleJSON` with two items.

### Request

```
PUT http://localhost:20223/data/fs/data/SampleJSON
```

```
Headers: Content-Type: application/ldjson;mode=precise
```

### Response

Status code 200 if successful. No response body.

If unsuccessful, then JSON is returned with two elements: *error*, which contains a short description of the error, and *detail*, which contains more detailed information. For example:

```
{
 "error": "some uploaded value(s) could not be processed",
 "details": [{
 "detail": "JSON contains invalid suffix content: , value: Str(parseerror: {
 \"city\": \"NEVERLAND\",
 \"state\": \"ZZ\",
 \"pop\": 2354,
 \"loc\": [-73.658, 41.443]
 },)
 }]
}
```

## 7.3.6 Appending Data

Appends data at the specified path.

Data is placed in the POST body in the format of one JSON object per line. If successful, it appends the existing data with the new data. If unsuccessful, it returns an error and the existing data is unchanged.

**Note:** An error will be returned if the path is a path to a view rather than a file.

### URL

```
/data/fs/{path}
```

where `{path}` is a path to the data to be appended.

### Method

```
POST
```



## Headers

Header	Description
Content-Type	Specifies the format of the PUT body. Set to <i>application/ldjson;mode=precise</i>

## Example

The following example appends one item to the data at the path */data/SampleJSON*.

## Request

```
POST http://localhost:20223/data/fs/data/SampleJSON
```

```
Headers: Content-Type: application/ldjson;mode=precise
```

## Response

Status code 200 if successful. No response body.

If unsuccessful, then JSON is returned with two elements: *error*, which contains a short description of the error, and *detail*, which contains more detailed information. For example:

```
{
 "error": "some uploaded value(s) could not be processed",
 "details": [{
 "detail": "JSON contains invalid suffix content: ,; value: Str(parseerror: {
 "city": "UTOPIA",
 "state": "ZZ",
 "pop": 2354,
 "loc": [-75.757,
 40.941
]
 },)
 }]
}
```

### 7.3.7 Deleting Data

Removes all data at the specified path.

**Note:** An error will be returned if the path is a path to a view rather than a file. Views may be added or deleted using the */mount* API requests.

Single files are deleted atomically, meaning that the equivalent MongoDB collection is removed, rather than a collection's documents being removed individually until the collection is empty.

## URL

```
/data/fs/{path}
```

where *{path}* is a path to the data to be deleted.

### Method

DELETE

### Example

The following example deletes all data at the path `/data/SampleJSON`.

### Request

```
DELETE http://localhost:20223/data/fs/data/SampleJSON
```

### Response

Status code 200 if successful. No response body.

If unsuccessful, then JSON is returned with two elements: *error*, which contains a short description of the error, and optionally *detail*, which contains more detailed information. For example:

```
{ "error": "./BadPath: doesn't exist" }
```

## 7.3.8 Moving Data

Moves data from one path to another. The origin path is specified in the URL and the destination path is specified in the *Destination* header. Single files are moved atomically.

**Note:** An error will be returned if either the origin or destination path is a path to a view rather than a file. Views may be moved using the `/mount` API requests.

Single files are moved atomically, meaning that the equivalent MongoDB collection is moved, rather than a collection's documents being moved individually.

### URL

`/data/fs/{path}`

where `{path}` is a path to the data to be moved.

### Method

MOVE

### Headers

Header	Description
Destination	Path to the new location of the data

## Example

The following example moves all data at the path `/data/SampleJSON` to `/data/SampleJSON2`.

## Request

```
MOVE http://localhost:20223/data/fs/data/SampleJSON
```

```
Headers: Destination: /data/SampleJSON2
```

## Response

Status code 200 if successful. No response body.

---

## 7.3.9 Retrieving a Mount

Retrieves the configuration for the mount point at the specified path.

## URL

```
/mount/fs/{path}/
```

where `{path}` is a path for the mount to retrieve.

**Note:** Be sure to end with a slash.

## Method

```
GET
```

## Example

The following example returns mount at the path `/`.

## Request

```
GET http://localhost:20223/mount/fs/
```

## Response

```
{
 "mongodb": {
 "connectionUri": "mongodb://myusername:mypassword@myserver.example.com:20223/data"
 }
}
```

### 7.3.10 Deleting a Mount

Deletes the MongoDB mount point at the specified path.

If there is no mount at the specified path, the request succeeds, but with no response.

#### URL

`/mount/fs/{path}/`

where `{path}` is a path for the mount to delete.

**Note:** Be sure to end with a slash.

#### Method

DELETE

#### Response

Returns the text “*deleted {path}*”, where *{path}* is the path to the deleted mount.

#### Example

The following example deletes mount at the path `/`.

#### Request

```
DELETE http://localhost:20223/mount/fs/
```

#### Response

```
deleted /
```

---

### 7.3.11 Changing the Port

Shuts down the running instance and restarts the server on the specified port.

The port is specified in the PUT body.

#### URL

`/server/port`

#### Method

PUT

## Response

Returns the text “*changed port to {port-number}*”, where *{port-number}* is the new port number.

## Example

The following example changes the port number from 20223 to 20224.

## Request

PUT http://localhost:20223/server/port
----------------------------------------

PUT body: 20224
-----------------

## Response

changed port to 20224

---

## 7.3.12 Setting the Port to the Default Value

Shuts down the running instance and restarts the server on the default port, which is 20223.

## URL

/server/port

## Method

DELETE

## Response

None

## Example

The following example sets the port number from 20224 back to the default.

## Request

DELETE http://localhost:20224/server/port \\\"\\\"\\\"\\\"



---

## Administration Guide - Community Edition

---

This Administration Guide can assist with installing and configuring SlamData. For information on how to use SlamData from a user perspective see the SlamData User Guide

### 8.1 Prerequisites

- 2 GB available memory
- 300 MB disk space
- Credentials to one or more databases
- Latest version of Chrome, Firefox, Safari or Internet Explorer
- Java 1.8 if using Linux (*Java Runtime is included with OS X and Windows installers*)
- We recommend the latest version of [SlamData Community Edition](#)

---

### 8.2 Limitations

- When using MongoDB, version 2.6 or newer is required

---

### 8.3 Installing SlamData

SlamData may be installed on a workstation or a server. Workstation installation is typically best for Administrator or Business Analyst work and only needs to run when something must be configured or changed. Server installation is best for providing services for a larger group of users (or SaaS environment) and should remain running at all times for end user access.

### 8.3.1 Mac OS X

SlamData has been tested and runs on OS X versions 10.10 and newer. It may run on older versions but has not been tested and is not yet supported.

1. [Download SlamData](#)
2. Open the downloaded disk image (.dmg file)
3. Double-click the installer. You may need to adjust 'Security & Privacy' settings to allow downloaded apps from Anywhere.
4. Complete the installation setup program

### 8.3.2 Linux

SlamData has been tested and runs on Ubuntu version 14.04 and newer and CentOS version 7 and newer. It may run on different Linux distributions and versions but has not been tested and is not yet supported.

1. [Download SlamData](#)
2. Ensure you have the [Java 8 JRE](#) or [Java 8 JDK](#) installed.
3. Make the installer executable: `chmod +x slamdata_unix_version.sh`
4. Run the installer, providing answers to the prompts: `./slamdata_unix_version.sh`

### 8.3.3 Windows

SlamData has been tested and runs on Windows 7 Desktop and newer and Windows Server 2008 and newer. It may run on older versions but has not been tested and is not yet supported.

1. [Download SlamData](#)
  2. Run the installer
  3. Complete the installation setup program.
- 

## 8.4 Launching SlamData

SlamData is comprised of a frontend interface and a backend analytics engine. Launching the SlamData application will start both.

### 8.4.1 Mac OS X

1. Open the Applications folder
2. Double-click on the SlamData icon

A new browser window or tab will open displaying the SlamData interface. The SlamData icon will appear in the OS X dock. As with other dock applications the SlamData icon may be right-clicked and the application terminated.



### 8.4.2 Linux

1. Change directory to the location of the SlamData executable: `cd SlamData-version`
2. Execute the SlamData executable: `./SlamData`

Some Linux systems may not launch a browser automatically. If this is the case, open a browser and point it to the following URL: <http://localhost:20223/slamdata>

### 8.4.3 Windows

1. Open the Start menu
  2. Click on the newly installed SlamData icon or use the app search bar and type `slamdata` and press return to launch it. Select appropriate network security settings if prompted.
- 

## 8.5 Connecting to a Database

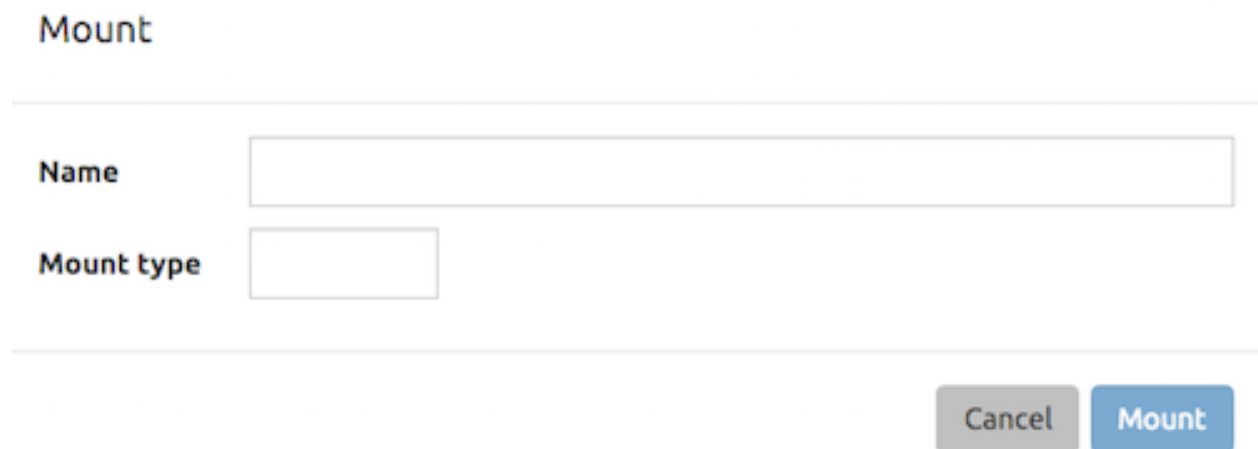
Connecting to a database is the first step to analyzing data. SlamData does not provide a database to connect to. As more databases are supported by SlamData they will be listed below.

### 8.5.1 MongoDB

**Note:** Only MongoDB versions 2.6 and newer are supported by SlamData.

To connect to MongoDB click on the Mount  icon in the upper right.

A mount dialog will be presented:



The image shows a 'Mount' dialog box. It has a title bar with the word 'Mount' in blue. Below the title bar, there are two labels: 'Name' and 'Mount type'. Each label is followed by a text input field. The 'Name' field is wider than the 'Mount type' field. At the bottom right of the dialog, there are two buttons: 'Cancel' (grey) and 'Mount' (blue).

Fig. 8.1: SlamData Mount Dialog

Enter a name for the database mount. This name is used in the SlamData UI as well as SQL2 query paths. Use a name that makes sense for the environment. For instance if this database were hosted on Amazon AWS/EC2 it might be named `aws` or `aws-1`.

Select **MongoDB** as the mount type. Other mount types will be discussed later. Once a Mount type is selected additional fields will appear in the dialog based on the mount type selected.

Use the following table to assist in providing example values for the remaining fields:

Field	Example
Host	db.example.com
Port	27017
Username	joe
Password	*****
Database	joesdb
Settings	

An example form might look like this:

**Note:** When using MongoDB the database field value should be the database the username and password will authenticate against. This value will depend on which database the user was created in; as such it could be `admin`, the name of the user or something completely different.

Click **Mount** to mount the database in SlamData.

## 8.5.2 Several Mounts

After mounting several databases the SlamData UI might look like the following image. In this image there are three separate mounts named `aws`, `kirk` and `macbook`, the last likely representing a locally mounted database.

**Note:** Keep in mind that with SlamData Advanced you can limit which databases, collections and charts can be seen based on OAuth authentication and the SlamData authorization model.

## 8.6 Advanced Configuration

### 8.6.1 Configuration File

The SlamData configuration file allows an administrator to change settings such as the port number SlamData listens on, the mounts available and more. The location of the configuration file depends upon the operating system being used.

Operating System	Configuration File Location
Apple OS X	<code>\$HOME/Library/Application Support/quasar/quasar-config.json</code>
Microsoft Windows	<code>%HOMEDIR%AppDataLocalquasarquasar-config.json</code>
Linux (various vendors)	<code>\$HOME/.config/quasar/quasar-config.json</code>

SlamData Advanced edition has an extended format of the configuration file which is not covered in this document. Please refer to the SlamData Advanced Administration Guide that you were given after purchase of the Advanced edition.

An example configuration file might appear like this:

```
{
 "server": {
 "port": 8080
 },
 "mountings": {
 "/aws/": {
 "mongodb": {
 "connectionUri": "mongodb://myUser:myPass@aws-box.example.com:27017/admin"
 }
 }
 }
}
```

## Mount

<b>Name</b>	<input type="text" value="aws"/>		
<b>Mount type</b>	<input type="text" value="MongoDB"/>		
<b>Server(s)</b>			
<b>Host</b>	<input type="text" value="db.example.com"/>	<b>Port</b>	<input type="text" value="27017"/>
	<input type="text"/>		<input type="text"/>
<b>Authentication</b>			
<b>Username</b>	<input type="text" value="joe"/>		
<b>Password</b>	<input type="password" value="*****"/>		
<b>Database</b>	<input type="text" value="joesdb"/>		
<b>Settings</b>			
<b>Name</b>	<b>Value</b>		
<input type="text"/>	<input type="text"/>		

Fig. 8.2: SlamData MongoDB Dialog



Fig. 8.3: SlamData Multiple Mounts

```

 }
 },
 "/kirk/": {
 "mongodb": {
 "connectionUri": "mongodb://myUser2:myPass@win-box.example.com:27017/admin?ssl=true"
 }
 },
 "/macbook/": {
 "mongodb": {
 "connectionUri": "mongodb://localhost:27017"
 }
 }
}
}

```

### 8.6.2 Mount Options

The mount dialog will display the appropriate fields based on the mount type selected. For each database type that SlamData supports a section below explains the options available.

#### MongoDB

For MongoDB the values listed in the Connection Options on the MongoDB web site are supported. As of MongoDB 2.6 these options are listed below.

Options	Example	Description
ssl	true	Enable SSL encryption
connectTimeoutMS	15000	The time in milliseconds to attempt a connection before timing out
socketTimeoutMS	10000	The time in milliseconds to attempt a send or receive on a socket before the attempt times out

#### SQL2 View

SQL2 Views are covered in detail in the SlamData Users Guide.

#### Other Databases

Support for other relational and NoSQL databases is coming in 2016.

### 8.6.3 Enabling SSL

If you have trouble following the steps below you may also view our [SSL tutorial video](#).

If a database connection supports SSL encryption, which is to say encryption between a client and server such as SlamData and the database, additional configuration is necessary.

The backend engine of SlamData is written in [Scala](#) and executes within a Java Virtual Machine (JVM). To enable SSL encryption several options must be passed to the JVM when running SlamData. SlamData simplifies this by allowing these options to be listed in a text file that the SlamData launcher will reference when executed. The file location for each operating system is listed below:

Operating System	File Location
Apple OS X	/Applications/SlamData-version.app/Contents/vmoptions.txt
Microsoft Windows	C:\Programs Files (x86)\slamdata-version\SlamData.vmoptions
Linux (various vendors)	\$HOME/slamdata-version/SlamData.vmoptions

There are two important options that must be passed to the JVM at startup to enable SSL. These options point the JVM to a Java key store (JKS).

JVM Options	Purpose
<code>javax.net.ssl.trustStore</code>	The location of the encrypted trust store file
<code>javax.net.ssl.trustStorePassword</code>	The password required to decrypt the trust store file

The example contents of the file may look something like this:

```
-Djavax.net.ssl.trustStore=/Users/me/ssl/truststore.jks
-Djavax.net.ssl.trustStorePassword=mySecretPassword
```

This guide does not provide exhaustive steps to create a Java key store in every scenario, but we hope the following simple example is helpful. Assuming you are hosting MongoDB with a service provider, that provider might make a signed (or self-signed) certificate available so that MongoDB can connect securely via SSL. Also assuming this is in the form a `your_provider.crt` text file, you might follow these steps based on the JKS configuration above:

**First** - import the certificate into your Java trust store:

```
keytool -import -alias "your_providers_name" -file your_provider.crt \
-keystore /users/me/ssl/truststore.jks -noprompt -storepass mySecretPassword
```

**Second** - ensure you've made the appropriate changes to the JVM options file referenced above.

**Third** - restart SlamData so it reloads the JVM options file and picks up the new certificate in the JKS.

**Fourth** - Mount the database with SSL as shown in the attached screenshot:

## Mount

---

<b>Name</b>	<input type="text" value="service_provider"/>		
<b>Mount type</b>	<input type="text" value="MongoDB"/>		
<b>Server(s)</b>			
<b>Host</b>	<input type="text" value="service_provider_host_name"/>	<b>Port</b>	<input type="text" value="27017"/>
	<input type="text"/>		<input type="text"/>
<b>Authentication</b>			
<b>Username</b>	<input type="text" value="myuser"/>		
<b>Password</b>	<input type="password" value="....."/>		
<b>Database</b>	<input type="text" value="admin"/>		
<b>Settings</b>			
<b>Name</b>		<b>Value</b>	
<input type="text" value="ssl"/>		<input type="text" value="true"/>	
<input type="text"/>		<input type="text"/>	

---

Fig. 8.4: SlamData SSL Mounts





---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`