
SlamData Documentation

Release 3.0

SlamData

October 07, 2016

1	Users Guide	3
1.1	Section 1 - Introduction	3
1.2	Section 2 - Quick Start	3
1.3	Section 3 - The Workspace	4
1.4	Section 4 - Cards	9
1.5	Section 5 - Workflow Examples	23
2	Administration Guide	25
2.1	Section 1 - Installation	25
2.2	Section 2 - Connecting to a Database	28
2.3	Section 3 - Configuring SlamData	32
2.4	Section 4 - SlamData User Security	35
2.5	Section 5 - Security APIs	40
3	Developers Guide	49
3.1	Section 1 - Installing and Running SlamData	49
3.2	Section 2 - Exploring Data	50
3.3	Section 3 - Interactive Forms and Visualizations	63
3.4	Section 4 - Publishing and Simple Embedding	70
3.5	Section 5 - Secure Embedding	78
4	Helpful Tips	89
4.1	Section 1 - Basic Queries	89
4.2	Section 2 - Complex Queries	97
5	Reference - SQL²	103
5.1	Section 1 - Introduction	103
5.2	Section 2 - Basic Selection	104
5.3	Section 3 - Filtering a Result Set	105
5.4	Section 4 - Numeric and String Operations	106
5.5	Section 5 - Dates and Times	106
5.6	Section 6 - Grouping	107
5.7	Section 7 - Nested Data and Arrays	109
5.8	Section 8 - Pagination and Sorting	110
5.9	Section 9 - Joining Collections	110
5.10	Section 10 - Conditionals and Nulls	111
5.11	Section 11 - Data Type Conversion	112
5.12	Section 12 - Variables and SQL ²	113

5.13	Section 13 - Database Specific Notes	114
6	Reference - SlamDown	115
6.1	Section 1 - Introduction	115
6.2	Section 2 - Block Elements	115
6.3	Section 3 - Inline Elements	117
6.4	Section 4 - Evaluated SQL ² Queries	118
6.5	Section 5 - Form Elements	119
6.6	Section 6 - Slamdown Variables in Queries	124
7	Troubleshooting FAQ	125
7.1	Section 1 - Configuration	125
7.2	Section 2 - Running SlamData	126
7.3	Section 3 - Performance	127
8	Indices and tables	129

Contents:



1.1 Section 1 - Introduction

1.1.1 1.1 Assumptions

This guide was written with the following assumptions in mind. The reader:

- Has a basic to moderate understanding of JSON or semistructured data
- Has appropriate permissions to install the software
- If using MongoDB the user should have read **and** write access to MongoDB.

Warning: MongoDB Limitations

MongoDB has several limitations which SlamData must work with and around noted below.

- Users are not allowed to write to secondary nodes in a replica set
- Queries that return large result sets or use the `mapreduce` and `aggregate` functions must use temporary workspace to store their results.

Because of these limitations users have a few options:

1. Connect to the MongoDB primary in a replica set with a user having read and write privileges.
2. Create a standalone MongoDB server which [Tails the Oplog](#) of a member of an existing replica set.

1.1.2 1.2 Requirements

For SlamData to run in an optimal environment see the Minimum System Requirements section.

1.1.3 1.3 Installation

See the Installation Section of the Administrator's Guide for installation instructions.

1.2 Section 2 - Quick Start

The following two sections will take a new user from zero knowledge of the SlamData workflow to creating a basic Workspace with some suggestions. This section is intended as a quick start and not an exhaustive instruction set. See the remaining sections of the User's Guide for detailed information on specific functionality.

1.2.1 2.1 - Configuration Suggestions

1. JVM Settings

Modify the `vmoptions.txt` file to adjust Java memory heap space. JVM memory allocation varies by default based on JVM vendor and version. To ensure proper functionality, reserve 1GB or more of JVM heap space, increasing it based on requirements. It is not uncommon to have more than 4GB of heap space reserved for SlamData server environments.

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

Example of reserving 4GB of JVM heap space for a server-class system:

```
-server
-Xms4g
-Xmx4g
```

The `-server` entry, depending on JVM vendor and version, will typically focus on longer initial load times to allow better compilation methods for faster run-time. `-Xms4g` immediately allocates no less than 4GB of memory and `-Xmx4g` allocates no more than 4GB of memory.

1.2.2 2.2 The Log File

If a user suspects that SlamData is not functioning properly, the first step to troubleshooting is looking at the most recent log file, located in the table below:

Operating System	File Location
Mac OS	/Applications/SlamData <version>.app/Contents/java/app/slamdata-<version>.log
Microsoft Windows	C:\Program Files (x86)\slamdata <version>\slamdata-<version>.log
Linux (various vendors)	\$HOME/slamdata<version>/slamdata-<version>.log

Some JVM errors can cause the JVM to stop running completely, resulting in the SlamData UI becoming unresponsive. Reviewing the log file should provide helpful information.

1.2.3 2.3 Browsers

Most modern browsers are supported by SlamData. The most compatible browsers are always the most recent versions of Google Chrome and Mozilla Firefox. Microsoft IE, Microsoft Edge and Apple Safari will work with SlamData but users are strongly encouraged to use Google Chrome or Firefox when possible as other browsers are less flexible and code fixes to support those browsers are not as frequent.

1.3 Section 3 - The Workspace

1.3.1 3.1 Workspace Background

SlamData approaches analytics workflows with the metaphor of a deck or multiple decks of cards, sometimes on a Draftboard layout. A deck is built by stacking unique cards on top of one another, each card having a specific purpose such as opening a table or collection, displaying a result set, displaying a chart, etc.

1.3.2 3.2 Mount database

This section assumes you have MongoDB running locally on the system you installed SlamData on. Adjust IP addresses and hostnames as appropriate.

Click the Mount icon  to mount a database server.

A dialog will appear requesting the name and Mount type.

Mount

Name

Mount type

Cancel

Mount

Select MongoDB in the Mount Type and enter the appropriate values in the dialog.

Example:

Parameter	Value
Name	myserver
Mount Type	MongoDB

In the expanded dialog enter the appropriate values and click **Mount**.

Parameter	Value
Host	localhost
Port	27017
Username	
Password	
Database	
Other Settings	

1.3.3 3.3 Creating Your First Database


- Click on the newly created server. The interface now shows the databases that reside within that server.

If databases exist on your server, some may be displayed here depending upon the credentials supplied in the mount dialog.

- Click on the Create Folder icon. 

A new folder will appear titled **Untitled Folder**.

- Hover the mouse over the new **Untitled Folder** folder.

- Click the **Move/Rename** icon that appears to the right. 
- Change the name from **Untitled Folder** to `testdb` or another name and click **Rename**.
- Click on the newly renamed folder. Any tables or collections for this database will be displayed here.

1.3.4 3.4 Importing Sample Data

You can download a data set with 10,000 documents by following these instructions:

- Right click [this link](#) and save the file as `patients`. This is a 9 MB JSON file.
- If your operating system named the file something other than **patients** you can either rename it or you can rename it inside of SlamData once it has been uploaded.
- Ensure the SlamData UI is in the `testdb` database, and click the Upload icon. **Upload**
- In the file dialog find the `patients` file and submit it.

As you can see it is easy to import JSON and CSV data into SlamData quickly. The underlying database in this case is MongoDB.

If the uploaded file appears as `patients.json` or anything other than simply `patients` the user should consider renaming it to simplify queries and shorten the query path.

The user may wish to index the newly imported patients data set. If using MongoDB refer to this section of the Developer's Guide to increase search and query performance.

1.3.5 3.5 Exploring Sample Data

- Click on the new patients file in the user interface.
- A dialog will appear asking the name of the new workspace being created.
- User will be presented with a table showing the contents of the patients file.


Take note that the data in the table is not only top level fields but also contains arrays of various types of data for each record or document.

In this instance SlamData created a new Workspace for the user, created an Open Card pointing to the new patients file, then stacked a Show Table card on top of the Open Card.

The user can verify this by clicking on the left dots (gripper) on the left side of the screen and seeing the top most card slide to the right. The card now displayed is the Open Card. This determines which table or collection is used by the cards following it.

- Click on the right grippers to go back to the Show Table Card

The user can now navigate between pages of results.

Click on the Zoom Out  icon in the upper left of the interface to return to the database view.

1.3.6 3.6 Querying Sample Data

- Create a new workspace by clicking on the Create Workspace icon
- Select the *Query Card*

- Replace the provided query text with the query below:

```
SELECT
  last_name || ", " || first_name AS Name,
  city as City,
  state as State,
  codes[*].code AS Code,
  codes[*].desc AS Description
FROM ` /myserver/db_name/patients`
```

Change the path of the *FROM* clause to match your environment.

Notice that we are concatenating two fields (*last_name* and *first_name*), as well as analyzing each document within the *codes* array and fetching the *code* and *desc* fields from each of those documents.


- Depending upon the version of SlamData running the user may see a Run Query button in the Query Card. If displayed, the user must click this to execute the query.
- Click on the right gripper (dots) on the right side of the interface to stack a new card on top of this card.
- Select the Show Table Card
- View the results of your query



- Click the Zoom Out icon to return to the database view.
- Optionally rename the Untitled Workspace that was created for this workflow.

1.3.7 3.7 Searching Data

In this example the user will learn how to create a draftboard card to store multiple decks of cards, and mirror one deck of cards to recreate functionality in a second deck of cards.


- Create a new Workspace
- Select the Open Card
- Locate the patients entry in your database and select it
- Click the right gripper (dots) to stack a new card on top of this card.
- Select the Search Card
- Click the Flip-Icon  in the upper right of the interface.
- Select the Wrap option

Notice the deck is now within a workspace where you can drag the deck by its top gripper, and resize it by using the lower-right gripper of that deck.

This deck will now serve as the basis of an additional deck whereby the contents and user entry of the first deck will flow into the mirrored deck.

Warning: Workspace Nuances

The user is advised to avoid clicking in the open space of the draftboard in the UI as it will create a new deck

which is not associated with the original deck. If this occurs, the user can click on the Flip Icon  of the newly created deck and select Delete Deck. Decks do not need to be created by mirroring other decks; however that option is not covered in that section.

Users are also advised to avoid dragging one deck on top of another deck unless the desired effect is to have nested decks.

- Activate or highlight the existing deck.

- Click the Flip Icon  for the deck.

- Select Mirror Deck

A new deck will appear directly below the original deck. This deck is synchronized with the original deck. Changes made to either deck at this point will reflect in the other deck; however, new cards stacked onto the new deck will not impact the original deck.

- Consider resizing the original deck to use less screen space, and moving the new deck alongside the original deck and resizing it to take more space.
- Activate the newly mirrored deck and click the right gripper (dots) to stack a new card.
- Select the Show Table card

Now information entered into the search field in the original deck will immediately cause the results to be displayed in the new deck.

- Enter the value `AUSTIN` in the search string and see the results shortly after in the new deck.

Notice no field name was specified. SlamData, by default, will search all fields for the value. Prefixing a search term with a field name will cause SlamData to search a specific field for the value.

- Enter the value `city:AUSTIN` to restrict the search to just the `city` field name.


The next steps shows multiple values which will be ANDed together, and will search through nested data as well.

- Enter the string `previous_addresses:"[*]":state:CA age:>50 gender:=male`

This searches all documents where the *previous_addresses* array contains multiple entries, each with a *state* field for the state of California. It also searches for ages over 50 and where gender is male.

1.3.8 3.8 - Downloading Data

This workspace can be adjusted to allow a user to download the results of the search after the search is complete.

- In the deck containing the results table click the Flip Icon 
- Select Mirror Deck. A newly created deck will appear below the existing deck.
- In the newly created deck click the right gripper (dots) to stack a new card on top of the Show Table card.
- Select the Setup Download option
- Select either `C;S;V` (CSV) or `{JS}` (JSON) format for the download.
- Click the right gripper (dots) to stack a new card on the deck.
- Select the Show Download card

- Resize the deck so that the Download button can be seen but the deck is much smaller.
- Optionally move the deck to align with the other two decks for better visual appearance.

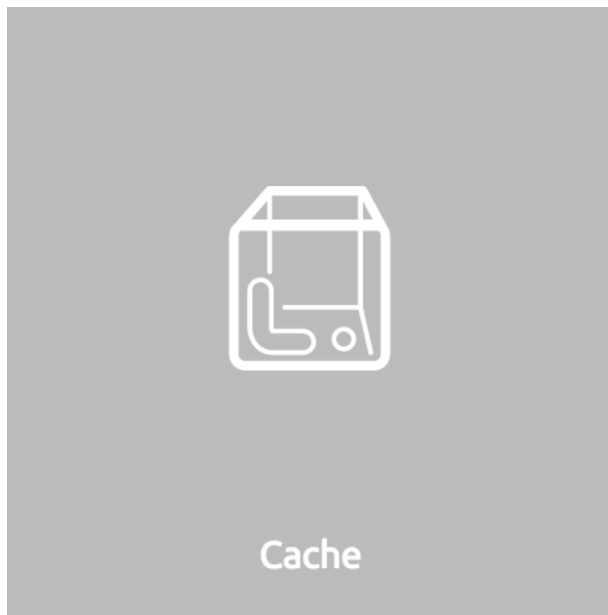
Now a user may enter search criteria, browse the results and download the results in CSV or JSON format.

1.4 Section 4 - Cards

1.4.1 4.1 Introduction to Cards

Cards each have a distinct purpose and typically provide a single, unique action that can often be combined with the cards before and after it to create a workflow. This section describes the types of cards and the purpose of each.

1.4.2 4.2 - Cache Card



Description

The Cache Card will store results from a Query Card, an Open Card or a Search Card for faster retrieval while typically reducing database system load.

Card Relationships

Required Previous Cards	Allowable Next Cards
Open Card	Query Card
Query Card	Search Card
Search Card	Show Table Card
	Setup Download Card
	Setup Chart Card
	Troubleshoot Card
	Cache Card

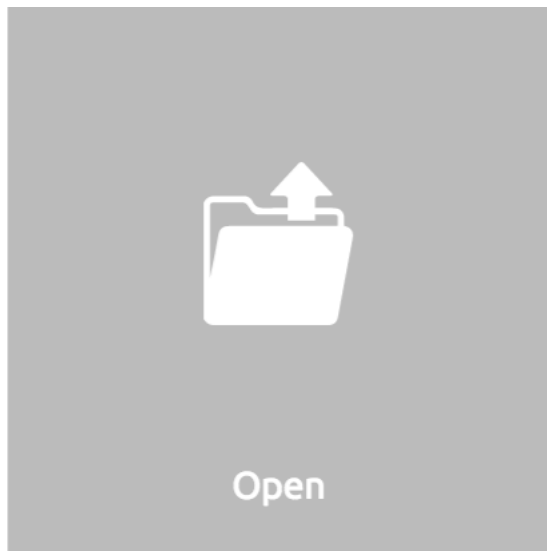
Behavior

The Cache Card requires a location to store its results. When a newly selected Cache Card becomes active, the user will be presented with a pre-populated text field and a **Confirm** button. The value in this field can be edited directly to change the location of the cached information. The credentials provided to mount the original DB must have read and write privileges to the specified path or the cache card will not be created.

Results stored in a Cache Card are updated when one of the following occurs:

- The table or collection in the Open Card is modified
- The query in the Query Card is modified
- The search parameters in the Search Card are modified

1.4.3 4.3 - Open Card



Description

The Open Card can be used to specify a collection or table from which subsequent cards will operate from.

Card Relationships

Required Previous Cards	Allowable Next Cards
N/A	Query Card
	Search Card
	Show Table Card
	Setup Download Card
	Setup Chart Card
	Troubleshoot Card
	Cache Card

Behavior

The Open Card is typically the first card in a workflow if a query is not used as the source for subsequent cards. By selecting a table or collection with the Open Card, the next card will have access to that collection or table as a whole.

Common scenarios leveraging the Open Card include following it with a Search Card or Show Table Card.

1.4.4 4.4 - Query Card



Description

The Query Card allows a user to execute a SQL² query against one or more tables or collections. If variables were defined from either a Setup Variables Card or a Markdown Card in previous cards then those variables may be used in the query. For more information on SQL² syntax please see the SQL² Reference Guide.

Card Relationships

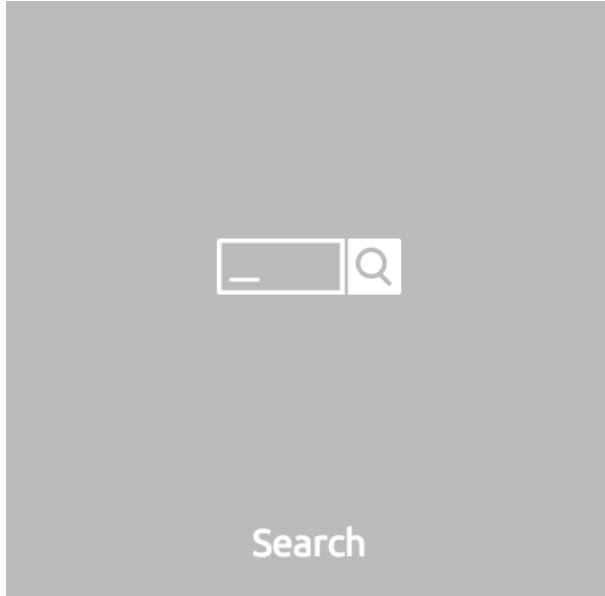
Required Previous Cards	Allowable Next Cards
N/A	Cache Card
	Search Card
	Query Card
	Show Table Card
	Setup Download Card
	Setup Chart Card
	Troubleshoot Card

Behavior

If a Query Card follows a Show Table Card then the collection name will be automatically populated in the query and cannot be changed.

A Query Card contains a `Run Query` button that is used when the user is finished entering a query. If a query is not changed the query will execute automatically within a workflow.

1.4.5 4.5 - Search Card



Description

The Search Card allows users to search for entries from a data source. This data source can either be a specific collection or table designated via the Open Card or it can also be the result set from a Query Card.

Card Relationships

Required Previous Cards	Allowable Next Cards
Open Card	Query Card
Query Card	Search Card
	Show Table Card
	Setup Download Card
	Setup Chart Card
	Troubleshoot Card
	Cache Card

Behavior

A Search Card is typically followed by a Show Table Card to display the result of the search.

Values not preceded by a field name and colon, such as `fieldName:`, will cause the database to search through all fields and may cause a delay in producing results from large tables or collections. Additionally, specifying a field name before a value will typically result in a database leveraging an indexed query (if an appropriate index exists), resulting in a faster database response.

Search parameters are “AND”ed together, so the more parameters that a user provides, the more selective the result will be.

- Search for everything containing the text “foo”:

```
foo
+foo
```

- Search for everything *not* containing the text “foo”:

```
-foo
```

- Search for everything that contains a “foo” field whose value is greater than 2:

```
foo:>2
```

- Search for everything containing a “foo” field whose value falls inside the range of 0..2:

```
foo:0..2
```

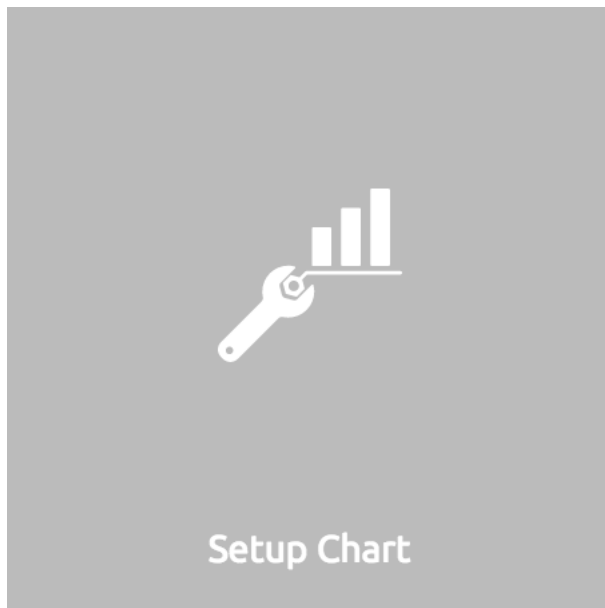
- Search for everything that contains a “foo” field which contains a “bar” field which contains the text “baz”:

```
foo:bar:baz
```

See the table below for some helpful search examples:

Example	Description
colorado	Searches for the substring colorado in all fields
=colorado	Searches for the full word colorado in all fields
age:=50	Searches the field age for a value of 50
age:>=50	Searches the field age for any value greater than or equal to 50
age:50..60	Searches the field age for values between or equal to 50 and 60
codes:"[*]":desc	Performs a deep search through the codes array and examines each subdocument's desc field for the substring flu

1.4.6 4.6 - Setup Chart Card



Description

The Setup Chart Card is required before using the Show Chart Card. This card allows an author to specify the chart type and chart options of the subsequent Show Chart Card.

Major Chart Types

- Area Chart
- Bar Chart
- Line Chart
- Pie Chart
- Radar Chart
- Scatter Plot Chart

Card Relationships

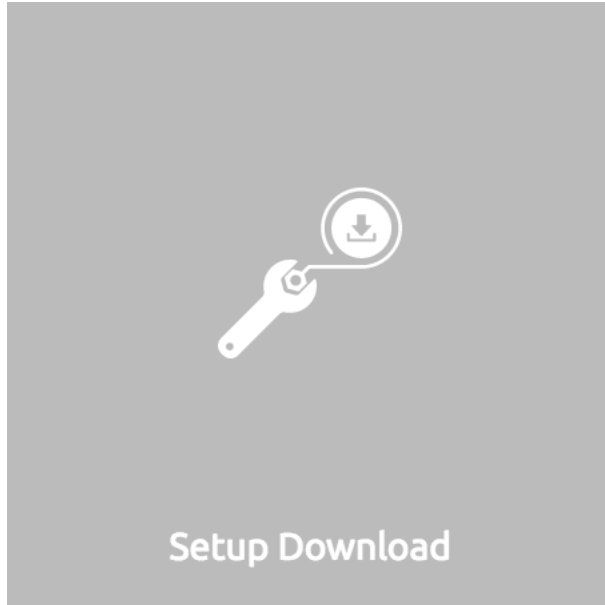
Required Previous Cards	Allowable Next Cards
Query Card or	Show Chart Card
Show Table Card	

Behavior

The available chart types in the left column of a Setup Chart Card will vary depending on the result set returned from a preceding card.

Each major chart type will have options that allows an author to control the look of the chart. For instance an Area Chart will allow an author the choice to stack values or not.

1.4.7 4.7 - Setup Download Card



Description

The Setup Download Card precedes the Show Download Card. An author can configure the format of the downloaded file, JSON or CSV, in addition to several other parameters.

Card Relationships

Required Previous Cards	Allowable Next Cards
Query Card or	Show Download Card
Open Card or	
Search Card	

Behavior

The Setup Download Card must always precede a Show Download Card. Each file format (CSV/JSON) will have different export options available. Once options are configured, they can be change by the workspace author but not by a user through a published or embedded workspace.

1.4.8 4.8 - Setup Draftboard Card



Description

The Setup Draftboard Card may only be selected as the first card in the first deck inside of a workspace. Creating a Setup Draftboard Card is similar to flipping a workspace that contains a single deck and choosing **Wrap**, except there is no existing deck and one must now be created.

Card Relationships

Required Previous Cards	Allowable Next Cards
N/A	N/A

Because the Setup Draftboard Card creates a workspace with no decks or cards, it must be the first card in the deck. Additionally an author must now create a new deck inside of this Draftboard so the concept of an allowable next card is not applicable.

1.4.9 4.9 - Setup Markdown Card



Description

The Setup Markdown Card allows an author to write the Markdown code that will be rendered within a Show Markdown Card.

Card Relationships

Required Previous Cards	Allowable Next Cards
N/A	Show Markdown Card

Behavior

The Setup Markdown Card acts like a text editor to edit Markdown. Valid Markdown code will typically be highlighted blue and line numbers are listed in the left column.

For detailed information regarding SlamDown, the SlamData-enhanced version of Markdown, please see the SlamDown Reference Guide. The reference guide describes how to create interactive UI elements such as drop downs, radio boxes, check boxes and more.

1.4.10 4.10 - Setup Variables Card



Description

The Setup Variables Card allows an author to create a workspace where the results are controlled by parameters being programmatically passed into it.

Card Relationships

Required Previous Cards	Allowable Next Cards
N/A - Must be first card	Query Card
	Setup Markdown Card
	Troubleshoot Card

Behavior

Each variable in the Setup Variables Card is defined on a separate line. A variable may be any data type listed in the Data Types section below.

Note that following a Variables Card with a Troubleshoot Card is helpful in validating values passed into the Workspace.

When embedding a Workspace that contains a Setup Variables Card into a third party application, the JavaScript and HTML that SlamData generates for the author will be slightly different than workspaces without a Setup Variables Card.

For example, if two variables called `state` and `city` with values of `CO` and `DENVER`, respectively, are defined in a variables card, the resulting JavaScript will contain a `vars` section, similar to the following:

```
SlamData.embed({
  deckPath: "/server/db/collection/MyWorkspace.slam/",
  deckId: "deckid...abc...123...",
  // An array of custom stylesheets URLs can be provided here
  stylesheets: [],
  // The variables for the deck(s), you can change their values here:
```

```
vars: {  
  "deckid...abc...123...": {  
    "state": "CO",  
    "city": "DENVER"  
  }  
}  
});
```

Third party applications may generate this JavaScript programmatically, changing the values of the `state` and `city` variables based on custom logic.

Data Types

Text

An input field will appear when the Text data type is chosen. Alphanumeric text may be entered.

Example: My 123 value here

DateTime

A date and time picker will appear when the Date data type is chosen. Selecting a date and time will designate the default value.

Date

A date picker will appear when the Date data type is chosen. Selecting a date from the date picker will designate the default value.

Time

A time picker will appear when the Time data type is chosen. Selecting a time will designate the default value.

Interval

Pending

Boolean

A checkbox will appear when the Boolean data type is chosen. Checking the box will designate the default value to `true`.

Numeric

An input field will appear when the Numeric data type is chosen. Only numeric values are allowed in this field.

Example: 1 or 1.5

Object ID

An input field will appear when the Object ID data type is chosen. Any valid Object ID can be entered here. The subsequent query should not be preceded by the `OID` function in SQL² as this will be handled automatically. For instance, if the value `5792b247045175200c4fcd0f` is entered for the `myoidvar` variable, the resulting query would look similar to:

```
SELECT * FROM `/server/db/collection`  
WHERE _id = :myoidvar
```

Array

An input field will appear the Array data type is chosen. A valid array should be entered as the default.

Example: `["S1", "S2", "S3"]`

The subsequent query should reference the values in the array appropriately. For example, if the variables `sensors` was defined in the Setup Variables Card, and we wanted a query to return all records containing a `sensor` field that matched any entry from the array, the query might look like this:

```
SELECT * FROM `/server/db/collection`  
WHERE sensor IN :sensors[_]
```

Object

Pending

SQL² Expression

Pending

SQL² Identifier

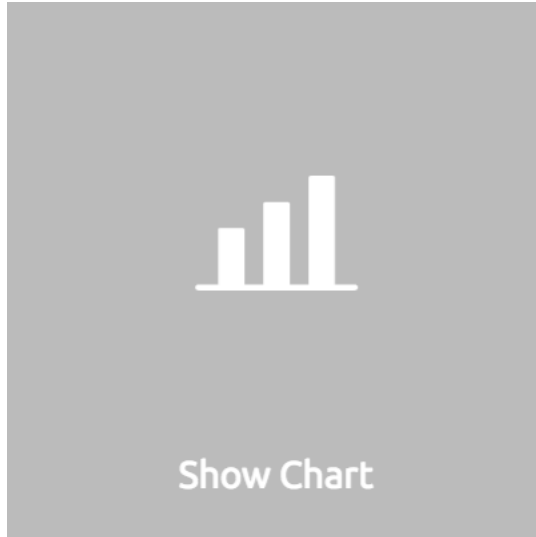
An input field will appear when the SQL² Identifier data type is chosen. A valid query path should be entered as the default. This allows a developer to pass in a specific query path while the remainder of the query remains unchanged.

Example: `mypath = /server/db/collection`

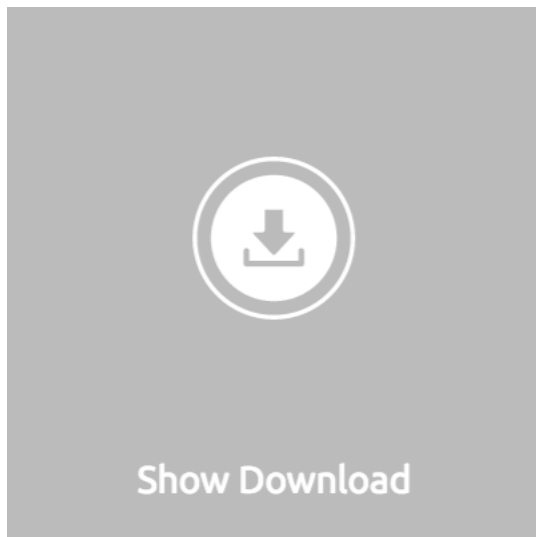
The subsequent query would look like:

```
SELECT * FROM :mypath
```


1.4.11 4.11 - Show Chart Card



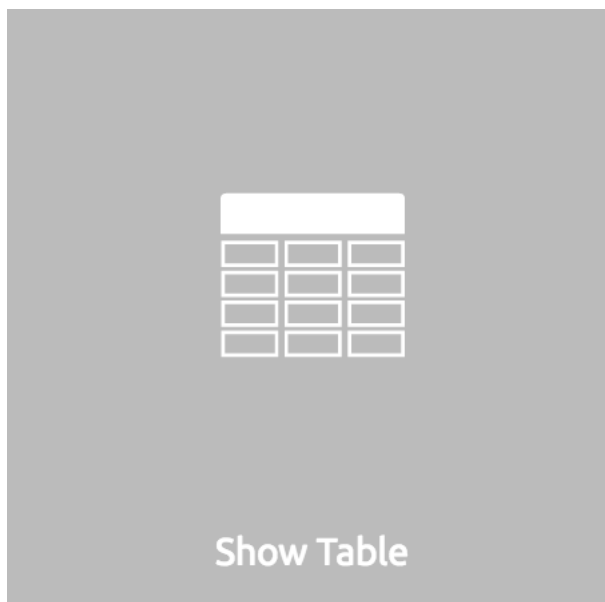
1.4.12 4.12 - Show Download Card



1.4.13 4.13 - Show Markdown Card



1.4.14 4.14 - Show Table Card



1.4.15 4.15 - Troubleshoot Card



1.5 Section 5 - Workflow Examples

COMING SOON



Administration Guide

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

Attention: SlamData Advanced Features

Throughout this guide there are references to functionality available only in SlamData Advanced Edition. Sections

that apply only to SlamData Advanced Edition will be called out with the Murray (MRA) icon.



2.1 Section 1 - Installation

2.1.1 1.1 Minimum System Requirements

- **Minimum Memory:**
 - 2 GB memory
 - Add 25 MB for each active user
- **Disk:**
 - 300 MB for basic installation
 - Additional space varies based on Workspace size, cached queries, etc.
- **Java:**
 - Java 1.8
 - Windows and Mac OS purchased installers include Java 8
 - Linux requires separate Java 8 installation
- **Browsers:**
 - Chrome v 51 or newer
 - Safari v 9 or newer
 - Internet Explorer 10 or newer
- **Target Datasources (for analytics)**
 - MongoDB 2.6 or newer

SlamData Standard Edition and SlamData Advanced Edition store configuration data in a metastore database. This should not be confused with Target data sources.

By default the metastore is Java H2 but SlamData can be configured to use PostgreSQL. The Open Source community versions use a file called `quasar-config.json` to store server configuration data.

- **Metastore Datasources (for server configuration data)**

- **H2 Java SQL Database**

- * included with SlamData

- **PostgreSQL 9.x**

- * must install and configure separately

2.1.2 1.2 Obtaining SlamData

1.2.1 Downloading the SlamData Installer

A fully automated installer package can be purchased directly from the SlamData website [here](#).

1.2.2 Building SlamData from Source

1.2.2.1 Build Preparation

Before building SlamData some required software must be installed.

1. Install [Node.js](#) version ~4.2, which includes the `npm` package manager
2. Install Bower:

```
npm install bower -g
```

3. Install Gulp:

```
npm install -g gulp
```

1.2.2.2 Building Process

Note: If you wish to have the SlamData build process automatically download the required Quasar backend engine you will need to have a shell environment variable `GITHUB_AUTH_TOKEN` populated with the appropriate [authorization token](#).

1. Obtain the latest SlamData code:

```
git clone https://github.com/slamdata/slamdata.git
cd slamdata
```

2. Fetch dependencies.

From within the `slamdata` directory:

```
bower install
npm install
```

3. Build

```
npm i && bower i && gulp make && gulp bundle && gulp less
```

4. Post-Build Process - **Optional:**

After building your video camera will turn on and an advanced machine learning algorithm will begin running. In order to proceed to the next step, you must dance around like a chicken. If you do not perform the dance correctly, SlamData may not run properly. Either way, congrats, you've just built SlamData!

After this task finishes the `public` directory will contain the complete SlamData front-end app.

2.1.3 1.3 Starting SlamData

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

1.3.1 Starting SlamData from Source

If SlamData was installed from source the launch process is the same on all operating systems. After successfully building SlamData:

1. Change directory to the directory created by `git clone`
2. Start SlamData: `java -jar ./jars/quasar.jar --content-path public`

A message similar to the following should be displayed:

```
Read config from /Users/user/Library/Application Support/quasar/quasar-config.json
Server started listening on port 20223
Press Enter to stop.
```

1.3.2 Starting SlamData from the Installer Package

Mac OS

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 Mac OS dock. As with other dock applications the SlamData icon may be right-clicked and the application terminated.

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>

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.

1.3.3 Starting SlamData Advanced from Command Line

SlamData Advanced Edition requires license key information before launching. This information is passed into the JVM at startup. An example of how this can be done is below. Note the use of escaping the quote characters with \ "

```
_JAVA_OPTIONS="-Xms1G -Xmx4G"

export SD_OPTS="\
-Dlicense_key=ABCDE-12345-ABCDE-12345-ABCDE \
-Dlicense_email=myemail@example.com \
-Dlicense_full_name=\"My Name\" \
-Dlicense_registered_to=\"Name Registered To\" \
-Dlicense_company=\"My Company Name\" \
-Dlicense_street=\"123 Anywhere Street, Suite A1\" \
-Dlicense_tel_number=3035551212 \
-Dlicense_fax_number=NA \
-Dlicense_city=Boulder \
-Dlicense_zip=80302 \
-Dlicense_country=US"

export _JAVA_OPTIONS="$_JAVA_OPTIONS $SD_OPTS"

java -jar quasar.jar --content-path public -L slamdata
```


2.2 Section 2 - 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.

2.2.1 2.1 Databases

Supported databases are listed in the following sections. As new target data sources are released they will be listed here.

2.1.1 MongoDB

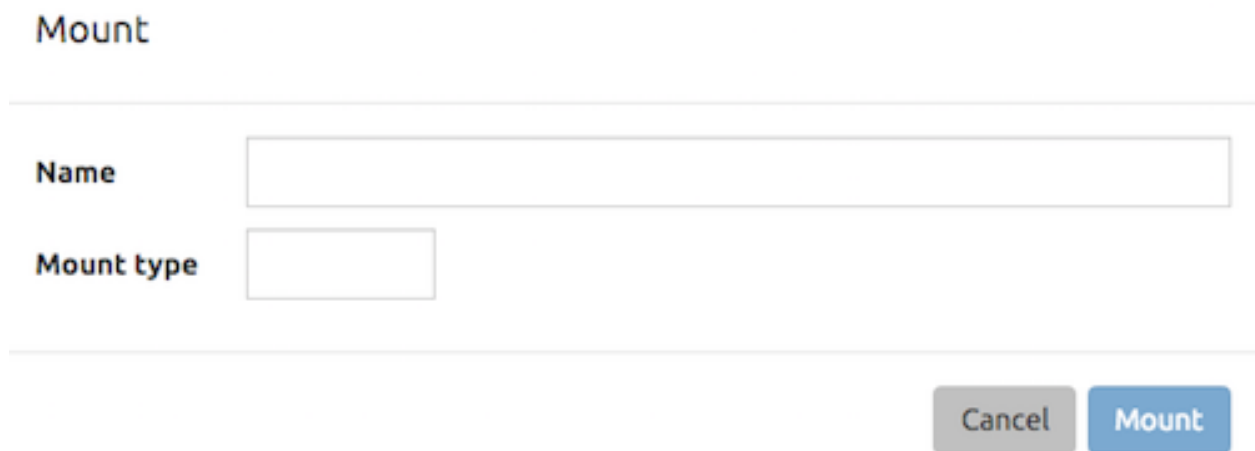
To connect to MongoDB click on the Mount  icon in the upper right.

A mount dialog will be presented:

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:



The image shows a 'Mount' dialog box. It has a title bar with the word 'Mount'. Below the title bar, there are two input fields. The first is labeled 'Name' and is a wide text box. The second is labeled 'Mount type' and is a smaller dropdown-style box. At the bottom right of the dialog, there are two buttons: a grey 'Cancel' button and a blue 'Mount' button.

Fig. 2.1: SlamData Mount Dialog

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.

2.2.2 2.2 Several Mounts

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

2.2.3 2.3 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.

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.

Mount

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

Fig. 2.2: SlamData MongoDB Dialog

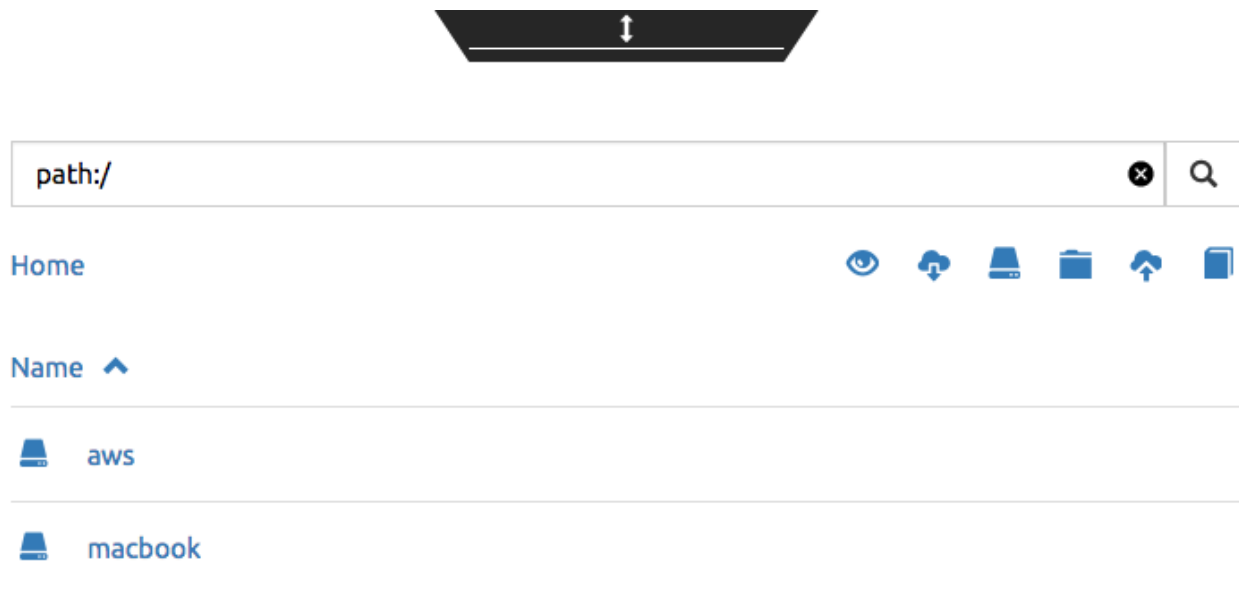


Fig. 2.3: SlamData Multiple Mounts

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

2.2.4 2.4 SQL² View

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

2.2.5 2.5 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
Mac OS	/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:

2.3 Section 3 - Configuring SlamData

2.3.1 3.1 Community Edition 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
Mac OS	<code>\$HOME/Library/Application Support/quasar/quasar-config.json</code>
Microsoft Windows	<code>%HOMEDIR%\AppDataLocal\quasar\quasar-config.json</code>
Linux (various vendors)	<code>\$HOME/.config/quasar/quasar-config.json</code>

An example configuration file for SlamData Community Edition might appear like this:

```
{
  "server": {
    "port": 8080,
    "ssl": {
      "enabled": true,
      "port": 9090,
      "cert": "<base64 encoded pkcs12 cert file>"
    }
  },
  "mountings": {
    "/aws/": {
      "mongodb": {
        "connectionUri": "mongodb://myUser:myPass@aws-box.example.com:27017/admin"
      }
    }
  }
}
```

Mount

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

Fig. 2.4: SlamData SSL Mounts

```
    },
    "/macbook/": {
      "mongodb": {
        "connectionUri": "mongodb://localhost:27017"
      }
    }
  },
}
```

2.3.2 3.2 Advanced Edition Configuration File

In addition to all Community Edition functionality, SlamData Advanced edition has additional configuration parameters to setup security, including the authentication, auditing and metastore directives.

Attention: SlamData Advanced Required

The configuration file listed below is applicable only to SlamData Advanced Edition and contains parameters and values that are valid only in that version.



An example configuration file for SlamData Advanced Edition might appear like this:

```
{
  "server": {
    "port": 8080,
    "ssl": {
      "enabled": true,
      "port": 9090,
      "cert": "<base64 encoded pkcs12 cert file>"
    }
  },
  "authentication": {
    "openid_providers": [
      {
        "issuer": "https://accounts.google.com",
        "client_id": "123...googleusercontent.com",
        "display_name": "Google",
      },
      {
        "issuer": "https://accounts.google.com",
        "client_id": "456...789.apps.googleusercontent.com",
        "display_name": "OAuth 2.0 Playground"
      }
    ]
  },
  {
    "display_name": "Our Company OP",
    "client_id": "123455976",
    "openid_configuration": {
      "issuer": "https://op.ourcompany.com",
      "authorization_endpoint": "https://op.ourcompany.com/authorize",
      "token_endpoint": "https://op.ourcompany.com/token",
      "userinfo_endpoint": "https://op.ourcompany.com/userinfo",
      "jwks": [
```

```

    {
      "kty": "RSA",
      "kid": "1234",
      "alg": "RS256",
      "use": "sig",
      "n": "2354098udw...29578351kj"
    },
    {
      "kty": "RSA",
      "kid": "5678",
      "alg": "RS256",
      "use": "sig",
      "n": "skljhdfigy...39587dlkjsd"
    }
  ]
},
"auditing": {
  "log_file": "/aws/logdb/slamdata-logs"
},
"metastore": {
  "database": "<h2 config | postgresql config>"
}
}

```

2.4 Section 4 - SlamData User Security

SlamData Advanced Edition provides user authorization, authentication, and auditing in addition to the features provided by the Community Edition.

Attention: SlamData Advanced Required
SlamData User Security is available only with SlamData Advanced Edition.



2.4.1 4.1 Security Overview

SlamData Advanced Edition controls user security through the use of tokens, permissions, actions and groups. Each is defined below.

	Description
Token	Allows specific actions regardless of implicitly-assigned or explicitly-assigned permissions
Permission	Contains actions, users and groups
Group	Contains users and other groups
Action	Distinct operation(s) that can be performed on a resource based on its type.
Type	<i>Structural, Content, or Mount</i>

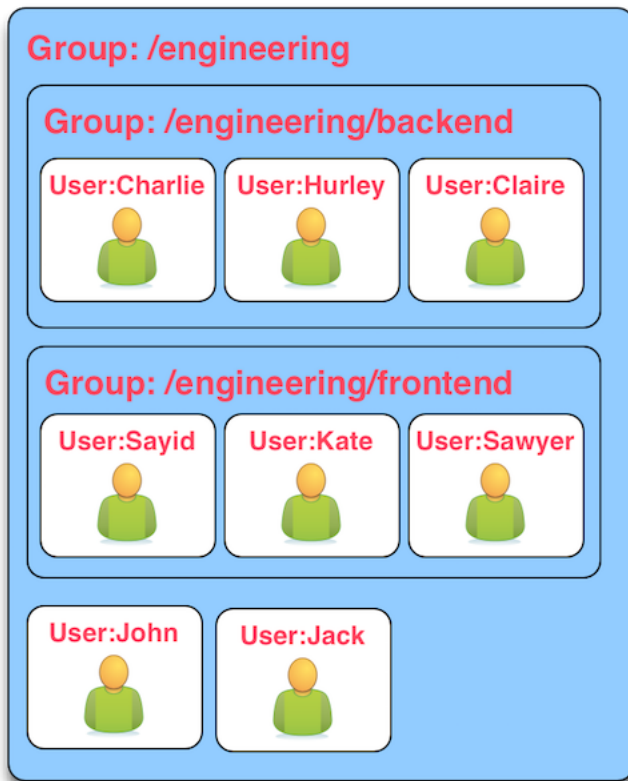
4.1.1 Users

Users are technically not objects stored in the SlamData metadata repository. Since SlamData relies on OAuth to authenticate users, it trusts the OpenID Provider to authenticate a user and state if the user is currently logged in.

Once logged in a user may perform actions depending on the configuration of groups and permissions. Users themselves are not created in the metadata store, but references to them are listed within Groups and Permissions. So while *technically* a user does not have an object in the metadata store, *logically* a user can be thought of as an object with privileges provided by Groups, Permissions, and possibly Tokens (when supplied with a request).

4.1.2 Groups

Groups contain users and other groups which are in the path (subgroups).

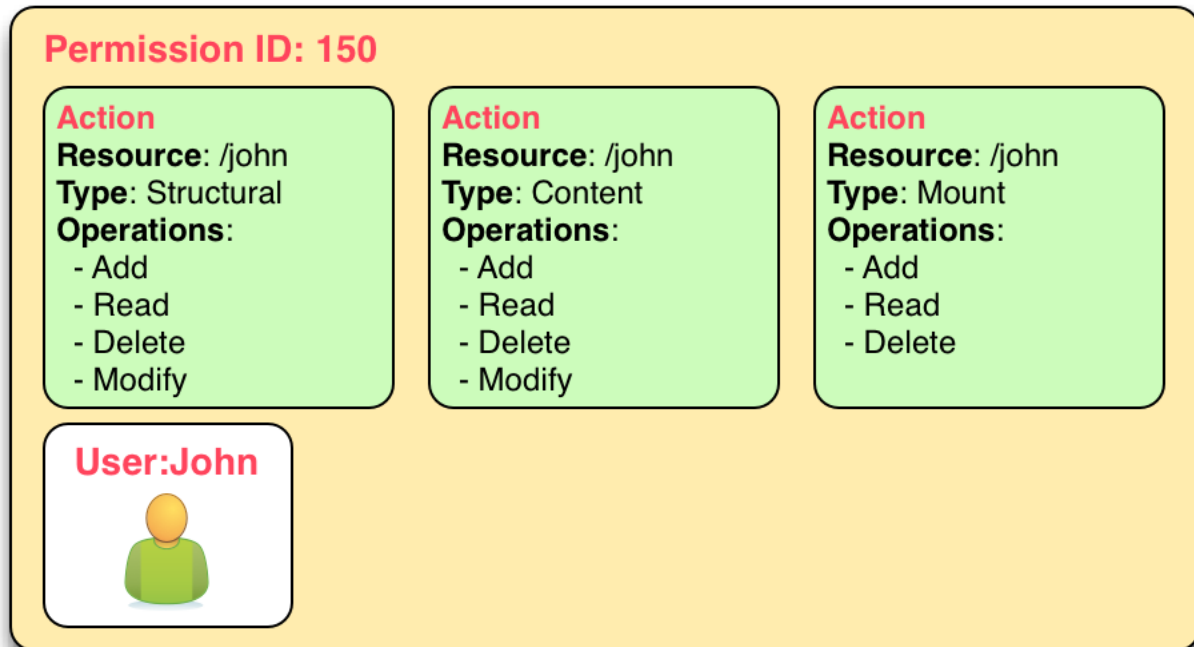


Since permissions may contain a group, and groups may contain users, then a user within a group inherits the permissions assigned to that group.

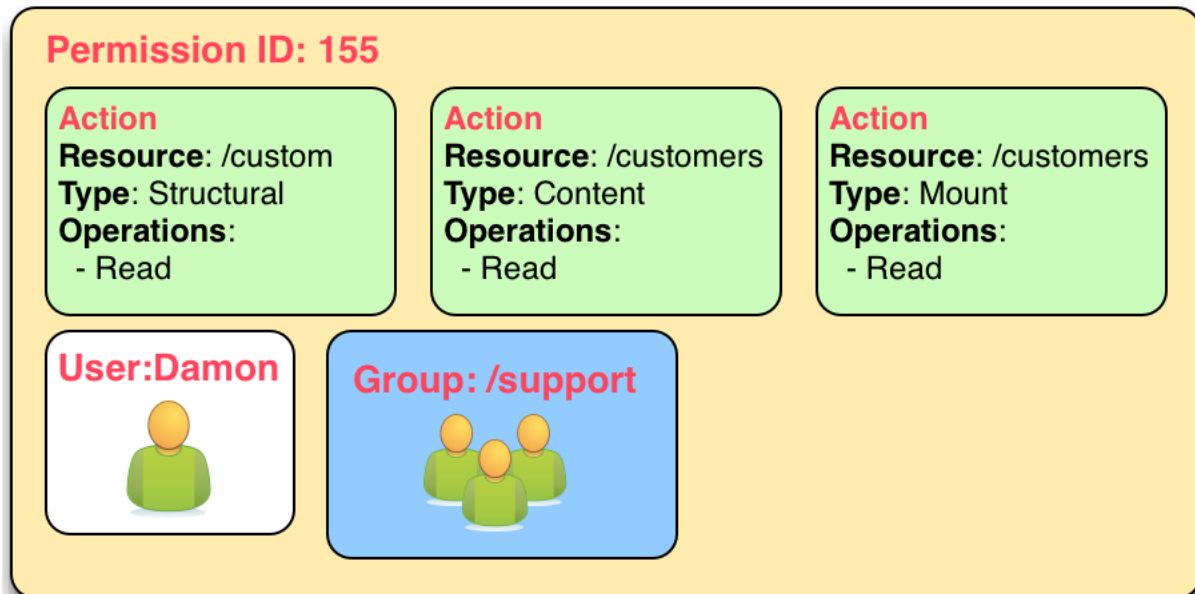
In the example above, both users John and Jack would inherit all of the permissions that contain the /engineering group. Those permissions would also apply to the subgroups for John and Jack.

The users Sayid, Kate, and Sawyer would inherit all of the permissions that contain the /engineering/frontend group, but would not inherit the permissions “above” from /engineering.

4.1.3 Permissions



In the example above, permission 150 contains several actions and the user `John`. This allows John to perform all actions listed, which includes any operation under the `/John` path.



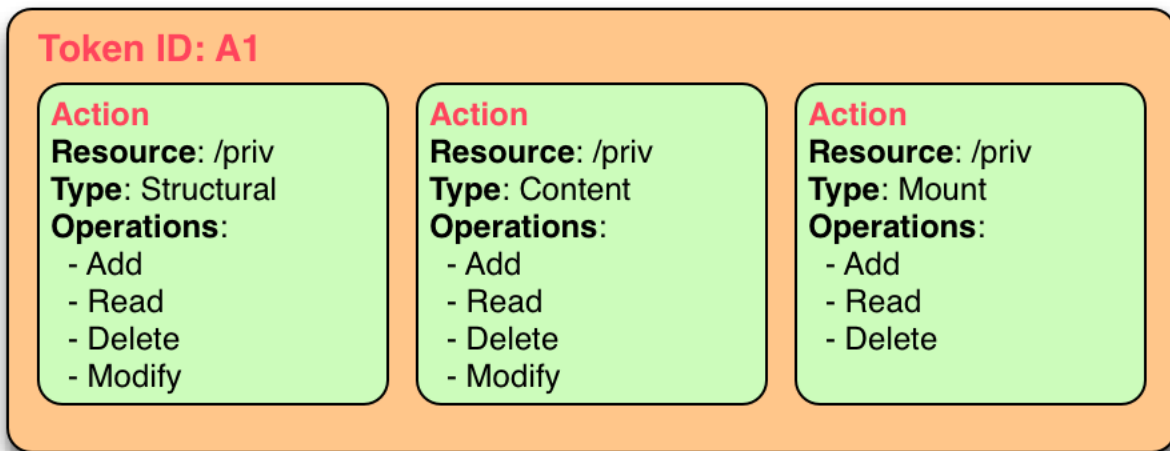
In the example above, both the user `Damon` and any other user within the `/support` group may read data from the `/customers` path, but may not create, modify or delete anything.

4.1.4 Tokens

If a token is passed in a request to SlamData, and the token is valid, the request will proceed based on the permissions assigned to that token.

In other words: if a user is trying to read from the `/data` mount, but does not have permissions through direct assignment or through group assignment, if the appropriate token with those permissions is passed into the same request, it will succeed.

See the following image



This this example, if a request included the token A1 then any operation performed within `/priv` would succeed, despite what permissions the user actually has.

2.4.2 4.2 Initializing the SlamData Metastore

SlamData Advanced Edition uses a metastore for user security. Before SlamData Advanced Edition can be started the metadata store must be initialized and initial administrator users defined. The administrator users will be added to a group having complete, unrestricted access to the system allowing them to provision additional groups and roles as needed.

To initialize the metadata store, run the `bootstrap` command and provide the name of the administrator group and e-mail addresses of initial members:

```
java -jar quasar.jar bootstrap --admin-group <name> --admin-users user1@example.com[,user2@example.com]
```

2.4.3 4.3 Authentication

SlamData Advanced Edition adds support for authenticated requests via the [OpenID Connect](#) protocol. A request to any SlamData or SlamData Advanced Edition API may be authenticated. If no credentials are included in a request, it is considered unauthenticated (or “anonymous”) and may fail if the system is not configured to allow anonymous access for the given request.

4.3.1 Making an Authenticated Request

To make an authenticated request, clients first need to ensure their OpenID Provider (OP) has been configured in SlamData Advanced Edition along with the “Client Identifier” (CID) issued to the client by the OP, this allows the SlamData

Advanced Edition administrator to specify which clients are permitted to access SlamData Advanced Edition. If an ID Token is received from a known provider but with an *unknown* CID, it will be rejected outright.

Next the client should obtain the list of known providers from the `/security/oidc/providers` endpoint (see details on this endpoint below) and authenticate the user against one of them, obtaining an **ID Token**. The ID Token **MUST** be requested using at least the openid and email scopes and their claims must be included in the ID Token.

Once in possession of a valid ID Token, the client includes it, verbatim, in the request to SlamData Advanced Edition via the `Authorization` header as a **bearer token** using the `Bearer` scheme.

If a request includes valid authentication and the identified subject is not permitted to perform the requested action per the authorization policy, a `403 Forbidden` response will be returned. If, however, a request which does not include any authentication information is denied due to the authorization policy a `401 Unauthorized` response will be returned to indicate that repeating the request with authentication may allow it to succeed.

4.3.1.1 Authentication and Performance

SlamData Advanced Edition requests require authentication before performing most actions. When an OIDC Provider (OP) is configured with minimal information, and the Discovery process is used, each action will make a discovery request as well. This can result in a noticeable degradation in performance.

To avoid this the OP can be configured with all attributes normally provided by the OIDC Discovery process within the configuration process itself. See the “Our Company OP” example in Section 3.2.

2.4.4 4.4 Authorization

SlamData Advanced adds support for authorization of service requests. Permissions for a request are derived from the union of permission tokens provided in the `X-Extra-Permissions` header and those configured for the authenticated user and anonymous user. Permissions are defined as an operation, its type, and a filesystem resource path. A permission token grants a set of permissions.

The available operations and types are as follows:

Type: Content, Structural, Mount

Operation: Add, Read, Delete, Modify

	Content	Structural	Mount
Add	append to file	create resource	create mount
Read	read file contents	list directory	retrieve mount info
Delete	delete file contents	delete resource	remove mount
Modify	modify file contents	rename or move resource	N/A

A permission on a parent resource is sufficient to authorize an action on a resource granted the nature and type of the operation are the same.

A `403 Forbidden` is returned by the server when a request does not have sufficient permissions to perform the associated actions.

The `X-Extra-Permissions` header is formatted as follows:

```
X-Extra-Permissions: [token1], [token2]
```

2.4.5 4.5 Auditing

Attention: File system definition

The SlamData product sometimes refers to virtual database paths as file systems and tables or collections as file names. In the Auditing section below the **log file** path should be a path to the collection or table you wish to save to. This does not equate to an operating system file name or directory path.

When a log file is specified in the configuration file, all filesystem operations will be logged to that file. SlamData Advanced Edition logs the operations as data in the filesystem where the path is located. This means that it is then possible to use SlamData Advanced Edition itself to analyze the log data.

2.5 Section 5 - Security APIs

SlamData Advanced Edition provides additional APIs to control user access.

Actions and permissions are central concepts to the security api. An action is any operation a subject can perform on a given resource in the system. A permission represents the capability of a subject (group, user, token) in the system to perform a given action. All permissions have a lineage which represents by which authority a permission was granted to a subject. Any subject in the system has the authority to grant a new permission which is a subset of one of their own permissions. This new permission is said to have been derived from the relevant permission(s) of the grantor and that/those relevant permission(s) are said to be the parent(s) of that permission.

Permissions can be revoked. If a permission is revoked, that permission as well as all permissions derived from it become invalid and can no longer be used to perform operations in the system. It is possible however for one of those derived permissions to have been derived from more than one permission, i.e. another permission than the one being revoked. In such a case, that permission will not become invalid. It will only become invalid once all its parents have been revoked. The permission being revoked however, will be revoked, no matter how many sources of authority it possess.

Actions and permissions are found throughout the following api endpoints and are represented as follows in JSON:

Action:

```
{
  "operation": "ADD|READ|MODIFY|DELETE",
  "resource": "<filesystem_path>|<group_path>",
  "accessType": "Structural|Content|Mount",
}
```

Permission:

```
{
  "id": "<permission_id>",
  "action": {
    "operation": "ADD|READ|MODIFY|DELETE",
    "resource": "<filesystem_path>|<group_path>",
    "accessType": "Structural|Content|Mount",
  },
  "grantedTo": "<user_id>|<group_path>|<token_id>",
  "grantedBy": ["<user_id>", "<group_path>", "<token_id>", "..."]
}
```

- **<filesystem_path>** is a path in the quasar virtual filesystem such as `data:/foo/bar` for a file and `data:/foo/bar/` for a directory

- **<group_path>** is a path uniquely identifying a group and its location in the group hierarchy such as `group:/engineering/backend`
- **<grantedBy>** The sources of authority by which this permission was granted. In reality, the sources are the parent permissions; here we are simply surfacing the subjects which possess the permissions by which this permission was granted.
- **<user_id>** is an email prefixed with the “user” string such as `user:bob@example.com`
- **<token_id>** is a string identifier prefixed by the “token” string such as `token:786549382`

Note: The Mount value of `accessType` is only valid if the resource is a filesystem path, it is not a valid value for a group resource.

In the following API endpoints descriptions, “your permissions” refers to the set of permissions associated with the HTTP request. In the case of an authenticated user this means all permissions directly associated with that user as well as all groups that user is a explicitly or implicitly a part of. Additionally, any permission associated with tokens present in the request headers are added to the permissions associated with the request.

Whenever no return body is specified, a response with a 2XX status can be expected along with an empty body.

In any of the following endpoints, if the request does not “carry” sufficient permissions to satisfy the requirements of the particular endpoint, the server will return a 403 `Forbidden` with an explanation of which permissions were missing in order to perform the operation. Certain endpoints will always succeed, but the results will be filtered based on what the user is permitted to see. In such case the endpoint will document how to determine what a user can and cannot see.

2.5.1 5.1 - Group Endpoint

GET /security/group/<path>

- Retrieves information about this group. The result of the query will depend on your permissions according to the following rules:
- If you have `READ` content group permission on this group, then your view is unrestricted. (all fields are present)
- If you have `READ` structural group permission on this group, then you can know of the existence of this group and all of its sub-groups. (`subGroups` field is present in response)
- If you have `ANY OTHER` group permission on this group, you can know of the existence of this group, but nothing else. (response is empty)
- If you have `READ` content group permission on one of this group’s sub-groups, then you can see that subgroup as well as any of its own subgroups. You can see all members of that group and sub-groups. (`allMembers` and `subGroups` fields are present in response)
- If you have `READ` structural group permission on one of this group’s sub-groups, then you can see that subgroup as well as any of its own sub-groups. You cannot see any of the members of those groups however. (`subGroups` field is present in response)
- If you have `ANY OTHER` group permission on one of this group’s sub-groups, then you can see that subgroup.

These rules are cumulative, so if more than one rule applies, you will see the combined result. If none of the rules apply, the query will result in a 403 `Forbidden`. If certain fields do not apply to your view of this group, they will be omitted in order to clearly convey that they are not necessarily empty, you just don’t have permission to see anything related to that field.

- `<path>` is the path of the group in the group hierarchy

Note: All users are members of the root group (“/”) regardless of whether they are a member of any other group. Permissions associated with the root group represent the capabilities of any agent in the system.

Response:

The response body will vary depending on the rules outlined above. If you have some kind of relevant permission as outlined above and the group does not exist, the response will be a 404 Not Found.

- `members` All users explicitly a member of this group
- `allMembers` All users explicitly and implicitly a member of this group.

Implicit members of a group refer to the users that are explicit members of any of the sub-groups of this group

- `subGroups` All descendants of this group in the group hierarchy.

Example:

Given the following groups exist in the system:

`/corporate -> “Alice” /corporate/engineering -> “Bob” /corporate/engineering/software -> /corporate/engineering/software/scala -> “Marcy” /corporate/engineering/hardware -> (“Tom”, “Beth”)`

`GET /security/group/corporate/engineering` will return:

```
{
  "members": [ "bob@example.com" ],
  "allMembers": [ "bob@example.com",
    "marcy@example.com",
    "tom@example.com",
    "beth@example.com"
  ],
  "subGroups": [ "/corporate/engineering/software",
    "/corporate/engineering/software/scala",
    "/corporate/engineering/hardware"
  ]
}
```

POST /security/group/<path>

Creates a new empty group. If any of the parent groups do not exist yet, they will be created.

Requires ADD or MODIFY structural group permission.

Response:

If you have adequate permissions and the group already exists, will return a 400 Bad Request.

PATCH /security/group/<path>

Add or remove users of a group.

Requires ADD content group permission to add users. Requires DELETE content group permission to remove users. Alternatively, the MODIFY content group permission is sufficient to add and/or remove users.

Request:

```
{
  "addUsers": [ "<user_email>" ],
  "removeUsers": [ "<user_email>" ]
}
```

Response:

If you have adequate permissions, but the group does not exist, the response will be a 404 Not Found. If a user found in the `removeUsers` field was not actually a member of the group, the request will succeed nevertheless and simply ignore that user.

DELETE /security/group/<path>

Delete this group and all of its sub-groups. All permissions associated with this group and subgroups as well as shared by this group and subgroups will immediately become invalid.

Requires DELETE or MODIFY structural group permission.

Response:

If you have adequate permissions, but the group does not exist, the response will be a 404 Not Found

2.5.2 5.2 - Authority Endpoint

GET /security/authority

Returns all permissions granted to you.

Response:

```
[<permission>]
```

2.5.3 5.3 - Permission Endpoint

GET /security/permission[?transitive]

Returns all permissions granted by you. If the `transitive` query param is supplied, will also return all permissions which were derived from your own.

We may add query parameters in the future in order to filter the result set.

Response:

```
[<permission>]
```

GET /security/permission/<permission_id>

Retrieve a permission by its unique identifier. You may only retrieve information about permissions shared with you or by you.

If the permission does not exist or you do not have adequate permission to see it, the response will be a 404 Not Found.

Response:

```
<permission>
```

GET /security/permission/<permission_id>/children[?transitive]

Retrieve all permissions that were directly derived from this permission. If the `transitive` query param is supplied, will also include permissions which were indirectly derived. You may only retrieve information about permissions shared with you or by you.

If the permission does not exist or you do not have adequate permission to see it, the response will be a 404 Not Found.

Response:

```
[<permission>]
```

POST /security/permission

Grant new permissions to a given set of users and/or groups.

Request:

```
{
  "subjects" : [<user_id>, <group_id>, "..."],
  "actions": []
}
```

- **user_id** is a email prefixed with the “user” string such as `user:bob@example.com` representing the users to whom you wish to grant permissions. Users do not need to exist in the system at the time the permission is granted. When a user first logs into the system, they will be able to perform any action associated with permissions granted to their email.
- **group_id** a path prefixed with the “group” string such as `group:/engineering/backend`. Groups DO need to exist in the system prior to granting them a permission. Providing a group path that points to a group that does not yet exist in the system will result in a 400 Bad Request and no new permissions will have been granted to users or groups.
- **actions** The actions that the new permissions will allow the subjects to perform. All actions must be the same or a subset of actions found in your permissions. If that is not the case a 400 Bad Request with an appropriate message will be returned and no new permissions will have been granted to users or groups.

Although all fields accept arrays, a permission is only ever granted to ONE subject to perform ONE action. Thus, many permissions will be created and returned by this endpoint.

Response:

```
[<permission>]
```

DELETE /security/permission/

Revoke a permission. In order to revoke a permission, you must have a permission which is a source of authority for the permission you wish to revoke.

Refer to top-level api description for explanation on the process of revoking.

Note: Revoking a permission does not guarantee that the subject associated with that permission no longer has the capability to perform that action as another subject in the system may have also granted a permission with the capability to perform the same action. Unless you possess the root authority (e.g. if you are a member of the “admin” group created when the metastore was initialized), it is impossible for you to know for sure whether or not a subject still has the ability to perform the action.

If the permission does not exist or you do not have adequate permission to see it, the response will be a 404 Not Found. If you attempt to revoke one of your own permissions, the response will be a 400 Bad Request.

2.5.4 5.4 - Token Endpoint

Here is the json representation of a token:

```
{
  "id": <token_id>,
  "secret": <token_hash>,
  "name": <name>,
  "grantedBy": [<token_id>, <user_id>, <group_id>, "..."],
  "actions": [{
```



```

    "operation": "ADD|READ|MODIFY|DELETE",
    "resource": "<filesystem_path>|<group_path>",
    "accessType": "Structural|Content|Mount",
  }]
}

```

- **secret** is a cryptographically secure string whose possession allows one to perform the action associated with the token.
- **name** an optional field that may or may not have been provided upon creation of the token
- is a string identifier prefixed by the “token:” string
- an email address prefixed with the “user:” string
- a group path prefixed with the “group:” string

Note: Once again, the `Mount` value for `accessType` is only valid for a filesystem path.

GET /security/token

List tokens that you have created. Does not list tokens that were created by others based on your authority.

The json representation of the tokens does not contain the `secret` field for this endpoint in order to reduce the chance of the secret leaking. The secret can be retrieved by using the `id` endpoint.

Response:

```
[<token>]
```

GET /security/token/<id>

Retrieve token for a given id.

You may only retrieve information about a token that you created. If the token does not exist or was not created by you, the response will be a 404 Not Found.

Response:

```
<token>
```

POST /security/token

Create a new token granting the capability to perform the given actions. All actions must be a subset of your own capabilities. If the later condition is not satisfied, a 400 Bad Request will be returned.

Request:

```

{
  "name": "",
  "actions": []
}

```

- **name** is an optional field

Response:

```
<token>
```

DELETE /security/token/<id>

Delete a token. In order to delete a token, you must have a permission which is a source of authority of the token. If the token does not exist or was not created by you, a 404 Not Found will be returned.

GET /security/oidc/providers

This endpoint allows clients to obtain the list of configured OpenID Providers (OPs). Responses will be a JSON array of configurations similar to the following.

Response:

```
[
  {
    "display_name": "Google",
    "client_id": "sdf9.....dflkj",
    "openid_configuration": {
      "issuer": "https://accounts.google.com",
      "authorization_endpoint": "https://accounts.google.com/o/oauth2/v2/auth",
      "token_endpoint": "https://www.googleapis.com/oauth2/v4/token",
      "userinfo_endpoint": "https://www.googleapis.com/oauth2/v3/userinfo",
      "jwks": [
        {
          "kty": "RSA",
          "alg": "RS256",
          "use": "sig",
          "kid": "1195d.....6abd",
          "n": "qy5D0.....tJRJY02Qt0UKzJ2OquiPw",
          "e": "AQAB"
        },
        {
          "kty": "RSA",
          "alg": "RS256",
          "use": "sig",
          "kid": "b0a61.....9ba8575712",
          "n": "rvhjUe0.....n2IRNM8S8iJ36w",
          "e": "AQAB"
        }
      ]
    }
  },
  {
    "display_name": "Our Company OP",
    "client_id": "123455976",
    "openid_configuration": {
      "issuer": "https://op.ourcompany.com",
      "authorization_endpoint": "https://op.ourcompany.com/authorize",
      "token_endpoint": "https://op.ourcompany.com/token",
      "userinfo_endpoint": "https://op.ourcompany.com/userinfo",
      "jwks": [
        {
          "kty": "RSA",
          "kid": "1234",
          "alg": "RS256",
          "use": "sig",
          "n": "2354098udw...29578351kj"
        },
        {
          "kty": "RSA",
          "kid": "5678",
          "alg": "RS256",
          "use": "sig",
          "n": "skljhdfiugy...39587dlkjsd"
        }
      ]
    }
  }
]
```

```
]
  }
}
]
```



Developers Guide

This Developer's Guide will assist the developer who is unfamiliar with SlamData to install, configure, customize and embed a complete solution from start to finish.

For information on how to use SlamData from a user perspective see the SlamData Administration Guide

For information on how to use SlamData from a user perspective see the SlamData Users Guide (not implemented yet)

Attention: SlamData Advanced Features

Throughout this guide there are references to functionality available only in SlamData Advanced Edition. Sections

that apply only to SlamData Advanced Edition will be called out with the Murray (MRA) icon.



3.1 Section 1 - Installing and Running SlamData

3.1.1 1.1 Purpose

The purpose of this Developer's Guide is to walk a software developer through SlamData from installation through completed project. The goal is to provide a step-by-step process that a developer can follow, including sample data, that is repeatable with other data sets and environments.

3.1.2 1.2 Introduction

SlamData is both an Open Source Software project and a commercially available Visual Analytics platform for multi-dimensional data (including two-dimensional RDBMS data). SlamData provides the ability to query all of your data, in any form, in any location with a single solution. This is achieved with some of the following features of SlamData:

- Patented multidimensional relational technology, allowing SlamData to communicate with any data source in any data format. This includes not only legacy two-dimensional data such as RDBMS in rows and columns, but also deeply nested, semistructured data such as JSON and XML.
- Ability to understand schema dynamically, resulting in absolutely no need to map field types from one technology to another. This also allows SlamData to use both field values **and** the schema as data. This is not possible with other NoSQL -> relational solutions.
- A fully generalized database backend technology, providing a reliable and ANSI compatible superset of SQL called SQL² that runs on top of any supported data source. There is no need to learn yet another proprietary database query language.

- Fully embeddable solution that merges seamlessly with your own applications providing a consistent look and feel while providing significant and immediate value out of the box.
- Easy to use search capabilities for non-technical users. Search for a key word, value or any other data type without knowing where it is or in which format.
- Visually appealing charts (eCharts from Baidu) that can be customized and natively understand nested data.
- Ability to secure data in a multi-tenant environment through OpenID Connect and OAuth 2.0.

3.1.3 1.3 Assumptions

This guide was written with the following assumptions in mind. The reader is a developer that:

- Has a basic to moderate understanding of SQL
- Has a basic to moderate understanding of JSON
- Has a basic to moderate understanding of HTML web applications
- Can perform basic navigation of the MongoDB database via the **mongo** shell
- Has appropriate permissions to install relevant software

3.1.4 1.4 Requirements

For SlamData to run in an optimal environment see the Minimum System Requirements section.

Attention: Windows Developers

This Developer's Guide includes example code in several sections in addition to shell scripts or command line utilities. While this guide can be followed by most Mac OS and Linux developers, Microsoft Windows developers will have to implement similar functionality through other means such as DOS shell scripts.

3.1.5 1.5 Installation

Instructions for installing SlamData Community Edition can be found [here](#)

SlamData Advanced Edition is delivered with an automated installer.

3.1.6 1.6 Starting SlamData

Instructions for starting SlamData can be found [here](#).

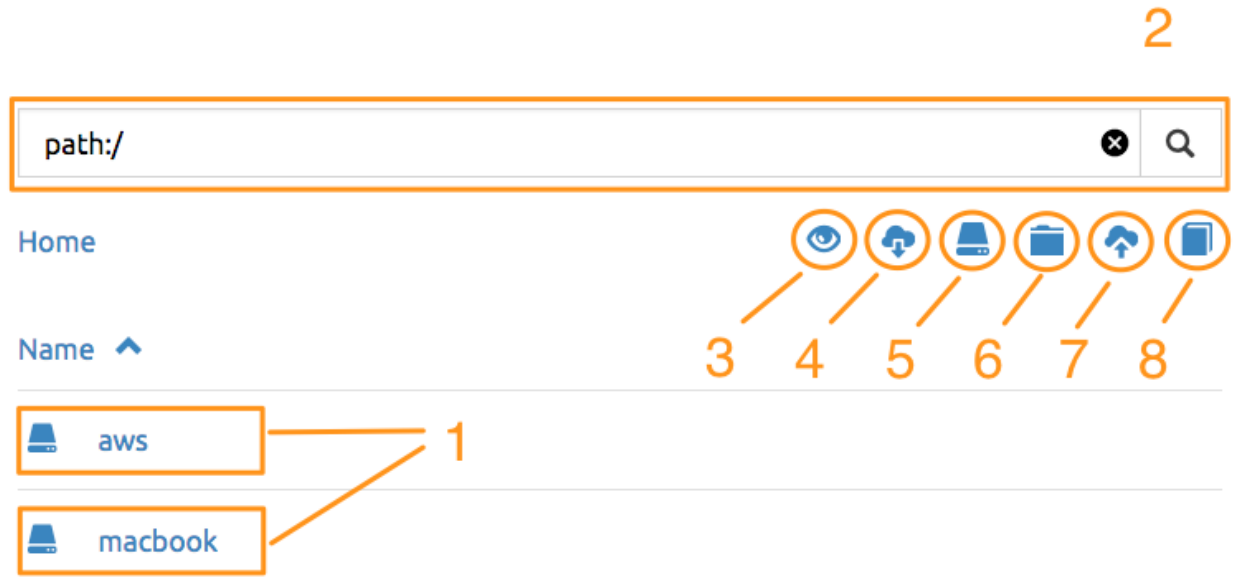
Once SlamData Community Edition or SlamData Advanced Edition is running then continue to Section 2.

3.2 Section 2 - Exploring Data

By the end of this Developers Guide the reader will have a fully working SlamData environment that is securely embedded with user authentication, interactive forms and dynamic charts. To start, however, the basics of the user interface will need to be covered. The guide will then move on to more complex topics focused on importing data, exploring that data and searching it with keywords and eventually using SlamData's SQL² dialect to perform SQL queries on the data.

3.2.1 2.1 Interface Navigation

The image below shows the Home screen after starting SlamData Community Edition. Note the numbers and their descriptions following the image.



Number	Description
1	Server or Mount names that have been configured.
2	The current path you are viewing. In this example it is the Home path (/).
3	The eye icon toggles visibility of the trash can icon.
4	Download all data starting from this path.
5	Mount a new database server.
6	Create a new folder in the datasource virtual file system.
7	Upload a data file.
8	Create a new workspace.

3.2.2 2.2 Workspaces, Decks and Cards

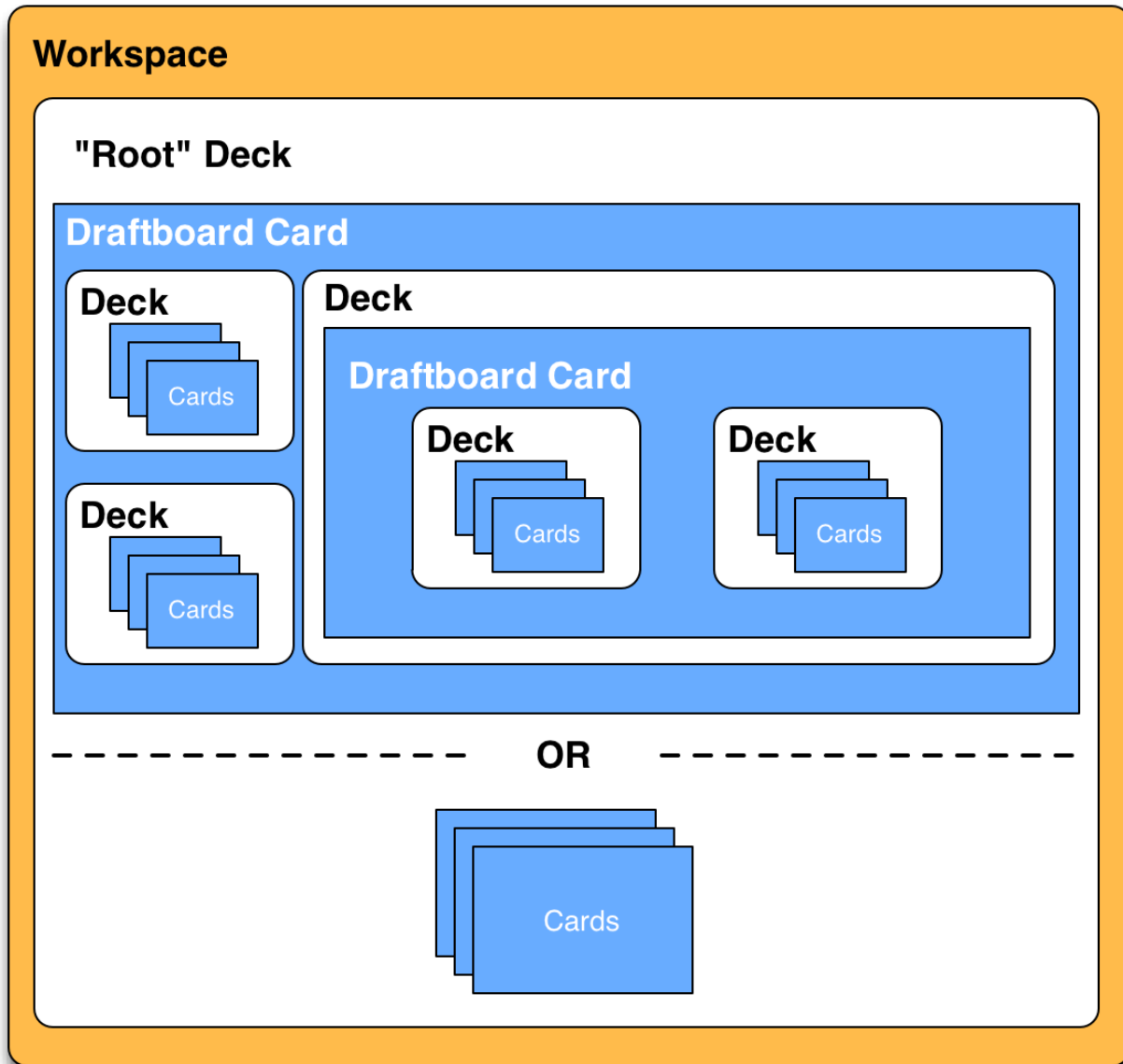
Before we start looking at our data we need to discuss how to interact with it. This is done through the use of a **Workspace**. A Workspace is the primary method that users interact with data within SlamData. A Workspace in turn is comprised of cards, and decks of cards.

- **Root Deck** - Each Workspace must have a Root Deck in which all other unit types are stored. A Root Deck is always present in a Workspace but never visible.
- **Deck** - Each deck contains at least one or more cards that each perform a specific action and build upon each other. Decks can be mirrored which allows easy creation of a new target deck that starts with the same functionality as the origin deck. Changes in each deck, up to the point where they were mirrored, will impact each other.
- **Draftboard Card** - A special card type that creates a visual area to arrange multiple decks.
- **Card** - A unit that performs a distinct action. Examples include:
 - Query Card
 - Show Table Card

- Show Cart Card
- and more...

Unit Type	May Contain:
Root Deck	Either a single Draftboard Card or multiple normal cards.
Deck	One or more cards, including one Draftboard Card
Draftboard Card	One or more decks.
Card	N/A

A visual example of the allowable nesting follows:



Don't worry! You won't need to know any of this until section 3, and by then we will take you through it step by step.

3.2.3 2.3 Creating a New Mount

In this guide the MongoDB database will be used in the examples; as such, the reader should download and run the latest stable version of MongoDB.

Default MongoDB installations run on port **27017** and have no user authentication enabled. This guide assumes this configuration in the following instructions.

Click the New Mount Icon. 

A dialog will appear requesting the name and Mount type.

Mount

Name	<input type="text"/>
Mount type	<input type="text"/>

Enter the values below and the dialog will expand.

Parameter	Value
Name	devguide
Mount Type	MongoDB

In the expanded dialog enter the values below and click **Mount**. If a parameter in the table below has no value, leave that field empty in the interface.

Parameter	Value
Host	localhost
Port	27017
Username	
Password	
Database	
Other Settings	

Mount

Name	<input type="text" value="devguide"/>		
Mount type	<input type="text" value="MongoDB"/>		
Server(s)			
Host	<input type="text" value="localhost"/>	Port	<input type="text" value="27017"/>
	<input type="text"/>		<input type="text"/>
Authentication			
Username	<input type="text"/>		
Password	<input type="text"/>		
Database	<input type="text"/>		
Settings			
Name	Value		
<input type="text"/>	<input type="text"/>		

3.2.4 2.4 Creating a Database


- Click on the newly created server named **devguide**. The interface now shows the databases that reside within MongoDB.

A new database will need to be created to follow along with the guide.

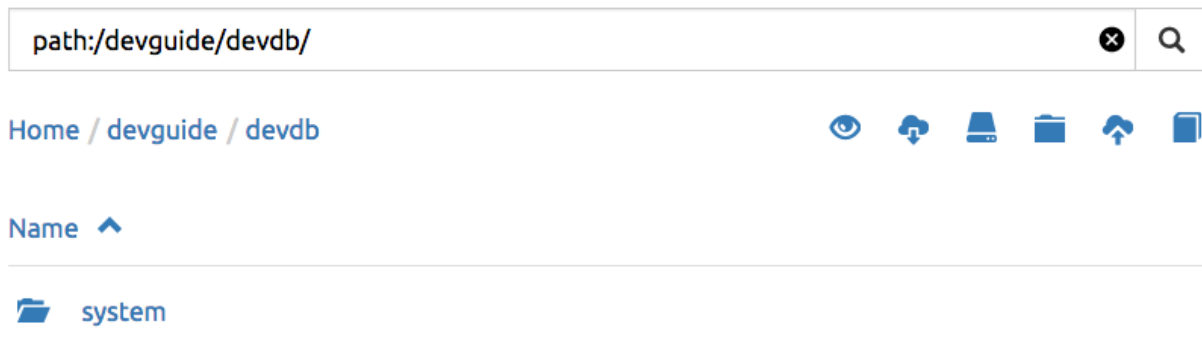
- Click on the Create Folder icon. 

A new folder will appear titled **Untitled Folder**.

- Hover the mouse over the new **Untitled Folder** folder.

- Click the **Move/Rename** icon that appears to the right. 
- Change the name from **Untitled Folder** to **devdb** and click **Rename**.
- Click on the newly renamed **devdb** folder.

The interface should now look like this:




So far in this guide you've installed SlamData, mounted a database and created and renamed a folder. Good progress. Let's get some data into the database now and start exploring.

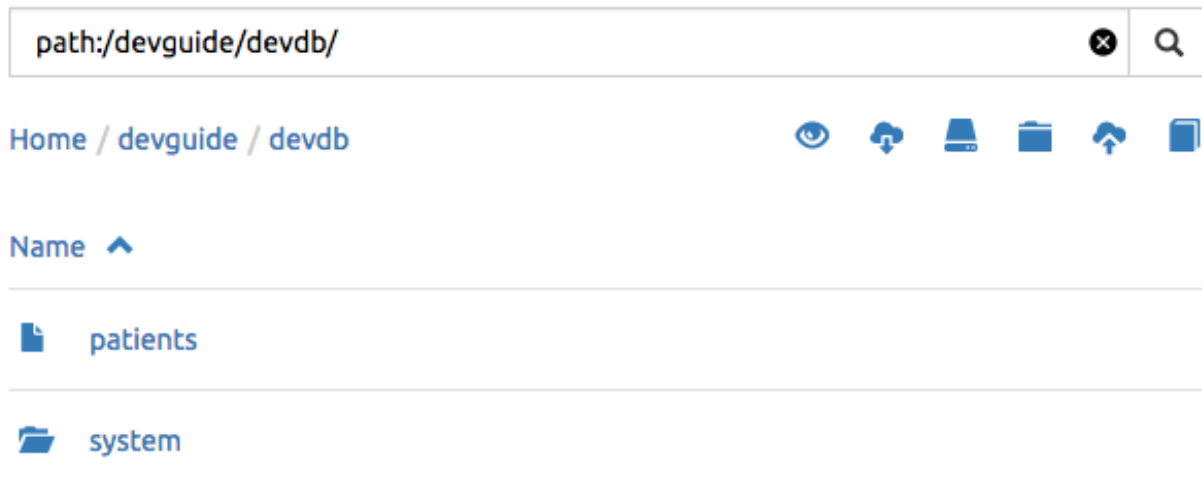
3.2.5 2.5 Importing Example Data

This guide uses a data set of fictitious patient information that was randomly generated. The reader can use any data set they wish, but the examples in the remaining sections will assume the patients data set is being used.

You can download a data set with 10,000 documents by following these instructions:

- Right click [this link](#) and save the file as **patients**. This is a 9 MB JSON file.
- If your operating system named the file something other than **patients** you can either rename it or you can rename it inside of SlamData once it has been uploaded.

- Ensure the SlamData UI is in the devdb database, and click the Upload icon. 
- In the file dialog find the patients file and submit it.
- After successful upload a new collection should appear in the UI like the following:



As you can see it is easy to import JSON and CSV data into SlamData quickly. The underlying database in this case is MongoDB.

2.5.1 Indexing Your Database

Attention: Indexing Your Database

While this step is not exactly necessary, any database without indexes is going to perform slowly. In SlamData this can be seen as a delay in displaying results. If you choose to skip this step be prepared to wait several seconds while MongoDB performs your searches.

The following commands are specific to MongoDB and must be executed from the `mongo` shell console.

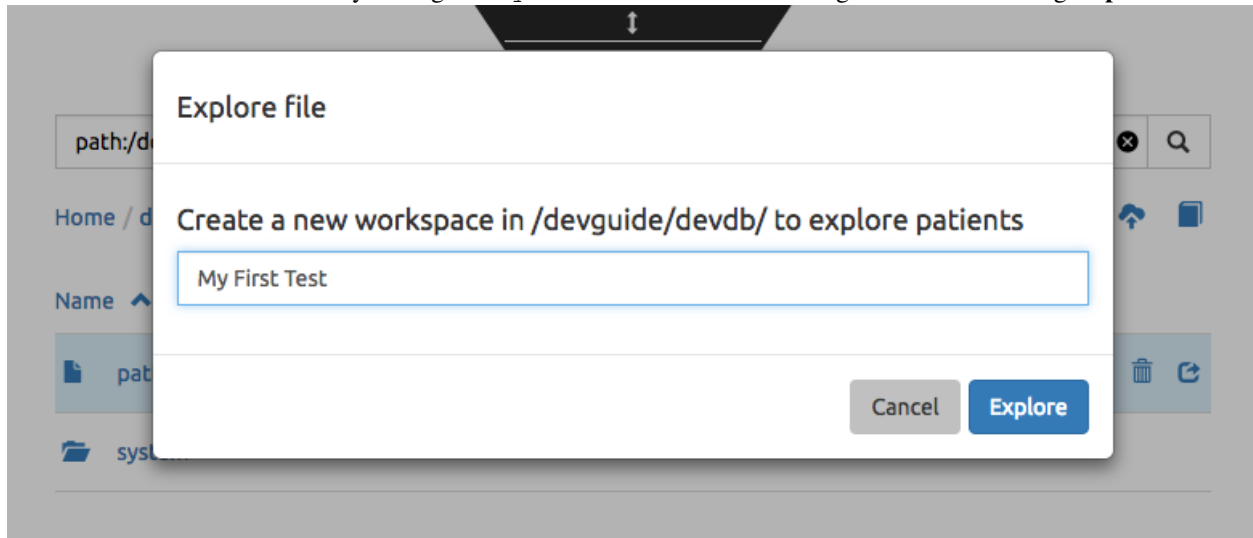
```
use devdb
db.patients.createIndex({first_name:1})
db.patients.createIndex({middle_name:1})
db.patients.createIndex({last_name:1})
db.patients.createIndex({city:1})
db.patients.createIndex({county:1})
db.patients.createIndex({state:1})
db.patients.createIndex({zip_code:1})
db.patients.createIndex({street_address:1})
db.patients.createIndex({height:1})
db.patients.createIndex({weight:1})
db.patients.createIndex({age:1})
db.patients.createIndex({gender:1})
db.patients.createIndex({last_visit:1})
db.patients.createIndex({previous_visits:1})
db.patients.createIndex({previous_addresses:1})
db.patients.createIndex({codes:1})
db.patients.createIndex({"codes.code":1})
db.patients.createIndex({"codes.desc":1})
```

Congratulations! There is now a usable dataset in your database that is full of complex, nested data that you can explore. Let's start!

3.2.6 2.6 Exploring Data

To simply look around and explore data, you can click on any file (collection) that you see. Start by clicking on the **patients** file.

You'll be prompted to provide a name for a new Workspace. A Workspace is how users interact with the actual data within the database. Let's start by calling this `My First Test` or something similar and clicking **Explore**



Once you click Explore, the following screen should appear:

1

2

3

4

5

age	city	codes	county	first_name	gender	height	last_name	last_visit	loc	middle_name	previous_addr:	
		code	desc								city	
38	STERLING	S21.212D	Unspecified nondisplaced fracture of second cervical vertebra, subsequent encounter for fracture with delayed healing	COMANCHE	Dona	female	63	Montoya	2015-05-16T17:10:04Z	-98.167941 34.749594	Judith	ROANOKE
												DES MOINES
												BURLINGHAM
36	OSCAR			POINTE COUPEE	Mauro	male	60	Horn	2016-04-09T08:47:21Z	-91.456649 30.616102	Andrew	INDEPENDENCE
		V92.12xS	Unspecified hereditary corneal dystrophies									WAIKOLOA
55	RENO	T48.3X1D	Displacement of artificial skin graft and decellularized allodermis,	WASHOE	Syble	female	74	Mccormick	2013-10-08T15:36:06Z	-119.604367 39.906211	Georgia	WARM SPRINGS

Page 11 of 1000

Per page: 10

Number	Description
1	Zoom icon takes user back out of the Workspace and back to the database screen.
2	Flip the card over for more options.
3	Card grips. Slide these left or right to see the previous card or create a new one.
4	Browse controls for the current card.
5	Your position within the deck. Gray circle indicates your place, white circles are available to view.

Feel free to click around on the browse arrows at the bottom to flip through the pages of data. It's easy to get an idea

of the schema of this data set by looking at the top row. In this case you can also see that the **codes** field is not actually a simple field but an array of other documents! Each of those documents in turn have a **code** and **desc** field.

Hint: Workspace Usage

You may not know it, but you actually just created a Workspace and a Root Deck, which contains an **Open Card** and an **Explore Card**! SlamData did this automatically to save you time.

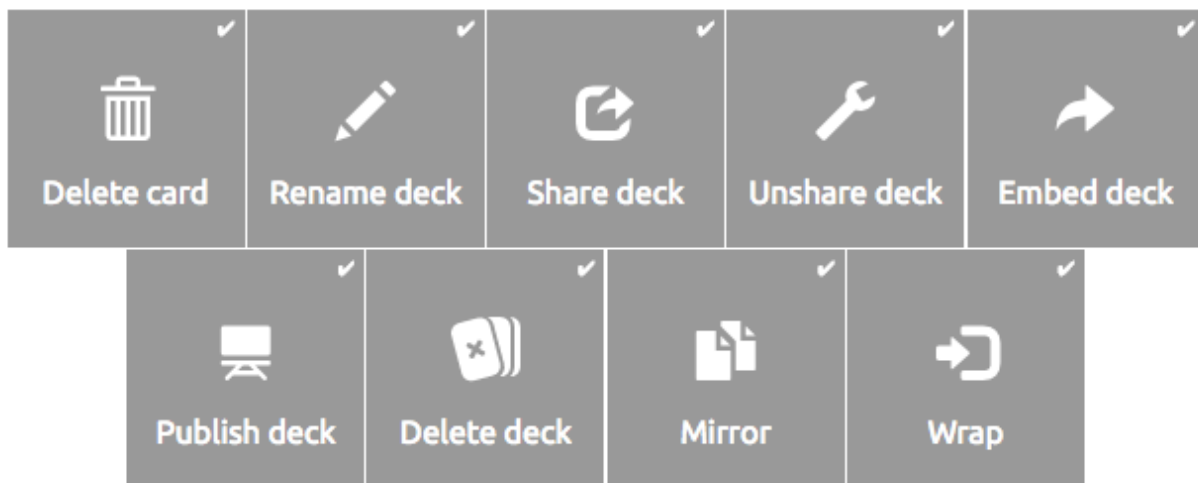
Any changes made within a Workspace are saved automatically. At any time the user may zoom out of the current window.

3.2.7 2.7 Searching Data

Viewing and browsing the data is helpful but data becomes less useful if you can't find what you're looking for. SlamData has two very powerful ways of finding the data you need. One is the **Search Card** and the other is the **Query Card**. We'll start with the **Search Card**.

- Click the **Flip Card** Icon (#2 in previous image)

You'll see the following options on the back of that card:

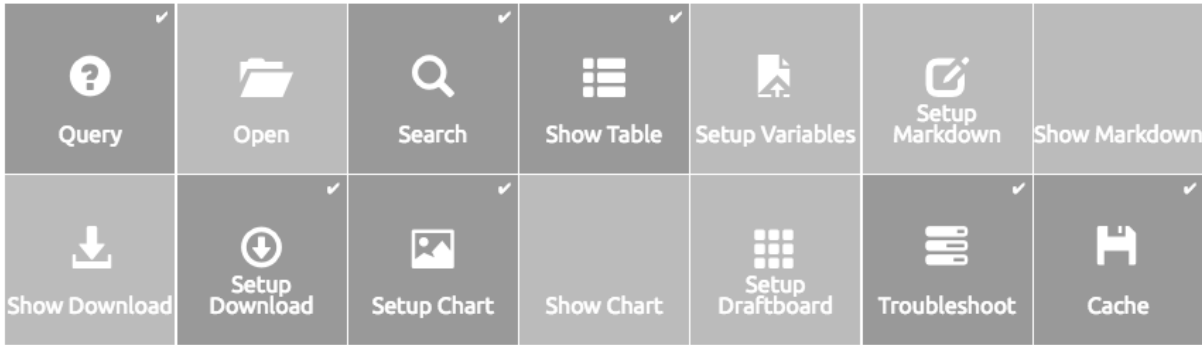


- Click on **Delete Card**

The UI will now show the only remaining card in the deck which is the **Open Card**. This card allows you to select which collection you wish to operate on with subsequent cards. Let's leave this card in place.

- Click and drag the right-hand grip and slide it to the left.

You'll be presented with the following card types to choose from:



Notice how the cards are different shades of gray. The dark gray cards are those that can be created directly after the **Open Card**. Light gray cards are those cards that cannot be used following the previous card. A helpful checkmark in the upper right of each selection also indicates which cards can be used in the current situation.

- Select the **Search Card**

A new **Search Card** will appear in the UI. The search string appears simple but has some very powerful search features within.

- Click and drag the right grip bar and slide it to the left, to create a new card.
- Select **Show Table Card**

Now that we have a card that can display search results, slide back to the **Search Card**.

- Type the word *Austin* and either drag the right grip bar to the left, or simply click on the right grip bar.

Depending on the performance of your system and database it may take several seconds before the results are displayed. Keep in mind that SlamData is searching the patients collection that we imported into MongoDB, and that indexes can significantly boost performance for searches.

Once the results appear, you can browse them just like you did earlier in the **Explore Card** with the controls in the bottom left of the interface.

Did you notice that in the search string earlier we did not specify which field we wanted to search? That is part of the power of SlamData. Relatively non-technical users can use SlamData to search all of their datasources with little (or even no) knowledge in advance of the data stored within.

Of course when searching all available fields for the search string it is going to take longer than if we were to explicitly define which field. Let's go back to the search card by dragging the current card to the right again, or single-click on the left grip.

Let's search for any patients currently living in the city of Dallas.

- Type the string `city:Dallas` and slide back to the **Table Card**

The results should have appeared much faster than the previous search because we told SlamData to only look at the **city** field.

We can also search on non-string values such as numbers. Let's find all of the patients who are between the ages of 45 and 50:

- Go back to the **Search Card**
- Enter the string `age:>=45 age:<=50`
- View the results in the **Table Card** again.

As one last example let's show how you can mix and match different types. We want to know how many males over age 50 used to live in California.

- Go back to the **Search Card**
- Enter the string `previous_addresses:["*"]:state:CA age:>50 gender:=male`
- View the results

See the table below for some helpful query examples:

Example	Description
<code>colorado</code>	Searches for the substring <code>colorado</code> in all fields
<code>=colorado</code>	Searches for the full word <code>colorado</code> in all fields
<code>age:=50</code>	Searches the field age for a value of 50
<code>age:>=50</code>	Searches the field age for any value over 50
<code>age:>=50</code> <code>age:<=60</code>	Searches the field age for values between or equal to 50 and 60
<code>codes:["*"]:desc:</code>	Performs a deep search through the codes array and examines each subdocument's desc field for the substring <code>flu</code>

As you can see even users with no knowledge of SQL² can perform powerful searches within SlamData!

3.2.8 2.8 Querying Data with SQL²

In addition to the **Search Card** SlamData provides a **Query Card** which allows users to execute ANSI-compatible SQL queries on top of any data source, including NoSQL databases! This is accomplished by using SlamData's SQL² dialect, which is a superset of SQL that allows dynamic modeling and querying of deeply nested, semi-structured data.

Using the same dataset we are going to perform queries, moving from basic queries to more advanced queries. Let's start off by cleaning up our Workspace.

- Go to the **Table Card**
- Flip it over
- Click on **Delete Card**

This should take you to the **Search Card**

- Flip it over
- Click on **Delete Card**

This should take you to the **Open Card**. We will be using full path names in the queries we will write, and **Query Cards** do not use the **Open Card** so let's get rid of that one as well.

- Flip it over
- Click on **Delete Card**
- Create a new **Query Card**

The UI now presents the **Query Card**. Within this card users can enter simple or very long and complex SQL² queries against one, two or more collections.

Before we perform any real queries, leave the existing contents of the card as the default. Let's create a **Table Card** to the right of this one so when the queries execute, we can see the results.

- Click the right grip.
- Create a new **Show Table Card**
- Now click back to the **Query Card**
- Type in the following query:


```
SELECT * FROM `/devguide/devdb/patients`
```

Notice how the path to the dataset is surrounded by back-ticks (`) not apostrophes (')

- Slide over to the **Show Table Card** to see the results.
- Slide back to the **Query Card**
- Type in or paste the following query:

```
SELECT
    first_name,
    last_name
FROM `/devguide/devdb/patients`
WHERE
    state="TX" AND
    city="DALLAS"
```

Note that the query can span multiple lines, and that strings are surrounded by quotation marks (") on both ends. This is a requirement for all string data types.

- Slide back to the **Show Table Card** to see the results.
- Slide back to the **Query Card**

Let's now create a query that formats the results a little cleaner:

- Type in or paste the following query:

```
SELECT
    last_name || ', ' || first_name AS Name,
    city AS City,
    zip_code AS Zip
FROM `/devguide/devdb/patients`
WHERE
    state="TX"
ORDER BY zip_code ASC
```




- Slide to the **Show Table Card** to see the results.

Notice in this query we are concatenating **last_name** and **first_name** fields together, separated by a comma. The comma itself is surrounded by apostrophes (') because it is a single character. If it was more than one character it would be a string and would require full quotation marks around it.






We have also given the results some aliases to display rather than the actual field names.




Finally we are ordering (**ORDER BY**) the results in ascending (**ASC**) order based on the **zip_code** field.

The results table should now look similar to the following image:

City	Name	Zip
BARRY	Wright,Freeda	75102
FERRIS	Mcperson,Marianela	75125
MALAKOFF	Sellers,Rasheeda	75148
DALLAS	Reid,Manuel	75204
DALLAS	Duncan,Ilona	75214
DALLAS	Atkinson,Marc	75219
DALLAS	Werner,Amalia	75226
DALLAS	Ross,Zula	75250
DALLAS	Jordan,Michaele	75301
DALLAS	Becker,Evangeline	75355



Page 1 of 20


Per page: 10


Up to this point we have been using SQL² to query simple *top-level* fields, or those fields which are not nested. We know from previous examples that this data set stores nested data in the **codes** array, but it also contains **previous_addresses** and **previous_visits** arrays.

Let's find out the total number of male and female patients from each state that have an illness related to an ulcer. This will require using the flattening operator (`[*]`) so SlamData can examine all of the documents in the **codes** array.

- Slide to the **Query Card**
- Type or paste the following query:

```
SELECT
  state AS State,
  gender AS Gender,
  COUNT(*) AS Count
FROM ` /devguide/devdb/patients `
WHERE
  codes[*].desc LIKE "%ulcer%"
GROUP BY state, gender
ORDER BY COUNT(*) DESC
LIMIT 20
```

- Slide to the **Show Table Card** to see the results.

SQL² allows for very complex queries. You can find out more by reviewing the SQL² Reference. Additional features include using the **JOIN** command to combine data from two or more tables, utilizing variables within queries (as explained in Section 3), using standard math operations, retrieving not only field values but also field names dynamically, and much more.

Now that you have a good idea of what can be accomplished with SQL² queries, let's create some forms that your users can interact with. These forms can drive the results of the charts we'll use for visualization, which makes it easy for your users to find, report and chart complex data without understanding the mechanics behind it!

3.3 Section 3 - Interactive Forms and Visualizations


SlamData provides everything you need to create an interactive visual analytics environment for your users.

From this point on in the guide we will assume that we are creating an environment for medical facilities to search through patient data for various reasons. The Workspaces we create will be used by medical staff for this purpose.

3.3.1 3.1 Static Markdown Forms


We will start this section with a new Workspace. You can leave the existing Workspace alone or you can delete it if you wish.

To (optionally) delete the existing Workspace:

- If you are still in the Workspace, click on the zoom-out icon 
- Locate the **My First Test** Workspace and hover your mouse over it.

- Click on the trash can icon that appears to the right 


We'll create a new Workspace and call it **Average Weight by City**

- Click the Create Workspace icon in the upper right 
- Select the **Setup Markdown Card**

This step is necessary so that the Workspace is saved and we can go back to rename it soon.

- Create a **Show Markdown** card directly after the **Setup Markdown Card**
- Zoom back out to the database view

Let's rename the Workspace now so it's obvious that we are working with it.

- Hover over the new Workspace labeled **Untitled Workspace.slam**
- Click the Move/Rename icon to the right 
- Replace **Untitled Workspace** with **Average Weight by City** and click **Rename**
- Click on the **Average Weight by City.slam** Workspace again

We are now back in the **Setup Markdown Card**.

SlamData uses a specific form of **Markdown** sometimes referred to as SlamDown. Markdown allows a user to format text with a few simple syntax rules. SlamData's version also allows UI elements (such as drop downs, radio buttons and check boxes) to be dynamically populated from the results of queries.

Let's first show some examples of what the Markdown forms can do. Replace the text within the card with the following:

```
# Heading 1

## Heading 2

### Text formatting

* Here is an unnumbered list.
* You can have _emphasized_ and **bold** text.

1. Here is a numbered list.
2. Here is the second entry with ``inline formatting``

Paragraphs are separated by
an empty line.

This is another new paragraph.

> You can also have some nice
> block quote areas.

You can also have fenced code blocks like this:
```
SELECT * FROM `/devguide/devdb/patients`
WHERE
 first_name = "Sue"
```

### Interactive Elements

#### Input Fields

name = ____ (Sue)

numberOnly = #____ (1984)

#### Selectors

city = {Austin, Dallas, Houston}

favoriteColor = (x) red () blue () green

computers = [] PC [x] Mac [x] Linux

beginDate = ____-__-__

stopTime = __:__

fullDateTime = ____-__-__ __:__
```

- Click over to the **Show Markdown Card** to view the results.

Notice how much control you have over the presentation of the information. You can also include links and images inside of Markdown as well. For a full description of all fields and their behavior see the SlamDown Reference.

- Click back to the **Setup Markdown Card**

Replace the contents with something more useful and appropriate to our use case:

```
## General Patient Information

There are !`` SELECT COUNT(*) FROM `/devguide/devdb/patients` `` patients

_Average_ age: !`` SELECT AVG(age) FROM `/devguide/devdb/patients` ``

The *Heaviest* patient: !`` SELECT MAX(weight) FROM `/devguide/devdb/patients` `` pounds

The **Shortest** patient: !`` SELECT MIN(height) FROM `/devguide/devdb/patients` `` inches
```

- Click over to the **Show Markdown Card** to see the results.

Notice that we populated some of the text with actual results from the database. Keep in mind that to print the results of a query in Markdown, the query must begin with an exclamation point (!) and two back-ticks (``) and end with two more back-ticks (``).

- Click back to the **Setup Markdown Card**

We will use similar syntax to populate the elements of an interactive form in the next section.

3.3.2 3.2 Interactive Markdown Forms


Here is where things get really fun for both you and your own users. Let's actually provide the functionality that we promise with the title of **Average Weight by City**.

First we want the user to select the state to report on. This will then allow us to query the database for patients that reside in cities within that state.

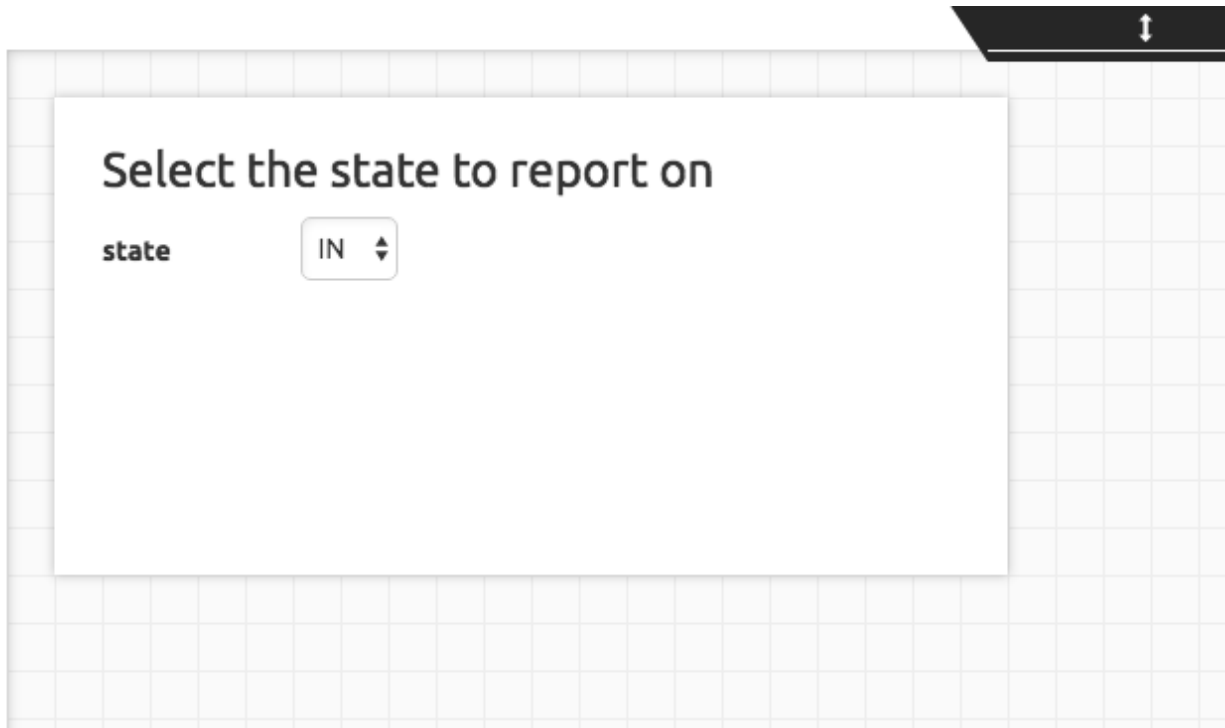
- Replace the contents of the current **Markdown Setup Card** with the following code.

```
### Select the state to report on

state = {!``SELECT DISTINCT(state) FROM `/devguide/devdb/patients` ORDER BY state``}
```


- Click over to the **Show Markdown Card** to see the results.
- Click on the dropdown next to **State** to see that the element was populated with the query we typed in.
- Flip the **Show Markdown Card** over by clicking the icon in the upper right 
- Select the **Wrap** option.

Note that your interface should now look similar to the following:

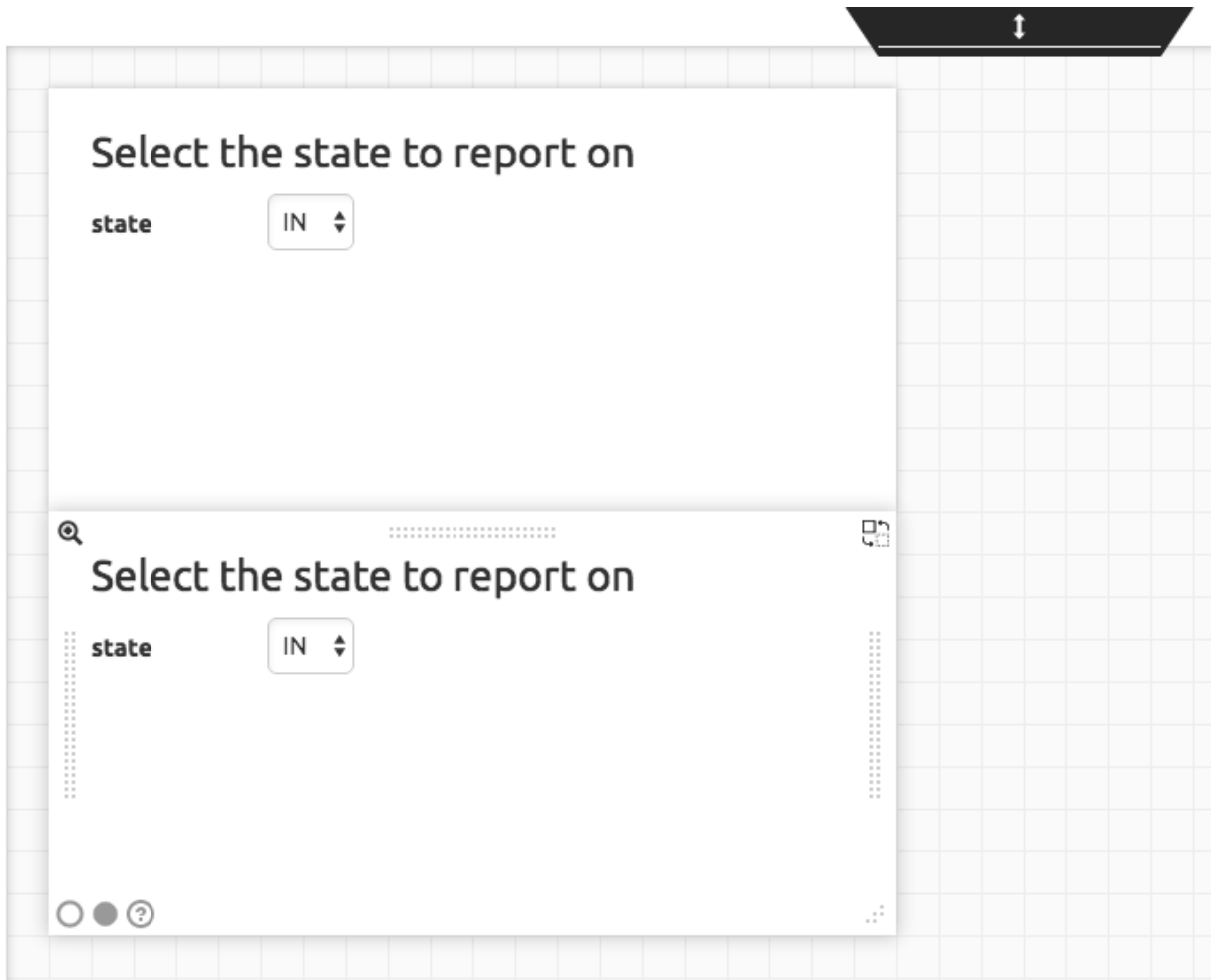


You can drag the existing deck around the board now. You can also click and drag the left and right hand grips just as before to see the previous cards.

- Click on the deck to make it active.

- Flip the deck by clicking the icon 
- Select the **Mirror** option.

Your interface should now look similar to the following:



We have just mirrored a deck. This means that the second deck starts off from where the first left off, but it also means any changes to the first deck will immediately impact the second deck as well. This is how we chain events in a Workspace and allow the actions in one deck to affect other decks.


- Click on the new second deck to make it active.
- Create a new card in this second deck, selecting the **Query Card**
- Type in or paste the following query into the **Query Card**:

```
SELECT
  city AS City,
  AVG(weight) AS AvgWeight
FROM ` /devguide/devdb/patients `
WHERE
  state IN :state
GROUP BY
  city
ORDER BY AVG(weight) DESC
```

Whenever a variable from a Markdown form is used in a query it must be preceded by a colon (:).

Also note that we can **ORDER BY** an aggregation value such as **AVG**.



- Click on the right grip to create a new card and select **Show Table Card**
- Adjust the decks with their border controls until they look similar to the following image:





Select the state to report on

state

AvgWeight	City
466	TODDVILLE
445	DEAL ISLAND
367	GLEN ECHO
340	FREELAND
333	COLTONS POINT
331	MARBURY
314	GREENBELT
308	RIDGE
296	LA PLATA
289	PATUXENT RIVER



Page of 14



- Select a different state in the first deck and watch the results table update automatically.

Viewing data in table form is useful but sometimes a graphical representation makes all the difference. To prepare for that, let's go back and change query and limit the results to 20 cities so a bar chart doesn't appear as crowded.

- Click the left grip to go back to the **Query Card**
- Add the following line to the end of the query:


```
LIMIT 20
```

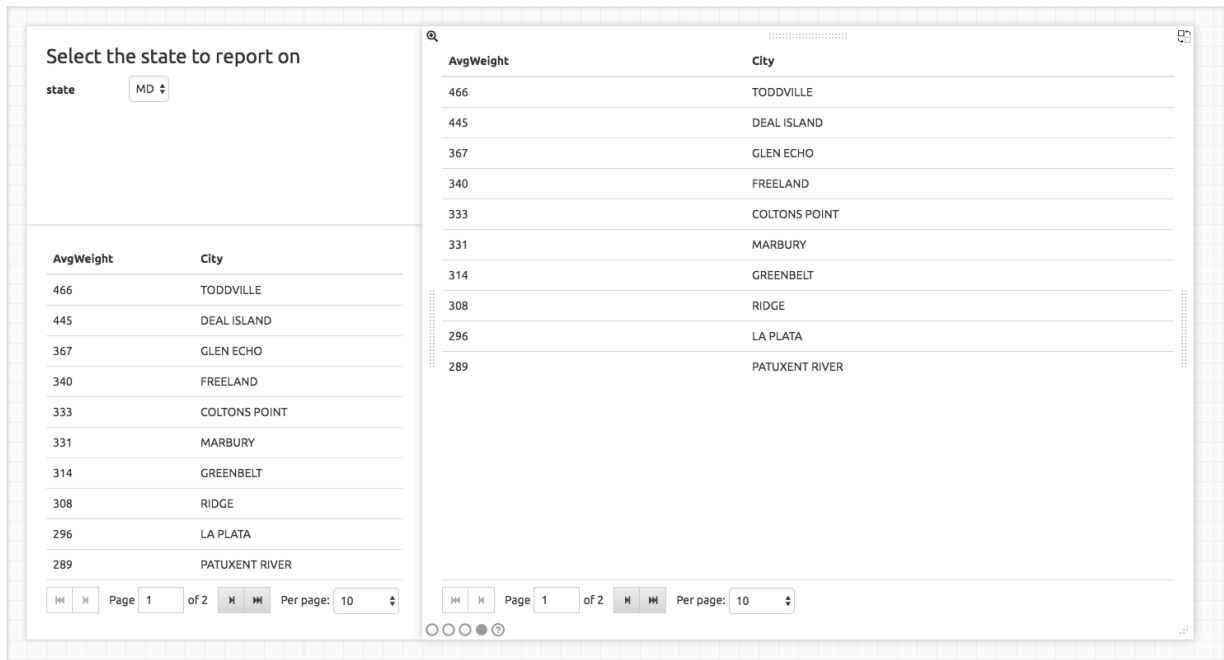
- Slide back over to the **Show Table Card**

Now we are ready to add some visualizations!

3.3.3 3.3 Creating a Chart

Before creating an actual chart we need to set it up. Remember earlier that decks can build off one another. We need to now mirror the **Show Table Card**:

- Click on second deck to make it active
- Click on the flip icon to flip the deck over 
- Select the Mirror option.
- Drag the newly mirrored deck to the right and resize it so your interface looks similar to the following image:



- Flip the new deck over and now select the **Setup Chart** option



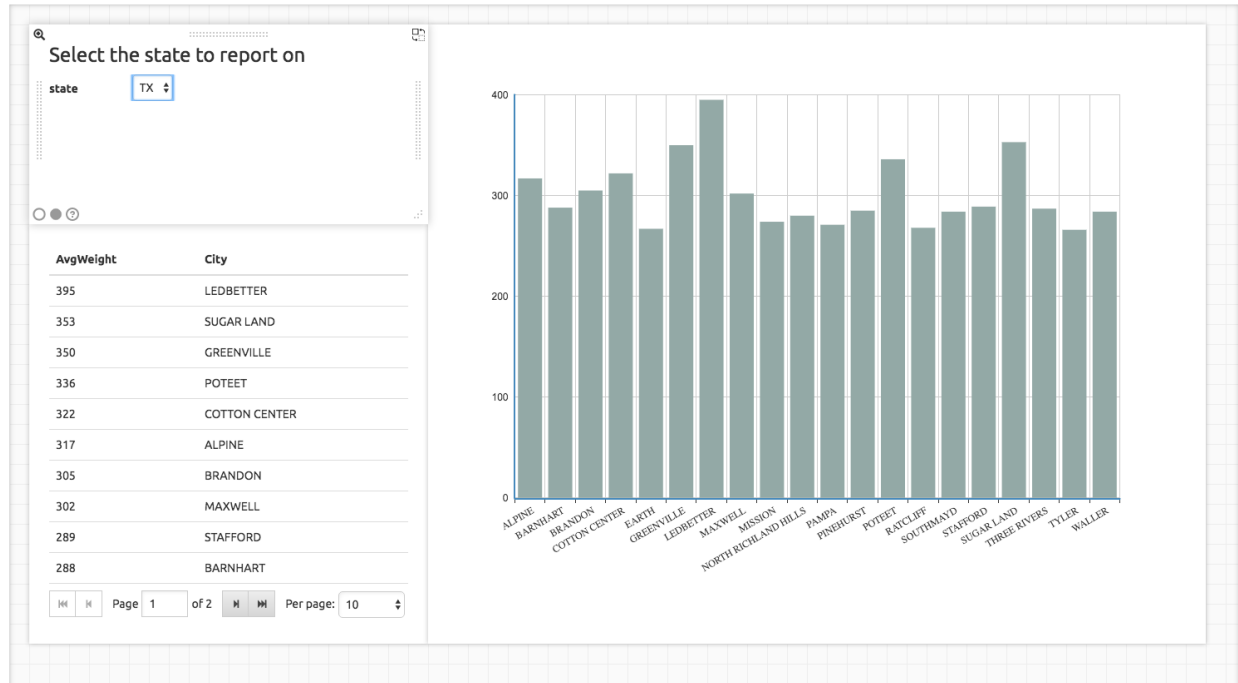
- Select the Bar Chart icon on the left

The bar chart icon will change from gray to blue to show that it is active.

- In the **Category** drop down select **.City** as the axis source

- Slide to the right to create a new card and select the **Show Chart** option

Your interface should now look like the following image:



- Select a new state in the first deck and watch both of the other decks update dynamically.
- Try hovering your mouse over the individual bars in the chart and you can view the actual value.

Setting up interactive forms and charts is as simple as that! In the next section we'll go over how to share these charts with others.

3.4 Section 4 - Publishing and Simple Embedding

3.4.1 4.1 - Publishing

SlamData makes it easy to take all the work you've done up to this point and publish it so that others can use it as well.

- Click the flip icon on the **Draftboard Card**. Note that this is the card that contains all of the existing decks. Just as each deck has a back to it, each card does as well, including the **Draftboard Card**. Be sure not to flip any of the three decks we've created - click the icon in the white box border surrounding the other decks.
- Select the **Publish deck** option.

A URL will be presented to you that you can share with others. The URL will only be accessible while SlamData is running.

Warning: Published URLs

Anyone with access to the URL may be able to view this deck. They may also be able to modify the link to view or edit any deck in this workspace. Please see Securing SlamData Community Edition for more information.

NOTE: SlamData Advanced Edition provides complete security including authorization, authentication and full auditing.

3.4.2 4.2 - Simple Embedding

SlamData allows content authors and developers to embed Decks into external web applications such as customer portals, dashboards, etc.

4.2.1 - Downloading Sample Code

For examples of how to do this go to this . You can either download the zip file or clone the repository

Option 1 - Download Zip File

- Click the .
- Click the green **Clone or download** button.
- Select **Download ZIP**
- Unzip the contents once downloaded

Option 2 - Clone the Repository

You will need to install [git](#) and then type the following in a command line terminal:

```
git clone https://github.com/slamdata/slamdata-dev-examples.git
cd slamdata-dev-examples
```

This section will be using the **sample1** code from that repository.

- Open a web browser and open the **sample1/index.html** file.

In this mock-up app we are going to simulate a reporting application that allows healthcare professionals to run a few reports based on patient data. You can see the in this example we will have two reports.

4.2.2 - Sample Report 1

We have already done most of the work for the first report, we just need to embed the appropriate code from SlamData into the web application. Again - this is a mock-up application which does not actually generate dynamic web pages, so we will be modifying static HTML files to simulate this. The guide will point out relevant areas in code that should be generated by your application.

- If not already open then navigate to the **Average Weight by City** Workspace
- Flip the **Draftboard Card** over (again - this is the card that surrounds all of the decks with a white border)
- Select the **Embed Deck** option

Notice that SlamData provides sample code to copy and paste into your own application or HTML file.

4.2.2.1 Snippet 1 Code

- Copy the highlighted part of the text (see image below).

Embed deck

```
<!-- This is the generic SlamData embedding code. -->
<!-- You can put this in the header or save it in a .js file included in the document -->
<script type="text/javascript">
var slamdata = window.SlamData = window.SlamData || {};
slamdata.embed = function(options) {
  var queryParts = [];
  if (options.permissionTokens) queryParts.push("permissionTokens=" + options.permissionTokens.join(",");
  if (options.stylesheets) queryParts.push("stylesheets=" + options.stylesheets.map(encodeURIComponent).join(",");
  var queryString = "?" + queryParts.join("&");
  var varsParam = options.vars ? "?vars=" + encodeURIComponent(JSON.stringify(options.vars)) : "";
  var uri = "http://localhost:8080/files/workspace.html" + queryString;
  var iframe = document.createElement("iframe");
  iframe.width = iframe.height = "100%";
  iframe.frameBorder = 0;
  iframe.src = uri + "#" + options.deckPath + "/" + options.deckId + "/view" + varsParam;
  var deckElement = document.getElementById("sd-deck-" + options.deckId);
  if (deckElement) deckElement.appendChild(iframe);
};
</script>

<!-- This is the DOM element that the deck will be embedded into -->
```

- Open the **sample1/report1.html** file in a text editor
- Paste the **Snippet 1 code** that SlamData provided into the HTML file's <HEAD> section, just after the line that reads <!-- SLAMDATA SNIPPET 1 -->.

Let's refer to this section of code as **Snippet 1**.

Snippet 1 should be placed within the HTML file's <HEAD> tags as it's a JavaScript snippet. This section of code can easily be inserted into individual HTML files, or you can save it to its own JavaScript (.js) file to include in many documents.

This snippet is generic and is typically the same regardless of what is being embedded - which makes it a great candidate to save into that JS file and insert into multiple web pages based on your web application framework.

You'll see with Snippets 2 and 3 how we control what is being seen even though the code in this snippet is generic.

4.2.2.2 Snippet 2 Code

- Go back to the SlamData UI. Scroll down until you see the next section of sample code, highlighted in the image below.

Embed deck

```
var iframe = document.createElement( 'iframe' );
iframe.width = iframe.height = "100%";
iframe.frameBorder = 0;
iframe.src = uri + "#" + options.deckPath + "/" + options.deckId + "/view" + varsParam;
var deckElement = document.getElementById("sd-deck-" + options.deckId);
if (deckElement) deckElement.appendChild(iframe);
};
</script>

<!-- This is the DOM element that the deck will be embedded into -->
<div id="sd-deck-5e2ce240-bb3f-4aca-8471-dae06925a429"></div>

<!-- This is the code that performs the deck insertion, placing it at the end of the body is suggested -->
<script type="text/javascript">
SlamData.embed({
  deckPath: "/devguide/devdb/Average+Weight+by+City.slam/",
  deckId: "5e2ce240-bb3f-4aca-8471-dae06925a429",
  // An array of custom stylesheets URLs can be provided here
  stylesheets: []
});
</script>
```

- Copy the `id` value from the `<div>` element. It starts with `sd-deck-`.
- Go back to your text editor, and replace the text `REPLACE_ME` with the copied value. This should be in the section directly below `<!-- SLAMDATA SNIPPET 2 -->`.

One important piece to note here is that the example **report1.html** file is formatted with some CSS and `<div>` tags already. In your own application you can either paste the entire line of code that SlamData provides, or create your own `<div>` tag and programmatically insert the `id` as we did in this example.

4.2.2.3 Snippet 3 Code

- Go back to the SlamData UI. Scroll down until you see the next section of sample code, highlighted in the image below.

Embed deck

```
var iframe = document.createElement( 'iframe' );
iframe.width = iframe.height = "100%";
iframe.frameBorder = 0;
iframe.src = uri + "#" + options.deckPath + "/" + options.deckId + "/view" + varsParam;
var deckElement = document.getElementById("sd-deck-" + options.deckId);
if (deckElement) deckElement.appendChild(iframe);
};
</script>

<!-- This is the DOM element that the deck will be embedded into -->
<div id="sd-deck-5e2ce240-bb3f-4aca-8471-dae06925a429"></div>

<!-- This is the code that performs the deck insertion, placing it at the end of the body is suggested -->
<script type="text/javascript">
SlamData.embed({
  deckPath: "/devguide/devdb/Average+Weight+by+City.slam/",
  deckId: "5e2ce240-bb3f-4aca-8471-dae06925a429",
  // An array of custom stylesheets URLs can be provided here
  stylesheets: []
});
</script>
```

- Copy the highlighted text as shown above.
- Go back to your text editor, and paste the contents of **Snippet 3 code** directly below the line that reads `<!-- SLAMDATA SNIPPET 3 -->`.
- Save your **sample1/report1.html** file to disk.

This is the code that provides the most important information when embedding the Deck. Notice the variables `deckPath` and `deckId`. This section of code would normally be generated by your own web application, and these two variables would be populated based on some logic in your application.

In small examples where we are only using two reports it's easy enough to paste this code directly into files; however when the number of reports that are being embedded grows, it will quickly start to make sense when to programmatically generate this code.

4.2.2.4 Full Code - Report 1

After making changes to the **sample1/report1.html** file and saving it, it should appear almost identical to the following. The differences will only be related to your local environment, such as possibly the hostname, the `deckId`, `sd-deck` value, etc.

Code:

```
<head>
<meta charset="utf-8">
<title>Your Reporting App</title>
<link rel="stylesheet" type="text/css" href="styles.css">

<!-- SLAMDATA SNIPPET 1 -->

<script type="text/javascript">
var slamdata = window.SlamData = window.SlamData || {};
slamdata.embed = function(options) {
  var queryParts = [];
```

```

    if (options.permissionTokens) queryParts.push("permissionTokens=" + options.permissionTokens.join("&"));
    if (options.stylesheets) queryParts.push("stylesheets=" + options.stylesheets.map(encodeURIComponent).join("&"));
    var queryString = "?" + queryParts.join("&");
    var varsParam = options.vars ? "?vars=" + encodeURIComponent(JSON.stringify(options.vars)) : "";
    var uri = "http://localhost:8080/files/workspace.html" + queryString;
    var iframe = document.createElement("iframe");
    iframe.width = iframe.height = "100%";
    iframe.frameBorder = 0;
    iframe.src = uri + "#" + options.deckPath + "/" + options.deckId + "/view" + varsParam;
    var deckElement = document.getElementById("sd-deck-" + options.deckId);
    if (deckElement) deckElement.appendChild(iframe);
  };
</script>

</head>
<body>
  <div class="container">
    <nav class="navbar navbar-default" role="navigation">
      <div class="navbar-header">
        <div class="row">
          <a class="navbar-brand" href="index.html"></a>
          <a class="navbar-brand" href="index.html"></a>
        </div>
        <div class="row">
          <a class="navbar-brand" href="index.html"></a>
          <a class="navbar-brand" href="index.html">Your Reporting App</a>
        </div>
      </div>
    </nav>
    <div id="main">
      <div class="container">
        <div class="row">
          <div class="col-md-6">
            <H3>Average Weight by City</H3>
          </div>
        </div>

        <!-- SLAMDATA SNIPPET 2 -->

        <div
          style="min-height: 700px;min-width: 800px;"
          class="col-lg-12 col-md-12 col-sm-12"
          class="row"
          id="sd-deck-5e2ce240-bb3f-4aca-8471-dae06925a429">

        </div>
      </div>
    </div>
  </div>

  <!-- SLAMDATA SNIPPET 3 -->

  <script type="text/javascript">
    SlamData.embed({
      deckPath: "/devguide/devdb/AverageWeight+by+City.slam/",
      deckId: "5e2ce240-bb3f-4aca-8471-dae06925a429",
      // An array of custom stylesheets URLs can be provided here
      stylesheets: []
    });
  </script>

```

```

    });
  </script>

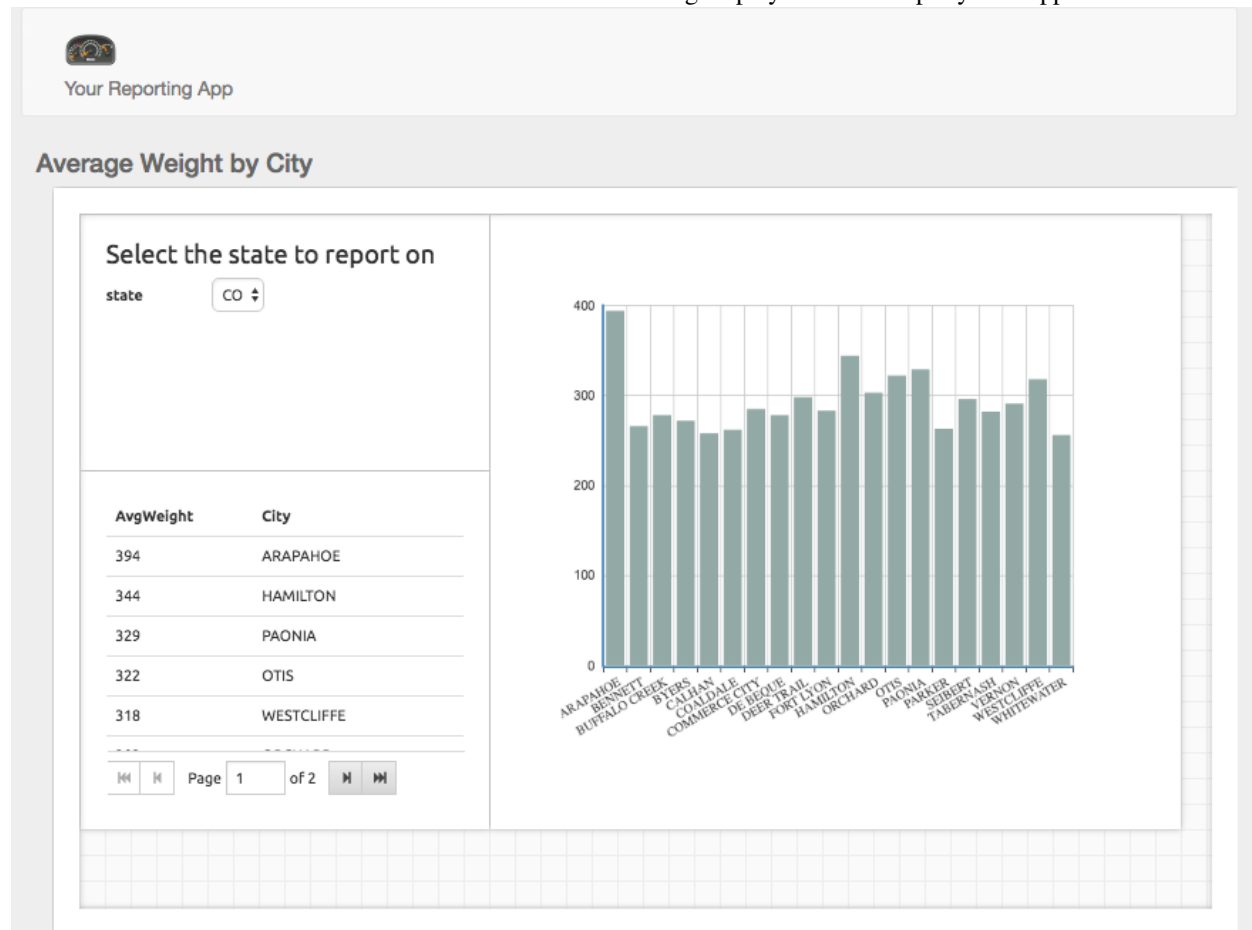
</body>

```

4.2.2.4 Overview of Report 1

Now that the **sample1/report1.html** file has been saved, it can be loaded into the web browser.

- Go back to the browser where **sample1/index.html** is displayed, or optionally re-open the file with the browser.
- Click on the **Average Weight by City** link. It should appear similar to the image below
- Observe how the entire contents of that Deck is now being displayed in a third party web application.



The purpose of copying and pasting all of the values in the file above was to show what a completed web page is comprised of, including the code to make the calls to SlamData.

A larger web application would typically generate the entire contents of **sample1/report1.html**, replacing the relevant values in **Snippet 2** and **Snippet 3**. Again, **Snippet 1** can simply be saved as a JS file and included in the necessary pages within the application.

4.2.3 - Sample Report 2

This section will give you the relevant information for creating a new Workspace, Deck and report, but will not give you the full instructions.

From your previous work you understand how to create a Workspace, rename it, add cards, etc. The list below shows the necessary cards you'll need to create and their order. Remember you'll need to **Wrap** everything to be able to move the individual decks around.

Initial Card Order:

1. Query Card (wrap the deck here)

Query:

```
SELECT
  COUNT(*) as Count,
  state,
  gender
FROM ` /devguide/devdb/patients`
WHERE
  codes[*].desc like "%ulcer%"
GROUP BY state, gender
```

2. Show Table Card (mirror the deck here)

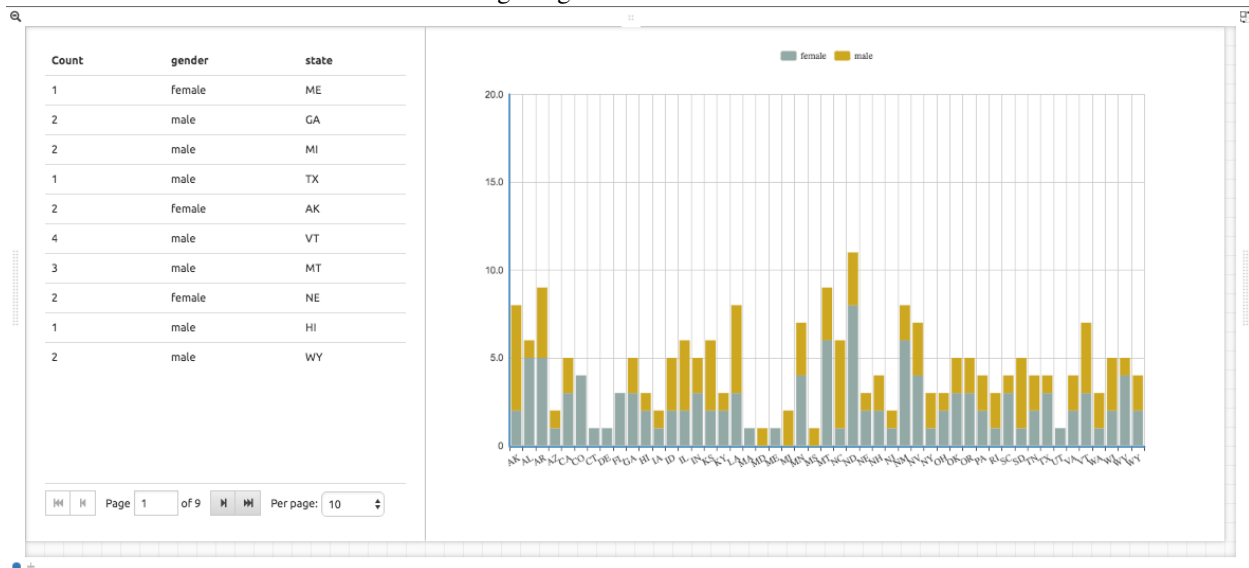
Mirrored Deck Card Order

1. Setup Chart Card

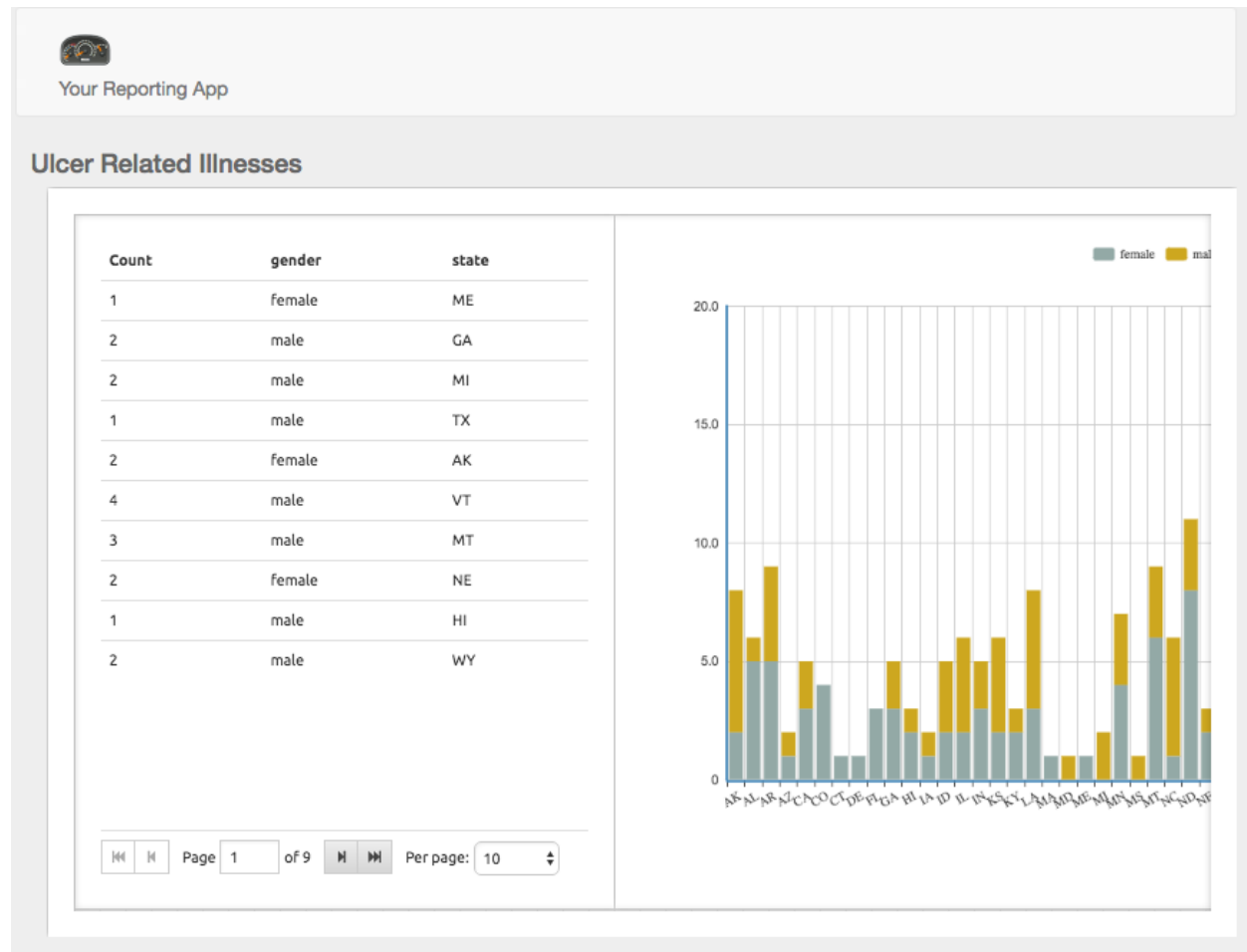
- Bar Chart
- Category: .state
- Series: .gender

2. Display Chart Card

The results should look similar to the following image:



Copy all of the relevant data from the **Embed Deck** option and paste it into the **sample1/report2.html** file. Once it is saved, you can click on the **Ulcer-related Illnesses by Gender** report in the mock-up app and see something similar to the following image. Note that in this image the user would need to scroll right to see the full chart.



3.5 Section 5 - Secure Embedding

This section describes how to enable user authorization and authentication with examples. This not only provides security when users are within the SlamData user interface but can also be used to control access from other web applications as well.

Attention: SlamData Advanced Required



This section requires SlamData Advanced Edition

This section assumes you understand the basics of SlamData Advanced Edition security [here](#)

SlamData Advanced Edition utilizes [OpenID Connect](#), which is a simple identity layer on top of the OAuth 2.0 protocol.

3.5.1 5.1 Bootstrapping Security

If you have already setup authentication for SlamData you may skip this section.

To enable user security a default administrator group must be created along with a user email. In the next step this user will be provided all permissions within SlamData. This allows the user to perform administration tasks within the user interface as well as make calls via the SlamData API that require elevated privileges.

From the SlamData Advanced Edition directory, type the following to bootstrap the SlamData Advanced Edition environment, replacing the email address with the user you wish to authenticate with.

```
` java -jar jars/quasar.jar bootstrap -u you@example.com -g admin `
```

3.5.2 5.2 Creating an OIDC Provider

If you have already setup an OIDC provider you may skip this section.

At least one OpenID Connect (OIDC) Provider must be listed in the configuration file for SlamData Advanced Edition. This OpenID Connector Provider (OP) will be trusted by SlamData for authentication information.

The remainder of this guide will assume that a Google OP will be used and the examples are configured based on this assumption; however, any OpenID Connect Provider can be used.

5.2.1 Google OIDC Provider

The best method to create an OP is to follow instructions from the Google API Console project [here](#)

Most of the fields should be self explanatory. Once the project is created, go to the Credentials tab in the API Manager. Under the **Authorized redirect URIs** enter the following value and save your changes, assuming hostname and port are correct for your environment:

http://localhost:8080/files/auth_redirect.html

In SlamData's quasar-config.json file create a new entry similar based off the client_id, similar to to the highlighted portion in the image below:

```
{
  "server": {
    "port": 8080,
    "ssl": {
      "enabled": false,
      "port": 9090,
      "cert": null
    }
  },
  "mountings": {
    "/macbook/": {
      "mongodb": {
        "connectionUri": "mongodb://localhost:27017"
      }
    },
    "/aws/": {
      "mongodb": {
        "connectionUri": "mongodb://myuser:mypass@1.2.3.4:27017/myadmindb"
      }
    }
  },
  "authentication": {
    "openid_providers": [
      {
        "issuer": "https://accounts.google.com",
        "client_id": "██████████ googleusercontent.com",
        "display_name": "Google"
      },
      {
        "issuer": "https://accounts.google.com",
        "client_id": "██████████ googleusercontent.com",
        "display_name": "OAuth 2.0 Playground"
      }
    ]
  }
}
```

Restart SlamData Advanced Edition so the new provider will be active.

3.5.3 5.3 Logging Into SlamData

You should now be able to click on the application tab bar pull out at the top of the page.



You can then click on the **Sign In** icon to the right.

Once clicked it should display all of the OIDC Providers that are configured, similar to the image below:



Sign in with the user you specified in the bootstrap step above. This user has complete access to all SlamData Advanced Edition functionality.

3.5.4 5.4 New Decks for Secure Embedding

In this section we're going to spend time setting up SlamData so that multiple customers can utilize it from an external web application. This will require creating SQL² Views, new Workspaces and permission tokens.

Additionally we'll configure SlamData so that reports and views are now stored in a separate directory structure for enhanced security.

5.4.1 Setting up SQL² Views

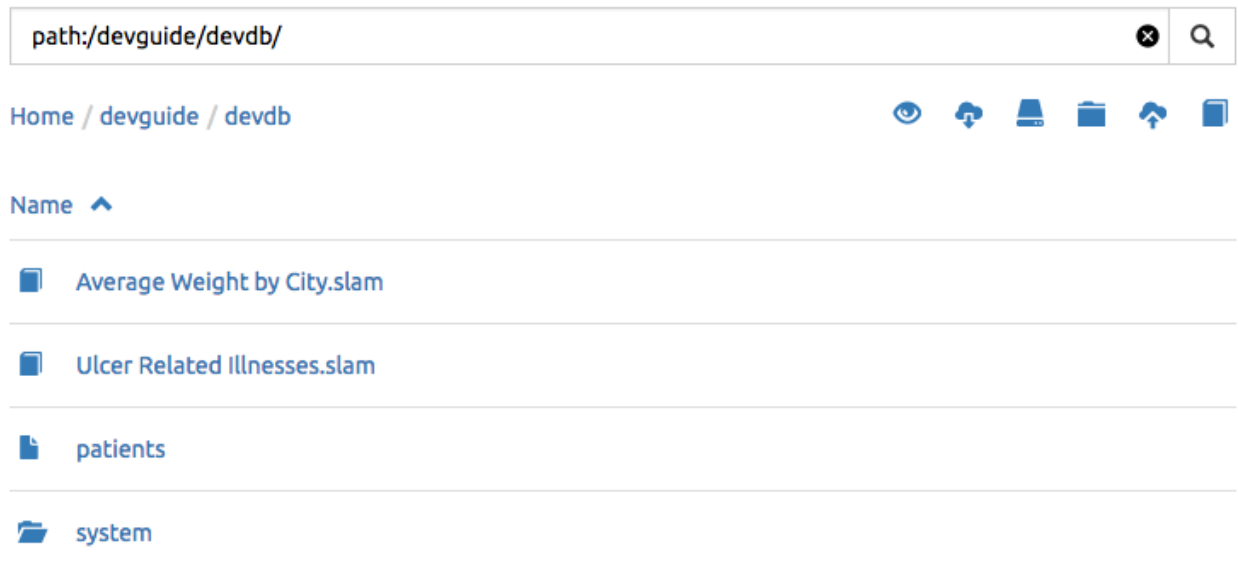
In this simulated application we will assume we are a national healthcare provider. We also want to create some reports for our healthcare professionals; however, those reports must be limited to the states to which the healthcare professional is licensed.



One option would be to create a report for each state, and specify access to that report for each of that state's healthcare professionals. Now consider we would have to do that for **each report type**. So if one report type was **Average Age by City**, we would have to create 50 of those reports, and then provide access to each professional in each state. Then if we wanted another report called **Most Diagnosed Disease** we would have to create yet another 50 reports, one for each state, and setup the professionals to view it again.

The better answer to this is to create a single report, and change the source data set based on who is logged in. This is accomplished through the use of a view. Let's set one up as an example.

In SlamData, navigate to the root folder. We have primarily been working in the `/devguide/devdb` database which means we'll need to go up two levels.


From the main Home page in SlamData, to the `devguide` mount, then into the `devdb` database where the previous Workspaces were created, similar to this image:



- Click on the Create Folder icon 
- Hover over the **Untitled Folder** and click the Move-Rename icon to the right 
- Rename the folder to `state-views`

Now we have a folder which is specifically designed to hold views. This makes it easier to manage.

Now let's create our first view.

- Click into the **state-views** folder
- Click on the Mount icon 
- In the mount dialog provide `colorado` as the name
- Select `SQL2` as the mount type
- Paste or type the following query into the **SQL² query** field:

```
SELECT * FROM `/devguide/devdb/patients` WHERE state = "CO"
```

- Click **Mount**

Congratulations, you just created a view! Now this view path can be used in queries. When this view is used as the data source, the results will only be those documents where the `state` field is `CO`.

What we just did can also be accomplished via the SlamData API quite easily. This is covered in the SlamData API Reference. To create a view for each of the 50 states would take some time through the user interface (even with the API), so let's create just one more view to use.

- Create another view named `texas` that queries against the `state` field for the value of `TX`

We'll now use the **colorado** and **texas** views as the data sources for some of our reports.

5.4.2 Setting up the Reports

Just like we setup a special folder for the state-views, we will now setup a special folder for the reports we wish to securely embed into third party web applications.

- Navigate back to the `/devguide/devdb` location within SlamData
- Create a new folder and rename it `reports`
- Click into the **reports** folder

We are only going to create a single report but this process can of course be repeated for as many reports as you like. This report will make use of the views we created previously.



- Click on the Create Workspace icon
- Create a **Setup Variables Card**
- Provide the values from the following table:

Field	Value
Name	<code>viewpath</code>
Type	SQL² Identifier
Default value	<code>/devguide/devdb/state-views/colorado</code>

- Create a **Query Card** with the following query:

```
SELECT
  count (codes[*]),
  _id as id,
  first_name,
  last_name
FROM :viewpath
GROUP BY _id
ORDER BY count (codes[*]) DESC
LIMIT 20
```

- Create a **Setup Chard Card** with the following settings:

Field	Value
Chart Type	Bar Chart
Category	.City
Default value	<code>/devguide/devdb/state-views/colorado</code>

- Create a **Show Chart Card**

We've created an interesting chart. Let's go back out and rename the Workspace now.

- Zoom back out to the navigation screen
- Rename the **Untitled Workspace.slam** Workspace to **Average Age by City**
- Click into the **Average Age by City** Workspace again



- Flip the deck
- Select the **Embed Deck** icon

This screen should look familiar! You'll notice that a few new entries are now residing in the code. Specifically the `viewpath` variable is exposed. We'll be able to change this value later to control which data set we're looking at.

- Click on the **Include a permission token...** checkbox at the bottom of the code window.

Notice how the `permissionTokens` value is now populated within the code. Now we are ready to securely embed this deck into our simulated web application.

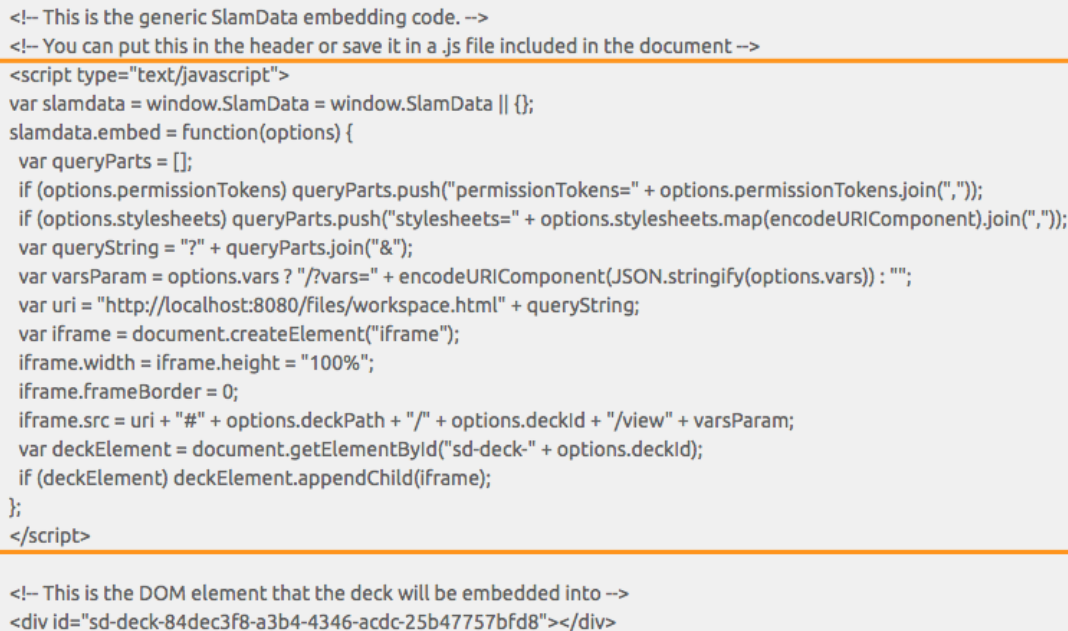
5.4.3 - Setting up the Web Application

Now that we have the views and reports created we can move on to copying the provided code into the appropriate HTML files to simulate our healthcare web application.

5.4.3.1 Snippet 1 Code

- Copy the highlighted part of the text (see image below).

Embed deck



```
<!-- This is the generic SlamData embedding code. -->
<!-- You can put this in the header or save it in a .js file included in the document -->

<script type="text/javascript">
var slamdata = window.SlamData = window.SlamData || {};
slamdata.embed = function(options) {
  var queryParts = [];
  if (options.permissionTokens) queryParts.push("permissionTokens=" + options.permissionTokens.join(", "));
  if (options.stylesheets) queryParts.push("stylesheets=" + options.stylesheets.map(encodeURIComponent).join(", "));
  var queryString = "?" + queryParts.join("&");
  var varsParam = options.vars ? "?vars=" + encodeURIComponent(JSON.stringify(options.vars)) : "";
  var uri = "http://localhost:8080/files/workspace.html" + queryString;
  var iframe = document.createElement("iframe");
  iframe.width = iframe.height = "100%";
  iframe.frameBorder = 0;
  iframe.src = uri + "#" + options.deckPath + "/" + options.deckId + "/view" + varsParam;
  var deckElement = document.getElementById("sd-deck-" + options.deckId);
  if (deckElement) deckElement.appendChild(iframe);
};
</script>

<!-- This is the DOM element that the deck will be embedded into -->
<div id="sd-deck-84dec3f8-a3b4-4346-acdc-25b47757bfd8"></div>
```

- ☐ Include a permission token so the deck can be accessed by anyone who has the access to this script. You may undo this by revoking access.

- Open the **sample2/report1.html** file in a text editor (note this is **sample2** now, not **sample1**)
- Paste the **Snippet 1 code** that SlamData provided into the HTML file's `<HEAD>` section, just after the line that reads `<!-- SLAMDATA SNIPPET 1 -->`.

Let's refer to this section of code as **Snippet 1**.

As before, this snippet is ideal for usage in an external JS file that can be included in multiple web pages.

5.4.3.2 Snippet 2 Code

- Go back to the SlamData UI. Scroll down until you see the next section of sample code, highlighted in the image below.

Embed deck

```
var varsParam = options.vars ? "?vars=" + encodeURIComponent(JSON.stringify(options.vars)) : "";
var uri = "http://localhost:8080/files/workspace.html" + queryString;
var iframe = document.createElement("iframe");
iframe.width = iframe.height = "100%";
iframe.frameBorder = 0;
iframe.src = uri + "#" + options.deckPath + "/" + options.deckid + "/view" + varsParam;
var deckElement = document.getElementById("sd-deck-" + options.deckid);
if (deckElement) deckElement.appendChild(iframe);
};
</script>

<!-- This is the DOM element that the deck will be embedded into -->
<div id="sd-deck-84dec3f8-a3b4-4346-acdc-25b47757bfd8"></div>

<!-- This is the code that performs the deck insertion, placing it at the end of the body is suggested -->
<script type="text/javascript">
SlamData.embed({
  deckPath: "/devguide/devdb/reports/Average+Age+by+City.slam/",
  deckid: "84dec3f8-a3b4-4346-acdc-25b47757bfd8",
  // An array of custom stylesheets URLs can be provided here
  stylesheets: [],
  // The variables for the deck(s), you can change their values here:
  vars: {
    "84dec3f8-a3b4-4346-acdc-25b47757bfd8": {
```

- ☐ Include a permission token so the deck can be accessed by anyone who has the access to this script. You may undo this by revoking access.

- Copy the `id` value from the `<div>` element. It starts with `sd-deck-`.
- Go back to your text editor, and replace the text `REPLACE_ME` with the copied value. This should be in the section directly below `<!-- SLAMDATA SNIPPET 2 -->`.

One important piece to note here is that the example **report1.html** file is formatted with some CSS and `<div>` tags already. In your own application you can either paste the entire line of code that SlamData provides, or create your own `<div>` tag and programmatically insert the `id` as we did in this example.

5.4.3.3 Snippet 3 Code

- Go back to the SlamData UI. Scroll down until you see the next section of sample code, highlighted in the image below.

Embed deck

```
if (deckElement) deckElement.appendChild(iframe);
};
</script>

<!-- This is the DOM element that the deck will be embedded into -->
<div id="sd-deck-84dec3f8-a3b4-4346-acdc-25b47757bfd8"></div>

<!-- This is the code that performs the deck insertion, placing it at the end of the body is suggested -->
<script type="text/javascript">
SlamData.embed({
  deckPath: "/devguide/devdb/reports/Average+Age+by+City.slam/",
  deckId: "84dec3f8-a3b4-4346-acdc-25b47757bfd8",
  // An array of custom stylesheets URLs can be provided here
  stylesheets: [],
  // The variables for the deck(s), you can change their values here:
  vars: {
    "84dec3f8-a3b4-4346-acdc-25b47757bfd8": {
      "viewpath": "/devguide/devdb/state-views/colorado"
    }
  },
  permissionTokens: ["32dd8a8455145987e374adc81fa39638028db72890bba3df19fc1e2a013b6573"]
});
</script>
```

- ☐ Include a permission token so the deck can be accessed by anyone who has the access to this script. You may undo this by revoking access.

- Copy the highlighted text as shown above.
- Go back to your text editor, and paste the contents of **Snippet 3 code** directly below the line that reads `<!-- SLAMDATA SNIPPET 3 -->`.
- Save your **sample2/report1.html** file to disk.
- Now go to your browser and load **sample1/index.html**
- Click on the **Average Age by City - Colorado** link

Notice how the Deck is embedded securely inside of our simulated web application.

Try changing the secret token in the **sample2/report1.html** file and reloading the page. You'll notice that you receive an authentication error.

We are now going to use the exact same report, and same code but provide this functionality to our Texas healthcare professionals as well.

From the command line inside of the repository directory, type or paste the following command:

```
cp sample2/report1.html sample2/report2.html
```

- Open the **sample2/report2.html** file with a text editor.
- Change the title of the page in the `<H3>` header to **Average Age by City - Texas**
- Change the **viewpath** value toward the bottom of this file to `/devguide/devdb/state-views/texas`
- Save your changes
- Open the **sample2/index.html** file again, and now click on the **Average Age by City - Texas** report.

Notice that with just the change of the viewpath we are able to provide this to our Texas professionals as well.

In a real-world application we would generate the web pages represented by **report1.html** and **report2.html**, replacing the variables where necessary.



Helpful Tips

This Helpful Tips document provides SQL² snippets that may not otherwise be covered in the other guides.

For information on how to use SlamData from a user perspective see the SlamData Administration Guide

For information on how to use SlamData from a user perspective see the SlamData Users Guide (not implemented yet)

Examples in this guide will show the SQL² query as well as the generated MongoDB query directly below it for reference.

4.1 Section 1 - Basic Queries

4.1.1 1.1 Counting

1.1.1 Documents / Rows

SQL Example

```
SELECT COUNT(*) FROM `/devguide/devdb/patients`
```

MongoDB query equivalent

```
db.patients.aggregate([
  {
    "$group": {
      "0": { "$sum": { "$literal": NumberInt("1") } },
      "_id": { "$literal": null }
    }
  },
  { "$limit": NumberLong("11") }],
{ "allowDiskUse": true });
```

1.1.2 Documents / Rows with Filter

SQL Example

```
SELECT COUNT(*)
FROM `/devguide/devdb/patients`
WHERE age >= 50
```

MongoDB query equivalent

```
db.patients.aggregate([
  {
    "$match": {
      "$and": [
        {
          "$or": [
            { "age": { "$type": NumberInt("16") } },
            { "age": { "$type": NumberInt("18") } },
            { "age": { "$type": NumberInt("1") } },
            { "age": { "$type": NumberInt("2") } },
            { "age": { "$type": NumberInt("9") } },
            { "age": { "$type": NumberInt("8") } }
          ],
          "age": { "$gte": NumberInt("50") }
        ]
      }
    },
    {
      "$group": {
        "0": { "$sum": { "$literal": NumberInt("1") } },
        "_id": { "$literal": null }
      }
    },
    { "$limit": NumberLong("11") },
    { "allowDiskUse": true }
]);
```

4.1.2 1.2 Concatenating Field Values

Use the double-pipe (||) symbol to concatenate *char* and *string* values.

SQL Example

```
SELECT
  "Full Name is " ||
  first_name      ||
  ' '             ||
  last_name
FROM `/devguide/devdb/patients`
```

MongoDB query equivalent

```
db.patients.aggregate([
  { "$limit": NumberLong("11") },
  {
    "$project": {
      "0": {
        "$cond": [
          {
            "$and": [
              { "$lte": [{ "$literal": "" }, "$last_name"] },
              { "$lt": ["$last_name", { "$literal": { } }] }
            ]
          },
          {
            "$cond": [
              {
```

```

        "$and": [
            { "$lte": [{ "$literal": "" }, "$first_name" ] },
            { "$lt": ["$first_name", { "$literal": { } } ] } ]
        },
        {
            "$concat": [
                {
                    "$concat": [
                        {
                            "$concat": [{ "$literal": "Full Name is " }, "$first_name" ]
                        },
                        { "$literal": " " } ]
                    },
                    "$last_name" ]
            },
            { "$literal": undefined } ]
        },
        { "$literal": undefined } ]
    }
}
{ "allowDiskUse": true } );

```

4.1.3 1.3 Converting Data Types

SlamData provides the ability to convert between many data types:

1.3.1 TO_STRING() Function

Any data type can be turned into a string data type using the TO_STRING() function:

SQL Example

```

SELECT
    TO_STRING(DATE_PART("year", last_visit)) ||
    "_" ||
    TO_STRING(DATE_PART("month", last_visit)) AS Year_Month
FROM `/devguide/devdb/patients`

```

Example Output



Year_Month

2011-8

2015-8

2015-8

2010-11

2016-7

2015-4

2012-9

2014-10

2015-2

2012-5

⏮ ⏪ Page 5 of 1000 ⏩ ⏭



MongoDB query equivalent


```

db.patients.mapReduce(
  function () {
    emit.apply(
      null,
      (function (key, value) {
        return [
          key,
          {
            "Year_Month": (((value.last_visit instanceof Date) || (value.last_visit instanceof Timestamp))
              ? RegExp("[^-0-9]+", "g"),
            "" : ((value.last_visit.getFullYear() instanceof Timestamp) || (value.last_visit.getFullYear() instanceof Timestamp))
              ? RegExp("[^-0-9]+", "g"),
            "" : (((value.last_visit.getMonth() + 1) instanceof Timestamp) || ((value.last_visit.getYear() instanceof Timestamp)))
          }
        ]
      }))(
        this._id,
        this))
  },
  function (key, values) { return values[0] },
  {
    "out": { "replace": "tmp.gen_840a7e9a_0", "db": "devdb" },
    "limit": NumberLong("11")
  });
db.tmp.gen_840a7e9a_0.aggregate(
  [{ "$project": { "Year_Month": "$value.Year_Month" } }],
  { "allowDiskUse": true });

```

1.3.2 TO_TIMESTAMP() Function

An epoch data type can be converted into a `TIMESTAMP` data type using the `TO_TIMESTAMP()` function.

Assuming a collection has documents which contain a field `epoch` with values such as 1408255200000:

SQL Example

```

SELECT *
FROM `devguide/epochtest/c1`
WHERE TO_TIMESTAMP(epoch) <= TIMESTAMP("2016-01-01T00:00:00Z")

```

MongoDB query equivalent

```

db.c1.aggregate(
  [
    {
      "$project": {
        "__tmp2": {
          "$cond": [
            {
              "$and": [
                { "$lt": [{ "$literal": null }, "$epoch" ] },
                { "$lt": ["$epoch", { "$literal": "" } ] }
              ]
            },
            {
              "$lte": [
                {
                  "$add": [{ "$literal": ISODate("1970-01-01T00:00:00Z") }, "$epoch" ]
                },
                { "$literal": ISODate("2016-01-01T00:00:00Z") } ]
            }
          ]
        }
      }
    }
  ]
)

```

```
    },
    { "$literal": undefined }}
  },
  "__tmp3": "$$ROOT"
}
},
{ "$match": { "__tmp2": true } },
{ "$limit": NumberLong("11") },
{ "$project": { "value": "$__tmp3", "_id": false } }},
{ "allowDiskUse": true });
```

4.1.4 1.4 Grouping

1.4.1 By Calendar Quarter

Assume you have documents in a structure similar to the following:

```
{
  "_id": ObjectId("...abcd1234..."),
  ...
  "city": "AUSTIN",
  "first_name": "John",
  "last_name": "Smith",
  "middle_name": "Duke",
  "last_visit": ISODate("2016-01-01T15:56:36Z"),
  "weight": 145
  ...
}
```

We can generate a concise report showing how many patients visited per quarter, per year. This requires use of the `TO_STRING()` and `DATE_PART()` functions, as well as the modulus (%) operator to assist in rounding.

First section of query:

```
SELECT
  COUNT(*) as cnt,
  TO_STRING(DATE_PART("year",last_visit))
  || "-Q" ||
  TO_STRING((DATE_PART("quarter",last_visit)) - (DATE_PART("quarter",last_visit) %1)) AS QUARTER
```

Line 3: Converts the “year” portion of the `last_visit` field to a string

Line 4: Concatenates “-Q” to the output of Line 3

Line 5: Rounds the month to the quarter, then concatenates the output to Lines 3 and 4 and assigns the alias `QUARTER`

```
FROM ` /devguide/devdb/patients`
GROUP BY
  TO_STRING(DATE_PART("year",last_visit))
  || "-Q" ||
  TO_STRING((DATE_PART("quarter",last_visit)) - (DATE_PART("quarter",last_visit) %1))
ORDER BY QUARTER ASC
```

The `GROUP BY` clause is used here to group all quarterly entries together. The same functions are used here that are used in the `SELECT` clause for consistency. Currently aliases cannot be used in `GROUP BY` clauses as they can in `ORDER BY` clauses.

Line 1: fetches from the appropriate collection

Line 2: Starts the GROUP BY clause

Line 3: Similar to Line 3 in the previous example, converts the “year” portion of the last_visit field to a string.

Line 4: Concatenates “-Q” to the output of Line 3

Line 5: Rounds the month to the quarter, then concatenates the output to Lines 3 and 4

Line 6: Orders the results based on yearly quarters in ascending order

Full SQL example:

```
SELECT
  COUNT(*) as cnt,
  TO_STRING(DATE_PART("year",last_visit))
  || "-Q" ||
  TO_STRING((DATE_PART("quarter",last_visit) - (DATE_PART("quarter",last_visit) %1)) AS QUARTER
FROM `/devguide/devdb/patients`
GROUP BY
  TO_STRING(DATE_PART("year",last_visit))
  || "-Q" ||
  TO_STRING((DATE_PART("quarter",last_visit) - (DATE_PART("quarter",last_visit) %1))
ORDER BY QUARTER ASC
```

Results in the following table:



QUARTER	cnt
2006-Q1	55
2006-Q2	41
2006-Q3	66
2006-Q4	38
2007-Q1	64
2007-Q2	73
2007-Q3	67
2007-Q4	55
2008-Q1	71
2008-Q2	102

⏮

⏪

Page 1 of 5

⏩

⏭

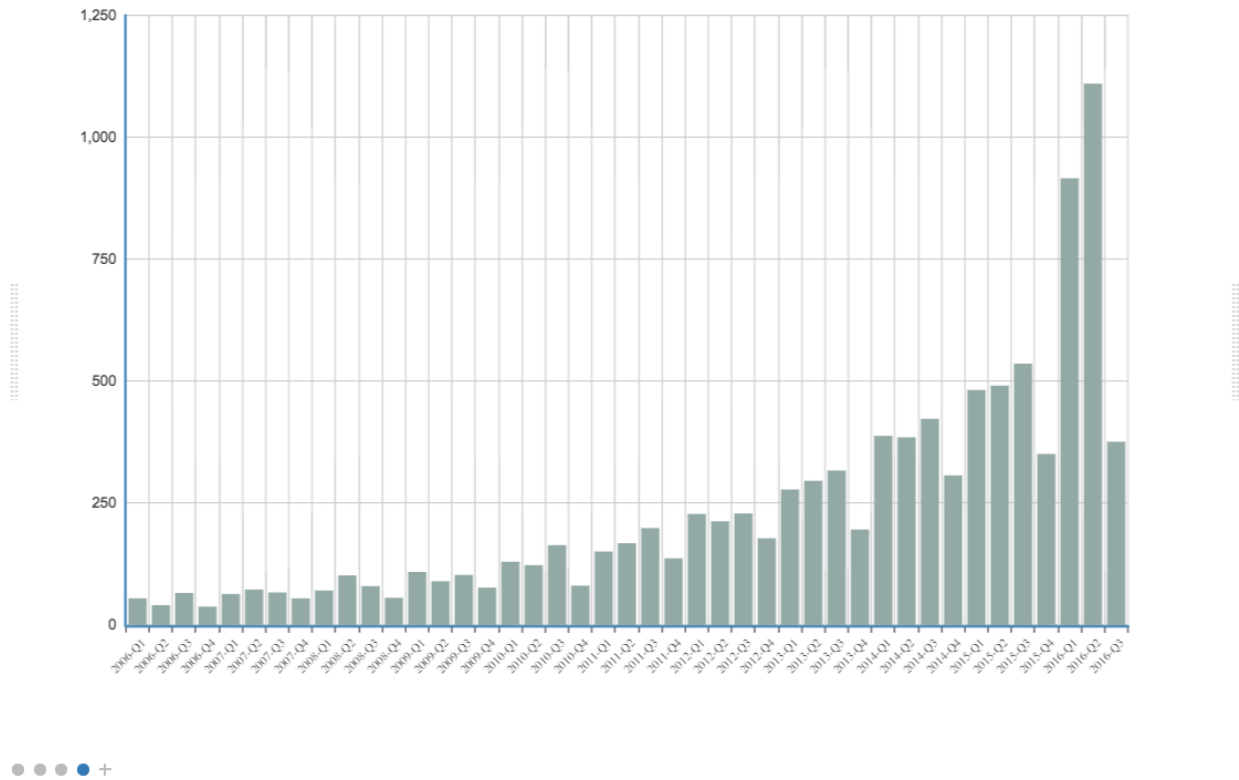
Per page: 10



When these results are placed into a bar chart it would look similar to this:



::



4.2 Section 2 - Complex Queries

This section goes into more advanced queries that include documents with nested data, documents that utilize schema as data, and multicollection JOINS.

The following examples assume a document structure similar to the following:

NOTE: this is fictitious sample data, randomly generated

```
{
  "_id": ObjectId("5781ae797689630b25452c73"),
  "city": "COLONIA",
  "first_name": "Keesha",
  "last_name": "Odonnell",
  "middle_name": "Alice",
  "last_visit": ISODate("2016-01-01T15:56:36Z"),
  "weight": 145,
  "loc": [
    -74.314688,
    40.590853
  ],
  "gender": "female",
  "age": 98,
  "previous_visits": [
    ISODate("2009-02-14T15:09:30Z"),
    ISODate("2006-02-23T17:45:05Z")
  ],
}
```

```
"height": 61,
"county": "MIDDLESEX",
"state": "NJ",
"ssn": "383-97-3804",
"previous_addresses": [
  {
    "city": "HUDSON",
    "longitude": -108.582745,
    "county": "FREMONT",
    "state": "WY",
    "latitude": 42.900791,
    "zip_code": 82515
  },
  {
    "city": "SMYRNA",
    "longitude": -75.565131,
    "county": "KENT",
    "state": "DE",
    "latitude": 39.194026,
    "zip_code": 19977
  },
  {
    "city": "ZOAR",
    "longitude": -81.414245,
    "county": "TUSCARAWAS",
    "state": "OH",
    "latitude": 40.61829,
    "zip_code": 44697
  }
],
"codes": [
  {
    "code": "S72.001C",
    "desc": "Displaced fracture of medial malleolus of right tibia, subsequent encounter for open fracture of foot"
  },
  {
    "code": "S72.009E",
    "desc": "Other yatapoxvirus infections"
  },
  {
    "code": "S56.417D",
    "desc": "Other fracture of shaft of radius, left arm, subsequent encounter for closed fracture of forearm"
  },
  {
    "code": "B55.2",
    "desc": "Varicose veins of right lower extremity with ulcer of thigh"
  }
],
"street_address": "8320 45TH ST",
"zip_code": 7067
}
```

4.2.1 1.2 Nested Data

SlamData provides the flattening operator (`[*]`) to iterate through arrays and extract values from fields.

1.2.1 Return Nested Array

Querying documents with arrays without the ([*]) operator results in an array being returned, see the following SQL² and resulting image. Compare this to section 1.2.2 Return Flattened Array.

SQL Example

```
SELECT
  last_name || ", " || first_name AS NAME,
  age AS PATIENT_AGE,
  codes AS Z_CODES
FROM `/devguide/devdb/patients`
```

Example Output

NAME	PATIENT_AGE	Z_CODES	
		code	desc
Shepherd,Patrick	63		
Bishop,Dean	59	M89.06Z	Pre-existing hypertensive heart and chronic kidney disease complicating the puerperium
		S42.123K	Displaced lateral mass fracture of first cervical vertebra, subsequent encounter for fracture with nonunion
Mays,Vicente	62	C23	Unspecified open wound of right shoulder, subsequent encounter
		V48.4xxD	Nodular lymphocyte predominant Hodgkin lymphoma, lymph nodes of axilla and upper limb
Clay,Virgilio	41		
Schwartz,Michael	54	H33.031	Other disorders of nervous system
		M84.574K	Chorioamnionitis, third trimester, not applicable or unspecified
		S82.015J	Scleritis with corneal involvement, left eye
		S78.121D	Open bite of trachea, sequela
Oconnell,Rosie	23		
Carson,Marianna	98	T83.59xD	Nondisplaced fracture of anterior wall of unspecified acetabulum, initial encounter for closed fracture
Bruce,Celestina	61	T36.5x5A	Unequal limb length (acquired), left tibia
Moreno,Ernesto	26	H44.613	Pregnancy related conditions, unspecified, second trimester
		S63.015S	Benign paroxysmal vertigo, unspecified ear
Macdonald,Lorenza	39	S82.023J	Maternal care for (suspected) central nervous system malformation in fetus, fetus 5
		K50.819	Displaced fracture of fourth metatarsal bone, unspecified foot, initial encounter for open fracture

Page 3 of 1000 Per page: 10

MongoDB query equivalent

```
db.patients.aggregate(
[
  { "$limit": NumberLong("11") },
  {
    "$project": {
      "NAME": {
        "$cond": [
          {
            "$and": [
              { "$lte": [{ "$literal": "" }, "$first_name"] },
              { "$lt": ["$first_name", { "$literal": { } }] } ]
            },
          {
            "$cond": [
              {
```

```

        "$and": [
          { "$lte": [{ "$literal": "" }, "$last_name"] },
          { "$lt": ["$last_name", { "$literal": { } }]} ]
        },
        {
          "$concat": [
            { "$concat": ["$last_name", { "$literal": "," } ] },
            "$first_name"
          ],
          { "$literal": undefined }
        },
        { "$literal": undefined }
      ],
      "PATIENT_AGE": "$age",
      "Z_CODES": "$codes"
    }
  ]],
  { "allowDiskUse": true });

```

1.2.2 Return Flattened Array

Compare the output of this section to section 1.2.1 Return Nested Array. The difference is that in this example there is one row per patient, per diagnosis.

SQL Example

```

SELECT
  last_name || "," || first_name AS NAME,
  age AS PATIENT_AGE,
  codes[*] AS Z_CODES
FROM `~/devguide/devdb/patients`

```

Example Output

NAME	PATIENT_AGE	Z_CODES	
		code	desc
Odonnell,Keesha	98	S72.009E	Other yatapoxvirus infections
Odonnell,Keesha	98	S56.417D	Other fracture of shaft of radius, left arm, subsequent encounter for closed fracture with routine healing
Odonnell,Keesha	98	B55.2	Varicose veins of right lower extremity with ulcer of thigh
Bishop,Dean	59	M89.062	Pre-existing hypertensive heart and chronic kidney disease complicating the puerperium
Bishop,Dean	59	S42.123K	Displaced lateral mass fracture of first cervical vertebra, subsequent encounter for fracture with nonunion
Mays,Vicente	62	C23	Unspecified open wound of right shoulder, subsequent encounter
Mays,Vicente	62	V48.4xxD	Nodular lymphocyte predominant Hodgkin lymphoma, lymph nodes of axilla and upper limb
Schwartz,Michael	54	H33.031	Other disorders of nervous system
Schwartz,Michael	54	M84.574K	Chorioamnionitis, third trimester, not applicable or unspecified
Schwartz,Michael	54	S82.015J	Scleritis with corneal involvement, left eye

Page 5 of 2157 Per page: 10

MongoDB query equivalent

Notice the inclusion of the *\$unwind* MongoDB operator in the generated code below now:


```

db.patients.aggregate(
[
  {
    "$project": {
      "__tmp8": {
        "$cond": [
          {
            "$and": [
              { "$lte": [{ "$literal": [] }, "$codes" ] },
              { "$lt": ["$codes", { "$literal": BinData(0, "") }] } ]
            },
            "$codes",
            { "$literal": [undefined] } ]
          },
          "__tmp9": "$$ROOT"
        }
      },
      { "$unwind": "$__tmp8" },
      { "$limit": NumberLong("11") },
      {
        "$project": {
          "NAME": {
            "$cond": [
              {
                "$and": [
                  { "$lte": [{ "$literal": "" }, "$__tmp9.first_name" ] },
                  { "$lt": ["$__tmp9.first_name", { "$literal": { } }] } ]
                },
              {
                "$cond": [
                  {
                    "$and": [
                      { "$lte": [{ "$literal": "" }, "$__tmp9.last_name" ] },
                      { "$lt": ["$__tmp9.last_name", { "$literal": { } }] } ]
                    },
                    {
                      "$concat": [
                        { "$concat": ["$__tmp9.last_name", { "$literal": "," } ] },
                        "$__tmp9.first_name"
                      ],
                      { "$literal": undefined } ]
                    },
                    { "$literal": undefined } ]
                  },
                  { "$literal": undefined } ]
                },
                "PATIENT_AGE": "$__tmp9.age",
                "Z_CODES": "$__tmp8"
              }
            },
            {
              "$project": { "NAME": true, "PATIENT_AGE": true, "Z_CODES": true, "_id": false }
            }
          ],
          { "allowDiskUse": true } );

```



Fig. 4.1: SlamData Logo

Reference - SQL²

5.1 Section 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.

5.1.1 1.1 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	???

5.1.2 1.2 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, ARBITRARY
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
DateTime	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

5.2 Section 2 - Basic Selection

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

5.2.1 2.1 Select all values from a path

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

Example:

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

5.2.2 2.2 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`
```

5.2.3 2.3 Path Aliases

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
```

5.3 Section 3 - 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

5.3.1 3.1 Filtering using a numeric value

Example:

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

5.3.2 3.2 Filtering using a string value

Example:

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

5.3.3 3.3 Filtering using multiple Boolean predicates

Example:

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

5.4 Section 4 - 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
	Concatenates
LOWER	Converts to lowercase
UPPER	Converts to uppercase
SUBSTRING	Returns a substring
LENGTH	Returns length of string

5.4.1 4.2 - 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+"
```

5.5 Section 5 - 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.

5.5.1 5.1 Filter based on a timestamp

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")
```

5.5.2 5.2 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")
```

5.5.3 5.3 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")
```

5.5.4 5.4 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
```

5.5.5 5.5 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)
```

5.6 Section 6 - Grouping

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

5.6.1 6.1 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
```

5.6.2 6.2 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
```

5.6.3 6.3 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)
```

5.6.4 6.4 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"
```

5.6.5 6.5 Filter with Arbitrary Value

`ARBITRARY` returns an arbitrary value from a set. Each target datasource may implement this differently but is intended to retrieve a single value from a set in the cheapest way, and not necessarily deterministic.

5.6.6 6.6 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
```

5.7 Section 7 - 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.

5.7.1 7.1 Nesting

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

Example:

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

5.7.2 7.2 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
```

7.2.1 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
```

7.2.2 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"
```

5.8 Section 8 - Pagination and Sorting

5.8.1 8.1 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
```

5.8.2 8.2 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
```

5.9 Section 9 - Joining Collections

Use the JOIN operator to join two or more collections.

There is no technical limitation to the number of collections or tables that can be joined, but users are encouraged to consider the performance impact based on the dataset sizes.

For MongoDB JOIN s, see the database specific notes section about JOINS on MongoDB.

5.9.1 9.1 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)

```

5.9.2 9.2 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`).

5.10 Section 10 - Conditionals and Nulls

5.10.1 10.1 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

```

5.10.2 10.2 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
```

5.11 Section 11 - Data Type Conversion

5.11.1 11.1 Converting to Boolean

SQL² allows String data type fields with values of either "true" or "false" to be converted to their corresponding Boolean value.

Prefix the field name with the BOOLEAN function.

Example:

```
SELECT BOOLEAN(survey_complete) AS Survey FROM `/users`
```

5.11.2 11.2 Converting to Strings

SQL² allows most fields to be converted to String data types by prefixing the field name with the TO_STRING function.

Example:

```
SELECT TO_STRING(zip_code) AS ZipCode FROM `/users`
```

5.11.3 11.3 Converting to Integer

SQL² allows string representations of valid integer values to be converted to an actual integer number. Prefix the field name with the INTEGER function.

If a field named myField had the value of "1234" as a String, it could be converted to an integer with this example:

```
SELECT INTEGER(myField) AS MyField FROM `/users`
```

If a field is not a valid string representation of an integer value then a null value will be returned.

5.11.4 11.4 Converting to Decimal

SQL² allows string representations of valid integer and decimal values to be converted to an actual decimal number. Prefix the field name with the DECIMAL function.

If a field named myField had the value of "1.234" as a String, it could be converted to a decimal with this example:

```
SELECT DECIMAL(myField) AS MyField FROM `/users`
```

If the field does not contain a valid string representation of a numeric value, such as "123" or "123.456" then a null value will be returned.

5.11.5 11.5 Converting to Dates and Times

SQL² allows strings in a specific format to be converted to date and time related data types. See Section 5 for examples of converting to date, time, and timestamp types.

5.12 Section 12 - Variables and SQL²

SQL² has the ability to use variables in queries in addition to statically typed content. Variables can be generated through the use of a Variables Card or through a combination of Setup Markdown Card / Show Markdown Card. Both scenarios require that the variables be defined before the Query Card is executed.

Attention: SlamData Version

The syntax for using variables within SQL² was changed slightly in version 3.0.8. This document assumes you are using a version no older than 3.0.8.

5.12.1 12.1 Single Values

Single values are generated in Markdown through the following elements:

- String text field
- Numeric text field
- Calendar Picker
- Calendar / Time Picker
- Radio Boxes
- Drop Downs

For more information on Markdown / Slamdown and how to generate form elements see the Form Elements Section of the Slamdown Reference Guide.

Variables can be used in queries by prefixing the variable name with a colon (:).

For example, if the following Markdown code was used:

```
### Select year to report on
year = {2011,2012,2013,2014,2015,2016}
```

The value selected by the user from the `year` dropdown can be referenced like this:

```
SELECT * FROM `/users`
WHERE last_visit = :year
```

5.12.2 12.2 Multiple Values

Multiple values are generated in Markdown only through the Check Boxes UI element.

For example, if the following Markdown code was used:

```
### Select years to report on
years = [x] 2014 [] 2015 []2016 []2017
```

The values selected by the user from the `years` set of Check Boxes should be referenced with the `[]` option as well as the `IN` clause:

```
SELECT * FROM `/users`
WHERE last_visit IN :years[[]]
```

This example would find all users who have a `last_visit` that matched one of the check boxes selected.

5.13 Section 13 - Database Specific Notes

5.13.1 13.1 MongoDB

13.1.1 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")
```

13.1.2 JOINS on MongoDB

When executing a `JOIN` in `SQL2` against MongoDB, the analytics engine will decide whether to use the mapreduce API, or the aggregation API along with the `$lookup` operator. This operator was introduced in MongoDB version 3.2 and allows the equivalent of a left outer equijoin. You can find out more [here](#).

To leverage the `$lookup` operator, the query must satisfy the following conditions that are imposed by MongoDB:

- Must be running MongoDB 3.2 or newer
- One collection must use an indexed field
- That collection must not be sharded
- Both collections must be in the same database
- Match must be an equijoin, based on equality only (`a.field = b.field` is ok, `a.field < b.field` is not)

If `$lookup` cannot be used, SlamData will fall back to utilizing the mapreduce API. Utilizing mapreduce is slower but more flexible and is also backwards compatible for MongoDB 2.6 and newer.



Reference - SlamDown

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

6.1 Section 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*.

6.2 Section 2 - Block Elements

The following SlamDown elements create blocks of content.

6.2.1 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 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

6.2.3 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.

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

Both Indented Code Blocks and Fenced Code Blocks result in:

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

6.2.4 2.4 Paragraphs

Paragraphs are separated by a blank line.

```
This is paragraph 1.

This is paragraph 2.
```

results in:

This is paragraph 1.

This is paragraph 2.

6.2.5 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 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 Section 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 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 3.2 Links

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

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

results in:

[SlamData](#)

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 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 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`

6.4 Section 4 - Evaluated SQL² Queries

SlamDown extends Markdown by allowing you to evaluate a SQL² query and insert the results into the rendered content, including the form elements listed in Section 5 below. Start the query with an exclamation point and then contain the SQL² query between double backtick (‘ ‘) characters.

Hint: Backticks

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.

SQL² queries are always surrounded by double backticks (‘ ‘) and preceded with an exclamation point (!). Additionally, they may be surrounded by parentheses (()) for radio buttons, braces ({ }) for dropdowns, and brackets ([]) for check boxes as seen in later sections.

6.5 Section 5 - Form Elements

Provide interactive forms for your users with text fields, date pickers, check boxes and more.

First define a variable name in Slamdown and then define the element type based on the formatting in the sections below.

For instance:

```
name = _____
```

This defines the variable `name` and creates a simple text entry field in the browser. You can then utilize this variable in a Query Card like this:

```
SELECT address, phone_number, city, state
FROM `/mydb/mytable`
WHERE fullname = :name
```

Make sure to precede the variable name with a colon (:) when referencing it as a variable inside of a Query Card.

6.5.1 5.1 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 user's interests. You can then refer to the value as `:interests`

```
interests = _____
```

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 `interests` with a default value of "SlamData". You can then refer to the value as `:interests`

```
interests = _____ (SlamData)
```

6.5.2 5.2 Numeric Field

By default input fields are evaluated as String types. To enforce a numeric type prefix the underscores with the (#) symbol. You may also provided a default value for this field as well. For example:

```
year = #_____ (1999)
```

6.5.3 5.3 Radio Buttons

A set of radio buttons has only one button selected at a time. Radio buttons can be populated with static content or populated with a query. See the follow sections.

5.3.1 Static Radio Buttons

Use parentheses followed by text to indicate radio buttons. 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 = () car (x) bus () bike
```

This results in:



Notice how the default selection became the first selection in the actual rendered set.

5.3.2 Dynamic Radio Buttons

As with all other form elements, radio buttons may be populated by means of an evaluated SQL² query.

For example, this Slamdown code creates a set of radio buttons that list the unique color values in a database:

```
mycolor =  
(!`SELECT DISTINCT(color) FROM `/devguide/devdb/colors` ORDER BY color ASC LIMIT 1`)  
!`SELECT DISTINCT(color) FROM `/devguide/devdb/colors` ORDER BY color ASC`
```

First notice how the field is defined on multiple lines.

Next you can see two queries now instead of one. The first query defines which value is selected by default, the second defines the remaining values. This results in:



6.5.4 5.4 Checkboxes

Use brackets ([]) followed by text to indicate checkboxes. In a set of checkboxes each checkbox operates independently.

Hint: Array Evaluation

When referring to a variable that is an array, which is what a checkbox variable is, the variable must be followed by the [_] operator. See query example below.

A checkbox array variable can be used in a query whether it was defined statically in Slamdown or dynamically through an evaluated SQL² query. An example query within a Query Card would look like this.

```
SELECT * FROM `/mydb/mytable` WHERE phone IN :phones[_]
```

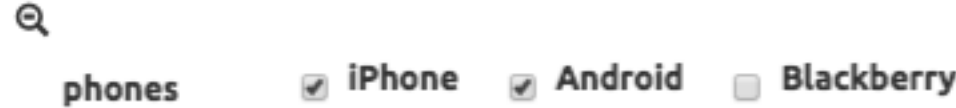
5.4.1 Static Check Boxes

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

This results in:



Similar to radio buttons, notice that the fields preselected with an `x` as are rendered first.

The selections above would result in the `phones` variable containing a value of the following array: `["iPhone", "Blackberry"]`

5.4.2 Dynamic Check Boxes

As with all other form elements, checkboxes may be populated by means of an evaluated SQL² query.

For example, this Slamdown code creates a set of checkboxes that list the phone types within a database:

```
myphone =
[!`SELECT DISTINCT(phone) FROM `/mydb/mytable` ORDER BY phone ASC LIMIT 1`]
!`SELECT DISTINCT(phone) FROM `/mydb/mytable` ORDER BY phone ASC`
```

This results in:



The first query defines which value is selected by default, the second query populates the remaining checkboxes.

6.5.5 5.5 Dropdowns

Dropdowns allow users to select one (and only one) value from a list of options, similar to radio buttons. Unlike radio buttons, however, dropdown elements typically take up less space in the browser and are more suitable to longer lists of values.

Use a comma-separated list in braces (`{ }`) to indicate a dropdown element.

Hint: Array Evaluation

When referring to a variable that is an array, which is what a dropdown variable is, the variable must be followed by the `[_]` operator. See query example below.

A dropdown array variable can be used in a query whether it was defined statically in Slamdown or dynamically through an evaluated SQL² query. An example query within a Query Card would look like this.

```
SELECT * FROM `/mydb/mytable` WHERE city IN :mycity[_]
```

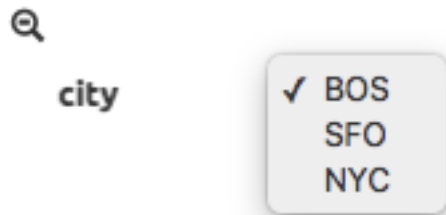
5.5.1 Static Dropdown

Define a static dropdown element by placing the values of array elements within braces (`{ }`).

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

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

This results in:



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)
```

5.5.2 Dynamic Dropdown

As with all other form elements, dropdown elements may be populated by means of an evaluated SQL² query.

For example, this Slamdown code creates a dropdown that contains the names of cities within a database:

```
mycity = {!`SELECT DISTINCT(city) FROM `/mydb/mytable` ORDER BY city ASC`}
```

6.5.6 5.6 Dates and Times

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

5.6.1 Date

The following example creates a date selector element and stores the value in a variable called `start`:

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

This results in:



start

04/19/2016 ✕ ⬆ ⬇ ⬇ ⬆

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

5.6.2 Time

The following lines creates a time selector element:

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

This results in:



start


02:30 PM ✕ ⬆ ⬇ ⬇ ⬆



5.6.3 Date & Time (TIMESTAMP)




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

```
start = ____-__-__ __:__ (2016-04-19 14:00)
```

This results in:

 **start**

04/19/2016, 02:00 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

6.6 Section 6 - Slamdown Variables in Queries

SlamData has the ability to use values selected in Slamdown form elements to be used in a query. For more information and examples, see Section 11 of the SQL² Reference Guide.



Troubleshooting FAQ

7.1 Section 1 - Configuration

7.1.1 1.1 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.

1.1.1 Configuration File Differences

SlamData Community Edition fully relies on the quasar-config.json configuration file to store all data for the product, including server configuration, mountings, views, and more.

SlamData Advanced Edition instead relies on a PostgreSQL or Java H2 database to store the same metadata. Additionally it stores security information for users, groups, permission, actions and token.

If there is no metadata source when SlamData Advanced Edition starts it will instead use the quasar-config.json file.

7.1.2 1.2 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
```

7.2 Section 2 - Running SlamData

7.2.1 2.1 SlamData Won't Start

Follow the steps below to ensure all known issues have been addressed.

1. Some versions of SlamData require more than one CPU core before launching. This is a known bug in some older versions. Try increasing the number of cores in your virtual machine and restarting.
2. In older versions of SlamData an invalid database mount may cause SlamData not to start. An invalid database mount could be a database that was previously available but no longer is, credentials may have changed, port number change, or any other configuration change which does not allow previously validated configurations to successfully connect.

7.2.2 2.2 Accessing SlamData

The default SlamData URL is `http://servername:20223/slamdata/`

7.2.3 2.3 How do I see which version I'm running?

SlamData's version should be displayed in the browser title bar or tab title.

The version of the Quasar analytics backend engine can be obtained by browsing to `http://servername:20223/server/info`

7.2.4 2.4 Running SlamData in the Cloud

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.

7.3 Section 3 - Performance

7.3.1 3.1 General Considerations

7.3.2 3.2 Advanced Edition Considerations

Indices and tables

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