
SlamData Documentation

Release 4.2

SlamData

Aug 02, 2017

Contents

1	User's Guide	3
1.1	Section 1 - Introduction	3
1.2	Section 2 - Quick Start	4
1.3	Section 3 - The Workspace	4
1.4	Section 4 - Cards	8
2	Administrator's Guide	21
2.1	Section 1 - Installation	21
2.2	Section 2 - Connecting to a Data Source	24
2.3	Section 3 - Configuring SlamData	32
2.4	Section 4 - SlamData User Security	35
2.5	Section 5 - Security APIs	40
3	Developer's 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
4	Helpful Tips	71
4.1	Section 1 - Basic Queries	71
4.2	Section 2 - Complex Queries	79
5	Reference - SQL²	85
5.1	Section 1 - Introduction	85
5.2	Section 2 - Basic Selection	86
5.3	Section 3 - Filtering a Result Set	87
5.4	Section 4 - Numeric and String Operations	88
5.5	Section 5 - Dates and Times	88
5.6	Section 6 - Grouping	90
5.7	Section 7 - Nested Data and Arrays	91
5.8	Section 8 - Pagination and Sorting	92
5.9	Section 9 - Joining Collections	93
5.10	Section 10 - Conditionals and Nulls	94
5.11	Section 11 - Data Type Conversion	95
5.12	Section 12 - Variables and SQL ²	96
5.13	Section 13 - Database Specific Notes	97

6	Reference - SlamDown	99
6.1	Section 1 - Introduction	99
6.2	Section 2 - Block Elements	99
6.3	Section 3 - Inline Elements	102
6.4	Section 4 - Evaluated SQL ² Queries	103
6.5	Section 5 - Form Elements	103
6.6	Section 6 - Slamdown Variables in Queries	109
7	Troubleshooting FAQ	111
7.1	Section 1 - Configuration	111
7.2	Section 2 - Running SlamData	112
8	Indices and tables	117

Contents:



This User's Guide will assist the user who is unfamiliar with SlamData to understand the key product features and interface.

For information on how to use SlamData from an administrator's perspective see the SlamData Administrator's Guide.

For information on how to use SlamData from a developer's perspective see the SlamData Developer's Guide.

Section 1 - Introduction

1.1 Assumptions

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

- Has a basic to moderate understanding of JSON or semi-structured data.
- Has appropriate permissions to install the software.
- Has read **and** write access to a data source, such as a database system.

1.2 Requirements

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

1.3 Installation

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

Section 2 - Quick Start

The following two sections will take a new user from no 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. The remaining sections of the User's Guide contain detailed information on specific functionality.

2.1 Browsers

The most compatible browsers with SlamData are always the most recent versions of Google Chrome and Mozilla Firefox.

Microsoft Edge and Safari are both limited in functionality and some UI elements, such as Date picker, do not render properly, or at all.

Section 3 - The Workspace

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, and so on.

3.2 Mount Data Source

In this guide the MongoDB database will be used in the examples.

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

Mount type

Cancel

Mount



Enter the values below and the dialog will expand.

Parameter	Value
Name	myserver
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	


3.3 Creating a Database

- Click on the newly created server named **myserver**. The interface now shows the databases that reside within the database system. 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 **Untitled Folder**.
- Click the **Move / rename** icon that appears to the right. 
- Change the name from **Untitled Folder** to `testdb` and click **Rename**.
- Click on the newly renamed **testdb** folder.

3.4 Importing Example Data

This guide uses a data set of fictitious patient information that was randomly generated. The examples in the remaining sections will assume that the patients data set is being used.

A data set with 10,000 documents can be downloaded 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 `testdb`, and click the Upload icon. 
- In the file dialog find the patients file and submit it.

As you can see, it is easy to quickly import JSON data into SlamData. Other formats, such as CSV, can also be quickly imported.

You 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.

3.5 Exploring Sample Data

- Click on **patients** in the user interface.
- A dialog will appear asking the name of the new Workspace being created.
- Give the Workspace a new name and click **Explore**.
- You will be presented with a table showing the contents of the patients data.


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 you, created an **Open Card** pointing to the patients data, then stacked a **Preview Table Card** on top of the **Open Card**.


You can verify this by clicking on the left dots (grippers) 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 **Preview Table Card**.

Click on the browse arrows at the bottom to scroll through the pages of data.


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

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/testdb/patients`
```

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.

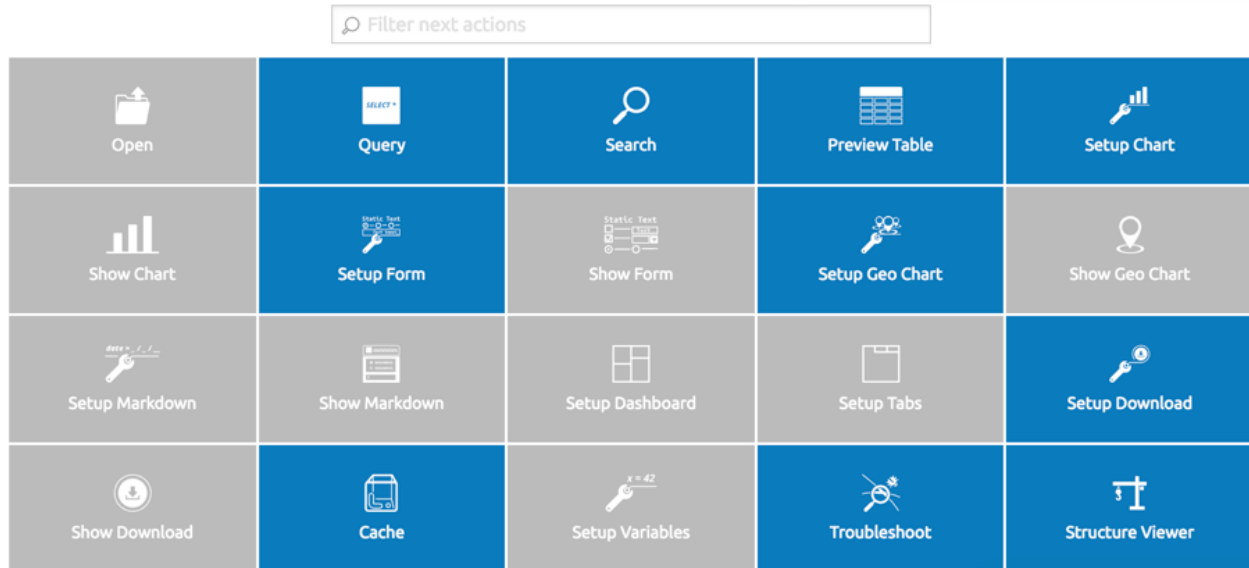
- Select **Run Query** in the bottom right.
- Click the right grip.
- Select the **Preview Table Card** to see the results.
- Click the Zoom Out  icon to return to the database view.
- Optionally rename the **Untitled Workspace** that was created for this workflow.

3.7 Searching Data

SlamData has several very powerful ways of finding the data you need. In the following example, you will use the **Search Card**.

- Select the Create Workspace icon. 
- Select **Open Card**.
- Locate the patients entry in your database and select it.
- Click and drag the right-hand grip and slide it to the left.

The following card types will be presented:



Notice how the cards are blue and gray. The blue cards are those that can be created directly after the **Open Card**. Gray cards are those cards that cannot be used following the previous card.

- 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.

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

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 the database system, and that indexes can significantly boost performance for searches.

Once the results appear, you can browse them just like you did earlier in the **Preview Table 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 data sources 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 either drag the right grip bar to the left, or simply click on the right grip bar.

- View the results in the **Preview Table Card** again.

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 **Preview Table Card** again.

As one last example let's see how we can mix and match different types. We want to know how many males over the age of 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.

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.

- Click the right gripper to stack a new card on top of the **Preview Table Card**.
- Select **Setup Download**.
- Select either `C;S;V` (CSV) or `{JS}` (JSON) format for the download.
- Click the right gripper to stack a new card on the deck.
- Select **Show Download**.
- Select the **Download button** to download the data.

You have now entered search criteria, browsed the results and downloaded the results in a CSV or JSON format.

Section 4 - Cards

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. The cards are described in alphabetical order.

4.2 - Cache Card



Description

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

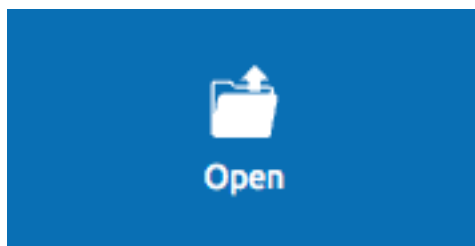
Behavior

The **Cache Card** requires a location to store its results. When a newly selected **Cache Card** becomes active, the user is presented with a text field and a **Confirm** button. The value of the text field can be edited directly to change the location of the cached information. The credentials provided to mount the original data source 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:

- Each time the workspace that writes to the cache location is opened
- The table or collection in the **Open Card** prior to the Cache Card is modified.
- The query in the **Query Card** prior to the Cache Card is modified.
- The search parameters in the **Search Card** prior to the Cache Card are modified.

4.3 - Open Card



Description

The **Open Card** can be used, for example, to specify a collection from which subsequent cards will operate from.

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 collection with the **Open Card**, the next card will have access to that collection as a whole.

Common scenarios for using the **Open Card** include following it with a **Search Card** or a **Preview Table Card**.

4.4 - Preview Table Card



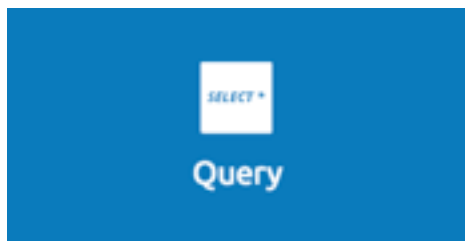
Description

The **Preview Table Card** provides a tabular view of data from a data source. It is particularly useful for data exploration and for presenting the results of a **Query Card** or a **Search Card**.

Behavior

When working with a data source, it is very useful to visualize data in a tabular format. The **Preview Table Card** provides a very convenient way to present data that is the result of a user action, such as a **Query Card**. Controls are available in the lower-left that allow the user to scroll through the result set.

4.5 - Query Card



Description

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

Behavior

If a **Query Card** follows a **Preview 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. This button is used after the query has been entered. If a query has not changed, the query will automatically execute within a workflow.

4.6 - Search Card



Description

The **Search Card** searches for entries from a data source. A data source can either be a specific collection or table designated by an **Open Card** or it can also be the result set from a **Query Card**.

Behavior

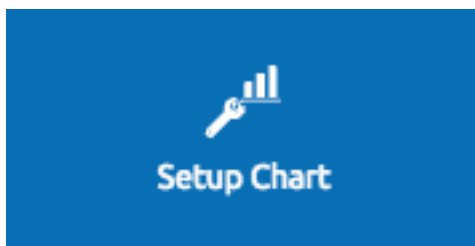
A **Search Card** is typically followed by a **Preview Table Card** to display the results of a search.

Values not preceded by a field name and colon, such as `fieldName:`, will cause the data source 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 data source using an indexed query (if an appropriate index exists), resulting in a faster response.

Search parameters are “AND”ed together, so the more parameters that are provided, the more selective the result will be. The following table shows some common search examples:

Example	Description
<code>foo, +foo</code>	Searches for the substring <code>foo</code> in all fields .
<code>-foo</code>	Searches for everything not containing the text <code>foo</code> .
<code>=foo</code>	Searches for the full word <code>foo</code> in all fields .
<code>foo:=50</code>	Searches the field <code>foo</code> for a value of 50.
<code>foo:>=50</code>	Searches the field <code>foo</code> for any value greater than or equal to 50.
<code>foo:50..60</code>	Searches the field <code>foo</code> for values inside the range 50 to 60, inclusive.
<code>foo:bar:baz</code>	Searches for everything that contains a <code>foo</code> field which contains a <code>bar</code> field which contains the text <code>baz</code> .
<code>foo:"[*]":bar:baz</code>	Performs a deep search through the <code>foo</code> array and examines each subdocument's <code>bar</code> field for the substring <code>baz</code> .

4.7 - Setup Chart Card



Description

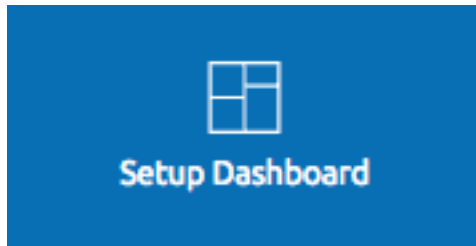
The **Setup Chart Card** is required before using the **Show Chart Card**. This card is used to specify the chart type and chart options of the subsequent **Show Chart Card**. Major chart types include the following:

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

Behavior

Each major chart type will have options that allow control over the look of the chart. For example, an **Area Chart** will provide the option to stack values.

4.8 - Setup Dashboard Card



Description

The **Setup Dashboard Card** may only be selected as the first card in the first deck inside of a workspace. Creating a **Setup Dashboard 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.

Behavior

Because the **Setup Dashboard Card** creates a workspace with no decks or cards, it must be the first card in the deck. Additionally, a user must now create a new deck inside of this Dashboard.

4.9 - Setup Download Card



Description

The **Setup Download Card** precedes the **Show Download Card**. The format of the download file can be configured to either CSV or JSON. Additionally, several other parameters can also be configured.

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 changed by the workspace author, but not by a user through a published or embedded workspace.

4.10 - Setup Form Card



Description

The **Setup Form Card** provides a graphical method to select fields to display from a data set.

Behavior

The **Setup Form Card** provides a wide-range of UI elements to choose from. After a UI element has been chosen, then the field to display is selected. An example workflow would be to select an **Open Card** and point it at a database collection, then follow it with a **Setup Form card**. The field in the **Setup Form Card** can subsequently be used in other cards, such as a **Query Card**. This provides an alternative to using the **Setup Markdown Card**, defining variables, and so on.

4.11 - Setup Geo Chart Card



Description

Behavior

4.12 - Setup Markdown Card



Description

The **Setup Markdown Card** allows a user to write the Markdown code that will be rendered within 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.

4.13 - Setup Tabs Card



Description

The **Setup Tabs Card** may only be selected as the first card in the first deck inside of a workspace. Creating a **Setup Tabs Card** is similar to creating a **Setup Dashboard** card, but instead of having multiple decks being shown on the same display, the decks are shown in separate tabs.

Behavior

Because the **Setup Tabs Card** creates a new workspace in each of the tabs created with no decks or cards in it. Additionally, the user must now create a new deck inside of the tab.

4.14 - Setup Variables Card



Description

The **Setup Variables Card** allows a user to create a workspace where the results are controlled by parameters that are programmatically passed into it.

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 a **Setup Variables Card** followed by 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 a user 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 programatically, changing the values of the `state` and `city` variables based upon custom logic.

Data Types

Text

An input field will appear when **Text** is chosen. Alphanumeric text may be entered.

Example: My 123 value here

DateTime

A date and time picker will appear when **DateTime** is chosen. Selecting a date and time will designate the default value.

Date

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

Time

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

Interval

An input field will appear when **Interval** is selected. Selecting an interval will designate the default value. Interval is defined using the ISO 8601 format.

Example: PT12H34M

In the above example, P is the duration, T is the time designator, 12H is 12 hours and 34M is 34 minutes.

Boolean

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

Numeric

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

Example: 1 or 1.5

Object ID

An input field will appear when **Object ID** 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 the following:

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

Array

An input field will appear when **Array** 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 variable `sensors` was defined in the **Setup Variables Card**, and the user wanted a query to return all records containing a `sensors` field that matched any entry from the array, the query could look similar to the following:

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

Object

An input field will appear when **Object** is chosen. Object is a JSON object.

Example: { "a": 1 }

SQL² Expression

An input field will appear when **SQL² Expression** is chosen. A valid SQL² Expression should be entered as the default.

Example:

```
SELECT *
FROM `/server/db/collection`
```

SQL² Identifier

An input field will appear when **SQL² Identifier** is chosen. A valid query path should be entered as the default. This allows a user 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 similar to the following:

```
SELECT *
FROM :mypath
```

4.15 - Show Chart Card



Description

The **Show Chart Card** follows the **Setup Chart Card**. Once the options have been selected in the **Setup Chart Card** and a chart is ready to be rendered, the **Show Chart Card** should be selected.

Behavior

The **Show Chart Card** renders the chart created using the **Setup Chart Card**.

4.16 - Show Download Card



Description

The **Show Download Card** follows the **Setup Download Card**.

Behavior

The **Show Download Card** provides a button to download data using the format and options selected using the **Setup Download Card**.

4.17 Show Form Card



Description

The **Show Form Card** follows the **Setup Form Card**.

Behavior

The **Show Form Card** displays the given form element that was chosen in the **Setup Form Card**.

4.18 Show Geo Chart Card



Description

The **Show Geo Chart Card** follows the **Setup Geo Chart Card**. Once the options have been selected in the **Setup Geo Chart Card** and a chart is ready to be rendered, you can select the **Show Geo Chart Card**.

Behavior

The **Show Geo Chart Card** renders the chart created using the **Setup Geo Chart Card**.

4.19 - Show Markdown Card



Description

The **Show Markdown Card** follows the **Setup Markdown Card**. Once the options have been selected in the **Setup Markdown Card** and the Markdown is ready to be rendered, the **Show Markdown Card** should be selected.

Behavior

The **Show Markdown Card** renders the Markdown created using the **Setup Markdown Card**.

4.20 - Structure Viewer Card



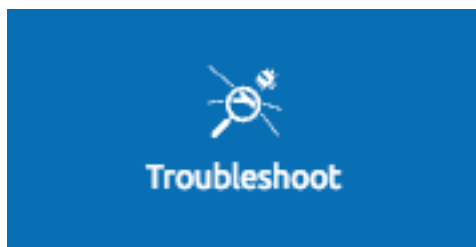
Description

The **Structure Viewer Card** will give you a quick overview of your data structure.

Behavior

The **Structure Viewer Card** can be put after any card that returns a data set. You can view the structure of the data that was passed in such as columns and their contents. The **Structure Viewer Card** will also show a grey bar in the column representing the percentage of documents that contain a value for that field. The larger the bar, the more documents that have a value.

4.21 - Troubleshoot Card



Description

The **Troubleshoot Card** is a useful tool to help find problem or issues in a Workspace.

Behavior

The **Troubleshoot Card** is helpful in validating values passed into a Workspace. For example, a **Setup Variables Card** followed by a **Troubleshoot Card** would enable variable values to be checked.



This Administrator's Guide describes how to install and configure SlamData.

For basic information on how to use SlamData please refer to the SlamData User's Guide.

For further information on how to use SlamData and instructions on how to integrate SlamData into other applications please refer to the SlamData Developer's Guide.

Section 1 - Installation

1.1 Minimum System Requirements

- **Minimum memory**
 - 2 GB memory
 - An additional 25 MB is required for each active user
- **Disk**
 - 300 MB for a basic installation
 - Additional space varies based upon Workspace size, cached queries, and so on
- **Java**
 - Java 1.8
 - Windows and Mac OS versions of SlamData with installers include Java
 - Linux requires a separate Java installation
- **Browsers**
 - The most compatible browsers with SlamData are always the most recent versions of Google Chrome and Mozilla Firefox

- Microsoft Edge and Safari are both limited in functionality and some UI elements, such as Date picker, do not render properly, or at all
- **Target data sources (for analytics)**
 - Apache Spark 2.1 and above
 - Couchbase 4.5.1 and above
 - MarkLogic 8 and above
 - MongoDB 2.6 and above

1.2 Obtaining SlamData

1.2.1 Obtaining a license

You will need a license to use SlamData Advanced. If you do not have a license or SlamData.com account please visit <https://slamdata.com/30-day-trial/> to obtain a trial license, <https://slamdata.com/slamdata-jump-start/> to purchase a trial with additional training and support or <https://slamdata.com/contact-us/> to get a quote for your SlamData Advanced License.

If you have lost your license key please visit <https://slamdata.com/my-account/>.

Updating your license information can be achieved by reinstalling.

1.2.2 Obtaining SlamData Advanced

There are two ways of using SlamData Advanced. If you want to try SlamData on your PC or Mac we recommend using the SlamData Advanced Launcher. If you want to use SlamData on your server(s) we recommend using the SlamData Advanced Jar.

1.2.3 SlamData Advanced Launcher

The SlamData Advanced Launcher is available for macOS, Windows and Linux. On Windows the SlamData Advanced Launcher allows you to launch SlamData Advanced from your Start Menu. On macOS the SlamData Advanced Launcher allows you to launch SlamData from your Applications folder or Launchpad.

To get started visit <https://slamdata.com/downloads/> and download the SlamData Advanced Installer, launch the installer and follow the instructions.

You will need to provide a license key or trial license key during installation. If you have lost your license key please visit <https://slamdata.com/my-account/>.

Updating your license information can be achieved by reinstalling.

1.2.3.1 SlamData Advanced Launcher Default Authentication

By default the SlamData Advanced Launcher is configured to authenticate with SlamData.com. You will need a SlamData.com account and access to the internet to use the SlamData Advanced Launcher in its default configuration.

If you signed up for a trial License at <https://slamdata.com/30-day-trial/> your SlamData.com account details are the username and password you provided.

1.2.4 SlamData Advanced Jar

To get started visit <https://slamdata.com/downloads/> and download the SlamData Advanced Jar Archive.

Next unzip the archive and navigate to the SlamData directory using the following commands.

```
tar jxf slamdata-advanced.tar.bz2
cd slamdata
```

Next save the following configuration file as `config.json`.

```
{
  "server": {
    "port": 20223
  },
  "authentication": {
    "openid_providers": [
      {
        "client_id": "RFQmEeS0Vw8UWUchQio5tQczsKIqpL",
        "display_name": "SlamData",
        "openid_configuration": {
          "issuer": "https://slamdata.com",
          "authorization_endpoint": "https://slamdata.com/oauth/authorize",
          "token_endpoint": "https://slamdata.com/oauth/token",
          "userinfo_endpoint": "https://slamdata.com/oauth/me",
          "jwks": [
            {
              "kty": "RSA",
              "alg": "RS256",
              "use": "sig",
              "n": "seduM0gTPqJWT57IFe0_QokLM-
↪fTuhp3lF8zd7AoOyP6yVsNJeEUf91YeuGxOIa3AZRQRX4SaiGfrv57JA8HEHLOIXBx680QjYGau9urKBFOeNNrWxAVy65CxbnM-
↪VKDajVZD0E-_1QQO9XKDix9Vlcmc5k6Ejx97tccMLhqYi6vhjglcgSGeNpM-
↪40K6WL3Y7qlpmEEPLkEkCCNJoEg7D5Xjxfi9a5xaUHRhVo8lpiKi5m9-
↪7uJaN4SzCqoYylwJT9agPzCaeWNT0tUYuo9ZCH_ev7NxYzzXTS08NXo_BBXypZ40Iw",
              "e": "AQAB"
            }
          ]
        }
      }
    ]
  }
}
```

If you would like SlamData to use a different port, additional or different authentication providers, different storage for metadata or enable auditing or HTTPS please edit the above configuration file using [Section 3 - Configuring SlamData](#) as a reference.

Finally start SlamData Advanced using the following command.

Please replace the details in the arguments starting with `-D` with your license information.

Please replace the numbers in the arguments starting with `-X` with the number of GB of memory you would like to allocate to SlamData.


```
java -Xms2G -Xmx2G -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 -jar quasar.jar --
↪content-path public --config config.json
```

Section 2 - Connecting to a Data Source

Connecting to a data source is the first step to analyzing data.

2.1 Data Sources

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

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

A mount dialog will be presented, as shown below.



The Mount dialog box is a light gray window with a title bar. It contains two labels, 'Name' and 'Mount type', each followed by a text input field. The 'Name' field is wider than the 'Mount type' field. At the bottom right of the dialog are two buttons: a gray 'Cancel' button and a blue 'Mount' button.

Enter a name for the data source mount. This name is used in the SlamData User Interface (UI) as well as SQL² query paths.

Hint: Mount Name

Use a name that makes sense for the environment. For example, if a data source were hosted on Amazon AWS/EC2 it might be named `aws` or `aws-1`.

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

2.2 Mount Options

The mount dialog will display the appropriate fields based upon the mount type selected. For each data source that SlamData supports, a section below describes the options available.

2.2.1 MongoDB

Select **MongoDB** as the mount type. Once the mount type has been selected, additional fields will appear in the dialog. The following table shows an example MongoDB server running on localhost with connection available on port 27017. No authentication is required in this case.

Parameter	Value
Host	localhost
Port	27017
Username	
Password	
Database	
Other Settings	

Note: Using Authentication

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. For example, it could be `admin`, the name of the user or something completely different.

The MongoDB values listed in the Connection Options on the MongoDB web site are supported. As of MongoDB 2.6 these options are as follows.

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.

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.

2.2.2 Couchbase

Select **Couchbase** as the mount type. Once the mount type has been selected, additional fields will appear in the dialog.

The following table shows an example Couchbase server running on localhost with connection available on port 8091.

Parameter	Value
Host	localhost
Port	8091
Username	Administrator
Password	*****

Note: To use SlamData with Couchbase, a Username and Password will be required. In the example table above, the Administrator account and password are used. The Administrator account is created when Couchbase is installed.

Hint: Memory Optimized Indexes

In the initial configuration of Couchbase, when it is being installed, memory optimized indexes should be enabled.

If the Couchbase default bucket is used with SlamData, it is necessary to create a primary index as well as an index on the `type` field. For example:

```
CREATE PRIMARY INDEX ON default;  
CREATE INDEX default_type_idx ON `default` (type);
```

2.2.3 MarkLogic

Select **MarkLogic** as the mount type. Once the mount type has been selected, additional fields will appear in the dialog.

The following table shows an example MarkLogic server running on localhost with connection available on port 8000.

Parameter	Value
Host	localhost
Port	8000
Username	Administrator
Password	*****
Database	/Documents

Note: To use SlamData with MarkLogic, a Username and Password will be required. In the example table above, the Administrator account and password are used. The Administrator account is created when MarkLogic is installed.

Hint: Directories

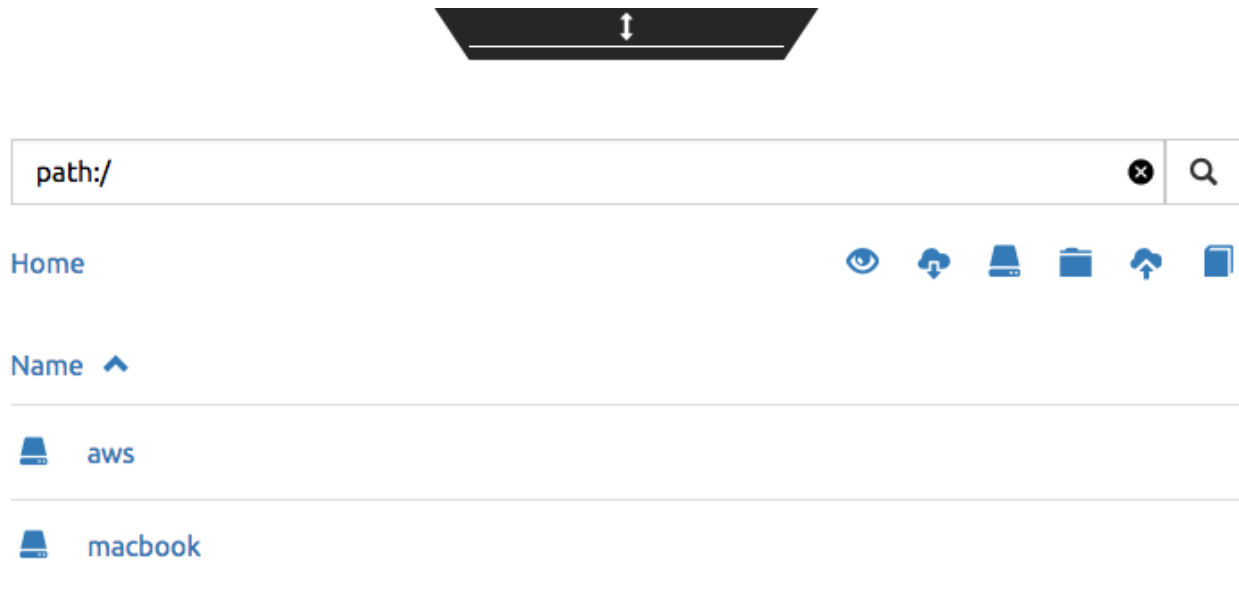
MarkLogic must contain one or more directories in the database before documents will be displayed. Additionally, documents must be located within a directory.

2.3 Several Mounts

After mounting several data sources, 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 data source.

2.4 SQL² View

SQL² Views are covered in detail in the SlamData Developer's Guide.



2.5 Enabling SSL for MongoDB

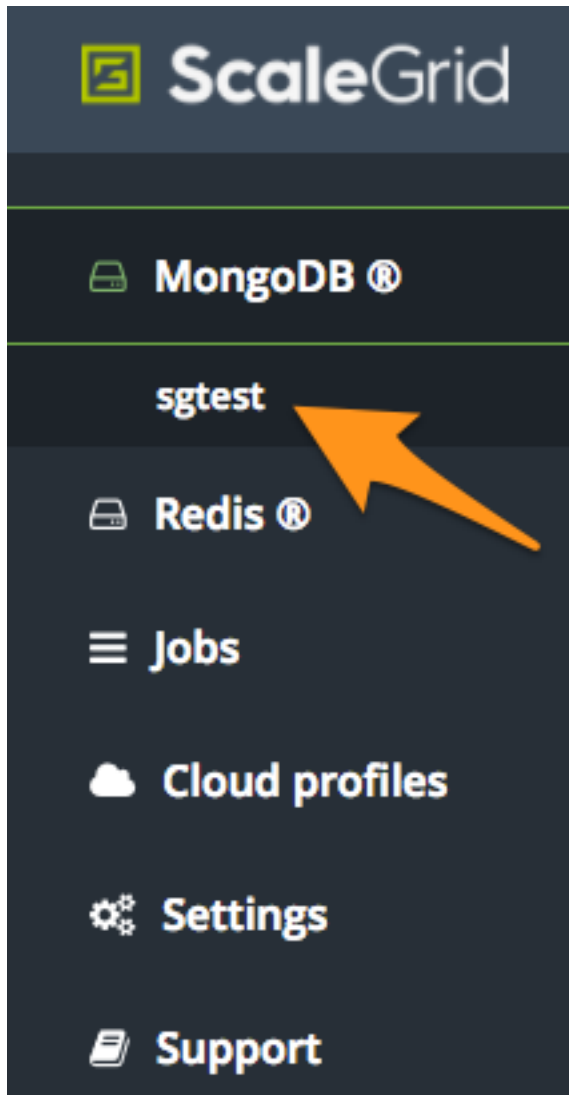
If a data source connection supports SSL encryption then additional configuration will be required.

This section does not provide exhaustive steps to create a Java Key Store in every scenario, but the following simple example should be helpful. It assumes the user is configuring SlamData to connect to MongoDB over SSL with an external service provider.

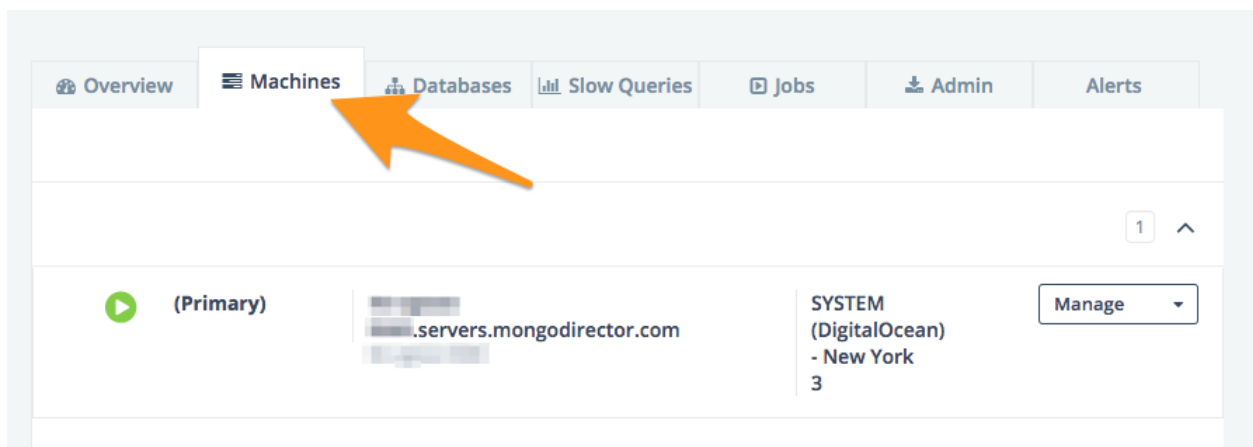
Let's consider a data source hosted with a service provider such as [ScaleGrid.io](#).

To make the following steps easier, you may want to obtain the available PEM file to your server for connecting via ssh. Specifically for ScaleGrid.io follow these steps:

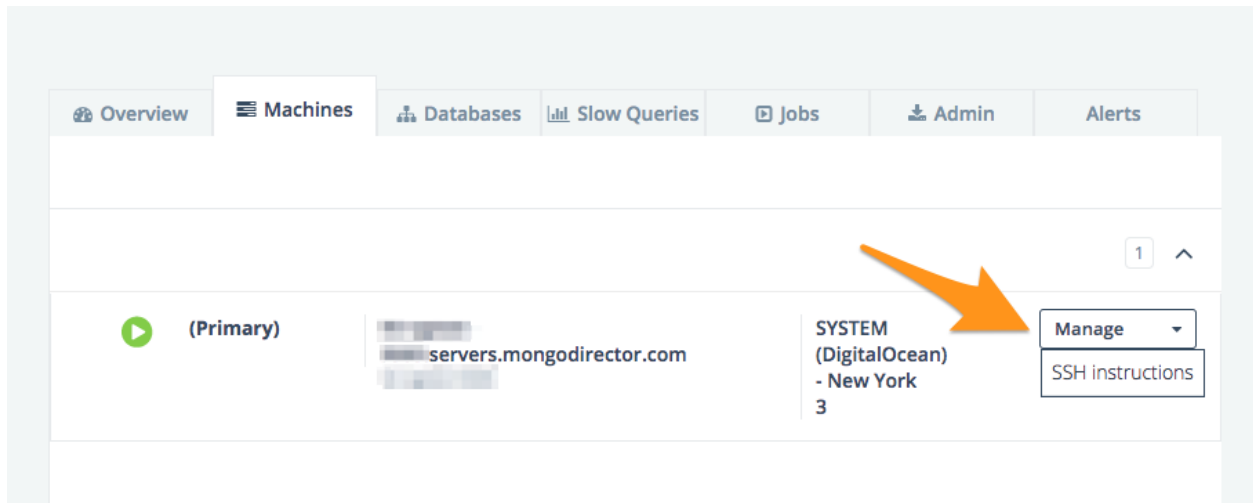
1. Click on the appropriate cluster in the left column menu.



2. Click on the Machines tab



3. Click on the Manage drop-down and select *SSH instructions*



4. Click the PEM File link. Copy and paste the contents into a text file such as `scalegrid_os.pem`

SSH instructions

1. Download your [PEM File](#).

2. Change the permissions of your pem file

```
chmod 400 <path to .pem>
```

3. SSH to your instance

```
ssh -i <path to .pem> root@[redacted].servers.mongodirector.com
```

[Back](#)

5. Verify connectivity by following steps 2 and 3 from that dialog.

Once you have verified connectivity, copying the MongoDB SSL files will be easier in the steps below.

Let's create a working directory on our local system so we keep track of our changes and to compartmentalize our changes.

```
mkdir ssl_config
cp scalegrid_os.pem ssl_config/
cd ssl_config
```

The service provider will make several files available. These files are needed to convert and import, so copy them over from the service provider's MongoDB system. If `scp` is installed locally, it can be used to simplify the transfer:

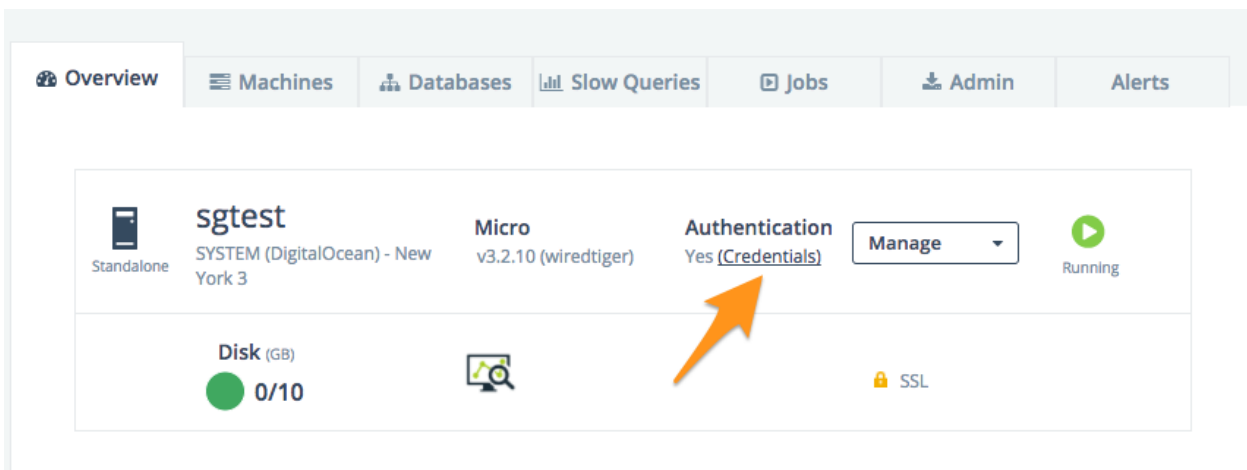
```
scp -i ./scalegrid_os.pem root@your_host.servers.mongodirector.com:/etc/ssl/mongodb* .
```

Alternatively the files can be copied manually, which are located on the remote MongoDB server at these locations:

```
/etc/ssl/mongodb-cert.crt
/etc/ssl/mongodb-cert.key
/etc/ssl/mongodb.pem
```

Now that we've copied over the important files, let's test MongoDB connectivity from the command line to ensure we can connect. This is a very important step before trying to connect with SlamData. This ensures that all network services are running properly (DNS, routing, firewalls, etc) and that both the SSL information and MongoDB user credentials are correct.

You will need the MongoDB password for the *admin* user. On ScaleGrid.io you can find that clicking on the Credentials link under Authentication as the following screenshot shows:



If you don't already have MongoDB installed on your local system, you'll want to install the latest version. Some operating systems such as Linux allow you to install only the MongoDB shell utilities which should suffice.

From within the `ssl_config` directory, connect to the remote MongoDB server:

```
mongo your_server.servers.mongodirector.com/admin --ssl --sslAllowInvalidCertificates \
--sslPEMKeyFile ./mongodb.pem -u admin -p
```

We must pass the `--sslAllowInvalidCertificates` parameter because we are using ScaleGrid's self-signed certificate to connect. If we were using a trusted certificate signed by a Certificate Authority this wouldn't be necessary.

If you are unable to connect to MongoDB from the command line, you will *not* be able to connect through SlamData. Please be sure you can successfully connect with this method before contacting Support for assistance.

Now that we've verified connectivity to MongoDB over SSL, we can continue with importing the keys so that SlamData can use them.

2.5.1 Setup the Java Key Store

We'll need to do some file conversions to get these into the Java Key Store (JKS) format that the JVM requires. If you don't have [OpenSSL](#) installed on your system already, you'll need to install it to perform the following commands:

```
openssl pkcs12 -export -name ScaleGrid -in ./mongodb-cert.crt -inkey ./mongodb-cert.
↪key -out keystore.p12

keytool -importkeystore -destkeystore MyKeyStore.jks -srckeystore keystore.p12 -
↪srcstoretype pkcs12 -alias ScaleGrid
```

This converts the certificate and key file to PKCS12 format and then imports it into a Java Key Store that we'll use later.

Now we'll need to perform a similar process for the Java Trust Store.

2.5.2 Setup the Java Trust Store

The Java Trust Store is in a Java Key Store file format but holds the information about which certificates to trust. Since ScaleGrid gave us a self-signed certificate, we need to add ScaleGrid to our list of trusted providers:

```
openssl x509 -in mongodb.pem -out cert.der -outform der

keytool -importcert -alias ScaleGrid -file cert.der -keystore MyTrustStore.jks
```

2.5.3 Setup SSL for the JVM

The analytics compiler for 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 shown in the following table.

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 several important parameters that must be passed to the JVM at startup to enable SSL. These parameters are shown in the table below and point the JVM to a Java Key Store (JKS).

JVM Option	Example Value	Purpose
javax.net.ssl.keyStore	/dir/MyKeyStore.jks	The location of the encrypted key store file.
javax.net.ssl.keyStorePassword	MySecretPassword	The password required to decrypt the key store file.
javax.net.ssl.trustStore	/dir/MyTrustStore.jks	The location of the encrypted trust store file.
javax.net.ssl.trustStorePassword	MySecretPassword	The password required to decrypt the trust store file.
javax.net.debug	ssl	Optional for troubleshooting.

Examples for these parameters are shown below.

```
-Djavax.net.ssl.keyStore=/my/dir/ssl_config/MyKeyStore.jks
-Djavax.net.ssl.keyStorePassword=MySecretPassword
-Djavax.net.ssl.trustStore=/my/dir/ssl_config/MyTrustStore.jks
-Djavax.net.ssl.trustStorePassword=MySecretPassword
-Djavax.net.debug=ssl
```

Adjust the values above accordingly based on the password you provided during certificate import and proper directory path.

Once the changes are saved, restart SlamData so the new parameters are loaded.

2.5.4 Configuring the SSL Mount

The final step is to add a single parameter to the Mount dialog in SlamData. Add the parameter *ssl* and set the value to *true*.

Section 3 - Configuring SlamData

An example configuration file for SlamData Advanced might appear as follows.

```
{
  "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",
```

Mount

Name

Mount type

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

Password

Database

Settings

Name	Value
ssl	true

Cancel

Mount

```
        "n": "skljhdfiugy...39587dlkjsd"
      }
    ]
  }
},
"auditing": {
  "log_file": "/aws/logdb/slamdata-logs"
},
"metastore": {
  "database": "<h2 config | postgresql config>"
}
}
```

3.1 Configuring HTTP SSL

The subsection of the configuration file below shows an example of the SlamData server listening on port 9090 with SSL encryption enabled.

```
"ssl": {
  "enabled": true,
  "port": 9090,
  "cert": "<base64 encoded pkcs12 cert file>"
}
```

Note: The `cert` value must be the actual contents of the base64 encoded pkcs12 cert file, not the path to it. This will be a very long, multi-line string that will be copied and pasted into the configuration file.

The example steps below walk through how to create a valid certification to include in the configuration file.

Assuming you have been given the following files by your certification provider:

- `private-key.txt`
- `your_server_name_com.ca-bundle`
- `your_server_name_com.crt`

Follow these steps:

1. Create a `.pem` key file from the server certificate and the CA bundle certificate. The order of the files is important. First the server crt, then the ca-bundle file:

```
cat your_server_name_com.crt your_server_name_com.ca-bundle > your_server_name_com.pem
```

2. Create a pkcs12 file from the `.pem` file and the private key file. (scroll to the right if you can't see the entire command)

```
openssl pkcs12 -export -in your_server_name_com.pem -inkey private-key.txt -passout_
↪pass: -out cert-private-key-pair.p12
```

3. Base64 encode the pkcs12 file:

```
base64 cert-private-key-pair.p12 > cert.base64
```

Now copy the contents of the `cert.base64` file into the `cert` field of the configuration file and restart SlamData.

3.2 Configuring Postgres as Metastore

SlamData Advanced defaults to using an H2 java database as its metastore database. Alternatively PostgreSQL 9.x may be used instead.

A Postgres metastore allows SlamData to be clustered to scale.

The following example `quasar-config.json` shows an example:

```
"metastore": {
  "database": {
    "postgresql": {
      "host": "192.168.99.100",
      "port": 5432,
      "database": "slamdata",
      "userName": "postgres",
      "password": "postgres"
    }
  }
}
```

Section 4 - SlamData User Security

SlamData Advanced provides additional features not available in other editions, such as user authorization, authentication, and auditing.

4.1 Security Overview

SlamData Advanced controls user security through the use of tokens, permissions, groups, actions and types. Each of these is defined in the table 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 upon its type.
Type	<i>Structural, Content, or Mount.</i>

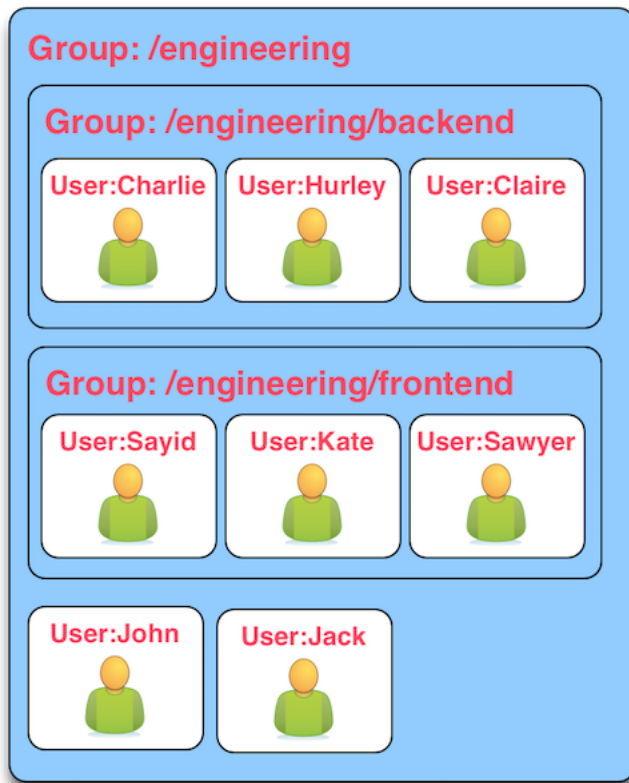
4.1.1 Users

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 upon the configuration of groups and permissions. Users 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).

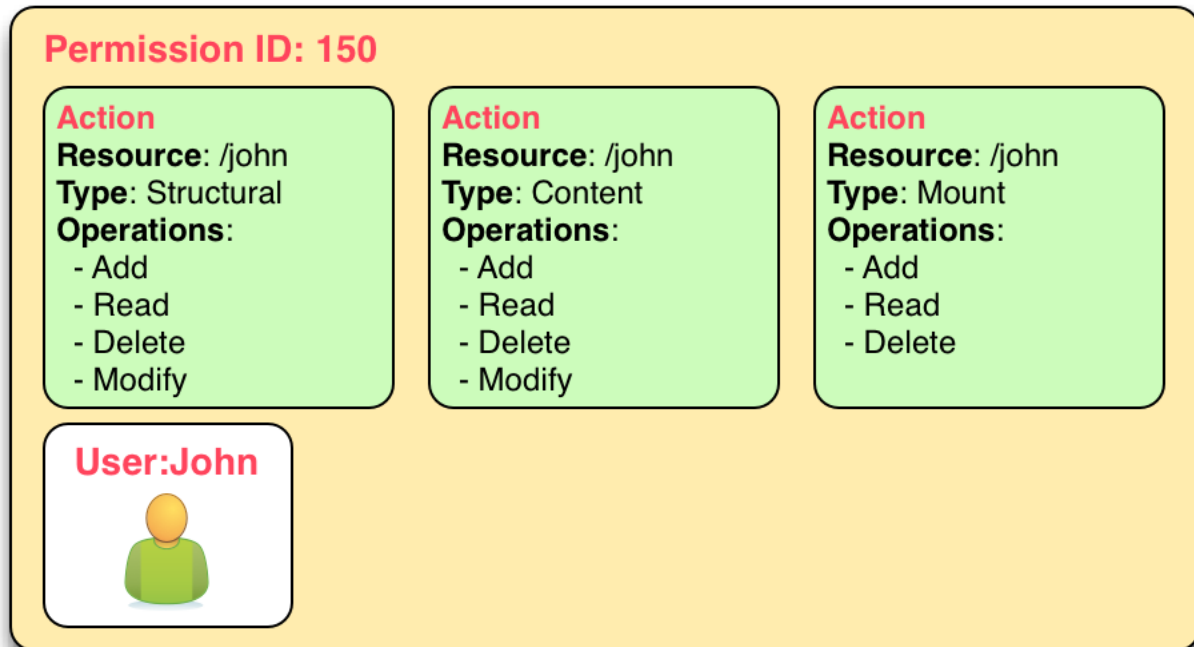


Since permissions may contain a group, and groups may contain users, then a user within a group inherits the permissions assigned to that group.

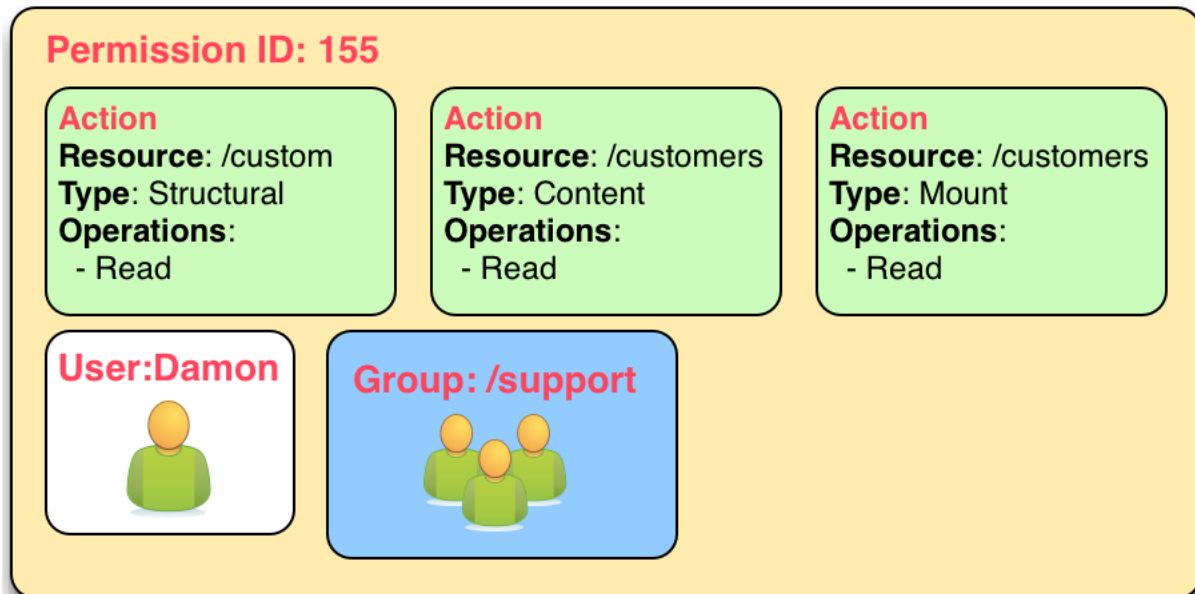
In the example above, both users John and Jack would inherit all of the permissions that contain the /engineering group. Those permissions would also apply to the subgroups for John and Jack.

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.



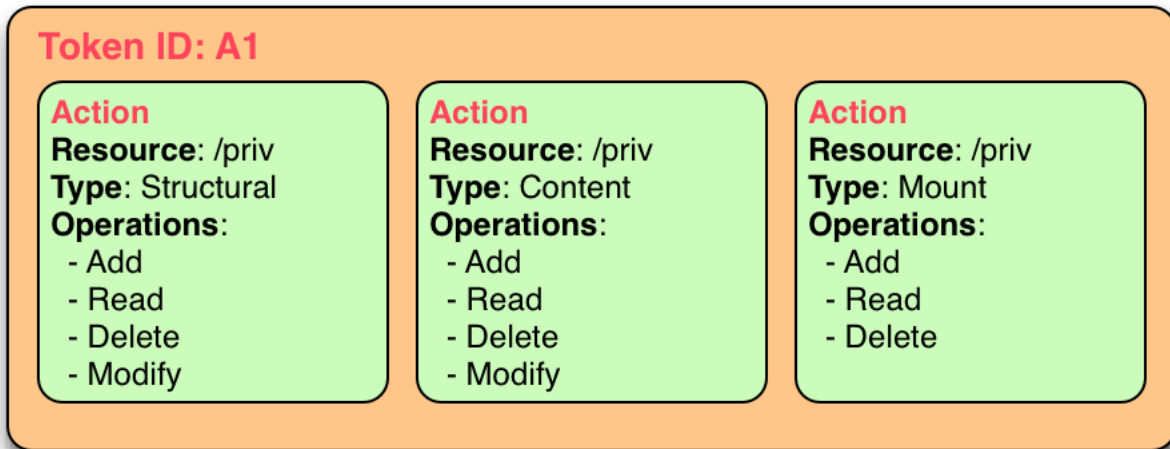
In the example above, both the user `Damon` and any other user within the `/support` group may read data from the `/customers` path, but may not create, modify or delete anything.

4.1.4 Tokens

If a token is passed in a request to SlamData, and the token is valid, the request will proceed based upon 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.

In the following example, if a request included the token `A1`, then any operation performed within `/priv` would succeed, despite the permissions the user actually had.



4.2 Initializing the SlamData Metastore

SlamData Advanced uses a metastore for user security. Before **SlamData Advanced** can be started, the metadata store must be initialized and initial administrator users defined. The administrator users are added to a group having complete and 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, as shown in the following example.

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

4.3 Authentication

SlamData Advanced adds support for authenticated requests via the [OpenID Connect](#) protocol. A request to any SlamData or **SlamData Advanced** 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** along with the “Client Identifier” (CID) issued to the client by the OP, this allows the **SlamData**

Advanced administrator to specify which clients are permitted to access **SlamData Advanced**. 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** 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 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.

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	Not Available

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]
```

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** logs the operations as data in the filesystem where the path is located. This means that it is then possible to use **SlamData Advanced** to analyze the log data.

Section 5 - Security APIs

SlamData Advanced 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 a case, the endpoint will document how to determine what a user can and cannot see.

5.1 - Group Endpoint

GET /security/group/<path>

- Retrieves information about this group. The result of the query will depend upon your permissions according to the rules described below.
- 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 relevant permission as outlined above and the group does not exist, the response will be a 404 Not Found.

```
{
  "members": [ "<user_email>", "..."],
  "allMembers": [ "<user_email>", "..."],
  "subGroups": [ "<group_path>", "..."],
}
```

- `members` All users are explicitly a member of this group.
- `allMembers` All users are 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 the following:

```
{
  "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

5.2 - Authority Endpoint

GET /security/authority

Returns all permissions granted to you.

Response:

```
[<permission>]
```

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 the 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.

5.4 - Token Endpoint

The following 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 you 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.....df1kj",
    "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...2957835lkj"
  },
  {
    "kty": "RSA",
    "kid": "5678",
    "alg": "RS256",
    "use": "sig",
    "n": "skljhdfiugy...39587dlkjsd"
  }
]
}
```



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 an administrator's perspective see the SlamData Administrator's Guide.

For information on how to use SlamData from a user's perspective see the SlamData User's Guide.

Section 1 - Installing and Running SlamData

1.1 Purpose

The purpose of this Developer's Guide is to walk a software developer through SlamData from installation through to a 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.

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 historical two-dimensional data such as RDBMS in rows and columns, but also deeply nested, semi-structured data such as JSON and XML.
- Ability to understand schemas dynamically, resulting in absolutely no requirement 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 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.

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 a data source, such as a database system.
- Has appropriate permissions to install relevant software.

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.

1.5 Installation

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

1.6 Starting SlamData

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

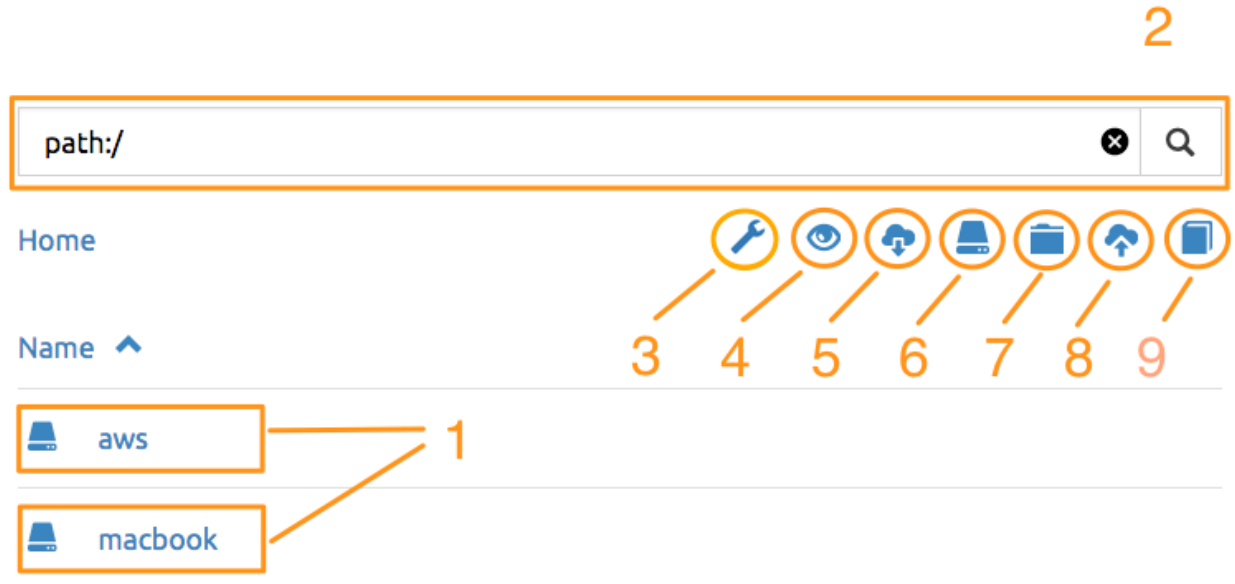
Once SlamData is running then continue to Section 2.

Section 2 - Exploring Data

By the end of this Developer's 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.

2.1 Interface Navigation

The image below shows the Home screen after starting SlamData. 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 wrench icon configures a mount.
4	The eye icon toggles visibility of the trash can icon.
5	Download all data starting from this path.
6	Mount a new data source.
7	Create a new folder in the datasource virtual file system.
8	Upload a data file.
9	Create a new workspace.

2.2 Workspaces, Decks and Cards

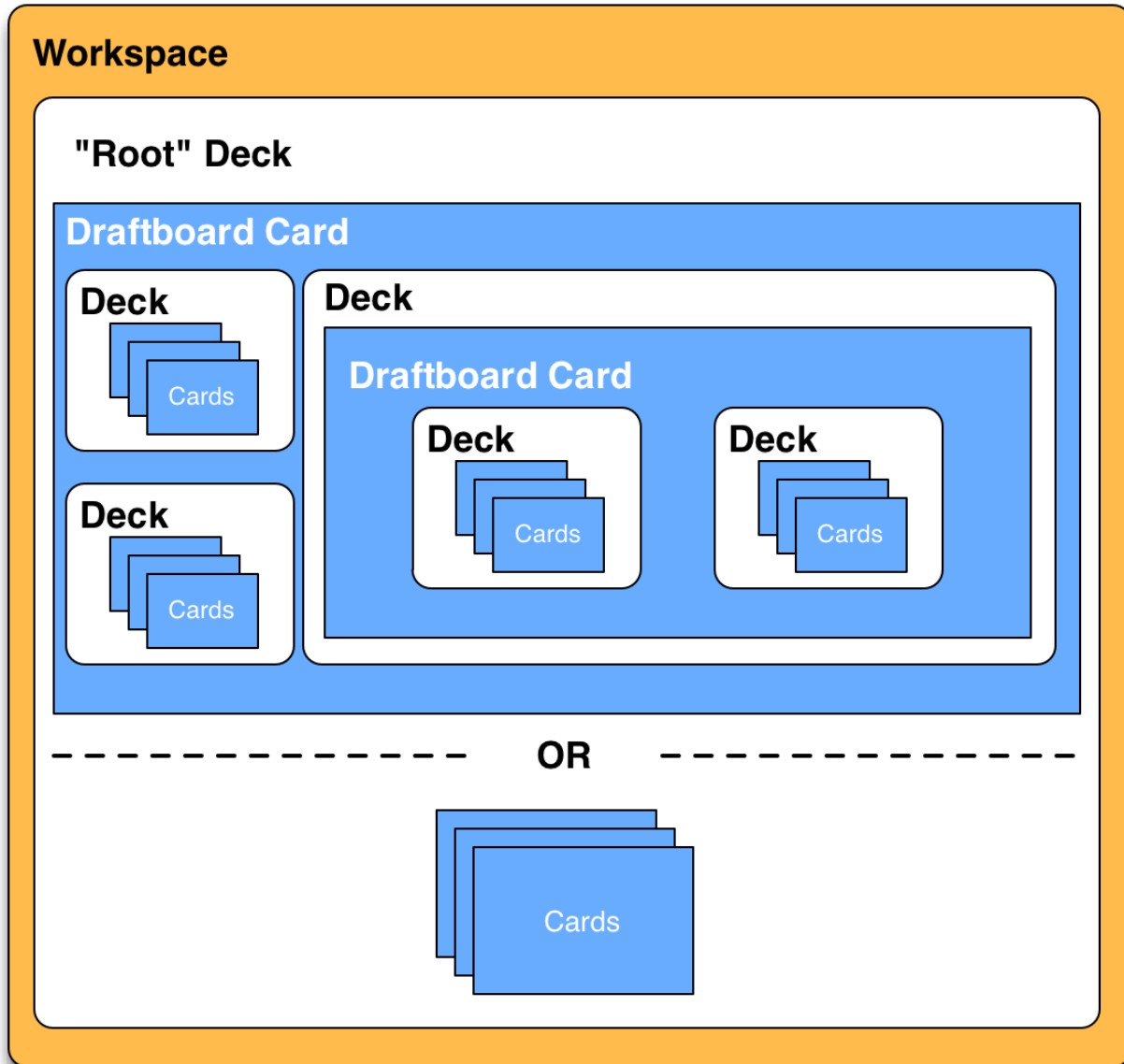
Before we start looking at our data we need to discuss how to interact with it. This is done through the use of a **Workspace**. A Workspace is the primary method that users interact with data within SlamData. A Workspace in turn is comprised of cards, and decks of cards.

- **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.

- Search Card.
- Preview Table 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.

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

Mount type

Cancel

Mount

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"/>		

2.4 Creating a Database

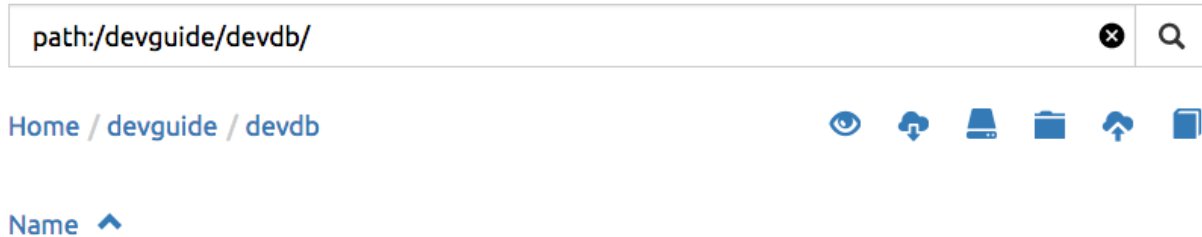
- Click on the newly created server named **devguide**. The interface now shows the databases that reside within the database system. A new database will need to be created to follow along with the guide.

- 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:




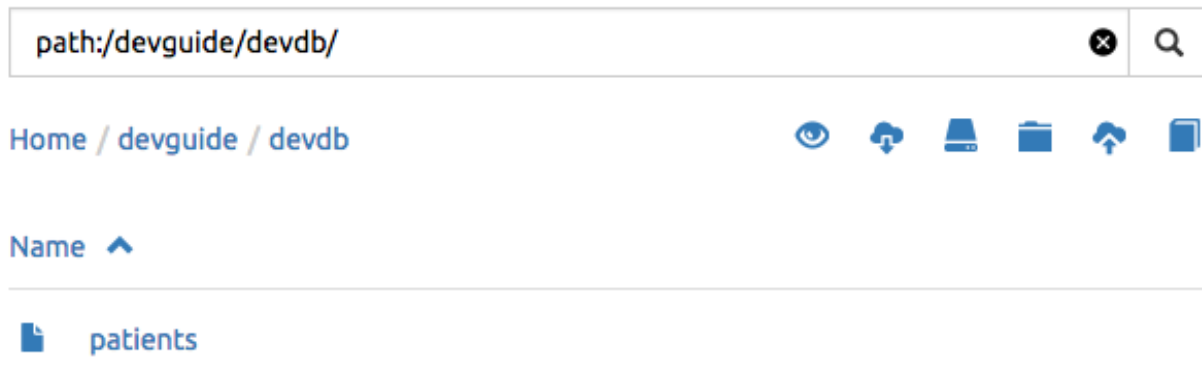
So far in this guide you've installed SlamData, mounted a database and created and renamed a folder. Good progress. Let's now get some data into the database and start exploring.

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 that the SlamData UI is in **devdb**, 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 as follows:



As you can see, it is easy to quickly import JSON data into SlamData. Other formats, such as CSV, can also be quickly imported.

2.5.1 Indexing Your Database

Attention: Indexing Your Database

While this step is not 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 the database system performs your searches.

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

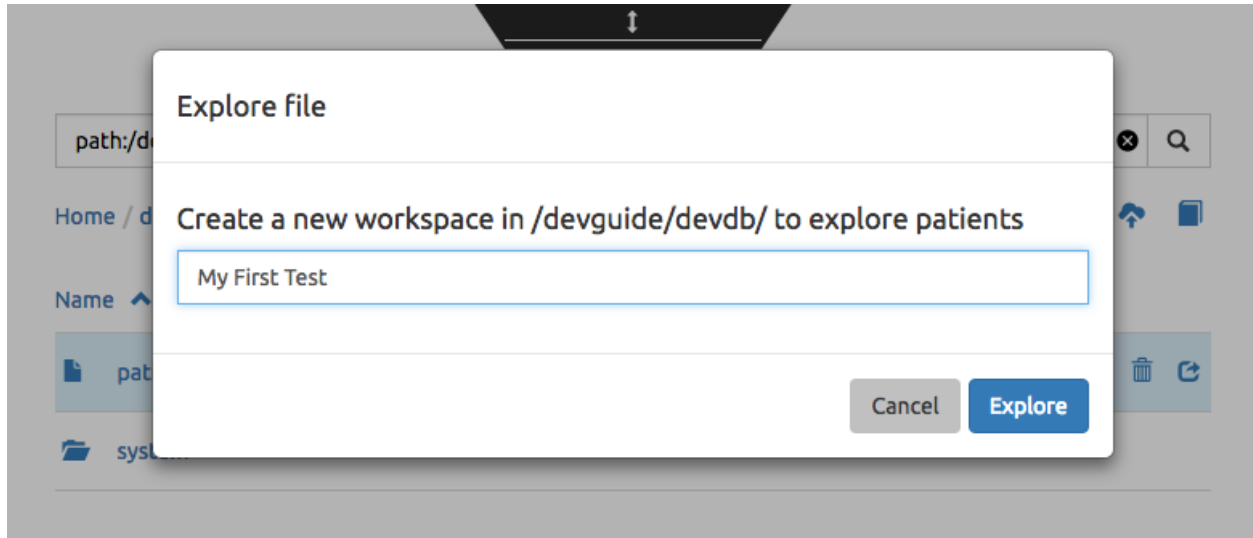
```
use devdb
db.patients.createIndex({first_name:1})
db.patients.createIndex({middle_name:1})
db.patients.createIndex({last_name:1})
db.patients.createIndex({city:1})
db.patients.createIndex({county:1})
db.patients.createIndex({state:1})
db.patients.createIndex({zip_code:1})
db.patients.createIndex({street_address:1})
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!

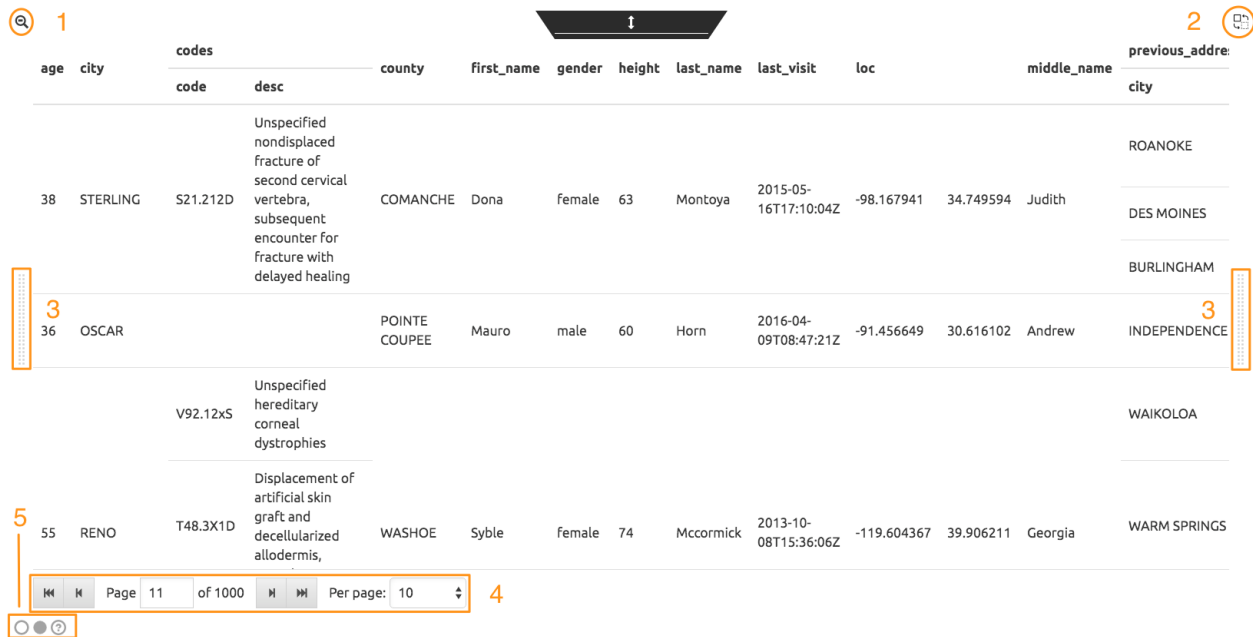
2.6 Exploring Data

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` and clicking **Explore**.



Once you click Explore, the following screen should appear:



Number	Description
1	Zoom icon takes user 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 a **Preview Table Card**! SlamData did this automatically to save you time.

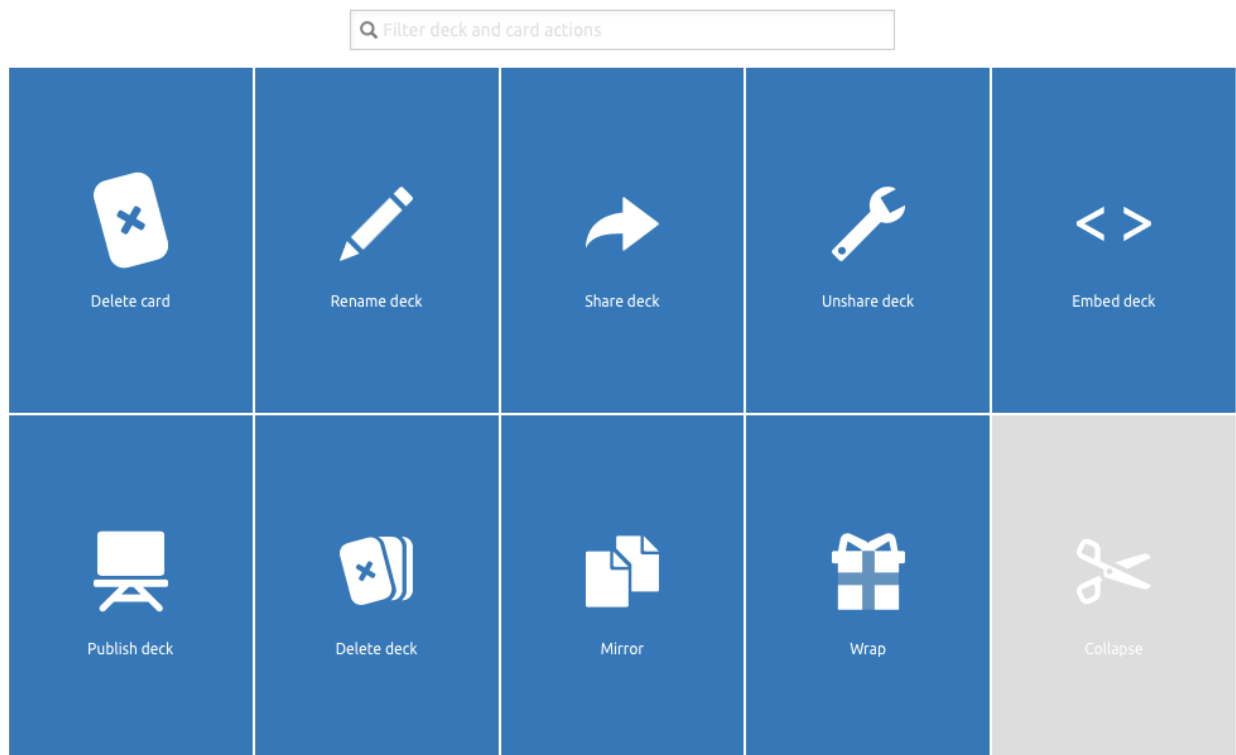
Any changes made within a Workspace are saved automatically. At any time the user may zoom out of the current window.

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 the previous image).

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

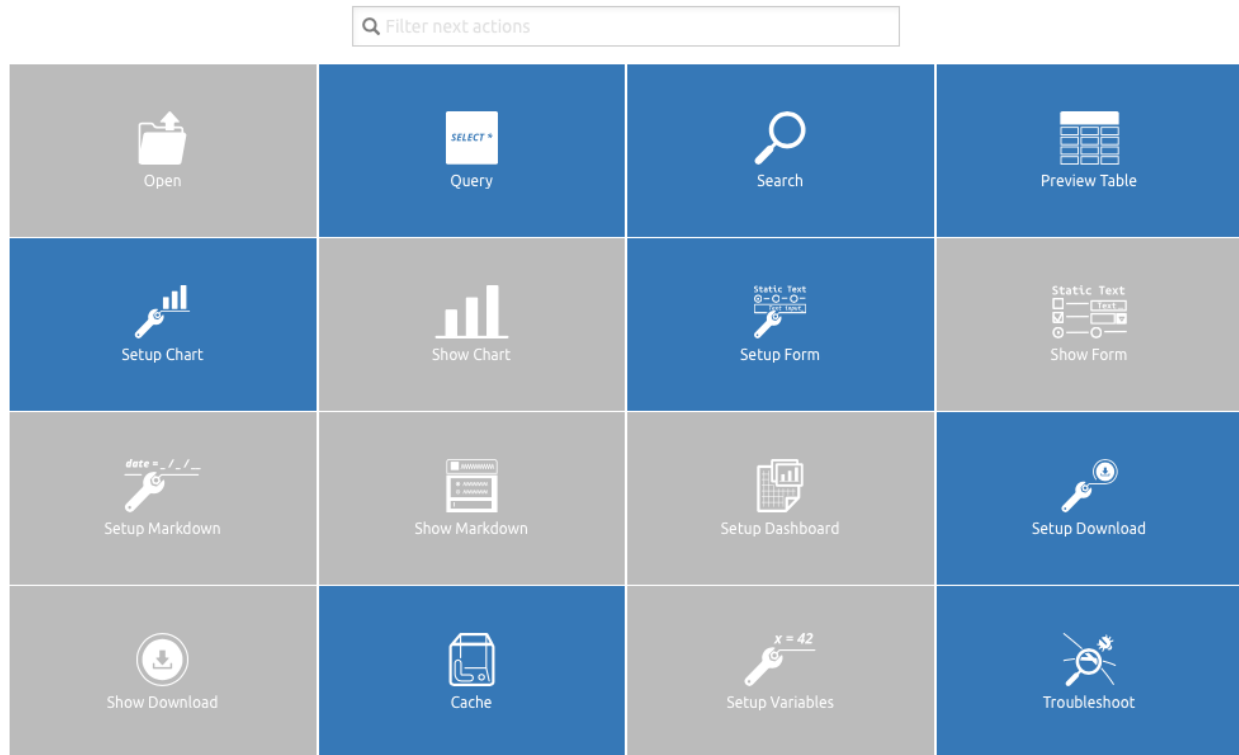


- Click on **Delete card**.

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 colors. Blue 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.

- 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.

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

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 the database system, and that indexes can significantly boost performance for searches.

Once the results appear, you can browse them 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 data sources 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 either drag the right grip bar to the left, or simply click on the right grip bar.
- View the results in the **Preview Table Card** again.

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 **Preview Table Card** again.

As one last example let's see how we can mix and match different types. We want to know how many males over the age of 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 greater than or equal to 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!

2.8 Querying Data with SQL²

In addition to the **Search Card**, SlamData provides a **Query Card** that 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 **Preview 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 delete 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.

- Type in the following query:

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

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

- Select **Run Query** in the bottom right.
- Click the right grip.
- Select the **Preview 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.

- Select **Run Query** in the bottom right.
- Slide back to the **Preview Table Card** to see the results.
- Slide back to the **Query Card**.

Let's now create a query that formats the results a little better.

- 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
```

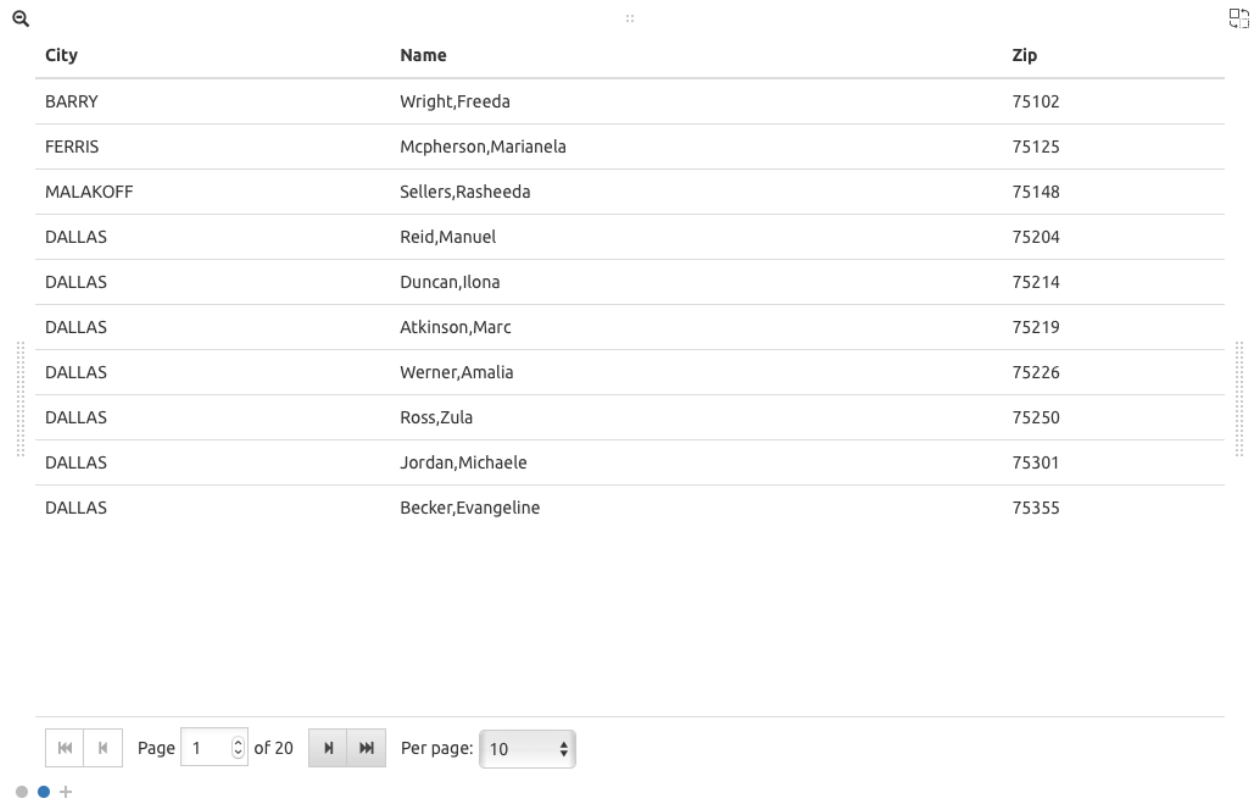
- Select **Run Query** in the bottom right.
- Slide back to the **Preview Table Card** to see the results.

Notice in this query we are concatenating the **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
```

- Select **Run Query** in the bottom right.
- Slide to the **Preview 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!

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.


3.1 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.

Ensure that you are 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. Paste the following text into the card:

```
# 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 = ____-__-__ __:__
```

- Select **Run Query** in the bottom right.
- 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
```

- Select **Run Query** in the bottom right.
- 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.


3.2 Interactive Markdown Forms

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

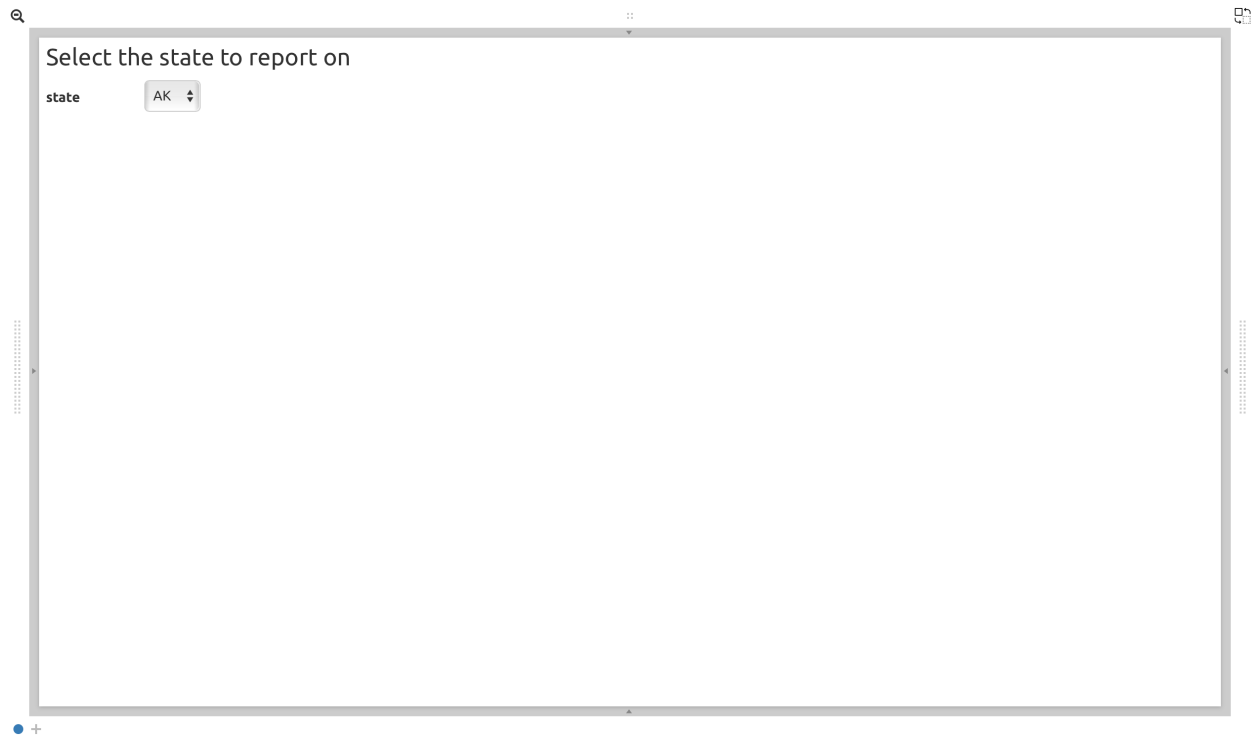
First we want the user to select the state to report on. This will then allow us to query the database for patients that reside in cities within that state.

- 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``}
```

- Select **Run Query** in the bottom right.
- 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 **Wrap**.

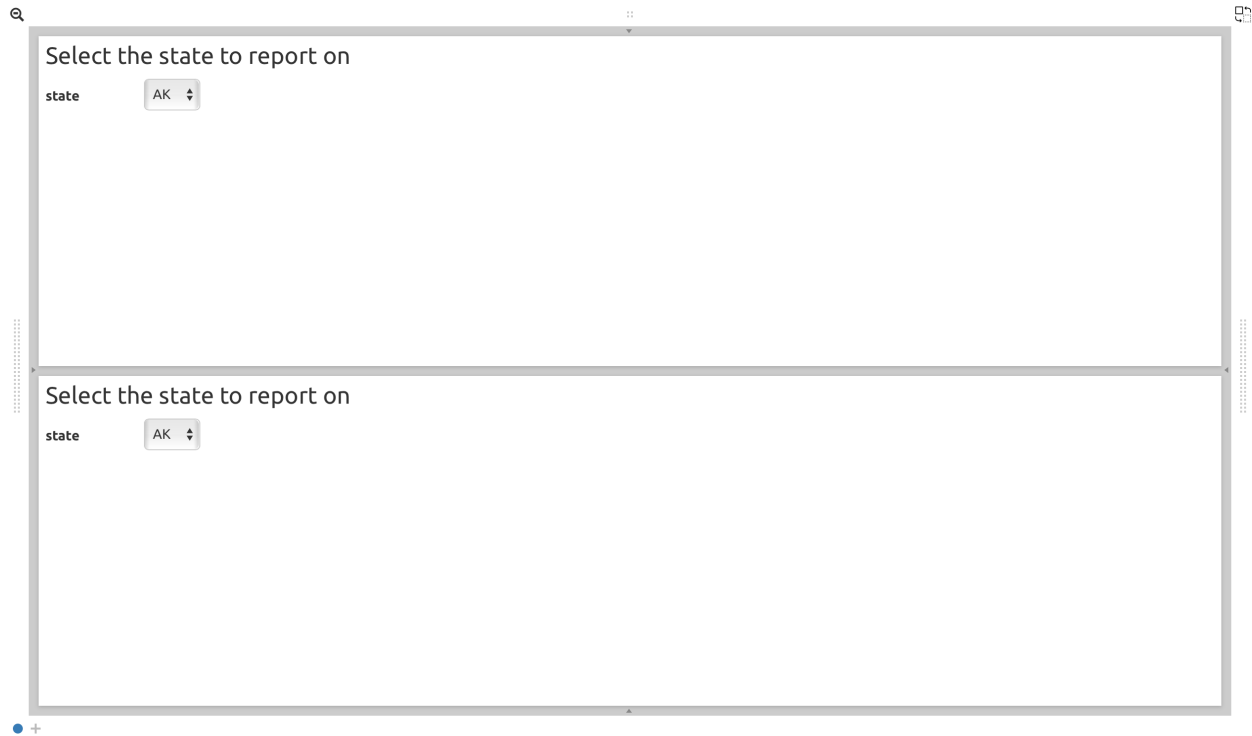
Note that your interface should now look similar to the following:



You can 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 **Mirror**.

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**.

- Select **Run Query** in the bottom right.
- Click on the right grip to create a new card and select the **Preview Table Card**.

Select the state to report on

state MD

AvgWeight	City
466	TODDVILLE
445	DEAL ISLAND
367	GLEN ECHO
340	FREELAND
333	COLTONS POINT
331	MARBURY
314	GREENBELT

Page 1 of 14 Per page: 10

- 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 the query and limit the results to 20 cities, so a bar chart doesn't appear crowded.

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


```
LIMIT 20
```

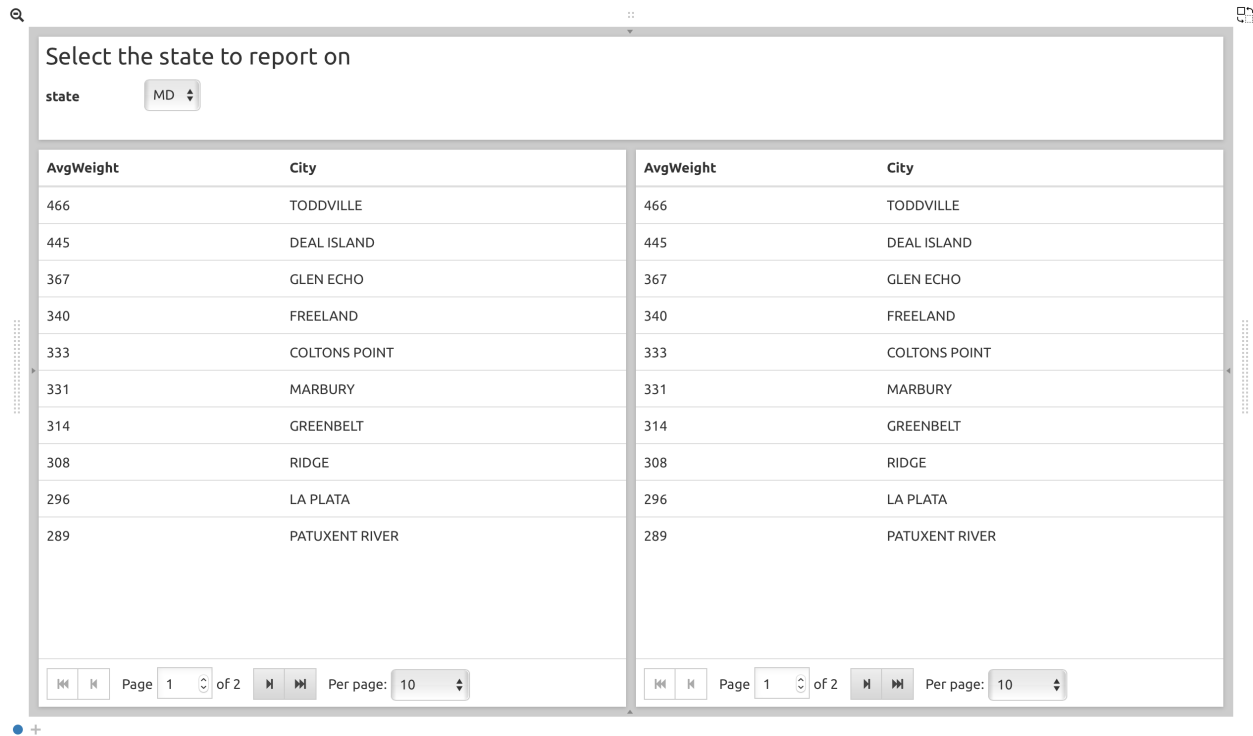
- Select **Run Query** in the bottom right.
- Slide back over to the **Preview Table Card**.

Now we are ready to add some visualizations!

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 **Preview Table Card**:

- Click on second deck to make it active.
- Click on the flip icon to flip the deck over. 
- Select **Mirror**.
- Resize so that your interface looks similar to the following image:

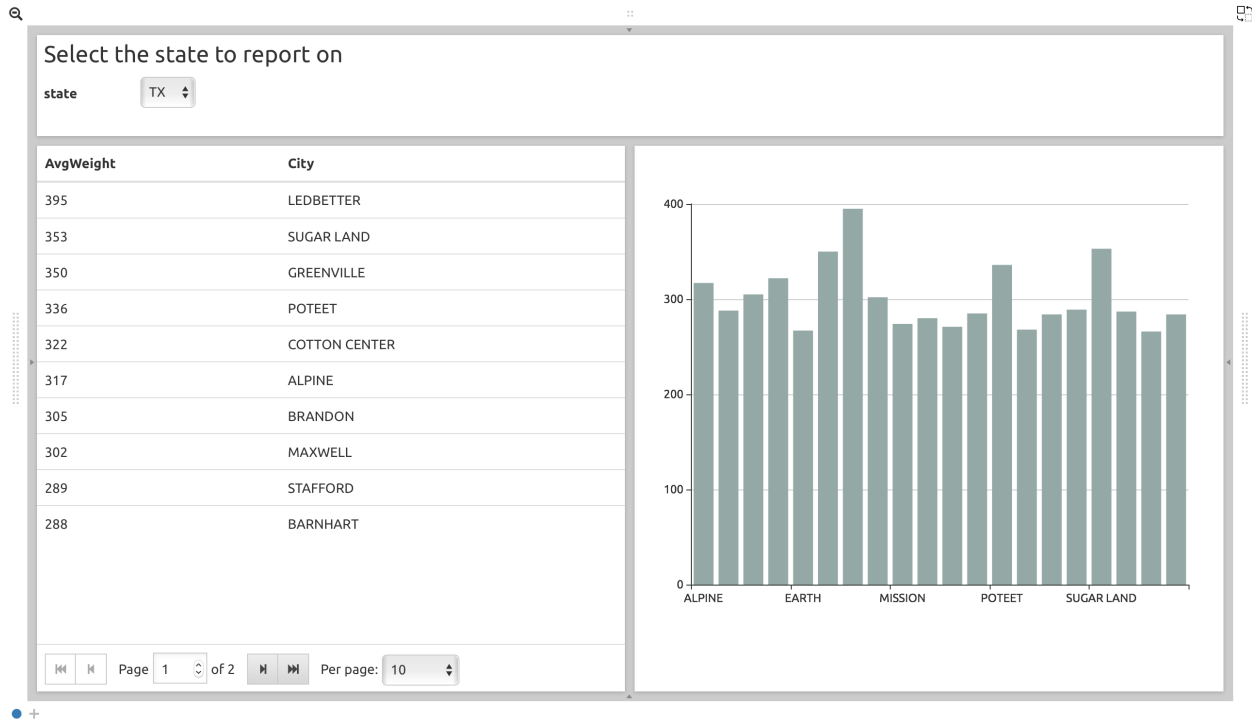


- Select the new deck and click on the right grip and then select the **Setup Chart Card**.
- Select the **Bar Chart** icon.

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

- For the **Category**, select **.City** as the axis source.
- Slide to the right to create a new card and select **Show Chart**.

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.

Section 4 - Publishing and Simple Embedding

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 **Publish deck**.

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



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

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

Section 1 - Basic Queries

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 });
```

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": [
                      { "$literal": "Full Name is " }, "$first_name"
                    ],
                    { "$literal": " " }
                  ],
                  "$last_name"
                ],
                { "$literal": undefined }
              ],
              { "$literal": undefined }
            ]
          ]
        }
      ],
      { "allowDiskUse": true }
    ];

```

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 converted 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 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)) && ((value.last_visit instanceof Date) || (value.last_visit_
↪ instanceof Timestamp))) ? (((value.last_visit.getFullYear() instanceof NumberInt)
↪ || (value.last_visit.getFullYear() instanceof NumberLong)) ? String(value.last_
↪ visit.getFullYear()).replace(
              RegExp("[^-0-9]+", "g"),
              "") : ((value.last_visit.getFullYear() instanceof Timestamp) || (value.
↪ last_visit.getFullYear() instanceof Date)) ? value.last_visit.getFullYear().
↪ toISOString() : String(value.last_visit.getFullYear())) + "-" + (((value.last_
↪ visit.getMonth() + 1) instanceof NumberInt) || ((value.last_visit.getMonth() + 1)
↪ instanceof NumberLong)) ? String(value.last_visit.getMonth() + 1).replace(
              RegExp("[^-0-9]+", "g"),
              "") : (((value.last_visit.getMonth() + 1) instanceof Timestamp) ||
↪ ((value.last_visit.getMonth() + 1) instanceof Date)) ? (value.last_visit.getMonth()
↪ + 1).toISOString() : String(value.last_visit.getMonth() + 1)) : undefined
          }
        ]
      })(
        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.

The following example assumes a collection that 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 } );
```

1.4 Grouping

1.4.1 By Calendar Quarter

The following example assumes a document 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 part of the 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`.

Second part of the query:

```
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 first part of the query, 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.



Complete 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
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
```



This 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



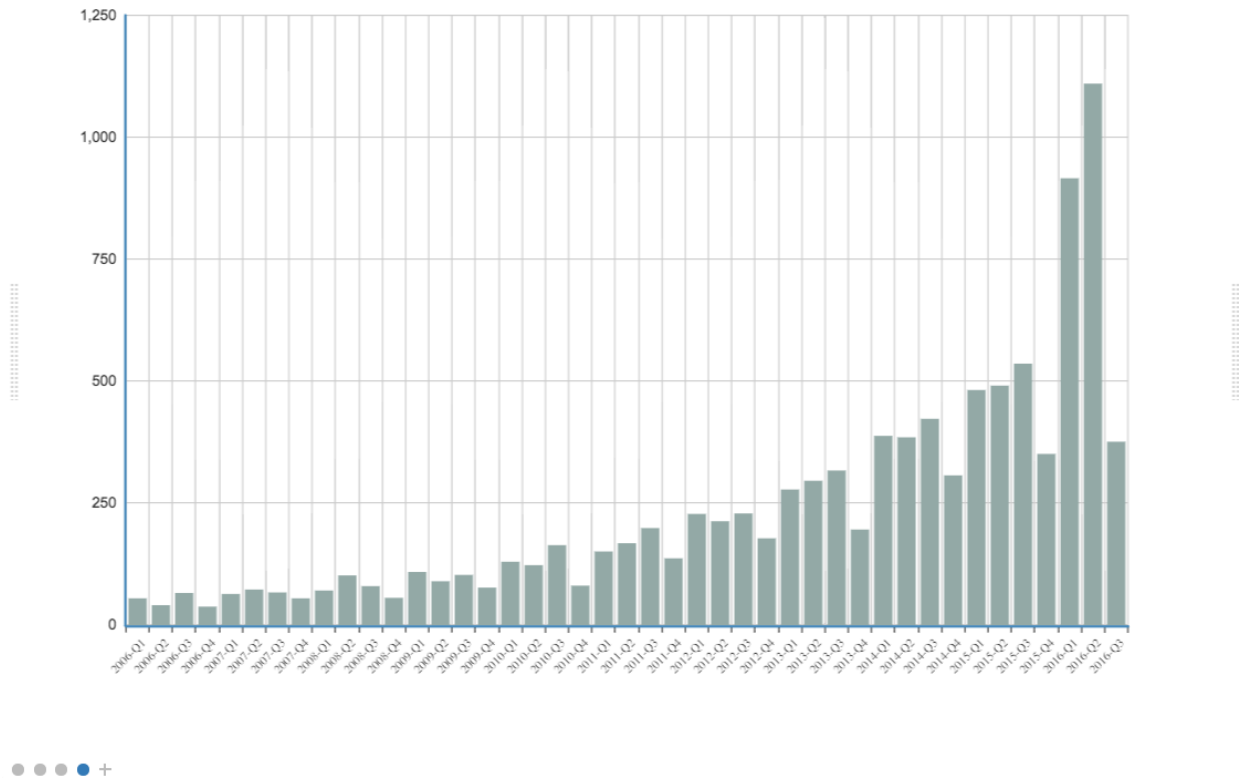
Page 1 of 5



Per page: 10



When the query results are rendered as a bar chart, the output would look similar to the following:



Section 2 - Complex Queries

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

The following examples assume a document structure similar to the following, using 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 type IIIA, IIIB, or IIIC with routine healing"
  },
  {
    "code": "S72.009E",
    "desc": "Other yatapoxvirus infections"
  },
  {
    "code": "S56.417D",
    "desc": "Other fracture of shaft of radius, left arm, subsequent encounter for_
↪ closed fracture with routine healing"
  },
  {
    "code": "B55.2",
    "desc": "Varicose veins of right lower extremity with ulcer of thigh"
  }
],
"street_address": "8320 45TH ST",
"zip_code": 7067
}
```

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, as shown in the example output below. 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

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 the following 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

Q :: 92

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	M89.06Z	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 now of the MongoDB **\$unwind** operator in the code below.

```
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" ] },
```

```
        { "$lit": ["$__tmp9.last_name", { "$literal": { } } ] } }
    },
    {
        "$concat": [
            { "$concat": ["$__tmp9.last_name", { "$literal": ",", } ] },
            "$__tmp9.first_name"
        ],
        { "$literal": undefined }
    },
    { "$literal": undefined }
},
{
    "PATIENT_AGE": "$__tmp9.age",
    "Z_CODES": "$__tmp8"
}
},
{
    "$project": { "NAME": true, "PATIENT_AGE": true, "Z_CODES": true, "_id": false }
},
{ "allowDiskUse": true });
```



Section 1 - Introduction

SQL² is a subset of ANSI SQL. SQL² is designed for queries on NoSQL database systems.

SQL² has support for every major SQL **SELECT** clause, such as **AS**, **WHERE**, **JOIN**, **GROUP BY**, **HAVING**, **LIMIT**, **OFFSET**, **CROSS**, and so on. It follows PostgreSQL where SQL dialects diverge.

1.1 Data Types

The following data types are used by SQL².

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

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

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

Section 2 - Basic Selection

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

2.1 Select all values from a path

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

Example:

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

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`
```

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
```

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

3.1 Filtering using a numeric value

Example:

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

3.2 Filtering using a string value

Example:

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

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"
```

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

4.1 - 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+"
```

Section 5 - Dates and Times

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

Note: Some database systems will automatically convert strings into dates or date/times. SlamData does not perform this conversion, since the underlying database system 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: 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.

5.1 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")
```

5.2 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")
```

5.3 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")
```

5.4 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), `epoch`, `hour`, `isodow`, `isoyear`, `microseconds`,

millennium, milliseconds, minute, month, quarter, second, week and year, although some values are not supported by all connectors.

Example:

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

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)
```

Section 6 - Grouping

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

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
```

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
```

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)
```

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"
```

6.5 Filter with Arbitrary Value

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

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
```

Section 7 - Nested Data and Arrays

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

7.1 Nesting

Nesting is represented by levels separated by a full stop (.).

Example:

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

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"
```

Section 8 - Pagination and Sorting

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
```

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
```

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 upon the dataset sizes.

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

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)
```

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 b` = 32 which is less than `lb c` = 128, but with the filter, `lb c` is limited to the number of documents in b, which is 8 (and which is lower than the unconstrained `la b`).

Section 10 - Conditionals and Nulls

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
```

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
```

Section 11 - Data Type Conversion

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`
```

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`
```

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.

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.

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.

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.

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
```

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 using 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.

Section 13 - Database Specific Notes

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`
```

Note: When using the `_id` field, it must be escaped in backtick characters or you will get an error. You must also give the `_id` an alias or it will not show up, even if you have it in your `SELECT` statement.

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, as shown in the example below.

```
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 is 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 later.



Reference - SlamDown

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

Section 1 - Introduction

SlamData contains its own markup language called SlamDown, that 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*.

Section 2 - Block Elements

The following SlamDown elements create blocks of content.

2.1 Horizontal Rules

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

Example:

```
Text here

---

More text here
```

This results in the following output:

Text here

More text here

2.2 Headers

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

Example:

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

This results in a first, second, and third level heading, as follows:

Top Level

Second Level

Third Level

2.3 Code Blocks

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

1. Indented code blocks

Indent by four spaces.

Example:

2. Fenced code blocks

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

Example:

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

Both **Indented Code Blocks** and **Fenced Code Blocks** result in the following output:

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

2.4 Paragraphs

Paragraphs are separated by a blank line.

Example:


```
This is paragraph 1.  
This is paragraph 2.
```

This results in the following output:

This is paragraph 1.

This is paragraph 2.

2.5 Block quotes

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

Example:

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

This results in the following output:

This is a block quote.

2.6 Lists

Ordered lists start with numbers followed by a full stop (.). The actual numbers in the SlamDown do not matter, as the list will be displayed with ascending indices.

Example:

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

This results in the following output:

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

Unordered lists start with either an asterisk (*), dash (-), or a plus sign (+). All three are interchangeable.

Example:

```
- First item  
- Second item  
- Third item
```

This results in the following output:

- First item
- Second item
- Third item

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 an SQL query* and put the result into the content.

3.1 Emphasis and Strong Emphasis

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

Example:

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

This results in the following output:

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

3.2 Links

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

Example:

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

This results in the following output:

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

If the link title and destination are the same, an autolink can be used, where the URI is contained in angled brackets (<>).

Example:

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

This results in the following output:

<http://slamdata.com>

3.3 Images

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

Example:

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

This results in the following output:

3.4 Inline code formatting

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

Example:

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

This results in the following output:

Start SQL statements with `SELECT * FROM`

Section 4 - Evaluated SQL² Queries

SlamDown extends Markdown by allowing you to evaluate an 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 following example, there are 20 documents in the `/col` file.

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

This results in the following output:

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.

Section 5 - Form Elements

Form elements provide interactive forms for user’s with text fields, date pickers, check boxes, and so on.

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

Example:

```
name = _____
```

This defines the variable `name` and creates a simple text entry field in the browser. This variable can then be used in a **Query Card**.

Example:

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

Note that the variable name needs to be preceded by a colon (:) when referencing it as a variable inside a **Query Card**.

5.1 Text Field

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

The following code creates an input field for a user's interests. The value can then be referred to as `:interests`.

Example:

```
interests = _____
```

Optionally, the input field can be pre-filled with a default value by having it after the underscores in parentheses. The following code creates an input field called `interests` with a default value of "SlamData". The value can then be referred to as `:interests`.

Example:

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

5.2 Numeric Field

By default, input fields are evaluated as string types. To enforce a numeric type, prefix the underscores with the (#) symbol. A default value can also be provided.

Example:

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

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 by a query.

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.

This following code creates a set of radio buttons with the values "car", "bus", and "bike", where "bus" is marked as the default. The result is stored in the string variable named `commute` for later use.

Example:

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

This results in the following output:



Note that the default selection became the first selection when the radio buttons are rendered.

5.3.2 Dynamic Radio Buttons

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

The following code creates a set of radio buttons that list the unique color values in a database.

Example:

```
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, note how the field is defined on multiple lines.

Second, there are now two queries instead of one. The first query defines which value is selected by default, the second query defines the remaining values.

This results in the following output:



5.4 Checkboxes

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

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 as follows.

Example:

```
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.

The following code 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.

Example:

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

This results in the following output:



phones



iPhone



Android



Blackberry

Similar to the behavior of radio buttons, the fields pre-selected with an x are rendered first.

The selections above would result in the `phones` variable array containing the following values: ["iPhone", "Android"]

5.4.2 Dynamic Check Boxes

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

The following code creates a set of checkboxes that list the phone types within a database.

Example:

```
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 following output:



myphone



Android



Blackberry



iPhone

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

5.5 Dropdowns

Dropdowns allow user's 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 for longer lists of values.

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

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 as follows.

Example:

```
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 ({ }).

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

Example:

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

This results in the following output:



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

Example:

```
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.

The following code creates a dropdown that contains the names of cities within a database.

Example:

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

5.6 Dates and Times

Provide a date, time or both date and time selector by implementing the following syntax.

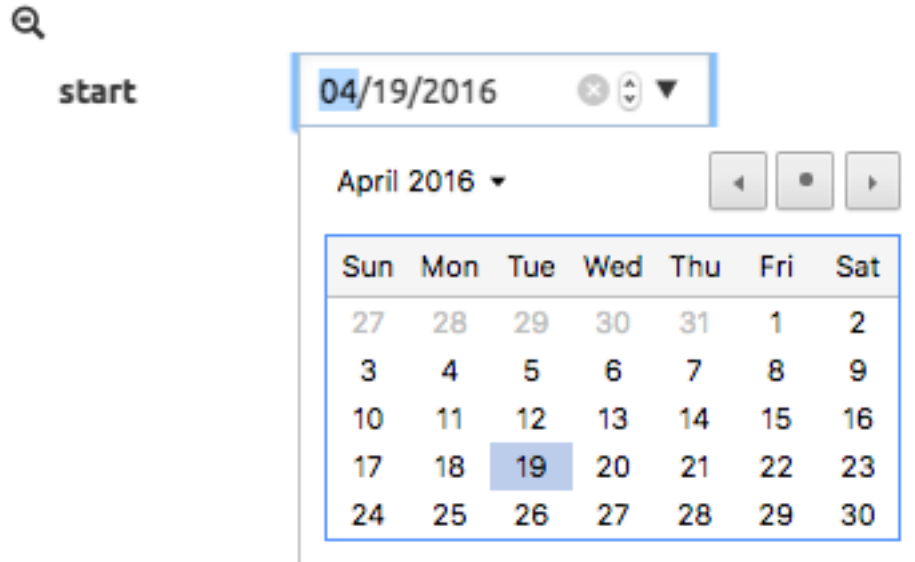
5.6.1 Date

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

Example:

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

This results in the following output:



5.6.2 Time

The following code creates a time selector element.

Example:

```
start = __:__ (02:30 PM)
```

This results in the following output:



5.6.3 Date & Time (TIMESTAMP)

The following code creates both a date and time selector element.

Example:

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

This results in the following output:



start

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

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.



Section 1 - Configuration

1.1 Configuration File Locations

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

Operating System	File Location
Mac OS	\$HOME/Library/Application Support/quasar/quasar-config.json
Microsoft Windows	%HOMEDIR%\AppData\Local\quasar\quasar-config.json
Linux (various vendors)	\$HOME/.config/quasar/quasar-config.json

Warning: Modifying the configuration file

If the configuration file needs to be modified by hand, a backup copy should be created first. Furthermore, if the file is modified while SlamData is running, any changes may be overwritten.

1.1.1 Configuration File Differences

SlamData Community Edition relies on the **quasar-config.json** configuration file to store all metadata for the product, including server configurations, mount points, views, and so on.

SlamData Analyst and Advanced Editions rely upon a PostgreSQL or Java H2 database to store metadata. Depending upon the edition, additional information will be stored such as security information for users, groups, permissions, actions and tokens.

If there is no metadata source when SlamData Analyst or Advanced Edition start, the **quasar-config.json** file will be used.

1.2 Log File Locations

SlamData has a single log file whose location depends upon the Operating System. Replace `version` in the table below with the actual version number that you are running.

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

Section 2 - Running SlamData

2.1 SlamData Won't Start

2.1.1 License Problems

This section is primarily for users experiencing errors when activating a license. If no errors occur during SlamData start up, and none appear in the UI then you may skip this section.

SlamData requires a valid license key to start. Users receive a license key when requesting a trial of SlamData at <https://slamdata.com/get-slamdata> or when purchasing SlamData.

When signing up for the trial, a SlamData.com user account is also created. This user account and password should be remembered as it is needed to log into SlamData later. (See section 2.1.2 Authentication below) Make sure to note whether the registered email address contains capitalization anywhere.

An email containing the license key will be sent and the license will be displayed in-browser after registration is complete. This license will be entered during installation through the installer or can be entered manually into the VMOptions file after manual installation is complete. The location of this file varies based on OS:

Operating System	File Location
macOS	/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

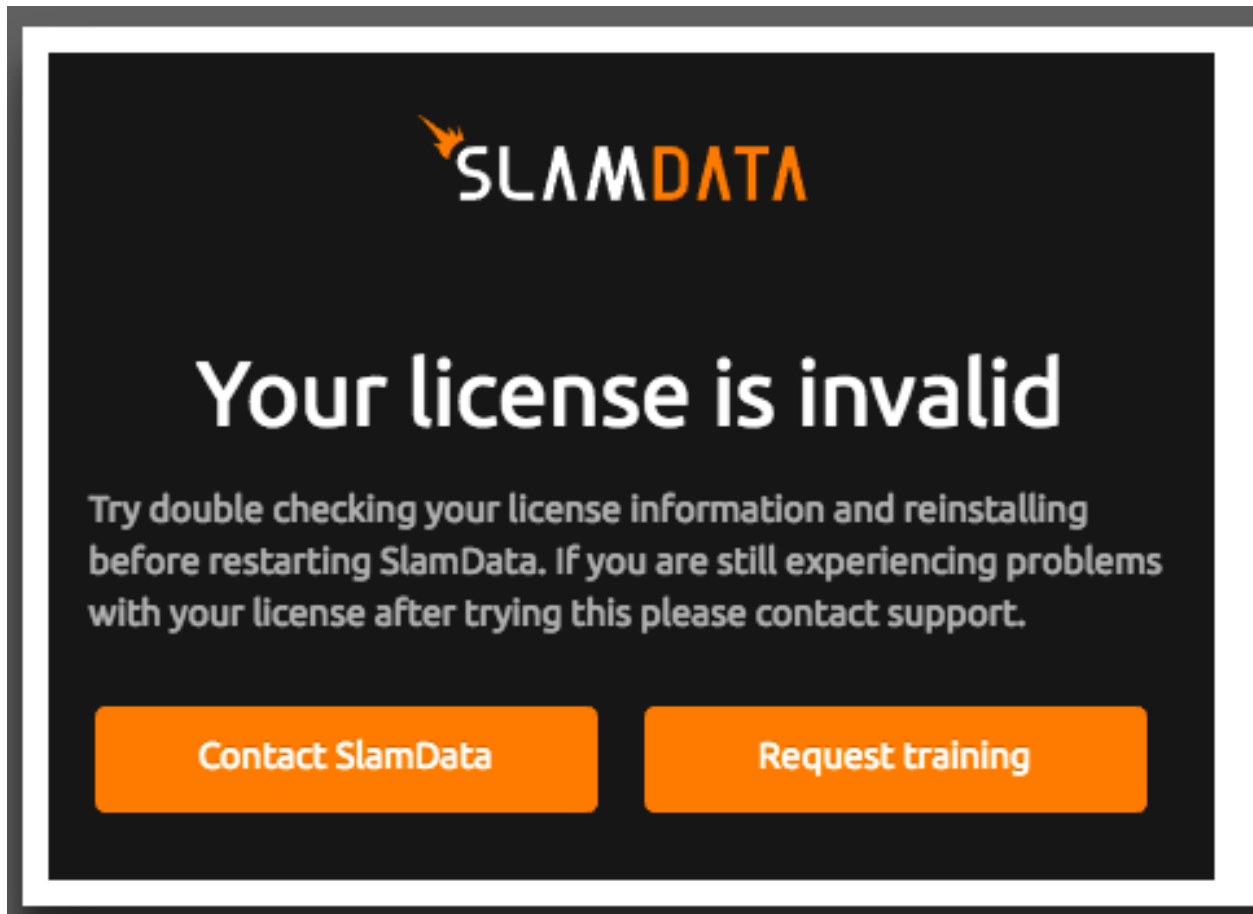
This file may be located elsewhere if SlamData was installed manually.

The license key is in the form ABCDE-FGHIJ-KLMNO-PQRST-UVWXY.

All license parameters inside of the vmoptions file must be defined even if they do not contain a value, in which case simply use quotation marks. The only field that *must* contain a value is the license key as shown below:

```
-Dlicense_key=ABCDE-FGHIJ-KLMNO-PQRST-UVWXY
-Dlicense_full_name=" "
-Dlicense_registered_to=" "
-Dlicense_email=" "
-Dlicense_company=" "
-Dlicense_tel_number=" "
-Dlicense_fax_number=" "
-Dlicense_street=" "
-Dlicense_city=" "
-Dlicense_zip=" "
-Dlicense_country=" "
```

If any parameter shown above is not defined then the activation of a license will fail.



SlamData must be able to validate the license key against a server located on the public Internet. The http server built into the SlamData product does not allow configuration of a proxy web server. If your network utilizes a proxy server for web traffic, this may require a network administrator to allow the SlamData server to communicate directly with the license server, sometimes referred to as “punching a hole” in the firewall.

License Server Information:

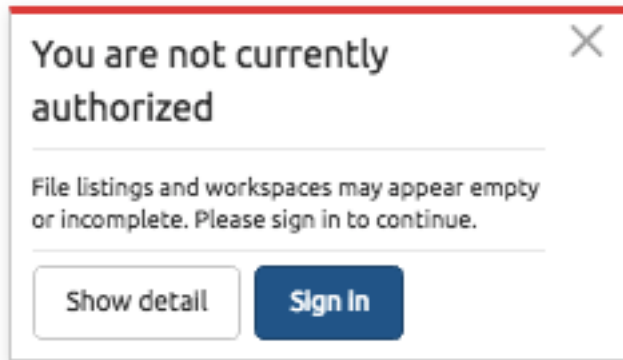
```
IP Address: 97.74.234.176
Port       : 443
```

The SlamData server date and time should also be synchronized with an NTP server. If the date and time are skewed by too much the license check may fail.

You may also check the following URLs to gather more information which may assist in troubleshooting the problem:

```
http://your_host:20223/server/licenseInfo
http://your_host:20223/server/licensee
```

2.1.2 Authentication Problems



SlamData requires that users authenticate before using it. This means that the user is proving they are who they say they are. This happens by providing an email address and a password. This action does not determine what the user can do, only if the user is valid.

SlamData utilizes an OAuth2 server located on the Internet to do this. Users will need to authenticate with the exact email address and password used when registering for the trial. Case sensitivity is important with the email address and password. If the registered email address and password contained specific case, it will need to be used here too.

Clicking the “Sign in” icon in the upper right should result in the option to log into SlamData.com. If it is not listed as an option this typically means SlamData is unable to contact the authentication server.

Connectivity between the SlamData server and the public authentication server should be verified. Again if a proxy server is used on the network then a network administrator will need to open a route to the authentication server:

```
IP Address: 67.207.95.29
Port       : 443
```

Once connectivity has been verified SlamData may need to be restarted.

2.1.3 Authorization Problems

Once a user has authenticated against the authentication server then the internal SlamData authorization model controls what the user has access to. Note that a user may successfully log into SlamData but not have any authorization permissions to perform actions.

The email address provided by the user during installation is automatically configured as part of the Admin group in the local SlamData installation. Members in the Admin group are allowed to perform all actions. If the user email address does not exactly match what was entered during both during registration *AND* installation then the user may not be able to perform any actions.

Check to make sure that the same email address, including proper case, was used:

1. During registration
2. In the UI during installation
3. When logging in

2.1.4 Less Common Errors

1. If an older version of SlamData (3.x and older) is installed in a Virtual Machine (VM), it may require more than one CPU core before it will launch. If you are experiencing problems running an older version of SlamData in a VM, try increasing the number of cores and restarting.

2. In older versions of SlamData (3.x and older), an invalid database mount may prevent SlamData from starting. An invalid database mount could be a database that was previously available but is no longer available, credentials may have changed, port number changed, or any other configuration change that does not allow previously validated configurations to successfully connect.

2.2 Accessing SlamData

The default SlamData URL is `http://<servername>:20223`

Example: `http://localhost:20223`

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

SlamData's version will 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`

Example: `http://localhost:20223/server/info`

2.4 Running SlamData in the Cloud

When running SlamData with a hosting provider, such as Amazon EC2, the most common error encountered is a security policy misconfiguration. SlamData will need to connect to a data source over the same port as a standard database client.

A data source or database server and the SlamData server do not need to run on the same system.

Use the following checklist to ensure network problems are minimized.

1. Verify the security policy for the data source or database server is:
 - Accepting incoming connections from the SlamData server IP address.
 - Accepting incoming connections on the correct port.
2. If you are still unable to connect to your hosted data source or database system:
 - Verify that you can connect with a standard database client from any system.
 - Connect with a standard database client from the same system SlamData is running on.

CHAPTER 8

Indices and tables

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