Sensei

A Distributed Elastic Real-Time Searchable Database

Sensei Development Team

Sensei: A Distributed Elastic Real-Time Searchable Database

Sensei Development Team

Table of Contents

List of Figures

List of Examples

Chapter 1. Introduction

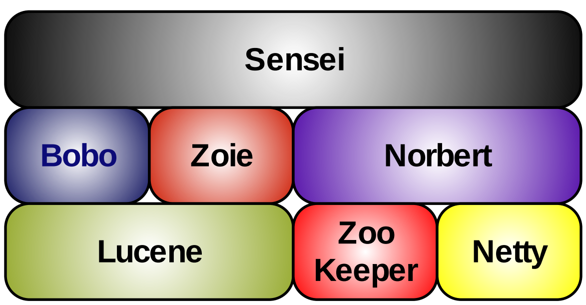
Sensei is an open-source, real-time, full-text searchable distributed database that is designed to handle the following type of queries:

```
SELECT f1,f2...fn FROM members
WHERE c1 AND c2 AND c3..
MATCH (fulltext query, e.g. "java engineer")
GROUP BY fx,fy,fz...
ORDER BY fa,fb...
LIMIT offset,count
```

Sensei is written in Java and is built on top of several other open-source software systems (see Figure 1.1, “Sensei and Its Foundation”):

- Bobo (http://sna-projects.com/bobo/): a faceted search implementation written in Java, using Lucene as the underlying search and indexing engine.
- Zoie (http://sna-projects.com/zoie/): a real-time search and indexing system built on Lucene.
- Lucene (http://lucene.apache.org/): a high-performance, full-featured text search engine library written entirely in Java.
- Norbert (http://sna-projects.com/norbert/): a library that provides easy cluster management and workload distribution. Norbert is built on ZooKeeper (http://zookeeper.apache.org/) and Netty (http://www.jboss.org/netty).

Figure 1.1. Sensei and Its Foundation



1.1. Design Considerations

As another NoSQL system (http://nosql-database.org/), Sensei is designed and built with the following considerations:

- **Data**
  - Fault tolerance - when one replication is down, data is still accessible
  - Durability - N copies of data is stored
  - Through-put - Parallelizable request-handling on different nodes/data replicas, designed to handle internet traffic
  - Consistency - Evently consistent
  - Data recovery - each shared/replica is noted with a watermark for data recovery
  - Large dataset - designed to handle 100s millions - billions of rows

- **Horizontally Scalable**
  - Data is partitioned - so work-load is also distributed
  - Elasticity - Nodes can be added to accomodate data growth

- Online expansion - Cluster can grow while handling online requests
- Online cluster management - Cluster topology can change while handling online requests
- Low operational/maintenance costs - Push it, leave it and forget it

- **Performance**
  - Low indexing latency - real-time update
  - Low search latency - millisecond query response time
  - Low volatility - low variance in both indexing and search latency

- **Customizability**
  - Plug-in framework - custom query handling logic
  - Routing factory - custom routing logic, default: round-robin
  - Index sharding strategy - different sharding strategy for different applications, e.g. time, mod etc.

1.2. Comparing to Traditional RDBMS

RDBMS:

- Vertically scaled
- Strong ACID guarantee
- Relational support
- Performance cost with full-text integration
- High query latency with large dataset, especially for operations like Group By
- Indexes needs to be built for all sort possibilities offline

Sensei:

- Horizontally scaled
- Relaxed consistency with high durability guarantees
- Data is streamed in, so atomicity and isolation is to be handled by the data producer
- Full-text search support
- Low query latency with arbitrarily large dataset
- Dynamic sorting, index is already built for all sortable fields and their combinations

1.3. Architecture

At a high level, a Sensei system consists of two parts: a cluster of Sensei *servers* (a.k.a. *search nodes*) and a cluster of Sensei *brokers*.

1. The cluster of Sensei servers

   Each server covers one or more *partitions* (or *shards*) of the entire index space, and is responsible for real-time indexing and searching on the partition(s) belonging to the node.

   > **Partition and Shard**
   >
   > A partition or shard is just a slice of the index served by the entire Sense cluster. Each partition may have multiple replicas handled by different Sensei search nodes. How data is partitioned (or sharded) is up to the business logic of the application, and the sharding strategy can be passed to Sensei as a plug-in.

2. The cluster of Sensei brokers

Brokers receive search requests from clients, pass them to selected servers in the Sensei search cluster, and then merge/return search results back to the clients.

Sensei brokers are relatively simple and lightweight. Most of the work is done at Sensei servers. A Sensei server plays two roles: indexer and searcher, both of which are implemented by Zoie and Bobo embedded in the node.

Conceptually, a Sensei node consists of the following four components:

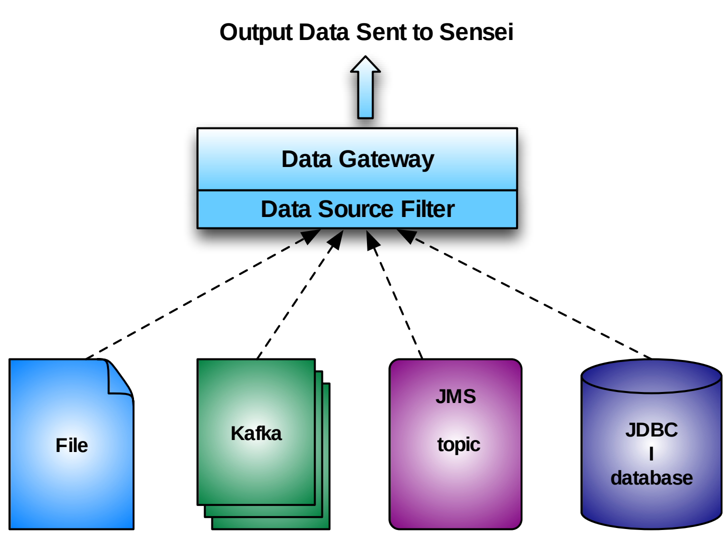
1. Data Gateway

   This is also simply called *gateway*, which is the component responsible for getting data from external sources and passing them to the indexer after an optional conversion. The input data to the gateway can be in different formats, while the output of the gateway has to be in the format acceptable by Zoie's streaming data provider. (By default, JSON is the format used by Sensei to communicate with Zoie.) The output data from the gateway also needs to match the table schema definition.

   Several types of built-in gateway are available in Sensei. They can be used to get data from common data sources including:
   - Line-based text file containing JSON objects.
   - Kafka (http://sna-projects.com/kafka/)
   - JMS
   - JDBC

   For each built-in data gateway, a filter can be plugged in to convert the original source data into the format defined by the table schema. (See Figure 1.2, “Sensei Data Gateway”.)

   Figure 1.2. Sensei Data Gateway



2. Indexing Manager

   Indexing manager acts as the bridge between the gateway and the Zoie system. It is responsible for passing data from the gateway to Zoie, controlling the pace of data consumption, and maintaining the index versions on the Sensei node.

3. Zoie System

This is the underlying system powering real-time indexing and search. Two types of Zoie systems are supported by Sensei today: regular Zoie and Hourglass.

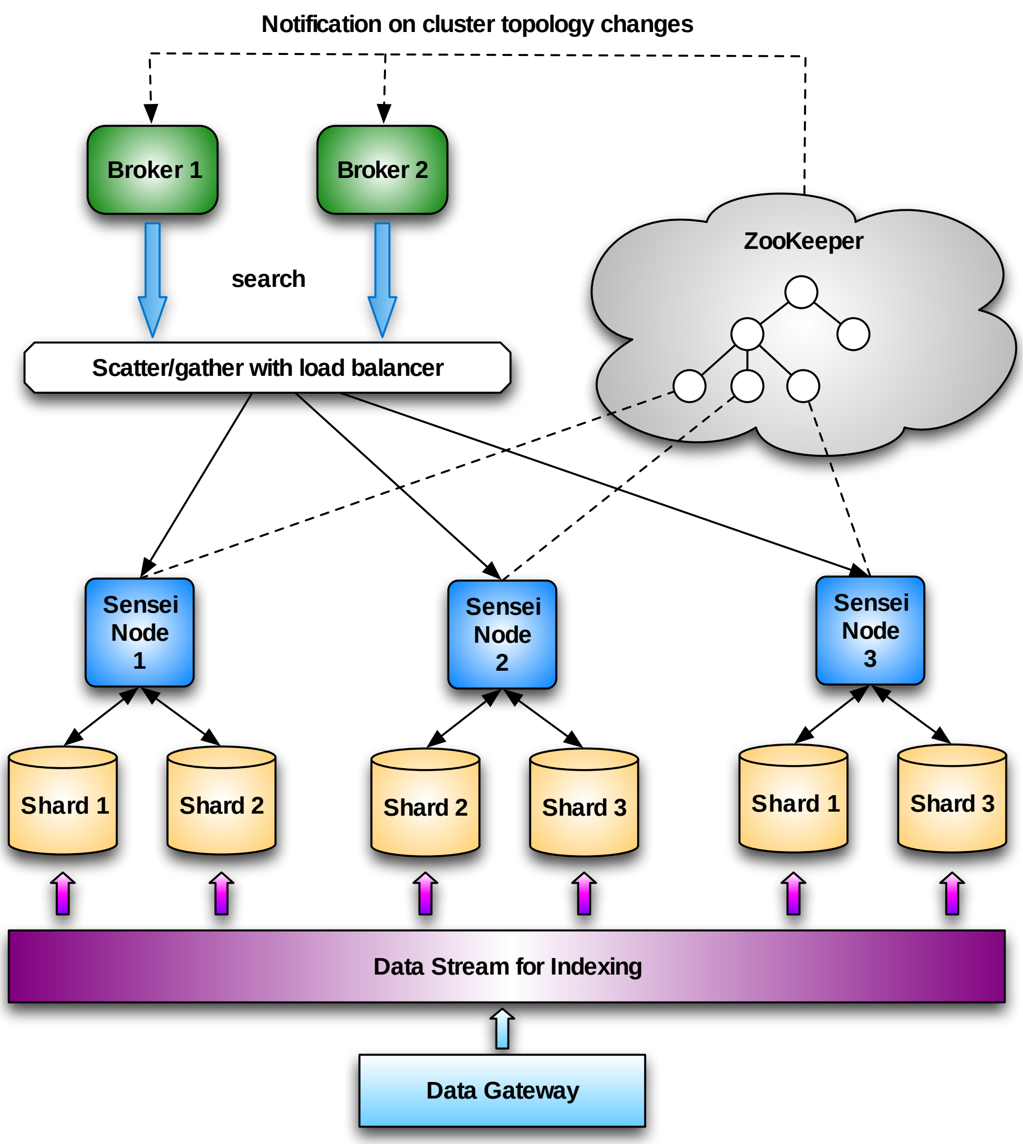
Hourglass (http://linkedin.jira.com/wiki/display/ZOIE/HourGlass+-+Forward-Rolling+Indexing) is a forward-rolling, append-only indexing system based on Zoie. It is used to power LinkedIn Signal (http://www.linkedin.com/signal).

4. Facet Handlers

Facet handlers are a key component in a Sensei server. They are the way we do faceted searches with Bobo. Without facet handlers, none of the column-based queries can be done in Sensei.

1.4. Architectural Diagram

Figure 1.3. Sensei Architectural Diagram



Chapter 2. Getting Started

2.1. Overview

- Real-time indexing/searching
- Cluster management
- Automatic data partitioning
- Support for structured and faceted search

2.2. Prerequisites

- Java 1.6 or higher
- maven 2.2.1 or higher
- ZooKeeper 3.2.0 or higher (http://hadoop.apache.org/zookeeper/)

2.3. Embedded Technologies

- Bobo (http://sna-projects.com/bobo/)
- Zoie (http://sna-projects.com/zoie/)
- Lucene (http://lucene.apache.org/)
- Norbert (http://sna-projects.com/norbert/)
- ZooKeeper (http://zookeeper.apache.org/)
- Spring (http://www.springsource.com/)

2.4. Details

2.4.1. Building Sensei

Getting the Sensei source code and building the entire system is straightforward. Only two commands are needed:

1. Checking out the source code from trunk:

   ```
   $ git clone git://github.com/javasoze/sensei.git sensei-trunk
   ```

2. Building Sensei using `ant`:

   ```
   $ ant
   ```

2.4.2. Starting ZooKeeper

In order to run the sample of Sensei search, you have to run an instance of ZooKeeper first.

You may download ZooKeeper from http://hadoop.apache.org/zookeeper/.

Using the sample configuration file in `zookeeper-3.2.0/conf` by copying `zookeeper-3.2.0/conf/zoo_sample.cfg` to `zookeeper-3.2.0/conf/zoo.cfg` and start an instance of ZooKeeper by running

```
$ zookeeper-3.2.0/bin/zkServer.sh start
```

For details, see http://hadoop.apache.org/zookeeper/docs/current/zookeeperStarted.html.

2.4.3. Starting Sensei Nodes

You can use command `bin/start-sensei-node.sh` to start a server node. This command takes one argument: `conf.dir`, which contains all configuration information for a given Sensei node.

Here is an example command-line that will work to fire up a single sensei node with some sample data:

```
$ bin/start-sensei-node.sh conf
```

Note

Do not expect to see any logs after running this command. If you run it, have ZooKeeper up and running, a REST server (as discussed below) will also be started, and you will be able to get some sample search results.

2.4.4. Web Application and RESTful End-Point

When developing applications using Sensei, we found it convenient to have a RESTful end-point that provides data in JSON format. Having a RESTful end-point allows one to query the Sensei system in an ad-hoc way, and having a JSON formatted output provides a way to investigate the result set without having the need to depend on jars or any other types of code binding.

When a Sensei node is started, a RESTful end-point along with a web interactive client would be started as well:

- RESTful end-point:

  ```
  http://localhost:8080/sensei?q=
  ```

- Web client

  ```
  http://localhost:8080
  ```

2.4.5. Starting Clients

After you start at least one node, you can run

```
$ bin/sensei-client.sh client-conf
```

to start a client. You can edit `client-conf/sensei-client.conf` to change the properties.

Type help to see command list:

```
$ bin/sensei-client.sh client-conf
> help
help - prints this message
exit - quits
info - prints system information
query <query string> - sets query text
facetspec <name>:<minHitCount>:<maxCount>:<sort> - add facet spec
page <offset>:<count> - set paging parameters
select <name>:<value1>,<value2>... - add selection, with ! in front of value indicates a not
sort <name>:<dir>,... - set sort specs
showReq: shows current request
clear: clears current request
clearSelections: clears all selections
clearSelection <name>: clear selection specified
clearFacetSpecs: clears all facet specs
clearFacetSpec <name>: clears specified facetspec
browse - executes a search
>
```

Chapter 3. Sensei Demo

3.1. Overview

We feel the best way to learn a new system is through examples.

Sensei comes with a sample application and this page aims to provide an anatomy of the Sensei car demo and to help new-comers in building a Sensei application.

File layout:

- Configuration files: `conf/*`
- Data file: `data/cars.json`
- Output index: `index/`
- Web-app: `src/main/webapp/`

3.2. Run the Demo

1. Build Sensei:

   ```
   $ ant
   ```

2. Make sure ZooKeeper is running:

   ```
   $ $ZK_home/bin/zkServer.sh start
   ```

3. Run the demo:

   ```
   $ ./bin/start-sensei-node conf/
   ```

3.3. URLs

- Sensei Web Client:

  ```
  http://localhost:8080/
  ```

- Demo Page:

  ```
  http://localhost:8080/demo.html
  ```

- RESTful End-Point:

  ```
  http://localhost:8080/sensei
  ```

3.4. Screenshots

Figure 3.1. Sensei Web Client

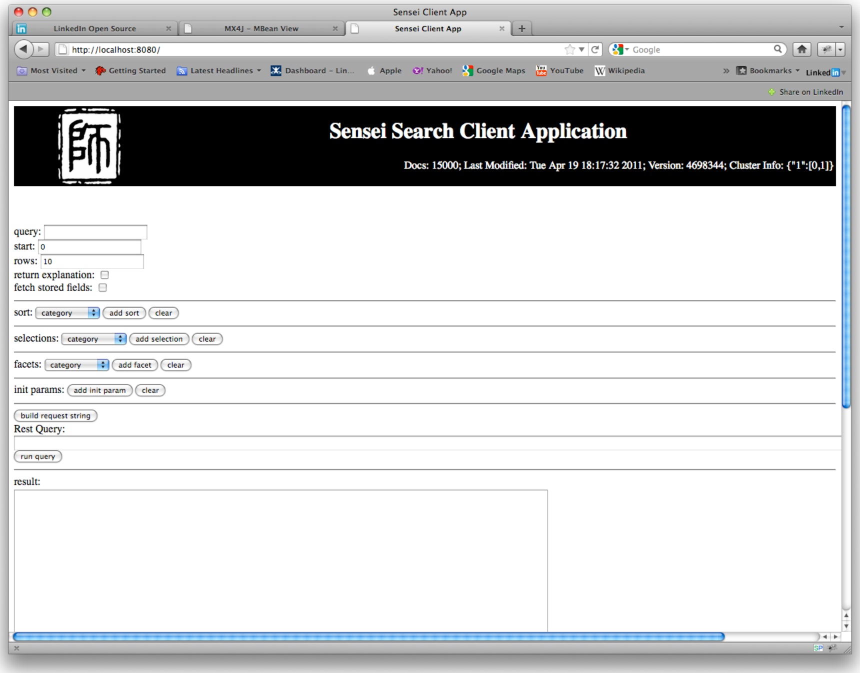


Figure 3.2. Demo

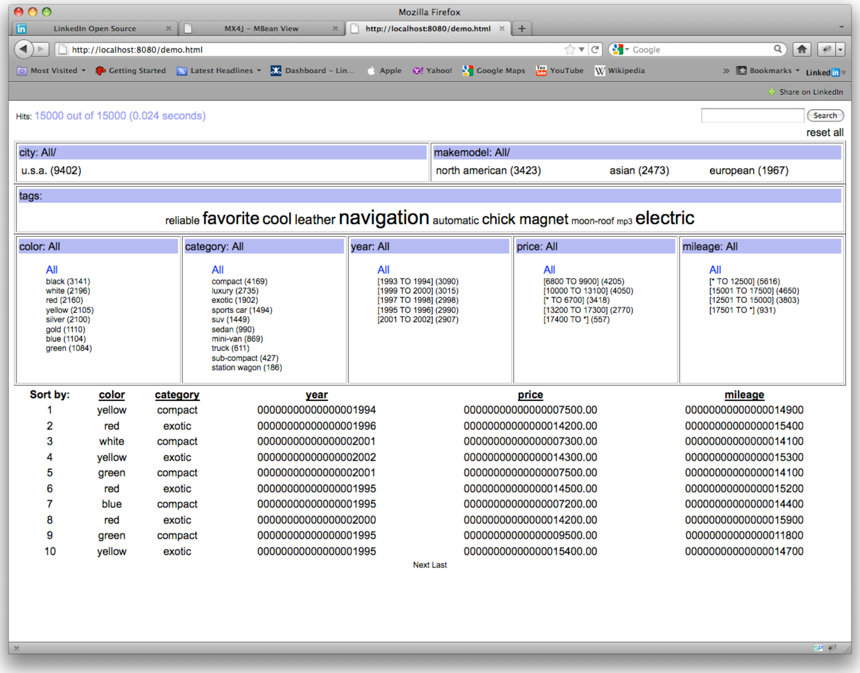
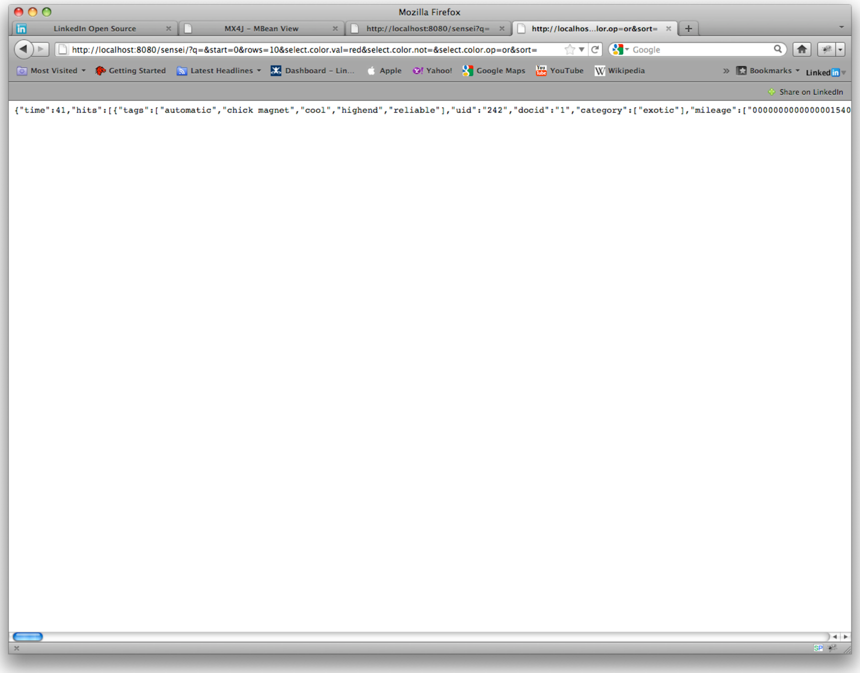
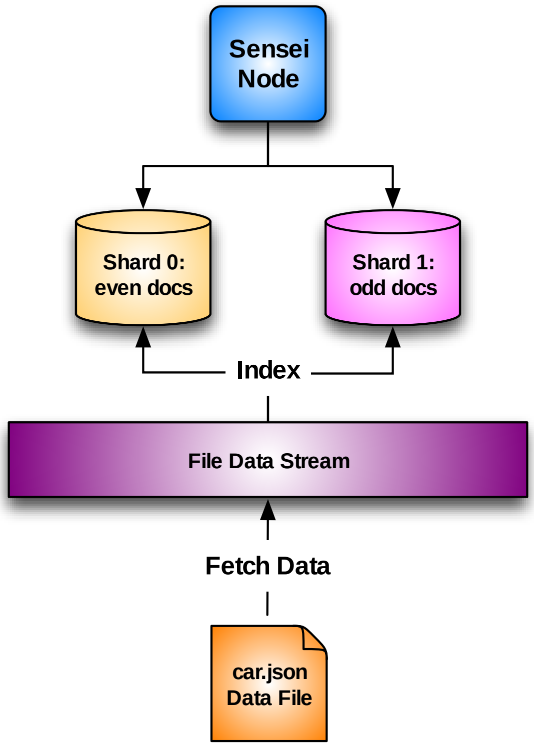


Figure 3.3. RESTful End-Point



3.5. Diagram

Figure 3.4. The Sensei Demo System



3.6. Demo Configuration

The configuration for the demo application can be found at: https://github.com/javasoze/sensei/tree/master/conf, which contains the following files:

- `sensei.properties`: main configuration file, describes the overall configuration of the system (we will dig into that in detail later)
- `schema.xml`: this describes the data model and the faceting behavior of the application
- `custom-facets.xml`: Spring file for adding custom facets, while Sensei comes with a set of facet types out of the box, this allows user-defined facets to be plugged into the application
- `plugins.xml`: any other plug-ins, e.g. Analyzer, Similarity, and Interpreter.

3.6.1. Data Model

The data model is described in `schema.xml`:

```
<table uid="id" delete-field="" skip-field="">
  <column name="color" type="string" />
  <column name="category" type="string" />
  <column name="city" type="string" />
  <column name="makemodel" type="string" />
```

```
    <column name="year" type="int" />
    <column name="price" type="float" />
    <column name="mileage" type="int" />
    <column name="tags" type="string" multi="true" delimiter=","/>
    <column name="contents" type="text" index="ANALYZED" store="NO" termvector="NO" />
  </table>

  <facets>
    <facet name="color" type="simple" depends=""/>
    <facet name="category" type="simple" />
    <facet name="city" type="path">
      <params>
        <param name="separator" value="/" />
      </params>
    </facet>
    <facet name="makemodel" type="path" />
    <facet name="year" type="range">
      <params>
        <param name="range" value="1993-1994" />
        <param name="range" value="1995-1996" />
        <param name="range" value="1997-1998" />
        <param name="range" value="1999-2000" />
        <param name="range" value="2001-2002" />
      </params>
    </facet>
    <facet name="mileage" type="range">
      <params>
        <param name="range" value="*-12500" />
        <param name="range" value="12501-15000" />
        <param name="range" value="15001-17500" />
        <param name="range" value="17501-*" />
      </params>
    </facet>
    <facet name="price" type="range">
         <params>
           <param name="range" value="*,6700" />
           <param name="range" value="6800,9900" />
           <param name="range" value="10000,13100" />
           <param name="range" value="13200,17300" />
           <param name="range" value="17400,*" />
        </params>
    </facet>
    <facet name="tags" type="multi" />
  </facets>
```

This corresponds to the following data table:

	color	**category**	**city**	**makemodel**	**year**	**price**	**mileage**	**tags**	**contents**
meta/ structured	yes	yes	yes	yes	yes	yes	yes	yes	no
type	string	string	string	string	int	float	int	string	text
facet	simple	simple	path	path	range	range	range	multi	N/A

Chapter 4. Indexing

Before it can be searched, all data has to be indexed first. Depending on where you get the source data from, you can use different data gateways to get the data converted to the format required by Sensei. No matter what gateway is used, you need to define the data model in file `schema.xml` (see Section 6.2, “Data Modeling” for more details), which is equivalent to the table definition in a RDBMS.

If the amount of data is small, and the run-time indexing and search rates are not high, then a single Sensei node may be all you need. If the amount of data is large and you cannot fit them into one box, however, you have to split them into multiple shards and store them on a cluster of Sensei nodes, each serving one or multiple shards. When a user query comes in, Sensei performs the search on all shards and merges the search results from all shards for you automatically.

If the run-time indexing or search rate is high, network bandwidth, memory or CPU may become the bottleneck on a Sensei server. In this case, sharding the index only is not enough. You have to *replicate* the index onto different nodes, and have multiple nodes share the indexing/search workload on the same shard(s).

In this chapter, we explain how to get the data indexed, and how to get the data sharded and replicated.

4.1. Data Acquisition

To get data indexed, the first thing to set up is the indexing manager (see `com.sensei.search.nodes.SenseiIndexingManager`). An indexing manager is responsible for:

- Initializing the Zoie system(s): one Zoie system is needed for every shard of index on one Sensei node.
- Building the data provider: a data provider needs to be built for the chosen data gateway.
- Starting and shutting down the data gateway.

For most of the cases you can simply use the default indexing manger provided by Sensei: `com.sensei.indexing.api.DefaultStreamingIndexingManager`. However you can always write your own version when it is needed.

The type of indexing manager is specified via configuration parameter `sensei.index.manager`, which is the bean ID of the indexing manager object that you use. When `sensei.index.manager` is not set, Sensei just uses `DefaultStreamingIndexingManager`.

Once the indexing manager is selected, the next thing to set up is the data gateway. What data gateway to use depends on how your original source data is stored. Sensei provides four types of built-in data gateways to cover the most common data sources (see Figure 1.2, “Sensei Data Gateway”), however you can also write your own version if needed.

The data gateway type is specified via configuration parameter `sensei.index.manager.<indexing-manager-type>.type`. Here `<indexing-manager-type>` is the bean ID of your indexing manager. If you use the default indexing manager, `DefaultStreamingIndexingManager`, `<indexing-manager-type>` should be `default`.

Additional configuration parameters may be needed for the data gateway you choose. These configuration parameters are named with the following prefix:

```
sensei.index.manager.<indexing-manager-type>.<data-gateway-type>
```

For example, if you use the default indexing manager and the Kafka data gateway, the following configuration parameters need to be specified:

```
sensei.index.manager.default.type = kafka
sensei.index.manager.default.kafka.host = my-kafka-host
sensei.index.manager.default.kafka.port = 1234
sensei.index.manager.default.kafka.topic = log-data
```

```
sensei.index.manager.default.kafka.batchsize = 10000
sensei.index.manager.default.kafka.filter = my-kafka-filter
```

In the rest of the document, we will use the default indexing manager in most of the examples, unless a different indexing manager type is specified explicitly.

4.2. Index Sharding

Index sharding is needed when the amount of data to be indexed is too big for one single machine to handle. Most of the times sharding is required because the disk space on a single machine is not big enough, but limited memory or limited CPU power can also be the reason.

Index sharding is controlled by the following two configuration parameters:

- `sensei.index.manager.<indexing-manager-type>.maxpartition.id`
- `sensei.node.partitions`

The first parameter tells Sensei how many shards in total the index will be divided into, and the second parameter tells Sensei how many and what shards are handled by the current node.

For example, if you want to divide the entire index into 10 shards (shard 0, shard 1, ..., shard 9), and you want to put the first two shards onto the first Sensei node, then you just need to add the following two lines to your configuration file:

```
sensei.index.manager.default.maxpartition.id = 9
sensei.node.partitions = 0,1
```

How to split your data into different shards is up to the business logic of the application. Sensei allows you to provide a *sharding strategy* plug-in (see `com.sensei.indexing.api.ShardingStrategy`) to let the indexing manager know what data should belong to which shards.

A simple but common sharding strategy is implemented in `com.sensei.indexing.api.ShardingStrategy`: `FieldModShardingStrategy`. This is basically the round-robin style. To make this sharding strategy work, you need to specify the total number of shards and on which data field the data should be sharded.

If data that does not belong to any partition on a Sensei node is passed to the indexing manager, it is discarded.

4.3. Index Replication

There are two reasons for index replication. First, index replication improves the reliability and fault-tolerance of the search service that you build. Second, by replicating the index and having multiple Sensei nodes handle the same shards of index, you can reduce the workload every Sensei node takes and thus improve the performance of the search service.

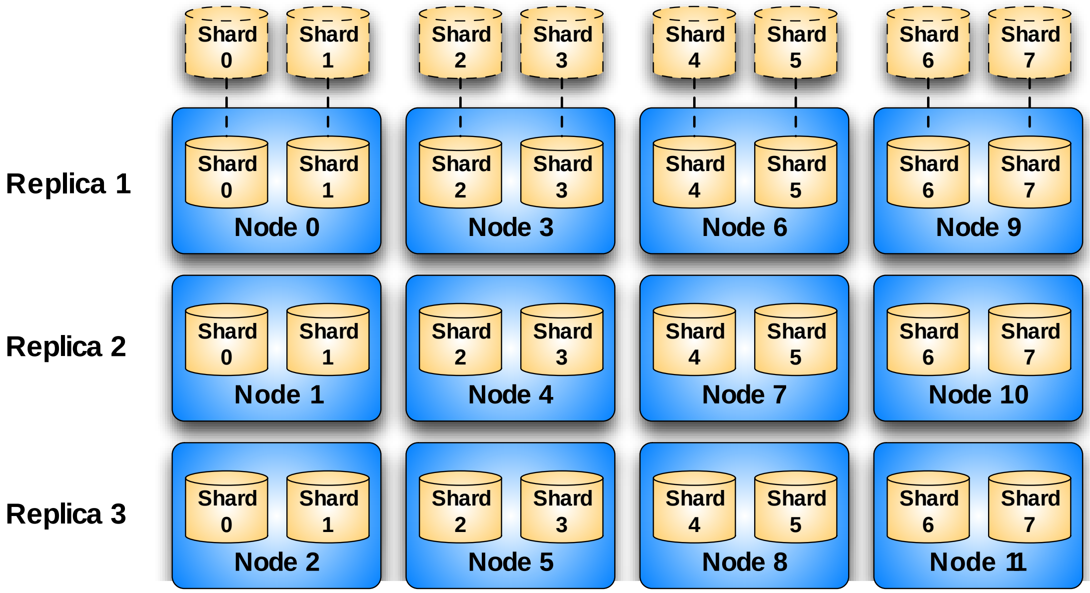
In Sensei, failovers of replicas are achieved by ZooKeeper and the built-in load balancer implemented based on consistent hashing. If one replica is down, the load balancer will be notified by ZooKeeper, and it will update the replica information maintained internally and route new search requests to a different replica.

It is very easy to add a Sensei node to an existing Sensei cluster and make it a replica of one or more index shards. Suppose you want to add one replica for a Sensei node, node 2, that contains two shards, all you need to do is copy over the config file `sensei.properties` on node 2 and change the node ID setting (`sensei.node.id`) to the node ID that you are going to assign to the newly added node.

4.4. Sensei Cluster Overview

The following figure shows an example of a Sensei cluster where the index is split into 8 shards, each having 3 replicas. There are 12 Sensei nodes in this cluster, and every node serves two shards.

Figure 4.1. Index Sharding and Replication



Chapter 5. Searching

Once the indexing process is started and some data has been indexed, you can start to perform searches against the Sensei cluster. A search query is done by building a Sensei request and sending it to a Sensei broker. You can do this using a native Java Sensei client, or other client APIs that support other languages like Python or JavaScript.

Upon receiving a request, a Sensei broker passes it to a set of selected Sensei nodes, merges the search results from these nodes, and then returns the merged results back to the requester. What set of Sensei nodes are selected for a request is decided using load balancer. Based on the need of the applications, you can plug in different customized load balancers. By default, Sensei uses a load balancer that distributes work load based on consistent hashing.

5.1. Sensei Request

A Sensei request contains the following main parameters:

- Query: the input query string
- Offset: the starting offset of search results
- Count: the number of search results to return
- Initializing parameters for runtime facet handlers: a map that contains the initializing parameters that are needed by all runtime facet handlers
- Flag indicating whether stored fields are to be fetched
- Partitions: shards of the index to be searched
- Flag indicating whether explanation information should be returned
- Routing parameter: the field value used for routing
- Group-by field: the field name used for the group-by operation (also called *field collapsing*)

5.2. Load Balancing

(To be finished)

5.3. RESTful API

5.3.1. Scalar Parameters

- **Query parameter:** `q`

  The `q` parameter specifies the query string.

- **Fetch stored fields:** `fetchstored`

  This is a boolean parameter that specifies whether stored fields should be fetched or not.

- **Show explanation:** `showexplain`

This is a boolean parameter that specifies whether query explanation information should be returned in the search result.

- **Starting offset:** `start`

  This is the offset value used for search result pagination.

- **Number of result:** `rows`

  This parameter specifies how many results should be returned for this query.

- **Routing parameter:** `routeparam`

  This parameter specifies the value of the field used for routing (i.e. load balancing).

- **Group-by parameter:** `groupby`

  This parameter specifies the field name used to do the group-by operation.

Example 5.1. Scalar Parameters

```
q=ipad 2&fetchstored=true&showexplain=false&start=0&rows=10
q=linkedin&start=0&rows=10&routeparam=12345&groupby=userid
```

5.3.2. Sort Fields Parameter

- **Sort fields parameter:** `sort`

  This parameter specifies how the search results should be sorted. The results can be sorted based on one or multiple fields, in either ascending or descending order. The value of this parameter consists of a list of comma separated strings, each of which can be one of the following values:

  - `relevance`: this means that the results should be sorted by scores in descending order.
  - `relrev`: this means that the results should be sorted by scores in ascending order.
  - `doc`: this means that the results should be sorted by doc ids in ascending order.
  - `docrev`: this means that the results should be sorted by doc ids in descending order.
  - `<field-name>:<direction>`: this means that the results should be sorted by field `<field-name>` in the direction of `<direction>`, which can be either `asc` or `desc`.

Example 5.2. Sort Fields Parameters

```
sort=relevance
sort=docrev
sort=price:desc,color=asc
```

5.3.3. Query Parameters

- **Query Parameters:** `qparam`

  This parameter specifies a list of arbitrary query parameters in the form of name-value pairs, separated by commas. Each pair is formatted as `<name>:<value>`.

  This parameter is commonly used to pass query-level parameters to dynamic runtime facet handlers.

Example 5.3. Query Parameters

```
qparam=arg1:123,arg2:abc
```

5.3.4. Selection Parameters

- **Not-value parameter:** `select.<field-name>.no`

  This parameter specifies what values for a field should be excluded from the search results.

- **Value parameter:** `select.<field-name>.val`

  This parameter specifies what values for a field should be included in the search results.

- **Operator parameter:** `select.<field-name>.op`

  This parameter specifies the operator used for selection values: `and` or `or`.

- **Property parameter:** `select.<field-name>.prop`

  This parameter specifies the additional properties if a selection is on a field with the *path* type.

Example 5.4. Selection Parameters

```
select.color.val=red,yellow&select.color.op=or&start=0&select.color.not=black
```

5.3.5. Facet Specification Parameters

- **Maximum count parameter:** `facet.<facet-name>.max`

  This parameter specifies the maximum count value for a facet with name `fact-name`.

- **Order-by parameter:** `facet.<facet-name>.order`

  This parameter specifies how facet values should be ordered:

  - `hits`: order-by hits
  - `val`: order-by values

- **Selection-expand parameter:** `facet.<facet-name>.expand`

- **Minimum hits parameter:** `facet.<facet-name>.minhit`

Example 5.5. Facet Specification Parameters

```
facet.color.minhit=1&facet.color.max=3&facet.color.order=val&facet.color.expand=True
```

Chapter 6. Sensei Configuration

6.1. Overview

A Sensei node is configured by a set of files. These files describe a Sensei node in terms of data models, server configuration, indexing tuning parameters, customizations, etc.

This chapter aims to describe how different parts are pieced together via these configuration files.

6.2. Data Modeling

Data models are described in the `schema.xml` file. The XSD definition of this XML file can be found from http://javasoze.github.com/sensei/schema/sensei-schema.xsd.

The schema file is composed by 2 sections:

1. Table schema
2. Facet schema

6.2.1. Table Schema

A Sensei instance can be viewed as a giant table with many columns and many rows. The concept of such table directly correlates to that of traditional RDBMS's.

A table may have the following attributes:

- **uid** (mandatory) - defines the name of the primary key field. This must be of type `long`.
- **delete-field** (optional) - defines the field that would indicate a delete event (we will get back to this later).
- **skip-field** (optional) - defines the field that would indicate a skipping event (we will get back to this later).
- **src-data-store** (optional) - defines the format of how the source data is saved. Currently the only supported value is "`lucene`".
- **src-data-field** (optional) - specifies the field name used to keep the original source data value.

> **Note**
>
> If this attribute is not specified, the default value is set to "`src_data`". If this field is not set by the data source filter, the string representation of the original data source is saved in this field by default. If part of the source data or a modified version of the source data is to be saved in the index, then you need to set this field using the value you prefer in the data source filter.

- **compress-src-data** (optional) - defines if the source data is compressed.

A table is also composed of a set of columns. Each column has a name and a type. Below is the list of supported types:

- **string** - value is a string, e.g. "`abc`"
- **int** - integer value
- **long** - long value
- **short** - short value
- **float** - a floating point value
- **double** - double value
- **char** - a character
- **date** - a date value, which must be accompanied by a format string to be used to parse a date string

- **text** - a searchable text segment, standard Lucene indexing specification can also be specified here, e.g. `index="ANALYZED"`, `termvector="NO"`.

A column that is not of type "text" is considered a *meta* column. Any meta column can be specified to be either *single* (default) or *multi*. When a column is specified to be *multi*, e.g. `multi="true"`, it means that, given a row, the column can have more than one value. A delimited string can be provided to help the indexer parse the values (default delimiter is `","`). To specify a different delimiter, say `":"`, we can simply set `delimiter=":"`)

Here is an example of the table schema (see https://github.com/javasoze/sensei/blob/master/conf/schema.xml):

```
<table uid="id" delete-field="" skip-field="">
  <column name="color" type="string" />
  <column name="category" type="string" />
  <column name="city" type="string" />
  <column name="makemodel" type="string" />
  <column name="year" type="int" />
  <column name="price" type="float" />
  <column name="mileage" type="int" />
  <column name="tags" type="string" multi="true" delimiter=","/>
  <column name="contents" type="text" index="ANALYZED"
          store="NO" termvector="NO"/>
</table>
```

6.2.1.1. JSON

By default, data objects inserted into Sensei are JSON objects.

Example:

Given the following table definition:

```
<table uid="id">
  <column name="color" type="string" />
  <column name="year" type="int" />
  <column name=tag" type="string" multi="true" />
  <column name="description" type="text" index="ANALYZED" store="NO" />
</table>
```

The following table shows as an example how a JSON object is mapped into the table:

JSON object

```
{
  id:1
  color:"red",
  year:2000,
  tag:"cool,leather",
  description:"i love this car"
}
```

Table view

id	color	year	tag	description
1	red	2000	cool, leather	i love this car

6.2.1.2. Deletes

To delete a row from Sensei, simply insert a data object with the specified delete-field set to true.

Example:

Given the table schema:

```
<table uid="id" delete-field="isDelete">
...
</table>
```

The following JSON object would delete the row where id=5:

```
{
  id:5,
  isDelete:"true"
}
```

6.2.1.3. Skips

In cases where runtime logic decides whether a data object should be skipped, the skip field can be useful.

Example:

Given the table schema:

```
<table uid="id" skip-field="isSkip">
...
</table>
```

The following JSON object would be skipped from indexing:

```
{
  id:7,
  isSkip:"true"
}
```

6.2.1.4. Source JSON

For many cases, you may want to save the original source data from which we extract all the fields into the index. You can do this by setting the attributes **src-data-store** and **src-data-field**.

6.2.2. Facet Schema

The second section is the facet schema, which defines how columns can be queried.

If we think of the table section defines how data is added into Sensei, then the facet section describes how these data can be queried.

The facet sections is composed of a set of facet definitions.

A facet definition requires a name and a type.

Possible types:

- **simple**: simplest facet, 1 row = 1 discrete value
- **path**: hierarchical facet, e.g. a/b/c
- **range**: range facet, used to support range queries
- **multi**: 1 row = N discrete values
- **compact-multi**: similar to multi, but possible values are limited to 32
- **histogram**: similar to a range facet, but a histogram facet automatically calculates the distribution of facet values over a predefined series of ranges with the same size. (A histogram facet depends on another numeric facet, and it requires several mandatory parameters, see Section 6.2.2.2.1, “Parameters for Histogram Facets”.
- **custom**: any user defined facet type

Example: https://github.com/javasoze/sensei/blob/master/conf/schema.xml

6.2.2.1. Optional Attributes

6.2.2.1.1. Column

The column attribute references the column names defined in the table section. By default, the value of the `name` attribute is used.

This can be useful if you want to name the facet name to be different from the defined column name, or if you want to have multiple facets defined on the same column.

6.2.2.1.2. Depends

This is a comma delimited string denoting a set of facet names this facet is to be depended on.

When attribute `depends` is specified, Sensei guarantees that the depended facets are loaded before this facet.

This is also how Composite Facets are constructed. (Another advanced topic).

6.2.2.1.3. Dynamic

Dynamic facets are useful when data layout is not known until query time.

Some examples:

- Searcher's social network
- Dynamic time ranges from when the search request is issued

This is another advanced topic to be discussed later.

6.2.2.2. Parameters

A facet can be configured via a list of parameters. Parameters are needed for a facet under some situations, for example:

- For path facets, separator strings can be configured
- For range facets, predefined ranges can be configured

The parameters can be specified via element `params`, which contains a list of elements called `param`. For each `param`, two attributes need to be specified: `name` and `value`.

How parameters are interpreted and used is dependent on the facet type.

Here is an example of a facet with a list of predefined ranges:

```
<facet name="year" type="range">
  <params>
    <param name="range" value="1993-1994"/>
    <param name="range" value="1995-1996"/>
    <param name="range" value="1997-1998"/>
    <param name="range" value="1999-2000"/>
    <param name="range" value="2001-2002"/>
  </params>
<facet>
```

6.2.2.2.1. Parameters for Histogram Facets

A histogram facet requires 5 parameters:

- **datatype**: the data type. Only the following 5 numeric data types are allowed:
  1. int
  2. short
  3. long
  4. float
  5. double

- **datahandler**: this is the name of the facet that the histogram facet depends on. The values of this facet are used to generate the distribution information.

- **start**: the minimum value of the facet.

- **end**: the maximum value of the facet.

- **unit**: the unit value used to divide facet values into ranges.

Here is an example configuration for a histogram facet over a facet called `score`:

```
<facet name="scoreHistogram" type="histogram">
  <params>
    <param name="datatype" value="int"/>
    <param name="datahandler" value="score"/>
    <param name="start" value="0"/>
    <param name="end" value="100"/>
    <param name="unit" value="10"/>
  </params>
</facet>
```

6.2.2.3. Customized Facets

We understand we cannot possibly cover all use cases using a short list of predefined facet handlers. It is necessary to allow users to define their own customized facets for different reasons.

If a customized facet handler is required for a column (or multiple columns), you can set the facet type to "`custom`", and declare a bean for the facet handler in file `custom-facets.xml`.

For example, if a customized facet called `time` is declared in `schema.xml` like this:

```
<facet name="time" type="custom" dynamic="false"/>
```

and the implementation of the facet handler is in class `com.example.facets.TimeFacetHandler`, then you should include the following line in file `custom-facets.xml`: [1]

```
<bean id="time" class="com.example.facets.TimeFacetHandler"/>
```

The id of the bean should match the name of the facet.

6.3. System Configuration

A Sensei node is configured via the `sensei.properties`, which uses the format supported by Apache Commons Configuration (http:/commons.apache.org/). This file consists of the following five parts:

1. **server**: port to listen on, rpc parameters, etc.
2. **cluster**: cluster manager, sharding, request routing, etc.
3. **indexing**: data interpretation, tokenization, indexer type, etc.
4. **broker and client**: e.g. entry into Sensei system

[1] Here we assume that the time facet handler does not take any arguments.

5. **plugins**: e.g. customized facet handlers

Below is the configuration file for the demo (available from https://github.com/javasoze/sensei/blob/master/conf/sensei.properties):

```
# sensei node parameters ❶
sensei.node.id=1
sensei.node.partitions=0,1

# sensei network server parameters
sensei.server.port=1234
sensei.server.requestThreadCorePoolSize=20
sensei.server.requestThreadMaxPoolSize=70
sensei.server.requestThreadKeepAliveTimeSecs=300

# sensei cluster parameters ❷
sensei.cluster.name=sensei
sensei.cluster.url=localhost:2181
sensei.cluster.timeout=30000

# sensei indexing parameters ❸
sensei.index.directory = index/cardata

sensei.index.batchSize = 10000
sensei.index.batchDelay = 300000
sensei.index.maxBatchSize = 10000
sensei.index.realtime = true
sensei.index.freshness = 10000

# index manager parameters

sensei.index.manager.default.maxpartition.id = 1
sensei.index.manager.default.type = file
sensei.index.manager.default.file.path = data/cars.json

# plugins: from plugins.xml ❹

# analyzer, default: StandardAnalyzer
# sensei.index.analyzer = myanalyzer

# similarity, default: DefaultSimilarity
# sensei.index.similarity = mysimilarity

# indexer type, zoie/hourglass/<custom name>

sensei.indexer.type=zoie

#extra parameters for hourglass

#sensei.indexer.hourglass.schedule

# retention
#sensei.indexer.hourglass.trimthreshold

# frequency for a roll, minute/hour/day
#sensei.indexer.hourglass.frequency

# sensei
# version comparator, default: ZoieConfig.DefaultVersionComparator
# sensei.version.comparator = myVersionComparator

# extra services
sensei.plugin.services =

# broker properties ❺
sensei.broker.port = 8080
```

```
sensei.broker.minThread = 50
sensei.broker.maxThread = 100
sensei.broker.maxWaittime = 2000

sensei.broker.webapp.path=src/main/webapp
sensei.search.cluster.name = sensei
sensei.search.cluster.zookeeper.url = localhost:2181
sensei.search.cluster.zookeeper.conn.timeout = 30000
sensei.search.cluster.network.conn.timeout = 1000
sensei.search.cluster.network.write.timeout = 150
sensei.search.cluster.network.max.conn.per.node = 5
sensei.search.cluster.network.stale.timeout.mins = 10
sensei.search.cluster.network.stale.cleanup.freq.mins = 10

# custom router factory
# sensei.search.router.factory = myRouterFactory
```

❶ This lines starts the server configurations.
❷ This lines starts the cluster configurations.
❸ This lines starts the indexing configurations.
❹ This lines starts the plugins configurations.
❺ This lines starts the broker and client configurations.

In the following sections, we are going to explain every configuration property in each part: what the property type is, whether it is required, what the default value is, and how it is used, etc.

6.3.1. Server Properties

sensei.node.id

- Type: int
- Required: Yes
- Default: None

This is the node ID of the Sensei node in a cluster.

sensei.node.partitions

- Type: String (comma separated integers or ranges)
- Required: Yes
- Default: None

This specifies the partitions IDs this the Sensei server is going to handle. Partition IDs can be given as either integer numbers or ranges, separated by commas. For example, the following line denotes that the Sensei server has six partitions: 1,4,5,6,7,10.

```
sensei.node.partitions=1,4-7,10
```

sensei.server.port

- Type: int
- Required: Yes
- Default: None

This is the Sensei server port number.

sensei.server.requestThreadCorePoolSize

- Type: int
- Required: No
- Default: 20

This is the core size of thread pool used to execute requests.

sensei.server.requestThreadKeepAliveTimeSecs

- Type: int
- Required: No
- Default: 300

This is the length of time in seconds to keep an idle request thread alive.

sensei.server.requestThreadMaxPoolSize

- Type: int
- Required: No
- Default: 70

This is the maximum size of thread pool used to execute requests.

6.3.2. Cluster Properties

sensei.cluster.name

- Type: String
- Required: Yes
- Default: None

This is the name of the Sensei server cluster.

sensei.cluster.timeout

- Type: int
- Required: No
- Default: 300000

This is the session timeout value, in milliseconds, that is passed to ZooKeeper.

sensei.cluster.url

- Type: String
- Required: Yes
- Default: None

This is the ZooKeeper URL for the Sensei cluster.

6.3.3. Indexing Properties

sensei.index.analyzer

See **sensei.index.analyzer** in Section 6.3.5, “Plug-in Properties”.

sensei.index.batchDelay

- Type: int
- Required: No
- Default: 300000

This is the maximum time to wait in milliseconds before flushing index events to disk. The default value is 300000 (i.e. 5 minutes).

sensei.index.batchSize

- Type: int
- Required: No

- Default: 10000

This is the batch size to control the pace of data event consumption on the back-end. It is the *soft* size limit of each event batch. If the events come in too fast and the limit is already reached, then the indexer will block the incoming events until the number of buffered events drop below this limit after some of the events are sent to the background data consumer.

sensei.index.custom

See **sensei.index.custom** in Section 6.3.5, “Plug-in Properties”.

sensei.index.directory

- Type: String
- Required: Yes
- Default: None

This is the directory used to save the index.

sensei.index.freshness

- Type: long
- Required: No
- Default: 500

This controls the freshness of entries in the index reader cache.

sensei.index.interpreter

See **sensei.index.interpreter** in Section 6.3.5, “Plug-in Properties”.

sensei.index.manager

See **sensei.index.manager** in Section 6.3.5, “Plug-in Properties”.

sensei.index.manager.default.batchSize

- Type: int
- Required: No
- Default: 1

This is the batch size to control when data events accumulated in the default index manger should be consumed by the data consumer. The default value is 1.

sensei.index.manager.default.eventsPerMin

- Type: int
- Required: No
- Default: 40000

This is the maximum number of data events that the indexer can consume per minute. If this threshold is exceeded, the indexer will pause for a short period of time before continuing to consume incoming data events.

This property is helpful in preventing the indexer from being overloaded. The default value is 40,000.

sensei.index.manager.default.maxpartition.id

- Type: int
- Required: Yes, if the default indexing manager is chosen; No, otherwise.
- Default: None

This is the maximum partition ID number served by this Sensei cluster if the default Sensei indexing manager is used.

Warning

This property is different from the total number of partitions in a Sensei cluster. For example, if a cluster contains 4 partitions, 0, 1, 2, and 3, then **sensei.index.manager.default.maxpartition.id** should be set to 3.

sensei.index.manager.default.shardingStrategy
See **sensei.index.manager.default.shardingStrategy** in Section 6.3.5, “Plug-in Properties”.

sensei.index.manager.default.type
See **sensei.index.manager.default.type** in Section 6.3.5, “Plug-in Properties”.

sensei.index.maxBatchSize
- Type: int
- Required: No
- Default: 10000

This is the maximum number of incoming data events that can be held by the indexer in a batch before they are flushed to disk. If this number is exceeded, the indexer will stop processing the data events for one minute.

sensei.index.realtime
- Type: boolean
- Required: No
- Default: true

This specifies whether the indexing mode is real-time or not.

sensei.index.similarity
See **sensei.index.similarity** in Section 6.3.5, “Plug-in Properties”.

sensei.indexer.type
- Type: String
- Required: Yes
- Default: None

This is the internal indexer type used by the Sensei cluster. Currently only two options are supported: `zoie` and `hourglass`. If `hourglass` is used, three more properties need to be set too:
1. **sensei.indexer.hourglass.schedule**
2. **sensei.indexer.hourglass.trimthreshold**
3. **sensei.indexer.hourglass.frequency**

sensei.indexer.hourglass.frequency
- Type: String
- Required: No
- Default: "`day`"

This is the rolling forward frequency. It has to be one of the following three values:
- `day`
- `hour`
- `minute`

sensei.indexer.hourglass.schedule
- Type: String
- Required: Yes, if property **sensei.indexer.type** is set to "`hourglass`"; No, otherwise.
- Default: None

This is a string that specifies Hourglass rolling forward schedule. The format of this string is "`ss mm hh`", meaning at `hh:mm:ss time` of the day that we roll forward for *daily* rolling. If it is *hourly* rolling, we roll forward at `mm:ss` time of the hour. If it is *minutely* rolling, we roll forward at `ss` seond of the minute.

sensei.indexer.hourglass.trimthreshold

- Type: int
- Required: No
- Default: 14

This is the retention period for how long we are going to keep the events in the index. The unit is the rolling period.

6.3.4. Broker and Client Properties

sensei.broker.maxThread

- Type: int
- Required: No
- Default: 50

This is the maximum size of thread pool used by a broker to execute requests.

sensei.broker.maxWaittime

- Type: int
- Required: No
- Default: 2000

This is the maximum idle time in milliseconds for a thread on a broker. Threads that are idle for longer than this period may be stopped.

sensei.broker.minThread

- Type: int
- Required: No
- Default: 20

This is the core size of thread pool used by the broker to execute requests.

sensei.broker.port

- Type: int
- Required: Yes
- Default: None

This is the port number of the Sensei broker.

sensei.broker.webapp.path

- Type: String
- Required: Yes
- Default: None

This is the resource base of the broker web application.

sensei.search.cluster.zookeeper.url

- Type: String
- Required: Yes
- Default: None

This is the ZooKeeper URL for the Sensei search cluster that a broker talks to.

sensei.search.cluster.name

- Type: String
- Required: Yes
- Default: None

This is the Sensei cluster name, i.e. the service name for the network clients and brokers.

sensei.search.cluster.zookeeper.conn.timeout

- Type: int
- Required: No
- Default: 10000

This is the ZooKeeper network client session timeout value in milliseconds.

sensei.search.cluster.network.conn.timeout

- Type: int
- Required: No
- Default: 1000

This is the maximum number of milliseconds to allow a connection attempt to take.

sensei.search.cluster.network.write.timeout

- Type: int
- Required: No
- Default: 150

This is the number of milliseconds a request can be queued for write before it is considered stale.

sensei.search.cluster.network.max.conn.per.node

- Type: int
- Required: No
- Default: 5

This is the maximum number of open connections to a node.

sensei.search.cluster.network.stale.timeout.mins

- Type: int
- Required: No
- Default: 10

This is the number of minutes to keep a request that is waiting for a response.

sensei.search.cluster.network.stale.cleanup.freq.mins

- Type: int
- Required: No
- Default: 10

This is the frequency to clean up stale requests.

6.3.5. Plug-in Properties

sensei.index.analyzer

- Type: String
- Required: No

- Default: ""

This specifies the bean ID of the analyzer plug-in for analyzing text. If not specified, `org.apache.lucene.analysis.standard.StandardAnalyzer` will be used.

sensei.index.similarity

- Type: String
- Required: No
- Default: ""

This specifies the bean ID of similarity plug-in for Lucene scoring. If not specified, `org.apache.lucene.search.DefaultSimilarity` is used.

sensei.index.custom

- Type: String
- Required: No
- Default: ""

This specifies the bean ID of the custom indexing pipeline implementation. A custom indexing pipeline can be plugged into the indexing process to allow users to modify generated Lucene documents at the last step before they are indexed.

A custom indexing pipeline has to implement interface `com.sensei.indexing.api.CustomIndexingPipeline`.

sensei.index.interpreter

- Type: String
- Required: No
- Default: ""

This specifies the bean ID of the interpretor of Zoie indexables. If not specified, `com.sensei.indexing.api.DefaultJsonSchemaInterpreter` is used.

sensei.index.manager

- Type: String
- Required: No
- Default: ""

This specifies the bean ID of the indexing manager object implementing `com.sensei.search.nodes.SenseiIndexingManager`. If not specified, `com.sensei.indexing.api.DefaultStreamingIndexingManager` is used.

sensei.index.manager.default.type

- Type: String
- Required: Yes if **sensei.index.manager** is not specified, i.e. the default indexing manager is used.
- Default: None

This specifies the type of gateway that will be used by the default indexing manager. The value identifies the bean ID of an object of `com.sensei.indexing.api.gateway.SenseiGateway`.

Several built-in gateways are provided by Sensei, but you can always define your own based on your need. No matter a built-in gateway or a custom gateway is used, additional parameters can be specified under the names with prefix **sensei.index.manager.default.<gateway-type>**.

Currently the following built-in gateway types are supported:

- `file`:

  This type of gateway takes a regular text file as the input. Each line in the file contains a data entry in JSON format.

  Only one property needs to be set for this gateway type. See Section 6.3.5.1, “File Gateway Properties”

- `kafka`:

  This type of gateway takes Kafka messages as input.

  See Section 6.3.5.2, “Kafka Gateway Properties” for additional property information.

- `jms`:

  This type of gateway takes JMS (Java Messages Service) messages as input. The publish-and-subscribe messaging model is used by Sensei, so parameters like topic need to be provided.

  See Section 6.3.5.3, “JMS Gateway Properties” for additional property information.

- `jdbc`:

  This type of gateway takes JDBC data as input.

  See Section 6.3.5.4, “JDBC Gateway Properties” for additional property information.

sensei.index.manager.default.<gateway-type>.filter

- Type: String
- Required: No
- Default: None

This is the bean ID of `com.sensei.indexing.api.DataSourceFilter` object. No matter what gateway type the indexing managers uses, a filter can be plugged in to get the original source data converted to the JSON format defined by the table schema. If the input data is already in the right format, then this filter is not needed.

sensei.index.manager.default.shardingStrategy

- Type: String
- Required: No
- Default: ""

This is the bean ID of the sharding strategy.

sensei.search.router.factory

- Type: String
- Required: No
- Default: ""

This is the bean ID of the Sensei request router factory. This factory builds the load balancer that is used by Sensei brokers to route incoming requests to different Sensei nodes.

sensei.version.comparator

- Type: String
- Required: No
- Default: ""

This specifies the bean ID of version comparator plug-in to be used by the indexer. If not specified, Zoie's default version comparator is used.

6.3.5.1. File Gateway Properties

For `file` gateway, the following property has to be specified:

sensei.index.manager.default.file.path

- Type: String
- Required: Yes
- Default: None

This is the path to the input data file.

6.3.5.2. Kafka Gateway Properties

For `kafka` gateway, the following properties should/can be specified: [2]

sensei.index.manager.default.kafka.batchsize

- Type: String
- Required: Yes
- Default: None

This is the batch size for each pull request.

sensei.index.manager.default.kafka.host

- Type: String
- Required: Yes
- Default: None

This is the host name of the Kafka server.

sensei.index.manager.default.kafka.port

- Type: int
- Required: Yes
- Default: None

This is the port number on which the Kafka server is listening for connections.

sensei.index.manager.default.kafka.timeout

- Type: int
- Required: Yes
- Default: 10000

This is the socket timeout in milliseconds.

sensei.index.manager.default.kafka.topic

- Type: String
- Required: Yes
- Default: None

The topic of the messages to be fetched.

6.3.5.3. JMS Gateway Properties

For `jms` gateway, the following properties should/can be specified:

[2] These properties are basically the parameters needed by the Kafka consumer API. The Simple Consumer API from Kafka is used by Sensei.

sensei.index.manager.default.jms.clientId

- Type: String
- Required: Yes
- Default: None

This is the client identifier used to connect to the JMS provider.

sensei.index.manager.default.jms.topic

- Type: String
- Required: Yes
- Default: None

This is the topic name that the JMS client subscribes to.

sensei.index.manager.default.jms.topicFactory

- Type: String
- Required: Yes
- Default: None

This is the bean ID of the `proj.zoie.dataprovider.jms.TopicFactory` object. This object is used to generate a topic object based on the given topic name.

sensei.index.manager.default.jms.connectionFactory

- Type: String
- Required: Yes
- Default: None

This is the bean ID of the `javax.jms.TopicConnectionFactory` object, which is used by the JMS client to create a `javax.jms.TopicConnection` object with the JMS provider.

6.3.5.4. JDBC Gateway Properties

For `jdbc` gateway, the following properties should/can be specified:

sensei.index.manager.default.jdbc.adaptor

- Type: String
- Required: Yes
- Default: None

This is the bean ID of the `com.sensei.indexing.api.jdbc.SenseiJDBCAdaptor` object. This object is used to build a `proj.zoie.dataprovider.jdbc.PreparedStatementBuilder` object, which is required by `proj.zoie.dataprovider.jdbc.JDBCStreamDataProvider`.

sensei.index.manager.default.jms.driver

- Type: String
- Required: Yes
- Default: None

This is the class name of the JDBC driver that you want to use.

sensei.index.manager.default.jms.password

- Type: String
- Required: Yes
- Default: None

This is the password for the user name that you use to connect to the database.

sensei.index.manager.default.jms.username

- Type: String
- Required: Yes
- Default: None

This is the user name that you use to connect to the database.

Chapter 7. BQL: Browsing Query Language

BQL stands for *Browsing Query Language*. It was originally proposed as a query language for Bobo. Now it is being developed and bundled with Sensei, and used by both Sensei and Sin.

7.1. BNF Grammar for BQL

7.1.1. BNF Notation

We use a modified BNF notation to describe BQL. The following table lists the meaning of all the meta-symbols we use.

Symbol	**Meaning**
|	Or. Choose one of the items.
[ ]	Enclose optional items.
*	Flags items that can be repeated 0 or more times.
( )	Group items so that they can be marked with one of the symbols: [ ], |, or *.

7.1.2. BQL Grammar

```
<statement> ::= ( <select_stmt> | <describe_stmt> ) [';']

<select_stmt> ::= SELECT <select_list> <from_clause> [<where_clause>] [<given_clause>]
                  [<additional_clauses>]

<describe_stmt> ::= ( DESC | DESCRIBE ) <index_name>

<select_list> ::= '*' | <column_name_list>

<column_name_list> ::= <column_name> ( ',' <column_name> )*

<from_clause> ::= FROM <index_name>

<where_clause> ::= WHERE <search_condition>

<search_condition> ::= <predicates>
                     | <cumulative_predicates>

<predicates> ::= <predicate> ( AND <predicate> )*

<predicate> ::= <in_predicate>
              | <contains_all_predicate>
              | <equal_predicate>
              | <not_equal_predicate>
              | <query_predicate>
              | <between_predicate>
              | <range_predicate>
              | <time_predicate>
              | <same_column_or_pred>

<in_predicate> ::= <column_name> [NOT] IN <value_list> [<except_clause>] [<predicate_props>]

<contains_all_predicate> ::= <column_name> CONTAINS ALL <value_list> [<except_clause>]
```

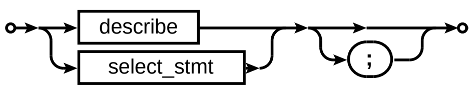
```
                             [<predicate_props>]

<equal_predicate> ::= <column_name> '=' <value> [<predicate_props>]

<not_equal_predicate> ::= <column_name> '<>' <value> [<predicate_props>]

<query_predicate> ::= QUERY IS <quoted_string>

<between_predicate> ::= <column_name> [NOT] BETWEEN <value> AND <value>

<range_predicate> ::= <column_name> <range_op> <numeric>

<time_predicate> ::= <column_name> IN LAST <time_span>
                   | <column_name> ( SINCE | AFTER | BEFORE ) <time_expr>

<same_column_or_pred> ::= '(' <cumulative_predicates> ')'

<cumulative_predicates> ::= <cumulative_predicate> ( OR <cumulative_predicate> )*

<cumulative_predicate> ::= <in_predicate>
                         | <equal_predicate>
                         | <between_predicate>
                         | <range_predicate>
                         | <time_predicate>

<value_list> ::= '(' <value> ( ',' <value> )* ')'

<value> ::= <quoted_string> | <numeric>

<range_op> ::= '<' | '<=' | '>=' | '>'

<except_clause> ::= EXCEPT <value_list>

<predicate_props> ::= WITH <prop_list>

<prop_list> ::= '(' <key_value_pair> ( ',' <key_value_pair> )* ')'

<key_value_pair> ::= <quoted_string> ':' <quoted_string>

<given_clause> ::= GIVEN FACET PARAM <facet_param_list>

<facet_param_list> ::= <facet_param> ( ',' <facet_param> )*

<facet_param> ::= '(' <facet_name> <facet_param_name> <facet_param_type> <facet_param_value> ')'

<facet_param_name> ::= <quoted_string>

<facet_param_type> ::= BOOLEAN | INT | LONG | STRING | BYTEARRAY | DOUBLE

<facet_param_value> ::= <quoted_string>

<additional_clauses> ::= ( <additional_clause> )+

<additional_clause> ::= <order_by_clause>
                      | <group_by_clause>
                      | <limit_clause>
                      | <browse_by_clause>
                      | <fetching_stored_clause>

<order_by_clause> ::= ORDER BY <sort_specs>

<sort_specs> ::= <sort_spec> ( ',', <sort_spec> )*

<sort_spec> ::= <column_name> [<ordering_spec>]

<ordering_spec> ::= ASC | DESC
```

```
<group_by_clause> ::= GROUP BY <group_spec>

<group_spec> ::= <facet_name> [TOP <max_per_group>]

<limit_clause> ::= LIMIT [<offset> ','] <count>

<offset> ::= ( <digit> )+

<count> ::= ( <digit> )+

<browse_by_clause> ::= BROWSE BY <facet_specs>

<facet_specs> ::= <facet_spec> ( ',' <facet_spec> )*

<facet_spec> ::= <facet_name> [<facet_expression>]

<facet_expression> ::= '(' <expand_flag> <count> <count> <facet_ordering> ')'

<expand_flag> ::= TRUE | FALSE

<facet_ordering> ::= HITS | VALUE

<fetching_stored_clause> ::= FETCHING STORED [<fetching_flag>]

<fetching_flag> ::= TRUE | FALSE

<quoted_string> ::= '"' ( <char> )* '"'
                  | "'" ( <char> )* "'"

<identifier> ::= <identifier_start> ( <identifier_part> )*

<identifier_start> ::= <alpha> | '-' | '_'

<identifier_part> ::= <identifier_start> | <digit>

<column_name> ::= <identifier>

<facet_name> ::= <identifier>

<alpha> ::= <alpha_lower_case> | <alpha_upper_case>

<alpha_upper_case> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O
                     | P | Q | R | S | T | U | V | W | X | Y | Z

<alpha_lower_case> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
                     | p | q | r | s | t | u | v | w | x | y | z

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<numeric> ::= <time_expr> | <number>

<number> ::= <integer> | <real>

<integer> ::= ( <digit> )+

<real> ::= ( <digit> )+ '.' ( <digit> )+

<time_expr> ::= <time_span> AGO
              | <date_time_string>
              | NOW

<time_span> ::= [<time_week_part>] [<time_day_part>] [<time_hour_part>]
                [<time_minute_part>] [<time_second_part>] [<time_millisecond_part>]

<time_week_part> ::= <integer> ( 'week' | 'weeks' )

<time_day_part>  ::= <integer> ( 'day'  | 'days' )
```

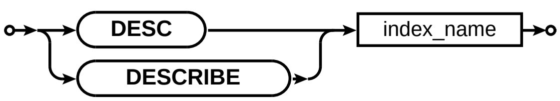
```
<time_hour_part> ::= <integer> ( 'hour' | 'hours' )

<time_minute_part> ::= <integer> ( 'minute' | 'minutes' | 'min' | 'mins')

<time_second_part> ::= <integer> ( 'second' | 'seconds' | 'sec' | 'secs')

<time_millisecond_part> ::= <integer> ( 'millisecond' | 'milliseconds' | 'msec' | 'msecs')

<date_time_string> ::= <digit><digit><digit><digit> ('-' | '/' | '.') <digit><digit>
                       ('-' | '/' | '.') <digit><digit>
                       <digit><digit> ':' <digit><digit> ':' <digit><digit>
```

7.2. Syntax Diagrams

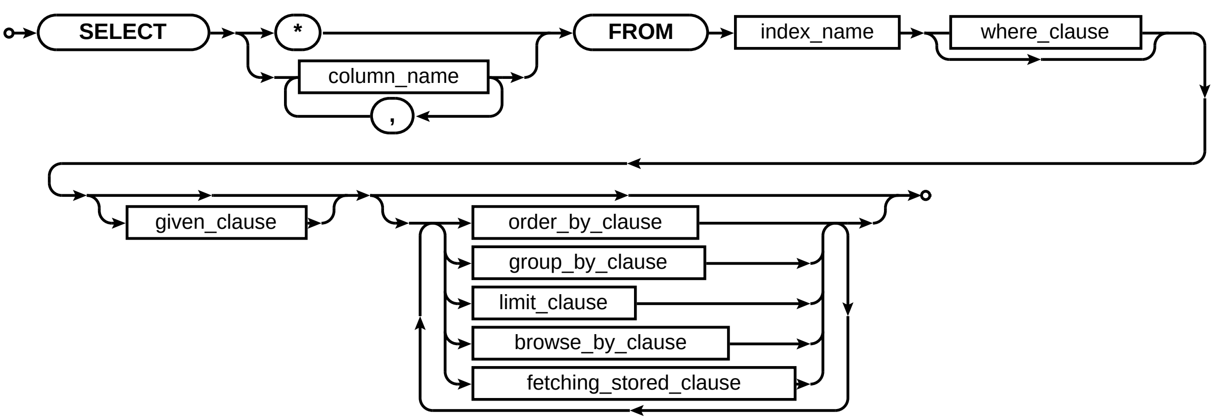
7.2.1. Top Level Statement



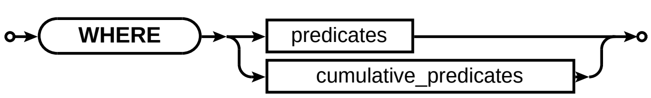
7.2.2. DESCRIBE Statement



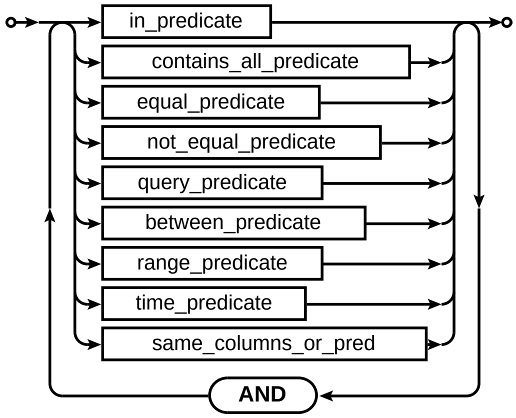
7.2.3. SELECT Statement



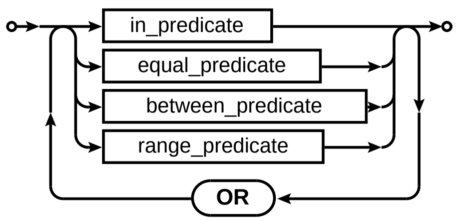
7.2.4. WHERE Clause



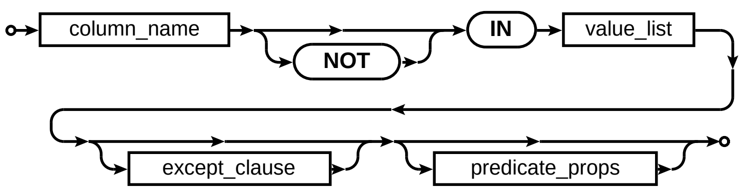
7.2.5. Predicates



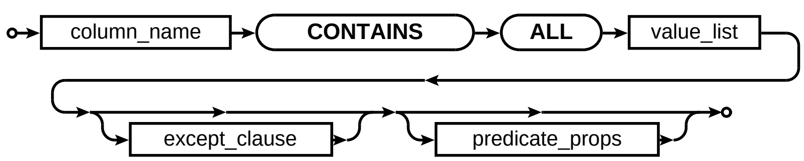
7.2.6. Cumulative Predicates



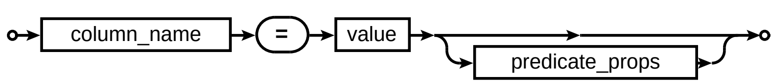
7.2.7. IN Predicate



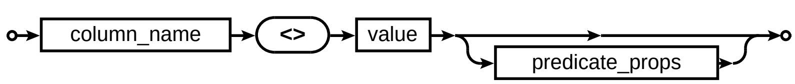
7.2.8. CONTAINS ALL Predicate



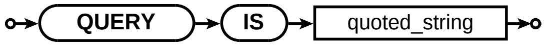
7.2.9. Equal Predicate



7.2.10. NOT equal Predicate

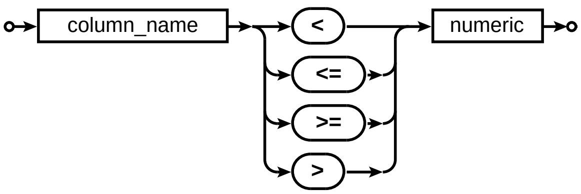


7.2.11. Query Predicate

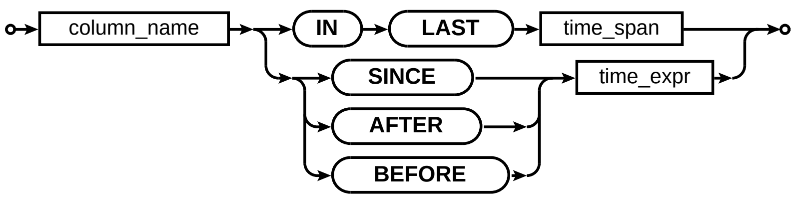


7.2.12. BETWEEN Predicate

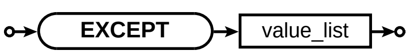
7.2.13. Range Predicate



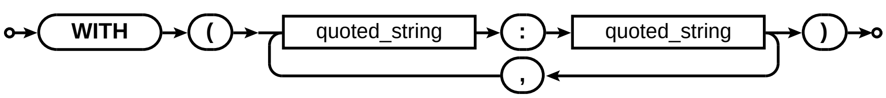
7.2.14. Time Predicate



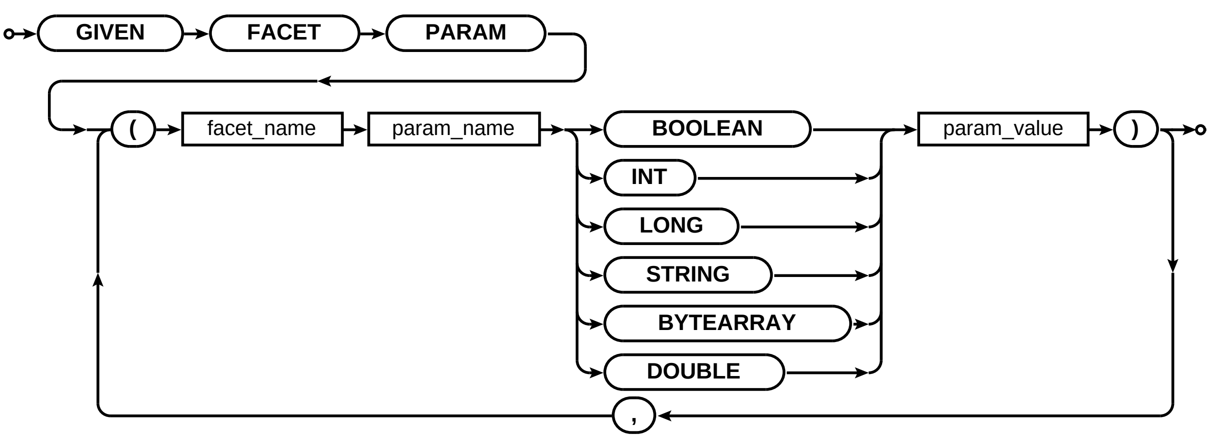
7.2.15. EXCEPT Clause



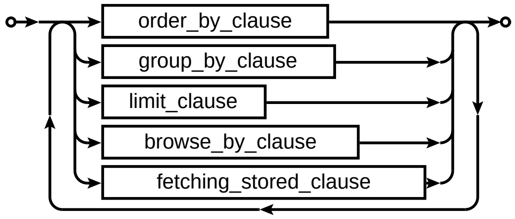
7.2.16. Predicate Properties



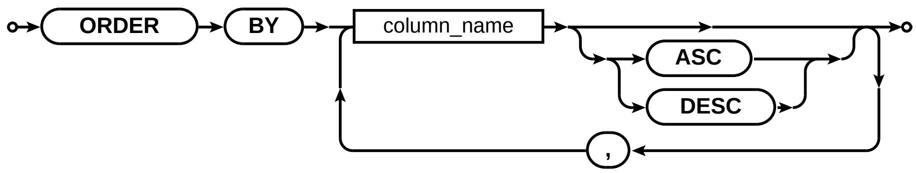
7.2.17. GIVEN Clause



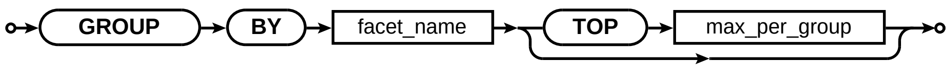
7.2.18. Additional Clauses



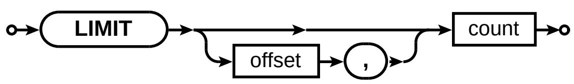
7.2.19. ORDER BY Clause



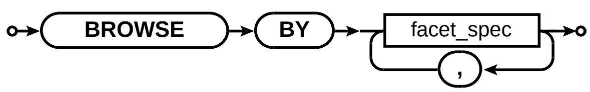
7.2.20. GROUP BY Clause



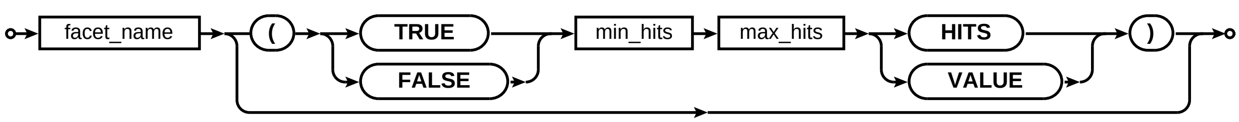
7.2.21. LIMIT Clause



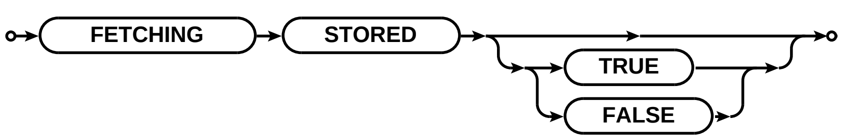
7.2.22. BROWSE BY Clause



7.2.23. Facet Specification



7.2.24. FETCHING STORED Clause



7.3. Query Examples

7.3.1. DESCRIBE Statement

Statement:

```
DESCRIBE cars;
```

Results:

```
+------------+------------+---------+-----------+-------------+---------+
| facet_name | facet_type | runtime | column    | column_type | depends |
+------------+------------+---------+-----------+-------------+---------+
| category   | simple     | false   | category  | string      | []      |
| groupid    | simple     | false   | groupid   | long        | []      |
| city       | path       | false   | city      | string      | []      |
| color      | simple     | false   | color     | string      | []      |
| tags       | multi      | false   | tags      | string      | []      |
| price      | range      | false   | price     | float       | []      |
| makemodel  | path       | false   | makemodel | string      | []      |
| mileage    | range      | false   | mileage   | int         | []      |
| year       | range      | false   | year      | int         | []      |
+------------+------------+---------+-----------+-------------+---------+
```

7.3.2. Basic Query

Find the first 10 cars in the index:

```
SELECT *
FROM cars;
```

This query searches for the top 10 cars, sorted by time, from index cars. All columns are selected. Number 10 is the default number of results to be returned, and time is the default order-by column.

Results:

```
+---------------+---------------------------------+ ... +-------+
| category      | city                            |     | uid   |
+---------------+---------------------------------+ ... +-------+
| station wagon | u.s.a./florida/tampa            |     | 14999 |
| compact       | u.s.a./florida/tampa            |     | 0     |
| exotic        | u.s.a./florida/tampa            |     | 242   |
| compact       | u.s.a./texas/houston            |     | 1     |
| compact       | u.s.a./california/sunnyvale     |     | 241   |
| compact       | u.s.a./texas/houston            |     | 2     |
| exotic        | u.s.a./california/san francisco |     | 244   |
| compact       | u.s.a./california/san francisco |     | 243   |
| compact       | u.s.a./california/sacramento    |     | 3     |
| compact       | u.s.a./california/san diego     |     | 4     |
+---------------+---------------------------------+ ... +-------+
10 rows in set, 15000 hits, 15000 total docs (server: 20ms, total: 68ms)
```

7.3.3. Queries with Limited Number of Results

You can use the limit clause in a query to specify the starting offset and number of documents to be returned in the results. By default, the starting offset is 0, and number of results to be returned is 10. If only one number is specified in the limit clause, then it is treated the number of results.

The limit clause is mainly used for pagination on search result age.

1. **Find the next 10 cars starting from offset 5:**

   ```
   SELECT *
   FROM cars
   LIMIT 5, 10
   ```

   Results:

   ```
   +----------+---------------------------------+ ... +-----+
   ```

```
| category | city                             |     | uid |
+----------+----------------------------------+ ... +-----+
| compact  | u.s.a./texas/houston             |     | 2   |
| exotic   | u.s.a./california/san francisco  |     | 244 |
| compact  | u.s.a./california/sacramento     |     | 3   |
| exotic   | china/beijing                    |     | 245 |
| compact  | u.s.a./california/san diego      |     | 4   |
| exotic   | u.s.a./new york/rochester        |     | 246 |
| compact  | china/shanghai                   |     | 5   |
| exotic   | u.s.a./new york/binghamton       |     | 247 |
| compact  | u.s.a./utah/salt lake city       |     | 6   |
| exotic   | china/shanghai                   |     | 248 |
+----------+----------------------------------+ ... +-----+
10 rows in set, 15000 hits, 15000 total docs (server: 10ms, total: 60ms)
```

2. **Just get the top 3 results:**

```
SELECT *
FROM cars
LIMIT 3
```

Results:

```
+----------+-----------------------------+ ... +-----+
| category | city                        |     | uid |
+----------+-----------------------------+ ... +-----+
| compact  | u.s.a./california/sunnyvale |     | 1   |
| compact  | u.s.a./florida/tampa        |     | 0   |
| exotic   | u.s.a./florida/tampa        |     | 242 |
+----------+-----------------------------+ ... +-----+
3 rows in set, 15001 hits, 15001 total docs (server: 7ms, total: 27ms)
```

7.3.4. Queries with Basic Conditions

1. **Find all red cars:**

```
SELECT color, year, makemodel
FROM cars
WHERE color = "red"
```

Results:

```
+-------+----------------------+-----------------------+
| color | year                 | makemodel             |
+-------+----------------------+-----------------------+
| red   | 00000000000000001996 | european/bentley/azure |
| red   | 00000000000000001995 | european/bentley/azure |
| red   | 00000000000000002000 | european/bentley/azure |
| red   | 00000000000000001995 | european/bentley/azure |
| red   | 00000000000000001994 | european/bentley/azure |
| red   | 00000000000000001995 | asian/acura/3.2tl      |
| red   | 00000000000000001996 | asian/acura/3.2tl      |
| red   | 00000000000000001999 | european/bentley/azure |
| red   | 00000000000000002002 | european/bentley/azure |
| red   | 00000000000000001996 | asian/acura/integra    |
+-------+----------------------+-----------------------+
10 rows in set, 2160 hits, 15000 total docs (server: 5ms, total: 41ms)
```

2. **Find all red and blue cars:**

```
SELECT color, year, makemodel
FROM cars
WHERE color in ("red", "blue")
```

Results:

```
+-------+---------------------+-----------------------+
| color | year                | makemodel             |
+-------+---------------------+-----------------------+
| blue  | 0000000000000001999 | asian/acura/tl        |
| red   | 0000000000000001996 | european/bentley/azure |
| blue  | 0000000000000001998 | asian/acura/tl        |
| red   | 0000000000000001995 | european/bentley/azure |
| blue  | 0000000000000001995 | asian/acura/1.6el     |
| red   | 0000000000000002000 | european/bentley/azure |
| blue  | 0000000000000001993 | asian/acura/3.2tl     |
| blue  | 0000000000000001998 | asian/acura/tl        |
| red   | 0000000000000001995 | european/bentley/azure |
| red   | 0000000000000001994 | european/bentley/azure |
+-------+---------------------+-----------------------+
10 rows in set, 3264 hits, 15000 total docs (server: 4ms, total: 33ms)
```

3. **Find all 1999 or 2000 cars that are not black nor red:**

```
SELECT color, year, makemodel
FROM cars
WHERE color not in ("black", "red")
  AND year BETWEEN 1999 AND 2000
```

Results:

```
+--------+---------------------+-----------------------+
| color  | year                | makemodel             |
+--------+---------------------+-----------------------+
| blue   | 0000000000000001999 | asian/acura/tl        |
| white  | 0000000000000001999 | asian/acura/1.6el     |
| yellow | 0000000000000001999 | asian/acura/3.2tl     |
| silver | 0000000000000002000 | asian/acura/3.5rl     |
| silver | 0000000000000002000 | asian/acura/3.5rl     |
| yellow | 0000000000000002000 | asian/acura/integra   |
| yellow | 0000000000000002000 | asian/acura/integra   |
| yellow | 0000000000000002000 | european/bentley/azure |
| yellow | 0000000000000002000 | european/bentley/azure |
| yellow | 0000000000000002000 | asian/acura/tl        |
+--------+---------------------+-----------------------+
10 rows in set, 1934 hits, 15000 total docs (server: 4ms, total: 35ms)
```

4. **Find all cars in New York state:**

```
SELECT color, city, price
FROM cars
WHERE city in ("u.s.a./new york")
```

Results:

```
+--------+---------------------------+------------------------+
| color  | city                      | price                  |
+--------+---------------------------+------------------------+
| white  | u.s.a./new york/albany    | 0000000000000007500.00 |
| red    | u.s.a./new york/rochester | 0000000000000014500.00 |
| green  | u.s.a./new york/syracuse  | 0000000000000009500.00 |
| yellow | u.s.a./new york/binghamton | 0000000000000007200.00 |
| blue   | u.s.a./new york/new york  | 0000000000000009300.00 |
| yellow | u.s.a./new york/new york  | 0000000000000015400.00 |
| yellow | u.s.a./new york/new york  | 0000000000000015200.00 |
| black  | u.s.a./new york/albany    | 0000000000000009200.00 |
| gold   | u.s.a./new york/new york  | 0000000000000011100.00 |
| red    | u.s.a./new york/rochester | 0000000000000009500.00 |
+--------+---------------------------+------------------------+
10 rows in set, 2781 hits, 15000 total docs (server: 5ms, total: 37ms)
```

7.3.5. Queries with AND, OR, and NOT Logic in Value Selections

1. **Find all cars tagged with both "`cool`" and "`hybrid`" but not "`favorite`":**

```
SELECT tags, price from cars
WHERE tags CONTAINS ALL ("cool", "hybrid") EXCEPT("favorite")
LIMIT 5
```

Results:

```
+--------------------------------+------------------------+
| tags                           | price                  |
+--------------------------------+------------------------+
| automatic,cool,hybrid,reliable | 00000000000000009400.00 |
| cool,hybrid,moon-roof,navigation | 00000000000000011500.00 |
| automatic,cool,hybrid,reliable | 00000000000000006300.00 |
| cool,hybrid,moon-roof,reliable | 00000000000000006500.00 |
| cool,hybrid,moon-roof,reliable | 00000000000000007100.00 |
+--------------------------------+------------------------+
5 rows in set, 491 hits, 15000 total docs (server: 9ms, total: 28ms)
```

2. **Find all cars tagged with either "`cool`" or "`hybrid`" but not "`mp3`"**:

```
SELECT tags, price
FROM cars
WHERE tags IN ("cool", "hybrid") EXCEPT ("mp3")
LIMIT 5
```

Results:

```
+--------------------------------------------+------------------------+
| tags                                       | price                  |
+--------------------------------------------+------------------------+
| hybrid,leather,moon-roof,reliable          | 00000000000000007500.00 |
| automatic,chick magnet,cool,highend,reliable | 00000000000000014200.00 |
| cool,electric,favorite,navigation          | 00000000000000007300.00 |
| cool,electric,favorite,reliable            | 00000000000000007200.00 |
| automatic,hybrid,leather,reliable          | 00000000000000007100.00 |
+--------------------------------------------+------------------------+
5 rows in set, 8176 hits, 15000 total docs (server: 6ms, total: 25ms)
```

7.3.6. Having Search Results Sorted

You can sort the search result based on one or more columns, in either ascending (the default) or descending order.

1. **Find the top 5 cheapest but newest cars.**

```
SELECT year, makemodel, price
FROM cars
ORDER BY year desc, price
LIMIT 5
```

Results:

```
+----------------------+------------------------------+------------------------+
| year                 | makemodel                    | price                  |
+----------------------+------------------------------+------------------------+
| 00000000000000002002 | asian/subaru/justy           | 00000000000000002100.00 |
| 00000000000000002002 | asian/subaru/justy           | 00000000000000002100.00 |
| 00000000000000002002 | north american/dodge/colt    | 00000000000000002400.00 |
| 00000000000000002002 | north american/mercury/tracer | 00000000000000002400.00 |
```

```
| 000000000000000002002 | north american/mercury/tracer | 000000000000000002500.00 |
+-----------------------+-------------------------------+--------------------------+
5 rows in set, 15000 hits, 15000 total docs (server: 22ms, total: 50ms)
```

2. **Find the top 5 most expensive but oldest cars:**

```
SELECT year, makemodel, price
FROM cars
ORDER BY year asc, price desc
LIMIT 5
```

Results:

```
+-----------------------+-----------------------------+--------------------------+
| year                  | makemodel                   | price                    |
+-----------------------+-----------------------------+--------------------------+
| 000000000000000001993 | european/ferrari/360 modena | 000000000000000019500.00 |
| 000000000000000001993 | asian/acura/nsx             | 000000000000000019500.00 |
| 000000000000000001993 | european/aston martin/db7   | 000000000000000019500.00 |
| 000000000000000001993 | european/ferrari/360 modena | 000000000000000019500.00 |
| 000000000000000001993 | asian/acura/nsx             | 000000000000000019500.00 |
+-----------------------+-----------------------------+--------------------------+
5 rows in set, 15000 hits, 15000 total docs (server: 5ms, total: 27ms)
```

7.3.7. Queries with Full Text Search

In the WHERE clause, you can add a condition for full text search, which is called a (text) query. This condition is to find the documents that contain matching text. You can use Lucene Query Syntax in the text string that you search on.

1. **Find all the cars that are tagged with "hybrid" and "navigation":**

```
SELECT tags, makemodel
FROM cars
WHERE QUERY IS "hybrid AND navigation"
```

Results:

```
+-------------------------------------+-----------------------------+
| tags                                | makemodel                   |
+-------------------------------------+-----------------------------+
| hybrid,leather,moon-roof,navigation | asian/acura/tl              |
| favorite,hybrid,mp3,navigation      | asian/acura/tl              |
| favorite,hybrid,mp3,navigation      | asian/acura/3.2tl           |
| favorite,hybrid,mp3,navigation      | asian/acura/tl              |
| cool,hybrid,moon-roof,navigation    | asian/acura/tl              |
| cool,favorite,hybrid,navigation     | asian/acura/vigor           |
| automatic,cool,hybrid,navigation    | asian/acura/vigor           |
| cool,hybrid,moon-roof,navigation    | asian/acura/vigor           |
| cool,hybrid,moon-roof,navigation    | north american/asuna/se/gt  |
| automatic,hybrid,leather,navigation | european/saab/900           |
+-------------------------------------+-----------------------------+
10 rows in set, 778 hits, 15000 total docs (server: 186ms, total: 209ms)
```

2. **A more complicated example:**

```
SELECT color, tags, city
FROM cars
WHERE QUERY IS "(hybrid OR mooon-roof) AND mp3 NOT cool NOT navigation"
  AND city in ("u.s.a./new york")
  AND color = "red"
```

Results:

```
+-------+-------------------------------+-----------------------------+
| color | tags                          | city                        |
```

```
+-------+------------------------------+---------------------------+
| red   | automatic,hybrid,mp3,reliable | u.s.a./new york/new york   |
| red   | hybrid,moon-roof,mp3,reliable | u.s.a./new york/buffalo    |
| red   | favorite,hybrid,mp3,reliable  | u.s.a./new york/syracuse   |
| red   | favorite,hybrid,mp3,reliable  | u.s.a./new york/buffalo    |
| red   | automatic,hybrid,mp3,reliable | u.s.a./new york/syracuse   |
| red   | favorite,hybrid,mp3,reliable  | u.s.a./new york/binghamton |
| red   | automatic,hybrid,mp3,reliable | u.s.a./new york/syracuse   |
+-------+------------------------------+---------------------------+
7 rows in set, 7 hits, 15000 total docs (server: 17ms, total: 44ms)
```

7.3.8. Queries with Group By

1. You can group the search results of a query by one facet, which can be a simple column or a facet built upon a group of columns. You can also specify how many hits you want to keep in each group (the default is 10).

   Find the first 10 groups of cars, with 2 hits in each group:

   ```
   SELECT category, city, makemodel
   FROM cars
   GROUP BY category TOP 2
   ```

   Results:

   ```
   ========================================================================================
   | category      | city                             | makemodel                          |
   ========================================================================================
   | compact       | u.s.a./texas/houston             | asian/acura/tl                     |
   | compact       | u.s.a./florida/tampa             | asian/acura/1.6el                  |
   +---------------+----------------------------------+------------------------------------+
   | exotic        | u.s.a./florida/tampa             | european/bentley/azure             |
   | exotic        | u.s.a./california/san francisco | european/bentley/azure             |
   +---------------+----------------------------------+------------------------------------+
   | luxury        | u.s.a./florida/orlando           | asian/acura/3.5rl                  |
   | luxury        | u.s.a./new york/rochester        | asian/acura/3.5rl                  |
   +---------------+----------------------------------+------------------------------------+
   | sports car    | u.s.a./california/sunnyvale      | asian/acura/integra                |
   | sports car    | u.s.a./texas/austin              | asian/acura/integra                |
   +---------------+----------------------------------+------------------------------------+
   | sedan         | china/beijing                    | north american/eagle/vision        |
   | sedan         | australia/perth                  | north american/eagle/vision        |
   +---------------+----------------------------------+------------------------------------+
   | suv           | china/shanghai                   | north american/ford/bronco         |
   | suv           | u.s.a./california/sacramento     | north american/ford/bronco         |
   +---------------+----------------------------------+------------------------------------+
   | van           | u.s.a./florida/palm beach        | north american/ford/club wagon     |
   | van           | u.s.a./new york/albany           | north american/ford/club wagon     |
   +---------------+----------------------------------+------------------------------------+
   | truck         | u.s.a./utah/provo                | asian/mazda/b-series               |
   | truck         | australia/melbourn               | asian/isuzu/pickup                 |
   +---------------+----------------------------------+------------------------------------+
   | station wagon | u.s.a./texas/dallas              | north american/saturn/sw           |
   | station wagon | u.s.a./california/san jose       | north american/saturn/sw           |
   +---------------+----------------------------------+------------------------------------+
   | mini-van      | u.s.a./texas/austin              | north american/chevrolet/astro van |
   | mini-van      | u.s.a./california/san jose       | north american/chevrolet/astro van |
   +---------------+----------------------------------+------------------------------------+
   ========================================================================================
   10 groups in set, 15000 hits, 15000 total docs (server: 55ms, total: 130ms)
   ```

2. **Find the numbers of cars in different categories:**

   ```
   SELECT category, grouphitscount
   FROM cars
   GROUP BY category top 1
   ```

Results:

```
+---------------+----------------+
| category      | grouphitscount |
+---------------+----------------+
| compact       | 4169           |
| exotic        | 1902           |
| luxury        | 2735           |
| sports car    | 1494           |
| sedan         | 990            |
| suv           | 1449           |
| van           | 168            |
| truck         | 611            |
| station wagon | 186            |
| mini-van      | 869            |
+---------------+----------------+
10 rows in set, 15000 hits, 15000 total docs (server: 3ms, total: 38ms)
```

7.3.9. Getting Facet Information Using Browse By Clause

BQL is designed to support faceted search, so we have to make it possible to get facet information along with the search results. This can be done using the Browse By clause, where you can specify one or more facets for which you want to get the facet count information. For each facet in the Browse By clause, you can optionally include

- whether the selection is expanded (default false)
- the minimum number of hit counts (default 1)
- the maximum number of hit counts (default 10)
- facet ordering method ("hits" or "value") (default "hits")

Here is an example:

Query:

```
SELECT color, year, tags, price
FROM cars
WHERE QUERY IS "cool"
  AND tags CONTAINS ALL ("cool", "hybrid") EXCEPT ("favorite")
  AND color in ("red")
ORDER BY price desc
LIMIT 0,10
BROWSE BY color(true, 1, 10, hits), year(true, 1, 10, value), price
```

Results:

```
+-------+----------------------+----------------------------------+-------------------------+
| color | year                 | tags                             | price                   |
+-------+----------------------+----------------------------------+-------------------------+
| red   | 00000000000000002000 | cool,hybrid,moon-roof,navigation | 00000000000000014500.00 |
| red   | 00000000000000001993 | cool,hybrid,moon-roof,navigation | 00000000000000014400.00 |
| red   | 00000000000000002002 | automatic,cool,hybrid,navigation | 00000000000000014200.00 |
| red   | 00000000000000001998 | automatic,cool,hybrid,navigation | 00000000000000012100.00 |
| red   | 00000000000000002002 | automatic,cool,hybrid,reliable   | 00000000000000011500.00 |
| red   | 00000000000000002002 | automatic,cool,hybrid,reliable   | 00000000000000011400.00 |
| red   | 00000000000000001998 | automatic,cool,hybrid,reliable   | 00000000000000011400.00 |
| red   | 00000000000000001996 | automatic,cool,hybrid,reliable   | 00000000000000011200.00 |
| red   | 00000000000000001999 | automatic,cool,hybrid,reliable   | 00000000000000011100.00 |
| red   | 00000000000000002001 | cool,hybrid,moon-roof,reliable   | 00000000000000010500.00 |
+-------+----------------------+----------------------------------+-------------------------+
10 rows in set, 59 hits, 15000 total docs (server: 337ms, total: 372ms)
+-------------+
```

```
| color       |
+-------------+
| white  (73) |
| yellow (73) |
| blue   (62) |
| silver (61) |
| red    (59) |
| green  (58) |
| gold   (53) |
| black  (52) |
+-------------+
+----------------------+
| price                |
+----------------------+
| [6800 TO 9900]   (27) |
| [* TO 6700]      (21) |
| [10000 TO 13100] (8)  |
| [13200 TO 17300] (3)  |
+----------------------+
+---------------------+
| year                |
+---------------------+
| [1993 TO 1994] (16) |
| [1995 TO 1996] (13) |
| [1997 TO 1998] (10) |
| [1999 TO 2000] (9)  |
| [2001 TO 2002] (11) |
+---------------------+
```

7.3.10. Queries with Run-Time Facet Handler Initialization Parameters

In a faceted search system, a run-time facet handler usually requires initialization parameter(s) to be provided at search time for each query. For example, in a search system that searches LinkedIn shares or Twitter tweets, one or more run-time facets usually require the searcher to provide his/her user Id (or user name) and the time when the search is performed.

One run-time facet may need multiple initialization parameters, each of which has a different name and/or a different data type. These parameters can be specified in the SELECT statement using the GIVEN FACET PARAM clause. Every parameter is specified in the clause as a 4-tuple, (*facet-name*, *param-name*, *param-type*, *param-value*).

1. **On a search system for LinkedIn shares, find recent updates from member 12345678 himself (i.e. value 0 for the Network facet) and all his first degree connections (i.e. value 1 for the Network facet) in US:**

   ```
   SELECT uid, Network, userid, country
   FROM shares
   WHERE country = "us"
     AND Network in (0, 1)
   GIVEN FACET PARAM (Network, "member_id", int, 12345678)
   ```

   Results:

   ```
   +---------------------+---------+---------------------+---------+
   | uid                 | Network | userid              | country |
   +---------------------+---------+---------------------+---------+
   | 5527797854963249152 | 0       | 0000000000012345678 | us      |
   | 5527805402646839296 | 1       | 0000000000042593551 | us      |
   | 5527816561408086016 | 0       | 0000000000012345678 | us      |
   | 5527825082430267392 | 1       | 0000000000022593551 | us      |
   | 5527829323551084544 | 0       | 0000000000012345678 | us      |
   | 5527848889647902720 | 1       | 0000000000004730909 | us      |
   | 5527853965330358272 | 1       | 0000000000004730909 | us      |
   | 5527884781573898240 | 1       | 0000000000026325826 | us      |
   ```

```
| 5527487070454427648 | 0       | 00000000000012345678 | us      |
| 5527488521884930048 | 0       | 00000000000012345678 | us      |
+---------------------+---------+----------------------+---------+
10 rows in set, 14 hits, 10749644 total docs (server: 432ms, total: 472ms)
```

Chapter 8. FAQ

8.1. About Sensei

8.1.1. Why is it called Sensei?

Sensei means *teacher* or *professor* in Japanese (http://en.wikipedia.org/wiki/Sensei). It shares the same pronunciation and writing with the Chinese word that has the same meaning. This name indicates that the system can be used in place of Oracle database in many applications.

8.2. Sensei Configuration

(To Be Finished)

8.3. Problems Running Sensei

(To Be Finished)

Index