

BlackRay 0.10.0 Manual



The BlackRay Development Team

BlackRay 0.10.0 Manual

The BlackRay Development Team

Copyright © 2004-2009 SoftMethod GmbH

License Information

This document is licensed under the Creative Commons Attribution-Share Alike 3.0 License.



The following is a human-readable summary of the legal code, which can be obtained from <http://creativecommons.org/licenses/by-sa/3.0/legalcode>

You are free:



to Share — to copy, distribute and transmit the work



to Remix — to adapt the work

Under the following conditions:



Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

With the understanding that:

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the web page.

Trademarks

In this document, all trademark references are made without the TM or ® signs commonly used in the United States. SoftMethod, nexgenda, and BlackRay are registered trademarks of SoftMethod GmbH. Windows is a registered trademark of Microsoft Corporation. Solaris is a trademark of Sun Microsystems, Inc. SPARC is a registered trademark of SPARC International, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. Linux is a registered trademark of Linus Torvalds. Red Hat is a registered trademark of Red Hat, Inc. All other product or corporate references may be trademarks or registered trademarks of their respective companies.



Table of Contents

Preface	v
1. Getting Started	1
1.1. BlackRay Overview	1
1.2. Installation	1
1.3. Running the Samples	1
1.4. Configuring BlackRay	1
1.5. Starting BlackRay	2
1.6. Loading Data	3
1.7. Taking a Snapshot	5
2. Installation	7
2.1. Installation from a Binary Distribution	7
2.1.1. Installing the Binary Tarball	7
2.1.2. Installing the RPM Distribution	7
2.1.3. Installing the Debian Distribution	7
2.1.4. Installing the Solaris Distribution	8
2.2. Installation from the Source Distribution	8
2.2.1. Requirements	8
2.2.2. Getting the Source	9
2.2.3. Installing the BlackRay Third-Party Libraries	9
2.2.4. Installing BlackRay	10
3. Configuration	13
3.1. Management Server Configuration	13
3.2. Instance Server Configuration	14
3.3. SNMP Module Configuration	16
4. Operation	17
4.1. Querying BlackRay	17
4.1.1. Querying with the Postgresql Protocol	17
4.1.2. Querying with the BlackRay API	19
4.2. Loading Data	21
4.2.1. Introducing the blackray-loader utility	21
4.2.2. Function based indexing	21
4.2.3. Defining Multiple Schemas and Tables	22
A. Server Applications	25
blackray	26
blackray_instd	27
blackray_mgmttd	28
B. Client Applications	29
blackray_cli	30
blackray_loader	31
C. Load Script Reference	33
load-script	34
table-def	35
column-def	36
function	38
index	39
insert-function	41
select-function	42



load-data	43
column	45
D. SQL Key Words	47
keywords	48
E. SQL Grammar Reference	51
drop table	52
alter table	53
drop schema	55
show	56
select	58



Preface

1. About BlackRay

Today's world of connected data continues to pose a great challenge to modern organizations. Ever growing heaps of data need to be sorted, categorized, analyzed and made available to decision makers and other stakeholders, in order to make sound and efficient choices.

BlackRay technology makes just that possible, with greatly reduced complexity and at a fraction of the cost of competing technologies. Powerful features, straightforward architecture and great extensibility make BlackRay technology a premier choice for your high volume data needs.

BlackRay is structured like a traditional database, and supports relational data modeling using tables and joined indices. Support for instances and schemas allows for separation of data into different domains, enabling data protection and versioning.

2. What's new in BlackRay 0.10?

Besides various bug fixes, BlackRay 0.10 introduces several changes to existing features as well as some new features:

- This release introduces support for the postgres protocol. You can now use your postgresSQL JDBC/ODBC drivers to connect to BlackRay. Please refer to Chapter 4, *Operation* for further information.
- The SQL parser has undergone some heavy changes. SQL parsing is now handled on the server side. Appendix E, *SQL Grammar Reference* lists the statements currently supported by BlackRay.
- Initial support for python language bindings has been added.
- The usage of the command-line tools has been simplified. Please see Appendix A, *Server Applications* for usage information.
- The syntax of the configuration file has also changed. The new syntax is documented in Chapter 3, *Configuration*.
- We've added some more examples to demonstrate the usage of the language bindings. The examples are located in `share/samples`.

For a full list of changes please have a look at the file `CHANGES` included in the distribution.

3. Conventions Used in This Document

This document uses the following typographical conventions:

Screen output, file names, file contents, program code and everything else that represents machine input or output uses a `monospaced font`. Placeholders are indicated in *italics*. This means that you have to put in an actual value instead of the placeholder. Required user input within phrases is emphasized using a **bold** font.



4. Support and Further Information

BlackRay is an open-source project and thus depends on your feedback and contributions. If you need support or further information the following resources might be helpful:

Main Web Site

The main BlackRay website, located at <http://www.blackray.org>, is the major resource for news and links about the project.

Project Page

The BlackRay project page is hosted at <https://forge.softmethod.de/projects/show/blackray>. Providing a bug tracker, a wiki as well as the latest source, this is the point of reference for developers and users seeking support.

Mailing Lists

The mailing lists are a good place for asking questions, sharing experience and getting in touch with the developers. Besides the user and developer mailing lists there is also a low-traffic mailing list for official announcements. Visit <https://lists.softmethod.de> for information on how to subscribe.

Additionally, some of the developers might be blogging about BlackRay on their websites. If you're interested have a look at the `AUTHORS` file for some further links.



1. Getting Started

1.1. BlackRay Overview

The following terminology is important for understanding the BlackRay architecture and functionality:

Instance

A single installation of BlackRay consists of at least one instance. An instance is a collection of schemas containing tables and data. One installation can handle several instances. All instances can be started, stopped, loaded and saved independently. An instance can have several schemas.

Schema

A schema is a collection of tables with data stored. A schema is a logical separation of tables. Refer to the SQL standards to learn more about schemas. BlackRay handles schemas according to the SQL standard.

Snapshot

A snapshot can be taken from instances. For each instance there can exist several snapshots. Snapshots actually save the schemas, tables and data stored in one instance to the hard disk. Snapshots are stored in binary format and can be reloaded. Each snapshot in the history can be specifically loaded.

Management Server

The management server of a BlackRay installation is the central unit handling the connections and queries. One management server can serve for several instances.

1.2. Installation

In order to use BlackRay, you need to have a working installation. If BlackRay is not yet installed on your system, please refer to Chapter 2, *Installation* for further instructions.

1.3. Running the Samples

After successfully installing BlackRay, the best way to get started is to run the sample programs. They are located in the directory `share/samples` of your BlackRay distribution.

Each sample is located within a separate sub-directory and should contain a file named `README` with instructions on how to run the sample.

1.4. Configuring BlackRay

In order to run a BlackRay instance, you have to configure your installation first. The default BlackRay configuration is located in a centralized configuration file called `/opt/blackray/etc/blackray.conf`. Let's start by creating a simple configuration file for a single blackray instance:

```
ManagementServer.Endpoints=tcp -h localhost -p 8888 ❶  
ManagementServer.Log=/opt/blackray/log/mgmt.log ❷
```

```

Instance.Samples.Directory=/opt/blackray/data ❸
Instance.Samples.Endpoints.Search=tcp -p 8889 ❹
Instance.Samples.Endpoints.Infra=tcp -h localhost -p 8890 ❺
Instance.Samples.Autostart=true ❻
Instance.Samples.ReadOnly=false ❼
Instance.Samples.Log=/opt/blackray/log/instd.samples.log ❸
Instance.Samples.LazyWrite=true ❾

```

- ❶ This tells the management server to listen on tcp port 8888 on localhost. For further information on the syntax of this parameter please take a look at the `ManagementServer.Endpoints` (string, mandatory) configuration parameter documentation.
- ❷ This configures the location of the management server's log file. Leave this option empty if you do not want the management server to write a log file. In this example we tell the management server to log to the file `/opt/blackray/log`.
- ❸ Sets the directory where the instance `Samples` reads and writes its snapshots. Make sure that the directory exists and that the instance server has read and write permissions in this directory.
- ❹❺ Sets the endpoints for querying (port 8889) and loading (port 8890) the instance. The syntax is the same as for the management server.
- ❻ Tells the management server to automatically start the instance.
- ❼ Allows manipulation of the instance.
- ❸ Sets the instance server output to `/opt/blackray/log/instd.samples.log`. Leave this option empty if you do not want the instance server to write a log file.
- ❾ Configures the instance to *not* write snapshots to disk after modification. Snapshots have to be taken manually.

1.5. Starting BlackRay

After the configuration has been completed, the BlackRay Data Engine can be started. by using the **blackray** command:

```

shell> /opt/blackray/bin/blackray start

  _ _ _ _ _
 / / _ / / _ _ _ _ _ / / _ _ _ _ _ _ _ _
 / _ _ \ / _ _ \ / _ _ / / / _ _ / _ _ \ / / /
 / / / / / / / / / _ / , > / / / / / / / / /
 / _ _ _ / _ \ _ _ , _ \ _ _ / _ / | _ / _ \ _ _ , /
                                     / _ _ /

Blackray Management Server Administrator V0.10.0

[Starting] Management Server
[Started] Management Server
shell>

```

This starts the management server as well as all instances configured to start automatically. The same command can be used for stopping the BlackRay Data Engine.

To check if everything is working correctly connect to the instance using command line interface (**blackray_cli**):



```

</column>
</load-data>

</load-script>

```

- ❶ Defines a table named `my_table` in the schema named `my_schema`.
- ❷ The schema holds a single column named `my_column`. The column is of type `string` and a search index is created for this column, as the attribute `searchable` is set to `yes`. The content of this column is tokenized with an empty space as delimiter (parameter `tokenized`), so you can do a full text search in this column. Also, searching with *leading* wildcards is enabled, as the attribute `wildcards` is set to `yes`. Note that trailing wildcards are always enabled by default if the column is searchable.
- ❸ Instructs the loader to load data into our table `my_table` from the file `data.csv`.
- ❹ The data for the column `my_column` is loaded from the column at position 0 in the csv file.
- ❺ Additionally, the column is *trimmed* before it is loaded into the database. If the sequence of columns in `my_table` is the same as in the csv file, and if you do not want to specify some additional load functions, you can omit the definition of the columns for the loader.

Now you need to fill your csv file `data.csv` with some content:

```

"Lorem ipsum dolor sit amet",0
"consectetuer sadipscing elitr ",1
"sed diam nonumy eirmod tempor",2

```

To load the data into your running instance, use the command **blackray_loader** (let's assume your csv data file is contained in the directory `/tmp`):

```

shell> /opt/blackray/bin/blackray_loader -c load.xml -d /tmp \
      -e "tcp -p 8890"

  _ _ _ _ _
 / / _ / / _ _ _ _ / / _ _ _ _ _ _ _ _ _
 / _ \ / / _ \ / _ / // / _ / _ \ / / /
 / / / / / / / / / / , < / / / / / / / /
 / _ _ / / \ _ , _ \ _ / / | _ / / \ _ , _ \ _ /
                                     / _ _ /

Blackray Loader V0.10.0

Connecting to endpoint: tcp -p 8890
Creating schema my_schema
Creating table my_schema.my_table
Loading data into table my_schema.my_table from file data.csv
shell>

```

Now that you have created your first BlackRay table and loaded some data into it, you can start playing around on the command line interface:

```

shell> /opt/blackray/bin/blackray_cli -e "tcp -p 8889" -q
blackray_cli> select * from my_schema.my_table;

|-----|

```



```
| my_column |
|-----|
| Lorem ipsum dolor sit amet |
| consectetur adipiscing elit |
| sed diam nonummy eirmod tempor |
|-----|

3 rows selected.

blackray_cli> select * from my_schema.my_table where my_column = 'dolor';

|-----|
| my_column |
|-----|
| Lorem ipsum dolor sit amet |
|-----|

1 row selected.

blackray_cli> select * from my_schema.my_table where my_column like '%m';

|-----|
| my_column |
|-----|
| Lorem ipsum dolor sit amet |
| sed diam nonummy eirmod tempor |
|-----|

2 rows selected.

blackray_cli>
```

1.7. Taking a Snapshot

Once your data has been loaded into blackray, you can make the data permanent by creating a snapshot. This way, your data will automatically be reloaded when your instance is restarted. To do so, use the **blackray** command to take a snapshot of your `Samples` instance:

```
shell> /opt/blackray/bin/blackray take_snapshot Samples

  _ _ _
 / /_ / /__ ___/ / _____ _ _
 / _ \ / / _ ` / __/ // / __/ _ ` / / /
 / / / / /_ / /_ / ,< / / / /_ / /_ /
 /_ .__ /_ \_ ,_ \_ /_ / |_ /_ \_ ,_ \_ /
                               /___/

Blackray Management Server Administrator V0.10.0

Taking snapshot on Instance Server: Samples.
Taking snapshot on Instance Server: Samples finished after 0s.
```



```
shell>
```

2. Installation

This chapter describes the installation of the BlackRay Data Engine. The easiest way to install BlackRay is from a binary distribution. If you like to, or if there are no binaries available for your platform, you can also install BlackRay from the source distribution.

2.1. Installation from a Binary Distribution

Binary distributions are the easiest way to install BlackRay on your system. Visit the BlackRay website at <http://www.blackray.org> and download the distribution for your operating system. If there is no binary distribution available for your environment, you can try to install BlackRay from the source distribution.



Note

Installation of software packages usually requires administrator privileges. The following instructions assume that you are performing the required steps as `root`. When unsure, please contact your system administrator.

2.1.1. Installing the Binary Tarball

To install BlackRay from a binary tarball, download the archive matching your system environment from the BlackRay website. The naming scheme for the binary tarball is `blackray_VERSION_SYSTEM.tar.gz`, where `VERSION` is the BlackRay version number and `SYSTEM` describes the operating system and architecture the package was built for. The binary archive contains the third-party libraries as well as the BlackRay binaries, so there is no need to download any additional packages. When the download is finished, install the packages by entering the following commands (Make sure you have administrator privileges):

```
shell> cd /opt
shell> tar xzf blackray_VERSION_SYSTEM.tar.gz
```


2.1.2. Installing the RPM Distribution

To install BlackRay from rpm packages, download the packages matching your environment from the BlackRay website. The naming scheme for the packages is `blackray_VERSION_SYSTEM.rpm` and `blackray-thirdparty_VERSION_SYSTEM.rpm`, where `VERSION` is the BlackRay version number and `SYSTEM` describes the operating system and architecture the package was built for. When the download is finished, install the packages by entering the following commands:

```
shell> rpm -i blackray-thirdparty_VERSION_SYSTEM.rpm
shell> rpm -i blackray_VERSION_SYSTEM.rpm
```

2.1.3. Installing the Debian Distribution

To install the BlackRay Debian packages, download the packages matching your environment from the BlackRay website. The naming scheme for the packages is `blackray_VERSION_SYSTEM.deb` and `blackray-thirdparty_VERSION_SYSTEM.deb`, where `VERSION` represents the BlackRay version



number and *SYSTEM* describes the operating system and architecture the package was built for. When the download is finished, you can install the packages by entering the following commands:

```
shell> dpkg -i blackray-thirdparty_VERSION_SYSTEM.deb
shell> dpkg -i blackray_VERSION_SYSTEM.deb
```

2.1.4. Installing the Solaris Distribution

To install the BlackRay Solaris packages, download the packages matching your environment from the BlackRay website. The naming scheme for the packages is `blackray_VERSION_SYSTEM.pkg` and `blackray-thirdparty_VERSION_SYSTEM.pkg`, where *VERSION* represents the BlackRay version number and *SYSTEM* describes the operating system and architecture the package was built for. When the download is finished, you can install the packages by entering the following commands:

```
shell> pkgadd -d blackray-thirdparty_VERSION_SYSTEM.pkg
shell> pkgadd -d blackray_VERSION_SYSTEM.pkg
```

2.2. Installation from the Source Distribution

This chapter describes how to build and install the BlackRay Data Engine from the source code distribution.

2.2.1. Requirements

The following prerequisites are needed for building BlackRay from source:

- cmake 2.6.3 or higher. Earlier versions will have problems finding the boost libraries with some compiler combinations.
- GNU make. The recommended GNU make version is 3.81 or later. Other make programs will not work. Note that on some systems GNU make is installed as **gmake** or has to be installed separately. This document will refer to GNU make with the command name **make**.
- A C++ compiler. BlackRay has been tested and is known to compile with the following compilers:
 - gcc 4.1.3, 4.3.2 and 4.4.2 on Linux
 - gcc 4.0.1 as supplied by XCode on Mac OS X
 - Sun Studio 12 on Solaris
- Also, make sure that the development header files of your operating system are installed. For example on Solaris you have to install the package `SUNWhea` manually, because it is not installed by default.
- Sun JDK 1.6 or later
- In addition, BlackRay requires several third-party libraries, which are bundled into a separate package. See Section 2.2.2, “Getting the Source” for more details.
- To build some of the third-party libraries, Apache Ant 1.5 or higher is required.



- Optionally, if you want to build the BlackRay SNMP library, you need to have the Net-SNMP library and headers installed.

2.2.2. Getting the Source

BlackRay source distributions are provided as gzip compressed tar archives. The naming scheme for the distribution is `blackray-VERSION.tar.gz`, where `VERSION` is the BlackRay version number (e.g. 0.10.0).

BlackRay requires several third-party libraries. The third-party sources are bundled into a separate package. The naming scheme for the third-party distribution is `blackray-thirdparty-VERSION.tar.gz`, where `VERSION` is the BlackRay version number.

For obtaining the latest source distributions, visit the BlackRay website at <http://www.blackray.org>. After you have downloaded the sources, unpack them by executing the following commands:

```
shell> gunzip blackray-thirdparty-VERSION.tar.gz
shell> tar xf blackray-thirdparty-VERSION.tar
shell> gunzip blackray-VERSION.tar.gz
shell> tar xf blackray-VERSION.tar
```

By executing these steps, two directories `blackray-thirdparty-VERSION` and `blackray-VERSION` will be extracted, containing the sources of the third-party libraries and the BlackRay sources.

2.2.3. Installing the BlackRay Third-Party Libraries

The basic installation procedure for the BlackRay third-party libraries can be summarized by the following steps, which we will describe in more detail afterwards. Please change to the directory containing the third-party sources before executing those steps.

```
shell> cmake -DCMAKE_INSTALL_PREFIX=/opt/blackray/thirdparty .
shell> make
shell> make install
```



1. Configure

Before you build the third-party source, you have to configure the source tree for your build environment. This is achieved by invoking `cmake` in the root directory of the third-party sources. To configure with the default settings for your build environment simply run `cmake -DCMAKE_INSTALL_PREFIX=/opt/blackray/thirdparty .` on the command line. This will generate the necessary Makefiles needed for building the third-party libraries on your system.

Please note that you have to specify the install destination on the command line using the `-DCMAKE_INSTALL_PREFIX` parameter. Not doing so will cause the third-party libraries to be installed in `/usr/local`.

The compilers used for building the libraries can be controlled by setting the environment variables `CC` and `CXX`. `cmake` then uses the specified compilers for the configuration of your build environment.



Note

To compile BlackRay on Solaris, you should use the Sun Studio compilers. If it is found on your system, `cmake` will choose `gcc` by default. In order to use the Sun compilers on Solaris, you have to invoke `cmake` by running `CC=cc CXX=CC cmake -DCMAKE_INSTALL_PREFIX=/opt/blackray/thirdparty .` on the command line.

2. Build

After you have configured the source tree, you can start the build process by running GNU `make` from the root directory of the third-party sources. To start the build, type `make` on the command line.

Building the third-party sources can take up to several hours depending on your hardware.

3. Install

After the third-party libraries have been successfully compiled, install them by entering `make install`. The compiled libraries will be installed in the directory `/opt/blackray/thirdparty` if you have set your `CMAKE_INSTALL_PREFIX` accordingly. This normally needs to be done as `root`. If you are installing the files as a normal user, make sure you have the appropriate permissions to create the install directories, or install into a different directory.

2.2.4. Installing BlackRay

Before you start building and installing BlackRay from Source, make sure that the third-party libraries are installed on your system. If you want to install them from source, please refer to Section 2.2.3, "Installing the BlackRay Third-Party Libraries".

The installation procedure for BlackRay can be summarized in the following steps, which will be described in more detail later. Please change to the directory containing the blackray sources before executing those steps.

```
shell> cmake -DCMAKE_INSTALL_PREFIX=/opt/blackray .
shell> make
shell> make test ⓘ
```

```
shell> make install
```

- ❶ This is an optional step. You can skip it if you do not want to run the regression tests.

1. Configure

Before you can build the BlackRay sources, you have to configure the source tree for your build environment. This is achieved by executing `cmake` in the root directory of the source tree. Execute `cmake -DCMAKE_INSTALL_PREFIX=/opt/blackray .` to configure the source tree with the default settings for your build environment. This will generate the Makefiles needed for building BlackRay on your system.

Please note that you have to specify the install destination on the command line using the `-DCMAKE_INSTALL_PREFIX` parameter. Not doing so will cause blackray to be installed in `/usr/local`. Also please make sure that you have installed the third-party libraries in the subdirectory `thirdparty` beneath the directory specified by `CMAKE_INSTALL_PREFIX`.

The compiler used for building the source can be controlled by setting the environment variables `CC` and `CXX`. `cmake` will use the specified compilers to configure your build environment.



Note

To compile BlackRay on Solaris, you should use the Sun Studio compilers. If it is found on your system, `cmake` will choose `gcc` by default. In order to use the Sun compilers on Solaris, you have to invoke `cmake` by running `CC=cc CXX=CC cmake -DCMAKE_INSTALL_PREFIX=/opt/blackray .` on the command line.

2. Build

After you have configured the source tree, you can start building the sources by running GNU `make` from the toplevel directory of the source tree. To start the build, type `make` on the command line.

The build will take several minutes depending on your hardware. So it's time to grab yourself a cup of coffee, or just sit back and relax.

3. Test

After the binaries have been compiled, you can run the regression tests if you like to. To do so, type `make test` on the command line.

4. Install

When BlackRay has been successfully built, you can install it by entering `make install`. BlackRay will be installed in the directory `/opt/blackray` if you have set your `CMAKE_INSTALL_PREFIX` accordingly. This normally needs to be done with superuser permissions. If you are installing the files as a normal user, make sure you have the appropriate permissions to create the installation directories, or install into a different directory.





3. Configuration

The BlackRay Data Engine by default reads its configuration data from the file `/opt/blackray/etc/blackray.conf`. To avoid overwriting an existing configuration, this file will not be created during installation. If you're installing BlackRay for the first time, you can either adapt the included example configuration `blackray.conf.example` located in the same directory, or create a new configuration from scratch.

Each BlackRay environment consists of two types of server processes:

- The management server process is responsible for management of the data instances. It automatically starts and stops the configured instances and controls the update of replicated instance nodes.
- The instance server process is basically a single database instance. It holds the instance data in memory and is responsible for processing queries to the instance.

Note that while there can be several instance servers running on the same machine, there can only be one management server running.

3.1. Management Server Configuration

The following parameters control the operation of the management server.

`ManagementServer.Endpoints` (string, mandatory)

Configures the protocols, interfaces and ports on which the management server is listening. The endpoint is configured by a string of the format `protocol -h host -p port`, where

- `protocol` sets the protocol to be used for connections. While ICE supports `tcp`, `udp` and `ssl` here, BlackRay has only been tested with `tcp`. Setting the value to `default` will use the default protocol (`tcp`).
- `-h host` specifies the host name or IP address on which the management server is listening for connections. If not specified (or specified as `-h *` or `-h 0.0.0.0`), the server will be listening on all network interfaces.
- `-p port` specifies the port number on which the management server will be listening.

Example: The value `default -h localhost -p 8888` configures the management server to be listening for `tcp` connections on port 8888 of the loopback interface.

`ManagementServer.Log` (string, optional)

Configures the management server's log file. Although relative paths are accepted here, it is recommended that you use an absolute path. If you use a relative path here, it will be relative to the directory where the management server has been started. Leave this parameter empty to disable logging.

`ManagementServer.ThreadPool.Server.Size` (integer, optional)

Sets the number of threads that will be started initially for processing of queries.

`ManagementServer.ThreadPool.Server.SizeMax` (integer, optional)

Sets the maximum number of threads allowed for processing of queries.



`ManagementServer.MessageSizeMax` (integer, optional)

Sets the maximum size (in kB) allowed for messages between the management server and connected clients.

3.2. Instance Server Configuration

The following parameters control the operation of an instance server. They have to be configured for each separate instance. Replace the variable `NAME` with the name of the instance.

`Instance.NAME.Directory` (string, mandatory)

Configures the data directory of the instance. The data directory contains the snapshots and redologs of the BlackRay instance. Although relative paths are accepted here, it is recommended that you use an absolute path. If you use a relative path here, it will be relative to the directory where the instance server has been started.

`Instance.NAME.Endpoints.Search` (string, mandatory)

Configures the protocols, interfaces and ports on which the instance server is listening. These settings will be used by client connections for querying the instance. For information on the syntax of this parameter please refer to the `ManagementServer.Endpoints` configuration parameter.

`Instance.NAME.Endpoints.Infra` (string, mandatory)

Configures the protocols, interfaces and ports on which the instance server is listening. The settings will be used by management connections. For information on the syntax of this parameter please refer to the `ManagementServer.Endpoints` configuration parameter.

`Instance.NAME.Autostart` (boolean, mandatory)

Controls whether the instance will be automatically started by the management server:

- If set to `true`, the management server will automatically start the instance.
- If set to `false`, the instance has to be started manually.

`Instance.NAME.LazyWrite` (boolean, mandatory)

Configures the snapshot behavior of the instance:

- If set to `true`, snapshots will automatically be taken after data has been loaded into the instance.
- If set to `false`, snapshots have to be taken manually.

`Instance.NAME.ReadOnly` (boolean, mandatory)

Configures whether the loaded data can be altered.

- If set to `true`, the instance cannot be modified after the initial snapshot has been loaded during startup.
- If set to `false`, the instance can be altered by data operations.

`Instance.NAME.Log` (string, optional)

Configures the instance's log file. Although relative paths are accepted here, it is recommended that you use an absolute path. If you use a relative path here, it will be relative to the directory where the instance server has been started. Leave this parameter empty to disable logging.



`Instance.NAME.LoadSnapshotVersion` (integer, optional)

If this parameter is set, the instance server will load the specified snapshot version in read-only mode during startup. If this parameter is not set, the instance server will load the newest snapshot available.

`Instance.NAME.MaxSnapshotBackupCount` (integer, optional)

Sets the maximum number of snapshots that will be saved in the data directory of the instance. If the number of snapshots exceeds the allowed maximum, the oldest snapshots will be deleted.

`Instance.NAME.pgsql.Port` (integer, optional)

Sets the port where the postgresql listener should be started. By default the listener is started on port 5432. Set to -1 to disable the postgres protocol.

`Instance.NAME.username` (string, optional)

Sets the username of the user which is allowed to connect to the instance. Leave empty to disable authentication. Note that authentication is currently only supported in the postgres protocol.

`Instance.NAME.password` (string, optional)

Sets the password of the user which is allowed to connect to the instance. Leave empty to disable passwords. If you specify a password, you also need to specify the username. Note that authentication is currently only supported in the postgres protocol.

`Instance.NAME.Replicates.Endpoints` (string, optional)

Needed for multi-node configurations where one instance is replicated to several copies. This parameter needs to be configured on the master node and should contain a comma-separated list of the replicated nodes' management endpoints.

`Instance.NAME.Replicates.MaxRetry` (integer, optional)

Needed for multi-node configurations where one instance is replicated to several copies. This parameter sets the maximum number of times the master node should retry to update a replicated node in case of an error. If the number of retries exceeds the defined maximum, the previous snapshot will be loaded as a fallback solution.

`Instance.NAME.Replicates.MaxError` (integer, optional)

Needed for multi-node configurations where one instance is replicated to several copies. This parameter sets the maximum number of failed instances after which an update will be aborted.

`Instance.NAME.Replicates.MinActive` (integer, optional)

Needed for multi-node configurations where one instance is replicated to several copies. Sets the minimum number of nodes that have to remain in service during updates. If there are less nodes available for serving requests during the update, the currently running update will be aborted.

`Instance.NAME.ThreadPool.Server.Size` (integer, optional)

Sets the number of threads that will be started initially for processing of queries.

`Instance.NAME.ThreadPool.Server.SizeMax` (integer, optional)

Sets the maximum number of threads allowed for processing of queries.

`Instance.NAME.MessageSizeMax` (integer, optional)

Sets the maximum size (in kB) allowed for messages between the instance server and connected clients.

3.3. SNMP Module Configuration

If BlackRay was previously built with SNMP support enabled, you can now configure the support module for the SNMP server. For this, you need to have the net-snmp server installed on the machine running BlackRay.

To configure the SNMP module, please follow these steps:

1. Add the SNMP module

To add the SNMP module to your net-snmp server configuration, attach the contents of the file `/opt/blackray/etc/snmpd.conf.attachment` to the its config file. The config file is usually located in `/etc/snmp/snmpd.conf`:

```
shell> cd /opt/blackray
shell> cat etc/snmpd.conf.attachment >> /etc/snmp/snmpd.conf
```

2. Configure the SNMP module

To configure the BlackRay SNMP module, copy the module's example configuration to the net-snmp config directory:

```
shell> cp etc/smthBlackray.conf.example /etc/snmp/smthBlackray.conf
```

After the module configuration has been copied, adapt the file to match your environment. Make sure that `endpoint` matches the endpoint configured in your management server.

3. Restart the net-snmp server

```
shell> /etc/init.d/snmpd restart
* Restarting network management services:
...done.
```

4. Test the SNMP module

After restarting the net-snmp server, check the configuration by executing e.g. `snmpwalk`. The MIB files for BlackRay are located in `share/mibs`:

```
shell> snmpwalk -v 2c -c smth \  
> -m share/mibs/SMTH-SMI.txt:share/mibs/SMTH-BLACKRAY-MIB.txt \  
> localhost smthBlackRay
SMTH-BLACKRAY-MIB::smthBlackRaySystemName.0 = STRING: "BlackRay"
SMTH-BLACKRAY-MIB::smthBlackRaySystemLocation.0 = STRING: "Unknown"
SMTH-BLACKRAY-MIB::smthBlackRaySystemHostname.0 = STRING: "x4600-r0"
SMTH-BLACKRAY-MIB::smthBlackRaySystemSoftwareVersion.0 \  
= STRING: "0.10.0"
```



4. Operation

This chapter describes the different aspects of operating a BlackRay installation.

4.1. Querying BlackRay

There are currently two alternatives for connecting to and querying a running BlackRay instance:

- Using the postgres protocol, as described in Section 4.1.1, “Querying with the Postgresql Protocol”
- Using the blackray API, as described in Section 4.1.2, “Querying with the BlackRay API”

4.1.1. Querying with the Postgresql Protocol

BlackRay 0.10.0 introduces a postgres listener for connecting to and querying a running instance.

The BlackRay distribution does not provide any custom drivers. Instead you can use the standard postgres JDBC or ODBC drivers for connecting to blackray. We believe this approach reduces development overhead and enables us to focus on development of the engine itself. The postgresql drivers can be obtained from <http://jdbc.postgresql.org/> and <http://pgfoundry.org/projects/psqlodbc/>.

Currently only a subset of SQL as well as a limited subset of the BlackRay API is supported. For a list of supported commands please refer to Appendix E, *SQL Grammar Reference*.

4.1.1.1. Interactive querying from the postgresSQL terminal

You can use the postgresSQL interactive terminal (**psql**) for querying a running blackray instance. If you have configured authentication, make sure to specify the correct username on the command line:

```
shell> psql -h localhost -p 5432 -U blackray
Password for user blackray:
Welcome to psql 8.3.7 (server 0.10.0), the PostgreSQL interactive \
terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

WARNING:  You are connected to a server with major version 0.10,
but your psql client is major version 8.3.  Some backslash commands,
such as \d, might not work properly.

blackray=> show schemas;
  name  | size
-----+-----
 <Default> | 0
(1 row)
```

```
blackray=>
```

4.1.1.2. Querying using the JDBC and ODBC drivers

For programmatic querying, the PostgreSQL JDBC and ODBC drivers can be used. The following example illustrates the usage of the JDBC driver for connecting to BlackRay from a Java application.

```
import java.sql.*;

public class JDBCTest {
    public static void main(String[] argv) {

        try {
            Class.forName("org.postgresql.Driver"); ❶
        } catch (ClassNotFoundException cnfe) {
            System.out.println("Could not register driver");
            cnfe.printStackTrace();
            System.exit(1);
        }

        System.out.println("Driver registered, trying to connect");

        String user = "blackray"; ❷
        String pass = "blackray";
        String host = "localhost";
        String db = "sample";

        try {
            Connection con = DriverManager.getConnection(
                "jdbc:postgresql://" + host + "/" + db, user, pass); ❸

            System.out.println("Connected, executing query");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SHOW SCHEMAS"); ❹
            while( rs.next() ) {
                System.out.println(rs.getString(1));
            }

            rs.close();
            stmt.close();
            con.close();
        } catch (SQLException se) {
            System.out.println("Could not connect:");
            se.printStackTrace();
        }

    }
}
```



- ❶ Registers the postgres JDBC driver.
- ❷ Defines username, password, hostname, and database name for the connection. The database name is currently ignored by BlackRay, but should be set to the name of the instance.
- ❸ Connects to the BlackRay instance.
- ❹ Executes a query.

Please make sure that the postgresSQL JDBC driver is in your classpath when trying to run the example:

```
shell> javac -cp postgresql-8.4-701.jdbc4.jar JDBCTest.java
shell> java -cp postgresql-8.4-701.jdbc4.jar:. JDBCTest
Driver registered, trying to connect
Connected, executing query
<Default>
shell>
```

4.1.2. Querying with the BlackRay API

BlackRay also provides a custom API for connecting to a running instance. The API is based on zeroC ICE and can be used for querying data from an instance as well as loading data into a running instance.

While the postgresSQL listener provides a more generic approach, only the custom API currently provides the full featureset. However, future versions of BlackRay will provide all of BlackRay's features for the postgres protocol as well.

The full API is currently available for Java. Rudimentary support for the python language has been added in BlackRay 0.10.0.

The following example demonstrates usage of the API for running simple queries.

```
import org.blackray.connection.*;
import org.blackray.select.*;
import Ice.InitializationData;

public class SelectWhereSample {

    public static void main(String[] args) {
        InitializationData init = new InitializationData();
        init.properties = Ice.Util.createProperties();

        ConnectionChannel chan = new ConnectionChannel(init,
            "ice://localhost:8889", "Default"); ❶
        Connection con = chan.getConnection();

        System.out.println("SELECT member_name FROM samples.band_members");
        System.out.println("  WHERE member_name='John'");

        Select select = new Select("samples.band_members"); ❷

        select.addSelectFieldValue("member_name"); ❸
    }
}
```

```

SelectWhere selectWhere = Select.createSelectWhereEqual("member_name",
    "John"); ❹
select.setSelectWhere(selectWhere);

try {
    CommandResult cr = con.execute(select); ❺
    ResultSet rs = (ResultSet) cr.results.get("resultset"); ❻

    while (rs.next()) {
        System.out.println(rs.getString(0));
    }

} catch (Throwable t) {
    t.printStackTrace();
    System.exit(1);
} finally {
    try {
        con.close();
    } catch (Throwable t1) {
        t1.printStackTrace();
        System.exit(1);
    }
    chan.close();
}
}
}

```

- ❶ Creates the connection to the instance Default running on localhost port 8889.
- ❷ Creates a new select object for selecting from the table `samples.band_members`.
- ❸ Adds the column `member_name` to the result.
- ❹ Adds a where condition.
- ❺ Executes the select.
- ❻ Retrieves the resultset.

Please make sure to add BlackRay API driver as well as the ICE API to the classpath when trying to run the example:

```

shell> javac -cp blackray.jar:Ice.jar SelectWhereSample.java
shell> java -cp blackray.jar:Ice.jar:. SelectWhereSample
Connection to default://localhost:8889 established
Working connection found
SELECT member_name FROM samples.band_members
WHERE member_name='John' ;
John Lennon
shell>

```

The BlackRay distribution contains several additional examples for demonstrating the usage of the API. They are located in `/opt/blackray/share/samples/`. For information on how to use them please have a look at the `README` file in each sample directory.



4.2. Loading Data

This Section will describe how to load data on to a running Instance of Blackray.

4.2.1. Introducing the blackray-loader utility

For a running instance, data can be loaded by the use of the **blackray_loader** utility. The loader essentially parses the xml load-script, creates the schemas and tables and finally loads the csv data onto the running instance. Optionally, functions can also be added to the load-script by using the `function` tag in the loader-script.

To load data successfully, `blackray_loader` accepts 3 mandatory arguments:

- The loader-script file (switched by `-c`)
- The location of the csv data file (switched by `-d`)
- The endpoints of the running instance (switched by `-e`)

For Example:

```
shell> blackray_loader -c load.xml -d /tmp -e "tcp -p 8890"
```

4.2.2. Function based indexing

One of the most interesting features of blackray is its support of Function Based indices.

Currently 5 types of functions are supported by the loader utility:

- `trim` : Trims the leading and trailing whitespaces.
- `lower` : Converts every string to its respective lowercase letters.
- `upper` : Converts every string to its respective uppercase letters.
- `phonetic` : Converts every string to their respective phonetic (currently only German Phonetic is supported). The syllable is also displayed along with the phonetic, after a '#' symbol.
- `normalize` : Normalizes every string (Not to be confused with Database Normal Forms). It basically uses a set of rules like : Converting to lower cases, NFD (Normalization form D) and NFC (Normalization form C) etc to normalize the strings. This function is useful especially when using non English Texts or special symbols.

Functions can either be applied:

- Before loading the data onto the running instance or
- During events like "select" or "insert" from `blackray_cli`.

Implementing the both of the above are easy and the former is done by using the `function` tag of a `column` in the loader-script. For Example:

```
<load-script>
```

```

...
<load-data table="foo" schema="bar" file="test_data.csv">
<column name="col1" csv-column="0">
<function type="trim"/>
</column>
</load-data>
...
</load-script>

```

To implement function-based indices, the tags "insert-function" and "select-function" are used under the root node "index". Refer to Appendix C, *Load Script Reference* for more details and examples.

Interestingly, Functions can also be nested or stacked, providing a rich featureset all together. For Example (Stacked functions):

```

<load-script>
...
<function type="lower"/>
<function type="trim"/>
...
</load-script>

```

The code above, converts the whole data to lowercase and also trims the leading and trailing whitespaces, before loading it to the database.

Caution: When the functions are stacked, the implementation is always "in order". Sometimes, this can totally shadow out other functions. For Example:

```

<function type="lower" />
<function type="upper" />

```

In the above case, the resulting data will have only uppercases (as it overwrites the previous function). Additionally, more functions can be accessed by using Blackray's native Java API, using the method `setFunction()`.

4.2.3. Defining Multiple Schemas and Tables

As the loader-script is essentially a xml file, multiple Schemas and Table can be defined in the same loader-script file too. The following sample shows how to define multiple tables:

```

<load-script>

<table-def name="bar1" schema="foo">
<column-def name="col1" type="string" searchable="yes" compress="yes"
wildcards="yes" />
</table-def>

<table-def name="bar2" schema="foo">
<column-def name="col1" type="string" searchable="yes" compress="yes"
wildcards="yes" />

```



```
</table-def>

<load-data table="bar1" schema="foo" file="test_data.csv">
<column name="col1" csv-column="0">
<function type="trim"/>
<function type="upper" />
</column>
</load-data>

<load-data table="bar2" schema="foo" file="test_data.csv">
<column name="col1" csv-column="0">
<function type="lower"/>
</column>
</load-data>

</load-script>
```





Appendix A. Server Applications



Name

blackray — The BlackRay Server Administrator

Synopsis

blackray -h

blackray -v

blackray [*OPTIONS*] {*COMMAND* [*ARGS*]}

Description

The **blackray** command can be used to perform administrative tasks for the blackray servers.

Options

-c *FILE*, --config=*FILE*
Use *FILE* as configuration file instead of default.

-h, --help
Print a help message and exit.

-v, --version
Print the version number and exit.

Commands

start [*INSTANCE*]
Starts the BlackRay management server and all instances with `Instance.NAME.Autostart` enabled. If *INSTANCE* is specified, only the instance with that name will be started.

stop [*INSTANCE*]
Stops the BlackRay management server and all instances. If *INSTANCE* is specified, only the instance with that name will be stopped.

restart [*INSTANCE*]
Restarts the BlackRay management server and all instances with `Instance.NAME.Autostart` enabled. If *INSTANCE* is specified, only the instance with that name will be restarted.

status
Print the status of the management server and configured instances.

take_snapshot {*INSTANCE*}
Takes a snapshot of the running instance specified by *INSTANCE*. The snapshot will be placed in the directory specified by the `Instance.INSTANCE.Directory` configuration parameter.

update_replicas {*INSTANCE*}
Updates the replicas configured for the instance specified by *INSTANCE*.

See Also

blackray_instd(1) blackray_mgmtd(1)



Name

blackray_instd — The BlackRay Instance Server

Synopsis

```
blackray_instd -h
```

```
blackray_instd -v
```

```
blackray_instd [OPTIONS] {INSTANCE}
```

Description

Starts the BlackRay instance server specified by *INSTANCE*.

Options

`-c FILE, --config=FILE`

Use *FILE* as configuration file instead of default.

`--daemon`

Run as a daemon.

`-h, --help`

Print a help message and exit.

`-q, --quiet`

Do not write to stdout.

`-v, --version`

Print the version number and exit.

See Also

blackray(1)



Name

blackray_mgmtmd — The BlackRay Management Server

Synopsis

```
blackray_mgmtmd -h
```

```
blackray_mgmtmd -v
```

```
blackray_mgmtmd [OPTIONS]
```

Description

Starts the BlackRay management server.

Options

```
-c FILE, --config=FILE  
    Use FILE as configuration file instead of default.
```

```
--daemon  
    Run as a daemon.
```

```
-h, --help  
    Print a help message and exit.
```

```
--noauto  
    Do not automatically start any instances.
```

```
-q, --quiet  
    Do not write to stdout.
```

```
-v, --version  
    Print the version number and exit.
```

See Also

blackray(1)



Appendix B. Client Applications



Name

`blackray_cli` — The blackray command line interface.

Synopsis

```
blackray_cli -h
```

```
blackray_cli -v
```

```
blackray_cli [-q] [-e endpoint] [-c sql]
```

Description

`blackray_cli` is the BlackRay command line interface. You can use it execute SQL commands interactively on a running BlackRay instance.

Options

`-c sql`

Run the SQL command specified by *sql* and exit.

`-e endpoint`

Connect to the specified instance specified by *endpoint*.

`-h`

Print a help message and exit.

`-q`

Quiet mode.

`-v`

Print the version number and exit.

Examples

The following command connects to the instance running at the host `foo` on port `8889`:

```
blackray_cli -e "default -h foo -p 8889"
```

Use the following command to connect to the instance running at the host `foo` on port `8889` and run the sql query `select * from schema.table`:

```
blackray_cli -e "default -h foo -p 8889" -c "select * from schema.table"
```



Name

`blackray_loader` — The blackray data loader.

Synopsis

```
blackray_loader -h
```

```
blackray_loader -v
```

```
blackray_loader {-c control-file} [-d directory] {-e endpoint} [-q]
```

Description

`blackray_loader` is used to load data into the BlackRay data engine.

Options

`-c control-file`

Read the table and loader definitions from the specified *control-file*.

`-d directory`

Use the specified directory as the data directory. All data files specified in the control file are relative to this data directory.

`-e endpoint`

Connect to the instance specified by *endpoint*. Be sure to use the infrastructure endpoint of the instance.

`-h`

Print a help message and exit.

`-q`

Quiet mode.

`-v`

Print the version number and exit.

Examples

The following command executes the load statements in the control-file `./foo.xml` and assumes the required data files are located in the directory `/tmp`:

```
blackray_cli -c ./foo.xml -d /tmp
```





Appendix C. Load Script Reference



Name

load-script — The root element of a loader script.

Synopsis

load-script ::=

- Sequence of:
 - table-def
- Sequence of:
 - load-data

Attributes

None.

Description

The `load-script` element is the root element for all loader scripts. It contains one or more `table-def` elements to define the structure of the tables to be loaded, followed by one or more `load-data` elements to actually load the data into those tables.

Parents

None.

Children

The following elements occur in `load-script`: `table-def` , `load-data`

Examples

```
<load-script>
  <table-def name="mytable" schema="myschema">
    <!-- ... -->
  </table-def>

  <load-data table="mytable" schema="myschema" file="data.csv">
    <!-- ... -->
  </load-data>
</load-script>
```



Name

table-def — Table definition.

Synopsis

table-def ::=

- Sequence of:
 - column-def

Attributes

- **name**
- **schema**

Required attributes are shown in **bold**.

Description

Defines the structure of a table which will be created in the BlackRay instance.

Attributes

name

The name of the table that is to be defined.

schema

The schema where the table should be created in.

Parents

These elements contain table-def: load-script

Children

The following elements occur in table-def: column-def

Examples

```
<table-def name="mytable" schema="myschema">
  <column-def name="column_1" type="string" tokenizing=" "
              searchable="yes" wildcards="no" />

  <!-- ... -->
</table-def>
```



Name

column-def — Column definition.

Synopsis

column-def ::=

- `function?`
- Zero or More of :
 - `index`

Attributes

- **name**
- **type**
- searchable
- tokenizing
- compress
- wildcards

Required attributes are shown in **bold**.

Description

Defines a column for a table. If a `function` is defined for this column, the specified function will be used as a base function. This means the specified function will be applied to data before inserting it into the column as well as to search terms before searching in this column.

Additionally, multiple `indexes` can be defined for this column, which do not modify the original data stored in the column but instead create a separate function-based index for searching the column.

Attributes

name

The name of the column defined by this element.

type

The type of the column defined by this element. One of: `integer`, `long`, `string`

searchable

If set to `yes`, a search index for the column is created. If set to `no`, the column cannot be searched.

tokenizing

If set to `yes` the content of this column will be tokenized in order to enable a full-text search on this column.

compress

If set to `yes` the search index for this column will be compressed.

wildcards

If set to `yes`, searches with leading wildcards will be enabled for this column. Searches with trailing wildcards are supported by default.



Parents

These elements contain `column-def`: `table-def`

Children

The following elements occur in `column-def`: `index`, `function`

Examples

```
<column-def name="column_1" type="string" tokenizing=" "
            searchable="yes" wildcards="no">
  <function type="normalize" />
  <!-- ... -->

  <index name="myindex" type="string" tokenizing=" " wildcards="true">
    <!-- ... -->
  </index>
</column-def>
```



Name

function — Function definition.

Synopsis

function ::=

- function?

Attributes

- **type**

Required attributes are shown in **bold**.

Description

Defines a function. Functions can be wrapped around other functions by nesting another `function` element inside the current function element.

Attributes

type

The type of the function defined by this element. Currently the following function types are defined:

- lower
- upper
- trim
- normalize
- phonetic

Parents

These elements contain `function`: `column-def`, `column`

Children

The following elements occur in `function`: `function`

Examples

```
<function type="normalize">
  <function type="trim" />
</function>
```



Name

index — A Function-based index.

Synopsis

index ::=

- insert-function?
- select-function?

Attributes

- **name**
- **type**
- tokenizing
- wildcards

Required attributes are shown in **bold**.

Description

Defines a function-based index. The index can be used for searching in the column in which the index is defined.

The function-based index is composed of a `select-function` and an `insert-function`. The `insert-function` is applied when data is inserted into the column. The `select-function` is applied when searching on the function-based index of a column. Usually, the `insert-function` and `select-function` are equal.

Attributes

name

The name of the function-based index.

type

The type of the function-based index. One of: `integer`, `long`, `string`

tokenizing

If set to `yes` the content of the function-based index will be tokenized .

wildcards

If set to `yes`, searches with leading wildcards will be enabled for the function-based index. Searches with trailing wildcards are supported by default.

Parents

These elements contain index: `column-def`

Children

The following elements occur in index: `insert-function` , `select-function`



Examples

```
<index name="myindex" type="string" tokenizing=" " wildcards="true">  
  <insert-function type="phonetic" />  
  <select-function type="phonetic" />  
</index>
```



Name

insert-function — Insert function definition for a function-based index.

Synopsis

insert-function ::=

- function?

Attributes

- **type**

Required attributes are shown in **bold**.

Description

Defines an insert function for a function based index. The insert function is applied when data is inserted into the owning column. Functions can be wrapped around other functions by nesting another `function` element inside the current function element.

Attributes

type

The type of the function defined by this element. Currently the following function types are defined:

- lower
- upper
- trim
- normalize
- phonetic

Parents

These elements contain `insert-function: index`

Children

The following elements occur in `insert-function: function`

Examples

```
<insert-function type="phonetic">
  <function type="normalize />
</insert-function>
```



Name

select-function — Select function definition for a function-based index.

Synopsis

select-function ::=

- function?

Attributes

- **type**

Required attributes are shown in **bold**.

Description

Defines a select function for a function based index. The select function is applied when a search on the owning index is performed. Functions can be wrapped around other functions by nesting another `function` element inside the current function element.

Attributes

type

The type of the function defined by this element. Currently the following function types are defined:

- lower
- upper
- trim
- normalize
- phonetic

Parents

These elements contain `select-function: index`

Children

The following elements occur in `select-function: function`

Examples

```
<select-function type="phonetic">
  <function type="normalize />
</select-function>
```



Name

load-data — Loads data into a table.

Synopsis

load-data ::=

- Sequence of:
 - `column`

Attributes

- **file**
- **schema**
- **table**
- encoding
- separator

Required attributes are shown in **bold**.

Description

Loads a data file into a previously defined table.

Attributes

`file`

The name of the data file.

`schema`

The schema of the table to load data into.

`table`

The name of the table to load data into.

`encoding`

The encoding of the data file.

`separator`

The separator for the columns in the data file.

Parents

These elements contain `load-data`: `load-script`

Children

The following elements occur in `load-data`: `column`

Examples

```
<load-data table="mytable" schema="myschema"
```



```
    file="data.csv" encoding="UTF-8">  
  <column name="column_1" csv-column="0" />  
  <!-- ... -->  
</load-data>
```



Name

column — Specifies the data source of a table column.

Synopsis

column ::=

- function?

Attributes

- **name**
- **csv-column**

Required attributes are shown in **bold**.

Description

Specifies the origin of a table column when loaded from a data file. During loading of the data, a `function` can be specified, which will be applied before inserting data into the table column.

Attributes

`name`

The name of the target column.

`csv-column`

The column position in the data file from which the column data should be loaded.

Parents

These elements contain `column`: `load-data`

Children

The following elements occur in `column`: `function`

Examples

```
<column name="column_1" csv-column="0">
  <function type="trim" />
</column>
```





Appendix D. SQL Key Words



Table D.1, “SQL Key Words” lists all tokens that are key words in the Blackray SQL grammar.

SQL distinguishes between reserved and non-reserved key words. According to the standard, reserved key words are the only real key words; they are never allowed as identifiers. Non-reserved key words only have a special meaning in particular contexts and can be used as identifiers in other contexts. Most non-reserved key words are actually the names of built-in tables and functions specified by SQL. The concept of non-reserved key words essentially only exists to declare that some predefined meaning is attached to a word in some contexts.

Key Word	BlackRay	SQL:2008
ADD		non-reserved
ALL	reserved	reserved
ALTER	reserved	reserved
AND	reserved	reserved
AS	reserved	reserved
BETWEEN	reserved	reserved
COLUMN	reserved	reserved
COLUMNS	reserved	non-reserved
DROP	reserved	reserved
FROM	reserved	reserved
FULL	reserved	reserved
INNER	reserved	reserved
JOIN	reserved	reserved
LEFT	reserved	reserved
LIKE	reserved	reserved
LIMIT	reserved	
NATURAL	reserved	reserved
NOT	reserved	reserved
OFFSET	reserved	reserved
ON	reserved	reserved
OR	reserved	reserved
OUTER	reserved	reserved
RENAME	reserved	
RIGHT	reserved	reserved
SCHEMA	reserved	non-reserved
SCHEMAS	reserved	
SELECT	reserved	reserved
SHOW	reserved	
TABLE	reserved	reserved
TABLES	reserved	
TO	reserved	reserved
USING	reserved	reserved
WHERE	reserved	reserved





Appendix E. SQL Grammar Reference



Name

DROP TABLE — remove a table

Synopsis

```
DROP TABLE {name}
```

Description

DROP TABLE removes tables from the database.

Parameters

name

The schema-qualified name of the table to drop.

Examples

To destroy table phonetic in schema samples:

```
DROP TABLE samples.phonetic;
```

See Also

[alter table\(1\)](#)



Name

ALTER TABLE — change the definition of a table

Synopsis

```
ALTER TABLE {name} {action}
```

where action is one of:

- DROP [COLUMN] {column}
- RENAME [COLUMN] {column} TO {new_column}

Description

ALTER TABLE changes the definition of an existing table. There are several subforms:

DROP COLUMN

This form drops a column from a table.

RENAME COLUMN

The RENAME forms change the name of an individual column in a table. There is no effect on the stored data.

Parameters

name

The schema-qualified name of an existing table to alter.

column

Name of a new or existing column.

new_column

New name for an existing column.

Notes

The key word COLUMN can be omitted.

Examples

To rename column lastname to firstname in table samples.phonetic:

```
ALTER TABLE samples.phonetic RENAME COLUMN lastname To firstname;
```

To drop a column lastname from table samples.phonetic:



```
ALTER TABLE DROP COLUMN lastname;
```



Name

DROP SCHEMA — remove a schema

Synopsis

```
DROP SCHEMA {name}
```

Description

DROP SCHEMA removes schemas from the database.

Parameters

name

The name of the schema to drop.

Examples

To remove schema samples:

```
DROP SCHEMA samples;
```

Name

SHOW — show the values of a specified objects

Synopsis

```
SHOW {action}
```

where action is one of:

- SCHEMAS [LIKE { *schema* }]
- TABLES [FROM { *schema* }] [LIKE { *table* }]
- COLUMNS FROM { *table* } [LIKE { *column* }]

Description

SHOW displays schemas, tables or table columns matching specified criteria. There are several subforms:

SHOW SHEMAS

This form shows all schemas matching specified criteria.

SHOW TABLES

This form shows all tables matching specified criteria. If no schema is specified, the tables from the default schema will be shown.

SHOW COLUMNS

This form shows all columns from a table matching specified criteria.

Parameters

schema

The name (possibly wild carded) of existing schema to display.

table

The name (possibly wild carded) of existing table to display. The table name in the SHOW COLUMN must be schema-qualified.

column

the name (possibly wild carded) of existing column to display.

Examples

To display all existing schemas:

```
SHOW SCHEMAS;
```

To display all existing schemas starting with 's':



```
SHOW SCHEMAS LIKE 's%';
```

To display all tables in the default schema:

```
SHOW TABLES;
```

To display tables from schema samples starting with 't':

```
SHOW TABLES FROM samples LIKE 't%';
```

To display all columns from table samples.phonetic:

```
SHOW COLUMNS FROM samples.phonetic;
```



Name

SELECT — retrieve rows from a table

Synopsis

```
SELECT * | {expression...} FROM from_item ...  
[ WHERE condition ]  
[ LIMIT count | ALL ]  
[ OFFSET start ]
```

where *from_item* is one of:

- *table_name* [AS *alias*]
- *from_item* *join_type* *from_item* [ON *join_condition* | USING ([*join_column* ...])]]

Description

SELECT retrieves rows from zero or more tables. The general processing of SELECT is as follows::

1. All elements in the FROM list are computed. (Each element in the FROM list is a real or virtual table.) If more than one element is specified in the FROM list, they are cross-joined together. (See FROM Clause below.)
2. If the WHERE clause is specified, all rows that do not satisfy the condition are eliminated from the output. (See WHERE Clause below.)
3. If the LIMIT or OFFSET clause is specified, the SELECT statement only returns a subset of the result rows. (See LIMIT Clause below.)

Parameters

FROM Clause

The FROM clause specifies one or more source tables for the SELECT. If multiple sources are specified, the result is the Cartesian product (cross join) of all the sources. But usually qualification conditions are added to restrict the returned rows to a small subset of the Cartesian product.

The FROM clause can contain the following elements:

table_name

The schema-qualified name of an existing table.

alias

A substitute name for the FROM item containing the alias. An alias is used for brevity or to eliminate ambiguity. When an alias is provided, it completely hides the actual name of the table; for example given FROM foo AS f, the remainder of the SELECT must refer to this FROM item as f not foo.

join_type

One of



- INNER JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN

A join condition must be specified, either ON `join_condition`, or USING (`join_column` [, ...]).

A JOIN clause combines two FROM items. JOINS nest left-to-right. and binds more tightly than the commas separating FROM items.

INNER JOIN produces a simple Cartesian product, but restricted by the join condition.

ON `join_condition`

`join_condition` is an expression resulting in a value of type boolean (similar to a WHERE clause but only AND is supported) that specifies which rows in a join are considered to match.

USING (`join_column` [, ...])

A clause of the form USING (`a`, `b`, ...) is shorthand for ON `left_table.a = right_table.a AND left_table.b = right_table.b`

WHERE Clause

The optional WHERE clause has the general form

WHERE `condition`

`where condition` is any expression that evaluates to a result of type boolean. Any row that does not satisfy this condition will be eliminated from the output. A row satisfies the condition if it returns true when the actual row values are substituted for any variable references.

SELECT list

The SELECT list (between the key words SELECT and FROM) specifies expressions that form the output rows of the SELECT statement. The expressions refer to columns computed in the FROM clause.

LIMIT Clause

The LIMIT clause consists of two independent sub-clauses:

LIMIT { `count` | ALL }

OFFSET `start`

`count` specifies the maximum number of rows to return, while `start` specifies the number of rows to skip before starting to return rows. When both are specified, `start` rows are skipped before starting to count the `count` rows to be returned.

If the `count` expression evaluates to NULL, it is treated as LIMIT ALL, i.e., no limit. If `start` evaluates to NULL, it is treated the same as OFFSET 0.

Current Limitations

Only INNER joins are supported in the `join_type` element.

The number of joined tables specified in the FROM Clause is currently restricted to two.



Select from the disjointed tables (join_type element is not specified) is not supported.

Only logical AND operator is supported in the ON join_condition element.

The condition from the WHERE Clause can only be applied to the indexed column since table scan is not supported yet. See column-def for details.

Examples

To select all entries from table samples.phonetic:

```
SELECT * FROM samples.phonetic;
```

To select lastname column from table samples.phonetic where lastname starts with 'A' and 'B':

```
SELECT a.lastname FROM samples.phonetic AS a
WHERE a.lastname LIKE 'A%'
AND a.lastname LIKE 'B%';
```

To join tables samples.table1 and samples.table2:

```
SELECT t1.lastname, t2.address
FROM samples.person AS t1
INNER JOIN samples.address AS t2
ON t1.lastname=t2.familyname AND t1.firstname=t2.firstname;
```





softmethod gmbh
spieljochstr. 34
81825 münchen
fon +49 89 437787-0
fax +49 89 437787-99

www.softmethod.de

