Boolware
Operations Guide

softbool

SOFTWARE CORPORATION
http://www.softbool.com

Chapter 1
Introduction

Who should use this guide

The Boolware Operations Guide is written for database administrators or system administrators who are responsible for operation and maintenance of a Boolware Index server.

System requirements and server sizing

Boolware server runs on the Windows and Linux platforms (RHEL) and supports a number of data sources. Browse the Softbool web site for the latest list of supported platforms and data sources. Boolware also uses Apache Tika to extract text from a file (file system table) or from a binary field to be indexed. To be able use Apache Tika, Java 11 or later must be installed on the same computers as Boolware are installed.

Processor model:	X86 64-bits 12 physical core or **more recommended**
Number of open files:	For Linux at least 10.000 (ulimit -n 10000).
Memory:	32 GB or **more recommended** Needed memory depends on: number of connected users (sessions), type of use (sort and similarity requires more than a plain query), size of the database (number of records), online-update frequency etc.

General formula to compute the memory for each connected user is: ((no of records / 4) * 1.25).

Example:
If the current database contains around 15 million records each user requires around 4.7 MB.
If 500 users should be connected at the same time Boolware requires around 2.35 GB available memory.

Multiple network cards
When Boolware is started, Boolware tries to determine what external IP address the host machine has.

Boolware looks for a network card with the IP address last used by Boolware and if any network card has this IP address, the name of this network card will be used and also saved in settings.

Windows
If no network card has the same IP address last used by Boolware or if no IP address is specified, Boolware will use the first network card with a valid IP address.

Linux
If no network card has the same IP address last used by Boolware or if no IP address is specified, Boolware looks for the network card named "eth0" and if it is not found, Boolware looks for "em1" and if not found Boolware searches for the name "bond0" in the specified order. If none of these network cards are found, Boolware will use the first network card with a valid IP address.

NOTE!
If no IP address (ie empty IP address) is found according to any of the steps above, Boolware will log this in Boolware's system log and then terminate execution.

If another network card is to be used by Boolware, this can be changed using the Boolware Manager. Right-click on the top node of the Boolware Manager tree and select "Network Configuration". Read more about the network configuration in the Boolware Manager help file.

The default settings when communicating with Boolware are:
The Search application communicates via port 7008 (can be changed).
Boolware Manager communicates via port 14007 (can be changed).
The data source communicates via port 8008 (can be changed).
The nodes in a Boolware Cluster communicate via port 14700.

You could from outside, control if Boolware is active on the specified IP-address.
To do this, you send the STAT command over socket to port 14007 which give the following answer if Boolware is running:

STAT Answers OK if node is active regardless if the node is a member in a Boolware cluster or not.

To find out if the current node is inactivated in a Boolware cluster you send the ACTV command over socket to port 14007 which gives the following answers if Boolware is running:

ACTV Answers NO if the current node is inactive in a Boolware cluster.
Answers YES if the current node is active in a Boolware cluster or if the node is active but is not included in a Boolware cluster; i.e. "Standalone".

The following command - LBST - can be used to allow a load balancer to decide whether to send traffic to the current node or not. To find out if the current node shall receive traffic or not, the load balancer sends the LBST command over socket to port 14007 which gives the following answers if Boolware is running:

LBST Answers with YES or NO. This can be set via Boolware Manager or with the bwc command: `bwc -c setlbst <YES>|<NO>`

Boolware specifications and restrictions

This section defines the limits of a number of Boolware characteristics. Some values are design limits, but most are restricted by finite resource restrictions in the operating system or computer hardware.

Characteristic	**Value**
Max number of users (sessions) connected to Boolware Index Server	There is no single number for the maximum number of users the Boolware server can serve – it depends on a combination of factors including capability of the operating system, limitations of the hardware, and the demands that each client puts on the server.
Max. number of query terms and operators in a single query	The maximum number of operators and terms are only limited by the amount of available memory.
Max. Primary key length	126 bytes
Max. Index term length	126 bytes
Max. size of an .idx file	268 GB
Max. size of .ref file	1024 GB
Max. size of .pxw file	1024 GB
Max. size of .data file	1024 GB

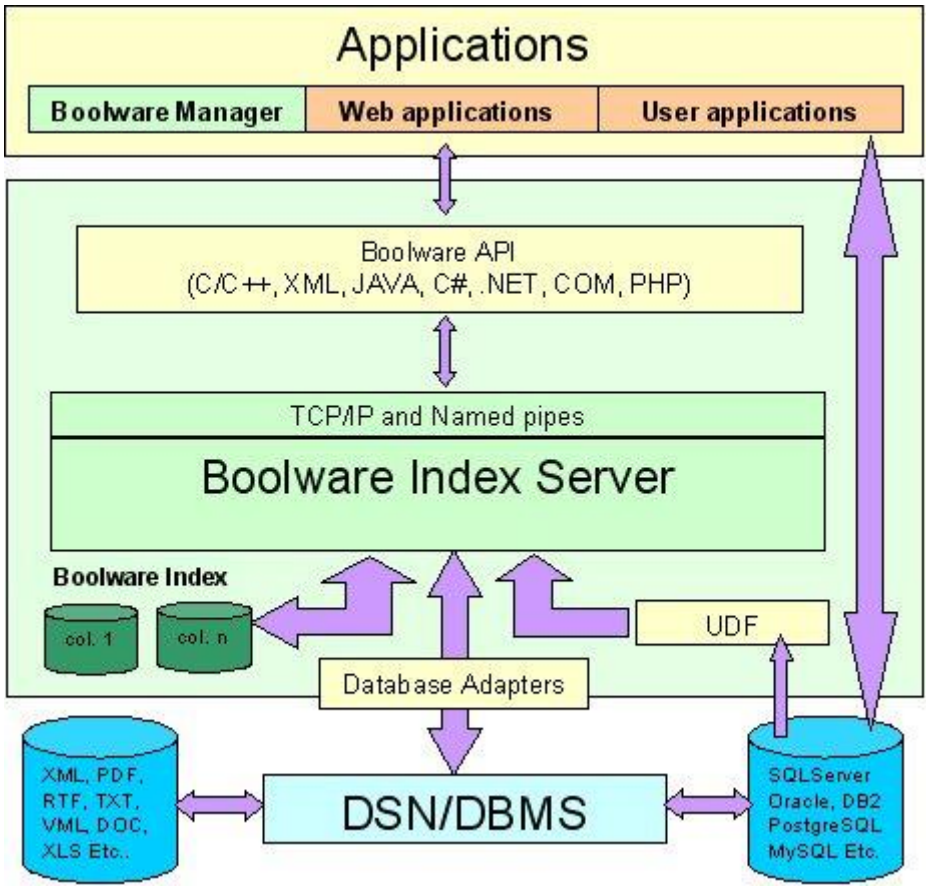
In addition, Boolware requires the following:

- The Table must have a primary key. The length of a primary key must not exceed 126 bytes (note that a primary key could be built up from several columns).
- A primary key is treated in many respects as a normal term; the primary key is **not** case sensitive in Boolware. Two primary keys built up by the same characters but different case are **not** unique in Boolware despite the current data source regards them as unique primary keys. Moreover all leading and trailing white-spaces: horizontal tab, vertical tab, form feed, carriage return, new line and space (0x09-0x0d, 0x20) are removed from the primary key before it will be saved in the primary key index. This means that Boolware does **not** distinguish between primary keys that only differ depending on leading and trailing white-spaces.
- All columns that are part of the primary key **must** have a value; no part could be "NULL".
- The Table cannot have the combination of columns of type text combined with referential integrity specifying cascading deletes.
- Boolware does not support schema completely. Schema could be used, but all tables **must** have **unique** names even if they appear in different schemas in a database.

Pictorial overview of Boolware

This is a pictorial overview of Boolware. The picture shows how Boolware - with different APIs - is connected to User written applications; XML-script and Boolware Manager. More information about the Boolware Clients (APIs): XML, PHP, .NET, Java, COM etc. could be found in the document: "Programmer's Guide".

The picture also shows how Boolware communicates with different data sources and Boolware Index.



The connection to the data source is done by "Adapters". There is one Adapter for each data source. A description of the Adapters will be found in Chapter 10 in this documentation.

Chapter 2
Installation

This chapter gives instructions for installing Boolware.

Installation of Boolware Index Server

There should be only one instance of the Boolware Index Server on the on the machine where Boolware should be running.

Following requirements must be fulfilled before installing the Boolware Index Server:

1. You must be logged in as administrator/root to be able to install Boolware in a proper way. You must be authorized to register and start the system as a service and Boolware must be authorized to create, read and write files etc.

2. The data sources you want to connect to Boolware after the installation of Boolware must be started and be accessible.

   Note! Each data source access client must be installed and accessible on the same server as Boolware will be installed. The owner of Boolware (the user) must have access to the environment variables (profiles where: language, paths etc. are stored) of each data source after the installation.

   It is very important that all products used together with Boolware Index Server are thread safe to ensure that Boolware Index Server should operate faultless.

3. Read more (see Chapter 10 Data sources) about special requirements during installation for respective data source.

Install the Boolware Index Server in Windows environment

a) Copy the file *Windows_Setup.exe* to local directory on the Windows computer
b) Logon as Administrator and start the install program *Windows_Setup.exe* or just start the program as Administrator
c) Follow the instructions in the installation program

Install the Boolware Index Server in Linux environment

a) Copy the *Linux_install.zip* to a local directory on the Linux computer
b) Logon as root
c) Unzip the file *Linux_install.zip* using an ordinary unzip utility e.g.
```
unzip Linux_install.zip
```
d) Add execution permission to the installation program e.g. `chmod +x install`
e) Start the installation program and follow the instructions as they occur: `./install`

The Boolware installation program (Windows/Linux) does the following:

=> Copies all necessary files to the installation directory

=> Registers Boolware Server as a service. When the platform is Windows you may have to use the Control Panel Services Manager to set-up the login account for Boolware Server, and how you want Boolware to be started.

=> For all data sources that support triggers and User Defined Function the installation programs Setupudf/Installudf will configure the data source by registering a User Defined Function called 'udf**_ds**' which is used by Boolware for online-updates. The current name of the data source should replace **ds**.

=> Starts the Boolware Index Server service.

Under Linux Boolware is started by "/etc/init.d/boolware start", this script will start another script, *boolwares*, which in turn will start Boolware as a service. The *boolwares* script will be generated at installation and will reside in Boolware´s installation directory, SOFTBOOL_HOME.

To stop Boolware under Linux type: "/etc/init.d/boolware stop".

In Linux it is possible to use the external memory manager *jemalloc* to get better performance and throughput at high load. In order to use *jemalloc* with Boolware, *jemalloc* need to be installed on the server and Boolware must be able to access the script *jemalloc.sh* that normally is installed in /usr/ bin. The Linux installation asks if *jemalloc* should be used or not.

Uninstall:
To uninstall Boolware in a correct way you must be logged in as administrator/root.

Start the program Uninstall.exe (Windows), uninstall (Linux) which is found in the installation directory of Boolware.

Install Boolware Index Server in an existing Boolware cluster including a Master and a Failover.

To install a new version of Boolware Index Server over an existing in a Boolware cluster you should follow the following procedure.

1. Disable online-updates in the data source.
2. Connect the Boolware Manager to the Master and set the Failover node **inactive** in the clusterdefinition and press Apply.
3. Route all traffic (searching) to the Failover node so no searching is performed on the Master node.
4. Stop Boolware service on the Master.
5. Install the new version on the Master (install over the old version).
6. Check that Boolware service has started on the Master. Use Boolware Demo to make a couple of searches to see that the new installed version is working as expected.
7. Route all traffic (searching) to the Master node so no searching is performed on the Failover node.
8. Stop Boolware service on the Failover.
9. Install the new version on the Failover (install over the old version).
10. Check that Boolware service has started on the Failover. Use Boolware Demo to make a couple of searches to see that the new installed version is working as expected.
11. Connect the Boolware Manager to the Master and set the Failover node **active** in the clusterdefinition and press Apply.
12. Route all traffic (searching) to both the Master node and the Failover node.
13. Enable online-updates in the data source

Using Boolware Index Server via a web server

The following extensions should be used when Boolware is used via a web server:

1. SoftboolXML.dll (Windows, IIS 7 or later)
2. libsoftboolxml.so (Linux, Apache 2 or later)
3. php_boolware.dll (Windows, PHP 5 or later)
4. php_boolware.so (Linux, PHP 5 or later)

These should be copied to the proper directory and configured as usual for the current web server.

Note
During the installation you could get messages that indicates that a file that should be installed is older than the corresponding file on the current machine. It concerns mostly two files, **default.chr** and **SoftboolSrv.ini**. If changes have been made to these files you should answer **Yes** (as recommended) not to destroy your changes.

Chapter 3
Overview Boolware Manager

The Boolware product include a graphical tool, with which you can perform every task necessary to configure and maintain a Boolware server. Moreover it is used to register and unregister data sources and to create and administer Indexes on the Boolware server.

Overview

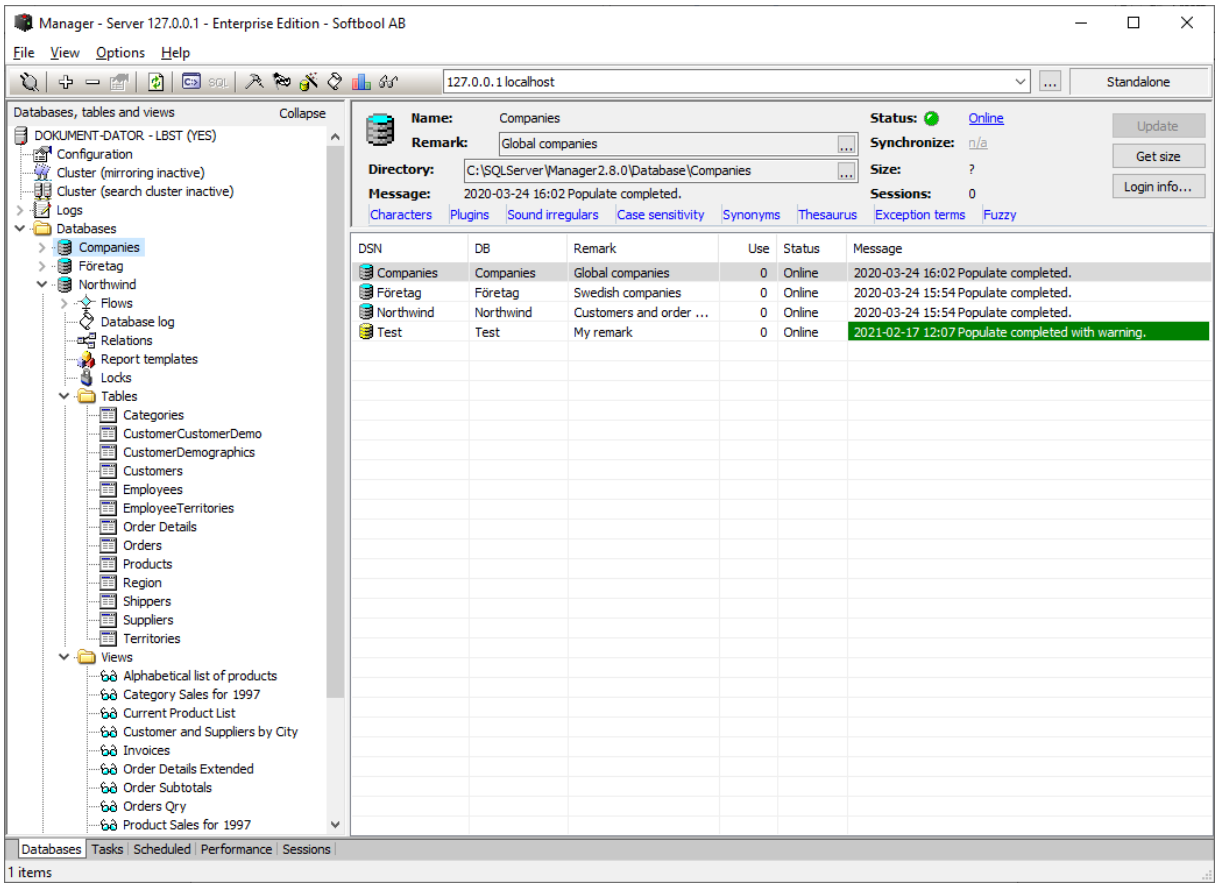
The Boolware Manager runs on Windows only, but can manage Boolware Server and Boolware Indexes running both on Windows and Linux.

Boolware Manager enables you to:

- Manage Boolware Server configuration
- Register / Unregister data sources
- Edit index- and search properties for a Boolware Index
- Create a Boolware Index
- View Boolware Server and Boolware index statistics
- Perform Boolware Index maintenance, including
  - Validating the integrity of a Boolware Index
  - Reorganize (defragment) a Boolware Index

The Boolware Manager window:

To start Boolware Manager, choose Manager from the Boolware Start menu. The Boolware Server Manager window opens:



The Menu bar, across the top of the window, contains actions you can choose to perform DBA tasks with Boolware Manager.

The Toolbar, a row of shortcut buttons for menu commands, just below the menu bar.

Under the Toolbar a tree structure of all Databases appears on the left side below where the databases Configuration, Cluster and Logs appear. On the right side all databases are listed.

Under each database in the tree structure you have the following alternatives: Flows, Database log, Relations, Report templates, Locks, Tables and Views. By using the right click menu in the tree structure you could perform many of the functions described below. An alternative is to use the database list at the right side.

The Status bar shows help for menus and the toolbar.

Boolware Manager menus

The Boolware Manager menus are the basic way to perform tasks with Boolware Manager. There are four pull-down menus:

File
This menu allows you to connect and disconnect to a server, register and unregister a data source, edit properties for a table, print and change the Print setup, reorganize the Sysindex and exit Boolware Manager.

View
This menu lets you view Boolware system statistics via the Server log, the Query log and the Indexing history. All valid server extensions and external plugins are listed under Server extensions. By using the Explore server... function you could see all files on the machine where the current Boolware server resides.

Options
This menu enables you to edit server configurations: server settings for stop words, synonyms, thesaurus, stemming and character sets. In this menu you could mange the configuration of the Boolware server. Moreover you could be directed to administration of ODBC- and other data sources. In this menu you could also tell in what language you want Boolware Manager and its help files to appear.

Configuration of the Boolware Server see chapter 9 "Maintenance of the Boolware system" section "Server configuration".

Help
This menu provides online help which could be activated anywhere when using the system.

Toolbar

The toolbar is a row of buttons which are shortcuts for menu commands. The toolbar buttons are:

- Connect to Boolware server (Connect)
- Register database with Boolware (New)
- Unregister selected database (Delete)
- Edit table indexing properties (Properties)
- Refresh the list (Refresh)
- System command to Boolware (Command)
- SQL-command for a Boolware record table (SQL)
- Build indexes for selected database/table (Build)
- Reorganize selected Boolware database/table (Reorganize)
- Validate indexes of selected database/table (Validate)
- View the last execution log for selected Boolware database (Log)

- Start the Boolware Dash board application
- Start the Boolware interactive test application (Test)

Also select a database and use the right click menu for more options.

Databases

The tab Databases shows all databases that are registered to the current Boolware Server.

DSN:	the name of the dsn that Boolware uses to connect to the datasource. This is usually the same as the corresponding database name.
DB:	the name of the corresponding database. (usually same as the DSN).
Remark:	a text that describes the database (only for information).
Use:	the number of online sessions currently using this database.
Status:	shows the current status. This may be "online" if the database is OK and accessible for online users. It is "offline" if the database is not accessible for online users. When load several indexes at once it can happen, if there are not enough resources, that an index can get the status "Pending". This means that as soon as resources are available, load is automatically started. Another status, "ReadOnly" exists; which means that an index can be searched, but not updated.
Message:	the most recent system message from Boolware.

Above this list of databases you could find some additional information on the marked database: Name of the database, Remarks for the database, Directory where the database is stored, Message from the last performed task, the current Status of the database and the number of currently attached users etc.. Moreover you could see if modified: Character sets, Index plugins, Sound irregulars, Case Sensitivity, Synonyms and Thesaurus appear on the database level.

Via the buttons to the right you can: start an online-update, get the size of the current database and change the login to the data source.

When a database is marked in the tree structure or is double clicked in the right hand list all tables/views for the database appear in the right hand list.

Above this list of tables you could find some additional information on the marked table/view: Name of the table, Remarks for the table, directory where the indexes are stored, message from the last performed task, the current Status of the table/view, the chosen update type, the number of rows in the table/view and the number of currently attached users. Moreover you could see if modified: Character sets, Index plugins, Sound irregulars, Case Sensitivity, Synonyms and Thesaurus appear on the table level.

By right click a marked database/table/view you can, among other things, perform the below functions. When a database is marked all tables/views that belong to (are owned by) this database will be affected. Some of the functions are just valid for tables/views: Import, Print table definition, Update and Properties.

Index	Builds new Boolware Index
Check/verify	Validates all Boolware files
Repair/reorganize	Removes unused space in all Boolware files
Stepwise indexing	Runs one Boolware step at a time
Synchronize updates	Changes update type or start an online-update
Print table definition	Prints all properties for the table/view and its columns
Configure	Some settings regarding sorting and online-updates.
Import	Import settings from an other table/view
Export	Export the settings of a database or table to another server
+	Add a new database/table/view
-	Unregister indexes for the marked database

Properties	Show properties for the table/view and its columns

Export database settings

Databases could be designed in many different ways and the content of the databases could differ a lot and moreover you will retrieve information in many different ways.

To achieve this Boolware has a lot of settings on different levels. Boolware choose a default setting depending on the content of the database, but all special adjustments must be performed by the user.

When you - in a test environment - have made all adjustments and are pleased with the result you can use this function to export all settings into a production environment.

See the chapter 19 Export/Import settings for a detailed description.

Tasks

The tab *Tasks* shows the progress of all tasks that are currently active.

Scheduled

The tab *Scheduled* contains all scheduled tasks and the time when they were last active and the time when they should be active the next time.

Use the +/- buttons to add or delete a scheduled activity. Double click on an existing activity to edit.

Performance

The tab *Performance* gives statistics on the following Boolware activities: Boolean queries, similarity queries, fetching data, get index terms, online sorting and different types of online-updates (Insert, Update and Delete).

Besides this statistics you get information on the current Boolware version and when it was started, memory used by Boolware, memory used now, total memory, memory available and the version on the current Operating System.

Here you can also see the time the current Boolware server has been running and the number of current, executing and rejected users.

You could at any time clear this statistics to measure performance during a certain time period.

Rejected users are those who were unable to execute in Boolware due to the high number of simultaneously executing users. How long these sessions will retry (default value 2 seconds) can be configured via Boolware Manager ("execution queue timeout" in Boolware server configuration in the Boolware Manager help file). If this value is high, you can adjust the value of "Max Threads" in Boolware Manager so that this value is as close to the number of processors as you have in the machine.

Sessions

The tab *Sessions* contains all logged in users in the Boolware system.

The following information on each user is listed:
The name of the current user, the data source used, the IP-address, the type of protocol used, the time the current user has been idle since last activity, the time the current user has been executing the current command and finally the command.

Chapter 4
Adding a database to Boolware

In order to make a data source searchable for Boolware, you first have to register the data source in Boolware.

In Boolware, you can register relational databases from different data sources/providers. User name and password can be specified if required.

To handle online-updates in this type of data source, "queue tables" and "triggers" are used. A detailed description on how to handle the queue table and the triggers will be found in Chapter 10 "Configuration and online-updates".

You can also register a so-called "Boolware Database" which uses different type of files as data source. To handle online-updates in this type of data source, "files" are used.

Boolware database

You can register a so-called Boolware database which uses files as a data source for its tables.

Other types of tables can be used; Link table, Record table, Filesystem table and Webcrawler table.

A Link table is a connection to a table from another database. A Link table will be treated by Boolware as a member of the database it is linked to. You could set Foreign keys in the Link table to be related to other tables in the current database. A Link table is owned by the database which it belongs to in the data source. All manipulation of the Link table: loading, online-updates, validating, reorganizing, editing indexing properties etc. could only be performed by the database that owns the table in the data source.

A Record table is a table that has a record file as a data source. You create the Record table and edit indexing properties in the same way as for a table in a relational database.

A Filesystem table, is a table which uses selected files in the filesystem as data source. This type of table has a set of fixed columns, which are managed by Boolware. The columns are: URL (primary key), Type (file suffix), Date, Size and Content (NCLOB). All data is stored in Boolware's internal data file.

A Webcrawler table contains rows and columns just like a normal table but uses files in an Apache Nutch database as the data source. Via the Manager, you can start a Crawl in Nutch to let Nutch retrieve pages from the website or start a Recrawl in Nutch to let Nutch go through pages that have already been downloaded and add changes to its database. When indexing a Webcrawler table, the pages are read from the Nutch database and the data is stored in Boolware's internal data file.

Create a new Boolware database

Add a Boolware database by clicking the "+" toolbar button in Boolware Manager.

Follow the instructions given in the Boolware Manager.

To get a detailed information consult the Boolware Manager Help function (Add database).

Create a new table

Mark the database and choose "New Table..." from the right click menu.

Follow the instructions given in the Boolware Manager.

To get a detailed information consult the Boolware Manager Help function (Boolware tables).

Boolware Record table

The Boolware Record tables contain rows and columns exactly as any other database table.

The input when loading a Record table is a "record file", where rows and columns are separated by user defined characters. A common format is a so called CSV format, where the rows are separated by "CR/LF" and the columns by comma (,).

Follow the instructions given in the Boolware Manager.

To get a detailed information consult the Boolware Manager Help (Boolware Record table).

Boolware Filesystem table

The Boolware Filesystem tables contain rows and columns exactly as any other database table.

The input when loading a Filesystem table is taken from specified files in the current file system.

Note that a Filesystem table can only be run if external text extraction modules are installed. See Chapter 17 "Text extraction in Boolware".

Follow the instructions given in the Boolware Manager's help file to create a Boolware Filesystem table.

Boolware Link table

A Boolware Linktable is a link to a physical table in another database.

Follow the instructions given in the Boolware Manager.

To get a detailed information consult the Boolware Manager Help (Boolware Link table).

Boolware Webcrawler table.

A Webcrawler table contains rows and columns just like a regular table.

Note that a Webcrawler table can only be run if Apache Nutch and the Boolware Webcrawler restserver are installed. See Chapter 18 "Boolware Website indexing".

Follow the instructions provided in the Boolware Manager help file to create a Boolware Webcrawler table.

Chapter 5
Editing indexing properties

This chapter describes how to determine which Tables/Views that should be indexed and which indexing settings all Columns in those Tables/Views can have.

Editing database properties

To perform database maintenance and configuration, first choose a database from the database tree view. Additional options are also available via the right-click menu.

Double-click a database in the database tree view to show all tables/views for that database. The checkbox for each table/view tells Boolware to index the table/view or not.

Double-click a table/view to show indexing properties for that table.

"Directory" is used to move the Boolware database - all included tables - to another directory. To do this the status of the database must have been set to "Offline".

When a database is registered for indexing in Boolware the system will read metadata from the data source for the current database (see Chapter 4).

Boolware uses these meta data to configure the indexes regarding tables and columns.

By fetching the properties for each column Boolware could set up a default configuration which is the most likely way you want to index the database.

The following applies when choosing the default settings:

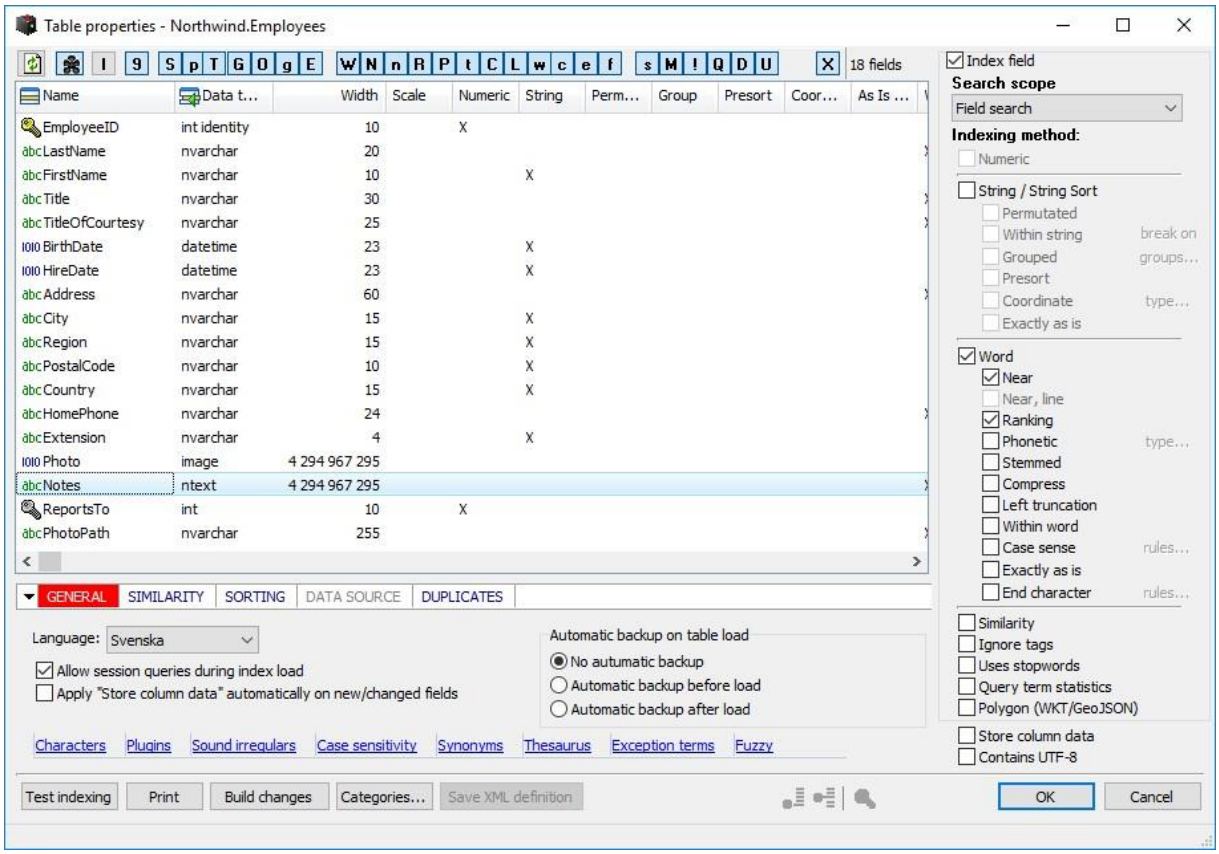
- all numeric fields in the data source (int, smallint, tinyint, numeric, float, real, decimal, money, smallmoney etc.) will get the indexing method numeric and this could not be changed.
- all character fields in the data source (varchar, char etc), where the size is less than 16 will get String as the indexing method.
- all character fields in the data source (varchar, char etc), where the size is greater than 15 will get the indexing method word.

A table in a data source could contain columns of type BLOB. Such a column contains a Binary Large Object which could be anything: pictures, bitmaps, compressed files, different types of word processing documents etc.

Boolware Server tries to identify the contents of these columns and ignores the following formats: bitmap, GIF pictures, JPG pictures and ZIP files.

Editing indexing properties for a table/view

If a table/view is marked for indexing double click on the table/view or right-click and choose "Properties" to get to the window "Table properties".



Some of the functions in this window are global - affect the complete table - while others only affect the marked column.

The global functions are found under five tabs: GENERAL, SIMILARITY, SORTING, DATA SOURCE and DUPLICATES.

Regardless of which tab is chosen the following buttons could be used when applicable: Test indexing, Print, Build changes, Categories... and Save XML definition.

Test indexing:
Using this function you could see how different words will be indexed by Boolware using the current settings. Change the settings and test indexing until the desired result is achieved without loading any Boolware index. You specify the words to be test indexed in the input field or if you leave the input field empty all words in the first 10 records with text from up to the first 100 records from the current table, will be test indexed.

Print:
When dealing with many tables containing several columns it could be convenient to save them as print outs. This function prints all indexing methods for all the columns within the table.

Build changes:
By choosing this option you could rebuild only columns that have changed settings. When changing index methods for a column, Boolware marks the name of the column in bold face and gives a message that the column must be rebuilt. When this option is used Boolware will re-build as little as possible. The bold face mark will be removed when the columns has been rebuilt.

The changes that will be done could be divided into six groups depending on their affect on the system:

1. Fetch from data source, wordextract, sort and indexing (as when the table is built)
2. Fetch from data source and build a Boolware data file; no wordextract/sort/indexing

3. All will be done in the index step and the existing index is used; no fetch from datasource/wordextract/sort
4. Fetch from data source and build special indexes; only affects the special indexes
5. Just remove files; no fetch from data source/wordextract/sort/indexing
6. Just change attributes in the descriptor; no fetch from data source/wordextract/sort/indexing

Below follows some examples of changes and what category they belong to:

Group 1:
When any of the following indexing methods - *word* and *string* are **added**. **Add** or **delete** of the following indexing methods: *compress* and *case sense*. The index methods *near* and *ranking* are linked together; *ranking* does not require a file when near is set. **Add** of *near* and **delete** of *near* when *ranking* is set will be handled in this group. **Add** of *ranking* when *near* is not set will also be handled in this group.

Group 2:
When one or more columns have been **added** or **deleted** from the Boolware data file. All columns stored in the Boolware data file must be fetched from the data source, but no sort/indexing is needed.

Group 3:
The following changes could be built directly from the existing Boolware index:
a) **add**/**delete** of: *left truncation*, *stemmed*, *grouped* and *phonetic*,
b) **add** of: *presort* and
c) **delete** of: *word*, *string* and *case sense*

Group 4:
The special indexes handled are: *Similarity*, *Categorize* and *Free-text*. In this case only words for the current special index will be handled; the original indexes will not be affected.

Group 5:
Delete of *presort* and **delete** of **all** "*Free-text columns*", "*Similarity columns*" and "*Category columns*".

Group 6:
Add/delete of alias. When near is set and have not been changed and *ranking* is changed (**add**/**delete**).

Changes in some columns - part of -: *Free-text*, *Similarity*, *Categorize*, *Sub field and Primary key* could cause other columns - part of the current special index - to be re-indexed.

Categories:
This button is used for the automatic categorizing setting. Automatic categorizing means that the records in a table are automatically grouped into classes (categories) depending on the contents of the record. In this dialogue you could: define the categories for the table, specify the column that should contain the calculated category, set margin between categories, set category dominance etc. This function is described in detail below in the paragraph: *Automatic categorizing*.

Save XML definition:
This button is used to save an XML definition. See a separate description on XML definitions in the below section: *Indexing of fields within columns*.

Important: The "Test indexing" button is warmly recommended for those that want to check which terms that are being generated in the index for the current field. You can supply your own test text. If you do not, Boolware will extract terms from the first hundred (100) tuples from the data source as sample data.

Under the tab GENERAL you find: "Language" and "Allow session queries during index load".

Language:
The language applies to the entire table and affects words with indexing method stemmed and phonetic. It will also affect which synonym-, thesaurus- and stop word file Boolware will use as well.

There is another setting of the language that affects the language used in the Boolware Manager. This setting is described in *Chapter 10 Settings*.

Allow session queries during index load:
If this option is activated the current indexes will be kept during the re-load and the query sessions could continue searching. When the load is completed the search sessions will be switched to the new indexes and the old will be deleted from the disk. During the load the Boolware indexes will occupy twice the normal size on disk.

Special settings for a Column:
Characters, Index plugins, Sound irregulars, Case sensitivity, Synonyms and *Thesaurus…*

Settings in the above will only affect the current column. Boolware tries to find specified settings in the following order: column, table, database and global (system). As soon a setting is found it will be used.

Characters:
Each column could have its own tables for character mapping. The character mapping in these tables will only affect the current column.

Plugins:
If a column should have custom indexing (see Chapter 14 "Plugins") the user written function for custom indexing and its parameters should be registered here. This custom index function will only take effect for the current column. You could for each query determine whether custom indexed terms should be used by a session setting. At start of a session the custom indexing is activated.

Sound irregulars:
Specify phonetic irregularities that are special for this column.

Case sensitivity:
Specify irregularities from given rules for case sensitivity for this column.

Synonyms
Specify synonyms that are special for this column.

Thesaurus
Specify thesaurus terms that are special for this column.

Under the tab SIMILARITY the following settings could be done: Similarity on column level, Vector dimensionality reduction and Custom stemming definitions. See section "Similarity Search" in Chapter 11 for further information.

Similarity on column level:
By activating this option you could re-fine the similarity to limit the similarity search on column level.

Vector dimensionality reduction:
The similarity could be adjusted to balance speed and relevance. When moving the arrow towards 100% (to the right) will improve the relevance but hurt the speed. Moving the arrow

towards 0% it will improve the speed but perhaps get less relevance. As different types of databases should use different settings it is recommended to experiment with this function. For example gives a value around 5% the best relevance when the table consists of plain text which contains a lot of "common" words.

Custom stemming definitions:
Another way to improve the relevancy when it comes to similarity is to use "stemming-tables" to specify certain words as similar. The name of this file is "*vsm.stem*". The file "*stemming.xx*" is global for all databases while "*vsm.stem*" is for one table only. The file "*vsm.stem*" can be considered as a synonym file with the important exception that the table must be re-indexed for changes in the file "*vsm.stem*".

Under the tab SORTING you specify the sort order when loading the database.

Initial Order expression:
You could specify a SQL syntax string (maximum 256 characters) telling Boolware the order to read tuples when building a Boolware index. The specified order is only guaranteed after a build of the database indexes. After an online-update the order is unpredictable. This option can be used if it is important to present the retrieved records in a specific order without the need of an online sort of the search result. **NOTE**: This option increase the build index time and is not applicable to Boolware tables.

Online sort settings:
This button will give a new dialogue with two tabs: Sort options and Update options.

Sort options:
This setting is only meaningful for columns that are indexed as *string* or *numeric*. In this case Boolware index could be used to perform a very fast sort of millions of records. However, when the index is very large and the number of records to be sorted is small it is more efficient to use some of the other Boolware sort algorithms. If the number of records to sort is less than the specified value Boolware index sort will **not** be used. The different Boolware sort algorithms are described in Chapter 11 "Interactive Query" section "Rank by Sort".

Update options:
By specifying a value you could affect the priority between searching and online-updates. When an Index is being online-updated we must "lock" the index from searching when the updated result for a word is written. During this time (milliseconds) no searching could be performed and the search has to wait until the updating is finished. If no value is set, the Index will be "locked" for just one word and then searching is allowed. If you, however, set a value N, the searchers will be "locked out" for N words and the searchers are "locked out" for a longer period of time. The higher value you specify the higher priority the online-update process gets.

Large online-updates in Boolware will be divided into "partial online-updates". By using some constants Boolware tries to divide the online-update into parts that are time efficient. The following constants are used in that algorithm: maximum no. of records 15.000, maximum no. of words (if not proximity indexed) 5.000.000 and maximum no. of words when proximity indexed 2.500.000. The "partial online-update" will start when any of these limits are met. In some cases this is not sufficient. Therefore, in Boolware Manager you can control this by specifying the maximum number of Insert and Delete records, to be included in a "partial online- update".

Do in the following way:
1. Start the Boolware Manager
2. Click on the "Command" button
3. A Boolware console command window will appear
4. Write: update config dsn 'dsn_name' tables 'table_name' with 'maxpartialupdaterows' value

In the last parameter you specify a value - the maximum number of Insert and Delete records that should be part of a "partial online-update" -. If the value is set to -1 you will disable this function and will relay on the Boolware algorithm described above.

Under the tab DATA SOURCE you specify where data is located and how it is organized when using a Bolware Database. See detailed documentation in Chapter 4 "Adding a database to Boolware", section "Boolware database" and parts "Boolware record table" and "Boolware file system tables". More information could be found in Boolware Manager help function.

Under the tab DUPLICATES you specify rules for eliminating duplicates from the current result.

In Boolware there is a way to eliminate duplicates in a search result.

To be able to do that, a rule for elimination of duplicates must be defined.

The maximum length of data that a duplicate rule can consist of and handle is 254 bytes, regardless of how many fields are included in the duplicate rule. The formula for calculating the total length of the specified fields in a duplicate rule:

Total length = (<Byte length of field 1> + 2 bytes) + (<Byte length of field 2> + 2 bytes) + ... + (<Byte length of field n> + 2 bytes).

Ex. We have the following three fields that should be included in a duplicate:

- id char (10)
- name varchar (150)
- personid char (8)

In the example above, the total length (10+2) + (150+2) + (8+2) = 174 bytes

Edit rule by editing each cell in a grid row or right click and select proper menu item.

Boolware can handle up to 100 different duplicate rules per table.

A duplicate rule consists of: "Rule Name", "Columns for duplicate identification" and "Columns for duplicate order".

"Rule Name" should be unique and is used when eliminating duplicates by applications using the command "dupeliminate(ruleName)" or "dupeliminaterandom(ruleName)" where the ruleName is the name of the rule to be used when eliminating duplicates in a search result.

"Columns for duplicate identification" consists of the column name(s) that specifies what is considered to be one / several duplicat records if the content of these columns are equal. If multiple columns are specified, they must be separated by commas.

"Columns for duplicate order" consisting of column names that sort the duplicates on the contents of these columns so the record that should be kept after duplicate elimination is first. If multiple columns are specified, they must be separated by commas.

Ex.
A rule called "ADDRESS" with the "Columns for duplicate identification" ' are 'Street' and 'StreetNumber' with internal sort order "Columns for duplicate order" is 'Site' in descending order.

Rule Name	**Columns for duplicate identification**	**Columns for duplicate order**
ADDRESS	Street, StreetNumber	Site desc

Since in some cases there is no given length in the table declaration, such as for subfields from an XML definition, you can enter the length of the respective subfield in the duplicate rule yourself. If no length is specified for a subfield in the duplicate rule, the system will use the length 50 bytes for the subfield in question.

To specify a length for a field in a duplicate rule, this is specified at the end of the field name within parentheses.

For example:
XMLfield/invoice/CustomerID(10), XMLfield/invoice/description(150), XMLfield/invoice/price(8)

If the checkbox "Build duplicate rules on load" is checked, the rules will be rebuilt automatically when loading or reindexing the table . Build/rebuild the duplicate rules can also be done via the menu command "Stepwise indexing" | "Run duplicate rules"

For extended duplicate, see the Boolware Manager help file, the "Boolware Indexing and Search" chapter, and the "Duplicate" section.

The below attributes could be set using the right click menu when a single column is marked.

Reindex column:
By choosing this option you could rebuild a single column. There are many reasons why a column should be rebuilt: change of indexing methods, a column has been added to the RDBMS, the index is corrupt etc. When changing index methods for a column, Boolware marks the name of the column in **bold face** and gives a message that the column must be rebuilt. The bold face mark will be removed when the column has been rebuilt.

Primary key:
The connection between Boolware and the RDBMS is handled by a unique identification for each row, the Primary key. This indicates that it always must exist a Primary key in a table/view that should be indexed by Boolware. A Primary key could be built up by several columns; this is necessary when the contents of one single column are not unique in the table. If a primary key is not set in the data source it must be done here e.g. for views.

Important: Each table/view must contain a Primary key to be indexed by Boolware. The total length of a Primary key must not exceed 126 bytes. Restrictions on Primary keys are described in Chapter 1 "Boolware specifications and restrictions".

Foreign key:
In a RDBMS there are usually relations between the tables included in the database. The relations are established by primary keys and foreign keys.

If no meta data relations are found in the data source you could use the Boolware Manager to establish these relations. Of course it is the responsibility of the user to guarantee that a specified relation really exists between the specified tables.

To establish a relation between two tables follow the below procedure:

1. Chose an alternative: Edit.../Delete
2. When Delete the current relation will be erased
3. When Edit; choose table and column which should be the relation
4. Boolware will save this relation and it will be used later on when searching.
   In Chapter 11 "Interactive Query" section "Related search (Join)" the query is described in detail.

Add alias:
There a two types of alias; a non indexed alias field or an indexed alias field. A non indexed alias field is a virtual field that only contains references - no data - to columns within the table. By this function you could: give a column another name than in the data source, "connect" several columns and in that way search several columns at the same time.
An indexed alias field combines the data from selected fields for the alias and will be indexed. See more information about indexed alias field in Boolware Manager Help file.

The most common way to use this function is to combine a couple of columns and give it an alias name used when searching.

You could: add, delete and edit the columns that should be part of the alias field.

In order to get a correct result when searching via an non indexed alias field, all columns selected should have the same indexing settings.

Delete alias:
Delete the current alias column.

Edit alias:
Edit (add/remove columns) the marked alias column.

Add XML definition:
To be able to achieve searching and presentation of fields within columns - especially when they are repeated - the text in the column should be XML coded.
See separate description in this chapter below; "*Indexing fields within columns*".

Delete XML definition:
Delete XML definition.

Save XML definition:
Save the current XML definition to optional file.

Change data type:
This function could only be used on a field that belongs to an XML definition. All fields in an XML definition will get data type *varchar* by default. The data type could be changed to numeric and *Width* and *Scale* could be set.

Some attributes on the right click menu go for the entire table: Copy to clipboard and Foreign key.

Copy to clipboard:
All columns and their data types and widths will be copied to the clipboard.

Foreign key:
Foreign keys could be set by Boolware. The user has the full responsibility that the chosen columns are correct.

If it is a record table you could change the Data type, Width and Decimals by double click a column.

Selecting a column will show that column's indexing settings in the check boxes to the right. For each column you can decide if it should be indexed, and if so – how it should be indexed.

An overview will show all columns and their attributes. By using the filter buttons you could choose attributes and all columns matching these attributes will show. For instance you could choose to show all numeric columns.

Index field:
First of all; you can control if a column is indexed or not by manipulating the "Index field" checkbox located in the frame.

Search scope:
Then – if the column is indexed – you can control the indexing scope: the Search scope can be set to either "Field search" (default), "Free-text search" or a combination of the two.

Free-text search means that any term found in the column will be globally searchable on the table level; i.e. you will be able to query tables without having to specify a column. This is a variant of non indexed alias field, which is described above in this chapter.

Numeric:
This data type is handled in a special way. You could not enable or disable this index method, but it is determined by the data type within the data source.

Different data sources have different names of numeric data types and Boolware adapts to the requirement of the data source. Some examples of data types that will be enabled as numeric are: int, smallint, tinyint, numeric, float, real, decimal, money, smallmoney etc.

If a column is marked as numeric you could take advantage of the very fast column sort in Boolware.

String:
Indexing a text field as *String* means that every text line of the field will be indexed as a whole, only leading and trailing white space will be trimmed. This means that words within the line will not be searchable separately. The line is only found if you search for the beginning of the line (string). The maximum length of an indexed line is 126 bytes. The pipe character (|) is interpreted as line break.

If the data type in the RDBMS is char or varchar and the size is less than 16, the default setting for a column is *String*.

If a column is indexed as String the fast column sort in Boolware could be taken advantage of.

Permutated:
Indexing Permutated is similar to string, but generates more keywords since the words on the line are combined in every possible combination (permutation). The permutations avoid the normal string restriction that each separate word is not searchable. The pipe character (|) is interpreted as line break. The permutation is stopped when end of line or a comma is found. For example:

A B C, E F will generate the following permutations:

A B C
A C B
B A C
B C A
C A B
C B A

At most four terms within a string will be permutated. This will generate 24 different strings.

Important: permutated indexes generate huge amounts of keywords and therefore large indexes. This in turn puts a heavy workload on Boolware. You may be better off using the *Near line* option for finding words that must be on the same line. The decision is yours.

Within string:
In the Boolware system there is a possibility to select the "Within String". This will index a "String" so both left and right truncation is fast.

Grouped:
Indexing *Grouped* means that the index terms are divided into groups. This will improve the search speed significantly when using interval searches or when truncating. The width in each group is editable.

Example:

Suppose that a column contains a date in the form yyyymmdd. If the group widths are set to 4, 6 and 8 all terms in this column will be grouped in year, month and day. The performance when searching for intervals including years, months and days will be extremely good.

Presort:
If this function is enabled the data in this column will be presorted to obtain fast sorting of static data. Only columns indexed as String or Numeric could be presorted. A more detailed description will be found in Chapter 11 "Interactive Query" in section "Rank by Sort".

Coordinate:
If this function is enabled coordinate data in this column will be indexed and searchable. By using the button "types..." you select the format; either WGS84 or RT90, System32/34.

Exact string:
This means that every text line of the field will also be indexed and stored as a whole but without any modification at all.

Word:
Indexing *Word* means that each word will be picked as a searchable term. Boolware has special character tables that define what sequences of characters that form a "word". These character tables can be inspected and changed using the Boolware Manager.

Important: The hyphen is treated specially when it occurs last on a line (syllabification), it will merge the two parts into one word.

Example:
she was the fast-
est swim-
mer

will generate
```
SHE
WAS
THE
FASTEST
SWIMMER
```

Near:
Checking *Near* will include a word number for each extracted word. This makes it possible to do proximity searches in the column. A proximity search dictates that all words must be close to each other. The closeness can be specified in the query (see *Chapter 11 Interactive query*).

For example, a standard Boolean query for "blue" and "car" will find all records where both "blue" and "car" appears. But the records may contain "blue hat" and "red car" – which may not be what you wanted. Using proximity search you may search and find only those records containing "blue car".

The reason that *Near* is an option, is because it makes online-updates and batch loads slow down a little. Moreover, the column index grows substantially (in average about 25 % of the column text in the data source).

Near line:
Near line is very similar to standard Near, but enables proximity within the same text line.

For example, if you have a structured company database where all members of the board are stored in a column (Board) - each member in a separate line - you could use the *Near line* to distinguish between the different members. If you search for companies that have John Smith as a member of the board without the option *Near line*, you might retrieve companies with the following members as well: John Larson, Steve Smith, Mary Smith etc.

Ranking:

If you want to use the contents in column for ranking the search result in a special order the *ranking* index method should be used.

The rank order is based on the occurrence of the search terms in the result. All search terms used from the last FIND command will be part of the ranking.
The ranking has two basic forms: occurrence and frequency.

When occurrence all search terms from columns marked for *ranking* will be counted in each record in the result. The record with the highest occurrence will be presented first.

When frequency the occurrence for the record is divided by the total number of words in the column and the value (frequency) is a number between 0 and 1 in the form 0.xxx. The record with the highest frequency will be presented first.

In the Boolware query language (QL) you could affect the occurrence (and thus the frequency) by specifying a weight on the search terms. The occurrence for these search terms will be multiplied by the weight and thus records containing these terms will be higher ranked.

Example of a weighted query (the column Text is marked for ranking):

FIND Text:**car**/9/
AND Text:**small**
AND Text:**blue**/4/

When the occurrence is calculated all occurrences of **car** will be multiplied by 9, **small** by 1 and **blue** by 4. This indicates that records containing blue car should be ranked high when presenting the result.

You could also set a weight on a rankable column which means that all searched terms within that all occurrences of terms within this terms will be multiplied by that weight. See Chapter 11 "Interactive Query" section "Ranking the result from a Query".

Phonetic:
Phonetic indexing enables searches based on how words are pronounced, rather than exact spelling. If phonetic search is enabled for a field, spelling will be relaxed when it comes to searching.

For example, you will find the company "Ericsson" even if you search for "Erikson" or "Erixon" etc.

Different languages have different phonetic rules, so Boolware uses different phonetic rules for different languages.

By default Boolware selects the proper phonetic rule depending on the language chosen for the current table, either Swedish, English, European or Soundex. By using the button "types..." you could, however, change the phonetic rule for the current column or for all columns in the current table.

Phonetic searches are based on the human hearing, rather than spelling. All equal-sounding letters, and combination of letters, are replaced with a sound symbol, thus producing a "hearing code". These soundcodes can be used to compare similarity, and is indexable.

Phonetic coding cannot handle permutated letters (typing errors) or if words are entered in wrong fields (for example a company name in the street address field). Also, sound coding is more or less focused at handling a specific "language room" well, and will work less good with other languages. For this reason, Boolware offers different phonetic coders.

Differences between the available phonetic coders

Soundex handles 6 different sound, no vowels (except when first), no digits and codes only the first 4 characters. It is the most simplified and general, which can be useful. But it also leads to producing a lot of unexpected matches, especially in large databases.

Enhanced Soundex handles 12 different sounds, no vowels, no digits and codes the first 16 characters. This gives it a lot higher precision than Soundex. However, it may still produce a lot of unexpected matches in large databases.

European handles about 40 different sounds, groups vowels, keeps digits and has no length limitation. It is designed to handle western European languages pretty well, and its precision is a lot higher than Soundex, but more relaxed than the Swedish/English methods.

Swedish and English handle about 100 different sounds, handles 6 different vowels, keeps digits and has no length limitation. These methods are optimized for their language, which give them high precision when used with these languages. However, due to this precision, they are unreliable for use with other languages. For example, the word "jean" is pronounced with a "jay-sound" in English, while it's pronounced as a "sch-sound" in French. Leading "G" most often sound as "G" in English and German, but almost always as "J" in French. In Swedish, it is "G" or "J" depending of it is followed by a hard or soft vowel, or a consonant.

By using Boolware Manager you can edit a list of phonetic synonyms. Here you can specify words that are not phonetically like but nevertheless should be treated as if they were the same.

Example: ltd(limited, inc, incorporated). This means that all the specified words will be sought when any of them appears in a phonetic search. This list can appear on a column level, table level and a database level. Regardless of phonetic rule the same list will be used.

Stemmed:
This is a technique where declined (nouns, pronouns, adjectives) or conjugated (verbs) words are normalized to their base forms. This is done to group words that have the same conceptual meaning, such as WALK, WALKED, WALKER, and WALKING. Hence the user doesn't have to be so specific in a query.

Special language dependent methods are used to achieve this. In addition you could specify irregularities, such as GO, WENT and GONE, in a special file: *stemming*.*, where * should be replaced by the proper language code (en, sv, fr etc.). See *Language* above in this chapter.

Compress:
This option reduces single letter abbreviations, forming a single word. For example the terms "I&B&M", "I.B.M." and "I/B/M" will all match a search for "IBM". The characters that could appear between the single letters are: blank, dot (.), slash (/), plus sign (+) and ampersand (&) ("so called skip-characters").

Note; if any of the above "skip-characters" - blank, dot (.), slash (/), plus sign (+) and ampersand (&) - has been marked as "Letter" or "Digit" in the charater table 'Word forming' it will be treated as 'Letter' and 'Digit' respectively; not as "skip-character".

Left truncation:
In most search engines using inverted files hard truncations are very time consuming. Especially this occurs when the first part of the search term is truncated (*ball) because the entire inverted files has to be read.

In Boolware there is a possibility to get around this problem by marking a column that is frequently used for left truncation by marking this column for *Left truncation*.
In this case the searches are as fast as "normal" searches.

The reason this indexing method is an option, is because it makes the index twice as big.

Within word:
In some cases a requirement could be to search a string within a word very fast; both left and right truncated search terms.

By using the indexing method "Within word" you avoid any search through the whole index in order to find the specified search term.

The reason this function is an option, is because it makes the index several times as big.

Case sense:
In most information retrieval systems you want to "normalize" words regarding case so you don't need to bother when searching. In some cases, however, it could be convenient to save the words exactly as it is stored in the database - to distinguish between the branch IT and the common word it -. To achieve this you could specify certain rules for a column regarding case sensitive. All words that fulfill these rules will be saved in the Boolware index both exactly as they appear in the database and in the normal way (upper case).

The rules that determine whether the word should be saved case sensitive are composed of three components: the number of upper case characters, exceptions from upper case and a combined synonym- and exception file (Case sensitive terms).

Upper case in this context is connected to the character mapping tables (Neutralize) in Boolware. Upper case could be specified as number of Capitals. This means that the word must contain at least that number of capitals to be saved case sensitive. If a Capital mask is specified the capitals in the word must appear in exactly the same position and number. In the Capital mask an upper case is specified as capital X and lower case by a lower case x. If both number of capitals and a Capital mask are specified the Capital mask will apply. If neither is specified two (2) capitals is used. Different types of Capital mask could be specified:

XXX To be saved as case sensitive the word should only contain three Capitals
XXx To be saved as case sensitive the word should start with two Capitals that must be followed by at least one lower case character.
XXX? To be saved as case sensitive the word should start with three Capitals that must be followed by at least one character (any case)
xXx To be saved as case sensitive the word should start with one lower case followed by one Capital which in turn should be followed by a at least one lower case

If "Min upper chars" is used and set to e.g. three, all words that contain three or more Capitals (anywhere in the word) will be saved as case sensitive.

In the Exclude chars you specify all "Capitals" that should **not** be regarded as Capitals when counting the number of Capitals. Characters that are regarded as Capitals by Boolware but are not letters are already defined and you just have to specify additional character that should not be regarded as Capitals.

To edit the synonyms and exceptions use the "Case sensitivity" option. Terms in this file should be saved as case sensitive despite they do not fulfill the specified rules. Assume that the rules are four Capitals. The syntax of this file is:

IT	IT will be saved as case sensitive
West	West will be saved as case sensitive
Woods	Woods will be saved as case sensitive
US(USA America)	US, USA and America will be saved as case sensitive and be handled as synonyms

Exact word:
This means that every word of the field will also be indexed and stored without any modification at all.

End character:
Used to specify rules for the removal of the end character for a term.

Similarity:
Similarity is an advanced indexing method that enables content based searches and comparisons, rather than just querying for single words. See section "Similarty Search" in chapter 11 for further information.

Ignore tags:
If your data contains data including tags (such as HTML, XML, WML, PDF or RTF) you probably do not want the elements (tags) to be indexed. If so: check Markup data, ignore tags to achieve this. In Filesystemtables and binary fields this is always checked.

Example:

```
<html>
<head>
<title>My page</title>
</head>
<body>
<h1>This is a header</h1>
Text…
</body>
</html>
```

will generate the following search terms:

MY, PAGE, THIS, IS, A, HEADER, TEXT

Uses stopwords:
This method instructs Boolware to avoid words that are found in the stop word list. "Stop words" are special terms that may be undesired when indexing.

The use of stop words is controlled by you, on the field level. A list of stop words per language, can be edited in Boolware Manager, alternative "Options / Stop words…".

Query term statistics:
In a column marked for Query term statistics the application will store search terms or search strings that have been specified when using Boolware. This table could later on be used to see what kind of queries that have been used in Boolware and at what time. The Query term statistic function is described in more detail in Chapter 11 "Interactive Query" in section "Statistics on Query Terms".

Polygon (WKT/GeoJSON):
Boolware can index closed polygons in order to let users find polygons that contains a given point. The polygon should be described by the standard WKT ("Well Known Text") POLYGON or MULTIPOLYGON or as GeoJSON "type" : "Polygon" or "type" : "MultiPolygon" to be handled by Boolware.

Example: In the data for a column there is a polygon described in WKT-format:
POLYGON((-15.0 0, 0.0 -15.0, 18.0 0.0, 0.0 14.0, -15.0 0.0))

Example: In the data for a column there is a polygon described in GeoJSON-format:
{ "type" : "Polygon", "coordinates" : [[[-15.0, 0.0], [0.0, -15.0], [18.0, 0.0], [0.0, 14.0], [-15.0, 0.0]]]
}

Store column data:
Enabling this function means that Boolware stores the data of the current column in an "internal" data file.

All data for all marked columns will be stored in this file.

The function should be used for fast presentation of query result in a hitlist. This list usually contains information from columns that identifies the retrieved records. The function is also very useful when sorting and could not use the fast Boolware Index sort; the column is indexed as word or you want to sort on more than one column.

If **all** chosen columns in the hit list are marked for "Store column data" Boolware will fetch the requested data from the "internal" rather than from the data source. This could be of great help if the data source is occupied by other resource demanding functions. Another reason is when the data source resides on another machine with poor connections.

Important: You should avoid to store data from columns that contain lots of data that is frequently online-updated, as it will slow down the online-update. If the "internal" file contains a lot of data the performance will decrease. The recommendation is to store only columns that should be used for fast presentation or sorting.

Contains UTF-8:
To allow a multi-language environment Boolware internally uses the UTF-8 encoding. This means that all data will be transformed into UTF-8. If data in a column already is stored as UTF-8 in a column not defined as a "Unicode" data type, you should denote Boolware here; otherwise the data will be transformed from UTF-8 into UTF-8 which will cause incorrect data.

Important: Checking multiple indexing options may make it impossible for Boolware to automatically determine which option that an application requests. The application inform Boolware what kind of query it desires by using the Boolware Query Language subcommands.

Under the tab *Sorting* you could determine whether the online sort should use Boolware index or the current data source. Here you could also control the priority between online-updates and online search sessions by using a slider. A description on the different sort algorithms you could find in Chapter 11 "Interactive Query" section "Ranking by sort".

Under the tab *Data source* you could do settings for the Record table and File system table which are using the Boolware internal Data source.

The best way to get a good understanding of all the possibilities in the system is to run Boolware Manager and use the help function to be guided on desired functions.

Indexing of fields within columns

Some databases are built up by huge text columns that are unstructured. In some of these text columns there could appear structured parts grouped as fields (sub-field). When searching, it would be nice to distinguish the information in the different fields. A typical example could be the verbal description of a company. In that text the board of directors are described. Each member is identified by: first name, last name, address, company, position within the company, age and sex.

A typical query could be: give me all companies where John Anderson from IMB is a member of the board of directors.

A way to achieve this is to use XML coding to specify the different sub-fields within the column.

Short description of XML

XML is a standardized way to describe complex data structures. XML is always stored as text, where the fields are denoted a special code called "element" or "tag", which marks where the field starts and ends.

Example:

```
<?xml version="1.0" encoding="UTF-8">
<chapter no="5">
<head>
<title volume="Boolware">Chapter X</title>
<subtitle>Indexing fields within...</subtitle>
<overview>This chapter describes...</overview>
  <pages>27</pages>
</head>
<body>
<h1>Short description of XML</h1>
<para>XML is a standardized way to...</para>
<para>Example:
<code>
  ……
</code>
</para>
</body>
</chapter>
```

An element is marked by an identification enclosed within <> e.g. <title>. The text that follows the leading element is regarded as the contents of the element and is ended by </title>.

Thus XML is used to describe structures. A program is required to interpret the text; this program is called an XML parser.

Boolware has such an XML parser that could interpret the contents of an XML coded text and thus could index the contents of the different elements.

The element names in the above example: "xml", "chapter", "title", "subtitle" etc. will not be indexed as they just are used to denote the field.

In Programmer's Guide you could read more and get examples on how Boolware handles XML.

Encoding

The contents in the different elements could be encoded as UTF-8 or ISO-8859-1. This could be specified in two different ways: specify the proper encoding in the attribute encoding or you could mark it in the *Contains UTF-8* in the Table properties window. In both cases the encoding is set for the entire column, which in turn could contain several elements (sub-fields).

If the attribute encoding contains UTF-8, Boolware will assume that all data in this column is encoded as UTF-8. Anything else specified in this attribute will make Boolware to treat the data encoded as ISO-8859-1.

If the attribute encoding is omitted the setting in the *Contains UTF-8* in the Table properties window will apply. If *Contains UTF-8* is marked the data will be treated encoded as UTF-8 else it will be treated as encoded as ISO-8859-1.

Indexing XML using Boolware

First you must mark the columns that contain XML code. This is done in the Manager by right click on the column and choose "Add XML definition...". Boolware requires a description file where you describe which part of the XML code that should be indexed.

XML-definition file

The XML-definition file, decription file, is a simple text file where you indicate which elements and/or attributes should be part of the indexing.

Note that the XML-definition file is - like all XML - case sensitive (you have to specify the elements exactly as they appear in the text).

Example of an XML Description file ("SOFTBOOL XML" must reside on the first row in the file):

```
SOFTBOOL XML
EXCLUDE: a/b/i/para/u
/chapter/head/title WORD STRING
/chapter/head/subtitle WORD
/chapter/head/pages NUMERIC(5,0)
/chapter/body WORD NEAR SIMILARITY RANKING
```

In this XML-definition file you could specify index attributes for each field.
The field /chapter/head/title will be indexed as word and string. The field /chapter/head/subtitle will only be indexed as word. The field /chapter/head/pages will be indexed as numeric and you have to specify the number of integers and decimals. Finally the field /chapter/body is indexed for: word, proximity, similarity and ranking. These indexing properties can be edited by Boolware Manager.

Valid indexing settings in an XML definition are:

NOINDEX
FIELDSEARCH|FREETEXTSEARCH|FIELDSEARCHFREETEXTSEARCH
NUMERIC[(12,2)]
STRING
PERMUTATED
WITHINSTRING[('breakchars')]
GROUPED[(2,4,6,0,0,0)]
PRESORT
COORDINATE[(WGS84|METRIC,'latfieldname,longfieldname'|'sepstring'[,'commaseparatedfieldnames','fieldseparator','maxbytesperfield'])]
EXACTSTRING
WORD
NEAR
NEARLINE
RANKING
PHONETIC[('0|1|2|3|4|5'[,'allfields'])]
STEMMED
COMPRESS
LEFTTRUNCATION
WITHINWORD
CASESENSE[('casemask'|'mincaseletters'[,'excludechars'])]
EXACTWORD
ENDCHARACTER[('endingletters,...','minwordlen'[,'doubleletters,...'[,'allfields']])]
SIMILARITY[('threshold')]
IGNORETAGS[(FILTERBEFORE|FILTERAFTER,"'<starttag,endtag>","<starttag,endtag>"...','charsetname',XMLELEMENTS|XMLATTRIBUTES|XMLBOTH)]
USESTOPWORDS
QUERYTERMSTATISTIC

POLYGON
STORECOLUMNDATA
CONTAINSUTF8

Boolware can also index content stored within a CDATA section if the CDATA section contain a complete XML structure. Read more about this in the Boolware Manager Help file.

Ignore tags

In some cases the XML code could contain elements that do not have any meaning for the document structure; elements that marks typographical information as <b> for **bold face**, <i> for italic etc. As they appear in the document, in the same way as other elements, they will make it harder to decide what parts belong to the field to be indexed. In Boolware there is a possibility to mark what elements to ignore when indexing. In the description file you could specify these elements after the keyword EXCLUDE:, which must appear as the first word on a line. The different elements to be excluded should be separated by a slash (/).

Example of an XML Description file ("SOFTBOOL XML" must reside on the first row in the file):

```
SOFTBOOL XML
/chapter@no
/chapter/head/title@volume
/chapter/head/title
/chapter/head/subtitle
/chapter/body/code
```

Select a column that contains XML and right click and choose "Add XML definition...". Choose the associated XML definition file and click OK. Boolware will read the definitions and include them in the description of the database. The elements that are specified in the XML description file are called sub-fields.

Index attributes of an element

In Boolware even element attributes in xml could be indexed. This is done by including them in the XML Description file and specify which attributes should be indexed. In the above example there are two attributes that should be indexed; ‘no’ in element <chapter> and volume in element <title>.

Prefix indexing

Prefix the content of an element with attribute values to be able to separate different occurrences in an XML document. This is done according to X-PATH syntax.

Example of prefix indexing: "chapter/head/@language/title". This means that all data in the element "chapter/head/title" will be prefixed with the content (value) from the attribute "chapter/head/@language".

All element data with the same element name

To be able to find element data by the element tag name independently of its hierarchic position precede the tag name with two slashes ’//’. This must be preceded with at least the root element in the hierarchy.

An example of this could be "chapter//head". This means that all data in all elements with the name "head" below the root element "chapter" will be stored in the in the same Boolware index.

Set properties for a sub-field

The properties could be set for a sub-field in the same way as for a column via the Boolware Manager.

Limits

The Boolware XML parser handles all well-formed XML documents. By well-formed documents means that element is started and ended in a correct way.

These are the limitations:

- Documents cannot be validated.
- External DTDs cannot be used.

If validation is a demand the text should be validated before it is loaded into the database.

Error messages

If the Boolware parser detects an error it will be noted in the database log file (when loading) or in the Boolware Server log file (during online-updates).

Example

Below is a very simple example on how to use the "Prefix indexing".

This is a very short Boolware XML description file based on the two XML documents below.

```
SOFTBOOL XML
EXCLUDE:
article/originators/bor@TYPE/originator/names/nationality@NAT/name
```

In this example there are two prefix groups: TYPE and NAT.

The prefix group TYPE contains two attribute values: Author and Reader, and the prefix group NAT also contains two attribute values: Swedish and English.

In Boolware you could have a maximum of 50 prefix groups in an XML document. Each prefix group could contain an unlimited number of prefixes (attribute values).

Each prefix should be enclosed within quotation marks (") and each prefix group should be ended by a comma (,). The last (or only) prefix group should be ended by a semicolon (;) instead of a comma (,). The semicolon separates the prefixes from the "normal" search terms. See Chapter 11 "Interactive query" for the proper syntax of the prefix sub command.

The following Boolware Query Language example shows how to find "Jan Guillou" only when he is the "Author" and the nationality is "Swedish":

```
FIND
TEXT/article/originators/bor@TYPE/originator/names/nationality@NAT/name:Prefix
("Author", "Swedish"; Jan Guillou)
```

The following two Boolware Query Language examples shows how to find "Jan Guillou" and the nationality is "Swedish" but regardless of "Author" or "Reader":

```
FIND
TEXT/article/originators/bor@TYPE/originator/names/nationality@NAT/name:Prefix
("Author" OR "Reader", "Swedish"; Jan Guillou)
```

As Author and Reader are the only specified attribute values for the prefix group TYPE you could specifythe above query in the following way:

```
FIND
TEXT/article/originators/bor@TYPE/originator/names/nationality@NAT/name:Prefix
("*", "Swedish"; Jan Guillou)
```

The following Boolware Query Language example shows how to find "Jan Guillou" OR "Stephen King" only when they are the "Author" and regardless of the nationality:

```
FIND
EXT/article/originators/bor@TYPE/originator/names/nationality@NAT/name:Prefix(
"Author", "*"; (Guillou Jan) OR (King Stephen))
```

In the above example a word search on Guillou AND Jan and on King AND Stephen is performed. The two results are OR:ed to the final result. In this case you could specify the terms in any order and get the same result:

```
FIND
EXT/article/originators/bor@TYPE/originator/names/nationality@NAT/name:Prefix(
"Author", "*"; (Jan Guillou) OR (Stephen King))
```

In the below example a string search is performed and in this case it is important to specify the terms in the proper order. The result should be the same as in the two previous examples:

```
FIND
EXT/article/originators/bor@TYPE/originator/names/nationality@NAT/name:Prefix(
"Author", "*"; "Jan Guillou" OR "Stephen King")
```

Here follows a description on the two xml documents that the above example refers to:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<article>
  <originators>
    <bor TYPE="Author">
      <originator>
        <names>
          <nationality NAT="Swedish">
            <name>Jan Guillou</name>
          </nationality>
        </names>
      </originator>
    </bor>
  </originators>
  <originators>
    <bor TYPE="Reader">
      <originator>

        <names>
          <nationality NAT="Swedish">
            <name>Thomas Bolme</name>
          </nationality>
        </names>
      </originator>
    </bor>
  </originators>
</article>

<?xml version="1.0" encoding="ISO-8859-1"?>
<article>
```

```
  <originators>
    <bor TYPE="Author">
      <originator>
        <names>
          <nationality NAT="English">
            <name>Stephen King</name>
          </nationality>
        </names>
      </originator>

    </bor>
  </originators>
  <originators>
    <bor TYPE="Reader">
      <originator>
        <names>
          <nationality NAT="Swedish">
            <name>Jan Guillou</name>
          </nationality>
        </names>
      </originator>
    </bor>
  </originators>
</article>
```

Automatic categorizing

When should automatic categorizing be used

In some databases an automatic categorizing of new and changed records could be very useful.

A typical example is a newspaper database containing articles where the categories (classes) could be: Domestic, International, Economy, Culture and Sport. When all the settings for automatic categorizing are done the system will calculate the most probable category for each new or changed article. This stamp of category could be used later on when searching to include only one or a couple of categories.

Example:
In the above newspaper database you are interested in finding articles containing the name Johnson when it is part of a sport article. If you specify Johnson as the only search argument (FIND Text:Johnson) you will get a lot of articles dealing with Domestic, International, Economy, Culture and Sport. If you also specify the category, sport, you are interested in (AND Stampcol:sport) you will only get sport articles containing the name Johnson.

How could you affect the automatic categorizing

As the behavior of the automatic categorizing depends on many different factors, the Boolware system offers a number of different settings to tune the automatic categorizing to work in the best way. The settings are performed in different windows, where each window handles different types of settings. The window "Categories" contains the comprehensive settings, the window "Categorization settings" handles what terms to include and how they should look and finally the window "Define category" where you could examine and edit the category files.

The goal with the different settings is to get the best possible category files which are used when performing the automatic categorization. The best way to get a good automatic categorization is to be sure that the category files are as good as possible from the beginning.

Requirements for the automatic categorizing

The following definitions will be used in the below description:

1. Category file; this is a file that contains "typical" terms for a category. When a category is defined this file is created and is empty. To be able to do the automatic categorization each category must have one category file. By comparing the content in the new incoming record to each category file the most probable category is calculated.

2. Ground for categorizing; the columns that holds information that the categorizing should be based upon should be marked in the window: "Categorization settings". It is the information in these columns that are compared to the different category files to determine the most possible category.

3. Stamp column; this column is used to hold the calculated category. This column is part of the table and could be manipulated in the same way as any other column. It is only records not containing any information (NULL) in this column that will be automatically categorized by the system. By a setting you could ask the system to save more than one category per record in the stamp column. This is useful when the calculated scores from two or more categories are very close after an automatic categorizing.

The above three conditions must be fulfilled to start an automatic categorization:
1. there must be at least two categories,
2. there must be at least one column marked for the categorization to be based on and
3. there must be a stamp column.

Important: When categorizing a database - Categorize -, online-updates in the data source will not be taken care of. During the categorizing the mechanism that handles the synchronization of the Boolware Index (triggers) is closed. This means that all pending online-updates must be run before the categorizing starts. When the categorizing is finished Boolware automatically enables the mechanism that handles the synchronization (creates new triggers).

How does the automatic categorizing work

This section gives a comprehensive description on how the automatic categorizing works. In special sections below all possible settings are described in detail.

The calculation of the most probable category is based on the contents in the category files. Initially, when the database is empty the category files must be created by the user. The most common way is to copy some "typical" articles for each category into the proper category file. It is very important that the contents of these category files are relevant. By removing common (not relevant) information and adding relevant terms you will get the best basis for a good categorizing. After the build and the following online-updates the database will contain automatically categorized records based on the quality of the category files.

The automatically calculated category for the record will be saved in the specified stamp column. The user could state that the system should save more than one category in the stamp column when the score of the calculated categories are very close (the system could not unambiguously determine a category).

To be able to find all records stamped with multiple categories the system is provided with a sub command *dupstamp*() (see description of syntax in Chapter 11 "Interactive Query"), which will retrieve all records stamped with more than one category.

When the database contains a couple of thousand records it is recommended to manually adjust the value in the stamp column if it has been wrongly categorized by the system. Just exchange the current value in the stamp column for the correct one.

When the entire database is adjusted regarding the automatic categorizing you could create new category files from the contents in the database. The new category files are based upon the contents of the existing records in the database. There is a special individual step in Boolware Manager "Generate new category descriptions" which generates these category files.

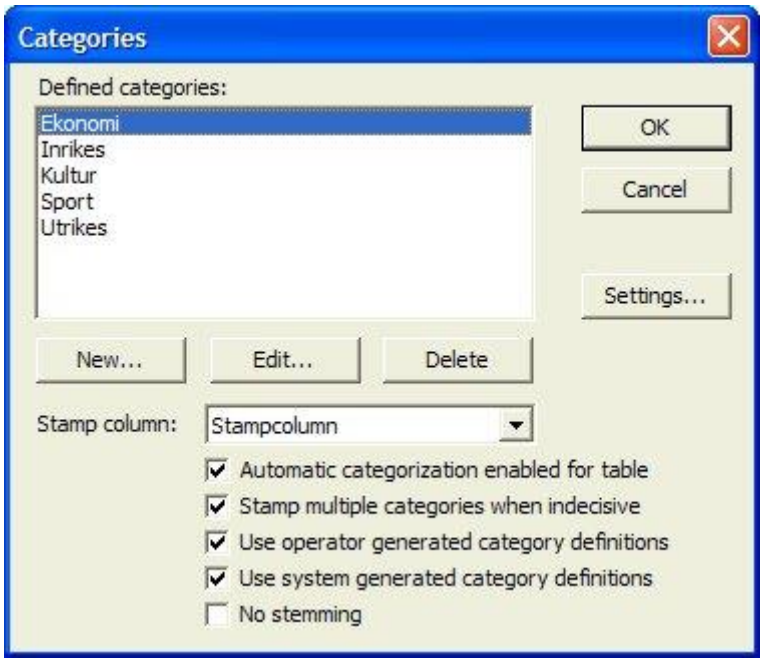
When does the automatic categorization take place

The automatic categorizing takes place when building and online-update the database, but it could - at any time - be run as a separate step in Boolware Manager.

In order to automatically categorize a record the stamp column in the data source must be empty (NULL).

Settings for automatic categorization

By pressing the button "Categories…" in the window "Table properties" you will get the following window:



Defined categories

The first thing to do is to determine which categories should be part of the database. In the above example, the newspaper database, the following categories have been chosen: *Domestic*, *International*, *Economy*, *Culture* and *Sport*.

Each category is defined by a category file which contains "typical" information about the current category.

By pressing the button "New…" you will get a window where you should specify the name of the category and insert text which is typical for the category. The most common way is to "copy" articles which are typical for the current category. It is very important that the chosen text is relevant for the category. This procedure is repeated for each category.

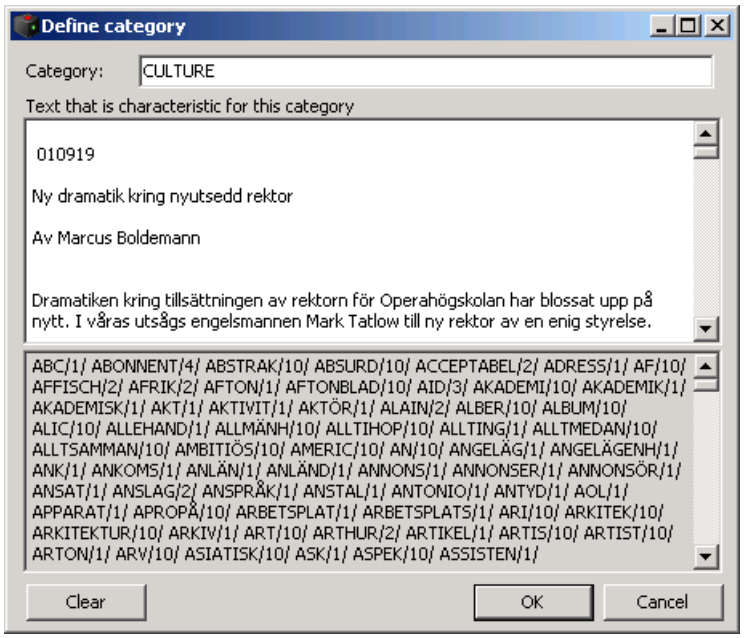
Another way to define a category file is write words that are very significant for the current category. If there are words that normally appear several times in the articles you could specify

these words in the following way: *term*/*n*/, where *term* is the word and *n* is the number of time it should appear. This means that you do not have to repeat the word several times.

There is of course a possibility to add, delete and change the categories. If you press the button "Edit…" you could adjust the contents of the current category file. By pressing the button "Delete" you could remove the current category.

The name of the category files are set by the system and must not be changed.

By double-clicking a marked category or by pressing the "Edit…" the following window will appear:



The upper part will contain the text created by the operator ("Use operator generated category definitions"), while the lower part will contain text generated by the system ("Use system generated category definitions") from the contents of the current database.

The text in the upper part could, of course, be edited by the operator, while the text generated by the system only could be erased by using the button "Clear".

Setting of Stamp column

A special column, the Stamp column, should be specified. This column will contain the calculated category for the current record. When this column is empty (NULL) Boolware will calculate the most probable category for this record and save it in the Stamp column.

If this column has a value this value will apply; the system will not automatically calculate a category and save it.

Enable/disable the automatic categorization

When the automatic categorizing is enabled for the table all records - added or changed - will be automatically categorized if the stamp column is NULL (the column is empty).

If the automatic categorizing is disabled no automatic categorizing for added or changed records will take place. The categorizing could be done at a later occasion by using the individual step "Categorize documents" in the Boolware Manager.

Stamp multiple categories when indecisive

If stamp multiple categories when indecisive is enabled all categories that are very close when the score is calculated will be stored in the stamp column. The categories will be sorted so that the best choice is presented first. What you mean by close is a setting in the window

"Categorization settings" (see below). If disabled only the "best" category will be stored in the stamp column. All records stamped with more than one category could be retrieved by using the Boolware sub command dupstamp() (see Chapter 11 "Boolware Query language").

Use operator generated category definitions

If this function is enabled the system will use the category definitions generated by the operator. If "Use system generated category definitions" is enabled as well, both definitions will be used when calculating the categories.

Use system generated category definitions

If this function is enabled the system will use the category definitions generated by the system. If "Use operator generated category definitions" is enabled as well, both definitions will be used when calculating the categories.

No stemming

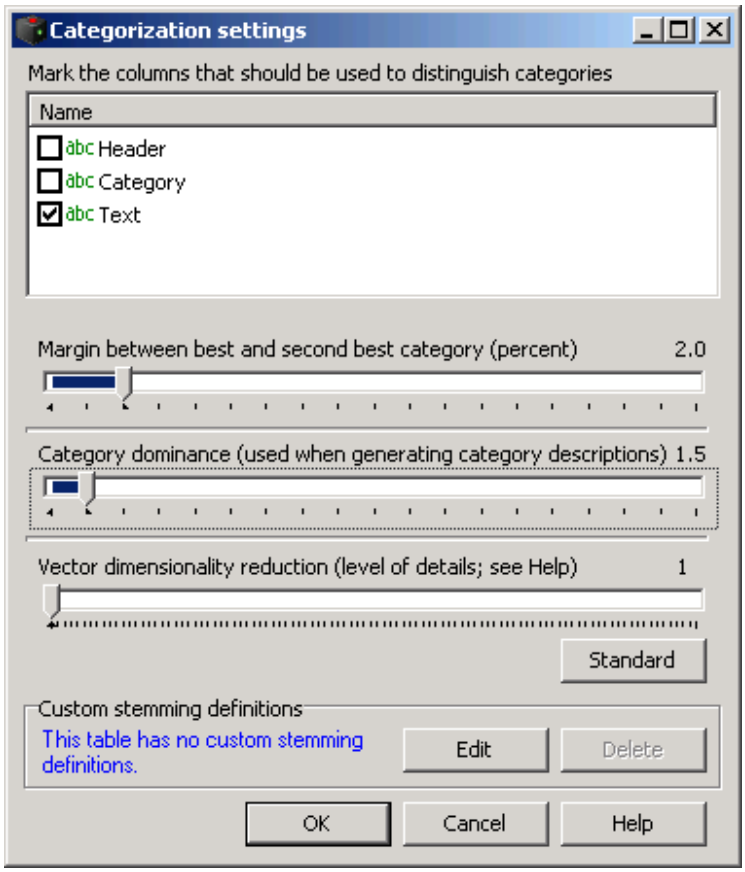
When Boolware treats terms from records which are in the table and from category definitions generated by the operator they are normally stemmed. This means that each term will be "normalized"; all conjugations for verbs all endings for nouns and adjectives will be removed so that only the "stem" of the term is used.

If "No stemming" is enabled the terms will be saved as is. This could be very useful if the category definitions generated by the operator only contains "keywords", which should not be manipulated when build category definitions generated by the operator.

Both the category definitions and the records in the table must be affected by the new setting.

Important: When the setting "No stemming" is changed it is very important to re-build the current table indexes.

By pressing the button "Settings…" the following window will appear:



Mark the columns that should be used to distinguish categories

When activating the button "Settings…" in the Categories window the "Categorization settings" window will show all columns that could be used to distinguish the categories. Information from the marked columns will be used when calculating the best category for each record.

Example:
In our newspaper database we select and mark the column (Text) that contains the text for the current article.

Setting of margin between best and second best category

The margin could be set to a value between 1.0 and 10.0 percent by using the slide control. The value is used to determine a warning level. When calculating the category scores for a record the scores for several categories could be very close; the record is qualified for more than one category. All categories that have a score that are closer to the best score than the specified value will be written to the database log file.

If the "Stamp multiple categories when indecisive" is enabled all the categories written to the log file will be stored in the stamp column as well.

Example:
In our newspaper database is the margin between the best and second best score set to 2.0%. During a load of the database the score from two categories for one record only differs 1.5% from the category with the best score. The best category is *Domestic*, the second best is *Culture* and the third best is *International*. Both *Culture* and *International* are closer to *Domestic* than 2% and thus the three categories and the identification of the record will be written to the

database log file. If the "Stamp multiple categories when indecisive" is enabled the *Domestic*, *Culture* and *International* will be stored in the stamp column as well.

Category dominance

This setting is used when creating new category files (see "Generate new category descriptions" below).

The setting is done by the middle slide control. The category dominance could be set to value between 1.0 and 10.0.

Which category a term belongs to is described below (see "How to run the automatic categorizing").

A frequency will be calculated for each term. The frequency is based on the number of records the term is contained in within each category. The frequency is then adjusted with respect to the category that contains most records.

A term is regarded as dominant within a category if the frequency for that category is much higher than for all other categories. How much higher is determined by the value set in the category dominance. If the value is set to 1.5, the frequency for the best category must be at least 1.5 times higher than the frequency for all other categories to be regarded as dominant.

Example:
In our newspaper database the category dominance is set to 1.5. A term has got the following frequencies for the different categories: *Domestic* (10), *International* (8), *Culture* (4), *Economy* (2) and *Sport* (16). In this case the term will only appear in the Sport category file as it is category dominant (the frequency for sport (16) is more than 1.5 times any of the other frequencies).

Vector dimensionality reduction for categorizing

The third slide control in the "Categorization settings" window specifies which words to ignore when the terms that should be used to distinguish categories are extracted. The higher value the less terms will be ignored. This setting works in the same way as the slide control for similarity search (see "Similarity options" earlier in this chapter).

This value is very hard to predict and thus it has to be tested to be relevant. As a rule you could say that normal text, which contains a lot of common words contained in all records are very frequent. In this case the value should be very low, around 5% to get the best result. On the other hand when the text contains a lot of unique terms the value should be much higher.

Custom stemming definitions

In the "Categorization settings" window you could specify a file that contains synonyms to improve the relevance when categorizing. This file works in the same way as the custom stemming definition file for similarity search (see "Similarity options" earlier in this chapter).

How to run the automatic categorizing

As mentioned earlier records are automatically categorized when loading and online-update the database if the three above conditions are met. There is also a possibility - at any time - to affect the automatic categorizing by running two individual steps.

The two individual steps that affect the automatic categorizing are named: "Categorize documents" and "Generate new category descriptions". Both can be activated via Boolware Manager.

Categorize documents

Categorize documents means that all records that have no value (NULL) in the Stamp column will a category calculated by the system. If the Stamp column contains a value, this value will be the category for the current record and therefore the system will not calculate a category.

To be able to do an automatic categorizing the category files must exist and contain representative data for each category.

By answering "Yes" to the question "Do you want to change all rows", including those that already have a category" all records in table will be categorized regardless of the value in the stamp column.

Generate new category descriptions

To be able to perform an automatic categorizing there must exist a category definition for all defined categories. The category definitions are saved in a file; one for each category. Each file should contain relevant information on the current category.

The category files could be created by the system if the database contains records by activating the "Generate new category descriptions". In this case the information in the database will be used to create the category files. To make the category files as relevant as possible the following algorithm is used to select proper terms for each category from the database:

1. Ignore terms that are common in all categories (conjunctions, prepositions etc.).
   This is controlled by a slide control in the window "Categorization settings" (see above).
2. Extract terms that are dominant for a category (see "Set category dominance" above).
3. One term could be contained in maximum half of the categories.

This means that a term could be contained in more than one category file (3), but a term that is dominant for a category could only be contained in that category (2).

A step-wise example of the automatic categorization

The following is an example how to run the automatic categorization to get the best result.

1. Determine categories for the current table.

2. Create relevant category definitions for each category.

3. Perform settings
3.1 Vector-reduction to ignore "common" terms.
3.2 Stemming and take care of synonyms.
3.3 Category dominance; terms only valid for one category.
3.4 Allow multiple categories in the stamp column.
3.5 Set tolerance between the best and second best category.

4. Build a part of the database (a couple of thousand records)

5. Analyze the result manually and by help of the system
5.1 Use the sub command dupstamp() to get all records with an ambiguous categorization.
5.2 Check if the calculated category is the best.

6. Make the necessary changes
6.1 Change category in the stamp column (manually online-update the record in database).
6.2 Make changes in the category definitions.

7. Make system generated category definitions
7.1 Use Boolware to find relevant records
7.1.1 Similarity searches combined with Relevance feedback could be very helpful when creating category descriptions
7.2 Run Generate new category descriptions

8. Re-categorize
8.1 Mark both operator and system generated category definitions.
8.2 Perform individual step "Categorize documents".

9. Analyze the result
9.1 Start over from 5.

To goal is to achieve the very best category definitions using only a small part of the entire database. The generated category definitions should then be used when loading the complete database.

New category definitions need only be generated when new categories are added to the table or when old categories are removed.

Another reason to change the category definitions is if a lot of new terms and expressions are added to a category; the language is developed.

Short about Views

A view is a virtual Table. Virtual means that no data is stored permanently in the database for a view; it will be created when requested. A view is in fact a SQL Select statement that could combine data from one or more tables.

Restrictions

When online-update a view in Boolware you should not use triggers. The online-update of a view must be "manual" where you use a "Queue" table.

Performance

Boolware needs fast access to individual rows within the view. In regular tables this is supported by using its primary key, but views lack primary keys which in turn may cause poor performance.

Therefore, please design your view in such a way that it uses an effective combination of indices for its underlying tables. Otherwise it may be very slow to fetch single rows.

If performance is slow in retrieving single rows from the view, we recommend you to test the corresponding SQL against the RDBMS, and inspect its "access plan" (how the RDBMS accesses single tuples). Look for "table scan" and attempt to eliminate these by adding one or more indices to the underlying table.

Important: Triggers could not be used when online-update views.

Chapter 6
Building a Boolware database

This chapter describes how to build a Boolware Index. An index is created for all columns within all tables marked for indexing.

Building Boolware index

If the newly registered data source is not empty, the Boolware indexes could now be built by clicking the "Build" toolbar button in Boolware Manager.

Large databases – many million rows – may take a while to load, but most databases will be indexed in a short time. As always, the build time is largely depending on several factors:

- The power of the CPU, speed of disks and amount of RAM.
- The indexing methods selected. More options selected take more time.

If the database exists in Boolware (there are indexes) the query sessions continues normally in the existing database during building.

If the Boolware index build executed without errors, the new indexes will automatically replace the old indexes.

If any errors were found, the existing (old) version of the indexes is used and you will be notified by an error message that the build failed. All errors and warnings during a build will be logged in the database log.

There is a possibility to remove the Boolware indexes automatically - to save space on disc - before the build takes place. In this case it is not possible to query the table indexes and if any error occurs there is no old index left in Boolware.

Important: When categorizing is a part of the building of the database, online-updates in the data source will not be taken care of. During the categorizing the mechanism that handles the synchronization of the Boolware Index (triggers) is closed. When the categorizing is finished (the last step in the Build) Boolware automatically enables the mechanism that handles the synchronization (creates new triggers).

Online-updates

After the Boolware index database has been built you should not have to do it again, since Boolware supports online synchronization with the data source and online-updates of its indexes.

This means that any time something changes in the data source, these changes are automatically reflected in the Boolware indexes. You as an administrator do not have to do anything for this to happen.

For more detailed information see Chapter 10 "Configuration and online-updates".

Chapter 7
Maintaining a Boolware Index

This chapter describes how to maintain a Boolware Index to get the best performance.

Database validation

In day-to-day operation, indexes is sometimes subject to events that pose minor problems to index structures. These events include:

- Abnormal termination of an application.
- Write errors in the operating system or hardware. These usually create a problem with Index integrity. Write errors can result in "broken" or "lost" data structures.

Boolware indexes can be verified, concerning the syntax and semantics, by running a special step; Validate, which checks all the index files for a table:

You should validate index:

- Whenever an application receives messages that the database is corrupt or wrong search results.
- Periodically, to monitor for corrupt data structures or misallocated space.
- Any time you suspect data corruption.

It is also recommended to validate a database after a database index build.

The Validate function is accessible from the Boolware Manager. After running Validate, use the database log file for information about the validation. If errors were found, only the first 100 errors will be logged in the database log file.

After a validation you could also get warnings. A warning means that the index is still OK, but it does not exactly reflects the indexing properties set when editing the indexing properties. This could happen if there has been a machine failure between the setting of the indexing properties and the re-build of the database/table/column. By re-building the indexes that have got a warning all indexes will reflect the database.

During a validation the status of the database/table is set to "read-only", which means that all search sessions could operate in a normal way but no online-updates can be run.

Database reorganization

Online-updates to the data source take effect immediately in the Boolware indexes, which is nice. But this comes at a price; the index files lose some of their initial efficiency as they gradually become fragmented by many online-updates. Fragmentation means that bits and pieces of the indexes gets moved around in the files, leaving them in a state of less than optimum efficiency.

To fight index file fragmentation, use the index reorganizer function. It is easy to use, just point at the database and select Reorganize from the right-click menu in the Boolware Manager.

The reorganizer has two levels of optimization:

- Defragmentation, reclaims wasted space and reorganizes indexes.
- Layout reorganization, arranges the pages of the index for optimum search access.

Defragmentation is the fastest method and is only used to reclaim wasted disk space.

Layout reorganization is more time-consuming as it rewrites the complete index for optimum search performance.

During reorganization the status of the database/table is set to "read-only", which means that all search sessions could operate in a normal way but no online-updates will be run.

These functions could and should be automated; see part Batch operations and automation in Chapter 9.

Chapter 8
Removing a Boolware index

This chapter describes how to unregister a Boolware Index for a database.

Unregister Boolware Index

In Boolware Manager, select the Boolware database that should be unregistered, and press the minus button "-" in the tool bar.

Note! The current data source must be started and available for Boolware when unregistering a Boolware Index.

Follow the instructions given in the Boolware Manager.

To get a detailed information consult the Boolware Manager Help function (Remove database).

Important: nothing is deleted from the data source (except the Boolware triggers). You only remove the Boolware indexes.

Chapter 9
Maintenance of the Boolware system

This chapter describes how to maintain the Boolware system. It includes topics as: files and file systems, start and stop of Boolware, configuration of the system, temporary files, connections to the Client, managing of log files, automation and scheduling, monitoring, loading a database stepwise and trouble shooting.

Index files

Boolware Index files are stored in one separate directory per table. The files are the following:

.**dsc** file,	description Table file, one for each RDBMS Table. This file contains the chosen indexing settings for all columns within each table in question.
.**tab** file,	one for each Table in the data source. The file is the interface between Boolware and the data source.
.**idx** file,	one for each indexed column. The file contains unique search terms.
.**ref** file,	one for each indexed column. The file contains references to records in the data source for each search term.
.**pxw** file,	one for each proximity indexed column. The file contains word positions for each search term for all records in the data source. (Optional)
.**rnk** file,	one for each indexed column that could be ranked. The file contains ranking information on each search term for all records in the data source. (Optional)
.**rnk2** file,	one for each indexed column that could be ranked. The file contains ranking information on all records in the data source. (Optional)
.**vsm** file,	one for each similarity indexed table. The file contains similarity information on all records in the data source. (Optional)
.**data** file,	one for each indexed table. The file contains data from columns that should be used for fast presentation in a "hit list" and sorting.

File naming conventions

Index files are named using Column names from the RDBMS. However, some characters are restricted in file names. These restricted characters will be replaced by an underscore (_).

The DSC, TAB, DATA and VSM files use the same name as the data source Table name in question.

The DSC and TAB are mandatory while VSM only occurs when similarity is used. The DATA file holds information from some or all columns to be used for fast presentation of column data.

The IDX, REF, PXW and RNK files use names of the Column in question. These files contain the index for each indexed column.

Two of the files are mandatory IDX and REF. The other files PXW and RNK are used when index method near and ranking is used for the current column.

The primary key is saved in a special index $PK which is built up by the mandatory IDX and REF files.

If Free text is selected, the files $FREETEXT.idx and $FREETEXT.ref are also created.

If Similarity is selected, the files $VSM.idx and $VSM.ref are also created.

If Categorization is selected, the files $CATEGORY.idx and $CATEGORY.ref are also created.

Run Boolware Index Server as a service

At installation of Boolware a service will be created and registered.

Starting and stopping the service

If Boolware runs on a Windows server, the service can be started and stopped using the Service Manager in Windows.
When Boolware starts, it determines which IP address this instance of Boolware should have. Boolware tries to wait for the configured network card which is not properly initialized but if it is still not initialized Boolware can be assigned the IP address 127.0.0.1. If this instance of Boolware is a member of a Boolware cluster, an error will be logged otherwise a warning will be logged. In such cases, the service setting can be changed to "Automatic (delayed start)" so that the network card has time to initialize and provide the correct IP address. See also System Requirements; Multiple network cards.

When running Boolware on a Linux server and you are authorized, you start and stop Boolware by using the shell-script created during the installation:

For example:
/etc/init.d/boolware start
/etc/init.d/boolware stop

You can not use the Boolware Manager to start/stop Boolware.

Monitoring client connections with Boolware Manager

In the tab "Sessions" in the Boolware Manager, you can see all connected end users.

The following information is available: Name of the user, the data source used, network address, what type of connection used, the time the session has been inactive since the last activity, the time the session currently is active and the command the session currently is running.

Users can be logged out by marking them and then use the right-click menu alternative "Logout".

Server configuration

To configure the Boolware system, select Options / Configuration… from the main menu of Boolware Manager.

Follow the instructions given in the Boolware Manager.

To get a detailed information consult the Boolware Manager Help function (Configure server).

Monitoring using Server Manager

To monitor server performance, select the Performance tab in Boolware Manager.

The most common and the most time-consuming activities in the system the time is measured continuously. By using the function Performance there is a chance to locate possible bottle-necks. The function is also usable when measuring the capacity in Boolware with the existing hardware.

The function is designed to measure the activities in the system within a certain time interval. By activating the "Clear" alternative all the variables will be cleared and the measuring restarts. The variables could be updated in two ways: implicitly by "Auto fresh" button or explicitly by activating the "Auto refresh" alternative which update the values every second.

A detaild description of the different values will be found in the Boolware Manager Help (Performance data).

Server log

By choosing "View / Server log" you could examine what has happened in Boolware.

You could choose which messages should appear in this log by selecting a Logging level in Options / Configuration..."

Configuration for this file is described in Chapter *10 Configuration and online-updates*.

Indexing history

By choosing "View / Indexing history" you could see history of: Load, Validate, Reorganize, Reindex etc.

The following information is logged: Date, Data source, Action and a Message indicating the time each action has taken.

Configuration for this file is described in *Chapter 10 Configuration and online-updates*.

Server extensions…

By selecting the menu alternative "View / Server extensions…" you could see what program extensions have been loaded by Boolware. Typical extensions are: adapters for different data sources, indexing plugins etc.

Batch operations and automation

Boolware scheduler

Use the Boolware scheduler to automate the operation of Boolware

A lot of Boolware tasks could be scheduled to simplify and secure the administration of Boolware.

In the Boolware Manager Help (Boolware scheduler) you will find a description on all tasks that could be scheduled.

Boolware Manager Command line interface (bwc)

A special program, bwc, supports command line parameters. These parameters can be used to submit jobs to Boolware server.

A detailed description of the different tasks and their parameters will be found in the Boolware Manager Help (Boolware scheduler).

Stepwise load of Boolware index

In some cases it could be convenient to build the Boolware Index step by step. This means that every step that is included in the load of an index is run separately.

The following steps are part of loading an Index: Extract keywords, Sort keywords, Build index and Calculate document vectors.

Some other tasks could also be started from this menu: Replicate, Run duplicate rules, Relate Execution Plan, Categorize documents and Generate new category descriptions.

These functions are found in Manager start window (database list) by marking a database and use the right-click menu "Individual steps").

Extract keywords

All records are fetched from the RDBMS and all words are extracted. Which tables and columns to fetch data from are fetched from the configuration done for the database and its tables.

The indexing methods chosen for each column determines which index terms should be created from each extracted word. Note that a column could have more than one indexing method indicating that more than one index term is created from each extracted word.

Index terms for each column are saved in separate files.

Sort keywords

The created index terms will be sorted in chosen sort order. Note you could manipulate the sort order; see *Chapter 10 Configuration and online-updates*.

Build index

The sorted index terms are read and are used to build the Boolware index files. Depending on the indexing methods a different number of files will be created.

The files IDX and REF must always exist (see above in this chapter File naming conventions).

If the indexing method near is chosen the PXW file will be created as well.

If the indexing method *ranking* is chosen the RNK file will be created. If both these indexing methods, *near* and *ranking*, has been chosen the only file that needs to be created is PXW.

Calculate document vectors

Only if any column within the table is marked for similarity this step will be run. The information in all columns marked for similarity in all records is extracted from the data source. The extracted terms will be used to create a similarity vector for each record which is saved in the file *tableName*.vsm.

Replicate

If you are running Boolware in a Mirroring cluster you could - from the Master node - at any time replicate the Boolware indexes for the current Database or Table to all search nodes in the current Mirroring cluster.

Run duplicate rules

Create files using the specified duplicate rules for this table. The created files are used when searching to eliminate duplicates. The subcommand *dupeliminate* or *dupeliminaterandom* should be used.

Relate Execution Plan

If there are one or more established relations in a database that are marked for optimized relation search (join) you could create them here. If a database has been chosen all marked relations in that database will be re-built. If a table has been chosen all marked relations in that table will be re-built. The files that will be built, the optimized relate files, will be re-created when the database/table is built. The optimized relate files will be updated during an online-update and will be optimized when the database/table is optimized.

Note: To be able to create an optimized relate file the involved columns must have been indexed.

Categorize documents

Categorize the records in the current table according to the current settings. Normally records will be categorized when loading the table but could be done without reloading the table. See Chapter 5. "Editing indexing properties" section "Automatic categorizing" to get a better understanding of this feature.

Generate new category descriptions

If changes has been made you could - at any time - create new category descriptions using this feature. See Chapter 5. "Editing indexing properties" section "Automatic categorizing" to get a better understanding of this feature.

Boolware Administrator alerts

Problems may occur with Boolware – as with any software – during normal operation. For example, the computer that hosts Boolware may run out of disk space. Or the account that runs the Boolware service may have insufficient access rights.

No matter what the causes might be, it is important that the Boolware administrator gets notified of such problems immediately, so that problems can be corrected as soon as possible. This is especially important in such conditions where application availability is critical.

With this in mind, Boolware has been given the ability to alert one or more persons of operational problems. The Alert mechanism is implemented as an operating system command that should be supplied by you. This command will be executed by Boolware when it detects errors. The Alert command is set-up in Boolware Manager, menu item "Options / Configuration".

This means that you have full freedom and control over who gets notified (and how), since you supply the full alert command yourself.

Boolware executes the alert command with the error message text appended. For example, assuming an alert of "net send william" and the error "Out of disk space" Boolware will execute the command:

```
net send william "Boolware: Out of disk space"
```

If you need more complicated broadcasts, you can put multiple commands in a .bat file and have Boolware execute the .bat file instead. Of course, you can also write a custom executable yourself and have Boolware execute that program instead.

Trouble-shooting Network connections

This section describes some troubleshooting guidelines for issues related to network configuration and client/server connections. If you are having trouble connecting the client to the server over a network, use the steps listed below to diagnose the cause.

Connection refused errors

If the client fails to reach the server host at all, or the Boolware server fails to answer, you will get a "connection refused" error. Below is a checklist that you can use to diagnose the source of this error.

Is there a low-level network access between the client and Boolware?

You can quickly test whether the client cannot reach Boolware because of a physically disconnected network or improper network configuration, by using the ping command. Usage is:

```
ping servername
```

Error messages from ping indicate that there is a network problem. Check that the network is plugged in, that the network wires are not damaged, and that the client and server software is properly configured.

Test connectivity from the client to the server; if it succeeds, this may rule out improper network configuration on the client.

Can the client resolve the server's hostname?

Boolware clients can specify the server both by name or IP address. If using names, the client must be able to resolve the server's host name. For TCP/IP, this is done either by maintaining a *hosts* file on the client, with mappings of host names to IP addresses, or by the client querying a DNS server to translate the name to an IP address. Make sure the client has a correct entry for the server host in question.

Is Boolware behind a firewall?

If the Boolware server is behind a firewall, all network traffic may be restricted and the client may not be able to reach the server at all. A firewall permits or restricts access to a server based on the port to which the client attempts to connect. If the client is separated from the server by a firewall, the client cannot connect.

Is Boolware listening on the port?

If the SoftboolSrv.exe service process on the server has not started, there is no answer to attempts to connect to Boolware on port 7008. Start the service. Check the given port number in Boolware Manager menu item Options / Configuration…

Connection rejected errors

Does Boolware have permissions on the database files?

The SoftboolSrv process must have permissions to read and write the database files at the operating system level. Check the permissions on the database files, and the user id of the SoftboolSrv process.

Does Boolware have permissions to create files in the Boolware Index directory?

The SoftboolSrv process must have write permission in the SoftboolSrv home directory (by default C:\Program\Softbool\Boolware on Windows). The server must be able to write to, and create the server log files and other temporary files.

Software support report

To automate and ease support cases, there is a menu option in Manager – Help / Create support report… This command compiles needed information such as versions, log files etc. which are the primary pieces of information that Softbool needs for support.

A dialog is shown, where the Boolware operator can choose which info to send:

- Server settings
- Server log files
- Database logs
- Addressed database tables
- Indexing settings
- Other settings, such as synonyms, stopwords etc.

Boolware compiles the chosen information into a zipped archive and sends it back to the Boolware Manager, so that the operator can, for example, attach it to an e-mail.

Chapter 10
Configuration and online-updates

This chapter describe how to configure and adjust the system to achieve good performance. Subjects handled are: lists and tables, synonyms and thesauri, stemming, stop words, temporary files, log files, languages, synchronizing (online-update) of Boolware Indexes, performance and operations, data sources and database adapters.

Lists and Tables

To tune your Index options further depending on language and contents, Boolware offers a number of tables and settings, where you could affect the way search terms are created and in what order they appear in the index list. The tables are: *Neutralize*, *Phonetic*, *Sort order* and *Word forming*.

The editing of the tables takes place one character at a time where you describe how the current character should be interpreted. In this document we call it to "map" one character to another. To make it easier to describe how to edit the above tables ISO/IEC 8859-1 is used. Each ISO/IEC 8859-1 character is defined by a value by 0 - 255 e.g. the letter A is represented by 65, the digit 0 by 48 and the special character ; by 58. Later versions than 2.4 of Boolware handle UTF8 which is described in Chapter 12 "UNICODE".

Neutralize
The *Neutralize* table is used to tell Boolware how to store it in the Boolware Index. By using this table it is possible to use lower case and upper case when searching regardless in what case the original text was written. You could also make Boolware treat letters with accents the same way as letters without accents. The default settings in this table are: lower case letters are interpreted as upper case letters. This means that Boolware is case insensitive. E.g. FIND car, FIND CAR, FIND Car and FIND CaR will give the same result as the search term in all cases will be CAR. By editing this table you could make Boolware case sensitive.

Phonetic
The *Phonetic table* is used in the same way, but neutralize more depending on the sound of the letters.

Sort order
To get the index terms presented in a desired order you could manipulate the Sort order table. In pure text columns it could be convenient - when presenting index words - to get all terms starting with letters first and terms starting with digits or special characters at the end of the index. To achieve this you could map digits and special characters to higher values than letters. The default settings in Boolware are for example: 0 (48) is mapped to 236, 1 (49) to 237 etc. Special characters are also mapped: & (38) to 254, / (47) to 247 etc. Changing of the sort order will by no means affect the searching; it is just how the index words will be presented.

Word forming
In the *Word forming* table you determine what characters should be regarded as word forming characters. Each character belongs to one of the following "categories": 1) Letter, 2) Digit, 3) Keep between digits else break, 4) Keep between digits or letters else break, 5) Break character, 6) Ignore, 7) Keep between digits else ignore 8) Keep between digits and letters else ignore and 9) Skip between digits else break.

1 and 2 means that the character is part of a search term. 5 means that the character ends a search term (typical characters are blank and tab). 6 means that the character should be ignored (possible character is hyphen (-)). All others 3, 4 , 7, 8 and 9 are conditional characters which should be part of a search term, be ignored or break the current search term depending on if it stands between digits and/or letters; typical characters are: dot (.), comma (,), slash (/)

etc. By manipulating this table you could determine what characters the search terms should contain.

All these tables have a default setting that is the most common so probably there is no need to change them.

After manipulating these tables it is convenient to use the built-in test indexing function in Boolware to see if the changes have the expected result. See Chapter 5 section "Editing indexing properties for a table/view".

The scope the delivered tables is the entire Boolware system. Via the Boolware Manger you could manipulate them on the following levels: database, table and column.

Important: In the Boolware query language some characters - that could be part of a search term - has a special meaning. Below you could see the characters that should be used with care when determine which characters should be part of a search.

Blank character is a very important character and is separator between: commands, terms, sub commands and operators and should therefore be handle with care.

*	is used to replace 0-126 bytes with any characters (truncate)
?	is used to replace 1 character with any character (truncate)
!	is used to replace 1 character with any letter (truncate)
#	is used to replace 1 character with any digit (truncate)
.	at end of a term to ignore truncation when automatic truncation is enabled (exact search)
:	is used to separate column name from search term
"	is used to mark string-search
#	is used to identify arguments in numeric similarity search
$	is used to identify arguments in numeric similarity search
()	is used to start a query enclosed within parentheses
..	is used to mark a closed interval
<[=]	is used to mark an open interval
>[=]	is used to mark an open interval

Important: If any of the tables has been manipulated concerned Boolware Indexes have to be indexed to get the proper search result.

All registered data sources are saved in the sysfile.ini file, should reside in the same directory as Boolware.

The SoftboolSrv.ini file holds information on the configuration of the Boolware system. This file must reside in the same directory as Boolware.

Important: If any of the two configuration files: sysfile.ini and SoftboolSrv.ini are edited outside Boolware Manager it is very important that they are saved with the ISO/IEC 8859-1 character encoding.

Synonyms and Thesaurus

When using the Boolean search in Boolware you search for terms, phrases and combinations of these. Sometimes it is convenient to specify a term and search for that term and all terms which have the same meaning; synonyms. One way is of course to specify all synonyms as terms with the operator Or between them, but this could be hard as you do not always know what synonyms that have been used in the original text.

A better way is to specify a synonym file where all synonyms are contained.

Two files are used to specify terms that belong to each other: the synonyms.xx and the thesaurus.xx. The xx should be replaced by a two character language code; e.g. en for English, sv for Swedish etc. In the *synonyms*.xx synonyms are stored, while the *thesaurus*.xx contains both synonyms and children.

As mentioned the index is not affected by these files, you only specify - by using the query sub-commands syn and thes - that you should use the synonyms.xx and thesaurus.xx respectively.

The design of the synonym- and thesaurus files will be presented in the Boolware Manager when editing them.

The synonym file has a very simple syntax:
main synonym1(synonym1, synonym2, synonym3, ... synonymN)

There is no limitation of the numer of synonyms or main synonyms.

You should be careful not to specify the same synonym for several main synonyms. It is not advisable to have duplicates of main synonyms.

Both the main synonym and the synonyms could contain more than one word:
new york(newyork, new-york)
weekend(week end, week-end)

When searching you must specify synonyms that contain more than one word within quotation marks: FIND text:syn("new york").

Important: These files could be manipulated: insert new terms, delete old terms and change existing terms without being forced to re-index the database. All new sessions will take advantage of the changes made in these files.

Stemming

Another way to let Boolware handle what terms to search for is to use the stemming function. This function is a bit different from the synonym and thesaurus functions in that way it affects the index.

The stemming just indexes the stem of a word which makes the words: walk, walked, walker and walking being indexed as the very same term: walk.

Most words follows stemming rules and thus will be "stemmed" automatically, but of course there are irregularities (e.g. be, is, are, am, was, were, been) which will be taken care of by the file *stemming*.xx.

The xx should be replaced by a two character language code; e.g. en for English, sv for Swedish etc.

Important: Build of Boolware Index must be done after any change in this file

Stop words

The *noise*.xx file contains all words that should not be a part of the Boolware index. Frequently used words that distinguish a document from another could be specified in this file. Typical "noise words" are: prepositions, conjunctions, numerals etc. However, not using any stop words at all (default) does not slow down searches and does not affect size of index files significantly.

A drawback is that a word specified in the noise.xx file could never be searched for.

The xx should be replaced by a two character language code; e.g. en for English, sv for Swedish etc.

Important: Build of Boolware Index must be done after any change in this file.

Temporary file management

Boolware creates a number of temporary files; sort files, keyword files, session history files, log files etc.

The temporary files have different scope, different sizes etc. Some of the temporary files, the sort files, can be very large, so please assign the TEMP environment variable a directory on a disk with sufficient space.

In Boolware Manager "Options / Configuration..." you could specify the proper disk and directory for these temporary files. If nothing is specified Boolware will store the files in the directory that has been specified in the system TEMP/TMP environment variable.

Log files

In the file "boolwarejobhistory.log", which reside in the same directory as the Boolware Index Server, is written message history relating to all databases in the system: loading, optimization, validation etc. With the help of this file, you can follow what happened to all databases.

Boolware server logs diagnostic messages in the file yyyymmdd.log in the log directory. Any messages generated by Boolware are sent to this file. This can be a very important source of diagnostic information if your server is having configuration problems.

The log file is in text format, containing one log record per line. One log file is created per day. The current date is used to build the file name; so for example "20020120.log" is the log file produced on the 20:th of January 2002.

By default this file will be created in the same directory as SoftboolSrv.exe, but you can direct the log files somewhere else by using Boolware Manager.

You can also control the level of detail in what is written to the log file. By default, Boolware will log exceptions, errors, warnings and informational messages. Using Boolware Manager you can change this. Perhaps you are just interested in exceptions and errors.

Language

Boolware supports two languages: English and Swedish. To change language, select Options / Language from the main menu of Boolware Manager. The language only affects the look of the Manager: all help texts, menus, buttons, texts in windows etc will be shown in the chosen language.

There is another setting of language that concerns the contents of the tables. The tables that could be manipulated are: Stop words, Synonyms, Thesaurus, Stemming, Phonetic and Character sets. This is described in *Chapter 5 Editing indexing properties*.

Synchronizing/online-update Boolware indexes

Boolware uses the RDBMS capability of triggers to synchronize the Boolware indexes. Every RDBMS that have support for triggers on INSERT, UPDATE and DELETE can use the Boolware automated online-update function. You can also synchronize Boolware indexes without using triggers, read more about this in section "Manual (custom) synchronization/online-update of index" below.

Important: When categorizing a database - Categorize -, online-updates in the data source will not be taken care of. During the categorizing the mechanism that handles the synchronization of the Boolware Index (triggers) is closed. This means that all pending online-updates must be run before the categorizing starts. When the categorizing is finished Boolware automatically enables the mechanism that handles the synchronization (creates new triggers).

Different RDBMS (data sources)

Most RDBMS support triggers in slightly different ways, if they do at all. So the triggers may look different in different RDBMSs but the purpose is the same; to notify Boolware about changes in the tables.

UDF (User Defined Function)

In order for a full synchronization to be possible, Boolware must receive a signal that a change has taken place and get the old and new version of the changed information.
By an UDF, which has to be installed/registered in the RDBMS, can Boolware be notified that changes has been done in the data source.

Important: The data source communicates with Boolware via the UDF on socket port 8008.

This UDF hold the following parameters:

```
<srv><table><reserved><type>[<mask>]
```

where:

<*srv*>	name of the server where the Boolware server resides
<*table*>	the complete table name
<*reserved*>	reserved for future use
<*type*>	type of action; "I" for insert, "U" for update and "D" for delete
<*mask*>	bit mask for affected columns within the table during an online-update

Install an UDF in RDBMS

Use the installation program setupudf.exe (windows) or installudf (Linux) to install a Boolware UDF in RDBMS decribed below.

The installation programs will display a list of data sources that Boolware supports. In the list you can select/give the data source (s) you want to install a UDF for. The installation program require access to the datasource clients to be able to register the UDF-function within the datasource and copy the udf library to proper locations, make sure the dynamic libraries can resolve selected client libraries via PATH and LD_LIBRARY_PATH environment variables.

DB2

Boolware uses a UDF (User Defind Function) to support full synchronization in DB2. It is called 'boolware_udf' and resides in the Java class file 'libudf_db2.jar' that will be copied to the directory $BD2_HOME/function

NOTE! Make sure that this file is stored in the Java CLASSPATH.

MySQL/MySQL8n
In the Boolware triggers there are calls to an external function, 'XPBOOLWARE', which is stored in the library *udf_mysql.dll / libudf_mysql.so*. To be able to register an external function in MySQL the instance of MySQL that is executed must be dynamically linked. See the MySQL documentation for further information. The Udf-library will be copied to directory $MYSQL/lib/plugin.

Windows:
The environment variable "path" must contain the directory where udf_mysql.dll resides.

Linux:
The environment variable LD_LIBRARY_PATH must contain the directory where libudf_mysql.so resides (usually: $MYSQL/lib/plugin).

Manual registration of UDF:
Copy the udf_mysql.dll or libudf_mysql.so to proper directory, see MySQL documentation.
Run the following command in mysql:

Windows:
```
CREATE FUNCTION xpboolware RETURNS INTEGER SONAME 'udf_mysql.dll';
```

Linux:
```
CREATE FUNCTION xpboolware RETURNS INTEGER SONAME 'libudf_mysql.so';
```

Before version 5.1.2 of MySQL, the library udf_*mysql.dll/libudf_mysql.so* only needed to be installed in a directory which could be found by MySQL via the environment variable PATH so this could be read and loaded dynamically.
From MySQL version 5.1.2 and later, MySQL server uses a specific directory to store the library is specified in MySQL server, see the global variable "plugin_dir" in MySQL reference manual.

Locate the directory with the following command in MySQL:
SHOW VARIABLES LIKE 'plugin_dir';

Oracle
Oracle requires that the "User Defined Function" should reside in the home directory where Oracle is installed $ORACLE_HOME/bin directory. The install program will copy the udf-library to this directory.

At uninstall the 'xpboolero' will automatically be uninstalled.

Manual registration of UDF:
First copy udf_oracle.dll or libudf_oracle.so to ORACLE_HOME/bin directory.

Log in with sufficient authority to requested schema.
(The file name must be a complete file name as in the example below.)

```
create or replace library softbool as
'C:\opt\Oracle\product\11.2.0\db_1\BIN\udf_oracle.dll';
/
create or replace function xpboolero
(srv in string, tbl in string, extra in string, code in string, mask in
string)
return binary_integer
as
```

```
external
language C
library softbool
name "xpboolero"
parameters (srv string, tbl string, extra string, code string, mask string);
/
grant execute on softbool to public;
/
grant execute on xpboolero to public;
/
```

Manual unregistration of UDF:
Log in with sufficient authority to requested schema.
Run the following commands:

```
drop function xpboolero
/
drop library softbool
/
```

Remote Procedure Calls (RPC)

Since Boolware updating mechanism relies on external function callouts, the Oracle instance need to have this configured in order to work.

The following example show how to configure the LISTENER.ORA and TNSNAMES.ORA files to use inter-process communication (IPC) to invoke external stored functions.

The files are located in the folder ORACLE_HOME/network/admin.
See Oracle Administrator's Guide for additional information on how to configure the LISTENER.ORA and TNSNAMES.ORA files for external functions.

```
#LISTENER.ORA Configuration File
LISTENER =
 (ADDRESS_LIST =
  (ADDRESS=
   (PROTOCOL= IPC)(KEY= EXTPROC0)
   )  )
SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC = (SID_NAME = extproc)
      (PROGRAM=extproc)))


#TNSNAMES.ORA Configuration File
extproc_connection_data.world =
 (DESCRIPTION =
  (ADDRESS =
   (PROTOCOL=IPC)(KEY=EXTPROC0))
    (CONNECT_DATA = (SID=extproc)
    ))
```

PostgreSQL
The name of this registered function is 'xpbo' and is located in the udf-library.
PostgreSQL requires that the "UDF" should be stored in the directory defined by PostgreSQL and the install program will copy the udf-library to the directory defined as $pghome/lib
It is very important to select correct version of library when register udf library in PostgreSQL.

At uninstall the 'xpbo' will automatically be uninstalled.

Manual registration of UDF:
Copy the udf_postgres.dll or libudf_postgres.so to $libdir directory.
Log on to PostgreSQL with their client and run the following commands:
(The file name must be with full path as in the example below.)

```
CREATE OR REPLACE FUNCTION xpbo(bytea, bytea) RETURNS int4 AS 'C:/Program
Files/PostgreSQL/9.0/lib/udf_postgresql.dll', 'xpbo' LANGUAGE C VOLATILE
STRICT;

GRANT EXECUTE ON FUNCTION xpbo(bytea, bytea) TO public;
```

Manual unregistration of UDF:
Log on to PostgreSQL with their client and run the following commands:

```
DROP FUNCTION IF EXISTS xpbo(bytea, bytea) CASCADE;
```

SQL Server
A "User Defined Function" for SQL Server to synchronize online-updates should be registered in the "master" database and have the name 'xp_boolero' and it exists in the dynamic link library udf_sql server.dll. The library can be located anywhere and the setup program will copy the library to the directory: "C:\BoolwareUDF\SQL Server"

The 'xp_boolero' will automatically be unregistered at uninstall.

How Boolware triggers operates

As mentioned above, triggers are used to keep the Boolware Indexes synchronized with the data source.

To be able to take care of changes from the data source a queue table is created by Boolware at the same time when creating the triggers. The name of the queue table is: <tablename>_Q. To handle the order of changes in the data source correctly, Boolware also creates sequence counters together with associated functions in the data source.

Trigger events

Whenever the data source is modified – as a result of either an INSERT, UPDATE or a DELETE – the trigger for this table will be activated. This trigger contains script code that the Boolware generates when instruct Boolware to index a specific table and syncronize (online-update) changes in the datasource.

If online-update type "Fully Automatic" is selected and a "User Defined Function" is installed, the "Trigger" ends with a call to this "User Defined Function" which notifies Boolware that a change has taken place in the current table. This online-update type is not suitable when updating large amounts of records – instead, select the online-update type "Semi-automatic" and schedule how often Boolware should check if there are records to take care of in the queue table.

The names of the trigger scripts generated by Boolware depends on the table and the events. Typically the trigger scripts will be named: <tablename>_tr_ins, <tablename>_tr_del and <table name>_tr_upd for the INSERT, DELETE and UPDATE event respectively.

NOTE! When triggers are created for MySQL InnoDB tables and colum data for all columns are not stored in Boolware, an additional table (boolware_sequence) and function (boolware_nextval) are created to handle trigger inserts in the queue table.

If "binary log" is activated in MySQL where the function "boolware_nextval" is created, the variable "*log_bin_trust_function_creators*" needs to be activated in order to create the function.

To activate the variable, enter the following in MySQL console:
"*SET GLOBAL log_bin_trust_function_creators = 1;*"
and add the following in mysql.ini:

log_bin_trust_function_creators = 1;

MySQL require that the collation should be the same on the database and the table were the trigger should be created.

Manual (custom) synchronization/online-update of index

Via a separate user application, you can handle direct updating of indexes and data in Boolware, which is applicable as the data source either does not support "triggers" or you do not want or can use triggers (such as in a view).

What is required for Boolware to be able to synchronize indexes with the data source without triggers is:

1. That all columns in the current table in question are also stored in Boolware, ie. the "Store column data" indexing feature is enabled for all columns for the table in Boolware so that Boolware can always retrieve the old wording of data in an instant update. **Tip:** Also turn on - Apply "Store column data" automatically - for new fields in the current table.

2. A signal that tells Boolware that a change (insert, update, delete) has occurred in the data source. To signal to Boolware that a change has occurred in the data source and that update records can be retrieved in the queue table, it is advisable to schedule checks of updates in the queue table via the scheduler in Boolware Manager or use the command line program "bwc" that comes with the installation of Boolware.

3. That a special queue table and sequence counter was created manually in the data source (done via Boolware Manager's option for manual synchronization) with the names: tablename_Q and tablename_sbseq where tablename is the name of the current source table/view in question. It is in the queue table that the custom user application must fill in with new, changed or deleted data.

The Queue table (tablename_Q) contains the same columns as the current table in question and three extra columns: **UPD$OLDPK** char(255), **UPD$TYPE** char(1) and **UPD$SEQ** integer
For **DB2** these are called: **UPD_OLDPK** char(255), **UPD_TYPE** char(1) and **UPD_SEQ** integer

UPD$OLDPK	shall contain the current primary key to be handled in Boolware. If the primary key consists of several fields, the values in the UPD$OLDPK must be joined and separated by the string/separator **\|$|** (i.e. PipeDollarPipe). Example: A table/view has a primary key consisting of three fields such as ID, PNR and DATE in the specified order and then the content of UPD$OLDPK should for example be: 234**\|$|**6203219999**|$\|**20170925 If you want one of several input values, from UPD$OLDPK, to be ignored when updating/deleting, the value should be replaced with an asterisk (*) for example: ***|$\|**6203219999**\|$|**20170925
UPD$TYPE	shall contain the type of update, ie. **I** (insert), **U** (update) or **D** (delete). For **I** (Insert), all columns in the queue table must contain all the data that must be present and that must be indexed and stored in Boolware. For **U** (update), all columns in the queue table must contain all data that must be present and that must be indexed and stored in Boolware For **D** (delete), only the three additional columns in the queue table, ie. **UPD$OLDPK**, **UPD$TYPE** and **UPD$SEQ** contain data because Boolware retrieves the old wording of all other columns from what is stored in Boolware.

UPD$SEQ	shall contain a unique sequence number which is automatically generated by the data source for the column by means of a function specified as value at *insert* in the queue table Example: to insert a '**D**' with a composite primary key of three fields with wild card specified for any of these three fields: **DB2** *insert into "ADM"."PERSON_Q"("UPD_OLDPK", "UPD_TYPE", "UPD_SEQ")* *values ('*|$\|6203219999\|$|*', 'D', NEXT VALUE FOR "ADM". "PERSON_sbseq")* **Oracle** *insert into "ADM"."PERSON_Q"(UPD$OLDPK, UPD$TYPE, UPD$SEQ)* *values ('*\|$|6203219999|$\|*', 'D', "ADM". "PERSON_sbseq" .nextval);* ***PostgreSQL*** *insert into "public"."person_Q"("UPD$OLDPK", "UPD$TYPE", "UPD$SEQ")* *values ('*|$\|6203219999\|$|*', 'D', nextval ('person_sbseq'));* **SQL Server** *insert into "dbo"."PERSON_Q"("UPD$OLDPK", "UPD$TYPE", "UPD$SEQ")* *values ('*\|$|6203219999|$|*', 'D', NEXT VALUE FOR "dbo"."PERSON_sbseq");*

Boolware tables

Boolware record table

Each Boolware table has a "q-file" associated, with the same name as the table, and the extension ".q". This file should uses the same format as the import file, i.e. use the same delimiters between rows and columns (the import file includes data used to load the table indexes at the first time). Each row in the q-file represents either a new row (Insert), a deleted row (Delete) or a changed row (Update).

Each record starts with a mandatory part of two fields - the update type field which is the first letter on each line: **I**, **D** or **U** - and the primary key field. If the primary key is composed of multiple columns, it shall be expressed as a string consisting of each of the primary key columns separated by binary one. For example, if ID and DATE form primary key together, and ID is 77 and DATE is 20080603, such a primary key is coded as:

77**1**20080603.

Note, the bold "**1**" at the third position is binary one, not the ascii character one.

Eample of Insert, Update and Delete of a line where the two first columns together form primary key:

```
I;"6109011234120080608";6109011234;20080608;Title;"New record"
U;"6109011234120080608";6109011234;20080608;Title;"New record, completed"
D;"6109011234120080608"
```

When an online-update on a record table is initiated, Boolware will at first check for an existing .qq-file and if exists Boolware will start the online-update using the .qq-file. When processed successfully or if no .qq-file exists Boolware will rename the .q-file to .qq-file, create a new empty q-file and process the online-update using the .qq-file. If a severe format error in the .qq-file is encountered, the online-update process will stop with an error message in the Boolware system log file and in this case the administrator of Boolware needs to correct the format error in the .qq-file manually and restart the online-update process again.

The program that creates records in the q-file must take into consideration that the q-file momentarily - when Boolware rename the q-file; and creates a new empty q-file; - is not available due to the renaming process of the q-file and makes sure to close the q-file after accessing the file.

Example: assume a table containing news articles. The table contains three fields: a numeric ID (primary key), a header and a text.

```
I;45;45;"Volvo sold by Ford";"Volvo is for sure worth 53 billions..."
```

Note that the primary key occurrs twice. Once in the fixed part (the two first columns), and then in the data row as well. The reason is if you want to change the primary key itself - both the old and new value must be known if so.

```
D;45
```

Deletes are the simplest form of changes. Only the primary key need be supplied. The old content will be read from Boolware's data file and be removed from the index.

```
U;45;46;"Volvo sold by Ford";"Volvo is maybe worth..."
```

This is an example of both the PK and the column value changed. The PK is changed from 45 to 46, and the words "for sure" changed to "maybe" in the text- Boolware will read the old content from its data file, replace it with the new content and make any necessary changes to its index.

Error messages

Always check database- and system log files for eventual error messages and check the q-files syntax if having any problems.

Boolware file system table

For these tables no q-file is needed. When the online-update starts, Boolware will review all the directories listed for the table and new documents will be added to the index, updated files will be changed in the index, and deleted files will be deleted in the index.

Error messages

Always check database- and system log files for eventual error messages.

Boolware web crawler table

For these tables no q-file is needed. When the online-update is started Boolware will go through its Apache Nutch database and new URLs will be added to the index, updated URLs will be changed in the index and deleted URLs will is deleted in the index.

Error messages

Always check database- and system log files for eventual error messages.

Automatic modifications by Boolware during online-update

To avoid unnecessary stops in the online-update and to make the process more efficient Boolware automatically changes the specified update type (**I**, **D**, **N**, **O** or **U**) in some special cases. When any of the below takes place Boolware will give log this in the Server log.

The update types are created a little bit differently. The update types "**I**" and "**D**" are created in the same way. The "**N**" and "**O**" records are created by a trigger connected to the current datasource. "**N**" and "**O**" are replaced by "**U**" if all data is stored in Boolware.

Records are read one at a time and a record with the "first" update type is obtained. If the update type is "**N**" the next record is read and in this, the update type must be "**O**".

Below is a list of modifications that Boolware performs:

Modifications to make online-update more efficient:

- If the update type "**D**" is immediately followed by an update type "**I**" containing the very same primary key, internally Boolware will change the update types "**I**" to "**N**" and "**D**" to "**O**" respectively. It is much more efficient to update an existing record than to first delete the old and then insert the new contents.

Modifications to avoid unnecessary stops in online-update:

- If the update type is different from "**I**", "**D**", "**N**", "**O**" or "**U**" the record will be ignored and removed from the queue table.

- If the update type is "**D**" and the specified primary key does not exist in the Boolware index the record will be ignored and and removed from the queue table.

- If update type is "**I**" and the primary key already exists in the Boolware index, this is treated as an update and the existing record's index is updated with terms from the "**I**" record that not already exist in the index for the existing record.

- If the update type is "**N**" and the specified primary key is illegal for Boolware (e.g. contains a NULL segment) the "**N**" record will be ignored and removed from the queue table and the following "**O**" record will be changed to a "**D**" record.

- If the update type is "**N**" and is followed by "**O**" record and the old primary key is not found in the Boolware index, the "**N**" record will be changed to an "**I**" record and the "**O**" record will be ignored and removed from the queue table.

- If the update type is "**N**" and is **not** follwed by an "**O**" record the "**N**" record will be changed to an "**I**" record.

- If update type is "**U**" and the old primary key is not found in Boolware index the "**U**" record will be changed to an "**I**" record.

- If update type is "**O**" and not preceded by an "**N**" record the "**O**" record will be ignored and removed from the queue table.

Performance

This part contains tips and information on how Boolware Index Server should be configured and maintained to get the best result. Both the RDBMS, Boolware Index and the hardware is part of this description.

Make sure that the computer has enough memory and disk. More memory and large and fast disks will improve performance.

Adding extra CPUs will improve performance. Boolware and many SQL data sources are capable of utilizing more than one CPU if available.

Spread your files over several disks, for example the SQL data source on one disk and the Boolware indexes on a second disk.

Some data sources uses, by default, Clustered primary keys. Although this is probably a good default choice, be aware that it hurts performance when building Boolware indexes. For this reason, Softbool recommends that you - if possible - avoid "Clustered" on primary keys in the data source.

If running in the same computer, the data source and Boolware may contend about memory which may hurt overall performance severely. By default, most data sources allocates very large amount of memory to speed up its processing. This may leave other programs too little to operate at all, or at least well. To remedy this, limit the maximum memory consumption allowed for the data source. Experiment with different settings until you achieve best overall performance.

Consider running the data source and Boolware on different computers to improve performance.

If you need to build several Boolware indexes, it will be faster to build these one after the other instead of building them all at once.

Online-updates of near-indexed (proximity) columns are expensive. The difference in the online-update of columns that have or have not proximity could be as much as a factor of 8.

Important: Online-updates means that all changes in the data source will immediately be reflected in the Boolware Indexes. The communication between the data source and Boolware is managed by "triggers" and "queue tables". To manage the "queue table" automatically via the data source or "manually" via a program requires a lot of resources and affects the online-updating of the data source as well. This means that large changes - in several hundred thousand records - will be **much** more efficient if you: deactivate the "triggers", online-update the data source and then re-load the affected tables and columns. After that you should activate the "triggers" and all online-updates will work as usually.

Test indexing

The "Test indexing" button is warmly recommended to check which terms that are being generated for the current field.

You can supply your own test text. If you do not, Boolware will extract the first 10 tuples with text from up to the first 100 tuples from the selected data source as sample data.

Multiple indexing options in the same column may make it impossible for Boolware to automatically determine which option that an application requests. In this case, the application must inform Boolware what kind of query it desires. This is done using the Boolware Query Language.

If Boolware cannot Index the table, you will get the message "Cannot index table ‘table name’, since it has no primary key". Boolware is restricted to tables with primary keys. Apply a primary key to the table if you want to use Boolware with it.

Data sources

In this section describes the way Boolware communicates with different SQL data sources through adapters; one adapter for each data source. In the next chapter the differences between the different data sources are covered. Browse the Softbool web site (http://www.softbool.com/) for the latest list of supported data sources.

Adapter

A SQL data source is a system that contains the information that is indexed by Boolware. Commonly used and supported SQL data sources are: SQL Server, Oracle, MySQL, Sybase etc.

In Boolware the SQL data source is implemented as an adapter, a plug-in module - one for each data source. Each adapter is a shared program (DLL for Windows and "shared library" for Linux) that handles the current data source. When Boolware is started it will find the different adapters. Each adapter has a standardized name e.g. "plugin_SQL Server.dll" or "plugin_Oracle.dll" for SQL Server and Oracle respectively.

RDBMS mirroring and Boolware

Several RDBMS maximize database availability by "mirroring". The idea is to duplicate the database from an original to a mirror, on another server. The database mirror shall be ready, at any time, to take over traffic in case the first database server fails. This fail-safety minimizes database downtime for business-critical applications by providing a continuously available standby system, increasing availability.

If the main server fails and there is a shift to the mirror, you want the Boolware traffic to be automatically rerouted to the new mirror server. This can be done using a simple SQL call at the time of the RDBMS takeover. Example: (Sybase and SQL Server)

```
exec xp_boolero '192.168.1.105', _
    '', _
    'reroute ip "192.168.1.110" to "192.168.1.112" dbms "Sybase"', _
    'CMD'
```

(underscores are used to break the long line for the sake of readability)

The call to Boolware is done through the same "udf" registered in the RDBMS to signal table changes to Boolware. It's called xp_boolero for MS SQL Server and Sybase, but may have slightly different names under other RDBMS:es.

- P1: IP number for the Boolware server
- P2: Shall be an empty string in this call (two apostrophes)
- P3: The Boolware REROUTE command to reroute traffic for a specific (or all) RDBMS from the failing server to the mirror. This command must contain both the old and the new IP numbers., as well as optionally a RDBMS name. If *dbms* is left out, all RDBMS:es are assumed.
- P4: The text "CMD" to make Boolware regard this is as a command, not a trigger notification.

RDBMS names

SQL Server, DB2, Oracle, Sybase, MySQL, PostgreSQL.

Boolware

This adapter handle the internal record table type, file system table type and crawling table type handling content in a Nutch database

DB2

This adapter implements support for the IBM DB2 UDB via CLI (Call Level Interface).

Boolware supports 64-bits LUW (Linux, UNIX, Windows) of DB2 UDB.

Special requirement at installation

System requirements when running Boolware:
Boolware must have access to the DB2 client library (CLI). If DBALIAS is used then the database must be cataloged so DB2 CLI can connect to DB2.

Windows:
DB2 Extended Windows security should be turned off (default for DB2 CLI) otherwise, user rights for the user running the Boolware server must exist for the directories DB2 is working with.

How Boolware identifies a DB2 instance

To make Boolware aware of one or more DB2 database instances one or more datasource connections must be created and configured and this is done via the Boolware Manager.

Tip

```
SQL1224N A database agent could not be started to service a request, or was
terminated as a result of a database system shutdown or a force command.
```

This is a DB2 configuration matter. It happens if Boolware and DB2 executes in the same computer, and Boolware gets DB2 local connections through CLI. Local connection in DB2 will attempt to use shared memory, and the amount of shared memory is probably not configured large enough.

Solution:
Avoid shared memory connections by re-cataloguing the database to be on a remote TCP/IP loopback node:

```
catalog tcpip node loopback remote 127.0.0.1 server db2_boolware

uncatalog database xxx
catalog database xxx as xxx at node loopback
```

SQL1042C An unexpected system error occurred. SQLSTATE: 58004.

This happens if DB2 isn't properly installed. The IBM setup program fails to setup some important paths in the shared objects path. Check the /home/user/db2dump/db2diag.log file for error messages. If it says that for example "libjava.so Cannot open shared object file" it is a symptom of this.

Solution:
Correct the shared object path either by setting the LD_LIBRARY_PATH environment variable for the current user, or by editing the /etc/ld.so.conf and then execute /sbin/ldconfig.

The paths that usually need to be included are:

```
/opt/IBM/db2/V8.1/lib
/opt/IBMJava2-141/jre/bin
/opt/IBMJava2-141/jre/bin/classic
```

MySQL

This Adapter implements support for MySQL.

MySQL supports data type "unsigned" for certain numeric data types: "tinyint unsigned", "smallint unsigned" and "int unsigned". Internally in Boolware the "unsigned" data type is taken care of by "mapping" the "unsigned" data type to the next higher data type.

Example: "tinyint unsigned", "smallint unsigned" and "int unsigned" will - internally within Boolware - be treated as "smallint", "int" and "bigint" respectively.

To be able to create "triggers" the user that is registered as the Boolware-user on a table must have authority 'SUPER'. This is achieved by the mysql command:

```
GRANT SUPER ON *.* TO 'boolwareuser';
```

where *.* is the database/table that should get 'SUPER' authority and the boolwareuser is the user that is used when Boolware is logging on to MySQL; see the MySQL documentation how to proceed when setting authority from different servers.

How Boolware identifies a MySQL instance

To make Boolware aware of one or more MySQL database instances one or more datasource connections must be created and configured and this is done via the Boolware Manager.

MySQL8n

This Adapter implements support for MySQL 8.0.28 and later.

System requirements when running Boolware:
Boolware must have access to dynamically linked MySQL version 8 and associated dependencies:
Windows: libmysql.dll, libssl-1_1-x64.dll, libcrypto-1_1-x64.dll
Linux: libmysqlclient.so, libssl-1_1.so, libcrypto-1_1.so

Otherwise, the same requirements apply as for ***MySQL*** above

Oracle

This Adapter implements support for Oracle via Oracle client library OCI.

Special requirement at installation

System requirements when running Boolware:
Boolware must have access to the Oracle OCI Client.
NOTE! OCI client must be 10.2 or later.

Linux:
Access to Oracle 64-bits OCI client 10.2 (libcIntsh.so.10.2) and accessible of Boolware (LD_LIBRARY_PATH and authorized to use/run it).

If Oracle OCI 11 is installed you could create a symbolic link to it with the proper name libclntsh.so.10.2

```
ln -s libclntsh.so.11.1 libclntsh.so.10.2
```

The environment variable LD_LIBRARY_PATH must contain (ORACLE_HOME)/lib directory

How Boolware identifies an Oracle instance

To make Boolware aware of one or more Oracle database instances one or more datasource connections must be created and configured and this is done via the Boolware Manager.

Restrictions and limitations for Boolware

Columns of data type LONG will not be indexed by Boolware because these are not allowed in Oracle triggers which means that online-update of Boolware Index cannot be done.

Tips, convert them to appropriate LOB type to get them indexed. See Oracle documentation for detailed information.

Views can be indexed. If all columns are contained in the same table and none of the columns are of the type LOB a trigger that applies on views, (**instead of** trigger) is created. These triggers does not handle the LOB columns in way to be able to automatically online-update the Boolware Index. Synonyms are not supported by Boolware.

PostgreSQL

This adapter implements support for PostgreSQL via the PostgreSQL client library LIBPQ

Special requirement at installation

System requirements when running Boolware:
Boolware Index Server must have access to PostgreSQL Client because Boolware's PostgreSQL adapter requires access to PostgreSQL client library LIBPQ.DLL/LIBPQ.SO and its components.

To give the PostgreSQL client access to the current PostgreSQL installation it must be allowed by the database installation. This should be controlled and handled by PostgreSQL's configuration file pg_hba.conf; see PostgreSQL manual for more information.

Linux:
Availability to the PostgreSQL client (libpq.so) and accessible by Boolware. Set path to the PostgreSQL client library in the Environment variable LD_LIBRARY_PATH.

Windows:
Availability to the PostgreSQL client (libpq.dll) and accessible by Boolware. Set path to the PostgreSQL client library in the Environment variable PATH

How Boolware identifies a PostgreSQL instance

To make Boolware aware of one or more PostgreSQL database instances one or more datasource connections must be created and configured and this is done via the Boolware Manager.

SQL Server

This adapter implements support for SQL Server via ODBC.

Special requirement at installation

System requirements when running Boolware under Windows:
SQL Server, ODBC (Open Database Connectivity).

System requirements when running Boolware under Linux:
To be able to run Boolware's Linux SQL Server adapter, download the Microsoft ODBC-driver for SQL Server Linux x64 and install it.

How Boolware identifies a SQL Server instance

Windows:
To make Boolware aware of one or more SQL Server database instances one or more ODBC-system connections must be created and configured and this is done via the Boolware Manager connected to Boolware server.

Linux:
Register one or more system connections in the file ODBC.ini resided in the directory: /etc

Sybase

This adapter implements support for Sybase through Sybase Open Client shared libraries libct and libcs.

Special installation requirements

System requirements when running Boolware:
Boolware must have access to Sybase Open Client libraries, libct.dll/so and libcs.dll/so and it's components.

Linux:
Environment variable LD_LIBRARY_PATH must contain the (SYBASE/SYBASE_ASE)/lib directory

How Boolware identifies a Sybase instance

To make Boolware aware of one or more Sybase database instances one or more datasource connections must be created and configured and this is done via the Boolware Manager.

Chapter 11
Interactive Query

This chapter describes the search capabilities of the Boolware system. From a user-written Application the Boolware server will be requested to perform certain actions on the Boolware indexes. In this chapter the following actions will be described: view the contents of a Boolware index, search a Boolware index, rank the result from a query in a Boolware index and finally the transfer of requested data from the data source.

Boolware Query Language

To be able to specify a query in a standardized way the "Boolware Query Language" (QL) is used. Below follows some example how to use this query language.

Schematic component overview

Query		**TABLES**(*scope*) **FIND**/**AND**/**OR**/**NOT**/**XOR**/**BACK**/**FORWARD** *sub-query*
	Or:	**RELATE**(*Tablename*) **FIND**/**AND**/**OR**/**NOT**/**XOR**/**BACK**/**FORWARD** *sub-query*
	Or:	**RELATE**(*Goal table*, *result table*)
Scope		**TABLES**(*Tablename1* [...,*TablenameN*]
	Or:	**TABLES** (*)
Sub command		*term* [*operator term*]
	Or:	**SIM**(*text*)[,*threshold*][,*optimize*]
	Or:	**NEAR**(*term term* [*term..term*] [,*distance*[,*order*]])
	Or:	**XNEAR**(*term term* [*term..term*]; [*exTerm exTerm*...] [,*distance*[,*order*]])
	Or:	**FUZZY**(*term* [*term..term*] [,*numchars* [,*limit* [,*usefuzzysyn* [,*withrank*]]]])
	Or:	**QUERYLIM**(term [term..term] [;n])
	Or:	**NUMRANGE**(*N*..M)
	Or:	**SEARCHVIAINDEX**(N[,*P*..*Q*, M])
	Or:	**SEARCHVIAINDEXNOGEN**(N[,*P*..*Q*, M])
	Or:	**STRING**(*string of terms*)
	Or:	"*phrase*"
	Or:	"*word ... word* "
	Or:	**WITHINSTRING**(*string of terms*)
	Or:	**WORD**(*term* [*term.. term*])
	Or:	**ORSEARCH**(*term* [*term.. term*])
	Or:	**ORSEARCH**(@complete file name)
	Or:	**ORSEARCHEX**(*term* [*term.. term*])
	Or:	**ORSEARCHEX**(*@complete file name*)
	Or:	**COMPRESS**(*term* [*term.. term*])
	Or:	**CASE**(*term* [*term.. term*])
	Or:	**WITHIN**(*term* [*term.. term*])
	Or:	**WORDASIS**(*term* [*term.. term*])
	Or:	**STRINGASIS**(*string of terms*)
	Or:	**GEOCIRCLE**(*latitude*; *longitude*; *r=n*)
	Or:	**GEORECTANGLE**(*latitude1*; *longitude1*; *latitude2*; *longitude2*)
	Or:	**GEOPOLYGON**(latColumn; longColumn; POLYGON((long lat,)))
	Or:	**GEOPOLYGON**(latColumn; longColumn; MULTIPOLYGON(((long lat,))))
	Or:	**GEOWITHINPOLYGON**(x, y)
	Or:	**SYN**(*word* [*word*...])

Or: **THES**(*word* [*word*...])
Or: **STEM**(*word* [*word*...])
Or: **SOUND**(*word*)
Or: **NOTI**(*word* [*word*...])
Or: **NOTI**(*@complete file name*)
Or: **PREFIX**("*p11*" OR "*p12*", "*p21*", "*p31*"; *term* [*operator term*])
Or: **DUPSTAMP**()
Or: **DUPELIMINATE**(duplicaterule [, numtosave])
Or: **DUPELIMINATERANDOM**(duplicaterule [, numtosave])
Or: **GETDUPLICATES_MAIN_RECORDS**(duplicaterule)
Or: **GETDUPLICATES_WITHOUT_MAIN_RECORDS**(duplicaterule)
Or: **GETDUPLICATES_ALL_RECORDS**(duplicaterule)
Or: **NOLINKWORDS**(*term* [*term … term*])
Or: **SET**(*id*)
Or: **QUERY**(*id*)
Or: **RESULT**(*id*)
Or: **SCRATCH(*name*)**
Or: **RPATH**(*tab1.col1*, *tab2.co1..tabN.colN*)
Or: **MAXQUERYEXECUTIONTIME**(*N*)
Or: **EXISTS**((search criteria) conditions)
Or: (- sub-query -)

Operator: **AND/OR/NOT/XOR/ANDF/ORF/NOTF/XORF**

Term: COLUMN_NAME: *term*
Or: COLUMN_NAME: *interval*

Interval: *term.. term*
Or: < *term*
Or: <= *term*
Or: > *term*
Or: >= *term*

BACK
FORWARD

A query string is built up by: a Command, Terms, Sub commands and Operators.

The Commands **FIND**/**AND**/**OR**/**NOT**/**XOR**/**BACK**/ **FORWARD** and the terms are case insensitive.

In the following examples the Commands are denoted in capitals to make it more readable.

A colon (:) is used to separate the Column name (field) from the search term; a Column name is defined by the ending colon.

If a Column name contains special characters: blank, colon, dot etc. the Column name must be enclosed within quotation marks.; "Column name":term.

If a search term is not prefixed by a Column name, it will be regarded as free text.

A search term could contain any number of **wild** cards (*****, **?**, **!** and #), where * replaces 0 - 126 bytes of any type, ? replaces one (1) character of any type, ! replaces one (1) character of type letter and # replaces one (1) character of type digit. The wild cards could appear anywhere and any number of times in a search term.

NOTE In a string could only the only allowed wild cards are: * and ?.

Using a setting command you could set automatic truncation which means that all specified Terms will automatically be truncated (see "**Programmer's Guide**"). To search on an exact Term you have to end that Term with a period (.) to ignore the truncation.

Left and right parentheses are part of the QL-synatax and if occur in a term they must be escaped with the sign '\'.

A search string must start with a Command. The following Commands are allowed: **FIND**/**AND**/**OR**/**NOT**/**XOR**/**BACK**/**FORWARD**.

The boolean operators **AND**/**OR**/**NOT**/**XOR** could also appear between the terms as Operators. The words AND, OR, NOT and XOR could also appear as a Term. A special type of Operators, ANDF, ORF, NOTF and XORF exist to simulate a new Command (see Query History and Related search (Join) below). If the term has the same 'name' as an operator (AND, OR, NOT, OR or XOR) the term/operator has to be repeated. If any of the words AND(F), OR(F), NOT(F) and XOR(F) ends with a wildcard character (*, ?, ! or #) or and ending dot (.) it will **not** be regarded as an operator.

Example:

FIND and	in this query string AND will be regarded as a term; an operator could not directly follow an other operator/command.
FIND car **AND** truck	in this query string AND will be regarded as an operator. The only terms that are search for are: car, truck with the operator AND.

To be able to search for all three terms: car, and, truck you must repeat the term/operator:

FIND car AND and truck	in this query string all three terms (car, and, truck) will be searched for with the operator AND.
FIND car and. truck	in this query string all three terms (car, and., truck) will be searched for with the operator AND. The term and. (final dot) is not an operator.

BACK/**FORWARD** are used to browse through a Query History.

A **FIND** Command start a new query session. All other Commands are affecting the earlier result.

The QL contains several sub-commands to refine the queries.

The following sub-commands are allowed: **sim**, **near**, **xnear**, **fuzzy**, **querylim**, **numrange**, **searchviaindex**, **searchviaindexnogen**, **string**, **word**, **orsearch**, **orsearchex**, **compress**, **within**, **case**, **wordasis**, **stringasis**, **withinstring**, **geocircle**, **georectangle, geopolygon**, **geowithinpolygon**, **syn**, **thes**, **stem**, **sound**, **noti**, **prefix**, **dupstamp**, **dupeliminate**, **dupeliminaterandom, nolinkwords, set**, **query**, **result**, **scratch** and **rpath**. All these sub-commands, except **sim, rpath** and **maxqueryexecutiontime**, could appear anywhere within the query string.

Example:
FIND Article:sport Text:**near**(Sven Johansson)
Will find "Sven Johansson" only in sport articles.

Some sub-commands can be nested:

near(**case**(IT company))	Search *IT* and *company* as case sensitive
near(**wordasis**(Ericsson Development))	Search *IT* and *company* exact
near(**sound**(Lars Eriksson))	Search *Lars* and *Eriksson* phonetically
nolinkwords(**sound**(LM Ericsson))	Search phonetic without "linkwords"

sound(**nolinkwords**(LM Ericsson)) Search phonetic without " linkwords "

There are two different ways to query within Boolware: the traditional boolean way and the similarity way (**sim**).

The boolean query combines terms with operators, while the similarity query compares the attached text to all records in the database and rank them in similarity order.

The sub-command sim means that a similarity search rather than a boolean search should be performed. See section "Similarty Search" in chapter 11 for further information.

The sub-command **near**, proximity search, means that the specified search terms should appear within a certain distance in the record and in certain cases in the specified order. There are two parameters that control the distance and the order of the specified terms and they should appear directly after the last term. The first parameter tells the number of "foreign words" minus 1 that could appear between the specified search terms. If the second parameter is a value other than 0 the search terms could be specified in any order. The default values for these parameters are 5 (4 "foreign words" could appear between the search terms) and 0 (the search terms could be specified in any order). A special case is when the specified words should be in the specified order and no other words could appear in between; this is a phrase search and the parameters should be 1, 1. Another way to specify this is to enclose the specified words within quotation marks or specify the words within the **string** sub-command. If the column is marked for string search a *string search* will be performed rather than a *proximity search*.

The difference between a string search and a proximity search is that the strings are searched directly in the Boolware index, while the proximity search has to examine the all words in the record to see if they are as close as requested. This means that phrases searched by proximity search could appear anywhere in the text while a string must appear at the beginning of the text.

Example 1:
```
FIND Text:near(red car)
```
Default value of the parameters (5, 0) will be used. Will find records where not more than 4 other words appear between the words **red** and **car**. The words could appear in any order.

A **red** big **car**	OK just one "foreign word" between red and car
A **red car**	OK no "foreign word" between red and car (phrase)
A **car** that is **red**	OK two "foreign words" and any order
A **red** apple on a big blue **car**	Not OK 5 "foreign words" between red and car

Example 2:
```
FIND Text:near(red car,,1)
```
Still allows 4 "foreign words" between **red** and **car** but they must appear in specified order.

A **red** big **car**	OK red and car in specified order
A **red car**	OK no "foreign word" between red and car (phrase)
A **car** that is **red**	Not OK wrong order
A **red** apple on a big blue **car**	Not OK 5 "foreign words" between red and car

Example 3:
```
FIND Text:near(car red,3,1)
```
Allows 2 "foreign words" between **car** and **red** and they must appear in specified order.

A **red** big **car**	Not OK wrong order
A **red car**	Not OK wrong order
A **car** that is **red**	OK 2 " foreign word" and proper order
A **red** apple on a big blue **car**	Not OK wrong order

Example 4:
```
FIND Text:near(red car,1,1)
```
This time you perform a "phrase search" and the words **red** and **car** must be adjacent.

A **red** big **car**	Not OK "foreign words" between red and car
A **red car**	OK phrase
A **car** that is **red**	Not OK wrong order and "foreign words"
A **red** apple on a big blue **car**	Not OK "foreign words" between red and car

The sub-command **xnear**, proximity search with exclude, is a very special sub-command, which will be described in the *Proximity Search* section below. The command contains two different groups of terms: the terms that should be included in the search (as the for the *near* sub-command) and the terms that should exclude records from the result. The two groups of terms are separated by two characters: semicolon (;) and blank. All terms **before** the semicolon must appear within a specified number of terms while the terms **after** the semicolon must not appear together with the terms before the semicolon.

Besides the term Johnson the following terms will also be part of the search: jonson, johanson, jonhson, joyson, jolison, jolson etc.
When using the near sub-command near you could also use two other (nestled) sub-commands: *case* and *wordasis*. In these cases the search will respect the *case-rules* and *exact* searching, respectively.

Example 5:
```
FIND text:near(case(IT) england).
```

In this example the word IT will be searched case sensitive (to avoid to be mixed up with the pronoun it) while the word england will be searched in an ordinary way. In this case you could mix words that should be searched using the *case-rules* and words that should be searched in an ordinary way.

Example 6:
```
FIND text:near(wordasis(IT England)).
```

In this example the words IT and England will be searched exact. It is **not** allowed to mix exact search with ordinary search.

The sub-command **fuzzy** is used when search should be approved even if the specified search term does not give an exact hit; spelling errors, characters in disorder etc. By comparing the terms in the index to the specified search term Boolware will accept index terms that just differ a little from the search term. By help of the parameters *numchars* and *limit*, (string distance), you could determine how much the terms could differ to be accepted in the search.
The parameter *numchars* tells how much the terms could differ in length while the parameter *limit*, (string distance), tells how many characters could be in disorder, the *usefuzzysyn* set to 1 if fuzzy synonyms should be used if there are any fuzzy synonyms and *withrank* set to 1 if the search result should be rank able in fuzzy order.
The fuzzy rank order is on lowest string distance first: if string distance is 0, i.e. no character disorder at all, will be ranked to 100, one character disorder will be ranked to 99, two characters disorder will be ranked to 98 and so on.
If no parameters are specified the default value for *numchars* is 1 (the terms could differ 1 character in length) and the default value for *limit* is 98 (means that 2 (100 - 98) characters could be in disorder), *usefuzzysyn* is set to 0 and *withrank* is set to 0. The very first character in the index term must however always be the same as in the search term.

Example:
```
FIND "Last name":fuzzy(johnson)
```
Besides the term Johnson the following terms will also be part of the search: jonson, johanson, jonhson, joyson, jolison, jolson etc.

If you change the tolerance to: difference in length (*numchars*) 2 and allow 3 characters in disorder (*limit*) and don't use synonyms, FIND "Last name":fuzzy(johnson, 2, 97, 0, 0), the following terms will also be part of the search: johnstone, johns, juneson, judson, johnny, john etc.

The sub-command **querylim** is used when terms that are contained in many records should be excluded from the query. If you have a query where the operator between the terms is OR and you only want terms that are contained in fewer than 100 records should be part of the query you should specify the following query:

```
FIND text:querylim(Caltex OR Usa OR Petroleum OR Partners OR inc;100)
```

Only terms that are contained in less than 100 records will be part of the current query.

The sub-command **numrange** is used when you want to search a numeric closed interval, when the column is not indexed as numeric (the column contains normal text as well). If you in a word indexed column search the closed interval 10..20, you will get all values greater than or equal to 10: 11, 12, 13 etc. but also 1111, 1211 etc. It means that all values where the two first digits are greater or equal to 10 are valid. It acts in a similar way with values less than or equal to 20: 20, 19, 18, 17 etc., but also 1911, 1894 etc. This means that all values where the two first digits are less or equal to 20 are valid. If you use numrange instead Boolware will search exactly on values in the specified interval (the same number of digits). This means that only the numbers that are between 10 and 20 inclusive will be valid: 10, 11, 13, 13 ,14, 15, 16, 17, 18 ,19 and 20.

The sub-command **searchviaindex** is used when you in a graphical application use Boolware index instead of a search form to get the search arguments for a query. You could browse in the Boolware index and mark the index terms you want to be part of the query. Instead of sending the index terms in the query you just give the corresponding absolute order number. See **Programmer's Guide** Chapter 3. "XML API" section "Fetch index terms". If you specify **searchviaindexnogen** you tell Boolware to skip strings generated by a plugin function (Shrink). Within the parentheses the absolute term numbers from the current index should be specified. These numbers could be single numbers or closed intervals separated by comma (,). A closed interval in this context could be specified in two different ways: 1..100 or 1-100. The numbers an the intervals could overlap and could be specified in any order. The intervals must of course be specified in a correct way (the start interval must be less than the end interval). By default the indextype is set to string (2), but it could be changed by specifying another index type (described in **Programmer's Guide** Chapter 3. "XML API" section "Fetch index terms") as the very first number and ended by a semicolon (;).
Below are a couple of correctly specified sub-commands:

```
FIND Företagsnamn:searchviaindexnogen(7, 5, 1..10, 100, 70-77, 62)
```

In this case the index terms will be fetched from the index type string and no terms generated by the plugin function (Shrink) will be part of the search.

```
FIND Företagsnamn:searchviaindex(1;7, 5, 1..10, 100, 70-77, 62)
```

In this case the index terms to be searched for will be fetched from the index type word.

The sub-command **orsearch**, efficient or-search, searches all specified terms and performs a logical OR between all terms. The command (FIND/AND/NOT/XOR) that precedes the **orsearch** determines how to handle the result. If the terms are stored in a file you could specify the complete file name directly preceded by a '@'.
Example:

```
AND Text:orsearch(@c:\boolware\testdatabase\bigsearch.txt)
```

. All terms in the specified file will be searched for in the column ***Text***. The terms will be **OR**:ed and the result will be AND:ed to the previous result.

The terms that are specified in the sub-command **orsearch** are handled in the same way as terms specified in an ordinary query. Note, that the automatic truncation will not be in effect; you must end the word(s) with an asterisk (*) to truncate the word. If the column is indexed only for word search the specified terms will be treated as words. If the column is indexed *only* for string search the specified terms will be treated as strings; note, that strings that contains special characters (i.e. blank etc.) must be enclosed within quotation marks (") to be treated in a correct

way. If the column is indexed both for word *and* string search you must indicate what kind of search you want (string or word). The common rules will be in effect: if the first term starts with a quotation mark all terms will be treated as strings, if not the terms will be treated as words. You could **not** mix words and strings in the same **orsearch**; it is *the first* term that determines what type of search (word or string) it will be.

The sub-command **orsearchex**, extended orsearch, searches in exactly the same way as the **orsearch** sub-command. The orsearchex sub-command differs from the orsearch sub-command by storing all terms **not** found in the Boolware Index in a special file. The content of this file could be fetched using the Execute command *fetchnotfound* (see **Programmer's Guide** Chapter 1. section "Execute commands in Boolware ").

The sub-command **compress** handles **single characters** that could be specified in different ways and give the same result. The single characters could be separated by the following "break characters": blank ( ), dot (.) slash (/), plus (+) and ampersand (&). The Column must be indexed as compress.

Example:

```
FIND Companyname:compress(IBM Sweden)     gives the same result as
FIND Companyname:compress(I B M Sweden)   gives the same result as
FIND Companyname:compress(I/B/M Sweden)   gives the same result as
FIND Companyname:compress(I&B&M Sweden)   gives the same result as
FIND Companyname:compress(I+B+M Sweden)   gives the same result as
FIND Companyname:compress(I.B/M Sweden)   gives the same result as
FIND Companyname:compress(I&B+M Sweden)
etc.
```

Moreover you do not need to specify the **compress** sub-command if the Column is indexed as compress; the search will automatically "compress" the specified "word".

```
FIND Companyname:IBM Sweden     gives the same result as the above compress
FIND Companyname:I.B.M Sweden   gives the same result as the above compress
FIND Companyname:I/B/M Sweden   gives the same result as the above compress
etc.
```

The sub-command **within** makes it possible to search a "phrase" within a word. A "phrase" in this context is a combination of adjacent characters in a word.

Example:

```
FIND Companyname:within(car*)     could give the following result
Elder<b>car</b>e National Inc
E<b>car</b>deals America Inc
Calling<b>car</b>d Inc
Rent-A-<b>Car</b> Inc
Homes By <b>Car</b>y Inc
Race<b>car</b>adscom Inc
Alchemy Wood<b>car</b>ving & Design
Rusticos Al<b>car</b>traz
Iac<b>car</b>ino & Son Inc
Al<b>car</b> Products Corp
Halo<b>car</b>bon Laboratories Inc
Dos<b>car</b> Corporation
etc.
```

```
FIND Companyname:within(car)     gives the following result from the same table as above
Rent-A-<b>Car</b> Inc
Al<b>car</b> Products Corp
Dos<b>car</b> Corporation
```

The sub-command **case** makes it possible to search case sensitive. Normally you do not want to distinguish between upper and lower case when searching but sometimes it could be impossible to get the proper precision: A text database contains articles describing different activities for companies. You want to search for companies involved in the IT business and it will be mixed up with the very common word ‘it’. To avoid this you should use the **case** sub-command to distinguish between the meaning of these two words. When searching you specify the words that are indexed as case sensitive exactly as they look:

Example:
```
FIND text:case(IT) and england.
```

In this case all articles containing IT (in upper case) and England (in any case) will be selected. Articles containing the word ‘it’ in combination with England will not be selected. In chapter 5 section "Editing indexing properties for a table/view" the rules how to get the best precision are described.

The sub-command *case* could also be used in proximity search.

Example:
```
FIND text:near(case(IT) england).
```

In this the word IT will be searched case sensitive while the word england will be searched in an ordinary way.

The sub-command **wordasis** makes it possible to search case sensitive (as is). Normally you do not want to distinguish between upper and lower case when searching but sometimes it could be impossible to get the proper precision; when words with and without diacrits have different meaning. In this case it could be wise to index the words exactly as is with no normalization. When searching you must specify the search terms exactly as they are written in the text to get the proper answer.

The sub-command *wordasis* could also be used in proximity search.

Example:
```
FIND text:near(wordasis(IT England)).
```

In this case all words (IT and England) must be searched exact; no mix between exact and ordinary search is allowed.

The sub-command **stringasis** works as **wordasis** but handles strings rather than words.

The sub-command **withinstring** makes it possible to search a "phrase" within a string. A "phrase" in this context is a combination of adjacent words in a string.

Example:
```
FIND Personname:withinstring(John Paul*)
```
could give the following result
```
Andersen John Paul
Beck John Paul Mike
Smith John Pauly Emerson
etc.
```

```
FIND Personname:withinstring(John Paul)
```
could give the following result
```
Andersen John Paul
```

The sub-command **geocircle** is used when you want to perform a geographic search for something within a special area. By help of latitude and longitude you determine a point from which you specify a radius to tell the area you want to search within. The sub-command is handled like other sub-commands and thus could be combined with other operators and commands. Latitude and longitude are defined by column names of the columns holding the

information. You could specify the coordinates in different ways: GPS (WGS84), RT90 and System32/34.

Example:
Suppose you want o find all restaurants within 300 meters from a given position.

```
FIND Bransch: restaurant AND geocircle(lat:N59 20 43.36; long:E17 57 54.65;
R=300)
```

NOTE It is very important that the parameters are specified in the proper order: latitude; longitude.

The sub-command **georectangle** is used when you want to perform a geographic search for something within a special area. By help of latitude and longitude you determine two points. These points will be the opposite corners in a rectangle and the area of this rectangle is the limit you search within. The sub-command is handled like other sub-commands and thus could be combined with other operators and commands. Latitude and longitude are defined by column names of the columns holding the information. You could specify the coordinates in different ways: GPS (WGS84), RT90 and System32/34.

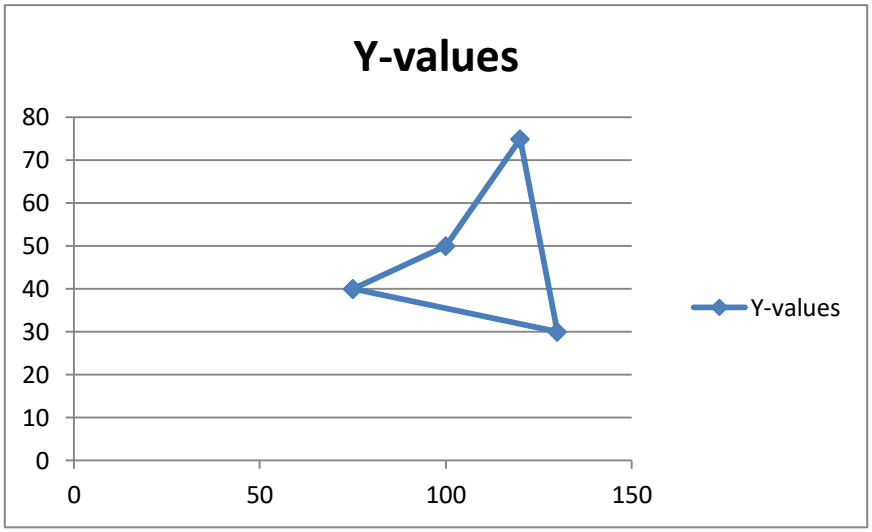
Example:
Suppose that you want to find all cinemas within an area (rectangle) defined by two points (opposite corners in the rectangle).

```
FIND Branch:cinema* AND georectangle(lat:N59 20 43.36; long:E17 57 54.65; lat:
N59 24 43.36; long:E18 00 17.65)
```

The sub-command **geopolygon** is used when you want to perform a geographical search within a given region. This region should be a closed polygon with coordinate pairs for each corner. The polygon should be described by the standard "Well Known Text";
POLYGON((x1 y1, x2 y2,…, x1 y1)) or MULTIPOLYGON(((x1 y2, x2 y2, …x1 y1))).
Note that the x-axis holds the longitude values and the y-axis holds the latitude values. The coordinates are separated by space and the coordinate pairs are separated by comma.
Example:
geopolygon(latitude_field; longitude_field; POLYGON((12.920521 58.469976,12.892362 58.510879,12.977524 58.54672,12.920521 58.469976)))

The sub-command is handled like other sub-commands and thus could be combined with other operators and commands. Latitude and longitude are defined by column names of the columns holding the information. You could specify the coordinates in different ways: GPS (WGS84), RT90 and System32/34.

Example:
Find all records in the actual table within the given region described by the polygon with the boundaries (75, 40), (100, 50), (120, 75), (130, 30), (75, 40). The latitude column name is *lat* and the longitude column name is *long*.




```
Find geopolygon(lat; long; POLYGON((75 40, 100 50, 120 75, 130 30, 75 40)))
```

If the data is stored as WGS84 the values must be quoted like:

```
Find geopolygon(lat; long; POLYGON(( "E 8 38 12.39" "N 49 52 11.83", "E 8 38
16.71" "N 49 52 7.10", "E 8 38 26.44" "N 49 52 3.37", "E 8 38 15.01" "N 49 52
5.51", "E 8 38 12.39" "N 49 52 11.83")))
```

The sub-command **geowithinpolygon** is used when one ore more polygons, stored in WKT/GeoJSON format, will found that contains a given coordinate; x, y. The values *x* and *y* is given as decimal values. For more details about polygons, in current result, that contains the given coordinate use the Execute command *fetchpolygonsfound* (see **Programmer's Guide** Chapter 1. section "Execute commands in Boolware ").

Example: Find polygons stored in the field "shape" that contains the coordinate: 1629353.50, 657956.70 :

```
Find shape:geowithinpolygon(1629353.50, 657956.70)
```

The sub-command **syn**, synonyms, is a way to neutralize search terms depending on meaning. Before the search starts all synonyms for the given term is fetch and the search will OR the result from all synonyms.

The sub-command **stem**, stemming, is another way to neutralize the meaning of the terms. The difference is that this is done automatically. All words are transformed into their common case; the verb to infinitive etc. and the search terms are transformed in the very same way. When searching for **walking** you will get all records containing: walk, walker, walking, walked etc. Compared to the synonyms only one word is search for; besides the index will get smaller. The disadvantage is of course that you could not distinguish between the different terms.

The sub-command **sound**, phonetic, is useful when searching for names etc. The search is more focused on the pronunciation than the spelling. E.g. the Swedish telecom company Ericsson will be found even if you have specified: Eriksson, Erikson, Ericson, Erixon etc. as a search term.

In some cases you could get "weird" combinations when using *sound* and "*link words*" together. To get around this problem you could combine the sub-commands *nolinkwords* (see description below) and *sound*.

Example:

```
FIND Name:nolinkwords(sound(jon Eriksson))
```

In this case the search only will use the phonetic terms for *jon* and *Eriksson* when searching; no "*link words*" will be part of the search.

The sub-command **noti**, excluding search, is useful when searching records that do not contain a specified Term. This is normally done with the **NOT** command/operator, but it could be awkward when you want to search for records that do not contain any information in one column (see example below).

The sub-command **prefix**, search prefixed, could be used in connection with hierarchy text (XML coded text) to extend the precision when searching. The sub-command contains two parts: the attribute values (prefixes) and search terms. The two parts are separated by a semicolon (;). The prefixes are divided into groups, which reflect the hierarchy in the current field. Each prefix should be enclosed in quotation marks (") and each prefix group should be ended by a comma (,). The last (or only) prefix group should be ended by a semicolon (;) instead of a comma (,). The search terms that follows the prefix groups are specified in an ordinary way. A short description of the syntax by help of an example will be found in Chapter 5 section "Indexing of fields within columns".

The sub-command **dupstamp**, multiple categories, is used when you want to retrieve all records that have more than one category in the stamp column (see Chapter 5 section "Stamp multiple categories when indecisive" above).

The sub-commands **dupeliminate** and **dupeliminaterandom** are used when you want to eliminate duplicates from the current result. These two commands takes two parameters: the name of the requested duplicate rule and number of duplicates to save from the current result. The parameter *numtosave* is optional and if omitted it will be set to **1**. This parameter shall be set to a positive integer value to specify number of duplicates to save from the current search result. If *numtosave* is set to **0, all** records that have duplicates (including the main record) will be eliminated from the current search result according to specified duplicate rule.
The sub-command **dupeliminaterandom** works the same way as the sub-command **dupeliminate** with the difference that the duplicates to be saved are randomly selected from current result.

The sub-commands **getduplicates_main_records**, **getduplicates_without_main_records** and **getduplicates_all_records** gives the result of the records involved in a duplicate rule. The command **getduplicates_main_records** gives the result of records that are main-records for the duplicate rule, **getduplicates_without_main_records** gives the result of records that are duplicates to the main-records but without the main-records itselfs and **getduplicates_all_records** gives the result for all records involved in a duplicate rule.

Duplicates are eliminated by help of different duplicate rules. For each rule you specify what should be regarded as a duplicate and what priority order they should have. A special case is when no result exists and the all "main records" for all duplicates will be selected. Several different rules could be specified for each table. Read more about this in Chapter 5 "Editing indexing properties" section "Editing indexing properties for a table/view" under tab DUPLICATES.

The sub-command **nolinkwords** is used when you want to override the link words when searching. When using this sub-command no link words will be part of the current search. This is most usable when dealing with the sub-command *sound* (see above).

The sub-command **set** - use result from a saved Set - when you want to re-use the result from a previous query. No query is performed but the saved result is used directly in the current query.

The sub-command **query** - use query-string from a saved Query - when you want to re-use the query-string from a previous query. The Command, Operators and search terms from the saved Query is used in the current query.

The sub-command **result** - use result from a saved Result - when you want to re-use the result from a previous query. No query is performed but the saved result is used directly in the current query.

The sub-command **scratch** - use a named saved result - when you want to re-use the result from a previous query. No query is performed but the named saved result is used directly in the current query.
The difference between this sub-command and **set/result** is that the result will **not** be saved on disk but will be kept in memory. This sub-command is normally used in a Boolware Flow.

The sub-command **rpath** is used when describing the path between different tables in a Relate command. See section Related search (Join) below. The parameters in **rpath** should be within parentheses and should always in pairs: source_table.column, target_table.column.

The sub-command **maxqueryexecutiontime** is used to set a maximum execution time in milliseconds for the current command. The sub command must immediately follow the current command. The specified max query execution time is just valid for the current command. If no max query execution time has been set the "Max execution time" for sessions set in the Boolware Manager will be used. When the max query execution time is exhausted the execution of the command will be terminated with an error code.

The **EXISTS** subcommand is used to narrow a search result based on whether data exists. The command consists of two parts, first a search part (search criteria) to be specified in parentheses and a conditional part, specified after the search part. See more in the section "Searching a Boolware Index" and "EXISTS search".

Query examples:

Boolware query and meaning

```
FIND name:robert AND city:boston
```
Find all Robert in Boston. Find all records that contain robert in name and boston in city.

```
FIND text:red AND text:car*
```
Find all red cars in text. Now all records containing the terms red and car will be found. Note; records containing red house and small car will also be found.

```
FIND text:near(red car*, 3, 1)
```
Find all records containing red car, where the terms must be within three words and in order. In this case all records containing red house and small car will not be found.

```
FIND text:near(red car, 1, 1)
```
Find the phrase red car. In this case only records containing red car as a phrase will be found.

```
FIND text:near(red car)
```
If no parameters for distance and order are specified the default values are: distance 5 and no order (0).

```
FIND text:xnear(red car; Volvo Saab, 10)
```
Find all records containing red car, where the terms must be within ten words. The terms could appear in any order. If the terms red car only appears together with Volvo or Saab, the record will not be included in the result.

```
FIND text:"red car"
```
Find the phrase red car. This is another way to specify the phrase search. In this case Boolware will check if the columns text is indexed for string. If so a string search rather than a proximity search will be performed.

```
FIND name:robert AND "":red AND car
```
Find all robert in name and red car as "Free text".

```
FIND car* AND (red or black)
```
Find all red or black car in "Free text".

```
FIND Profit:>1000
```
Find all companies that have a profit over 1 000.

```
FIND Profit:<3000
```
Find all companies that have a profit less than 3 000.

```
FIND Profit:500..900
```
Find all companies that have a profit between 500 and 900 (inclusive).

```
AND text:near(blue velvet)
```
Find all records that contain the terms blue and velvet within 5 words in any order. AND this result to the previous queries.

```
NOT text:syn(red)
```
Before the query starts all synonyms for red are fetched. All these words are searched for and their results are OR:ed. This result is then NOT:ed with the result from the previous queries.

```
OR text:thes(animal)
```
Before the query starts all children for the word animal are fetched. All these words are searched for and their results are OR:ed. This result is then OR:ed with the result from the previous queries.

```
AND text:stem(walking)
```
Boolware stems the word walking and will probably get walk. Boolware uses this word when searching and get records containing words like: walk, walking, walked, walker etc. This result is then AND:ed with the result from the previous queries.

```
FIND name:sound(eriksson)
```
The word eriksson is coded by a special function to get a word that gives priority to the pronunciation rather than the spelling. When searching records containing words like: ericsson, ericson, erixon, erikson will also be found.

```
FIND name:noti(*)
```
Extract all records that do not have any information in the name Column.

```
AND sim(any text....)
```
Any text is normally copied from some records in the database or from the internet. This text is compared to all records in the database and the result will be AND:ed to the previous result. The records will appear in similarity order; most like first.

```
FIND sim(any text <CRLF>  .#$Profit:20;10 <CRLF>  .#$Turnover:2000;20 <CRLF>
.#$NoEmployees:20;15)
```

The text is handled as described above. A new line that starts with the sequence .#$ indicates a numeric similarity. In this example the columns Profit, Turnover and NoEmployees are used. The value to search for is specified directly after the column name and the tolerance in percent follows after a separating semicolon. As a third parameter you could specify a weight.

```
TABLES(*) FIND text:eriksson
```
Search for eriksson in the text column in all tables in the database.

```
BACK
```
Back one command in the Query History

```
FORWARD
```
Move forward one command in the Query History

```
FIND dupstamp()
```
Finds all records that has been stamped by more than one category.

```
FIND dupeliminate(rule)
```

Finds all main records for the specified rule. See Chapter 5 "Editing indexing properties" tab DUPLICATES.

```
FIND set(s4) OR s8
```
Between the saved resultset for S4 and S5 a boolean OR is performed.

```
FIND query(query1) AND query(query4)
```
Queries on the commands and arguments saved in "query1" and "query4" are performed and a boolean AND is performed between the two results to get the final result.

```
FIND result(result1) NOT result(result4)
```
Between the saved result "result1" and the saved result "result4" a boolean NOT is performed.

```
RELATE(Customers)FIND Products.ProductName:Uncle Bob's Organic Dried Pears
```
Queries table Products column ProductName for Uncle Bob's Organic Dried Pears. The result will be transformed via other tables to table Customers. A more detailed example could be found in section "Related search" below.

```
FIND maxqueryexecutiontime(40) text:near(blue velvet) AND name:smith
```
If the specified query takes more than 40 milliseconds the query will be terminated with an error code.

Examples on interactive queries etc.

Below the search capability of Boolware will be described. The following functions will be described: inspect a Boolware index, search a Boolware index based on search criteria, rank the result from a query, browse through the Query History and get the result from the RDBMS.

What is a Boolware Index

When a data source has been registered for indexing with Boolware a set of index files is created for each chosen Column in all requested Tables. The terms in the different Columns could be indexed in a various number of ways. Before the Boolware indexes are built, different attributes (index types) are set for each Column and depending on the attributes set different search terms will be generated. Thus each single term could be indexed in several different ways. In short you could say that a Boolware index holds all variations of all the search terms for each Column.

The index types specified for each Column reflect the way you could search the Column; if the *near* index type is not set for a Column you could not perform a Proximity Search or rank the result set by search terms.

The Boolware indexes provide a very fast and flexible way to get an accurate answer to complex queries.

Viewing the contents of a Boolware Index

A very convenient way to browse a Boolware index and examine what search terms are available for a specific index type is to use the View Index API. You only specify the Column, the index type and a start point in the index. The start point is normally the beginning of the index, but by specifying the beginning of a term you are interesting in, the browsing starts from this point. As a reply you get the search terms and a numerical data indicating the number of rows the current term is contained in. The number of search terms presented at each time is determined by the Application software.

If you have performed a query and just want to see what index terms still are relevant for the current result you could use the function zoomed index.

By specifying wild cards in the start term you will only get index terms that match the specified term. This function could be used both for the Global and the Zoomed index. For examples see "Programmer's Guide" Chapter 3 "XML API".

There is also a possibility to present a grouped index in three different ways (see "Grouped View Index" below).

To present index terms depending on occurrence (frequency) is another possibility. In this case you specify a number rather than a term to positioning in the index.

Boolware will in all cases respond with the current term and a number which indicates in how many rows the current term is contained. The number of terms to be fetched is determined by the application.

You could also export the selected index terms to a specified file for later use.

View complete Index

The most common way to use this feature is to browse the entire index which means that all terms for the chosen Column and index type will be presented. The numerical data for each term reflects the occurrences within the entire database. The Global View Index feature could be performed at any time and no other action needs to be taken in advance.

View Zoomed Index

When there is a result from an earlier search session it could be interesting to see which terms still are contained within the current result set. To achieve this the Zoomed View Index feature is used. Now only search terms still within the result set will be displayed and two different numerical data are presented along with the current search term; the first value reflects the occurrences within the entire database (as for the Global View Index), while the second value indicates the number of occurrences within the current result set. The Zoomed View Index feature could only be used after a search session when there is a result set.

View Grouped Index

A Grouped Index could in Boolware be regarded as some sort of hierarchic Index. One example of Grouped Index is date (yyyymmdd).

A Grouped Index is used both when searching (see Chapter 5 "Editing Index properties" section "Grouped") and when presenting the terms that are contained in the Index.

In Boolware you could present the terms in a Grouped Index in four different ways:
1. The normal way; the lowest level of the terms will presented in alphabetical order
2. Hierarchic sorted on the number of occurrences; year, month day
3. True Index order, where each group will be presented separately
4. Hierarchic sorted in alphabetical order

These four different ways of presenting the Grouped Index could of course be combined with the Relevant Index; just present the terms that are contained in the latest search result.

Example:

In a Table there is a Column containing date. This Column has the index attribute Grouped and the groups are: year(4), year/month (6) and year/month/day (8). The numbers within parentheses indicates the number of characters in each group. After a query session you have got 100.000 records as a result and you want to present the Grouped Index in the four different ways described above.

1. The complete Term will be presented in alphabetical order (the number within parentheses represents the number of occurrences for the current Term).

19040101	(26)	This is the first relevant Term in this Index
19040102	(30)	This is the second relevant Term in this Index
•		
•		
19500101	(47)	This is a relevant Term in the middle of this Index
•		
•		
20021230	(21)	This is the second last relevant Term in this Index
20021231	(19)	This is the last relevant Term in this Index

2. Hierarchic presentation (year/month/day); highest number of occurrences first

1945	(14.837)	This year contains the highest number of occurrences
194503	(1.318)	This month in the current year contains the highest no. of occurrences
19450321	(60)	This day in the current month contains the highest no. of occurrences
19450312	(52)	
19450310	(48)	
19450327	(47)	
•		
•		
194504	(1.154)	This month in the curr year contains the second highest no. of occ.
19450411	(53)	
•		
•		
1950	(5.998)	This year contains least number of occurrences
195003	(718)	This month in the current year contains the highest no. of occurrences
19500310	(22)	This day in the current month contains the highest no. of occurrences
19500327	(19)	

3. True index order, all groups are presented separately (year/month/day)

1904	(7.286)	The first year in this Index
1905	(7.989)	The second year in this Index
•		
•		
2001	(6.994)	Second last year in this Index
2002	(6.004)	The last year in this Index
•		
•		
190401	(828)	The first month in the first year
190402	(698)	The second month in the first year
•		
•		
200211	(730)	The second last month in the last year
200212	(698)	The last month in the last year
•		
•		
19040101	(26)	The first day in the first year
19040102	(30)	The second day in the first year

•
•

20021230	(21)	The second last day in the last year
20021231	(19)	The last day in the last year

4. Hierarchic presentation sorted in alphabetical order

1904	(14.837)	This is the first year
190401	(1.318)	This is the first month in the first year
19040101	(60)	First day in the first month in the first year
19040102	(60)	Second day in the first month in the first year

•
•

19041230	(60)	Second last day in the last month in the first year
19041231	(60)	Last day in the last month in the first year

•
•

1905	(52)	The second year
19050310	(48)	
19050327	(47)	

•
•

194504	(1.154)	Something in the middle of the Index
19450411	(53)	

•
•

2002	(5.998)	This is the last year
200201	(718)	The first month in the last year
20020110	(22)	A day in the first month in the last day
20021231	(19)	The last day in the last moth in the last year

View Frequency Index

Normally you view a Boolware index in alphabetical order but sometimes it could be convenient to view the index in frequency order. Frequency in this context means the number of rows the current term appears in.

Like the presentation in alphabetical order you could present the index in frequency order from the entire index or from the relevant part of the index (only terms that appear in the current result). Of course you could choose what type of index type (word, string, phonetic etc.) you want to present.

You could also use the function export to write the selected terms to a specified file.

The terms could be presented in ascending or descending (default) order regarding the frequency.

If more than one term has the same frequency the terms will be sorted in alphabetical order (always ascending) within that frequency.

To make the presentation of frequency index more efficient you could specify a limit: >[=]**n** or <[=]**n**, where **n** is the frequency. If you are interested in the most common terms it could be a waste of resources to select and sort the terms appearing in just a few rows.

If an index contains a lot of unique terms - let's say 20 millions in a table containing 10 million rows - it could be unnecessary to sort all 20 million terms if just are interested in a few thousands of the most common terms. In this case you save a lot of resources by use a limit >1000, which will give you all terms that are contained in more than 1. 000 rows.

When the terms are extracted you could positioning in the selected terms by specifying a frequency to start from. This is done by specifying a frequency without leading (>,<,=). This has the same effect as specifying a term when browsing an index in alphabetical order.

Below are two examples that describes how you could use the frequency index.

In these examples the principal - not the correct syntax - is used. The correct syntax will be found in the manual "Programmer's Guide" chapter 1 – "Execute commands in Boolware".

Example 1:
After a query in the table, Articles, the most common terms in the column, Text, from the current result should be presented. The result from the query is 12.437 and you are just interested in terms (words) that appear in more than 200 rows. The terms should be presented in descending order (the most common terms first).

indexex **table**="Artiklar" **field**="Text" **max_terms**="50" **type**="14" **zoom**="yes" **freqtype**="1" **freqlimits**=">200" **continuation**="no" **order**="desc"

Example 2:
Position to the first term that appears in 500 rows (or the closest (less than 500)) before the presentation takes place. All terms that appear in less than 500 rows will be presented. In this selection there are only terms that are contained in more than 200 rows (see Example 1) so the last term to be presented is contained in at least 201 rows.

indexex **table**="Artiklar" **field**="Text" **max_terms**="50" **type**="14" **zoom**="yes" **freqtype**="1" **freqlimits**="<=500" **continuation**="no" **order**="desc"

Important: The function frequency index is used to select to most or least common terms in the complete index or in the current result. To select these terms the complete index has to be read. If no limit is specified all terms have to be ordered /ascending/descending) before presentation which could take a very long time when the index contains several million terms.

Thus it is a very good idea to specify a limit when not all terms are wanted. Assume there is a table containing a couple of million rows and an index containing 10 million unique terms. At least 70% of all terms appears in less than 500 rows which means that a limit >500 will reduce the sorting time substantially. The result will be the 3 million most common terms.

Hits projected hierarchically over one or multiple other values (SubZoom)

Boolware can project number of found hits hierarchically over one or more other values (columns). This functionality is an extension to the regular Zoomed index, and is hence called "sub zoom". Sub zoom extends ordinary zoom in that it can project multiple columns against each other in several levels, in increasing precision for each level.

Sub zoom doesn't have a command of its own, but instead uses the regular index command with an extended syntax for specifying which field to list. For a normal index listing, you would specify a single column from where you want searchable terms listed. The syntax for sub zoom is extended in that it is possible to specify more than one field, using parenthesis and commas. Parenthesis is used to specify a new level, and comma is used to separate fields within the same level. Up to five columns can be used.

Found values are listed in alphabetic order within each level if nothing else is said. However, it is possible to reorder the values by frequency (hit count), ascending or descending. This is done by specifying the parameters **type**="14" (frequency) and **order**="asc/desc ".

Example: Assume a table containing *Country*, *Language* and *MediaType*.

indexex **table**="table" **field**="Country(Language,MediaType)" **max_terms**="5000" **type**="2" **continuation**="no" **commandheader**="no"

```
[Country]
Germany, 13 found
  [Language]
  eng, 4 found
  ger, 9 found
  [MediaType]
  net, 7 found
  paper, 6 found
[Country]
Holland, 7 found
  [Language]
  eng, 2 found
  ger, 5 found
  [MediaType]
  net, 4 found
  paper, 3 found
```

The above index command orders a listing of number of hits spread over Countries, and within countries on Language and Media Type respectively. Note that Language and MediaType are shown on the same level, and that both are projected (zoomed) on their upper level Country.

Example 2: Show number of found companies spread on Geography and Size (number employees):

indexex **table**="table" **field**="City(Zip(Employees))" **max_terms**="5000" **type**="2" **continuation**="no" **commandheader**="no"

```
[City]
Paris, 2290 found
  [Zip]
  169 30, 11 found
    [Employees]
    1, 1 found
    2, 3 found
    3, 5 found
    4, 1 found
    8, 1 found
  [Zip]
  169 31, 17 found
    [Employees]
    1, 4 found
    2, 6 found
    3, 2 found
    5, 2 found
    8, 1 found
    11     , 1 found
    12, 1 found
  [Zip]
  169 32, 13 found
    [Employees]
    1, 1 found
    2, 2 found
    3, 7 found
    13, 2 found
    27, 1 found
```

The values are ordered alphabetically by default, but this can be changed. Let's order the values decreasingly on frequency:

indexex **table**="table" **field**="City(Zip(Employees))" **max_terms**="5000" **type**="14" **order**="desc" **continuation**="no" **freqtype**="2" **commandheader**="no"

```
[City]
```

```
Paris, 2290 found
  [Zip]
  169 31, 17 found
    [Employees]
    2, 6 found
    1, 4 found
    3, 2 found
    5, 2 found
    8, 1 found
    11     , 1 found
    12, 1 found
  [Zip]
  169 32, 13 found
    [Employees]
    3, 7 found
    1, 2 found
    2, 2 found
    13, 1 found
    27, 1 found
  [Zip]
  169 30, 11 found
    [Employees]
    3, 5 found
    2, 3 found
    1, 1 found
    4, 1 found
    8, 1 found
```

Extended statistics between several Boolware indexes

This function is an extension of the SubZoom function that is described above.

The function could be used to perform simple analyzes where the values from the different columns are "zoomed" from the current result.

To be able to use the columns for this purpose they have to be indexed as string or numeric and they must be saved in the Boolware data file when creating the template files.

The function could be divided into three main parts: 1. create a report template, 2. build report template files and 3. create input for presentation of statistics using the created report template.

Below you could see how to use this function:

1. Create a report template
1.1 Give the report template a unique name
1.2 Specify which columns that should be involved and their hierarchical order
2. Build the report template
2.1 Create files for the report template using all records in the current table
3. Use the created report template to create input for presentation of statistics
3.1 Choose "statistics" (max, min, sum or avg/mean)
3.2 Specify max number of observations on the lowest level

By using the Boolware Manager you could do the 1. and 2. while number 3. should be done in an application connected to Boolware (for test purposes the Boolware Demo could be used; however, it does not contain any graphic presentation of statistics).

The result from Boolware is records where the values are calculated according to the chosen "statistics". The records are sorted in an order that makes the "best" records appear first according to the chosen "statistics".

The records from Boolware should normally be sent to a program that could create some type of graphic presentation of the wanted statistics: histogram, bar chart, pie chart, scatter plot etc. after they have been "massaged" to suit the chosen program.

How to create a Report template

By help of Boolware Manager you could create any number of named report templates. Each report template contains the columns that are involved and their hierarchical order.

In the tree structure of databases, tables and views you should choose 'Report templates' for the wanted database. You will now enter the 'Reports' window where you choose the table that should contain the current report template.

After selecting a table you should hit 'New template' and fill in values in the window 'Edit report template'.

The first thing to do is to give the report template a unique name. Then you select a column for the first dimension (only valid columns are shown). When the first dimension is filled in you have a choice to select a second dimension. After this you select the columns for aggregated values. When all values have been filled in you enter OK to save the current report template.

Now you have created a report template and to make it usable you must build the corresponding report template base files by pushing the button 'Build template'. This means that values from the specified columns in all records in the current table will be extracted in the proper hierarchical order. The files will be saved with the name of the report template. Depending on the size of the chosen columns it could take some time. The progress of the building is shown under the 'Tasks' tab.

These base files will be used later on when you want to perform different "statistics" against different results. The following "statistical" methods are available: the max value, (max), the min value (min), the average value (avg/mean) and the sum (sum).

Some examples (that will be used in "*How to use a report template*" below):

Example 1:
In a table, Companies, that contains geographical and economic information about companies you want to perform statistics on number of employees distributed on cities. In the window 'Report templates' you choose the table Companies and press "New template'. In the window 'Edit report template' you give the report a unique name (*CompanyCityEmployees*) and then you specify City as the first dimension. In this example you don't want a second dimension but specify Employees directly as Column for aggregated value. When you press the 'Build template' base files for the current report will be built. These base files could later on be used to create input to programs that could present the wanted "statistics".

Example 2:
From the same table as in "Example 1" you want to create a report template where you get statistics on turnover distributed on companies within cities. In the window 'Report templates' you choose the table Companies and press "New template'. In the window 'Edit report template' you give the report a unique name (*CompanyCityCompanynameTurnover*) and then you specify City as the first dimension. In this example you want Companyname a second dimension and Turnover as Column for aggregated value. When you press the 'Build template' base files for the current report will be built. These base files could later on be used to create input to programs that could present the wanted "statistics".

How to change a Report template

By help of Boolware Manager you could change a selected report templates. It is the same procedure as create a report template but you mark a report template and hit 'Modify template' instead of 'New template' in the 'Reports' window.

When you have done the wanted changes you hit OK and the changed report template will be saved. Before using the changed report template it has to be re-built (Build template).

How to delete a Report template

By help of Boolware Manager you could remove a selected report templates. It is the same procedure as change a report template but you push the 'Delete template' button instead of the 'Modify template' button in the 'Reports' window.

The report template and its corresponding files will be removed.

How to use a Report template (create input for graphic presentation of statistics)

When you have created a report template and have built its base files it could be used to create simple analyzes based on available "statistics".

Boolware generates a number of records that should be input to a program that could present the "statistics" graphically. A detailed description of these records and all index attributes used to generate these records could be found in the manual "**Programmer's Guide**" Chapter 3 "XML API" section "Fetch index terms".

Below you will find a some examples on how an XML request could look like when creating different "statistics" using the two examples above.

Example 1:
Use the base files that was built for the report template *CompanyCityEmployees* and extract the cities that contains companies with most employees. In this case the "statistics" method is "sum" and you only want the 9 records that contain most employees.

As *zoom*=yes only companies that are part of the current result (companies within Florida) will be aggregated.

As *skipgeneratedterms*=yes no strings generated by a plugin function will be presented.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SoftboolXML_requests>
   <SoftboolXML_request type="query">
   <open_session name=""/>
   <database name="Companies"/>
   <table name="Companies"/>
   <query>FIND State:Florida</query>
   <response/>
   </SoftboolXML_request>

   <SoftboolXML_request type="index">
   <database name="Companies"/>
   <table name="Companies"/>
   <index
   field="City(Employees)" zoom="yes" type="2" max_terms="9"
     statistics="sum" skipgeneratedterms="yes"
     reporttemplatename="CompanyCityEmployees">
   </index>
   </SoftboolXML_request>
</SoftboolXML_requests>
```

Example 2:
Use the base files that was built for the report template *CompanyCityEmployees* and extract the cities that contains companies with most employees. In this case the "statistics" method is "sum" and you only want the 12 records that contain most employees.

As *zoom*=0 all companies in the table Companies will be aggregated. The result from the query FIND State:Florida will be ignored.

As *skipgeneratedterms*=yes no strings generated by a plugin function will be presented.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SoftboolXML_requests>
   <SoftboolXML_request type="query">
   <open_session name=""/>
   <database name="Companies"/>
   <table name="Companies"/>
   <query>FIND State:Florida</query>
   <response/>
   </SoftboolXML_request>

   <SoftboolXML_request type="index">
   <database name="Companies"/>
   <table name="Companies"/>
   <index
   field="City(Employees)" zoom="no" type="2" max_terms="12"
     statistics="sum" skipgeneratedterms="yes"
     reporttemplatename="CompanyCityEmployees">
   </index>
   </SoftboolXML_request>
</SoftboolXML_requests>
```

Example 3:
Use the base files that was built for the report template *CompanyCityCompanynameTurnover* and extract the cities that have companies with the highest turnover. In this case the "statistics" method is "max" and you only want the 15 companies that have the highest turnover.

As *zoom*=yes only companies that are part of the current result (companies in Chicago, Washington and Seattle) will be aggregated.

As *skipgeneratedterms*=yes no strings generated by a plugin function will be presented.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SoftboolXML_requests>
   <SoftboolXML_request type="query">
   <open_session name=""/>
   <database name="Companies"/>
   <table name="Companies"/>
   <query>FIND City:Chicago OR Washington OR Seattle</query>
   <response/>
   </SoftboolXML_request>

   <SoftboolXML_request type="index">
   <database name="Companies"/>
   <table name="Companies"/>
   <index
   field="City(Companyname(Turnover))" zoom="yes" type="2" max_terms="15"
     statistics="max" skipgeneratedterms="yes"
     reporttemplatename="CompanyCityEmployees">
   </index>
   </SoftboolXML_request>
</SoftboolXML_requests>
```

Execute commands

Commands can be sent as text to Boolware, using the BCConnectExecute and BCExecute functions, using XML/JSON (execute-element/property) calls. A description of each command can be found in the "**Programming Guide**" under the chapter "Execute commands to Boolware". The following execute commands can be sent to Boolware:

EXPORTRESULT
FETCHCOORDINATES
FETCHNOTFOUND
FETCHPOLYGONSFOUND
FLOWINFO
FLOWINFOPARAMETERS
GETAUTOTRUNC
GETENCODING
GETEXITPOINT
GETHISTORY
GETINDEXEXIT
GETMAXEXECUTIONTIME
GETQUERYLIMVALUE
GETSESSIONTIMEOUT
GETSETSEARCH
GETSTRICTASIS
HITLISTPOSITION
INDEXEX
LOG
PERFCOUNTERS
RANK
RANKTERMS
RELATE, TABLES
REVIEWSET
DELETESET
SAVEQUERY
REVIEWQUERY
DELETEQUERY
SAVERESULT
REVIEWRESULT
DELETERESULT
SAVESCRATCH
DELETESCRATCH
SETAUTOTRUNC
SETENCODING
SETHISTORY
SETINDEXEXIT
SETMAXEXECUTIONTIMEOUT
SETQUERYLIMVALUE
SETQUERYOPTIONS
GETQUERYOPTIONS
SETSESSIONTIMEOUT
SETSETSEARCH
SETSTRICTASIS
STATISTICS

The Boolware Query Language

The Boolware Query Language (QL) is a simple version of Common Command Language adjusted for Boolware. The main purpose is to be able to specify both simple and complex

queries in a uniform and simple way. The result from a query is the rows that satisfies the specified search criteria. The result set could be ranked to be presented in different orders.

The QL is built up by Commands, Operators, Sub-commands and search arguments.

The Boolean operators (AND, OR (inclusive), NOT and XOR (exclusive)) in any combinations are supported. To be sure of the result parentheses should be used when necessary.

Sub-commands like NEAR, XNEAR, ORSEARCH, SYN, THES, STEM, SOUND and SIM are used when the query is directed to a special index type.

A sub-command has the following syntax: SUBCOMMAND(terms). If there are unbalanced parentheses in any term in the sub-commands: NEAR, XNEAR, ORSEARCH and ORSEARCHEX, the syntax has to be changed to: SUBCOMMAND([[terms]]); the starting parenthesis is changed to ([[and the closing parenthesis is changed to]]).
Example:
FIND Text:ORSEARCH([["1.) Chapter 1" "2.) Chapter 2"]])
The two strings: 1.) Chapter 1 and 2.) Chapter 2 are OR:ed together.

The search arguments consist of search terms that could contain wild cards (*, ?, ! and #) in any combination. The * replaces 0 - 126 bytes of any type, the ? replaces one character of any type, the ! replaces one character of type letter and # replaces one character of type digit.

NOTE In a string could only the only allowed wild cards are: * and ?.

There are three important settings which could be activated (on) or deactivated (off) that affect the functionality of the Boolware Query Language:

1. Automatic truncation means that Boolware automatically will find everything starting with the specified search term.

2. Query History means that Boolware saves all necessary information in the memory to make it possible to navigate within the current search result (see section "Set Search" below).

3. Search Set means that Boolware saves all necessary information for the complete session on disk. This makes it possible to use earlier results without researching (see section "Set Search" below).

In the manual "**Programmer's Guide**" you will find a description how to manipulate the above settings.

Important: In the Boolware query language some characters - that could be part of a search term - have a special meaning. Below you could see the characters that should be used with care when determine which characters should be part of a search term (see Chapter 10; Lists and Tables).

*	is used to replace 0-126 bytes with any characters (truncate)
?	is used to replace 1 character with any character (truncate)
!	is used to replace 1 character with any letter (truncate)
#	is used to replace 1 character with any digit (truncate)
.	at end of a term to ignore truncation when automatic truncation is enabled (exact search)
:	is used to separate column name from search term
"	is used to mark string-search and to distinguish between a term and an operator (AND(F), OR(F), NOT(F) and XOR(F))
.#@	is used to identify column names in similarity search
.#$	is used to identify arguments in numeric similarity search

(is used to start a query enclosed within parentheses
.. is used to mark a closed interval
<[=] is used to mark an open interval
>[=] is used to mark an open interval

Searching a Boolware Index

Below will follow examples of both simple and more complex queries. Moreover a description of how to narrow and how to broaden the result set will be exemplified.

There are no restrictions how to combine the different queries.

Example

In a huge database containing millions of articles from several newspapers you want to find articles about specified search terms. The database contains the following Columns: *Article no.*, *Magazine, Publ. date*, *Author*, *Category*, *Size*, and *Text*. In the Column *Article no.* a unique identification (*Primary Key*) of the article is stored in the form of numeric data. The Column Magazine holds the name of the newspaper the article was published in. In the Column *Publ. date* the publication date - in the form yyyymmdd - is stored, and the name of the author is saved in the *Author* Column. The Column *Category* holds what type of article: sports, politics, culture etc. it is. The *Size* Column simply indicates the number of words contained in the article and finally the text of the article is stored in the Column *Text*.

Simple query

To find articles about **cars** in the Text Column just submit the following query:
`FIND Text:cars.`

Lets say you get 12.000 articles that contains the term **cars**.

When you examine the result you will probably see that you miss a lot of articles; you will only get the articles containing the term **cars** (in plural) and not the articles containing the term **car**. By slightly changing the query to: `FIND Text:car*`
you will get articles containing terms starting with **car** and thus both **car** and **cars**. Now you perhaps will get 97.000 articles, but of course this is far to much to browse through and you want to refine you query by just asking for **yellow cars**.

To achieve this you just add another query to the previous result: `AND Text:yellow*`
and the result will just consist of articles containing both the terms **car**/**cars** and **yellow**/**yellower**/**yellowest**. Perhaps the number of found articles at this point is reduced to 950, but when browsing through the articles you find articles containing your specified terms (**car*** and **yellow***)but not in the desired context; you find articles describing **yellow** houses and blue **cars** which of course fulfils your request, but it is not what you want.

Proximity Search

By using the sub-command near you could specify that you want your search terms to appear together in the articles. Just submit the following query: `FIND Text:near(yellow* car*)`
and your answer will perhaps be 130 articles describing yellow cars. You could specify up to 20 terms within the parentheses and of course you could specify the "gap" between the specified terms and if they must appear in specified order or not.

For example `FIND Text:near(car yellow big ford, 10)`
will give you all articles where the specified terms are within 10 words. More specific you could write `FIND Text:near(Harold Anderson, 1, 1)` and you will only get articles where **Harold Anderson** appears together (adjacent) and in this way simulate a string search without having to index the Column with the attribute string.

It is strongly recommended to avoid "truncated" terms in the near command. First of all you could get a not wanted result (many different combinations you are not interested in) and secondly it will take a lot of system resources.

Example:
`FIND Text:near(an* se*)`
will give you a very great number of hits as **an*** will generate thousands of terms that should be combined with many thousands of terms generated by **se***.

There is a special type of proximity command, *xnear*, that could be used when you want to exclude records that satisfies a normal proximity search. The syntax is the same as for the near sub-command but it contains two groups of terms separated by semicolon (;) and a blank. The terms before the semicolon are terms that must appear in the records, while the terms after the semicolon must not appear in connection with the terms before the semicolon. If you specify the following command:
`FIND Text:xnear(Ford Inc; Truck* Bus*, 6)`
you will get all records containing the company **Ford Inc** within 6 words. If, however, any of the terms **Truck**, **Trucks**, **Bus**, **Busses** etc appears within 6 terms this record should **not** be included in the result. If the terms Ford Inc appear only once without any of the terms after the semicolon are found within 6 words the record will be included in the result.

Example1:
This record will **not** be included in the result:
Record1:
text ... Ford Inc Trucks ... text

This record will be included in the result:
Record 2:
text ... Ford Inc Trucks ... text
text ..... text
text ... Ford Inc ... text
text ..... text
text ... Ford Inc Busses ... text

In Record1 the **Ford Inc** only appears once and the "exclude" term Trucks is found within 6 terms which will exclude this term from the result.

In Record2 the **Ford Inc** appears three times. In the first and third place the **Ford Inc** is followed by one of the "exclude" terms **Trucks** and **Busses** respectively. This fact should exclude the record from the result...**but** as the **Ford Inc** appears once without any of the "exclude" terms the record will be included in the result.

Example2:
Another example is when you want to find all cars of a certain brand and model. Let say that you are searching for all BMW and a model of 318 i, 318 ci, 318 is, 318 SALOON, 318 TDS etc.; in this case the 318 must be followed by **letters**.

```
FIND Text:near(bmw 318 !*,1,1) OR Text:near(bmw 318!*,1,1)
```
will give you the above models even if they are written as: BMW 318i, BMW 318ci, 318is, 318SALOON, 318TDS.

Another way to specify this is to perform the following command using xnear:
```
FIND Text:xnear(bmw 318; #*,1,1) OR Text:near(bmw 318!*,1,1)
```

Boolean Operators

You could very easily reduce the number of articles by adding more criteria like:
`FIND Text:near(yellow* car*) AND [Publ. date]:>=20000101 AND (Newspaper:Motor OR Newspaper:RoadRacing) NOT Category:culture`

This will mean that you get all articles about yellow cars published in this century and only from the newspapers Motor and RoadRacing and it must not be about culture.

This could be - as in this example - expressed in one query, but of course you could do it step by step. To do it step by step is more convenient as you could decide after each step if you want to browse the result set.

Efficient Or-search

If you want to combine many terms in an Or-search Boolware provides a special sub-command, **orsearch**. In stead of specifying all terms with the operator OR you could after this sub-command specify all terms within parentheses or specify a file which contains all terms to be or:ed. This could be convenient for the user if the search terms are exported to a file moreover it is more efficient in the Boolware system.

The result from the **orsearch** could be combined in the usual way by using the boolean commands: FIND, AND, OR and XOR.

Synonyms and Thesaurus

When searching unknown databases it could be hard to know what vocabulary the different authors are using and you could miss a lot of information by using the wrong search terms. One way to get around this problem is of course to use the Boolean OR operator between different terms. This is, however, a very tedious way and still you could miss some terms.

A more convenient way is to use the thesaurus and the synonym functionality. In the Boolware system you could - at any time without re-loading the database - change the synonym and the thesaurus files by adding and deleting terms. A detailed description of how to manipulate these files could be found in Chapter 10, Synonyms and Thesauri.

When using the synonym (*syn*) and the thesaurus (*thes*) sub-commands you could at every moment determine if you want a precise query or if you want to search using all synonyms. If you have specified the sub-command *syn* or *thes* all synonyms and children will be included in the query, while omitting them only searches for the specified term.

For example if a term in the synonym file has the following appearance *have* (*has*, *had*, *own*, *owned*) and your query is: `FIND Text:syn(have)` all articles that contain any of the terms **have**, **has**, **had**, **own** and **owned** will be part of the result set. Of course the result will be the same regardless of which of the five terms you use in the query. If you on the other hand just want articles where the word **own** appears you should specify your query like this: `FIND Text:own`.

It is also possible to specify a synonym that contains more than one word. When searching this synonym must be enclosed within quotation marks: FIND text:syn("new york"). In the synonym file you have - for instance - specified the following synonym: newyork(new york, new-york). In this case the search will be performed on all three terms: new york, newyork and new-york with an or-command; regardless which synonym has been specified.

Wild cards, Left Truncation and within word

If you are searching for a term like **car** you most likely also want all articles that contains the term **cars** as mentioned in the beginning of this example. One way to get around this is to use the wild card * which replaces 0 - 126 bytes of any type. If you just want to replace one character of any type you use the wild card **?**. The wild card **!** will replace one character of type letter, while # will replace one character of type digit

Those two wild cards could be used any number of times and in any combinations at any time:
```
FIND m*t?r
```
this query will find all articles that contains any of the words: **minister**, **monster**, **mister**, **master**, **m10t7r**, **mentor** etc. If you replace the **?** by **!** the term **m107r** will be excluded. If you replace the **?** by **!** the term **m107r** will be only approved term.

There is a special case when only the last part of a search term is of interest e.g. when searching for any kind of **balls** you could specify:
```
FIND *ball
```
and you will get all types of balls; **baseball**, **basketball**, **football**, **tennisball**, **handball** etc. This is usually called *left truncation* and is in most systems a very time-consuming operation as the entire index has to be processed. In the Boolware system you could specify any Column to be prepared for such an operation and thus be able to get the correct answer without processing the entire index. When using *left truncation* the search term must not have any wild card at the end. The *left truncation* indexing method is optional as the size of the index will grow with around 30%.

By using the *Within word* indexing you could use the Wild cards both at the beginning and the end of a search term and these kind of queries will not affect the performance.

This is an option as the index for the current column, in average, will be twice to six times as big as when not using this method.

Example:
```
FIND CompanyName:*compute*
```
will generate hit on the company name "Science**computers** Ltd"

Stemming

By using the wild cards (*****, **?**, **!** and **#**) you could very easily get variations of the very same word, but still there are cases when it is not enough. If you are searching for **bad** you would probably want all articles containing the words **worse** and **worst** as well.

There is a sub-command, stem, which takes care of these irregularities:
```
FIND stem(worse)
```
will in this case also find articles containing the words **bad** and **worst**. The stemming function is of course language dependent.

At a first glance this function seems to be the same as synonyms. However, some differences exist: all synonyms must be specified manually, while stemming automatically extracts the "common case" from the different inflectional forms of each word. When searching there is yet a difference: searching synonyms means that all synonyms are OR:ed together while stemming just searches for the stemmed term. The size of the index is also much smaller when using stemming (many different words will be transformed to one term). Finally when using synonyms you do not need to rebuild the index when changing the synonyms, while all changes in the stem file will cause a rebuild of the index.

Phonetic Search

A query normally tries to find the specified search terms in the articles by how the word is spelled. This is fully adequate in most cases, but not quite satisfactory in the Column *Author*.

You could spell the very same name in many different ways; many of them unknown for the one who searches the database.

There is a way to overcome this problem: *Phonetic search*, which rather searches for the sound than the spelling. If the Column is marked for the sound index type you could specify your query like this:

```
FIND Author:sound(Carlson)
```

and you will get all authors like Carlson (of course), Carlsson, Karlsson, Karlson, Carlzon, Karlzon etc.

It seems that this function could be replaced by the synonym or stemming function, but there are some differences: the synonym function is a manual function and the stemming which is automatic could not form the terms regarding the pronunciation.

Interval Search

When dealing with numeric data it could be very useful to be able to specify different types of intervals: *Greater than*, *Greater than or equal to*, *Less than*, *Less than or equal to and From .. To*.

You could for example ask for all articles published during 1999 that contain between 100 and 200 words;

```
FIND [Publ. date]:1999* AND Size:100..200
```

Of course all intervals could be used for Columns containing non-numeric data as well but it is not that obvious.

E.g. get all articles from authors whose names are "greater than" Smith;

```
FIND Author:>Smith
```

Now you will get all articles written by authors named like: **Wayne**, **Xerxes**, **Taylor**, **Spalding**, **Twain**, **Wilson** etc.

When a Column has an index type of numeric all data will be normalized i.e. the terms will be stored by value (**2** is less than **100**). When the index type is other than numeric (text) numeric data will be stored as text (**2** is greater than **100**) as they are compared character by character from left to right and **2** is bigger than **1**. This could give peculiar result when using intervals e.g.

```
FIND Text:100..200
```

will include articles containing **11**. Moreover numeric data in a text Column are mostly meaningless to search for as it could refer to anything: an age, number of cars, size of a lorry etc.

Excluding Search

There are occasions when you want to find all records that are missing information in a certain Column. For example you want to find all records in a company database that do not contain any information in the *CompanyName* Column. The primary key for the table is: *CmpNo* and each company must have a *CmpNo*. One way to achieve this to perform the following commands:

```
FIND CmpNo:* NOT CompanyName:*
```

Another - more convenient way - is to use the noti sub-command:

```
FIND CompanyName: noti(*)
```

You could of course "exclude" other terms by using this sub-command:

```
FIND CompanyName:noti(johnson) and CompanyName:noti(smith)
```

will search all records that do not contain **Johnson** or **Smith** in the *CompanyName* Column. Sometimes the search terms could have been exported to a file. In this case you could specify the complete file name as a parameter to the *noti* sub-command:

```
FIND CompanyName:noti(@c:\boolware\testdatabase\not-terms.txt)
```

The character @, that must be specified without any blanks, tells the system that the argument is a file.

Similarity Search

Searching for similarity means searching for records that contain similar content to what you enter in the search query.

In Boolware's search language QL, the sub-command SIM is used for similarity search.

Example:

```
SIM(Liverpool wins the Champion's league. Happy supporters cheered when)
```

The similarity search compares the specified text in the search query with all records in the database and sorts them in descending similarity order, i.e. the record with the best similarity is presented first.

Any Boolean commands can precede a SIM command, but if you want to keep the similarity order after the SIM command, you cannot use the Boolean OR operator after a SIM command.

When similarity search is used, there will be times when a large number of records will not be very similar to the text specified in the query and should therefore not be include in the search result.

To filter out these records from the search results, Boolware uses a threshold value that is default 0.200. This means that all records that have similarity score less than the threshold value will be excluded from the search result. You can easily change this value by setting a different threshold value after the SIM command. The threshold value is a decimal number between 0 and 1.

Example:

```
SIM(Liverpool wins the Champion's league. Happy supporters cheered when),0.3
```

In the above example, we have changed the threshold value to 0.3, instead of using the default value (0.200).

In addition to the threshold value, you can also specify an "optimization value" after the SIM command. The optimization value lets you prioritize between speed and relevance. An "optimization value" close to zero means prioritizing speed while a value close to 1 means prioritizing relevance. The default value for the optimization value is 0.5 if it is not specified. If you want to specify an optimization value, you must also always specify a threshold value. Since the threshold and optimization value are highly database dependent, we recommend experimenting with these values to obtain the best balance between speed and relevance.

Example:

```
SIM(Liverpool wins the Champion's league. Happy supporters cheered
when),0.2,0.6
```

In the above example, we have changed the optimization value to 0.6. But in order to be able to specify an optimization value, we also always have to specify the threshold value, even if we want to use its default value.

A similarity search can be done at any time during a search session, but a similarity search is much more time-consuming than a regular Boolean search. The search vectors must be compared with every possible record in the current search volume, which can be all records in

the database table. One way to make the similarity search more efficient is if possible, to reduce the search volume with regular Boolean searches, before the similarity search is done.

Below are two examples, which give exactly the same search results, but differ greatly in execution time. In the examples, we assume that we have a database containing 3,000,000 technical articles from 1970 onwards and that we have found an article about "Similarity Search", and are now interested in finding similar articles that was published 2001.

Example 1:

If the search query is specified as follows:

```
FIND [Publ. date]:2001*
ANDF SIM(text…), 0.400, 0.7
```

then we obtain the following search results:

Hits	Hits per query	Query
94.000	94.000	FIND [Publ. date]:2001*
2.800	2.800	ANDF SIM(text…), 0.400, 0.7

In this case 94.000 articles will be compared against the article containing a description on "similarity search".

Example 2:

If we instead specify the query as follows:

```
FIND SIM(text…), 0.400, 0.7
ANDF [Publ. date]:2001*
```

then we will instead obtain the following search results:

Hits	Hits per query	Query
14.000	14.000	FIND SIM(text…), 0.400, 0.7
2.800	94.000	ANDF [Publ. date]:2001*

In this case, all 3,000,000 articles must be compared with the description in the similarity search query. Only 14,000 met the set criteria, but all 3,000,000 articles must be compared to obtain this result. The last AND (ANDF) removes all articles that were not published in 2001.

Numeric Similarity Search

Similarity search could also be specified for columns containing numeric information.

Beside the Column name you could specify three parameters: *value*, *tolerance* and *weight*. The parameters are separated by semicolon (;). All missing parameters except the last must be marked by a semicolon. *Value* is a base value or an interval (start..end). *Tolerance* is specified in percent; when the *value* is an interval tolerance is not needed. If no *tolerance* is specified and the value is not an interval, the tolerance will be set to 10%. The third parameter, *weight*, is used to increase the importance of the current column when making the final compare. The default value for *weight* - when not specified - is 1.

The system compares all records in the database to the given values. A similarity score is calculated for each specified column and the different scores are weighted together and the result is saved as the final score. The records with the highest score will be presented first.

It is possible to combine text ("normal" similarity search) with numeric information.

The numeric information is identified by a string:**.#$** which must appear at the beginning of a line.

Example:

```
FIND sim(text
<CRLF>.#$"Turnover":100;12
<CRLF>.#$"Employees":20..40;;4
<CRLF>.#$"Solidity":1000;20;2)
```

First the similarity on the included text is calculated and after that the similarity on the numeric columns is calculated. The results are weighted together and the final similarity is calculated.

At least one of the following criteria must be fulfilled to approve a record for similarity ranking:

1. " Turnover" should be 100 +/- 12% (88 - 112). No weight is specified.
2. " Employees" should be between 20 and 40 and the weight is 4. As the value is an interval the tolerance is omitted (note the "extra" semicolon).
3. " Solidity" should be between 1000 +/- 20% (800 - 1200) and the weight is 2.

The maximal number of columns that could be used in Numeric Similarity Search is 100.

Important: After a similarity search the rank mode is automatically set to BSIMRANK.

Similarity Index

When indexing, all fields marked as "Similarity" are extracted by Boolware and used to calculate a vector for the record at indexing time. Each record is stored as a vector by Boolware. If the database table contains 100.000 tuples, Boolwares VSM-file will contain 100.000 vectors.

The system admin can choose whether to build VSM per field or on table level. This is done using a checkbox in the Manager indexing dialog: "Index similarity on field level". Checking this box instructs Boolware to build separate vectors for each field. Otherwise Boolware will build a single merged vector from all VSM fields.

To use VSM on field level has the following implications:

1. Applications are free to select which fields to use (rather than all) as long as they are marked as "Similarity" when indexing.
2. The vector reduction level can be adjusted per field.
3. The index will be somewhat larger, but the difference is small. The vector file (with the .vsm suffix) will grow about 5-10 percent, since vectors are stored separately, instead of merged for the whole record.
4. Applications must supply field names in the SIM query. Previously, this wasn't needed since all similarity info was collected for the whole record. The exception is if there is a single similarity field in the table, then that field doesn't have to be supplied.
5. Certain VSM searches on field level are somewhat slower, since fields are combined on the fly, but the difference is small. One or a few percent can be expected, depending on how many fields is part of the query.

Point four means that applications may have to be adjusted, to include field names in SIM queries. If not (and VSM is indexed per level), SIM queries without field names will produce an error. Here follows an example, to illustrate the SIM query format: the example uses a table containing sports articles, with a headline and a text.

First an example of the "old" syntax, without field names:

```
SIM(Liverpool wins the Champion's league.
Happy supporters cheered when …)
```

This is the "new" syntax, including field names:

```
SIM(.#@"Header":Liverpool wins the Champions league.
.#@"Text":Happy supporters cheered when …)
```

The difference is minor; each field begins on a new line, following the "magic" sequence .#@ + field name + colon. The order of the fields is insignificant.

The description of *Similarity*, Sorting and *Data source* is found under special tabs where you will be guided via different menus adapted to the function you want to use.

Global search

Sometimes it could be convenient to search more than one table using the same query. To do this you should use the GLOBAL command in Boolware. In the command you specify which tables to search. Of course the columns in the different tables must have the same name and attributes to be able to perform a global search. The most common way to use this command is to search through tables that have columns marked for "free text" searching.

Example:
Suppose there is a database that contains ten (10) tables. A column named Name appears in three of these tables. All ten tables contain columns marked for "free text" searching. In the first example you will find the name **Johnson** in the column *Name* for the three tables containing this column. The following query is performed:

```
GLOBAL(Table1, Table6, Table9) FIND Name:Johnson
```

Boolware will find 27 records in Table1, 97 records in Table6 and 567 records in Table9.

In the second example you want to search for **car** in all tables:

```
GLOBAL(*) FIND "":car
```

If * is specified in the GLOBAL Command all tables will be searched.

The command TABLES will automatically be used in the sub commands: *saved Set*, *saved Query* and *saved Result*. When saving any of these objects the tables used when searching will be saved along with the object. When using the saved object in a query you could specify which tables should be active by specifying TABLES.

Example:
The above query for **Johnson** in the column Name the following tables are used: Table1, Table6 and Table9. If you save this as *saved Query* "**Global Query1**" you could use it later when searching only Table1 and Table9:

```
TABLES(Table1, Table9)Find query("Global Query1")
```

regardless that the saved Query also contains Table6; you always determine which tables should be searched when specifying the query by using the command TABLES (it is **not** determined by the tables saved).

Related search (Join)

In a RDBMS there are usually relations between the tables included in the database; you could for example query one table and get the result from another table (join). To get the result in the desired table there is a possibility that you have to pass via several tables; there is a "path" from a table to another, which is defined by primary keys or foreign keys.

In Boolware you could use the command RELATE to specify which table you want the result in and by using the "normal" boolean queries to search in the other tables. Boolware will find the "best" path between the table(s) you are searching and the table that should contain the final result. All queries in the intermediate tables are performed automatically by Boolware.

If the "path" is established in the datasource Boolware uses that path, else you could specify a "path" along with the query in a RELATE command. In this latter case the specified "path" must

of course describe an existing relation between the tables. You use the sub-command *rpath*(t1.c1, t2.c2 .. tN.cN) to describe the path. The sub-command contains a number of tables and columns separated by a dot (.) that are related. The sub-command is written directly after the command for the current query. See example below.

Of course it is the responsibility of the user to guarantee that a relation really exists between the specified tables. If an invalid relation is specified Boolware will generate an error message.

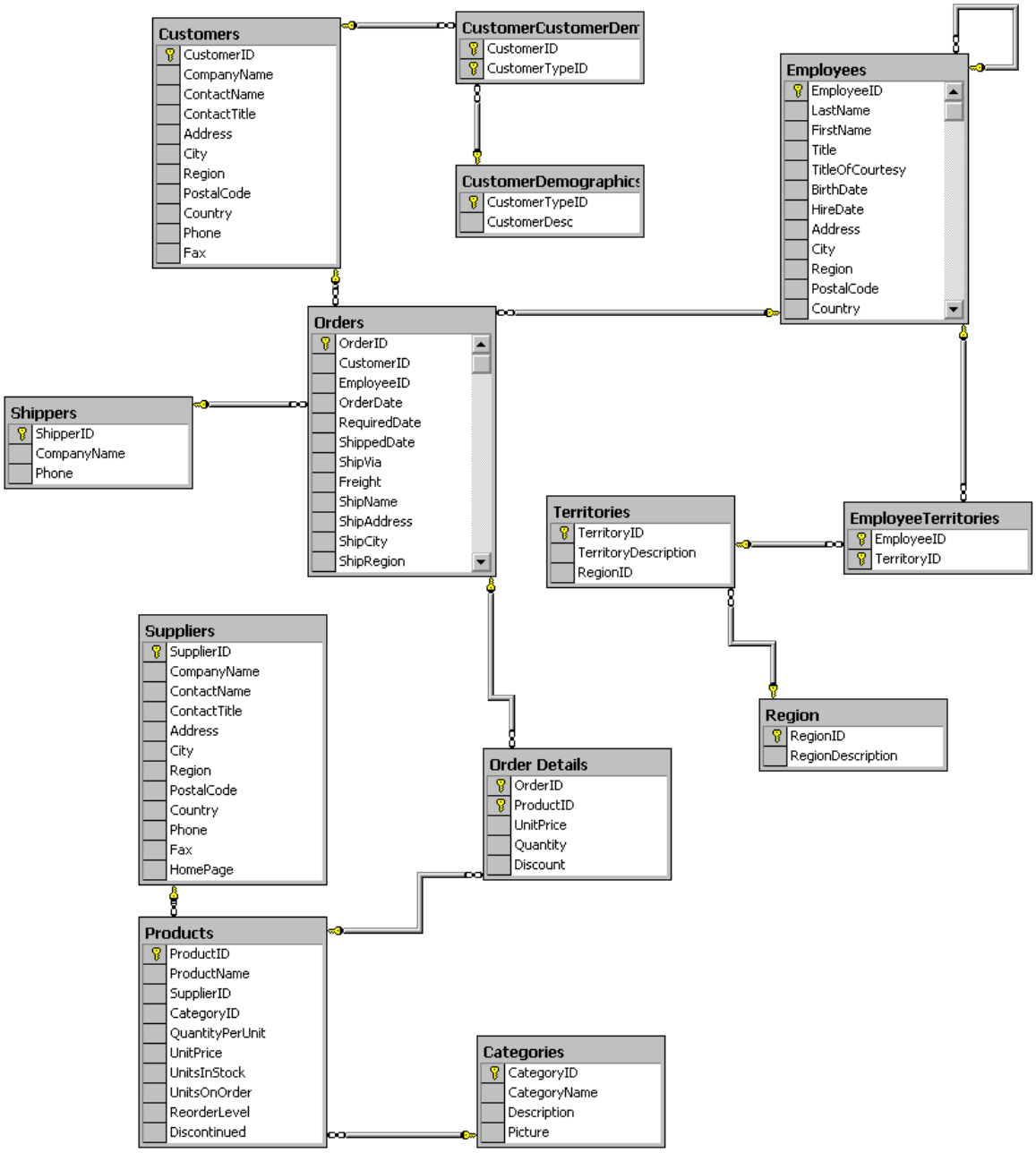
A query in Boolware is **always** within **one** Table. This indicates that you only could specify one Table within each Command. When using Join you always want to search between different Tables but the above rule still applies; each Command could only query one Table.

To get around this problem you could use the special Operators, ANDF, ORF, NOTF and XORF, which indicates a new Command. The difference between a Command an Operator in Boolware is that the Operator continues within the current Command, while the Command ends the previous Command and starts a new Command. In the Query History each Command will generate a result item, while an Operator is contained in the result item of its Command (see **Query History** below). By using the above special Operators you could search in several Tables as long as each Command (*F Operator) only searches one Table.

The easiest way to describe the join function in Boolware is by the below example:

Example:
Assume a database (NorthWind test database from Microsoft SQL Server) that contains 13 tables: Customers, CustomerDemographics, CustomersCustomerDemographics, Employees, Employee Territories, Order Details, Orders, Products, Categories, Region, Shippers, Suppliers and Territories. Between these tables there are different relations, but in our example we just look at the relation between the tables **Products** and **Customers**.

Below you will find a map describing the relations between the tables in the NorthWind database:



Below is a pictorial description of the relations between the tables contained in the "path" between Products and Customers. The column names in italic are primary keys or foreign keys which establish the relation between the tables (the path).

Table **Products**:
ProductID
ProductName
SupplierID
CategoryID
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued

Table **Order Details**:
OrderID
ProductID
UnitPrice
Quantity
Discount

Table **Orders**:	Table **Customers**:
OrderID	*CustomerID*
CustomerID	CompanyName
EmployeeID	ContactName
OrderDate	ContactTitle
RequiredDate	Address
ShipVia	City
Freight	Region
ShipName	PostalCode
ShipAddress	Country
ShipCity	Phone
ShipRegion	Fax
ShipPostalCode	
ShipCountry	

The result from the table **Products** is used to get a new result in the table **Order Details** using the column name *ProductID*. The result from the table **Order Details** is used to get a new result in the table **Orders** using the column name *OrderID*. Finally, the result from the table **Orders** is used to set the final result in the table **Customers** using the column name *CustomerID*.

If you want to find out which customers (Customs) in the US that have ordered the product: "Uncle Bob's Organic Dried Pears" or the product "Alice Mutton". The following query could be specified in Boolware:

```
RELATE(Customers)
FIND  Customers.Country:usa
ANDF  Products.ProductName:(uncle bob organic dried pears)  OR  (alice Mutton)
```

The following Query History will be presented for table Customers:

Hits	**Hits per query**	**Query**
13	13	FIND Customers.Country:usa
6	37	ANDF Products.ProductName:(uncle...) OR (alice...)

In this case the first query:
`FIND Customers.Country:usa generated 13 hits in the` **`Customers`** `table. The second query:`
`ANDF Products.ProductName:(uncle...)  OR  (alice...)`
generates 2 hits in the table **Products**, which will be transformed into 37 hits in the table **Customers** and the result after a boolean AND (ANDF) in the table **Customers** is 6.

In the FIND command the **Hits** is the same as the **Hits per query** result because the query is performed in the destination table.

In the ANDF command the **Hits** is the result after the specified ANDF command after the transforming to the **Customers** table and the **Hits per query** is the result from the transforming of the 2 hits in the table **Products** to the table **Customers**.

Note that the Command FIND searches the Table **Customers**, while the Command ANDF searches the Table **Products**. You could **not** use the Operator AND in this case as the search takes place in different Tables (must use ANDF).

If the above relation is not established in the datasource but still exists you will get exactly the same result by writing:

```
RELATE(Customers)
FIND  Customers.Country:usa
ANDF  rpath(Products.ProductID, [Order Details].ProductID,
           [Order Details].OrderID, Orders.OrderID,
           Orders.CustomerID, Customers.CustomerID)
           Products.ProductName:(uncle bob organic dried pears)
          OR (alice mutton)
```

In this case you have "manually" described the "path" between the query table (Products) and the destination table (Customers). It is of course the users responsibility that the specified "path" exists.

You could re-fine your search by specifying that you are only interested in customers with a postal code starting with 8.

```
RELATE(Customers)
ANDF Customers.PostalCode:8*
```

This command will add another record to the Query History which will look like this after this command:

Hits	Hits per query	Query
13	13	FIND Customers.Country:usa
6	37	ANDF Products.ProductName:(uncle...) OR (alice...)
3	6	ANDF Customers.PostalCode:8*

After this query you see that you have made a mistake; it was customers with a postal code starting with 9 you were interesting in. The following two commands will correct the mistake:

```
BACK
```
backs one step in the Query History:

Hits	Hits per query	Query
13	13	FIND Customers.Country:usa
6	37	ANDF Products.ProductName:(uncle...) OR (alice...)

after this you specify the correct query:

```
ANDF Customers.PostalCode:9*
```
which will give the following result:

Hits	Hits per query	Query
13	13	FIND Customers.Country:usa
6	37	ANDF Products.ProductName:(uncle...) OR (alice...)
3	10	ANDF Customers.PostalCode:9*

The function RELATE could also be used when you have an old result in one table and want to transform the result into another table.

Example:
You have an old result in the table Products which should be transformed into table Customers. The command to do this looks like this:

```
RELATE(Customers, Products)
```

The old result in table Products is in this example 3 and the result after transformation in the table Customers will be 54.

The Query History for table Customers is:

Hits	Hits per query	Query
54	54	Result from table - Products - transformed into table – Customers -

To be able to see the Query History for table **Products** and all tables in between you must specify the Query History for the wanted table (the Query History only operates on one table).

NOTE!

The time-consuming part of a relate query is the transforming of the result from the query table to the destination table. If the relation includes several tables this operation must be repeated for each table in the relation.

Therefore it is important to specify the query in such way that you avoid transforming when not needed.

The number of hits that should be transformed is also of great importance when it comes to efficiency; the fewer hits to transform the faster is the transform operation.

The below two examples will give the same result but will differ when it comes to resources.

Example 1:

```
RELATE(Customers)
FIND  Customers.Country:usa
ANDF  Products.ProductName:uncle bob organic dried pears
```

will give the result:

Hits	Hits per query	Query
13	13	FIND Customers.Country:usa
3	20	ANDF Products.ProductName:(uncle...)

Only one transform is performed when the result from the query in **Products** (1 hit) is transformed to **Customers** (20 hits). The final result after AND with Country:USA will be 3 hits.

Example 2:

```
RELATE(Customers)
FIND   Customers.Country:usa
ANDF  Products.ProductName:uncle
ANDF  Products.ProductName:bob
ANDF  Products.ProductName:organic
ANDF  Products.ProductName:dried
ANDF  Products.ProductName:pears
```

will give the result:

Hits	Hits per query	Query
13	13	FIND Customers.Country:usa
3	20	ANDF Products.ProductName:uncle
3	20	ANDF Products.ProductName:bob
3	20	ANDF Products.ProductName:organic
3	40	ANDF Products.ProductName:dried
3	20	ANDF Products.ProductName:pears

Five transforms are performed and one of them, *dried*, generates two hits which will be transformed to 40 hits in the **Customers** table. The final result after AND with Country:USA will be 3 hits.

NOTE!
As the above two ways of specifying a related query is handled in different ways in Boolware they could come up with different results.

Two tables - Table1 and Table2 - which are related by a zip code.

Example:
Query1:

```
RELATE(Table1)
FIND Table2.Code1:51 AND Table2.Code2:1003
```

and

Query2:

```
RELATE(Table1)
FIND   Table2.Code1:51
ANDF   Table2.Code2:1003
```

could give different results if the following data appear in the columns Code1 and Code2 in Table2.

Table2

Row	*Code1*	*Code2*	*Zip code*	*Corresponding row in Table1*
171	51	1005	10050	This row points to row 2003 in Table1
201	51	1003	10070	This row points to row 101 in Table1
294	51	2007	20083	This row points to row 81 in Table1
687	104	1003	17114	This row points to row 81 in Table1
987	89	1003	17114	This row points to row 3012 in Table1
1023	51	1003	21634	This row points to row 4012 in Table1

In the first query the only rows that holds both Code1:51 and Code2:1003 are row 201 and row 1023. The corresponding rows in Table1 - after the transforming - are row **101** and row **4012**. The final result in Table1 is **2**.

In the second query the following rows satisfy Code: 51: 171, 201, 294 and 1023. Those rows are transformed to Table1 giving the following rows: 2003, 101, 81, and 4012.
The rows 201, 687, 987 and 1023 satisfy the query Code2:1003. Those rows are transformed to Table1 giving the following rows: 101, 81, 3012 and 4012. This result is AND:ed to the earlier result in Table1. The following rows are the final result in Table1: **101**, **81** and **4012**.
The final result in Table1 is **3**.

Important: In the current version of the Boolware server multiple columns to form a Foreign Key is not taken care of. This means that if a relation between two tables (the Foreign Key) is built up by more than one column Boolware can't handle this.

A way to get around this problem is to create an extra column containing unique information as a Foreign Key.

Important: In the current version of the Boolware server there is no support for presenting information from several tables in one command.

The result from each table is presented separately and only columns within this table will be presented.

Self-join

A self-join is when a table references data in itself.

In the example below we search for three persons, using the ORSEARCH search function and the person's ID and then performing a self-join, using the RELEATE and RPATH command. With the RELATE command we tell that we want the result from the PERSONS table and with the RPATH command we tell that the field HUSHALL_ID should be used as matching field.

NOTE! The matching fields that you specify with the RPATH command must be string indexed.

If all found persons belong to different households and these households consist of two people then this search will return six people.

Example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SoftboolXML_requests>
  <SoftboolXML_request type="execute">
    <open_session queryhistory="1"/>
```

```
    <database name="EXAMPLE"/>
    <table name="PERSONS"/>
    <execute>
    RELATE(PERSONS)
    FIND RPATH(PERSONS.HUSHALL_ID, PERSONS.HUSHALL_ID)
         PERSONS.PERSON_ID:orsearch("429A4" "C97BB" "1342A" )
    </execute>
    <response >
      <records from="1" to="10" maxchars="0">
        <field name="PERSON_ID"/>
        <field name="HUSHALL_ID"/>
      </records>
    </response>
  </SoftboolXML_request>
</SoftboolXML_requests>
```

To perform the same type of search in a Boolware flow, use the following syntax:

```
<search cmd="RELATE(PERSON-KEY)
             FIND RPATH(PERSON-KEY.HUSHALL_ID, PERSON-KEY.HUSHALL_ID)
             PERSON-KEY.PERSON_ID:
             orsearch('429A4C97BB1342ACAF26CD55D6A81482')"/>
```

EXISTS search

If you want to restrict a search result based on whether data exists, the subcommode **EXISTS** can be used. The command consists of two parts, first a search part (search criteria) to be specified within parentheses and a condition part, specified after the search part.

E.g. EXISTS((value:*) groupid:$groupid NOT $id)

The search part specifies the search criteria that the **EXISTS** expression should use with AND/OR/NOT/XOR between search criteria. In the condition section, the fields you want in your condition and placeholder ($field name) are specified as the value of the condition. Data from the total result before the exist command will be used and replaced in the placeholders in the EXISTS query. The AND/OR/NOT/XOR operators can be used between the conditions.

If you want to search in another table in the search part of an EXISTS query, all field names must be prefixed with the table name. The fields and placeholders in the condition section must also be prefixed with table names. For placeholders, the table name is always the source table, ie. the table where the total result is located.

E.g. Find all men in Hägersten born between 1962-1963 who live with a woman born between 1973-1980:

```
FIND gender:"M" AND locality:"Hägersten" AND BirthDate:1962..1963
ANDF EXISTS((gender:"F" AND BirthDate:1973..1980) AND householdid:$householdid
            NOT personid:$personid)
```

E.g. Same question as above but with a search in another table:

```
FIND gender:"M" AND locality:"Hägersten" AND BirthDate:1962..1963
ANDF EXISTS(("personsmall".gender:"F" AND "personsmall".BirthDate:1973..1980)
            AND "personsmall".householdid:$person.householdid
            NOT "personsmall".personid:$person.personid)
```

E.g. First find all men in Stockholm born between 1962-1963 who live with a woman born between 1973-1980. Then add the women who live with these men:

```
FIND gender:"M" AND locality:"stockholm" AND BirthDate:1962..1963
ANDF EXISTS((gender:"F" AND BirthDate:1973..1980) AND
            householdid:$householdid NOT personid:$personid)
ORF EXISTS((gender:"F" AND BirthDate:1973..1980) AND householdid:$householdid
            NOT personid:$personid)
```

Note! For optimal performance please turn on "Store column data" on fields used in the EXISTS-command and also activate preread (se example below) with either Boolware Manager or bwc. Schedule also **preread** in Boolware Manager with a couple of hours interval (eg 2-3 hours).

E.g.

```
preread dsn persons table person files gender.idx, gender.ref, locality.idx,
locality.ref, birthdate.idx, birthdate.ref, householdid.idx, householdid.ref,
personid.idx, personid.ref, *.tab, *.data on
```

Browse the Query History

In some case you need several queries to get the wanted result. In these cases it could happen that you search too "far" and want to back one or more queries to continue your searching from this point.

In Boolware a search session is defined by two FIND Commands; all queries starting with a FIND up to the next FIND constitute a search session. During such a search session all queries and the associated results are saved in a Query History.

If you find - after many queries - that you want to back and continue the searching from an earlier point you should use the BACK Command. Of course there is also a FORWARD Command that is used to step forward in the Query History.

You could find some examples in the section **Query History** below.

Ranking the result from a Query

As the result set after a query could be large it is very important to get the most interesting articles at the top. In the Boolware system you could rank the result in the following ways: by search terms (occurrence and frequency), similarity and sorting on different Columns.

A new query will always reset the rank mode to *norank* (no ranking) except when a query on weighted search terms (FIND car/**9**/) is performed or a similarity search is performed; in these cases the rank mode will be automatically set to *weightedfrequency* (rank on weighted frequency) and *similrarity* (rank on similarity) respectively.

The setting of rank mode could only be done **after** a query has been performed to be valid (if not a query on weighted search terms or a similarity query). There must be a search result. When ranking on occurrence and frequency the query must contain search terms in columns marked for Ranking as well. If these criteria are not met the rank mode will be reset to *norank* (no ranking).

The type of ranking is determined by the commands RANK and RANKTERMS **before** displaying the result. A description of the rank commands could be found in "**Programmer's Guide**".

A special "ranking" could be to fetch the found records in randomly order. This could be done by setting the attribute *randomfetch*="1" in the current XML response.

Rank by Search Terms

Ranking is a method to be used for presentation of search result in a specific sort order (occurrence or frequency order based upon given search terms)

When ranking method *occurrence* is selected, tuples containing the most occurrences of the given search terms will be presented first.

When ranking method *frequency* is selected, a calculated frequency value (the total number of occurrences of search terms divided by the total number of words in the column) between 0 and 1 will be shown for each tuple and tuples with the highest value will be presented first.

When using the Boolware Query Language (QL) you could affect the order by giving a search term a weight.

Example:
FIND text:cars/9/ AND blue AND american/4/

When the number of occurrences should be calculated, the number of found *cars* is multiplied by 9, the number of found *blue* by 1 and the number of found *American* by 4.

In this case, tuples containing many *cars* and *Americans* in the given column will be favored when presenting the result.

NOTE in this case the rank mode will automatically be set to weightedfrequecy (rank on weighted frequency).

Important: The rank mode could only be set **after** a query if not query on weighted search terms or similarity search. If a rank on *occurrence* or *frequency* the query must contain search terms in rankable columns as well. If not both these criteria are fulfilled the rank mode is reset to *norank* (no ranking).

Important: When performing a query on weighted terms (FIND car/9/) the rank mode will automatically be set to *weightedfrequency* (rank on weighted frequency).

Important: When ranking all the specified terms - that belongs to columns indexed for ranking - and is part of the current search session will be used. The occurrence of each of these terms will be counted for all retrieved records.

If you are using truncation when searching several search terms will be generated for each specified term. You should be very careful with "hard" truncations in combination with ranking; e.g. **a*** will generate several thousands of search terms (all words starting with **a**). All these generated terms must be located and counted for each retrieved record which could take a very long time.

Our recommendation is to avoid "hard" truncations when ranking.

Statistics on ranked Search Terms

After a ranking on specified Search Terms you could get statistics on all ranked Search Terms. Note a ranking on a specified Search Term must precede this command. See examples in "**Programmer's Guide**".

Rank by Similarity

After a Similarity Search all records will automatically get a score count which reflects the similarity to the supplied text.

The Euclidean distance of the supplied text and the Euclidean distance from the current article are compared and the square root is taken from the result to get a score count in the interval 0 to 1.

The records are sorted on the score count in descending order leaving the most similar article at the top.

Important: After a Similarity search the rank mode is automatically set to *similarity*.

Rank by Sort

Another way to rank the records in a result set is to sort the records on one or more Columns.

There are four different sort algorithms within the Boolware system:

1) Pre-sorted indexes
2) The Boolware Index Sort (zoom sort)
3) Sorting after reading necessary information from the Boolware data file (column data stored by Boolware).
4) Sorting after reading necessary information from the Data Source (RDBMS).

Boolware tries the algorithms in the above specified order.

The Boolware Sort by pre-sorted indexes

If there is only **one** column to sort and this column is indexed as *String* or *Numeric* and the column is pre-sorted this sort algorithm might be used depending on the below.

If the number of hits to be sorted, is less than one million, this sort algorithm will be used.

If the number of hits to be sorted, is more than one million, but less than the value specified in the online sort setting parameter, this sort algorithm will be used.

The pre-sort can not be used if the column, to be sorted, has been online-updated.

All tuples must be handled before the sort is completed.

NOTE
When using the Boolware Sort by pre-sorted indexes, records that do not contain any data (null values) in the Column sorted on will be placed last regardless of the sort order (Ascending/Descending).

The Boolware Index Sort (incremental sort)

If the very **first** Column to sort is indexed as *String* or *Numeric* and the number of hits is more than the value specified in the online sort setting parameter, the Boolware incremental Index Sort is chosen.

As indicated by the name the Boolware Index is used when sorting. The strings/numeric in the Index is already sorted in alphanumeric order and Boolware just has to read the current Index to get the records in the proper sort order.

Boolware is taking advantage of the sorted Index and can thus use an incremental sorting strategy; just give the requested number of records in sorted order. If the application does not specify the number of records to be sorted the Boolware system will sort 50 records by default.

When more records (after the first 50) are requested, Boolware will "sort" the next requested number of records before presenting them. This means that Boolware only sorts the records at request and - of course - guarantees the proper sort order.

By specifying parameters directly after the Column name you could determine the sort order: Asc for ascending (default) and Desc for descending. The parameter emptydata = first will place

records not containing data in that column first while emptydata = last will place them last (default).

Example 1:
After a search you have got 2.345.087 records that fulfil your query. You want to sort on the numeric Column *Profit* and you want records with the highest profit first. In this case you only want to see the 20 records with the highest profits.

The Sort Command:
```
Profit Desc:20
```
Only the 20 first records will be sorted before you will get your requested records quickly regardless of the size of the .IDX file.

When you fetch more records Boolware only sorts as many records as needed to present the requested number of records in the proper order.

A drawback using the Boolware incremental Index Sort is when the Index is very large (contains several million unique terms) and the number of tuples (hits) to sort is small and the terms to sort is in the end of the Index (when ascending sort) or in the beginning of the Index (when descending sort); you have to read most of the Index to get the required tuples.

Example 2:
You have a database containing 30.000.000 companies. One of the Columns, *phoneNo*, is indexed as string and thus can use the Boolware incremental Index Sort. Most of the phone numbers are unique so the index will contain close to 30.000.000 unique terms. If your search result is 10 hits and you want to sort on the phone number and the phone numbers in the retrieved tuples are at the bottom of the index (99999...) Boolware has to read all 30.000.000 terms to get the tuples in proper order. It would probably be much faster to read the records from either the Boolware data file or the Data Source and sort on the content in the phone number Column.

There is a way to tell Boolware to read the records from the Boolware data file (if any) or the Data Source and sort (read chapter 3 or/and 4 below) on *phone number* rather than using the Boolware incremental Index Sort before presenting the tuples;
You simply specify a value in the online sort setting parameter for the current database and/or table. Boolware will automtically calculate a default value but this can be changed. Use the Boolware Manager and mark the current Database/Table/Column, right-click and choose the configure - sort online option to set the another value.

If the number of retrieved tuples after a query is less than the specified number in the online sort setting parameter the Boolware incremental Index Sort will not be used. The number that should be specified in the online sort setting parameter is not easy to calculate but must be tested to fit the current database/table and Data Source since it is very much related to the number of tuples in the current database/table.

Example 3:
You have a database containing 30.000.000 companies. After a query for companies within a special trade the number of found records is 738.976. You want to sort on three columns: *City*, *ZipCode* and *Street*. As the first column, *City*, is indexed string you could take advantage of the Boolware Index Sort.

You are only interested in the 20 first companies and the sort command could be as follows:

```
City asc, ZipCode asc, Street asc:20
```

Note that the number of records to sort each time - if other than the default value 50 - must be specified on the last column.

First 20 records will be sorted on *City* using the Boolware Index Sort. These 20 records will then be sorted using one of the other Boolware sort algorithms; e.g. if the two other column are stored in the Boolware data file the Boolware data file sort will be used.

Example 4:
Same as examle 3, but this time you want to place records not containing data in *City* first.

You are only interested in the 20 first companies and the sort command could be as follows:

```
City asc emptydata=first, ZipCode asc, Street asc:20
```

Note that the number of records to sort each time - if other than the default value 50 - must be specified on the last column.

First 20 records will be sorted on City using the Boolware Index Sort. These 20 records will then be sorted using one of the other Boolware sort algorithms; e.g. if the two other column are stored in the Boolware data file the Boolware data file sort will be used.

Read tuples from the Boolware Data file and sort

If the Boolware incremental Index Sort could not be used and there is a Boolware data file containing all Column data (to be sorted) Boolware extracts all necessary information from the Boolware data file and builds up a sort key corresponding to the specified columns in the sort command.

When all tuples have been read an internal sort algorithm is performed.

By specifying parameters directly after the Column name you could determine the sort order: Asc for ascending (default) and Desc for descending. The parameter emptydata = first will place records not containing data in that column first while emptydata = last will place them last (default).

Read tuples from the Data Source

If none of the above sort algorithms could be used Boolware extracts all necessary information from the retrieved tuples and builds up a sort key corresponding to the specified columns in the sort command.

When all tuples have been read an internal sort algorithm is performed.

This is by far the slowest algorithm to sort column data and it will only be chosen if no other of the above sort alternatives (algorithms) could be used.

By specifying parameters directly after the Column name you could determine the sort order: Asc for ascending (default) and Desc for descending. The parameter emptydata = first will place records not containing data in that column first while emptydata = last will place them last (default).

Statistics for a column

In databases containing much numeric information it could be interesting to get statistics for a column.

Suppose you have a large database containing information about all companies in Europe. The information in this database is of course: the name of the company, its identification number, address, board of directors, branch of trade etc. Other common information about companies is: number of employees, turnover, profit, solvency, liquidity etc.

By help of simple statistical functions in Boolware you could get the following statistics for a column containing numeric information: number of items in the statistics, the sum of all values, max value, min value, arithmetic mean value, median value, standard deviation, variance, the

most frequent value, number of items containing the most frequent value and upper and lower limit of chosen group. Group could be four-month period, quartile, quintile etc.

The above statistical values could be calculated for all numeric columns. You could choose to calculate the statistics on the complete database or on a result from a previous query.

In Boolware there are very efficient methods to extract statistical information from millions of companies and pass this information to an external program that could use the information for graphical presentation.

A description of the command could be found in "**Programmer's Guide**".

Calculation between columns

Like statistics could be interesting in databases containing much numeric information performing calculation between columns could be convenient.

By specifying a formula in a "calculating result column" you could get the result from the specified formula for each record in the result set. In the specified formula you could specify: column names, constants and other formulas.

Example:
In a database you want to perform the following arithmetic operations: turnover per employee, profit per employee, prognosis at a 20% improvement in sales. Use the calculated prognosis and calculate the new turnover per employee. Four formulas are required and you will get these four "columns" along with the other columns when displaying each record.

```
Turnover/Employee:    "Turnover" / "Employees"
Profit/ Employee:     "Profit" / "Employees"
Prognosis:            "Turnover" * 1.20
Turnover/Employee(P):"Prognosis" / "Employees"
```

When presenting records all "common" columns will be presented as usual in the result and in addition the four "calculating result columns" containing the calculated values will be presented.

Set Search

In Boolware you could make use of earlier performed queries. A common expression for this functionality in Boolware is: Set Search. Set Search contains a number of different functions: *Query History, saved Set*, *saved named Set*, *saved Query*, *saved Result*, *saved named Scratch*. *Saved named Scratch* are treated in the exactly same way at all levels. The functions: *saved Set*, *saved named Set*, *saved Query* and *saved Result* are saving on disk, while *Query History*, *saved named Scratch* only are handled in memory. For the time being, maximum 100 named scratches can be used per table in Boolware.

When should Set Search be used

As indicated by the name the above functions are connected to queries in Boolware. The functions are a flexible way to re-use earlier performed queries and their results. Moreover it should be possible to navigate between different commands within one search session. This means that you could go backwards or forward between the current commands from the last FIND command to take a new path in your searching.

There are different reasons to re-using queries: 1. the query is very complex and contains hundreds of commands, arguments and operators, 2. it is very time-consuming when executing,

3. you want so supply people with queries that gives relevant result, 4. there could be reasons to limit the access to a database for certain groups of sessions/users etc.

The *Query History*, the *saved Set*, the *saved named Set* and the *saved named Scratch* are always just for the current user, while the *saved Query* and the *saved Result* could be public for a group of sessions/users. The identification of a *saved Set*, *saved named Set*, *saved Query* and a *saved Result* is built up by the user identification and a given name which means that only the user that has saved the *Set*, *Query* or *Result* could get access to it. If however a *Query* or *Result* is saved as *public*, all sessions/users within that group will get access to the *Query* or *Result*. It is the application's responsibility to ensure that only authorized users get access to *saved Queries/Results* within a group. For example could the users within the sales department: **SalesDep**JohnSmith, **SalesDep**BobGates, **SalesDep**SteveGartner, **SalesDep**SueCurtis etc. be a group of user/session names called **SalesDep**, which will be used as *public* name when using public *saved Queries* and *saved Results*.

The characteristics for the different functions are not the same which indicates that you should use them for different purposes. *Saved Sets* and *saved Results* are always connected to one database and one or more tables within that database, while *saved Queries* could be used for any database and table. A *saved Set/saved named Set/saved named Scratch* is only alive during the current user session, while *saved Queries* and *saved Results* could be used until they are explicitly removed by the creator of the *saved Query* and the *saved Result* (*a saved Query* and a *saved Result* could only be removed by its creator). A *saved Set*, a *saved Result* and a *saved named Scratch* could not be used after an online-update (must be re-searched as the result could have been changed), while a *saved Query* always could be used as it is re-searched each time it is used.

Generally speaking you could say that a *saved Set/saved named Set/saved Result/saved named Scratch* is much faster than a *saved Query* as no searching is performed; you just get the current saved result. On the other hand a *saved Set* and a *saved Result* uses much more disk space as the result is saved along with the query string.

Important: The above functions should be used with care when search performance is important, as the *saved Query*, the *saved Result* and *the saved Set/saved named Set* stores information on disk.

When dealing with large result sets - many found records - the time to administrate *saved Result* and *saved Set/saved named Set* could be considerable.

The *Query History* and *saved named Scratch* are saved in memory and does not affect the performance noticeable.

Important: A *saved Query*, a *saved Result* and *a saved Set/saved named Set* contains all information needed in terms of column names and search words. When you combine these objects with new search terms is it very important that a new search term contains both a column name and a search word. If the column name is omitted the last used column name will be used. If the previous search term was a *saved Query*, a *saved Result* or a *saved Set/saved named Set* it could a completely different column name than you meant because the "last" column name is "hidden" in the search object.

How to use Set Search

The following commands are linked to the various functions in Set Search: *saveresult, savequery, savescratch*, *reviewset, reviewquery, reviewresult* och *deleteset, deletescratch, deleteresult, deletequery*. The API function *Execute* is used to perform these commands. When using these functions in a query session the following sub commands are used: *set*, *query*, *result* and *scratch(name)* respectively.

A major difference between the Execute commands and the QL commands is that the Execute commands uses a keyword (key=attribute), while the QL uses sub commands and the attributes enclosed within parentheses.

Below all the commands for the *saved Set/saved named Set*, *saved Query* and *saved Result* will be described.

The following differs between the two functions:
1. The name for the *saved named Set* is specified directly in the Query. When using the *saved Set* function the name is automatically generated. If a *saved named Set* is preceded by @@@ it will not be saved if the result is zero (no result).
2. The switch *setsearch* is not involved when using the *saved named Set* function.

Search History is, when enabled, is completely controlled by Boolware and you could only use the commands *back* and *forward* to navigate within the current query session. The APIs for the *Query History* are described in the "**Programmer's Guide**"

When referring to a *saved Set*, *saved named Set*, *saved Query* or *saved Result* you always use what has been specified in name when saving the object. When specifying the identification the normal Boolware rules are used; truncation is allowed and the name is not case sensitive. E.g. when reviewing *saved Queries* the following is valid: review query *name* **jOhN*1**, which means that all *saved Queries* starting with **john** and ending with **1** will be displayed.

Saved Set

The command SAVESET could not be done explicitly on a *saved Set*. If the switch *setsearch* is activated it will be automatically saved by Boolware and get an automatically generated name: S1, S2, S3... By specifying a colon (:) followed by a name directly on the command it will be regareded as a *saved named Set* and will be saved with the specified name.

```
FIND:"Search on London"  city:london AND profit:>100
```

If you only want to save the named set if the result is greater than 0 you write like this:

```
FIND:"@@@Search on London"  city:london AND profit:>100
```

The result will be saved as a saved named Set with the name "Search on London" and could be used in the ordinary way.

When using the saved Set/saved named Set in a query, a sub-command, set, should be used.

Example:
```
FIND set(S4) AND set("Search on London") NOT Text:culture
```

A boolean AND is performed between the *saved Set* **S4** and the saved *named S*et "**Search on London**" and after that a boolean NOT is performed with the term **culture** in the column *Text*. Note, that you always should specify the column name (*Text*: in this case) after you have used a *saved Set*, because the *saved Set* could contain other column names than what you expected.

Saved Query

When using the *saved Query* in a query, a sub-command, query, should be used.

Example:
```
FIND (query("Query 1") OR query(query4)) AND Text:sport
```

A boolean OR is performed between the *saved Query* **Query 1** and the *saved Query* **query4** will be performed and after that a boolean AND is performed using the term **sport** in the column

Text. Note, that you always should specify the column name (*Text*: in this case) after you have used a *saved Query*, because the *saved Query* could contain other column names than what you expected.

If you want to use a public *saved Query* it must be specified in the following way:

Example:
```
FIND query(public(SalesDep) "Query 1") AND Text:sport
```

A boolean AND is performed between the public *saved Query* **Query 1** belonging to the session-/username **SalesDep** and the term **sport** in column *Text*.

Saved Result

When using the *saved Result* in a query, a sub-command, result, should be used.

Example:
```
FIND result(Sportresult1) AND Text:basket
```

A boolean AND is performed between the *saved Result* **Sportresult1** and the term basket in the column *Text*. Note, that you always should specify the column name (*Text*: in this case) after you have used a *saved Result*, because the *saved Result* could contain other column names than what you expected.

If you want to use a public *saved Result* it must be specified in the following way:

Example:
```
FIND result(public(SalesDep) "Result 1") AND Text:sport
```

A boolean AND is performed between the public *saved Result* **Result 1** belonging to the session-/username **SalesDep** and the term **sport** in the column *Text*.

Saved named Scratch

The normal way of using a scratch result is:
1. Perform queries from a table
2. Save the current result in a named scratch result in current table
3. Perform more queries and use the saved named scratch result from current table
4. Delete the used named scratch result from table

To use this result specify the sub command *scratch(name)*.

```
FIND  scratch(culture) OR Text:sport
```

The FIND command will use the result saved in scratch named '*culture*'.
Between the *saved scratch(culture)* and the term **sport** in the column *Text* a logical OR will be performed. Note, that you always should specify the column name (*Text*: in this case) after you have used a *saved scratch*. The scratch sub command need no column name.

Functionality

Below is a description of the functionality on the different components belonging to Set Search.

As mention earlier the different functions should be used for different purposes. It is also important to note that the type of database could have a decisive influence on what function to choose; if a database is frequently online-updated *saved Query* is to prefer rather than *saved Result* as *saved Result* do not re-search but only uses the saved result. If, on the other hand, a

very complex query in a huge database takes long time to perform a *saved Result* is preferable to a *saved Query* as the *saved Query* has to re-search the complex query, while the *saved Result* just fetch the result.

As a *saved Set* only is alive during the current user session a *saved Query* or a *saved Result* is preferable if it should be used between user sessions.

Query History

In some case you need several queries to get the wanted result. In these cases it could happen that you search too "far" and want to back one or more queries to continue your searching from this point.

In Boolware a search session is defined by two FIND Commands; all queries starting with a FIND up to the next FIND constitute a search session. During such a search session all queries and the associated results are saved in a *Query History*.

If you find - after many queries - that you want to back and continue the searching from an earlier point you should use the BACK Command. Of course there is also a FORWARD Command that is used to step forward in the *Query History*.

Below you will find a very simple example how to use the *Query History*. First all the queries are presented and below that you will find the *Query History*. In a normal session you will of course get the *Query History* after each query.

Each line in the *Query History* contains the following information: number of hits after the current command, number of hits generated by the current command, the command and finally the search string (terms and operators max 500 characters) .

Example:

```
FIND car
AND red
AND big
AND (motor OR roadracing)
AND >=20000101
OR  culture
```

Query History:

97.147	97.147	FIND car
950	23.018	AND red
249	34.464	AND big
72	14.112	AND (motor OR roadracing)
14	69.005	AND >=20000101
70.256	70.262	OR culture

After this search session you could observe that the query on **culture** probably has destroyed the result of the earlier queries. The search term **culture** should perhaps have been included in the query that was about **motor** and **roadracing**. You could now use the BACK Command twice and rewrite the query about **motor** and **roadracing**: AND (**motor** OR **roadracing** OR **culture**) and then AND >=**20000101** and you will get a completely different result; hopefully the one you expected.

Query History:

97.147	97.147	FIND car
950	23.018	AND red
249	34.464	AND big
104	84.374	AND (motor OR roadracing OR culture)
27	69.005	AND >=20000101

The commands AND, OR, NOT and XOR could be followed by an ending F (ANDF, ORF, NOTF and XORF) to mark a new command - for the *Query History* - when specified as operators. The above operators are only used to get an "intermediate result".

By using the operator ANDF you could specify the above five commands as one command and get exactly the same *Query History* as when specifying the five commands:

```
FIND car ANDF red ANDF big ANDF (motor OR roadracing OR culture) ANDF
>=20000101
```

The Query History for this this query will be:

97.147	97.147	FIND car
950	23.018	ANDF red
249	34.464	ANDF big
104	84.374	ANDF (motor OR roadracing OR culture)
27	69.005	ANDF >=20000101

Saved Set

Saved Set is used to save the current query and its result. There are two types of *saved Sets*:

1. The system sets a generated name of the query and automatically saves the query: command, argument, operators and result.

   The name starts with the letter **S** and is followed by a serial number starting from 1 at start of a user session; e.g. **S1**, **S2** etc.

   This function is activated/deactivated via settings.

2. The user set a name on the *saved Set* when querying and the saved *named Set* will be automatically saved when the query is done using the specified name.

As this function slows down the queries and requires disk space to save the query and its result it should only be used when necessary. A query could contain hundreds of thousands character when it comes to similarity search and the result is dependent on the size of the database and the number of hits in a query. A typical *saved Set* for a table containing 1.000.000 records requires about 40 - 50 Kb disk space.

A *saved Set/saved named Set* is personal; each user has its own bunch of *saved Sets*.

A *saved Set/saved named Set* could be used in the query at any place at any time as long as it belongs to the current table. It is a convenient way of re-using a complex and popular query which could be tiresome to retype each time. Another advantage is that no re-search is performed; the old result is fetched directly which is much faster.

The *saved Set/saved named Set* is activated for querying by using the sub command *set*(**name**), where name is the **name** of the *saved Set*.

The *saved Sets/saved named Set* are only alive during the current user session. When the user logs out all *saved Sets/saved named Set* for that user will be removed from the system.

Below is a very simple example of how to use *saved Sets/saved named Set*. Before you get any *saved Sets* you must enable that function using settings and Boolware will *save Sets* until the function is deactivated.

Example:

```
FIND:"my car"  car
AND red
AND big
AND (motor OR roadracing)
```

```
AND >=20000101
```

```
REVIEWSET name="*"
```
will display all the saved Sets

Saved Sets:

Name	Time	DB	Table	Hits	Query
"my car"	2003-10-17 15:49:22	Magazines	News	97.000	car
S2	2003-10-17 15:50:42	Magazines	News	27.000	red
S3	2003-10-17 15:52:14	Magazines	News	82.000	big
S4	2003-10-17 15:52:57	Magazines	News	12.000	(motor OR roadracing)
S5	2003-10-17 15:54:31	Magazines	News	69.000	>=20000101

The original query:
```
FIND car AND red AND big AND (motor OR roadracing OR culture)
```

could be:
```
FIND set("my car") AND S2 AND S3 AND (S4 OR culture)
```
or
```
FIND set("my car") AND set(S2) AND set(S3) AND (set(S4) OR culture)
```
if **S1**, **S2** etc has a meaning as search terms.

When using the *saved Sets/saved named Sets* Boolware does not search the indexes but only fetches the saved result, which of course is much faster.

Important: When *setsearch* is activated all information about every query is saved on disk. A lot of resources are required both disk and CPU time and therefore it is wise to specify the *saved named Set* in the command. Two advantages: you determine at every query if it should be save or not and by setting your own name you could keep track of your *saved Sets* in a more controlled way.

Saved Query

The function *saved Query* is used to save all queries (commands, arguments and operators) since the last FIND command. The arguments could be built up by column name and terms or just terms if free text search. The user sets a unique and descriptive name of the *saved Query* so it will be easy to identify for later use.

By specifying a *session-/username* the *saved Query* could be made public; more than one user could take advantage of it. To use a public *saved Query* you must specify the *session-/username* in the key *public*. It is the application's responsibility to control that only authorized users have access to the *saved Queries* that are public.

A *saved Query* could be used in the query at any place at any time in the future.

Unlike *saved Sets* and *saved Results* a *saved Query* could be used regardless of database and table as long as the specified columns exist in the current table.

The *saved Query* will be activated by the sub command *query*(**name**), where **name** is the name set when the *saved Query* was saved. To activate a public *saved Query* the query should be specified in the following way:
```
query(public(id) name)
```
where **id** is the name of the session-/username and **name** is the name of the *saved Query*.

There are many reasons to use a *saved Query*: complex queries could be hidden, a *saved Query* could be used to restrict certain users from certain parts of the database etc.

A *saved Query* exists until it explicitly is removed by its creator.

The function *saved Query* requires comparatively little resources, disk space.

Below is a very simple example of how to use saved Queries.

```
FIND car
AND green
AND big
AND (motor OR roadracing)
AND >=20000101
```

Save all commands from the last FIND command:

```
SAVEQUERY name="green cars" table="Motor"
```

List all *saved Queries* for this user; by omitting the name keyword all *saved Queries* will be displayed.

```
REVIEWQUERY name="*" order="name"
```

Saved Queries:

Name	Time	DB	Table	Query
big cars	2003-10-17 15:52:57	Magazines	Motor	car OR big...
blue cars	2003-10-17 15:49:22	Magazines	Motor	car AND blue AND big...
dull articles	2003-10-17 15:52:14	Magazines	Articles	big...
good articles	2003-10-17 15:54:31	Magazines	Articles	>=20000101
green cars	2003-10-17 15:50:42	Magazines	Motor	car AND green AND big...

The following query:

```
FIND car AND green AND big AND (motor OR roadracing) AND >=20000101
```

could be:

```
FIND query("green cars")
```

In this case the Boolware indexes will searched exactly as when the commands were specified separately but you do not need to specify all the commands, arguments and operators again. Moreover this *saved Query* could be used for any database indexed for free text (no column names has been specified in the arguments).

Saved Result

The function *saved Result* is used to save all queries (commands, arguments and operators) and their total result since the last FIND command. The arguments could be built up by column name and terms or just terms if free text search. The user sets a unique and descriptive name of the *saved Result* so it will be easy to identify for later use.

By specifying a session-/username the *saved Result* could be made public; more than one user could take advantage of it. To use a public *saved Result* you must specify the session-/username in the key *public*. It is the application's responsibility to control that only authorized users have access to the *saved Results* that are public.

A *saved Result* could be used in the query at any place at any time in the future.

Unlike *saved Query* the *saved Result* could only be used in the same database and table as it was saved for. If you save a result for the database Magazine and the table Articles the *saved Result* could only be used for the table Articles in database Magazine.

The *saved Result* will be activated by the sub command *result*(**name**), where **name** is the name set when the *saved Result* was saved. To activate a public *saved Result* the query should be specified in the following way:

```
result(public(id) name)
```

where **id** is the name of the session-/username and **name** is the name of the *saved Result*.

There are many reasons to use a *saved Result*: complex queries could be hidden, a *saved Result* could be used to restrict certain users from certain parts of the database, a very complex query could be time-consuming etc.

A *saved Result* exists until it explicitly is removed by its creator.

The function *saved Result* requires much more disk space than a *saved Query*.

Below is a very simple example of how to use *saved Results*.

```
FIND car
AND green
AND big
AND (motor OR roadracing)
AND >=20000101
```

Save all commands from the last FIND command:

```
SAVERESULT name="green cars" table="Motor"
```

List all saved Results for this user; by omitting the name keyword all *saved Results* will be displayed.

```
REVIEWRESULT name="*" order="name"
```

Saved Results:

Name	Time	DB	Table	Hits	Query
big cars	2003-10-17 15:52:57	Magazines	Motor	32	car OR big...
blue cars	2003-10-17 15:49:22	Magazines	Motor	14	car AND blue AND big...
dull articles	2003-10-17 15:52:14	Magazines	Articles	74	big...
good articles	2003-10-17 15:54:31	Magazines	Articles	123	>=20000101
green cars	2003-10-17 15:50:42	Magazines	Motor	323	car AND green AND big...

The following two queries:

```
FIND car AND green AND big AND (motor OR roadracing) AND >=20000101
OR    car AND blue AND big AND (motor OR roadracing) AND >=20000101
```

Could be specified:

```
FIND result("green cars") OR result("blue cars")
```

In this case no searches at all will be done in the Boolware indexes. The two specified results are fetched and a boolean OR is performed. This is of course much faster than research the two queries. The disadvantage is that you could only use these *saved Results* for the database Magazine and the table Motor.

Saved named Scratch

The function *saved named Scratch(name)* is used to save the total result since the last FIND command.

A *saved named Scratch* could be used in the query at any place at any time in the future for this session.

A *saved named Scratch* is executed by the sub command *scratch(name)*.

This function is used when you temporarily - for a short while - want to save a result to be used later on. As a *saved named Scratch* is saved in memory it is more efficient than a *saved Set*, a *saved Result* or a saved result in the *Query History*.

A *saved named Scratch* exists until a new *saved named Scratch* with the same name is done or until the session terminates.

This function requires little resources except internal memory.
Below you will find a very simple example on how to use the *saved named Scratch*.

Example:
```
FIND Text:car
AND Text:green
AND Text:big
AND Magazines:(motor OR roadracing)
```

Save the current result temporarily using the saved named Scratch.

```
SAVESCRATCH name="greencar" table="Motor"
```

Search all articles that are produced during 2000 and perform an AND against the previous result. Take care of this result.

```
FIND Date: 20000101..20001231
AND  scratch(greencar)
```

Search all articles that are produced during 2003 and perform an AND against the previous result. Take care of this result.

```
FIND Date: 20030101..20031231
AND  scratch(greencar)
```

Getting data from RDBMS

As the Boolware system only stores the index from the database the actual data will be fetched from the RDBMS.

The Boolware system offers two different ways to get the data: by returning the Primary Keys to the Application or to return the actual data. In the first case the Application uses the Primary Keys to fetch data from the RDBMS, while in the second case all data will be provided by the Boolware system.

Statistics on Query Terms

This is a function where you could use the Boolware system to get statistics on query terms used during online sessions during certain time intervals. It is the application that determines which query terms that should be saved and the resolution of the time interval.

How to set up for statistics on query terms

There should be a special table containing at least two columns. In one column you save all query terms you want statistics on and in the other column - the primary key - you save the time for these query terms. The primary key should be of type *datetime* (yyyymmdd hh:mm:ss).

Normally you save all words and strings used when querying, but it is up to the application to determine. The query terms should be saved in the column or columns reserved for this purpose.

The primary key determines the resolution when you want the statistics; if you save a row with a primary key every 5th minute, the resolution when getting the statistics is 5 minutes (you could get statistics for every 5 minute interval). Of course you could choose a wider interval when you want the statistics: every hour, every day, every week etc., but you could - in this case - never get a higher resolution than 5 minutes.

How to handle the table containing query terms for statistics

This table should in all respects be handled exactly the same way as any table connected to the Boolware system.

The database containing this table should be registered as usually. The table should contain at least two columns: a primary key containing the *datetime* and the column(s) containing the query terms. The table should be activated and the column(s) containing the query terms should have the following indexing methods: *String* / *String Sort*, *Word* and *Query term statistics*.

The timestamp which is the primary key should be in the form yyyymmddhhmmss (datetime) to be able to handle intervals in a proper way in Boolware. Boolware could handle special characters between the different items e.g.: yyyy-mm-dd hh:mm:ss are valid for Boolware.

The query terms which are stored in the other columns could be both of type word and string. All query terms are separated by the pipe character (|).

When you have chosen the resolution - let's say 5 minutes - all query terms used during 5 minutes are saved in a record. Every fifth minute the record, which should be inserted in the data source, holding the primary key and all saved query terms during this time interval.

The application could - as usual - determine when it wants Boolware to online-update its indexes for this table.

As soon as an online-update in Boolware has taken place the query terms are ready to be used for statistics purposes.

How to get the query term statistics

The table containing query term statistics is handled exactly as any other table connected to Boolware.

You could view the different query terms (words and strings) that have been saved using the normal View Index function.

You could query rows containing a special query term as in an normal index.

But most of all you could extract all query terms during a specified time interval and have them presented in ascending or descending order according to occurrence.

An example:
There is a table named QueryTerms containing two columns: CurrTime and Terms. In CurrTime the primary key will be saved as datetime. In this case we have chosen a resolution of five (5) minutes, which means, that every fifth minute a record will be created where all used query terms during this 5 minutes will be saved in the column Terms.

You are interested in getting the most common query terms used between 11.00 and 13.00 the 14th of June 2005. You want them in descending order (the most common first).

The following command should be used:

In these examples the principal - not the correct syntax - is used. The correct syntax will be found in the manual "**Programmer's Guide**".

indexex **table**="QueryTerms" **field**="'Terms' count desc" **max_terms**="20"
start_term="searchterms(2005-06-14 11:00..2005-06-14 13:00)" **type**="1" **zoom**="no"
continuation="no" **commandheader**="no"

Chapter 12
UNICODE

Support for UNICODE in Boolware

Unicode is a contemporary and unifying alphabet standard that attempts to include all the character soups used around the world. This "alphabet" includes western characters, as well as arabian, african, asian etc. chars.

Older computer alphabets can only manage 256 different chars. This is often sufficient, but causes a lot of problems for those applications that need to address an international public. Hence Unicode.

Boolware has the following support for column data:

- supports columns of "Unicode" type in databases.
- can define up to 65.536 different chars when indexing.
- can index and use queries containing all 65.536 Unicode chars.
- can deliver Unicode tuples from the database, containing full Unicode.
- stop words, synonyms, thesauri, stemming now utilizes Unicode.

The character set ISO/IEC 8859-1 is used by default for compatibility when clients connect to the Boolware server. This means that the Unicode support is not activated by default – a client must take an action to activate the Unicode support. This ensures that old apps will not be broken when installing Boolware 2.4. Activate Unicode by using the Execute API:

```
setencoding value="UTF-8"
```

UTF-8 means that all strings, without exception, are transported between the application and Boolware as Unicode encoded as UTF-8.

Which of the options that would be preferred, is determined from what is most practical on a solution to solution basis.

Important: When a column contains information coded as XML (see section "Indexing of fields within columns" in Chapter 5 "Editing indexing properties") this information will always be transported to the application as is, regardless of the value specified in setencoding.

Old Indexes and applications, compatibility with Unicode

The new support for Unicode only affects columns of type Unicode. E.g. existing Indexes that do not use Unicode columns, are compatible without changes. They do not have to be converted or reindexed. Only tables that contain Unicode columns need be reindexed (to take full advantage of the Unicode functionality).

The Unicode data type is defined differently, from database to database, but nchar and nvarchar are common names. See the RDBMS manuals for further information about Unicode support in your chosen database.

Restrictions

If you want to store data coded as UTF-8 in the database, the data type must be either Unicode or Binary. Otherwise, the database will attempt to encode the data (which is already UTF-8) once more as UTF-8.

Boolware only supports the ISO/IEC 8859-1 character set for metadata: database name, table name, column name, etc.

Chapter 13
Boolware Cluster

General

To make Boolware more available for searchers there is a possibility to connect several machines - nodes - in a Boolware cluster. Note that all active nodes in a Boolware cluster must run on the same version of Boolware.

In Boolware there are two types of clusters; Mirroring cluster and Search cluster.

In a mirroring cluster the Master node replicates all index, configurations and settings to all other nodes in the cluster. A mirroring cluster includes a Master node, a Failover node and possibly other search nodes. Adding a new node/nodes in the cluster definition will default set the node/nodes to an inactive sate. This means that no replication will take place until each node is set to an active state.

In a search cluster the Master node replicates only configurations and settings to all other nodes in the cluster. All nodes in the cluster must have access to the disc device where the Master node store all indexes etc. A search cluster includes a Master node, a Failover node and possibly other search nodes

NOTE! In a search cluster, you cannot mix Windows nodes and Linux nodes. All nodes must run either under Windows or all nodes under Linux.

The below paths should in a search cluster point to the same disk space for each node in the cluster:
- Path for sysindex
- Default index path
- Alternative index path
- Path for backup
- Path for KWD-files (if empty the Default index path will be used)
- Online temp rank
- Online temp sort
- Save session data

The below paths should in a search cluster be machine specific for each node in the cluster:
- Log file path
- Temporary files path

In Boolware Manager you create and maintain the cluster. For a more detailed description, refer to the Help function in Boolware Manager.

LBST

In a Boolware cluster the Master node will set the LBST to NO on a Failover node or Slave node if status on the node change from "Inactive", "No contact" to "Active" or if a node is restarted. When all tables in all data bases and any data is replicated the Master node will set the LBST to the value that the node had when it was started (YES/NO).

Replication of data in a Boolware cluster

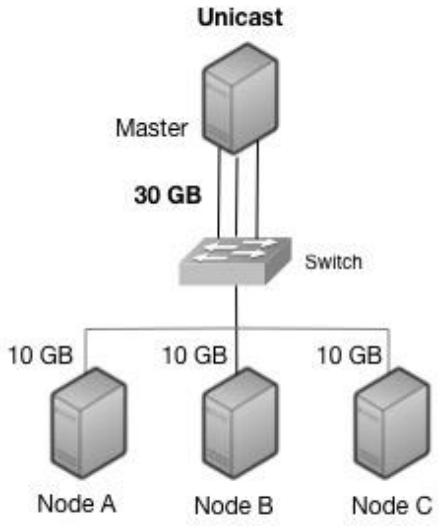
Replication of data in a Boolware cluster is done by one of the two options below.

- Unicast (TCP) - default
- Multicast (UDP)

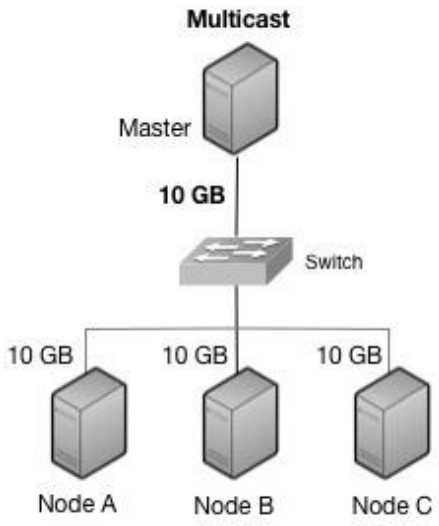
With Unicast replication, the master sends all data several times in the network.

With Multicast replication, the master sends all data just once in the network.

See examples below for the differences between these two options:
Example: Replication of 10 GB to three nodes



Unicast replication means that a total of **30 GB** is sent and transferred through the network.



Multicast replication means that a total of **10 GB** is sent and transferred through the network.

Prerequisites and requirements for Multicast

1. A Boolware cluster must be defined and configured.

2. The UDP port used for multicast traffic, default 9001, must be open in the firewall on all computers in the Boolware cluster.

3. Network switches and routers that connect the computers in the Boolware cluster must support and be enabled for multicast.

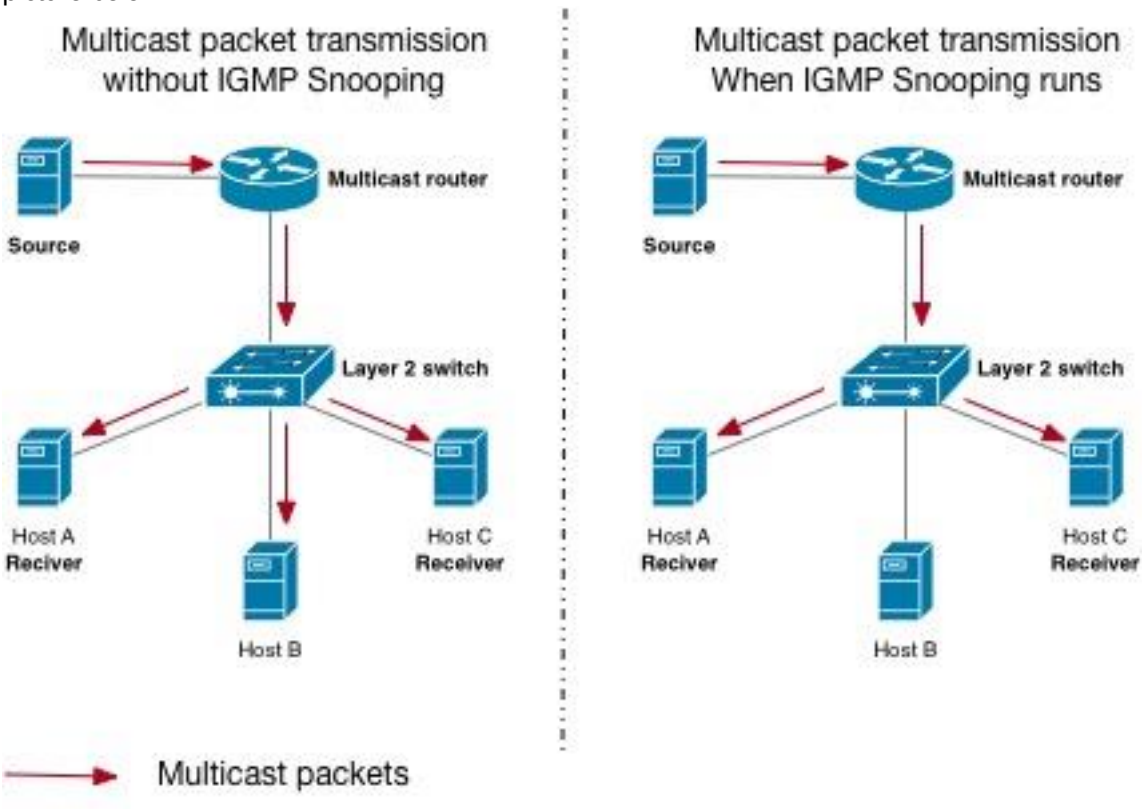
4. An interval of multicast IP addresses must be reserved for the Boolware cluster in the network. The number of multicast addresses that Boolware uses depends on the number of tables replicated and the number of eventual re-replicates. We recommend reserving 255 multicast IP addresses per Boolware cluster.

For a more detailed description of how to configure and activate Multicast in a Boolware cluster, refer to the help function in Boolware Manager.

IGMP Snooping

IGMP Snooping is the process that listens to the Internet Group Management Protocol (IGMP) to control the delivery of multicast packets. Network switches with IGMP-Snooping listens to the IGMP conversation and checks which network cards requested IP multicast transmission. This allows it to filter out network cards that do not need the multicast transmission, and thus maintain bandwidth on these network cards.

Without IGMP Snooping, Multicast traffic is treated in the same way as a "broadcast" transmission, i.e. the packets are forwarded to all ports on the network. With IGMP Snooping, Multicast traffic is only forwarded to the ports that are members of the Multicast group. See picture below:



Without IGMP Snooping, Host B gets traffic even though it is not a Multicast receiver.

With IGMP Snooping, Host B does not get traffic because it is not a Multicast receiver.

FAQ

For a Boolware cluster, the following applies:

1. that all nodes are configured with the same IP family (IPv4 / IPv6) or with DNS
2. that the same Boolware version runs on all active nodes in the cluster

Below are a number of scenarios that may occur in a cluster:

Scenario 1:
A Boolware cluster is activated but the Failover node runs another Boolware version. => The Failover node is set inactive and will not receive any files or settings if these are changed in the cluster. Install the same Boolware version as the other nodes in the cluster and activate the node

Scenario 2:
A Boolware cluster is activated but a Slave node runs another Boolware version. => The slave node is set inactive and will not get any files or settings if these are changed in the cluster. Install the same Boolware version as the other nodes in the cluster and activate the node.

Scenario 3:
A Boolware cluster is activated but a Failover / Slave node is not started / does not answer => The cluster can be activated but the node remains as inactive. If the node is set to active, the Master node will set it to "No contact". If the node comes up / responds again, it is set to active if the cluster is configured for auto-activation. See also the section on LBST above.

Scenario 4:
A Failover / Slave node falls off during operation => The node will be set to "No contact". When the node becomes contactable, the status will be reset to the value the node had before, if the cluster is configured for auto-activation. See also the section on LBST above.

Scenario 5:
The master node is dropped / stopped during operation and is gone for a longer period (configured "Timeout" time) => If the Failover node is active, it will take over and become a new Master node. When the "old" master node is started, it checks if the Failover node is Master and then becomes Failover node. Via Boolware Manager you can access the "new" Master node and switch the "old" Master node to Master.

Scenario 6:
The master node is started in a cluster that uses DNS addresses but it resolves its address to a different IP address than the one that Boolware Master started with => The cluster will be completely disabled. The cluster definition can be opened using Boolware Manager to fix the error.

Scenario 7:
A Failover or slave node is added to the cluster that is configured with DNS addresses but it cannot be accessed (the DNS server does not respond) => The node cannot be added to the cluster. Make sure the DNS server responds and the correct address is specified.

Scenario 8:
A cluster node starts up and the cluster is configured with DNS addresses but it cannot be resolved (the DNS server does not respond) => The latest IP address will be used. Check access to the DNS server.

Scenario 9:
Resetting the cluster but one or more nodes do not answer => Reset occurs on nodes that respond. Other nodes must be reset locally (with Boolware Manager) when started.

Scenario 10:
If a load replication gets error when replicating to a node => If it is a multicast replication, the Master node restarts the replication as unicast. The master node restarts a unicast replication as long as it replicates to other nodes.

Scenario 11:
If consistency replication gets error when replicating to a node => If it is a multicast replication, the Master node restarts the replication as unicast, otherwise trying again at the next consistency check.

Scenario 12:
If a node falls off during a replication job => The node is set to "No contact" and replication continue to the other nodes. If the node is started/answers again, when the replication is still in progress to other nodes, a unicast replication is restarted to the node.

Chapter 14
Plugins

General

Plugins are external components that can be used to tailor certain functions in Boolware.

Identification of plugins designed for Boolware starts with a lower case "e" followed by a double-digit type code. Several different types of plug-in modules can be used in Boolware. These types are:

e01 – custom indexing
e02 – custom scoring
e03 – custom phonetics
e04 – dynamic ranking of search results

The following four plugins for indexing (e01) are included in a Boolware delivery:

- Split – is used when you want to "split" words that consist of both letters and digits.
- Shrink – is used when you want to generate combinations of different types of abbreviations.
- Linkwords – is used when you want to merge adjacent words.
- Addindexwords – is used when you want to add extra words to your index.

A Boolware delivery also includes another plugin for dynamic ranking; e04.customrank.

Some types of plugins such as e.g. customized indexing (e01) can be activated at multiple levels; database, table and field level, while other types of plugins such as e.g. dynamic ranking (e04) can only be activated at the table level.

For more information how to use plugins, see the Help in the Boolware Manager.

A plug-in module is implemented as a .dll (Windows) or .so (Linux) and must be placed in Boolwares installation directory to be activated in Boolware Manager.

Note that Boolware must be restarted when a plugin is added to or removed from the installation directory.

For more information how to develop your own plugins that can be used with Boolware, you can read about this in "Programmers Guide", chapter 10.

Custom indexing

In Boolware there are many different ways to configure how a word or words are to be indexed in order to obtain as good search results as possible. Examples of general internal features are: string, phonetic, stemmed, etc.

In some cases it is not enough with these general methods, but it requires a customized function to take care of the words when searching. In Boolware there is the possibility that through plugins write your own function for managing the generation of index terms and query terms. This means that you can decide how search terms should look like and thus may affect the relevance of the search.

When is custom indexing used?

The customized function will be called each time Boolware will extract words from a text or a specified query for a column, as it has been registered.
This is done by:

1. Indexing, when all rows in the data source is read (may be deselected when activated and then used only during search).
2. Online-update, when only rows that have changed are read from the data source (may be deselected when activated and then used only during search).
3. Search, when the specified query should be interpreted (can be disabled with the help of a command) (may be deselected during search by the application)
4. Test Indexing, when the system administrator is experimenting with the indexing options

It is only the contents of columns, which have been marked for customized indexing, which will be affected by the custom function.

In addition to the words that Boolware create, words generated by the custom function will be added to the index or used in queries.

Example:

Imagine that you have the following street addresses of three or more records in the data source

10 D. Street
10 Downing st.
10 Downing Street

The custom plugin, for example, would be able to normalize the street addresses of these records by creating three terms that refers to all these records: "10 Downing Street"
So no matter which of the three ways above you type the address in the search, you will always get a hit on all records that have a normalized address in this example.

How to control the search for custom indexing?

If a customized indexing function has been registered for a column, it will automatically be used when searching. This means that the search will be carried out on both the words that Boolware generates and the words that the custom indexing function creates.

It is in the custom indexing function that you can tell which operator it should be between the generated terms, if more than one term are created (if no operator is specified, AND will be used). You can also tell which command it should be between these terms and the terms created by Boolware (if no command is specified, OR will be used).

One can at each query determine whether the terms generated by the custom function should be included or not in the search. This is done with the command "setindexexit value=no" or "setindexexit value=yes". The current setting is effective until a new setting is made. A column can have multiple customized plugins for indexing and the above command will apply to all of these. To only disable the custom plugin Linkwords, there is a sub-command, nolinkwords().

When a session logs on, the custom index function is always activated, if it exists.

Standard custom index plugins

The following four plugins for indexing (e01) are included in the Boolware delivery: Split, Shrink, Linkwords and Addindexwords. In the Boolware delivery another plugin for dynamic ranking; (e04) Custom Rank, is also included.

Split (e01)

This plugin is used when you want to "split" the words, which consist of both letters and numbers. A split occurs between each transition from letter to digits and vice versa.
This plugin creates only "splits" terms when index option is set to word.

The following control parameter can be specified for this plugin using Boolware manager:

OnlySearch
Controls if this plugin should only be used for searching. Allowed values are:
0 – (default) the plugin will be used for both indexing and searching.
1 – the plugin is used only when searching.

Example:

The following words sent by indexing: Mercedes SL500 Volvo S70.
The following words are stored in the current word index (terms generated by the plugin (hook function) in bold):

MERCEDES (word)
SL500 (word)
SL (word)
500 (word)
VOLVO (word)
S70 (word)
S (word)
70 (word)

You will get a hit no matter, when searching, if one writes:

FIND sl500
FIND sl AND 500

Shrink (e01)

This plugin is used when you want to generate combinations of different types of abbreviations. One can by using various parameters control the way in which you want to generate these words. This is useful when searching for company names that are abbreviated in different ways.

The following control parameters can be specified for this plugin using Boolware manager:

OnlySearch
Controls if this plugin should only be used for searching. Allowed values are:
0 – (default) the plugin will be used for both indexing and searching.
1 – the plugin is used only when searching.

Sep
Specifies separating words. The specified words are separated by a space.

Before
Number of characters in the word before the separating word.

After
Number of characters in the word after the separating word.

For this plugin to create new words, the column must be indexed as string. The words that are created will however be both string and word.

Example:

Enter the following parameters:

OnlySearch=0
Sep=& och og and et
Before=2
After=3

For example given the string: "Hennes & Mauritz", the following terms will be indexed (terms generated by the plugin in bold):

HENNES & MAURITZ (string)
HE & MAU (string)
HE&MAU (string)
H & MA (string)
H&MA (string)
H & M (string)
H&M (string)
HENNES (word)
MAURITZ (word)
HE&MAU (word)
H&MA (word)
H&M (word)

Regardless which of the following searches that are used, you will get search results:

FIND h&m	word search (not changed)
FIND he&mau	word search (not changed)
FIND henn&maur	word search (changed to: he&mau)
FIND "h&m"	string search (not changed)
FIND "h & m"	string search (changed to: "h&m")
FIND "henn & maur"	string search (changed to: "he&mau")
FIND "hennes&mau"	string search (changed to: "he&mau")

etc.

Linkwords (e01)

This plugin is used when you want to concatenate adjacent words. Fairly common in person name, where "double name" can be written in three ways: Karl-Erik, KarlErik and Karl Erik. By using parameters you can control how to "concatenate" words.

The following control parameters can be specified for this plugin using Boolware manager:

OnlySearch
Controls if this plugin should only be used for searching. Allowed values are:
0 – (default) the plugin will be used for both indexing and searching.
1 – the plugin is used only when searching.

starttermpos
Specifies which term you should begin with. Usually one starts on the first term (1), which is also the default value.

termstoconcatenate
Specifies the number of terms to be "concatenated". Usually one combine two (2) adjacent terms, which is also the default value.

nextstartpos
Specifies the next term that should be the first term in the "concatenation". Usually you want to select the next term (1) as the starting term at the next assembly. This is also the default value.

maxtermstogenerate
The maximum number generated (assembled terms). The joining ceases, then this value is reached, even if there are more terms. The default value is 10.

excludeterms
Terms not be included in the merge. If it's the company name that is to be "concatenated", usually one adds: AB, Public Company, Inc., Co., Ltd., oy, said, srl, etc.

supresscompress
Indicates if you do not want to "compress" the initials (words consisting of only one character). When choosing a field of names, there might be initials, which you do not want to do "compress". The default value is 0; allow "compress".

Example:

Enter the following parameters:

OnlySearch=0
starttermpos=1
termstoconcatenate=2
nextstartpos=1
maxtermstogenerate =10
excludeterms =AB, Aktiebolag, Inc, Co, Ltd, oy, sa, srl
supresscompress=0

Given the company name: "Agent 25 Norden AB Partner", the following terms will be indexed (terms generated by the plugin (hook function) in bold):

AGENT (word)
25 (word)
NORDEN (word)
AB (word)
PARTNER (word)
AGENT25 (word)
25NORDEN (word)
NORDENPARTNER (word)

In this case, one can search in the following way and get a hit:

FIND Agent 25 Norden – the plugin is not used; words are found anyway.
FIND Agent25 Norden – in this case, the plug-in module (Hook function) generated word Agent25 are used.

Since this plugin, in some cases, generate results that are not desirable, there is a sub-command, nolinkwords(), which can be used in the search for not creating "concatenated" words.

FIND nolinkwords(Agent 25 Norden) – In this case the the "concatenated" word Agent25 is not created.

Addindexwords (e01)

This plugin is used when you want to add variants of terms.
These variants of words can be synonyms, nicknames, startings and endings of terms.

The following control parameters can be specified for this plugin using Boolware manager:

OnlyLoad
Controls if this plugin should only be used for loading index and online-updates.
Allowed values are:
0 – the plugin will be used both when loading or online-update indexes and when searching.
1 – (default) the plugin is used only when loading or online-update indexes. The plugin will not be called during search.

<fieldname>
This specifies a section in the configuration and starts with '<' and ends with '>' and encapsulates one or more comma-separated field names to which the following rules will apply. Multiple sections may occur in the configuration with different field names for different rules. The field name should be the field name from which terms are generated. If the plug-in module is applied to an Alias Index field, enter the name of the input field here and not the name of the Alias Index field.

FindString
Allowed values are:
0 – (default) each word in field / fields should be used when matching rules.
1 – the entire contents of the specified field / fields should be used to match rules.

Example 1:
The plug-in module is applied to the "Address" field and will generate words with different types of name endings.

If a term from the data source ends with "*er", the following ending is added to the terms that are generated; "*er", "*est".

To achieve this, the following rules should be specified in the plug-in configuration:

```
OnlyLoad=1
<Address>
FindString=0
*er(*est)
```

Example 2:
The plug-in module is applied to the field "Address" and applies different types of synonyms to data from the data source. If the data source contains any of the terms "st", "street", or "road", the other variants will be added to the index and become searchable for the record.

To achieve this, the following rules should be specified in the plug-in configuration:

```
OnlyLoad=1
<Address>
FindString=0
st(street, road)
```

Example 3:
The plug-in module is applied to the field "Firstname" and applies nickname for some names. If the field "Firstname" of the data source contains the words Robert, Marie and Mary, the nicknames Bob and Mia will also be indexed and searchable for the record.

```
OnlyLoad=1
<Firstname>
FindString=0
Bob
  Robert
```

```
Mia
  Marie, Mary
```

Example 4:
The plug-in module is applied to the Alias Index field "Alias Address", which consists of the following three fields: address1, address2, address3.

```
OnlyLoad=1
<address1,address2,address3>
FindString=0
*er(*est)
st(street, road)
```

If any of the three included fields contains the text "Oxford Street" or "larger", then in addition to the words in the record, the following words are generated and indexed for the record: Largest, st, road.

Dynamic ranking

This plugin is used when you want to "dynamic" rank a search result to get the "best" matches first. The fields that you want to rank on must be indexed or have the "Store column data" attribute set.

The API for custom rank (e04) is defined in the file: e04.customrank.h.

Configuration
The following configuration parameters can be used for this plugin:

fieldnames
Here you enter the data fields you want to use in the ranking criteria. The field names are separated by a comma. If the field name contains comma, space or parentheses, then the field name must be enclosed with quotes. You can also define a group of field names by entering a group name, eg. "Name" followed by a left-hand left parenthesis and the field names that should be included in the group separated by comma character and a right-hand parenthesis to end the group. Groups are useful if you use alias fields for searches. You can then define ranking criterion with group name instead of field name. All fields in a group are assigned the same search arguments and ranked (sorted) in the order they have been entered in the group.

Example:

fieldnames=Name("First name", "Last name")

wordsep
Here you specify which characters should be considered as word breaking. In addition to the characters that you specify here, all control characters, ie character codes with a value lower than 32, will automatically be considered as word breaking.

max_to_rank
Here you specify how many records you want to rank at most.

rank_criterias1
Here we specify the sorting criteria you want to use, separated by a comma.
Each sorting criteria consists of three parts; field names, sorting function and sort order.

It is possible to specify additional ranking criteria by defining additional keys, such as rank_criterias2, rank_criterias3 and so on. In a flow one can then specify which of the defined ranking criteria to be used.

Here is an example of how a configuration might look for the plugin e04.customrank:

```
fieldnames="Company name", "Turnover (tkr)", Address, City
wordsep=" -\xA0"
max_to_rank=1000000
rank_criterias1="Company name" number_of_words_before_match asc, "Company name" asc
rank_criterias2="Company name" data_length asc, "Turnover (tkr)" desc, "Company name" asc
;rank_criterias3="Company name" number_of_words asc, "Company name" data_length asc
```

The semicolon that appears first on the line means that all the content on the line is treated as a comment. In the example above e.g. rank_criterias3 is commented out.

Sorting criteria
The following sorting criteria can be used in a ranking criteria:

Sorting criteria	Description
alphabetically	sort alphabetically.
data_length	sort by number of characters in the field. Max data length for a custom rank field is 32000 bytes.
geodistance	sort on distance from a reference point.
withingeodistance	sort search hits that are within the specified distance from the reference point before search hits that are outside the specified geographical area. If you want to reverse sorting order, you must specify sorting order desc.
number_of_words	sort by number of words in the field.
number_of_words_after_match	sort on number of words that remain after the first hit in the field.
number_of_words_before_match	sort on number of words that precedes the first occurrence in the field.

If no sorting criteria is specified, alphabetical sorting is used.

Sorting order
For each sorting criteria, one of the following sorting orders can specified:

asc – ascending order
desc – descending sort

If no sorting order is specified, the sorting order is automatically ascending.

Examples of ranking criteria with a sorting criteria:

```
rank_criterias1="Company name" number_of_words_before_match asc
```

The *emptyfirst* or *emptylast* command allows you to control how records that have missing data in a sort field should be sorted. This command specified after any sorting order. Default value is *emptylast* unless otherwise specified.

Example:

```
fieldnames= turnover
rank_criterias1=turnover asc emptyfirst
```

GEODISTANCE and WITHINGEODISTANCE
In order to use any of these sorting criteria, the table must contain coordinate data. To the left of the geodistance or *withingeodistance*, you must enter the field name or field names that contain coordinate data. If latitude and longitude coordinate data is stored in different fields, the field

name containing latitude data must be entered first, followed by a plus sign and then the name of the field containing longitudinal data.

Example:

fieldnames=latitude, longitude
rank_criterias1=latitude+longitude geodistance

If latitude and longitude coordinate data is stored in the same field, then enter only the name of that field.

Example:

fieldnames=geodata
rank_criterias1=geodata geodistance
Example:

In this example we have searched for hotels and want to sort the hits so that hotels located within **500** meters from Fiskekyrkan in Gothenburg (N 57° 42' 2.39", E 11° 57' 16.79") are sorted on distance from Fiskekyrkan in ascending order.

<resultset score="1"
action="add"
customrankexit="customrank"
customrankselection="rank_criterias1"
customrankgeodistance="57.700663864, 11.954662848, **500**"
customranklimit="10000"/>

Geo groups
The sorting criteria geodistance and withingeodistance can also be used to group search results geographically. A geo group is an parentheses expression with one or more sorting criteria, where the first sorting criteria must be either geodistance or withingeodistance.

In the example below, search results that are within the specified geographical area are sorted first, and within the geographic area, the hits are sorted by postal code and street name in ascending order. Search results outside the geographic area are sorted by the street name only.

Example:

fieldnames=latitude, longitude, "street name", "postal code"
rank_criterias1=(latitude+longitude withingeodistance, "postal code"), "street name"

Usage of custom ranking
Customized ranking is used when you want to dynamically rank a search result.
Usually, this function is called from a flow with the <result> element.

Example:

```
<resultset score="1" action="add"customrankexit="customrank"
                          customrankselection="rank_criterias1"
                          customranksearchcriteria="$searchArg"
                          customrankautotrunc="0"
                          customranklimit="10000"
                          customrankgeodistance=""
                          customrankuserdata=""
                          use_existing_score="0"
                          limit="250000"/>
```

You can also invoke this function in an XML call by entering a <sort> element in the <response> element with the following attributes:

```
<sort customrankexit="customrank"
     customrankselection="rank_criterias1"
     customranksearchcriteria="$searchArg"
     customrankautotrunc="0"
     customranklimit="10000"
     customrankgeodistance=""
     customrankuserdata=""
     use_existing_score="0"/>
```

Customrank attributes and their meanings

customrankexit
The name of the plug-in module that you want to use without the prefix "e04.".

customrankselection
The ranking criteria that you want to use from the configuration.

customranksearchcriteria
The search arguments are used for the each ranking field or group separated by comma. If the search argument contains commas, they must be enclosed with quotation marks.
Note: The order of the search argument must be the same as the order of the ranking fields in the plug-in configuration. If you use groups, you only specify the search arguments for the group and not for each field in the group.

customrankautotrunc
If you want the plug-in to use automatic right truncation when matching search arguments, enter 1. Allowed values are 0 or 1. The default value is 0 if the attribute is not specified, i.e. customrank will not use automatic right truncation to match search arguments.

customranklimit
Here you can specify maximum number of search hits that you want to rank. If no value is specified, the value from the configuration parameter max_to_rank will be used instead.

customrankgeodistance
Here you can enter one or more coordinates along with a distance expressed in meters. If you enter multiple coordinates along with it's distance, you must separate them with semicolon. The coordinates and distances specified here will be used by the sort criteria geodistance and withingeodistance in the order that these sorting criteria are defined in the configuration.

customrankuserdata
User defined data that you want to send to the plugin.
This attribute is not used by the included plugin module.

use_existing_score
If you want the plugin module to respect specified scores and only rank items with the same score, specify 1. Allowed values are 0 or 1. The default value is 0 if attributes are not specified. Rankings will then be done without consideration for score.

Chapter 15
Flow queries

Introduction

Boolware can handle complex queries that include application logic. These are called "flow Queries", since they start at a point, and flows its way down through the defined search logic (like a flow chart) until producing a desired result. An application can – in a single call - include several query fields as well as logic how the fields should be processed in the query. These rules can for example describe what to do if a search doesn't generate any results, or if it generates "too many" hits.

Flow queries are passed as XML to Boolware. Three parts can be identified:

- The query
- The flow description
- The response

String comparisons

To assess the similarity between strings (e.g. "Arvid E. Nordquist" and "Arvid Nordkvist") an internal compare function is used that produces a similarity measure expressed as number of errors.

This function – Compare – is specially adapted to be fault tolerant and is also accessible directly from flow scripts. The term fault tolerant means that it accepts minor differences (i.e. a missing letter/too much), without seeing the strings as totally different. The function Compare is based on Damerau-Levenshtein.

Normalization

Before comparison it is of great importance to normalize the strings. E.g. to make all letters capital, to remove accents and other diacritical characters, etc.

Boolware contains a slew of functions to normalize strings. For example different kinds of phonetic methods can be used, so that words will be considered the same if they sound the same, even if spelled differently. Boolware also contains powerful search and replace functions based on the *RegEx* unofficial standard, so that one can easily setup own functions for "data wash" and normalization before comparisons.

Weighting

Since certain fields are more powerful distinguishers than others you can assign weights to fields. For example social security number and name are more powerful identifications than zip code and street address, since people often move but rarely change their names. Weights are expressed as percentages, where 100 means "full weight" and 50 is interpreted as "half weight". In an example containing the fields social security number, name, street and city one could use 100, 90, 50 and 30 as suitable weights.

A weight percentage of 50 make differences less significant than a weight of 100.

Scoring

Scoring is based on comparisons of search fields and found fields, resulting in a grading controlled by different weights and parameters. The result of *Scoring* is a hit list ordered with the most relevant record first, followed by the second most etc.

The "scoring" element is used to centralize rules for how found records are arranged. Scoring contains one or more "score" elements, one per search field. Example:

```
<scoring maxrows="10" minscore="93" >
  <score var="name2" weight="90" norm="131" dbfield="Last+First" />
  <score var="street2" weight="50" norm="129" dbfield="Street" />
  <score var="zip2" weight="100" norm="1" dbfield="Zip" />
  <score var="city" weight="40" norm="1" dbfield="City" />
</scoring>
```

In the example can be seen that scoring will produce the ten best hits, however at least with a score of 93 (100 - no. of errors).

Each score element describes how to handle a single field. The query record's (often "massaged") field (var) will be compared with the database field (dbfield) of the found record.

In the example can be seen that the query record's "massaged" field *name* (stored in an own variable, *name2*) is being compared to the concatenated fields *Last* and *First* from the referential database.

Different grades of normalization can be applied to adjust both strings before comparison. These are controlled by the attributes *norm*=, *sortwords*=, *noice*=, *namefile*=, *lth*=, *washchars*= and *condignore*=. See the **API documentation** for details.

Creating / editing a Flow query

Flow queries are coded as XML files, at the server in Boolware's database directory. In the Manager, using the main window right-click menu, an administrator can create and edit these files. By using an XML structure the semantic of the flow is unambiguous without any further parsing than XML. The flow description contains all search logic; the rules that should be used for the current query.

For more detailed information see Programmer's Guide Chapter 4 "**Flow Queries**" and the help function in the Boolware Manager.

Chapter 16
Encryption

Overview

Boolware has the ability to encrypt communication between Boolware servers and Boolware clients (applications) as well as between servers in a Boolware cluster. In order for Boolware to use encryption, OpenSSL is required to be installed on the same computer.

Concepts, definitions and abbreviations

Concept, definitions and abbreviations	Explanation
OpenSSL	OpenSSL is an open source implementation of the SSL and TLS protocols. The libraries are written in the programming language C and implement basic encryption features.
SSL	SSL is short for "Secure Sockets Layer" and is a protocol used to encrypt network communication between computers.
TLS	TLS, which is an abbreviation for "Transport Layer Security" and is a successor to the SSL protocol.
Symbolic link	A symbolic link is a file that contains a reference to another file.
SSL certificate	An SSL certificate is a kind of digital identification.

Prerequisites and requirements

In order for Boolware to encrypt its network communication, the following is required:

1. The libraries *libssl* and *libcrypto* from OpenSSL version 1.1.0 or later must exist on computers using Boolware. I.e. on computers running Boolware Manager/BWC, on computers running any of the Boolware clients and computer running Boolware server.
   For Boolware Manager, Demo and Dashboard to run with encryption a 32-bit version of libssl and libcrypto must exist in the installation path.
   You can get the libraries by downloading and building OpenSSL on Windows/Linux on a computer. On computers using only Boolware's Java or .NET client, the OpenSSL libraries does not need to exist.

2. Two symbolic links that points to the OpenSSL libraries *libssl* and *libcrypto*.

3. An SSL certificate with its associated private key, which you create on the computer where you built OpenSSL or where OpenSSL is installed. This is how you create a self-signed SSL certificate:
   a) Start a command prompt/terminal window with administrator rights/root privileges.

   b) This step (b) applies only to Windows. Run the following command so OpenSSL finds its configuration file (replace the path below with the path to the OpenSSL configuration file)
   set OPENSSL_CONF=<*path to OpenSSL build folder*>\apps\openssl.cnf

   c) Run the following command to create a private key:
   openssl genrsa -des3 -out server.key 1024

d) Run the following command to generate a Certificate Signing Request (CSR):
openssl req -new -key server.key -out server.csr

e) Run the following commands if you want to remove the password phrase from the key:
Windows
copy server.key server.key.org
openssl rsa -in server.key.org -out server.key

Linux
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key

f) Run the following command to generate a self-signed certificate:
openssl x509 -req -days 3650 -in server.csr -signkey server.key -out server.crt

g) Run the following commands to install the private key and certificate in the Boolware installation directory:
Windows
copy server.crt c:\program\softbool\server.crt
copy server.key c:\program\softbool\server.key

Linux
cp server.crt /usr/local/boolware/server.crt
cp server.key /usr/local/boolware/server.key

Installation and configuration in Boolware

Make sure that the OpenSSL libraries *libssl* and *libcrypto* exists on the computers running the Boolware server. If you use any Boolware client other than Java or .NET, the OpenSSL libraries *libssl* and *libcrypto* must also be on these computers.

Path to OpenSSL
Verify that the path to the OpenSSL libraries *libssl* and *libcrypto* are included in the PATH environment variable on Windows computers. On Linux, make sure ldconfig is configured for the OpenSSL libraries that is needed or specify the path in LD_LIBRARY_PATH environment variable since there might be other versions of OpenSSL libraries on the machine.

Symbolic links
In order for Boolware to load OpenSSL libraries, you must create symbolic links to the OpenSSL libraries. Here's how to create symbolic links to the OpenSSL libraries:

1. Start a command prompt/terminal window with administrator rights/root priviliges

2. Change directory to the Boolware installation directory

3. Type the following two commands (replace the path in the following commands with the path where the OpenSSL libraries are installed):
Windows
mklink libcrypto-x64.dll \path\to\openssl\bin\libcrypto-1_1-x64.dll
mklink libssl-x64.dll \path\to\openssl\bin\libssl-1_1-x64.dll

Linux
ln –s /path/to/openssl/bin/libcrypto-1_1.so libcrypto.so
ln –s /path/to/openssl/bin/libcrypto-1_1.so libssl.so

Change the owner/group of the symbolic links to the user that should run Boolware.

Configuring encryption in Boolware
When the above is done you configure encryption in Boolware like this:

1. Start the Boolware Manager and connect to the Master or "Standalone" server.

2. Click the Settings menu and select the menu option "Configuration ...".

3. Click the "Encryption" tab
4. Browse and select the SSL certificate and the associated private key. If the private key is password protected, you must also enter the password.

Java and 256-bit encryption
If you want to use 256-bit encryption and you use a Java version earlier than 1.8.0_162, then Java must be configured for this on the computer where the Java client is used as follows:

1. Download the policy file (zip file) for the java version you are using (Oracle's version contains two .jar files, local_policy.jar and US_export_policy.jar).

2. Unpack the zip file

3. Go to the installation directory for the java version used and paste/replace with the two unpacked .jar files in the directory for this (for Oracle's version in the directory: "/jre {version_number}/lib/security").

FAQ

For a Boolware cluster and encryption, the following applies:
1. that encryption can be enabled via the master node if all of the nodes have prerequisites for encryption (OpenSSL installed / configured etc.)
2. that a node in the cluster can be activated if the node has the prerequisites for encryption

In case of failure:
If a Boolware server is started which has encryption enabled but an error occurs when the encryption is set alternatively, if an error with the encryption settings is discovered during operation, the encryption will be disabled on the server.

For a Boolware cluster, then the following applies:
1. If this occurs on a Master node, encryption will be disabled for the entire cluster.
2. If this occurs on a Failover node or Search node, the node will be set inactive.

Scenario 1:
We have a standalone Boolware server and an error with the encryption occurs during operation (error on certificate or key file). => The encryption will be disabled.

Scenario 2:
We have a cluster with a Master node, a Failover node and two Search nodes. We want to enable encryption for the entire cluster but one of the Search nodes lacks the prerequisites for encryption (OpenSSL libraries are missing). => Encryption cannot be enabled.

Scenario 3:
We have a cluster with a Master node, a Failover node and two Search nodes. The Master node is experiencing an error with encryption during operation (certificate or key file failure). => The encryption will be disabled for the entire cluster.

Scenario 4:
We have a cluster with a Master node, a Failover node. We want to activate a Search node in the cluster which lacks the prerequisites for encryption (OpenSSL libraries are missing). => The Search node cannot be activated.

Chapter 17
Text extraction in Boolware

Overview

Boolware extracts by default unformatted text for indexing and searching without any extra plugins (raw text without any kind of formatting).
For extracting formatted text, Boolware uses Apache Tika. Extractable formats are those supported by the version of Apache Tika installed on the machine running Boolware server. For example Word, Excel, PDF, HTML etc.

Apache Tika will be loaded into Boolware the first time you start an indexing that needs to extract formatted text.

Prerequisites and requirements

In order for Boolware to extract text, the following is required:
1. Java 8 or later must be installed on the computers using Boolware.
2. Apache Tika 1.x series or 2.x series (Tika application - tika-app-n.n.n.jar, at least version 1.28) or later must exist on computers using Boolware. For the 1.x series the version must be at least 1.28, for the 2.x series the version must be at least 2.4.1.
3. A symbolic link that points to tika-app-n.n.n.jar

Installation and configuration in Boolware

Make sure Java is installed where the Boolware server is running. Then define the environment variable "BOOLWARE_JAVA_HOME", which should point to the Java version to be used (minimum Java 8) .
Ex. BOOLWARE_JAVA_HOME = c:\program files\java\jdk-8

Boolware is tested against the following Java distributions:
- Oracle Java
- Red Hat Java
- OpenJDK

Apache Tika
Download the Tika app (at least version 1.28 for 1.x-series or 2.4.1 for 2.x-series) from "https://tika.apache.org/download.html" and place the jar file in the "jar" directory in the Boolware installation directory.

Symbolic link to Apache Tika
In order to load Apache Tika, Boolware uses a symbolic link, tika.jar. It should be created in the "jar" directory in the Boolware installation directory. To create a symbolic link, do the following:

1. Start a command prompt/terminal window with administrator rights/root priviliges.
2. Change directory to the "jar" directory in the Boolware installation directory.
3. Type the following command:
   Windows
   mklink tika.jar tika-app-1.28.jar

   Linux
   ln –s tika-app-1.28.jar tika.jar

After the symbolic link has been created / changed, Boolware must be restarted.

Chapter 18
Boolware Website Indexing

Overview

Boolware uses Apache Nutch (1.x series) to crawl websites. Via Boolware Manager, you can add websites to crawl, start the crawl, monitor its progress, and finally index the content. When indexing content from websites, Apache Tika is used, which must be installed and configured. See Chapter 17, Text Extracting in Boolware.

Apache Nutch will be loaded into Boolware the first time you start a crawl or index a crawled website.

Boolware also use the "Boolware web crawler rest server" to communicate to and from Apache Nutch. Boolware communicates with the "Boolware webcrawler restserver" via https.

Prerequisites and requirements

In order for Boolware to crawl websites, the following is required:
1. Java 11 or later must be installed on computers using Boolware.
2. Apache Tika "Application" (see Chapter 17) must be available on computers using Boolware.
3. Boolware webcrawler restserver must be installed and configured on the same computer/s as running Apache Nutch.

Boolware webcrawler restserver requires the following:
1. Java 11 or later must be installed on computers with Boolware webcrawler restserver.
2. Apache Nutch 1.19 or later (1.x series) must be present on computers with Boolware webcrawler restserver.
3. JAR file for Spark-Java 2.9.3 (https server) or later must be present on computers with Boolware webcrawler restserver.
4. The Chrome browser must be present on computers with Boolware webcrawler restserver. **NOTE!** The browser must be installed for all users on the computer!
5. Selenium JAR files must be present on computers with Boolware webcrawler restserver.
6. A "web driver" for Chrome must be installed on computers with Boolware webcrawler restserver.
7. On Windows - Apache Hadoop application (hadoop.dll, winutils.exe) and Apache commons Daemon service application (prunsrv.exe) must be present on computers with Boolware webcrawler restserver.

Configuration in Boolware

Make sure Java is installed on the Boolware server. Then define the environment variable "BOOLWARE_JAVA_HOME", which should point to the Java version to be used (minimum Java 11). For example: BOOLWARE_JAVA_HOME=c:\program files\java\jdk-11.

Boolware is tested against the following Java distributions:
- Oracle Java
- Red Hat Java
- OpenJDK

Installation and configuration in Boolware Webcrawler restserver

Make sure that Java is installed on the computer where the Boolware webcrawler restserver is to run.

Apache Nutch
Download Apache Nutch binaries (at least version 1.19) from "https://nutch.apache.org/".

1. Expand Apache Nutch "binaries" files to any folder on the computer where the Boolware webcrawler restserver will run.
2. If installing on Windows you must also download 64-bit binaries for Apache Hadoop.
   a) Download "hadoop.dll" and "winutils.exe" binaries from e.g. "*https://github.com/cdarlint/winutils*".Choose the version corresponding to the Apache Nutch version you use (e.g. Apache Nutch 1.19 => Hadoop 3.1.3/3.2.0).
   b) Create a new folder structure inside it named "hadoop\bin" and place the downloaded Apache Hadoop binaries "hadoop.dll" and "winutils.exe" in the "bin" folder.

Spark-Java
Download "spark-core-2.9.3.jar" or later from "https://repo1.maven.org/maven2/com/sparkjava/spark-core". Save the file to the same directory where the Boolware webcrawler restserver is located.

Boolware webcrawler restserver
Windows uses an "Apache Commons Daemon" service management application, prunsrv.exe, to manage Boolware webcrawler restserver.
On Linux, standard Linux service manager are used to manage Boolware webcrawler restserver.
Boolware webcrawler restserver is by default configured to listen to the https port, ie. port 443. If another port is required, this must be changed in the configuration of the service.

Installation in Windows
If Boolware webcrawler restserver is to run on the same computer as Boolware, no copying is required. Otherwise, copy the "restserver" directory located in the "jar" directory of the directory where the Boolware server is installed to the computer where the Boolware webcrawler restserver is to run.

1. Download the Apache Commons Daemon "binaries" from "https://dlcdn.apache.org//commons/daemon/binaries/windows" and unzip the "prunsrv.exe" file located in the "amd64" directory in the zip file. Copy this file to the directory where the Boolware webcrawler restserver is located.
2. Open the file "install.boolware.webcrawler.restserver.bat.org" in a text editor. The file is located in the directory where the Boolware webcrawler restserver is located.
   a. Replace "*** NUTCH DIRECTORY ***" with the directory where Apache Nutch is located. Ex. C:\nutch
   b. Replace "*** BOOLWARE_JAVA_HOME ***" with the directory where Java is installed. Ex. C:\Program Files\Java\jdk-11
   c. Replace "*** BOOLWARE WEBCRAWLER RESTSERVER DIRECTORY ***" with the directory where the Boolware webcrawler restserver is located.
   d. If the Boolware webcrawler restserver is to listen on a port other than 443, this is changed by replacing the port with the desired port on the line "SET REST_PORT = 443".
3. Save and rename the file without the extension ".org".
4. Copy the file "jsoup-1.8.1.jar" from "…\plugins\protocol-httpclient" directory from the Nutch installation to the directory "…\restserver\bw-nutch-plugins\protocol-bw-httpclient" in the Boolware webcrawler restserver installation directory.
5. Start a command prompt (cmd) with administrator privileges and go to the directory where the Boolware webcrawler restserver is located.
6. Launch the file "install.boolware.webcrawler.restserver.bat" which will install/configure and start the service in Windows service manager.

Uninstall in Windows
To uninstall the service, start a command prompt (cmd) with administrator privileges and go to the directory where the Boolware webcrawler restserver is located and launch the file "uninstall.boolware.webcrawler.restserver.bat".

Installation in Linux
If Boolware web crawler residual server is to run on the same computer as Boolware, no copying is required. Otherwise, copy the "restserver" directory located in the "jar" directory of the directory where the Boolware server is installed to the computer where the Boolware webcrawler restserver is to run.

1. Open the "boolware.webcrawler.restserver.service.org" file in a text editor. The file is located in the directory where the Boolware webcrawler restserver is located.
    a. Replace "*** NUTCH DIRECTORY ***" with the directory where Apache Nutch is located.
    Ex. /usr/local/nutch
    b. Replace "*** BOOLWARE_JAVA_HOME ***" with the directory where Java is installed. Ex. /opt/jdk-11.0.1
    c. Replace "*** BOOLWARE WEBCRAWLER RESTSERVER DIRECTORY ***" with the directory where the Boolware webcrawler restserver is located.
    d. If the Boolware webcrawler restserver is to listen on a port other than 443, this is changed by replacing port 443 with the desired port on the line 'Environment="REST_PORT=443"'.
    e. If another user than *boolware* should run the Boolware webcrawler restserver, the following lines in the file must be changed:
    User=*boolware*
    Group=*boolware*
    Environment="HADOOP_USER=*boolware*"
2. Save and rename the file without the extension ".org".
3. Launch a root privilege terminal window and go to the directory where the Boolware webcrawler restserver is located.
4. Launch the "install.boolware.webcrawler.restserver.sh" file that will install, configure, and boot the service in Linux Service Manager.

Uninstall in Linux
To uninstall the service, launch a terminal root privileges window and go to the directory where Boolware webcrawler restserver is located and launch the file "uninstall.boolware.webcrawler.restserver.sh".

Selenium
JAR files for Selenium are by default included with Nutch installation and must be copied to the the computer and directory where the Boolware webcrawler restserver is located.

Linux, copy the files below from "…/plugins/lib-selenium" directory from the Nutch installation to the directory "…/jar/bw-nutch-plugins/lib-bw-selenium" in the Boolware webcrawler restserver installation directory.
Windows, copy the files below from "…\plugins\lib-selenium" directory from the Nutch installation to the directory "…\jar\bw-nutch-plugins\lib-bw-selenium" in the Boolware webcrawler restserver installation directory.

Note that the version numbers of the files below depend on the version of Nutch installed. The examples below are for Nutch 1.19.

- byte-buddy-1.8.15.jar
- commons-exec-1.3.jar
- guava-25.0-jre.jar
- okhttp-3.11.0.jar
- okio-1.14.0.jar
- selenium-api-3.141.5.jar
- selenium-chrome-driver-3.141.5.jar

- selenium-edge-driver-3.141.5.jar
- selenium-firefox-driver-3.141.5.jar
- selenium-ie-driver-3.141.5.jar
- selenium-java-3.141.5.jar
- selenium-opera-driver-3.141.5.jar
- selenium-remote-driver-3.141.5.jar
- selenium-safari-driver-3.141.5.jar
- selenium-support-3.141.5.jar

"Webdriver" for Chrome
Apache Nutch uses a "web driver" via Selenium to be able to retrieve URLs with dynamically generated web pages.

A "web driver" for Chrome can be downloaded from
https://chromedriver.chromium.org/downloads.
(Windows: *chromedriver_win32.zip*, Linux: *chromedriver_linux64.zip*).
Download the version that corresponds to the version of Chrome running on your computer.

Unpack the archive in any directory on the disk. The path to the file should be specified in the Nutch configuration in Boolware Manager in the "selenium.grid.binary" key. See the Boolware Manager Help file for more information.

After the installation and configuration, Boolware must be restarted.

Chapter 19
Export/Import of settings

Database export

When choosing database export in Boolware Manager you select the database to export from. The Manager will collect a list of all tables that are indexed and will show them to be able to select one or more tables for the export

Select target machine, which computer that holds a Boolware installation and to which registered database that will receive all the settings.

Boolware exports all settings for selected tables as working copies (work files). If necessary, files from the upper level (system) are added if they are not included in the export (.syn, .tes and .chr).
The flows and associated .txt files can also be exported, but is optional.
If a link table is exported, the source of the link table in the correct database must exist on the target machine.

When the export is complete, any database-level working copies created on the target machine will be applied when any table in the database is reindexed. Working copies at the table and field level will be applied when the table is reindexed.

Receiving Boolware will check all tables, descriptors and adjust some settings accordingly:

1. If receiving data source is a SQL data source e.g. a DBMS data source
   a. the table is a SQL table, e.g. a table from a DBMS data source
   b. if corresponding table does not exist, a check will be performed against the DBMS data source to create a descriptor with default setting if the table is found in the data source
   c. make sure the index settings is the same in the descriptor as in the work descriptor e.g. indexing is on
   d. make sure proper DBMS code is set, if the table is not a Boolware table
   e. make sure correct schema is set in the descriptor
   f. make sure the descriptor have the correct database owner, see below regarding link tables

2. Receiving data source is a Boolware data source
   a. make sure the index settings is the same in the descriptor as in the work descriptor e.g. indexing is on
   b. make sure proper DBMS code is set in the descriptor e.g. Boolware table
   c. make sure correct schema is set in the descriptor for a Boolware table
   d. make sure the descriptor have the correct database owner, see below regarding link tables
   e. if it is a webcrawler table, the restserver IP/port and all paths in the nutch configuration will be changed to the system settings values on the target machine

3. Received table is a link table, regards both data source types above
   a. if table exists but it is not a link table only settings will be applied and table type will not change to link table
   b. if table does not exist a link table will be created
   c. the database owner will be preserved in a link table

4. Export "Global settings"
   a. Send the system files for noise, stemming, synonyms, thesaurus and the global sysindex

5. Received database and tables included in the export are reviewed to clean up files belonging to the them (.syn/.chr/.tes etc.)
    a. If a working copy exists for a file and is identical to the committed one or its higher level one (if no committed file exist), the working copy will be removed.
    b. If a committed file is identical to its higher level file and a working copy does not exist on the higher level, the committed file will be removed

Export table settings

The following will take place when exporting settings for a table:

1. Boolware Manager will produce a list of files that belong to the selected table that will be exported
2. Boolware server will create a zip-file and add files from the list. If needed, some files from nearest level above will be added if they don't exist: .syn, .tes, .fuzzy, .case, .gen, .phon, e01.conf, .e02.conf, .e03.conf.
3. Boolware server will add the file <db>.settings.conf to the zip file
4. The zip file is created in the Boolware TEMP-directory and is fetched by Boolware Manager and is stored locally at specified location. The zip file will then be removed from the server TEMP directory

Import table settings

The following will take place during a table settings import in Boolware:

1. The Boolware Manager will provide a sub-directory name to Boolware server to be created and send all the files stored in the zip file to that directory
2. All files from the zip file will be changed to work copies.
3. The server will then check that the table to be imported exist in the proper database, otherwise the import will stop with an error message
4. The import will create e01.utf8.conf files on proper field- and table level if they don't exist beside the corresponding e01.conf.
5. Load the target descriptor and load the import descriptor
6. Merge settings from the import descriptor into the target descriptor and save the target descriptor as a work copy, settings like synchronization, table type etc. will be preserved
7. Boolware file system tables and Boolware web crawler tables can only be imported to the same table types
8. Web crawler settings for Nutch, the restserver IP/port and paths in the nutch configuration will be changed to the system settings values
9. Import table settings from the file <db>.settings.conf
10. Copy the files from the sub-directory to the target directory as work copies (.work).
11. The import will then review all files belonging to the table to clear unnecessary files. If a working copy exists for a file and is identical to the committed one or its higher level file (if no committed file exist), the working copy will be removed. If a committed file is identical to its higher level file and a working copy does not exist on the higher level, the committed file will be removed.
12. Remove all files from the temporary sub-directory

The working copies created during the import will be applied when the table is reindexed.

Chapter 20
Backup/Restore

Backup/Restore

A backup of tables in one or more databases in Boolware is done using Boolware Manager or bwc. The backup files are placed in the directory configured in Boolware. A backup can also be scheduled in Boolware Manager.

Restore is done using Boolware Manager or bwc.

Read more about Backup/Restore in the Boolware Manager help file.