MonetDB Reference Manual

Version 5.0

The MonetDB Development Team

Last updated: Sept 9, 2007

Disclaimer The reference manual and underlying source code base are still under development. This may lead to incomplete and inconsistencies descriptions, for which we apologize in advance. You can help improving the manual using the MonetDB mailing list.

Table of Contents

1 General Introduction

The MonetDB reference manual serves as the primary entry point to locate information on its functionality, system architecture, services and best practices on using its components.

The manual is produced from a Texinfo framework file, which collects and organizes bits-and-pieces of information scattered around the many source components comprising the MonetDB software family. The Texinfo file is turned into a HTML browse-able version using *makeinfo* program. The PDF version can be produced using *pdflatex*. Alternative formats, e.g., XML and DocBook format, can be readily obtained from the Texinfo file.

The copyright(2007) on the MonetDB software, documentation and logo is owned by CWI. Other trademarks and copyrights referred to in this manual are the property of their respective owners.

1.1 Intended Audience

The MonetDB reference manual is aimed at application developers and researchers with an intermediate level exposure to database technology, its embedding in host environments, such as C, Perl, Python, PHP, or middleware solutions based on JDBC and ODBC.

The bulk of the MonetDB reference manual deals with the techniques deployed in the back-end for the expert user and researcher. Judicious use of the programming interfaces and database kernel modules for domain specific tasks lead to high-performance solutions. The grand challenge for the MonetDB development team is to assemble a sufficient and orthogonal set of partial solutions to accommodate a wide variety of front-ends.

Feedback on the functionality provided is highly appreciated, especially when you embark on a complex programming project. If the envisioned missing functionality is generally applicable it makes sense to contribute it to the community. Share your comments and thoughts through the MonetDB mailing list held at SourceForge.

1.2 How to read this manual

The reference manual covers a lot of ground, which at first reading may be slightly confusing. The material is presented in a top-down fashion. Starting at installing the system components, SQL & XQuery and the application interface layer, it discusses the MAL software stack at length. Forward references are included frequently to point into the right direction for additional information.

If you are interested in technical details of the MonetDB system, you should start reading ⟨undefined⟩ [Design Overview], page ⟨undefined⟩. Two reading tracks are possible. The Chapter 5 [MAL Reference], page 97 language and subsequent sections describe the abstract machine and MAL optimizers to improve execution speed. It is relevant for a better understanding of the query processing behavior and provides an entry point to built new languages on top of the database kernel. The tutorial on SQL to MAL compilation provides a basis for developing your own language front-end.

The second track, The Inner Core describes the datastructures and operations exploited in the abstract machine layer. This part is essential for developers to aid in bug fixing and to extend the kernel with new functionality. For most readers, however, it can be skipped without causing problems to develop efficient applications.

1.3 Features and Limitations

In this section we give a short overview of the key features to (not) consider the MonetDB product family. In a nutshell, its origin in the area of data-mining and data-warehousing makes it an ideal choice for high volume, complex query dominant applications. MonetDB was not designed for high-volume secure OLTP settings initially.

It is important to recognize that the MonetDB language interfaces are primarily aimed at experienced system programmers and administrators. End-users are advised to use any of the open-source graphical SQL workbenches to interact with the system.

1.3.1 When to consider MonetDB ?

A high-performance database management system. MonetDB is an easy accessible open-source DBMS for SQL-[XQuery-]based applications and database research projects. Its origin goes back over a decade, when we decided that the database hotset - the part used by the applications - can be largely held in main-memory or where a few columns of a broad relational table are sufficient to handle a request. Further exploitation of cache-conscious algorithms proved the validity of these design decisions.

A multi-model system. MonetDB supports multiple query language front-ends. Aside from its proprietary language, called the MonetDB Assembly Language (MAL), it aims at ANSI SQL-2003 and W3C XQuery with update facilities. Their underlying logical data model and computational scheme differs widely. The system is designed to provide a common ground for both languages and it is prepared to support languages based on yet another data model or processing paradigm.

A binary-relation database kernel. MonetDB is built on the canonical representation of database containers, called Binary Association Tables (BATs). The datastructures are geared towards efficient representation when they mimic an n-ary relational scheme.

This led to an architecture where the traditional page-pool is replaced by one with a much larger granularity based on BATs. They are sizeable entities -up to hundreds of megabytes- swapped into memory upon need. The benefit of this approach has been shown in numerous papers in the scientific literature.

A broad spectrum database system. MonetDB is continuously developed to support a broad application field. Although originally developed for Analytical CRM products, it is now being used at the low-end scale as an embedded relational kernel and projects are underway to tackle the huge database problems encountered in scientific databases, e.g. astronomy.

An extendable database system. MonetDB has been strongly influenced by the scientific experiments to understand the interplay between algorithms and hardware features. It has turned MonetDB into an extensible database system. It proves valuable in those cases where an application specific and critical component makes all the difference between slow and fast implementation.

An opensource software system. MonetDB has been developed over many years of research at CWI, whose charter ensures that results are easily accessible to others. Either through publications in the scientific domain or publication of the software components involved. The MonetDB mailing list is the access point to a larger audience for advice. A subscription to the mailing list helps the developer team to justify their hours put into MonetDB's development and maintenance.

1.3.2 When not to consider MonetDB ?

There are several areas where MonetDB has not yet built a reputation. They are the prime candidates for experimentation, but also areas where application construction may become risky. Mature products or commercial support may then provide a short-term solution, while MonetDB programmers team works on filling the functional gaps. The following areas should be considered with care:

Persistent object caches. The tendency to develop applications in Java and C based on a persistent object model, is a no-go area for MonetDB. Much like other database engines, the overhead of individual record access does not do justice to the data structures and algorithms in the kernel. They are chosen to optimize bulk processing, which always comes at a price for individual object access.

Nevertheless, MonetDB has been used from its early days in a commercial application, where the programmers took care in maintaining the Java object-cache. It is a route with great benefits, but also one where sufficient manpower should be devoted to perform a good job.

High-performance financial OLTP. MonetDB was originally not designed for highly concurrent transaction workloads. It was decided to make ACID hooks explicit in the query plans generated by the front-end compilers. Given the abundance of main memory nowadays and the slack CPU cycles to process database requests, it may be profitable to consider serial execution of all OLTP transactions.

The SQL implementation provides full transaction control and recovery.

Security. MonetDB has not been designed with a strong focus on security. The major precautions have been taken, but are incomplete when access to the hosting machine is granted or when direct access is granted to the Monet Assembly Language features. The system is preferably deployed in a sand-boxed environment where remote access is encapsulated in a dedicated application framework.

Scaling over multiple machines. MonetDB does not provide a centralized controlled, distributed database infrastructure yet. Instead, we move towards an architecture where multiple autonomous MonetDB instances are joining together to process a large and distributed workload.

In the multimedia applications we have exploited successfully the inherent data parallelism to speedup processing and reduce the synchronization cost. The underlying platforms were Linux-based cluster computers with sizeable main memories.

1.3.3 What are key features of MonetDB

The list below provides a glimpse on the technical characteristics and features of the MonetDB software packages.

The software characteristics for the MonetDB packages are:

- The kernel source code is written in ANSI-C and POSIX compliant.
- The application interface libraries source code complies with the latest language versions.
- The source code is written in a literate programming style, to stimulate proximity of code and its documentation.

- The source code is compiled and tested on many platforms with different compiler options to ensure portability.
- The source code is based on the GNU toolkit, e.g. Automake, Autoconf, and Libtool for portability.
- The source code is heavily tested on a daily basis, and scrutinized using the Valgrind toolkit.

The heart is the MonetDB server, which comes with the following innovative features.

- A fully decomposed storage scheme using memory mapped files.
- It supports scalable databases, 32- and 64-bit platforms.
- Connectivity is provided through TCP/IP sockets and SSH on many platforms.
- Index selection, creation and maintenance is automatic.
- The relational operators materialize their results and are self-optimizing.
- The operations are cache- and memory-aware with supreme performance.
- The database back-end is multi-threaded and guards a single physical database instance.

1.3.4 Size Limitations for MonetDB

The maximal database size supported by MonetBD depends on the underlying processing platform, e.g., a 32- or 64-bit processor, and storage device, e.g., the file system and disk raids.

The number of columns per tables is practically unlimited. The storage space limitation depends only on the maximal file size. For each column is mapped onto a file, whose limit is dictated by the operating system and hardware platform.

The number of concurrent user threads is a configuration parameter. Middleware solutions are adviced to serialize access to the database when large number of users are expected to access the database.

1.4 A Brief History of MonetDB

The Dark Ages [1979-1992] The development of the MonetDB software family goes back as far as the early eighties when the first relational kernel, called Troll, was delivered to a larger audience. It was spread over ca 1000 sites world-wide and became part of a software case-tool until the beginning of the nineties. None of the code of this system has survived, but several ideas and experiences on how to obtain a fast kernel by simplification and explicit materialization found their origin during this period.

The second part of the eighties was spent on building the first distributed main-memory database system in the context of the national project PRISMA. A fully functional system of 100 processors and a wealthy 1GB of main memory showed the road to develop database technology from a different perspective.

The Early Days [1993-1995] Immediately after the PRISMA project was termed dead, a new database kernel based on Binary Association Tables (BATs) was laid out. The original target was to aim for better support of scientific databases with their then archaic file structures.

The Data Distilleries Era [**1996-2003**] The datamining projects running as of 1993 called for better database support. It culminated in the spin-off Data Distilleries, which based their analytical customer relationship suite on the power provided by the early MonetDB implementations. In the years following, many technical innovations were paired with strong industrial maturing of the software base. Data Distilleries became a subsidiary of SPSS in 2003 and its development activity was shifted to Chicago in 2007.

The Open-Source Challenge [**2003-2007**] Moving MonetDB Version 4 into the open-source field required a large number of extensions to the code base. It became utmost important to support a mature implementation of the SQL-99 standard, and the bulk of application programming interfaces (PHP,JDBC,Perl,ODBC). The culprit of this activity was the first official release in 2004 and the release of the XQuery front-end in 2005.

The Road Ahead [**2007-** This manual describes the MonetDB Version 5 release, the result of a multi-year activity to clean up the software stack and to better support both simple and complex database requests.

The Future New versions in the MonetDB software family are under development. Extensions and renovation of the kernel are studied in the X100 project. Its Volcano-style interpreter aims to provide performance in I/O-dominant and streaming settings using vectorized processing and Just-In-Time (de)compression.

The scene of distributed database is (again) addressed in the Armada project, but not using the traditional centralized administration focus. Instead the Armada project seeks the frontiers of autonomous database systems, which still provide a coherent functional view to its users. In its approach it challenges many dogmas in distributed database technology, such as the perspective on global consistency, the role of the client in managing the distributed world, and the way resources are spread.

The MonetDB software framework provides a rich setting to pursue these alleys of database research. We hope that many may benefit from our investments, both research and business wise.

1.5 Manual Generation

The MonetDB code base is a large collection of files, scattered over the system modules. Each source file is written in a literal programming style, which physically binds documentation with the relevant code sections. The utility program Mx processes the files marked *.mx to extract the code sections for system compilation or to prepare for a pretty printed listing.

The reference manual is based on *Texinfo* formatted documentation to simplify generation for different rendering platforms. The components for the reference manual are extracted by

```
Mx -i -B -H1 <filename>.mx
```

which generates the file <filename>.bdy.texi. These pieces are collected and glued together in a manual framework, running *makeinfo* to produce the desired output format. The *Texinfo* information is currently limited to the documentation, it could also be extended to process the code.

A printable version of an *.mx file can be produced using the commands:

```
Mx  <filename>.mx
pdflatex <filename>.tex
```

1.5.1 Conventions and Notation

The typographical conventions used in this manual are straightforward. Monospaced text is used to designate names in the code base and examples. *Italics* is used in explanations to indicate where a user supplied value should be substituted.

Snippets of code are illustrated in small caps font. The interaction with textual client interfaces uses the default prompt-setting of the underlying operating system.

Keywords in the MonetDB interface languages are case sensitive; SQL keywords are not case sensitive. No distinction is made in this manual.

1.5.2 Additional Resources

Although this reference manual aims to be complete for developing applications with MonetDB, it also depends on additional resources for a full understanding.

This reference manual relies on external documentation for the basics of its query languages SQL, XQuery, its application interfaces, PHP, Perl, Pyhton, and its middleware support, JDBC and ODBC. Examples are used to illustrate their behaviour in the context of MonetDB only. The resource locations identified below may at times proof valuable.

Perl DBI	http://www.perl.org/
PHP5	http://www.php.net/
Python	http://www.python.org/
XQuery	http://www.w3c.org/TR/xquery/

The primary source for additional information is the MonetDB website, http://monetdb.cwi.nl/, and the code base itself. Information on the background of its architecture can be found in the library of scientific publications.

1.6 Downloads and Installation

For most people a binary distribution package is sufficient. The prime decision is to select either the SQL or XQuery product line. They currently rely on different and incompatible back-end servers.

The binary distribution contains all components for MonetDB application development, i.e. a back-end server, an SQL or XQuery compiler, and the client libraries. These components are packaged conveniently for several platforms in the download section at SourceForge. It can be installed in a private directory or in the Linux/Windows compliant default folder location.

The Developers distribution is meant for source experimentation and functional enhancements. A "stable" version is prepared regularly. It means that special care has been taken to assure that errors reported during the nightly builds have been solved on the platforms of interest. Major bug fixes are also applied to the latest stable version, while functional enhancements are kept for the next release or the daily builds.

The Experts distribution is meant for MonetDB kernel software developers only. They should have a clear understanding of Linux development tools, e.g. automake, config, CVS, and team-based software development and the interdependencies of the MonetDB components.

If you encounter errors during the installation, please have a look at the MonetDB mailing list for common errors and some advice on how to proceed.

1.6.1 Developers Distribution

Developers interested in source code to be linked with the MonetDB libraries or running on non-supported platforms may use the nightly builds tarballs to assemble a working system. Alternatively, they can check out the latest STABLE of CURRENT version from CVS.

The easiest way is use the All-In-One scripts provided. A quick installation based on the nightly tarballs and super source tarball is supported by the monetdb-install.sh script. Check it out and run it in an empty directory should be sufficient on most Linux platforms to get going. It takes about 10-20 minutes to install and compile from scratch on a modern PC. The nightly build source distribution comes with the complete test-bench to assure that changes do not affect (in as far as they get tested) its stability. A single stable release is maintained for external users while we concurrently work on the next release. Older versions are not actively maintained by the development team.

1.6.2 Experts

The experts may want more control then provided by the developer distribution support. Set up of a fully functional system requires downloading and installation of the latest packages from SourceForge. The compatibility table below illustrates the packages in the CVS repository.

	MonetDB/SQL	MonetDB/XQuery	Clients
buildtools	x	x	x
clients	x	x	x
MonetDB	x	x	
MonetDB4		x	
MonetDB5	x		
SQL	x		
Pathfinder		x	

Thanks to the GNU AUTOCONF and AUTOMAKE tools, the MonetDB software runs on a wide variety of hardware/software platforms. The MonetDB development team uses many of the platforms available to perform automated nightly regression testing. For more details see The Test Web.

The MonetDB code base -with daily builds available for users preferring living at the edge- evolves quickly. Application developers, however, may tune into the MonetDB mailing list to be warned when a major release has become available, or when detected errors require a patch.

1.6.3 Linux Installation

This section helps you compile and install the MonetDB suite from scratch on Unix-like systems (this includes of course Linux, but also MacOS X and Cygwin). It is meant to be used when you want to compile and install from CVS source. When you use the prepared tar balls, some of the steps described here should be skipped.

In case you prefer installing a pre-compiled binary distribution, please check out the binary distribution from the download area.

This document assumes that you are planning on compiling and installing MonetDB on a Unix-like system (e.g., Linux, IRIX, Solaris, AIX, Mac OS X/Darwin, or CYGWIN).

1.6.3.1 The Suite

The MonetDB software suite consists of the following parts which need to be built in the correct order:

'`buildtools`'
: Tools used only for building the other parts of the suite. These tools are only needed when building from CVS. When building from the source distribution (i.e. the tar balls), you do not need this.

'`MonetDB`' Fundamental libraries used in the other parts of the suite.

'`clients`' Libraries and programs to communicate with the server(s) that are part of the suite.

'`MonetDB4`'
: The MIL-based server. This is required if you want to use XML/XQuery (pathfinder), and can be used with SQL.

'`MonetDB5`'
: The MAL-based server. This can be used with and is recommended for SQL.

'`pathfinder`'
: The XML/XQuery engine built on top of MonetDB4.

'`sql`' The SQL server built on top of (targeted on) either MonetDB4 or MonetDB5.

MonetDB4 and MonetDB5 are the basic database engines. One or the other is required, but you can have both. Pathfinder currently needs MonetDB4, sql can run on both MonetDB4 and MonetDB5 (the latter is recommended).

The order of compilation and installation is important. It is best to use the above order (where pathfinder and sql can be interchanged) and to configure-make-make install each package before proceeding with the next.

1.6.3.2 Prerequisites

'`CVS`' You only need this if you are building from CVS. If you start with the source distribution from SourceForge you don't need CVS.

You need to have a working CVS. For instructions, see the SourceForge documentation and look under the heading CVS Instructions.

'`Python`' MonetDB uses Python (version 2.0.0 or better) during configuration of the software. See http://www.python.org/ for more information. (It must be admitted, version 2.0.0 is ancient and has not recently been tested, we currently use 2.4 and newer.)

'`autoconf/automake/libtool`'
: MonetDB uses GNU autoconf (>= 2.57) and automake (>= 1.5) during the BOOTSTRAP phase, and libtool (>= 1.4) during the MAKE phase. autoconf and automake are not needed when you start with the source distribution.

'`standard software development tools`'
: To compile MonetDB, you also need to have the following standard software development tools installed and ready for use on you system:

- a C compiler (e.g. GNU's GCC);
- GNU MAKE (GMAKE) (native MAKE on, e.g., IRIX and Solaris usually don't work).

The following are not needed when you start with the source distribution:

- a C++ compiler (e.g. GNU's G++);
- a lexical analyzer generator (e.g., LEX or FLEX);
- a parser generator (e.g., YACC or BISON).

The following are optional. They are checked for during configuration and if they are missing, the feature is just missing:

- swig
- perl
- php

'`buildtools (Mx, mel, autogen, and burg)`'
These tools are not needed when you start with the source distribution.

Before building any of the other packages from the CVS sources, you first need to build and install the buildtools. Check out buildtools with

```
cvs -d:pserver:anonymous@monetdb.cvs.sourceforge.net:/cvsroot/monetdb checkout b
```

and follow the instructions in the README file, then proceed with MonetDB. For this step only you need the C++ compiler.

'`libxml2`' The XML parsing library libxml2 is only used by XML/XQuery (pathfinder). The library is used for:

1. the XML Schema import feature of the Pathfinder compiler, and
2. the XML document loader (runtime/shredder.mx).

If libxml2 is not available on your system, the Pathfinder compiler will be compiled without XML Schema support. The XML document loader will not be compiled at all in that case. Current Linux distributions all come with libxml2.

A rough estimate of the packages space:

```
==========  =======  =======  =======
 Package    Source   Build    Install
==========  =======  =======  =======
buildtools  1.5 MB   8 MB     2.5 MB
MonetDB     2 MB     21 MB    4 MB
clients     9 MB     25 MB    10 MB
MonetDB4    35.5 MB  50 MB    14 MB
MonetDB5    26 MB    46 MB    12 MB
sql         100 MB   22.5 MB  8 MB
pathfinder  130 MB   43 MB    12 MB
==========  =======  =======  =======
```

Some of the source packages are so large because they include lots of data for testing purposes.

1.6.3.3 Getting the Software

There are two ways to get the source code:

1. checking it out from the CVS repository on SourceForge;
2. downloading the pre-packaged source distribution from SourceForge.

The following instructions first describe how to check out the source code from the CVS repository on SourceForge; in case you downloaded the pre-packaged source distribution, you can skip this section and proceed to BOOTSTRAP, CONFIGURE and MAKE.

1.6.3.4 CVS checkout

This command should be done once. It records a password on the local machine to be used for all subsequent CVS accesses with this server.

```
cvs -d:pserver:anonymous@monetdb.cvs.sourceforge.net:/cvsroot/monetdb login
```

Just type RETURN when asked for the password.

Then get the software by using the command:

```
cvs -d:pserver:anonymous@monetdb.cvs.sourceforge.net:/cvsroot/monetdb checkout \
buildtools MonetDB clients MonetDB4 MonetDB5 pathfinder sql
```

This will create the named directories in your current working directory. Then first follow the instructions in BUILDTOOLS/README before continuing with the others. Naturally, you don't need to check out packages you're not going to use.

Also see the SourceForge documentation for more information about using CVS.

1.6.3.5 Bootstrap, Configure and Make

Before executing the following steps, make sure that your shell environment (especially the variables PATH. LD_LIBRARY_PATH, and PYTHONPATH) is set up so that the tools listed above can be found. Also, set up PATH to include the *prefix*/bin directory where *prefix* is the prefix is where you want everything to be installed, and set up PYTHONPATH to include the *prefix*/lib/*python2.X* directory where *python2.X* is the version of Python being used. It is recommended to use the same *prefix* for all packages. Only the *prefix*/lib/*python2.X* directory for buildtools is needed in PYTHONPATH.

In case you checked out the CVS version, you have to run BOOTSTRAP first; in case you downloaded the pre-packaged source distribution, you should skip BOOTSTRAP and start with CONFIGURE (see CONFIGURE).

For each of the packages do all the following steps (bootstrap, configure, make, make install) *before* proceeding to the next package.

1.6.3.6 Bootstrap

This step is only needed when building from CVS.

In the top-level directory of the package type the command (note that this uses AUTOGEN.PY which is part of the BUILDTOOLS package — make sure it can be found in your $PATH):

```
./bootstrap
```

1.6.3.7 Configure

Then in any directory (preferably a *new, empty* directory and *not* in the MonetDB top-level directory) give the command:

```
.../configure [<options>]
```

where ... is replaced with the (absolute or relative) path to the MonetDB top-level directory.

The directory where you execute CONFIGURE is the place where all intermediate source and object files are generated during compilation via MAKE.

By default, MonetDB is installed in /USR/LOCAL. To choose another target directory, you need to call

```
.../configure --prefix=<prefixdir> [<options>]
```

Some other useful CONFIGURE options are:

```
--enable-debug          enable full debugging default=off
--enable-optimize       enable extra optimization default=off
--enable-warning        enable extended compiler warnings default=off
--enable-profile        enable profiling default=off
--enable-instrument     enable instrument default=off
--with-mx=<Mx>          which Mx binary to use (default: whichever
                        Mx is found in your PATH)
--with-mel=<mel>        which mel binary to use (default: whichever
                        mel is found in your PATH)
--enable-bits=<#bits>   specify number of bits (32 or 64)
                        default is compiler default
--enable-oid32          use 32-bit OIDs on 64-bit systems default=off
```

You can also add options such as CC=<COMPILER> to specify the compiler and compiler flags to use.

Use CONFIGURE –HELP to find out more about CONFIGURE options.

The –WITH-MX and –WITH-MEL options are only used when configuring the sources as retrieved through CVS.

1.6.3.8 Make

In the same directory (where you called CONFIGURE) give the command

```
make
```

to compile the source code. Please note that parallel make runs (e.g. MAKE -J2) are currently known to be unsuccessful.

1.6.3.9 Testing the Build

This step is optional and only relevant for the packages MonetDB4, MonetDB5, pathfinder, and sql.

If MAKE went successfully, you can try

```
make check
```

This will perform a large number of tests, some are unfortunately still expected to fail, but most should go successfully. At the end of the output there is a reference to an HTML file which is created by the test process that shows the test results.

1.6.3.10 Install

Give the command

```
make install
```

By default (if no –PREFIX option was given to CONFIGURE above), this will install in /USR/LOCAL. Make sure you have appropriate privileges.

1.6.3.11 Testing the Installation

This step is optional and only relevant for the packages MonetDB4, MonetDB5, pathfinder, and sql.

Make sure that *prefix*/bin is in your PATH. Then in the package top-level directory issue the command

```
Mtest.py -r
```

This should produce much the same output as MAKE CHECK above, but uses the installed version of MonetDB.

You need write permissions in part of the installation directory for this command: it will create subdirectories VAR/DBFARM and TESTS.

1.6.3.12 Usage

The MonetDB4 and MonetDB5 engines can be used interactively or as a server. The XQuery and SQL back-ends can only be used as servers.

To run MonetDB4 interactively, just run:

```
Mserver
```

To run MonetDB5 interactively, just run:

```
mserver5
```

The disadvantage of running the systems interactively is that you don't get readline support (if available on your system). A more pleasant environment can be had by using the system as a server and using MCLIENT to interact with the system. For MonetDB4 use:

```
Mserver --dbinit 'module(mapi); mil_start();'
```

When MonetDB5 is started as above, it automatically starts the server in addition to the interactive "console".

In order to use the XQuery back-end, which is only available with MonetDB4, start the server as follows:

```
Mserver --dbinit 'module(pathfinder);'
```

If you want to have a MIL server in addition to the XQuery server, use:

```
Mserver --dbinit 'module(pathfinder); mil_start();'
```

In order to use the SQL back-end with MonetDB4, use:

```
Mserver --dbinit 'module(sql_server);'
```

If you want to have a MIL server in addition to the SQL server, use:

```
Mserver --dbinit 'module(sql_server); mil_start();'
```

In order to use the SQL back-end with MonetDB5, use:

```
mserver5 --dbinit 'include sql;'
```

Once the server is running, you can use MCLIENT to interact with the server. MCLIENT needs to be told which language you want to use, but it does not need to be told whether you're using MonetDB4 or MonetDB5. In another shell window start:

```
mclient -l<language>
```

where *language* is one of MIL, MAL, SQL, or XQUERY. If no -L option is given, MIL is the default.

With MCLIENT you get a text-based interface that supports command-line editing and a command-line history. The latter can even be stored persistently to be re-used after stopping and restarting MCLIENT.

```
mclient --help
```

for details.

At the MCLIENT prompt some extra commands are available. Type a single question mark to get a list of options. Note that one of the options is to read input from a file using <. This interferes with XQuery syntax. This is a known bug.

1.6.3.13 Troubleshooting

BOOTSTRAP fails if any of the requisite programs cannot be found or is an incompatible version.

BOOTSTRAP adds files to the source directory, so it must have write permissions.

During BOOTSTRAP, the following might occur:

```
Remember to add 'AC_PROG_LIBTOOL' to 'configure.in'.
You should add the contents of '/usr/share/aclocal/libtool.m4' to 'aclocal.m4'.
configure.in:37: warning: do not use m4_patsubst: use patsubst or m4_bpatsubst
configure.in:104: warning: AC_PROG_LEX invoked multiple times
configure.in:334: warning: do not use m4_regexp: use regexp or m4_bregexp
automake/aclocal 1.6.3 is older than 1.7.
Patching aclocal.m4 for Intel compiler on Linux (icc/ecc).
patching file aclocal.m4
Hunk #1 FAILED at 2542.
1 out of 1 hunk FAILED -- saving rejects to file aclocal.m4.rej
patching file aclocal.m4
Hunk #1 FAILED at 1184.
Hunk #2 FAILED at 2444.
Hunk #3 FAILED at 2464.
3 out of 3 hunks FAILED -- saving rejects to file aclocal.m4.rej
```

For some technical reasons, it is hard to completely avoid them. However, it is usually safe to ignore them and simply proceed with the usual compilation procedure. Only in case the subsequent CONFIGURE or MAKE fails, these warning might have to be taken more seriously. In any case, you should include the BOOTSTRAP output whenever you report (see 'Reporting Problems') compilation problems.

CONFIGURE will fail if certain essential programs cannot be found or certain essential tasks (such as compiling a C program) cannot be executed. The problem will usually be clear from the error message.

If CONFIGURE cannot find package XYZ, it is either not installed on your machine, or it is not installed in places that CONFIGURE searches (i.e., /USR, /USR/LOCAL). In the first case, you need to install package XYZ before you can CONFIGURE, MAKE, and MAKE INSTALL MonetDB. In the latter case, you need to tell CONFIGURE via –WITH-XYZ=<DIR> where to find package XYZ on your machine. CONFIGURE then looks for the header files in <DIR>/include, and for the libraries in <DIR>/lib.

In case one of BOOTSTRAP, CONFIGURE, or MAKE fails — especially after a CVS UPDATE, or after you changed some code yourself — try the following steps (in this order; if you are using the pre-packaged source distribution, you can skip steps 2 and 3):

In case you experience problems after a CVS UPDATE, first make sure that you used "CVS UPDATE -DP" (or have a line UPDATE -DP in your ~/.CVSRC); -D ensures that cvs checks out directories that have been added since your last CVS UPDATE; -P removes directories that have become empty, because all their file have been removed from the cvs repository. In case you did not use CVS UPDATE -DP, re-run CVS UPDATE -DP, and remember to always use CVS UPDATE -DP from now on (or simply add a line UPDATE -DP to your ~/.CVSRC)!

1. In case only MAKE fails, you can try running:

   ```
   make clean
   ```

   in your build directory and proceed with step 5; however, if MAKE then still fails, you have to re-start with step 1.
2. Clean up your whole build directory (i.e., the one where you ran CONFIGURE and MAKE) by going there and running:

   ```
   make maintainer-clean
   ```

   In case your build directory is different from your source directory, you are advised to remove the whole build directory.
3. Go to the top-level source directory and run:

   ```
   ./de-bootstrap
   ```

   and type Y when asked whether to remove the listed files. This will remove all the files that were created during BOOTSTRAP. Only do this with sources obtained through CVS.
4. In the top-level source directory, re-run:

   ```
   ./bootstrap
   ```

   Only do this with sources obtained through CVS.
5. In the build-directory, re-run:

   ```
   configure
   ```

   as described above.
6. In the build-directory, re-run:

   ```
   make
   make install
   ```

   as described above.

If this still does not help, please contact us.

1.6.3.14 Reporting Problems

Bugs and other problems with compiling or running MonetDB should be reported using the bug tracking system at SourceForge (preferred) or emailed to monet@cwi.nl; see also http://monetdb.cwi.nl/Development/Bugtracker/index.html. Please make sure that you give a *detailed* description of your problem!

1.6.4 Windows Installation

This section helps you to install and compile MonetDB on a Windows system using the Microsoft Visual compiler.

1.6.4.1 Prerequisites

- *CVS* You need to have a working CVS. Several solutions are available. We use internally WinCVS and CVS under Cygwin. For general information about the SourceForge repository, see docman and look under the heading CVS Instructions. Pointers to CVS implementations for Windows can be found at e.g.: http://www.cvshome.org/cyclic/cvs/windows.html http://www.wincvs.org/ http://www.componentsoftware.com/products/CVS/
- *Python* MonetDB uses Python (version 2.0.0 or better) during configuration of the software. See python.org for more information.
- *Pthreads* Pthreads for Win32 should be installed to C:\Pthreads, otherwise you need to patch the "PTHREAD = C:\Pthreads" line in "NT\rules.msc" according to your setup. Newer versions of Pthreads seem to have the "include" &amp; "lib" directory not in C:\Pthreads, but in C:\Pthreads\prebuilt. In that case, you either have to move the "include" & "lib" directory from C:\Pthreads\prebuilt to C:\Pthreads, or set "PTHREAD = C:\Pthreads\prebuilt" in "NT\rules.msc".
- *UnxUtils* UnxUtils for Win32 must be installed in the root directory ("\", i.e., without the default "\UnxUtils\" prefix) of the drive where you want to compile MonetDB. The same drive is required, as the UnxUtils do not know about drive letters, and hence absolute paths start with "\" (i.e., without a leading "C:", "D:", ...). The default "\UnxUtils\" prefix has to be omitted, as otherwise the UnxUtils' "bison" does not find its own "bison.simple" file, which it expects to be in "/usr/share/bison/bison.simple" (aka. "\usr\share\bison\bison.simple").

  In addition to some tools used by Mtest.py for testing MonetDB (see below), UnxUtils provide the lexical analyzer generator "flex" and the parser generator "bison", which are required to compile MonetDB. Note: You have to install the UnxUtils using the original .zip file. The third-party executable installer, chooses another default installation directory that doesn't seem to work...
- *Microsoft Visual C++* You need Microsoft Visual C++ 5 or higher. Notice that if you do not own Microsoft Visual C++, you can still compile on Windows using the Cygnus Posix-emulation environment CYGWIN In that case, you should follow the standard instructions in the file 'HowToStart'. Alternatively, it might be possible to use the free WIN32-GNU compilers as a drop-in replacement for the Microsoft tools, but we have not tried this yet.
- *disk space* After downloading, MonetDB takes up about 40 MB of disk space; compilation adds another 70 MB. Testing takes about 45 MB, and the installation about 20 MB (not including any databases).

1.6.4.2 Optional Software

- PHP Download the Windows binaries in a zip package (i.e. not the Windows installer) and the source package of PHP 5 from http://www.php.net/. Unzip the binaries into e.g. "C:\php-5" (for PHP-5, the zip file does not contain a top-level directory, so create a new directory, e.g. "C:\php-5", and unzip the files there–for PHP-4, the zip file does contain a top-level directory, so you can unzip directly into "C:\"). In any case, make sure there are no spaces in the path chosen. Also extract the sources somewhere, e.g. in a directory parallel to MonetDB. If you don't use "C:\php-5" for the binaries or you don't extract the sources into "php-5.1.4" in the same directory where MonetDB is extracted, you will have to edit MonetDB's "NT\rules.msc" to change the cariables "PHP_SRCDIR" and "PHP_INSTDIR" to reflect this. Make sure there are no spaces in these two path names. Also note that the paths are only used from inside the "NT\src\mapi\clients\php" folder, so if they are relative, they have to be relative from there. In order to get MonetDB to compile with these sources, it was necessary to remove the line "#include "arpa/inet.h"" from the file "main\php_network.h". When compiling MonetDB, add the flag "HAVE_PHP=1" to the "nmake" command line. Note that only PHP version 5.1.4 has been tried recently.
- Pear: Pear is the PHP Extension and Application Repository. At the MonetDB side of things, nothing more than what is described for PHP needs to be done. PEAR support is automatically enabled.
- Java: The Java SDK (JDK) can be found at http://java.sun.com/. After installing, make sure that the directory with java.exe and javac.exe are in your "PATH" environment variable, and then compile MonetDB with the additional flag "HAVE_JAVA=1" on the "nmake" command line.

1.6.4.3 Getting the Software

This command should be done once. It records a password on the local machine to be used for all subsequent CVS accesses with this repository.

```
cvs -d:pserver:anonymous@monetdb.cvs.sourceforge.net:/cvsroot/monetdb login
```

Just type RETURN when asked for the password. The actual steps depend on your CVS tool, e.g. in WinCVS you have to set the Admin preferences. Then get the software by using the command::

```
cvs -d:pserver:anonymous@monetdb.cvs.sourceforge.net:/cvsroot/monetdb checkout MonetD
```

This will create a directory MonetDB in your current working directory.

Also see SourceForge documentation for more information on using CVS.

1.6.4.4 Compiling

Open a Windows command shell or a UnxUtils shell and go to the top-level directory of MonetDB. Make sure that the proper environment for MSVC++ is set, otherwise call the corresponding BAT file. (see C:\Program Files\Microsoft Visual Studio\VC98\Bin\VCVARS32.BAT). Make sure that Python, C:\Pthreads\lib (or wherever you installed Pthreads), and \usr\local\wbin are in the PATH of your command shell. When all prerequisites have been satisfied, you type

```
cd NT
nmake NEED_MX=1
```

to compile the source code. If you want to compile with Java and PHP support, the command is

```
nmake HAVE_JAVA=1 HAVE_PHP=1
```

If nmake went successfully, you can try

```
nmake check
```

This will do a lot of tests, some are unfortunately still expected to fail, but most should go successfully. At the end of the output there is a reference to an HTML file which is created by the test process and shows the test results. An explanation of the test results can be found in XXX–to be filled in. Testing takes about 46 MB of disk space in the build directory.

1.6.4.5 Installing

Give the command

```
nmake install
```

By default this will install MonetDB to "<sourcedir>\NT\" where "<sourcedir>" is the top-level directory of MonetDB source tree. The current version does not provide any means to change this default.

1.6.4.6 Testing the Installation

Make sure that <SOURCEDIR>\NT\BIN, <SOURCEDIR>\NT\LIB, and <SOURCEDIR>\NT\LIB\MONETDB are in your PATH. In the MonetDB top-level directory issue the command

```
Mtest.py -r
```

(Make sure Mtest.py can be found, it should be in the bin directory that was filled by the nmake install command.)

This should produce much the same output as nmake check above, but uses the installed version of MonetDB.

1.6.4.7 Usage

There are two alternatives to use MonetDB interactively. In both cases, you first need to make sure, that your environment is set as described above.

If you're "in a hurry", you can then simply start

```
mserver5
```

and you get the "server console", where you can type you commands. The server console is mainly meant for administrative use, hence there are no conveniences such as commandline editing or command history.

For a more convenient interface, you should use the `mclient`. To do so, start the server and then, in a second shell, set up the same environment and start `mclient`

You now have a text-based MAL interface that supports commandline editing and a commandline history. The latter can even be stored persistently to be re-used after stopping and restarting `mclient`; see

```
mclient --help
```

for details.

1.6.4.8 Alternative builds

It is also possible to compile MonetDB using MinGW (Minimalist GNU for Windows) on Windows. See the documentation in the source distribution.

1.6.5 Daily Builds

Next to functionality and performance, stability and portability are first class goals of the MonetDB project. Pursuing these goals requires to constantly monitor the evolving MonetDB code base. For this purpose, we developed a test environment that automatically compiles and tests MonetDB (and its most prominent add-on packages) every night on a variety of system configurations.

Software patches and functional enhancements are checked into the repositories on a daily basis. A limited set of distribution packages is prepared to disseminate the latest to developers and application programmers as quickly as possible. Such builds may, however, contain bugs or sometimes even break old functionality.

The TestWeb provides access to the test web-site that summarizes the results of the Automated Testing activities on various platforms. It is a good starting point before picking up a daily build version.

Two versions of MonetDB are tested daily on all available platforms:

- the *cutting edge* development version ("Current"), i.e. the head of the main CVS branch; and
- the latest release version ("Stable"), i.e. the head of the most recent release branch.

The test reports consist of three overview pages ("cross-check-lists") revealing the results of

1. all compilation steps (bootstrap, configure, make, make install),
2. testing via "make check" (using debugmask 10, i.e., exhaustive monitoring and correction of physical BAT properties is enabled in the server), and
3. testing via "Mtest.py -d0 -r" (using debugmask 0, i.e., all debugging is switched off in the server).

1.6.5.1 Stability

With a (code-wise) complex system like MonetDB, modifying the source code — be it for fixing bugs or for adding new features — always bears the risk of breaking or at least altering some existing functionality. To facilitate the task of detecting such changes, small test scripts together with their respective correct/expected ("stable") output are collected within the CVS repository of MonetDB. Given the complexity of MonetDB, there is no way to do anything close to "exhaustive" testing, hence, the idea is to continuously extend the test collection. E.g., each developer should add some tests as soon as she/he adds new functionality. Likewise, a test script should be added for each bug report to monitor whether/when the bug is fixed, and to prevent (or at least detect) future occurrences of the same bug. The collection consists for hundreds of test scripts, each covering many micro-functionality tests.

To run all the tests and compare their current output to their stable output, a tool called Mtest is included in the MonetDB code base. Mtest recursively walks through the source tree, runs tests, and checks for difference between the stable and the current output. As a result, Mtest creates the web interface that allows convenient access to the differences encountered during testing. Each developer is supposed to run "Mtest" (respectively "make check") on his/her favorite development platform and check the results before checking in her/his changes. During the automatic daily tests, "make check" and "Mtest" are run on all testing platforms and the TestWeb is generated to provide convenient access to the results.

1.6.5.2 Portability

Though Fedora Linux on AMD Athlon PC's is our main development platform at CWI, we do not limit our attention to this single platform. Supporting a broad range of hardware and software platforms is an important concern.

Using standard configuration tools like automake, autoconf, and libtool, we have the same code base compiling not only on various flavors of Unix (e.g., Linux, Cygwin, AIX, IRIX, Solaris, MacOS X) but also on native Windows. Furthermore, the very code base compiles with a wide spectrum of (C-) compilers, ranging from GNU's gcc over several native Unix compilers (IBM, SGI, Sun, Intel, Portland Group) to Microsoft's Visual Studio and Visual Studio .NET on Windows.

On the hardware side, we have (had) MonetDB running on "almost anything" from a Intel StrongARM-based Linux PDA with 64 MB of flash memory to an SGI Origin2000 with 32 MIPS R12k CPU's and a total of 64 GB of (shared) main memory.

1.7 Development Roadmap

In this section we summarize the MonetDB development roadmap. The information is organized around the major system components. A precise timeline can not be given. It depends too much on the available resources and urgency (= pressure) by our research needs and clients.

1.7.1 Server Roadmap

The MonetDB server code base is continously being improved. A few areas under develeopement in the kernel area are:

- *Parallelism* Exploitation of multi-core systems calls for renewed attention to parallel processing of the MonetDB kernel. Stress testing of concurrent processing may reveal race conditions hereto undetected.
- *Streaming Data* A separete area is support for streaming database functionality. It requires additions to the way we support io-channels and schedule query plans.
- *Functional Enhancements* Support for geographical application is underway. It consists of a concise library for managing geometric types.

1.7.2 SQL Roadmap

The long term objective for the SQL front-end is to provide all features available in SQL:2003. The priority for individual features is determined in an ad hoq way. The SQL features scheduled for implementation and those that won't be supported in the foreseeable future are shown below.

Our current assessment of the features planned for upcoming releases, in order of priority, are:

- *Window functions* Datawarehousing and data mining applications require support for windowing functions, e.g. (x() OVER (partition by order by)
- *Full text retrieval* A full text retrieval support function consists of a special constructed index over text appearing in multiple columns of a relational table. This index is built using well-known Information Retrieval techniques, such as stemming, keyword recognition, and stop-word reduction. Several IR projects are underway, which enhance MonetDB with IR capabilities.
- *Support for multi-media objects* MonetDB has been used in several multi-media projects, but mostly to store and manipulate derived features. Multimedia objects can be stored as unprotected URLs, i.e. there is no guarantee the object referred to exists upon answering a query. The functionality should be extended with image, audio, and video types.
- *Replication Service* A single-write multiple-read distributed replication service is prepared for release mid 2007. It will provide both the concept of merge tables and selective replication of tuples to different servers.
- *GIS support* Support for geographical application is underway. It consists of a concise library for managing geometric types.
- *General column and table constraint enforcement*
- *Internationalization of the character sets*
- *Full outer-join queries*

The database back-end architecture prohibits easy implementation of several SQL-99 features. Those on the list below are not expected to be supported.

- Cursor based processing, because the execution engine is not based on the iterator model deployed in other engines. A simulation of the cursor based scheme would be utterly expensive from a performance point of view.
- Multi-level transaction isolation levels. Coarse grain isolation is provided using table level locks.

1.7.3 XQuery Roadmap

The XQuery compiler is currently only available on MonetDB Version 4. It is based on the Pathfinder compiler project. The MonetDB/XQuery Front End "Pathfinder" actually does not much more than translating the XQuery statements to the relational 'MIL' (MonetDB Interpreter Language). If you want to know more about this intermediate language, you can also look at the code produced by the Pathfinder compiler, but be warned. Seemingly simple XQuery statements may be turned into hundreds of lines of MIL.

The XQuery compiler is currently be retargetted to 'MAL' (MonetDB Assembly Language). A separate track of the Pathfinder team is to provide a XQuery to SQL compiler as well.

Topics in need for attention and planned for the next releases are shown below in approximately our priority list. For more information about implementation of new features, keep a close eye on the Roadmap

- Implement additional API's (XML-DB, Perl, ???)
- Implement date/time functions
- Improve string-search in MonetDB/XQuery

1.7.4 Embedded MonetDB Roadmap

The embedded MonetDB software family provides support for both SQL and XQuery. The software has been tuned to run on small scale hardware platforms.

A broader deployment of the embedded technology requires both extensions in the distributed MonetDB versions and its replication services. Continual attention is given to the memory footprint and cpu/io resource consumptions on embedded devices.

A separate project, called the Datacell, is underway and geared at providing a streaming environment for embedded applications.

1.8 Server Management

This section presents the basics to manage a collection of MonetDB database servers. The system is designed to run out of the box for most end-users.

Additional finetuning by database administrators may be required in those cases where the MonetDB software is centrally made available or when mission critial databases are kept on highly-reliable production platforms.

1.8.1 Start and Stop the Server

Starting and stopping the server under Windows for local use doesn't need an extensive description. Just locate the server in the program list and start it. After the server has been started, you can activate a textual client interface using the same procedure or use any of the third-party GUIs. Close the window of the server and/or client and it ceases to exist.

If you plan to make the server accessible from remote locations then the configuration file should be editted. See Section 1.8.4 [Database Configuration], page 23 for more details.

On Linux systems you are confronted by the following programs: MEROVINGIAN, MSERVER5, MONETDB, and MCLIENT. The MEROVINGIAN is a daemon process that controls a collection of database servers, i.e. MSERVER5 processes, each looking after a single physical database. Start this program or make it part of a system initialization script.

With merovingian running in the background managing the databases and their connections is greatly simplified. After a fresh install the next step would typically be to create your first database, e.g. demo.

The program MONETDB is your aid here. It can create/destroy databases and provides options to inspect the stability/liveliness of all database servers. Database servers can be temporarily closed for external access for maintenance, and (to be delivered) support easy checkpoint and recovery. Let's create the demo database.

```
shell> monetdb create demo
successfully created database 'demo'
```

The status of all database servers can be inspected using:

```
shell> monetdb status -s
name         state  uptime  crashes
demo       stopped               0
```

This report is helpful to determine possible instabilities and heavy loaded servers. In this case, it indicates that our database exists, but that no server is running yet.

```
shell> monetdb start demo
shell> monetdb status
     name        state     uptime     crashes
demo            running          5s  0
```

You (the database administrator) can now establish a connection using any of the user interfaces. The most common one is MCLIENT, which provides a light-weight textual interface. For example, the statements below illustrate a short session. The session is closed using the mclient console command \q.

```
shell> mclient -lsql --database=demo
sql>CREATE USER "voc" WITH PASSWORD 'voc' NAME 'VOC Explorer' SCHEMA "sys";
sql>CREATE SCHEMA "voc" AUTHORIZATION "voc";
sql>ALTER USER "voc" SET SCHEMA "voc";
sql>\q
```

See for a more complete session VOC demo.

Once in a while a database should be closed for maintenance. This operation should be issued with care, because it affects running application. The first step is to block clients to establish new connections using the command:

```
shell> monetdb lock demo
```

The effect is that only the system administrator can gain access to the server. All other users are warned using the message 'Database temporarily unavailable for maintenance' upon an attempt to connect.

Step two is to connect as system administrator and inspect the state of all clients connections.

```
shell> mclient -lsql --database=demo
sql>select * from clients;
```

If all seem dormant, the server can be shut down. More details are given in the next section.

After maintenance has been completed, the database server can be opened for connections using 'monetdb release demo'.

For more details on merovingian and monetdb inspect their manual pages.

NETWORK SECURITY. The final step is to decide on the connection restriction. By default, the servers only accepts connections from the localhost. To change this policy edit the MonetDB configuration file. %A database demo can be made worldwide accessible %using the command 'MONETDB PUBLIC DEMO'.

1.8.2 Database Dumps

An ASCII-based database dump is a safe scheme to transport a database to another platform or to migrate to an (incompatible) new version of MonetDB. This feature is standard available in MCLIENT.

1.8.3 Server Architecture

Maintenance of the MonetDB servers is based on a clear separation of tasks between multiple processes, directories, and their dependencies.

The anchor point for MonetDB is a directory (or folder) called the *dbfarm*. It contains sub-directories, one for each database. Similar, the database logs and checkpoint anchor points are *dblogs* and *dbarchive*. They should preferably be mounted on different storage devices.

Access restrictions are inherited from the operating system authorization scheme. It may proof useful to introduce a separate account for controlling access to MonetDB resources.

1.8.4 Database Configuration

The database environment is described in a configuration file, which is used by server-side applications, e.g. MEROVINGIAN and MSERVER5. A default version is created upon system installation in PREFIX/ETC/MONETDB5.CONF. Below we illustrate its most important components, for the remaining details look at the configuration file itself.

- prefix=/ufs/myhome/monet5/Linux
- exec_prefix=${prefix}
- dbfarm=${prefix}/var/MonetDB5/dbfarm
- monet_mod_path=${exec_prefix}/lib(64)/MonetDB5
- mal_init=${prefix}/lib(64)/MonetDB5/mal_init.mal
- sql_init=${exec_prefix }/lib/MonetDB5/sql_init.sql
- sql_logdir=${prefix}/var/MonetDB5/dblogs
- merovingian_log=${prefix}/var/merovingian.log
- mapi_open=false

The header consist of system wide information. The PREFIX and EXEC_PREFIX describe the location where MonetDB has been installed. MONET_MOD_PATH tells where to find the libraries. The bootstrap file for the kernel is given by MAL_INIT. These arguments are critical for a proper working server.

The option SQL_INIT is a comma separated list of SQL files to be executed upon system restart. It is primarily used to make SQL library functions known to all users.

The logs are typically stored on a different storage medium to protect the database against accidental hardware loss.

Client connections are limited to those originating from the same machine. To make the database accessible from remote sites the option MAPI_OPEN should be set to TRUE.

1.8.5 Checkpoint and Recovery

Safeguarding the content of your database against disasters, both hardware and malicious use, requires carefully planned steps. The first line of defense is to keep the database logs physically separated from the database store itself, e.g. on different disks. The second line of defense is to regularly create a database dump or full checkpoint. This is a consolidated snapshot and should be stored away at a failure independent location, e.g. a vault. Since a dump is a rather expensive operation, the third line of defense is to keep differential lists

from the last dump based on the update logs. It forms a basis to rollback to a known correct state.

This part is under review and not realised yet

On Linux platforms the checkpoint and recover operations are provided by the `monetdb` toolkit. The process starts with taking a database dump or full checkpoint. It is a DBA maintenance action. [implemented as a compressed tar file of the dbfarm after consolidation of all logs]

Taking a checkpoint and its recovery is as simple as:

```
shell> monetdb checkpoint -f demo
Checkpoint store .../var/MonetDB5/dbarchive
Preparing checkpoint file 'demo/2006-01-02-223704'
```

A checkpoint should be made at regular intervals or using a crontab job. Stopping a server triggers a checkpoint being made.

```
shell> monetdb checkpoint demo
Checkpoint tag defined for 'demo':2006-01-02-224335
```

If there appears a need to go back in time a checkpoint date can be selected, whereafter the database is rolled forward from the latest full checkpoint using the log records.

```
shell> monetdb recover demo
Checkpoint tags defined for 'demo'
2006-01-02-224335
2006-01-02-224233
...
Specify the checkpoint tag for 'demo' ? 2006-01-02-224335
Move existing database out of the way 'demo2006-01-02-224416'
Move existing database logs out of the way 'demo2006-01-02-224416'
Restore from store .../var/MonetDB5/dbarchive
Reload checkpoint file 'demo/2006-01-02-224335'
Reload checkpoint log file 'demo/2006-01-02-224335-logs'
Database recovery finished
```

1.8.6 Detach/Attach Database

A low level approach is to manipulate the directories using the operating system facilities. The easiest way is to stop the server first. Then, the database directory holding the database can be copied and stored away. You may want to create a tarball with compressed files and name them clearly for later recall.

The checkpoint is not complete without the corresponding log files produced by SQL. They are stored by default in a mirror directory of DBFARM, called DBLOGS. They should also be picked up and safeguarded for future use.

On Linux the dbfarm subdirectories can be compressed and stored away for later use. Restoring a state then simple requires the reverse steps. The main source of errors is to ensure that components are all based on the same state and that no component is forgotten.

To be sure use the `monetdb` toolkit `attach` and `detach` options for a default. [to be considered]

1.8.7 Embedded Server

The Embedded Server version is optimized for running on small board computers as a database back-end for a single client. It is of particular interest if you need database functionality within a limited application setting, e.g a self-contained database distributed as part of the application. Within this context, much of the code to facilitate and protect concurrent use of the kernel can be disabled. For example, locking of critical resources in the kernel is not needed anymore, which results in significant performance improvements.

The approach taken is to wrap a server such that the interaction between client code and server can still follow the Mapi protocol. It leads to a C-program with calls to the Mapi library routines, which provides some protection against havoc behaviour. From a programming view, it differs from a client-server application in the startup and (implicit) termination.

You normally only have to change the call MAPI_CONNECT() into EMBEDDED_SQL() (or EMBEDDED_MAL()). It requires an optional argument list to refine the environment variables used by the server. In combination with the header file EMBEDDEDCLIENT.H it provides the basis to compile and link the program.

The behavior of an embedded SQL program can be simulated with a server started as follows:

```
mserver5 --set embedded=yes --dbinit="include sql;" &
```

As a result, the server starts in 'daemon' mode, loads the SQL support library, and waits for a connection. Only one connection is permitted.

1.8.7.1 Mbedded Example

A minimalistic embedded application is shown below. It creates a temporary table in the database, fills it, and retrieves the records for some statistics gathering.

The key operation is EMBEDDED_SQL() which takes an optional environment argument list. Upon success of this call, there will be a separate server thread running in the same user space to handle the database requests. A short-circuit interaction is established between the application and the kernel using in memory buffers.

The body of the program consists of the Mapi calls. It terminates with a call to MAPI_DISCONNECT() which lets the MonetDB thread gracefully die.

The tight coupling of application and kernel code also carries some dangers. Many of the MonetDB data structures can be directly accessed, or calls to the kernel routines are possible. It is highly advised to stick to the Mapi interaction protocol. It gives a little more protection against malicious behavior or unintended side-effects.

```
#include <embeddedclient.h>

#define die(dbh,hdl) (hdl?mapi_explain_result(hdl,stderr):      \
             dbh?mapi_explain(dbh,stderr):         \
                 fprintf(stderr,"command failed\n"),   \
             exit(-1))

#define close_handle(X,Y) if (mapi_close_handle(X) != MOK) die(X, Y);
```

```
int
main()
{
    Mapi dbh;
    MapiHdl hdl = NULL;
    int i;

    dbh= embedded_sql(NULL,0);
    if (dbh == NULL || mapi_error(dbh))
        die(dbh, hdl);

    /* switch off autocommit */
    if (mapi_setAutocommit(dbh, 0) != MOK || mapi_error(dbh))
        die(dbh,NULL);

    if ((hdl = mapi_query(dbh, "create table emp"
                            " (name varchar(20),age int)")) == NULL || mapi_error(
        die(dbh, hdl);
    close_handle(dbh,hdl);

    for(i=0; i< 1000; i++) {
        char query[100];
        snprintf(query, 100, "insert into emp values('user%d', %d)", i, i % 82);
        if ((hdl = mapi_query(dbh, query)) == NULL || mapi_error(dbh))
            die(dbh, hdl);
        close_handle(dbh,hdl);
    }

    if ((hdl = mapi_query(dbh, "select * from emp")) == NULL || mapi_error(dbh))
        die(dbh, hdl);

    i=0;
    while (mapi_fetch_row(hdl))  {
        char *age = mapi_fetch_field(hdl, 1);
        i= i+ atoi(age);
    }
    if (mapi_error(dbh))
        die(dbh, hdl);
    close_handle(dbh,hdl);
    printf("The footprint is %d Mb \n",i);

    mapi_disconnect(dbh);
    return 0;
}
```

The embedded MonetDB engine is available as the library LIBEMBEDDED_SQL.A (and LIBEMBEDDED_MAL.A) to be linked with a C-program. Provided the programming environment have been initialized properly, it suffices to prepare the embedded application using

```
gcc myprog.c -o myprog `monetdb5-config --cflags --libs`
```

You might also write a Makefile to build the program as follows.[todo]

```
CC= gcc
INCLUDE=`monetdb5-config --cflags`
LIBS=`monetdb5-config --libs`
myprog: myprog.o
        ${CC} myprog.o -o myprog ${LIBS}
myprog.o : myprog.c
        ${CC} -c ${INCLUDE} myprog.c

clean: myprog.o
        rm -f myprog myprog.o
```

The configuration parameters for the server are read from its default location in the file system. In an embedded setting this location may not be accessible. It requires calls to MO_ADD_OPTION() before you asks for the instantiation of the server code itself. The code snippet below illustrate how our example is given hardwired knowledge on the desired settings:

```
main(){
    opt *set = NULL;
    int setlen = 0;
...
 if (!(setlen = mo_builtin_settings(&set)))
        usage(prog);
...
 /* needed to prevent the MonetDB config file from being used */
    setlen = mo_add_option(&set, setlen, opt_config, "dbfarm", ".");
    setlen = mo_add_option(&set, setlen, opt_config, "dbname", "demo");
...
    setlen = mo_system_config(&set, setlen);
    mid = embedded_mal(set, setlen);
```

For a complete picture see the sample program in the distribution.

1.8.7.2 Limitations for Embedded MonetDB

In embedded applications the memory footprint is a factor of concern. The raw footprint as delivered by the Unix SIZE command is often used. It is, however, also easily misleading, because the footprint depends on both the code segments and buffered database partitions in use. Therefore it makes sense to experiment with a minimal, but functionally complete application to decide if the resources limitations are obeyed.

The minimal static footprint of MonetDB is about 16 Mb (+ ca 4Mb for SQL). After module loading the space quickly grows to about 60Mb. *This footprint should be reduced.*

The embedded application world calls for many, highly specialized enhancements. It is often well worth the effort to carve out the functionality needed from the MonetDB software

packages. The easiest solution to limit the functionality and reduce resource consumption is to reduce the modules loaded. This requires patches to the startup scripts.

The benefit of an embedded database application also comes with limitations. The one and foremost limitation of embedded MonetDB is that the first application accessing the database effectively locks out any other concurrent use. Even in those situations where concurrent applications merely read the database, or create privately held tables.

2 Client Interfaces

Clients gain access to the Monet server through a internet connection or through its server console. Access through the internet requires a client program at the source, which addresses the default port of a running server. The functionality of the server console is limited. It is a textual interface for expert use.

At the server side, each client is represented by a session record with the current status, such as name, file descriptors, namespace, and local stack. Each client session has a dedicated thread of control, which limits the number of concurrent users to the thread management facilities of the underlying operating system. A large client base should be supported using a single server-side client thread, geared at providing a particular service.

The number of clients permitted concurrent access is a compile time option. The console is the first and is always present. It reads from standard input and writes to standard output.

Client sessions remain in existence until the corresponding communication channels break or its retention timer expires The administrator and owner of a sesssion can manipulate the timeout with a system call.

There are many user-friendly tools to interact with a SQL database server. A few based on the JDBC library of MonetDB are included for reference only.

2.1 Mapi Client

The mclient program is the universal command-line tool that implements the MAPI protocol for client-server interaction with MonetDB.

On a Windows platform it can be started using start->MonetDB->MonetDB SQL Client. Alternatively, you can use the command window to start mclient.exe. Be aware that your environment variables are properly set to find the libraries of interest.

On a Linux platform it provides readline functionality, which greatly improves user interaction. A history can be maintained to ease interaction over multiple sessions.

A mclient requires minimally a language and host or port argument. The default setting is geared at establishing a guest connection to a SQL or XQuery database at a default server running on the localhost. The -H HOSTNAME specifies on which machine the MonetDB server is running. If you communicate with a MonetDB server on the same machine, it can be omitted.

The timer switch reports on the round-about time for queries sent to the server. It provides a first impression on the execution cost.

```
Usage: mclient --language=(sql|xquery|mal|mil) [ options ]

Options are:
 -d database | --database=database  database to connect to
 -e          | --echo               echo the query
 -f kind     | --format=kind        specify output format dm,xml for Xquery, or csv,tab,
 -H          | --history            load/save cmdline history (default off)
 -h hostname | --host=hostname      host to connect to
 -i          | --interactive        read stdin after command line args
 -l language | --language=lang      sql,xquery,mal,mil
```

```
 -L logfile  | --log=logfile        save client/server interaction
 -P passwd   | --passwd=passwd      password
 -p portnr   | --port=portnr        port to connect to
 -s stmt     | --statement=stmt     run single statement
 -t          | --time               time commands
 -X          | --Xdebug             trace mapi network interaction
 -u user     | --user=user          user id
 -?          | --help               show this usage message
 -| cmd      | --pager=cmd          for pagination

SQL specific opions
 -r nr       | --rows=nr            for pagination
 -w nr       | --width=nr           for pagination
 -D          | --dump               create an SQL dump

XQuery specific options
 -C colname  | --collection=name    collection name
 -I docname  | --input=docname      document name, XML document on standard input
```

The default MAPI_PORT TCP port used is 50000. If this port happens to be in use on the server machine (which generally is only the case if you run two MonetDB servers on it), you will have to use the -P PORT do define the port in which the `mserver` is listening. Otherwise, it may also be omitted. If there are more than one mserver running you must also specify the database name -D DATABASE. In this case, if your port is set to the wrong database, the connection will be always redirect to the correct one. Note that the default port (and other default options) can be set in the server configuration file.

Within the context of each query language there are more options. They can be shown usin the command \? or using the commandline. For SQL there are several knobs to tune for a better rendering of result tables (\w).

```
shell> mclient -lsql --help
\?      - show this message
\<file  - read input from file
\>file  - save response in file, or stdout if no file is given
\|cmd   - pipe result to process, or stop when no command is given
\h      - show the readline history
\t      - toggle timer
\e      - echo the query in sql formatting mode
\D table- dumps the table, or the complete database if none given.
\d table- describe the table, or the complete database if none given.
\A      - enable auto commit
\a      - disable auto commit
\f      - format using a built-in renderer csv,tab,raw,sql,xml
\w#     - set maximal page width (-1=raw,0=no limit, >0 max char)
\r#     - set maximum rows per page (-1=raw)
\L file - save client/server interaction
\X      - trace mclient code
\q      - terminate session
```

2.1.1 Online help

The textual interface [No value for “mclient”] supports a limited form of online help commands. The argument is a (partial) operator call, which is looked up in the symbol table. If the pattern includes a ’(’ it also displays the signature for each match. The ARGUMENT TYPES and ADDRESS attributes are also shown if the call contains the closing bracket ’)’.

```
>?bat.is
bat.isSynced
bat.isCached
bat.isPersistent
bat.isTransient
bat.isSortedReverse
bat.isSorted
bat.isaSet
bat.isaKey
>?bat.isSorted(
command bat.isSorted(b:bat[:any_1,:any_2]):bit
>?bat.isSorted()
command bat.isSorted(b:bat[:any_1,:any_2]):bit address BKCisSorted;
Returns whether a BAT is ordered on head or not.
```

The module and function names can be replaced by the wildcard character ’*’. General regulat pattern matching is not supported.

```
>?*.print()
command color.print(c:color):void
pattern array.print(a:bat[:any_1,:any_2],b:bat[:any_1,:int]...):void
pattern io.print(b1:bat[:any_1,:any]...):int
pattern io.print(order:int,b:bat[:any_1,:any],b2:bat[:any_1,:any]...):int
pattern io.print(val:any_1):int
pattern io.print(val:any_1,lst:any...):int
pattern io.print(val:bat[:any_1,:any_2]):int
```

The result of the help command can also be obtained in a BAT, using the commands MANUAL.HELP. Keyword based lookup is supported by the operation MANUAL.SEARCH; Additional routines are available in the INSPECT module to built reflexive code.

2.2 Jdbc Client

The textual client using the JDBC protocol comes with several options to fine-tune the interaction with the database server. A synopsis of the calling arguments is given below

```
java -jar $prefix/share/MonetDB/lib/jdbcclient.jar  \
        [-h host[:port]] [-p port] \
[-f file] [-u user] [-l language] [-b [database]] \
[-d [table]] [-e] [-X<opt>]
```

or using long option equivalents –host –port –file –user –language –dump –echo –database. Arguments may be written directly after the option like -p50000.

If no host and port are given, localhost and 50000 are assumed. An .MONETDB file may exist in the user’s home directory. This file can contain preferences to use each time the

program is started. Options given on the command line override the preferences file. The .MONETDB file syntax is <option>=<value> where option is one of the options host, port, file, mode debug, or password. Note that the last one is perilous and therefore not available as command line option. If no input file is given using the -f flag, an interactive session is started on the terminal.

NOTE The JDBC protocol does not support the SQL DEBUG <query>, PROFILE <query>, and TRACE <query> options. Use the mclient tool instead. OPTIONS

`-h --host` The hostname of the host that runs the MonetDB database. A port number can be supplied by use of a colon, i.e. -h somehost:12345.

`-p --port` The port number to connect to.

`-f --file` A file name to use either for reading or writing. The file will be used for writing when dump mode is used (-d –dump). In read mode, the file can also be an URL pointing to a plain text file that is optionally gzip compressed.

`-u --user` The username to use when connecting to the database.

`-d --database`
Try to connect to the given database (only makes sense if connecting to a DatabasePool, M5 or equivalent process).

`-l --language`
Use the given language, for example ’xquery’.

`--help` This screen.

`--version`
Display driver version and exit.

`-e --echo` Also outputs the contents of the input file, if any.

`-q --quiet`
Suppress printing the welcome header.

`-D --dump` Dumps the given table(s), or the complete database if none given.

EXTRA OPTIONS

`-Xdebug` Writes a transmission log to disk for debugging purposes. If a file name is given, it is used, otherwise a file called monet<timestamp>.log is created. A given file will never be overwritten; instead a unique variation of the file is used.

`-Xembedded`
Uses an "embedded" server instance. The argument to this option should be in the form of path/to/mserver:dbname[:dbfarm[:dbinit]].

`-Xhash` Use the given hash algorithm during challenge response. Supported algorithm names: SHA1, MD5, plain.

`-Xoutput` The output mode when dumping. Default is sql, xml may be used for an experimental XML output.

`-Xbatching`
Indicates that a batch should be used instead of direct communication with the server for each statement. If a number is given, it is used as batch size.

I.e. 8000 would execute the contents on the batch after each 8000 read rows. Batching can greatly speedup the process of restoring a database dump.

2.3 Aqua Data Studio

Aqua Data Studio is a graphical user interface to interact with MonetDB/SQL. It is available on Windows, Linux, and MacOS platforms from the distribution site. Download the Version 4.7 executable and install the software.

The first step to make Aqua Data Studio aware of MonetDB is to register the database server . Go to the SERVER->REGISTER SERVER panel and select the Generic- JDBC RDBMS theme. It requires the following additional field settings:

Name:	MonetDB SQL
Type:	*Whatever you want*
Login Name:	monetdb
Password:	monetdb
URL	jdbc:monetdb://localhost/database
Driver:	nl.cwi.monetdb.jdbc.MonetDriver
Driver Location:	C:\Program Files\CWI\MonetDB\share\MonetDB\lib\monetdb-1.4-jdbc.jar

The location of the JDBC driver under Linux and OSX is by default /usr/share/MonetDB/lib/monetdb-1.4-jdbc.jar.

Once the settings has been completed, start the MonetDB server and try to connect. If necessary extend the heap size of your java engine, e.g. use -Xmx1024M .

Consult your system administrator if other MonetDB user credentials and locality settings are required.

2.4 DbVisualizer

DbVisualizer is a platform independent tool aimed to simplify database development and management for database administrators and developers. It's a very cool tool (can even draw dependency graphs based on the schema and foreign keys.

Free, personal use versions are available from their website. Download and install the software. The following scheme works for their versions 6.1.

After starting DbVisualizer for the first time, it will load its default welcome screen and starts the wizard to select database driver. Cancel this wizard and open the Driver Manager, using the menu Tools->Driver Manager... In the Driver Manager add a new Driver using Driver->Create Driver... Type the name of the driver in the Name field, e.g. MonetDB. Type the URL format for the driver: jdbc:monetdb://hostname/database. In the Driver File Paths box, follow the directions to load a JAR file that contains the JDBC driver. Use the MonetDB JDBC driver that came with your distribution, typically installed in ${prefix }/share/MonetDB/lib/monetdb-X.Y-jdbc.jar.

After adding, the window should list the driver class nl.cwi.monetdb.jdbc.MonetDriver and automatically fills it in in the Driver Class field. Close the Driver Manager window.

From the menu select Database->Create Database Connection. A pop-up dialog will try to pursue you to use the wizard. You know better so, click "No". Fill in the name for the connection in the Alias field, e.g. MonetDB. Select the monetdb driver from the list. Copy the default URL by clicking on the "URL Format: ..." text field and change it to reflect the right hostname (usually localhost will do). Fill in the default userid and password (monetdb). Press the "Connect" button. It will report the database being used and the JDBC driver in use. In the left pane the monetdb database now becomes available from browsing.

Explore the application and have fun!

A caveat of the free-version system is its performance on SQL scripts. They are sent as a single string to the server for execution. This is not the most optimal situation for MonetDB. Running a batch script is better started from the MonetDB SQL client.

2.5 SQuirreL

SQuirreL SQL Client is a graphical Java program to view the structure of a JDBC compliant database, browse the data in tables, issue SQL commands, etc. It's a very cool tool. It can even draw dependency graphs based on the schema and foreign keys.

The latest versions is available from their website. Download and install the software. The following scheme works for version 2.5.

After starting SQuirreL for the first time, it will load its default welcome screen. Locate the Drivers window and click to add a driver. In the pope's form enter the driver name MonetDB, the example URL jdbc:monetdb://localhost/demo, the website http://monetdb.cwi.nl. Now goto Extra Class Path and add the MonetDB JDBC driver that came with your distribution, typically installed in ${prefix }/share/MonetDB/lib/monetdb-X.Y-jdbc.jar. Finally, add the class name `nl.cwi.monetdb.jdbc.MonetDriver`.

Restart SQuirreL. The MonetDB driver should be marked ok. Create an alias to setup your first session. Explore the application and have fun!

2.6 iSQL-Viewer

Another open-source graphical user interface is the iSQL-Viewer. It runs on any Java-enabled platform.

Once installed and started you have the option define a service (or let the system find one itself). [Ignore the auto detect of services option]

The Tools->Service Manager choice brings up a form to provide the detail for the MonetDB connections. Click the third button in tool bar and select 'Local Service' to define a service Immediately select the tab named 'Resource' and add the location of the MonetDB JDBC driver, installed by default in or C:\PROGRAM FILES\CWI\MONETDB\SHARE\MONETDB\LIB\MONETDB-1.4-JDBC.JAR under Windows and and in /USR/SHARE/MONETDB/LIB/MONETDB-1.4-JDBC.JAR under Linux and OSX

Go back to tab 'General' and enter the following items:

Connection Name	MonetDB SQL
JDBC Driver	nl.cwi.monetdb.jdbc.MonetDriver
JDBC URL	jdbc:monetdb://localhost/database
User name	monetdb

Password monetdb

Warning, disable the authentication prompt if you can not edit the user and password fields.

In the TOOLS->SERVICE MANAGER->RESOURCE panel enter the location of the MonetDB JDBC driver C:\PROGRAM FILES\CWI\MONETDB\SHARE\MONETDB\LIB\MONETDB-1.4-JDBC.JAR

Documentation and tutorial on iSQL-Viewer are available on their website.

2.7 Web Services

2.7.1 Apache Configuration

Here are the instructions to make .XQ files directly executable from the Apache web server.

Note: The code provided below is still in an experimental stage and is not intended to be used in a production environment.

2.7.2 httpd.conf

First you must adapt the HTTPD.CONF configuration file of the Apache web server, in order to:

- turn on cgi scripts (if not already the case)
- add a "handler" for .XQ files:

To do so, I made the following changes:

```
$ diff -w httpd.conf httpd.conf.default
 858c858
 >     AddHandler cgi-script .cgi
 ---
 >    #AddHandler cgi-script .cgi
 885,890d884
 < # redirect xquery files to our cgi script
 < AddType    text/xml .xq
 < AddHandler xquery-type .xq
 < Action xquery-type /cgi-bin/xquery.cgi
```

Don't forget to restart Apache; so it reads HTTPD.CONF

2.7.3 xquery.cgi

In the CGI-BIN/ directory, you must also place the following script (XQUERY.CGI) with executable file permissions:

```
chmod 755 xquery.cgi
```

Beware! You must potentially adapt:

- the location of bash
- WWWDIR (the HTDOCS directory where Apache stores its content)
- MONETDIR (the installation dir of MonetDB/XQuery)

```
#!/bin/bash
echo 'Content-type: text/xml'
echo

MONETDIR=/path_to_monetdb/
WWWDIR=/var/www/htdocs

XQFILE=$WWWDIR/$REDIRECT_URL

if [ x$QUERY_STRING == x ]
then
  $MONETDIR/bin/mclient --set prefix=$MONETDIR --set exec_prefix=$MONETDIR -fxml -lx $XQFI
else
   `echo "sed -e s/%$QUERY_STRING/g $XQFILE" | sed -e "s/&amp;/\/g -e s\/%/g" -e "s/=/%\//
  $MONETDIR/bin/mclient --set prefix=$MONETDIR --set exec_prefix=$MONETDIR -fxml -lx /tmp/
fi
```

2.7.4 passing parameters

The XQUERY.CGI script allows parameter passing. Suppose we have the following XQuery file (save it as EXAMPLE.XQ in the HTDOCS-folder):

```
for $i in 1 to %max%
return element { "mult" } {
  $i, " times ", %table%, " is ", $i * %table%
}
```

The TABLE and MAX have been parametrized. Parameters take the form: %NAME%

The parameters can be used in the usual URL convention:

```
@url{http://localhost/example.xq?table=3&max=10,
http://localhost/example.xq?table=3&max=10}
```

Beware: parameter substitution in this script is **very simple**; it won't work with special characters in it (that get escaped by the web server) or even spaces. For optimal web server performance, we may want to do all parameter substitution inside `mclient`; so that we don't need to fork a bash shell process on each web request.

2.7.5 Web Service Security

When starting the MonetDB Server, this will open a network port on your system (the MAPI_PORT, configurable in MONETDB.CONF, by default 50000). If the database is to be accessible from the outside, you must modify your firewall configuration to open this port.

Before doing so, take note of the following issues:

- Currently, MonetDB/XQuery lackes proper authentication. There is a single user with administrative rights and fixed passwords, and the SSL is not available yet. This will be fixed by the the upcoming **version 5** port of MonetDB/XQuery.
- As a stop-gap measure, XML documents that have not explicitly been loaded via SHRED_DOC() (and thus carry a respective alias) are *only* accessible if the MCLIENT connection comes from the same machine where MSERVER runs (i.e., for clients that

run on the local machine). Otherwise, all XML content readable by the user that started the Mserver process would be visible for the outside.

- Finally, the MonetDB server was not designed with security as a first goal, so may be susceptible to e.g. buffer overrun attacks. Thus, while in principle users are just able to execute XQueries, an open port could in the worst case lead to your computer being hacked.

Warning: As a consequence, anybody logged in to the machine where MSERVER runs can now read documents with the permissions of the user that started MSERVER!

Here are the instructions to make .xq files directly executable from a Apache web server.

Note: The code provided below is still in an experimental stage and is not intended to be used in a production environment.

2.8 httpd.conf

First you must adapt the httpd.conf configuration file of the Apache web server, in order to a) turn on cgi scripts (if not already the case), and b) add a "handler" for <code>.xq</code> files. To do so, I made the following changes:

```
$ diff -w httpd.conf httpd.conf.default
 858c858
 &lt;     AddHandler cgi-script .cgi
 ---
 &gt;     #AddHandler cgi-script .cgi
 885,890d884
 &lt; # redirect xquery files to our cgi script
 &lt; AddType    text/xml .xq
 &lt; AddHandler xquery-type .xq
 &lt; Action xquery-type /cgi-bin/xquery.cgi
```

Don't forget to restart Apache; so it reads httpd.conf

2.9 xquery.cgi

In the cgi-bin/ directory, you must also place the following script (xquery.cgi) with executable file permissions:</p>

```
chmod 755 xquery.cgi
```

Beware! You must potentially adapt:

1. the location of bash
2. WWWDIR(the htdocs directory where Apache stores its content)
3. MONETDIR (the installation dir of MonetDB/XQuery)

```
 #!/bin/bash
 echo 'Content-type: text/xml'
 echo

 MONETDIR=/path_to_monetdb/
 WWWDIR=/var/www/htdocs
```

```
 XQFILE=$WWWDIR/$REDIRECT_URL

 if [ x$QUERY_STRING == x ]
 then
   $MONETDIR/bin/mclient --set prefix=$MONETDIR --set exec_prefix=$MONETDIR -fxml -lx $XQFI
 else
    `echo "sed -e s/%$QUERY_STRING/g $XQFILE" | sed -e "s/&amp;/\/g -e s\/%/g" -e "s/=/%\//
   $MONETDIR/bin/mclient --set prefix=$MONETDIR --set exec_prefix=$MONETDIR -fxml -lx /tmp/
 fi
```

2.10 passing parameters

The <code>xquery.cgi</code> script allows parameter passing. Suppose we have the following XQuery file (save it as example.xq in the htdocs>-folder):

```
 for $i in 1 to %max%
 return element { "mult" } {
   $i, " times ", %table%, " is ", $i * %table%
 }
```

The table and max have been parametrized. Parameters take the form: %name% The parameters can be used in the usual URL convention:

```
<a href="http://localhost/example.xq?table=3&max=10"
class='external'>http://localhost/example.xq?table=3&max=10</a>
```

Beware: parameter substitution in this script is **very simple**; it won't work with special characters in it (that get escaped by the web server) or even spaces. For optimal web server performance, we may want to do all parameter substitution inside mclient; so that we don't need to fork a bash shell process on each web request.

2.11 How to use mclient with the XQuery language?

First start an Mserver, load the pathfinder module, and start listening for MAPI connections:

```
 % Mserver
 ...
 MonetDB&gt; module("pathfinder");
```

This was successful if you see the MonetDB server prompt, otherwise you have to recheck your installation steps and $PATH settings to assure that both the Mserver and the pathfinder module can be found.

You can now use MAPI to run queries as follows:

```
shell> mclient -lxquery [file.xq]
```

The result will be printed on stdout. If you would like to store the result in a file, you could of course type something like:</p>

```
shell> mclient -lxquery [file.xq] &gt; [output.xml]
```

2.12 MonetDB Server

2.12.1 Starting MonetDB Server

Simply click: 'Start' -> 'Programs' -> 'MonetDB XQuery' -> 'MonetDB XQuery Server'. This will start the MonetDB Server with XQuery support in a separate window. Although the window comes with an interactive prompt, you should (unless you know what you are doing) keep this window minimized.</p>

2.12.2 Shutting down MonetDB Server

To stop the MonetDB Server, you can simply close the <code>XQuery Server</code> window. This will stop the server.

2.12.3 MonetDB Client Side

Once you started the MonetDB Server, you can start executing queries and managing your document collection.

2.12.4 Executing a query

To execute a query; save the query as a file with an .xq extension. Then double-click it. It will display the result in your default XML-browser. </p>

2.12.5 How to add a document persistently to your collection?

To shred a document into your collection, you should start a MCLIENT MIL session by clicking: 'Start' ->'Programs' ->'MonetDB XQuery' ->'mclient MIL Session'. This will open a MAPI session with the MonetDB Server. On the command-line use the shred_doc(full_path, logical_name) function to add XML-documents to your collection. Type for instance something like:

```
mil>shred_doc("C:\\..\\HelloWorld.xml","HelloWorld.xml");
# Shredded XML doc("HelloWorld.xml"), total time after commit=0.084s
mil>
```

Note: that backslashes '\' need to be escaped as '\\'.

2.12.6 How to delete a document persistently from your collection?

If you are wondering how you can delete a document from your collection; here is how to you can delete the "HelloWorld.xml" document that we just added, by referring to its logical name.

```
mil>delete_doc("HelloWorld.xml");
mil>
```

If you want to delete all documents from your collection, type:

```
mil>delete_all_docs(TRUE);
mil>
```

to delete only the (implicitely) cached documents, or

```
mil>delete_all_docs(FALSE);
mil>
```

to delete also the (explicitely) shredded documents.

3 SQL

The de facto language for database applications is SQL. It evolved through several phases of standardization to the version currently known as SQL-2003. The SQL standard provides an ideal language framework, in terms of standardization committee viewpoints. It is, however, hardly met by any of the existing (commercial) implementations. This is largely due to software legacy and backward compatibility requirements from their client base. See for instance this on-line article on SQL standards.

In 2002 the first version of the SQL front end emerged. This late development made it possible to immediately start from the SQL'99 definition. As soon as the SQL'03 specifications became available, its content was taken as the primary frame of reference. The SQL development strategy is driven by immediate needs of the user base, so that less-frequently used features end up low on the development stack.

The architecture is based on a separate compiler module, which translates SQL statements into the MonetDB Assembly Language (MAL). In this process common optimization heuristics, specific to the relational algebra are performed. There are bindings for SQL with e.g. JDBC, PHP and C, to integrate seamlessly in existing developments environments.

The remainder of this chapter presents a synopsis of the language features. It provides a quick intro on the scope of the current implementation, mostly through examples. Proficiency in elementary use of SQL is assumed. If you are new to this world then pick up any of the introduction books and study it carefully, e.g. *J. Melton and A.R. Simon*, SQL:1999 Understanding Relational Language Components, ISBN 1558604561.

3.1 SQL Core

This section introduces the language components supported. It is not intended to be exhaustive, but illustrates the features using programming snippets. For details we refer to the SQL standard.

Where appropriate the SQL syntax is summarized in extended BNF. Alternative constructs are separated by | and grouped by parentheses. Optional parts are marked with square brackets. A repetition is marked with either '+' or '*' to indicate at least once and many times, respectively. Lexical tokens are illustrated in capitals. The syntax is displayed as text enclosed by a box.

Currently we have partial support for SQL-2003. Features are added when (enough) users express their interest in these. Some of the features currently not supported are:

- Cursors, because the underlying engine is not based on record iterators.
- Asserts They will be supported in a future release.
- Domains
- Collate
- Character sets
- SQL User Defined Types

The remainder of this chapter provides an overview of the functionality provided. In most cases it is sufficient to construct complex information systems, and be competitive with other solutions.

3.1.1 Syntax

3.1.1.1 Comments

Comments can be added to query scripts for documentation purposes. MonetDB/SQL supports two forms of comments. Any text beginning with '- -' and up to the end of line is ignored. Furthermore, C-style comments can be injected in a SQL query where a language token is expected. Comments are considered equivalent to white spaces.

```
sql>-- this is comment
sql>select /* ignore this */ 3;
+--------------+
| single_value |
+==============+
|            3 |
+--------------+
```

3.1.1.2 Identifiers and Keywords

SQL comes with a large collection of keywords, i.e. names reserved by the committee to designate language constructs. Keywords in MonetDB/SQL are case in-sensitive and adhere to the general convention for identifier denotation in programming languages.

Users can overrule the interpretation of an identifier as a keyword by encapsulation with double quotes, e.g. *select* denotes a keyword, while "*select*" denotes a user defined name. This scheme also permits inclusion of white space in the names. It is general advisable to reduce the escaped keywords.

Names are used to designate database objects. In that role, they are case in-sensitive.

3.1.1.3 Literal Constants

Numeric constants follow the convention of most programming languages. A numeric constant that contains neither a decimal point or exponential is considered of type INTEGER. The underlying engine determines whether it be a 32- or 64-bit value.

String constants in SQL are embraced with single quotes ('\"). They may be split over multiple lines with blanc space in between.

Often strings can be cast to other types, provided the coercion routine is available. For example:

```
CAST ( '#ccddee' AS color)
CAST ( '0.3' AS double)
```

illustrates conversion of a color value in hexadecimal notation is converted into an integers, and a value into a double precision floating point number.

3.1.1.4 Special Characters

String literals may contain the traditional escape characters: '\n' for new lines, '\t' for tab, '\r' for return, and '\\' for backslash. The conventions '\ddd' with d a digit stands for a number denoted in octal.

In addition to the comment brackets, SQL reserves several characters to designate common operators

```
+ - * / < > = ! %
```

The operator definitions can not be overloaded, nor can you define new operators.

3.1.1.5 Operator Precedences

Most operators in SQL have the same precedence and are left-associative. Parenthesis can be used to disambiguate the precedence order.

.	left	table/column name separator
-	right	unary minus
^	left	exponentiation
* / %	left	multiplication, division, modulo
+ -	left	unary addition, subtraction
IS, IS TRUE, IS FALSE, IS UNKNOWN		
ISNULL, NOTNULL		test for (not) null
IN		set membership
BETWEEN		range containment
OVERLAPS		time interval overlap
LIKE ILIKE SIMILAR		string pattern matching
< >		less than, greater than
=	right	equality, assignment
NOT	right	logical negation
AND	left	logical conjunction
OR	left	logical disjunction'

3.1.2 Value Expressions

The language is built around value- and table- expressions. Value expressions encompass denotation of literal constants, type casts, column references, operator and function invocation, and scalar returning subqueries.

The column references take the form

```
<database_name>.<correlation_name>.<column_name>
```

The default *database_name* is the one identified at the command line while seeking database access. Omission of the database name and separating dot denotes the session default database. The database name names the data stored under a schema, possibly residing at a remote location. (See Distribution.)

The *correlation_name* is either a table name or an alias introduced in a FROM clause. The correlation name may be omitted if the columnname uniquely identifies a column in the scope of current query.

Table expressions produce a relational table. It is internally referenced through correlation name, which supports attribute selection using the '.' denotation.

3.1.3 Data Types

SQL prescribes a large collection of built-in types, most of which are directly supported. The MonetDB database kernel can also be extended with application specific types.

3.1.3.1 Native Data Types

MonetDB/SQL supports the following types.

CHAR[ACTER] (L)	character string with length L
VARCHAR (L) | CHARACTER VARYING (L)	string with atmost length L
CLOB | CHARACTER LARGE OBJECT	
BLOB | BINARY LARGE OBJECT	
DECIMAL(P,S) | NUMERIC(P,S)	
SMALLINT	16 bit integer
INT	32 bit integer
BIGINT	64 bit integer
serial	special 64 bit integer (sequence generator)
REAL	32 bit floating point
DOUBLE [PRECISION]	64 bit floating point
BOOLEAN	
DATE	
TIME(T)	time of day
TIMESTAMP(T)	date concatenated with unique time
INTERVAL(Q)	a temporal interval

The integer types align with the storage of 2,4, and 8 bytes. Their domain contains a NIL representation, which may cause unexpected side effects if you recast them to a larger size.

The numeric types are represented as fixed length integers, whose decimal point is produced during result rendering.

The types FLOAT and DOUBLE map to the underlying implementation system. No special attention is given to the value NaN.

3.1.3.2 Serial Types

As of 2003 the SQL standard supports serial types. They are of particular use in auto-generating key values.

A serial type is defined as a primary database object over any of the built-in data types. The NEXT VALUE operation generates the next value and can be used anywhere a value expression is allowed. Its name should be unique. It can only be dropped when tables mentioning it have been previously been dropped.

```
CREATE SEQUENCE test_seq AS integer;
...
SELECT NEXT VALUE FOR test_seq;
...
DROP SEQUENCE test_seq;
```

A more complex example is shown below:

```
CREATE TABLE test (
d date,
id SERIAL,
count int auto_increment,
bla int GENERATED ALWAYS AS IDENTITY (
START WITH 100 INCREMENT BY 2
NO MINVALUE MAXVALUE 1000
CACHE 2 CYCLE)
);
```

It introduces the column COUNT, which is incremented with each row being added. It is conceptually identical to the value expression MAX(COUNT)+1 in each insert. The column BLA is a limited range with wrap around.

Much like other primary database objects, the sequence type can be altered at any time as illustrated below.

```
sql>CREATE SEQUENCE "my_test_seq" as integer START WITH 2;
sql>CREATE TABLE test (t int DEFAULT
more>        NEXT VALUE FOR "my_test_seq", v char);
ALTER SEQUENCE "my_test_seq"
sql>INSERT INTO test(v)  VALUES ('a');
       RESTART WITH (SELECT MAX(t) + 1 FROM test);
Rows affected 1
sql>INSERT INTO test VALUES (10, 'b');
Rows affected 1
sql>ALTER SEQUENCE "my_test_seq"
more>        RESTART WITH (SELECT MAX(t) + 1 FROM test);
sql>INSERT INTO test(v)  VALUES ('c');
SELECT * FROM test;
Rows affected 1
sql>SELECT * FROM test;
+----+--+
| t  |v |
+====+==+
|  2 |a |
| 10 |b |
| 11 |c |
+----+--+
```

3.1.3.3 Extended Data Types

The MonetDB kernel supports creation of user defined types, e.g. geometric types. The relationship between SQL and MAL world is expressed using an EXTERNAL NAME

```
CREATE TYPE <type_name> EXTERNAL NAME <type_identification>;

DROP TYPE <type_name> [RESTRICT | CASCADE];
```

Dropping types is constrainted. If RESTRICT is set ten the type can only be dropped if no other database object relies on its definition. If CASCADE is set, then all objects depending on the type are dropped as well. Beware, this may result in loss of valuable data.

3.1.4 Table Definition

3.1.4.1 Create Table

The parser currently supports the full <table scope> specifier, but the implementation is limited to LOCAL TEMPORARY tables (i.e. the tables are only visible in the client session) and

```
CREATE [ TEMPORARY | LOCAL TEMPORARY | GLOBAL TEMPORARY ] TABLE <table_name>
table_content_source
[ON COMMIT DELETE ROWS | ON COMMIT PRESERVE ROWS | ON COMMIT DROP ]

table_content_source:
   ( table_element, ... )
 | [(<column_name>, ...)] AS select_query [WITH NO DATA | WITH DATA]


table_element:
   <column_name> <data_type> [column_option ...]
 | CONSTRAINT <constraint_name> table_constraint_type
 | <column_name> WITH OPTIONS ( column_option, ... )
 | LIKE <table_name>


table_constraint_type:
   UNIQUE ( <column_name>, ...)
 | PRIMARY KEY ( <column_name>, ...)
 | FOREIGN KEY ( <column_name>, ...) REFERENCES <table_name> [( <column_name>, ..)]
[ match_options ] [ ref_actions ]


column_option:
   DEFAULT <default_value>
 | [ <constraint_name> ] column_constraint_type
 | GENERATED ALWAYS AS IDENTITY [ ( serial_parameters ) ]
 | AUTO_INCREMENT


column_constraint_type:
   NOT NULL
 | NULL
 | UNIQUE
 | PRIMARY KEY
 | REFERENCES <table_name> [ ( <column_name>, ..) ]
[match_options] [ ref_actions ]


serial_parameters:
   [START WITH <nonzero>]
 | [RESTART | RESTART WITH subquery | RESTART WITH <nonzero>]
 | [INCREMENT BY <nonzero>]
 | [MINVALUE <nonzero> | NOMINVALUE]
 | [MAXVALUE <nonzero> | NOMAXVALUE ]
 | [CACHE <nonzero>]
 | [CYCLE | NONCYCLE]


match_options:
   MATCH  FULL | PARTIAL | SIMPLE


ref_actions:
   ON UPDATE  NO ACTION | CASCASDE | RESTRICT | SET NULL | SET DEFAULT
 | ON DELETE  NO ACTION | CASCASDE | RESTRICT | SET NULL | SET DEFAULT
 |  ON DELETE | ON UPDATE
 NO ACTION | CASCADE | RESTRICT | SET NULL | SET DEFAULT
 |  ON UPDATE | ON DELETE
 NO ACTION | CASCADE | RESTRICT | SET NULL | SET DEFAULT
```

3.1.4.2 Default values

To make insert statements easier a default value can be associated with each column. Besides literal values, temporal and sequence functions can be used as default values. The value of these functions at insert time will be used.

```
CREATE TABLE <table_name> table_content_source

table_content_source:
   ( table_element, ... )

table_element:
   <column_name> <data_type> [column_option ...]
 | <column_name> WITH OPTIONS ( column_option, ... )

column_option:
   DEFAULT <default_value>
```

3.1.4.3 Identity Column

SQL-2003 added identity columns, which are columns for which the values are coming from a sequence generator. The syntax used in MySql (auto_increment) and Postgresql (serial data type) are also supported.

```
CREATE TABLE <table_name> table_content_source

table_content_source:
   ( table_element, ... )
 | [(<column_name>, ...)] AS select_query [WITH NO DATA | WITH DATA]

table_element:
   <column_name> <data_type> [column_option ...]
 | <column_name> WITH OPTIONS ( column_option, ... )

column_option:
 | GENERATED ALWAYS AS IDENTITY [ ( serial_parameters ) ]
 | AUTO_INCREMENT

serial_parameters:
   [START WITH <nonzero>]
 | [RESTART | RESTART WITH subquery | RESTART WITH <nonzero>]
 | [INCREMENT BY <nonzero>]
 | [MINVALUE <nonzero> | NOMINVALUE]
 | [MAXVALUE <nonzero> | NOMAXVALUE ]
 | [CACHE <nonzero>]
 | [CYCLE | NONCYCLE]
```

3.1.4.4 Constraints

Column and Table constraints are supported. Besides the simple NOT NULL check also UNIQUE, PRIMARY and FOREIGN keys are supported. The limitation stems from the missing triggers, ie we currently check constraints directly on insert, update and delete. The NULL matching on foreign keys is limited to the SIMPLE MATCH type (NULL values satisfy the

constraint). The FULL and PARTIAL MATCH types are not supported. The referential action is currently limited to RESTRICT, ie an update fails if other columns have references to it.

```
CREATE TABLE <table_name> table_content_source

table_content_source:
   ( table_element, ... )

table_element:
   <column_name> <data_type> [column_option ...]
 | CONSTRAINT <constraint_name> table_constraint_type
 | <column_name> WITH OPTIONS ( column_option, ... )

table_constraint_type:
   UNIQUE ( <column_name>, ...)
 | PRIMARY KEY ( <column_name>, ...)
 | FOREIGN KEY ( <column_name>, ...) REFERENCES <table_name>
[( <column_name>, ..)] [ match_options ] [ ref_actions ]


column_option:
   [ <constraint_name> ] column_constraint_type

column_constraint_type:
   NOT NULL
 | NULL
 | UNIQUE
 | PRIMARY KEY
 | REFERENCES <table_name> [ ( <column_name>, ..) ] [match_options] [ ref_actions ]

match_options:
   MATCH  FULL | PARTIAL | SIMPLE

ref_actions:
   ON UPDATE  NO ACTION | CASCASDE | RESTRICT | SET NULL | SET DEFAULT
 | ON DELETE  NO ACTION | CASCASDE | RESTRICT | SET NULL | SET DEFAULT
 |  ON DELETE | ON UPDATE   NO ACTION | CASCASDE | RESTRICT | SET NULL | SET DEFAULT
 |  ON UPDATE | ON DELETE   NO ACTION | CASCASDE | RESTRICT | SET NULL | SET DEFAULT
```

3.1.5 Table Management

3.1.5.1 ALTER and DROP a Table

The ALTER TABLE is used to add columns or constraints to a specific table. It not only add them, but it can also be used to drop them. Another possible operation is to alter the default value of a column. All the syntax explanation for all these operations can be founded in the subsections Changing a Column, ADD and DROP a Column, ADD and DROP a Constraint.

To drop a table from a database you must create a drop statement based in the following syntax:

```
DROP TABLE <table_name> [RESTRICT | CASCADE] ';'
```

3.1.5.2 Changing a Column

The ALTER TABLE statement can be used to set or remove a default value for a column. The syntax recognized is :

```
ALTER TABLE <table_name> ALTER [COLUMN] <column_name>
SET DEFAULT default_value | NULL ';'

ALTER TABLE <table_name> ALTER [COLUMN] <column_name>
DROP DEFAULT ';'
```

3.1.5.3 ADD and DROP a Column

The ALTER TABLE statement can be used to add or drop a column from a specific table. The syntax recognized is:

```
ALTER TABLE <table_name> ADD [COLUMN] <column_name>
<data_type> [column_options] ';'

ALTER TABLE <table_name> DROP [COLUMN] <column_name>
[ RESTRICT | CASCADE ] ';'
```

3.1.5.4 ADD and DROP a Constraint

The ALTER TABLE statement can be used to add or drop a constraint from a specific table.

```
ALTER TABLE <table_name> ADD [COLUMN]
[CONSTRAINT <constraint_name>] <constraint_type> ;

ALTER TABLE <table_name> DROP CONSTRAINT <constraint_name>
 [ RESTRICT | CASCADE ] ;

<constraint_type>:
UNIQUE ( <column_name>, ...)
| PRIMARY KEY (<column_name>, ...)
| FOREIGN KEY (<column_name> ,...)
REFERENCES <table_name> [(<column_name>,...)]
[MATCH [ FULL | PARTIAL | SIMPLE ] ] [ <ref_action> ]
| CHECK (<search_condition>)

<ref_action>:
   ON DELETE | UPDATE NO ACTION | CASCADE
| RESTRICT | SET NULL | SET DEFAULT
```

3.1.5.5 CREATE VIEW

Regular view specifications are supported. However, recursive views and reference-able views are not supported. Next to this 2003 feature we support creating a view on top of a set of bats. In this case the query expression is replaced by the keyword BATS, and the bats are found based on the view specification. As this feature requires in-depth knowledge of the system it is only open to the ADMIN_ROLE.

```
CREATE VIEW <view_name> [(<column>,...)]
AS <select_query> [WITH CHECK OPTION] ;

DROP VIEW <view_name>;
```

3.1.5.6 Create Table Like

It is possible to create a table which looks like an existing table. This can be done using the create table like statement. Currently there is no support for additional options. A work around is to use the alter statement to change the options.

```
CREATE TABLE <table_name> table_content_source

table_content_source:
   ( table_element, ... )

table_element:
   LIKE <table_name>
```

3.1.5.7 Create Table AS Subquery

```
CREATE TABLE <table_name> table_content_source

table_content_source:
   [(<column_name>, ...)] AS select_query [WITH NO DATA | WITH DATA]
```

3.1.6 Data Manipulation

3.1.6.1 Insertions

A table can be populated using INSERT statements. It takes a value expression list. The values should align with the attributes in the table definition. Otherwise the attribute-value association should be explicitly defined. Multiple rows can be inserted in a single statement.

```
INSERT INTO <table_name> [ ( <column_name>, ... ) ]
[ VALUES (<values>,...) , ...  | <select_query> ]
```

```
sql>create table t(i integer, s string);
sql>insert into t values( 1, 'cat'),(2,'dog');
sql>insert into t(s,i) values('pig',1+2);
```

3.1.6.2 Updates

The UPDATE statement syntax follows the standard, but it semantics for bulk updates on keys may be slightly different then in other systems. In particular, the update implementation ensures that you can freely update any column without the danger of run-away values.

```
UPDATE <table_name> SET ( assignment, ... ) [WHERE <search_condition> ]

assignment:
   <column_name> = <scalar_exp>
 | <column_name> = <search_condition>
 | <column_name> = NULL
```

```
sql>update t set i= i+1;
sql>select * from t;
sql>select * from t;
% sys.t,        sys.t # table_name
% i,    s # name
% int,  clob # type
% 1,    3 # length
[ 2,    "cat"   ]
[ 3,    "dog"   ]
[ 4,    "pig"   ]
```

3.1.6.3 Delete

```
DELETE FROM <table_name> [WHERE <search_condition> ]
```

```
sql>delete from t where i<3;
```

3.1.6.4 Copy from/into

The COPY FROM command enables fast insertion of multiple tuples. It takes the input from an ASCII file or STDIN. The values and tuples are separated by separators designated by string constants.

```
COPY [ <int_val> RECORDS ] INTO <table_name> FROM source_location
[ [USING] DELIMITERS <delimiter_name> ]

source_location:
   ( <table_name>, ... )
 | STDIN
```

3.1.7 Queries

3.1.7.1 Table expressions

When a query is created a table can be referenced in different ways, sometimes by its name or by a select query or a join result. Here is the syntax to refer a table.

```
table_ref:
   simple_table
 | ( <select_query> ) [AS] <table_name> [ (column_name, ... ) ]
 | ( <select_query> )
 | joined_table
 | ( joined_table ) [AS] <table_name> [ (column_name, ... ) ]

joined_table:
   ( joined_table )
 | table_ref CROSS | NATURAL JOIN table_ref
 | table_ref JOIN | UNINONJOIN table_ref join_spec
 | table_ref join_type JOIN table_ref join_spec
 | table_ref NATURAL join_type JOIN table_ref

join_spec
   ON <search_condition>
 | USING (<colunm_name>,...)

join_type:
   INNER
 | LEFT | RIGHT | FULL [OUTER]
```

3.1.7.2 Select lists, distinct, sorting, limit

```
SELECT [ALL | EMPTY] selection [ INTO <column_name>, ... ]
[FROM <table_name>, ...] [WHERE <search_condition> ]
[ GROUP BY <column_name>, ...] [HAVING <search_condition]

selection:
   *
 | <column_name>, ...
```

3.1.7.3 With expressions

The WITH clause provides the mechanism to introduce in-line view definitions as exemplified by the following example:

```
WITH with_element, ...  <select_query>

with_element:
   <table_name> ( <column_name>, ...) AS ( <select_query> )
```

```
WITH t AS (
SELECT * FROM tables )
SELECT * FROM t;
```

3.1.7.4 String Operations

The syntax supported for the string operations which can be used in SQL queries.

```
string_funcs:
   SUBTRING ( scalar_exp FROM scalar_exp FOR scalar_exp )
 | SUBTRING ( scalar_exp, scalar_exp, scalar_exp )
 | SUBTRING ( scalar_exp FROM scalar_exp )
 | SUBTRING ( scalar_exp, scalar_exp )
 | SUBTRING ( scalar_exp <int_value> scalar_exp )
 | scalar_exp CONCATSTRING scalar_exp
```

3.1.7.5 Pattern Matching Operations

The pattern matching operations are used over predicates. It is possible to compare them, see the differences between them, if a predicate is a sub-predicate of another, etc. The following syntax description cover all the operations supported by MonetDB.

```
predicate:
    comparison_predicate
 |  between_predicate
 |  like_predicate
 |  test_for_null
 |  in_predicate
 |  all_or_any_predicate
 |  existence_test
 |  '(' predicate ')'

comparison_predicate:
   pred_exp COMPARISON pred_exp
 | pred_exp = pred_exp

between_predicate:
   pred_exp [NOT] BETWEEN [SYMMETRIC | SYMMETRIC] pred_exp AND pred_exp

like_predicate:
   pred_exp [NOT] LIKE atom_exp

test_for_null:
   <column_name> IS [NOT] NULL

in_predicate:
   pred_exp [NOT] IN ( <element_name>, ... )

all_or_any_predicate
   pred_exp COMPARASION  ANY | ALL | SOME  <subquery>

existence_test:
   [NOT] EXISTS <subquery>

pred_exp:
   scalar_exp
 | predicate

atom_exp:
   [(] atom [)]
 | [(] atom [)] ESCAPE <string>
 | ?
```

3.1.7.6 Date/Time Functions and Operators

Operations over date and time are supported and the syntax recognized is:

```
datetine_func:
   EXTRACT ( datatime_field FROM scalar_exp )
 | CURRENT_DATE [()]
 | CURRENT_TIME [()]
 | CURRENT_TIMESTAMP [()]
 | LOCALTIME [()]
 | LOCALTIMESTAMP [()]
 | CURRENT_TIME [()]

datetime_field:
    YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
```

3.1.7.7 Logical and Comparison Operators

The logical operators supported are the OR, AND, NOT, FALSE, and TRUE. They are used for example in a query with a search condition.

The syntax for logical operators is used in the search condition of a query, and the syntax recognized is:

```
search_condition:
   search_condition OR search_condition
 | search_condition AND search_condition
 | NOT search_condition
 | FALSE
 | TRUE
```

The comparison operators are the <>, <, >, =, >=, and <=. They are used for example in a query with a comparison predicate. The syntax recognized is:

```
comparasion_predicate:
   pred_exp COMPARASION pred_exp
 | pred_exp = pred_exp
```

```
>SELECT name FROM table1, table2 WHERE (id = 1 OR age > 0)
AND NOT FALSE AND name <> 'monet';
```

3.1.7.8 Mathematical Functions and Operators

Several mathematical functions and operators are supported by MonetDB.

```
simple_scalar_expression:
   scalar_exp + scalar_exp
 | scalar_exp - scalar_exp
 | scalar_exp * scalar_exp
 | scalar_exp / scalar_exp
 | scalar_exp % scalar_exp
 | + scalar_exp %prec UMINUS
 | - scalar_exp %prec UMINUS
 | ( scalar_exp )

scalar_exp:
   simple_scalar_expression
 | ( <select_query> ) %prec UMINUS
```

3.1.7.9 Aggregate Functions

The aggregate functions supported are AVG, MIN, MAX, SUM, and COUNT. The syntax for these aggregations is:

```
aggregation_def:
   AGGR ( * )
 | AGGR ( ident . * )
 | AGGR ( DISTINCT <column_name> )
 | AGGR ( ALL <scalar_exp> )
 | AGGR ( <scalar_exp> )
```

3.1.7.10 SubQueries

MonetDB supports queries nested into another query. To this queries we call sub-queries.

```
>CREATE TABLE table1 (id int, name varchar(1024));
>CREATE TABLE table2 (id int, age int);
>INSERT INTO table1 values(1, 'monetdb');
>INSERT INTO table2 values(0, 23);
>INSERT INTO table2 values(1, 25);
>SELECT name FROM table1 WHERE id IN ( SELECT id FROM table2);
% sys.table1 # table_name
% name # name
% varchar # type
% 7 # length
[ "monetdb"     ]
>
```

3.1.8 Indexes

The INDEX statements in the SQL standard are recognized, but their implementation is different from competitive products. MonetDB/SQL interprets these statements as advice and often freely neglects its, relying on its own decision to create and maintain indexes for fast access.

```
CREATE [UNIQUE] INDEX <index_name> ON <table_name> ( <column_name>, ...) ;

DROP INDEX <index_name>;
```

3.1.9 Functions

The SQL standard allows you to create SQL functions and MonetDB support it. The syntax to create a function is:

```
CREATE FUNCTION <func_name> [(<param_name> <data_type>, ...)]
RETURNS func_data_type EXTERNAL NAME <external_func_name> ;

CREATE FUNCTION <func_name> [(<param_name> <param_type>, ...)]
RETURNS func_data_type routine_body ;

func_data_type:
<data_type>
    | TABLE (<column_name> <data_type>, ... )
    ;

routine_body:
procedure_statement
    |  BEGIN procedure_statement_list END
    | BEGIN ATOMIC procedure_statement_list END
    ;

procedure_statement:
        transaction_statement
    |   update_statement
    |   schema
    |   grant
    |   revoke
    |   create_statement
    |   drop_statement
    |   alter_statement
    |   declare_statement
    |   set_statement
    |   control_statement
    |   select_statement_single_row
    ;

control_statement:
        call_statement
    |   while_statement
    |   if_statement
    |   case_statement
    |   return_statement
    ;
```

To drop a function the syntax is:

```
DROP FUNCTION <func_name> [(parameters_type_list)] [CASCADE | RESTRICT] ;

DROP ALL FUNCTION <func_name> [CASCADE | RESTRICT] ;
```

Note: If you do not specify the full signature of the function the DROP query will successfully executed if there is only one function with this name, if not the query is aborted. The DROP ALL is used to drop all the functions with the name specified in the query.

3.1.10 Transactions

MonetDB/SQL supports a multi-statement transaction scheme marked by START TRANSACTION and closed with either COMMIT or ROLLBACK. The session variable AUTO_COMMIT can be set to TRUE if each SQL statement should be considered an independent transaction.

The transaction management scheme is based on the optimistic concurrency control scheme. It provides each transaction with a consistent view on the database, but updates are collected in an addendum processed on transaction commit. If at commit time it can be assured that the data prepared for update affects tables not changed in the mean time, the results are merged.

This scheme is particularly useful for query dominant environments. It negatively affects long running transactions which concurrently are affected by updates on their underlying tables.

```
START TRANSACTION;
...
SQL_STATEMENTS;
...
COMMIT; | ROLLBACK;
```

3.1.11 Schema Management

The scheme operations follow the standard definition.

```
CREATE SCHEMA <schema_name>
| AUTHORIZATION <auth_identifier>
| <schema_name> AUTHORIZATION <auth_identifier>
[DEFAULT CHARACTER SET <char_set>] ;

DROP SCHEMA <schema_name> [RESTRICT | CASCADE] ;
```

creation, public, search path, privileges system catalog

3.1.11.1 Database Roles

To define Database roles you should create your SQL statement base in the following syntax:

```
CREATE ROLE <role_name>
[ WITH ADMIN CURRENT_USER |
  WITH ADMIN CURRENT_ROLE ] ;

DROP ROLE <role_name>;
```

Note: By default the admin is the current user.

3.1.11.2 Privileges

3.1.11.3 Locale

3.1.12 Database Management

creation, destruction of the database

3.1.12.1 Backup and Recovery

The MONETDB script supports creation and restores of the database.

Alternatively, the database is stored by default in the directory (folder) DBFARM. Taking a snapshot of this directory is a save scheme to backup your database, provided you have first stopped its server.

3.1.12.2 Managing the Logs

The transaction logs are assembled in a database log directory whose location by default is next to DBFARM. The MONETDB script supports its management and the guardian program watches over the disk resource usage.

3.1.12.3 Monitoring Resources

The guardian process keeps an eye on the amount of free disk space and space occupied by the log files. If a high-water mark is reached, it will inform the database administrator through an email message.

3.1.13 Client Authorization

To create a user in the Database you should use the following syntax:

```
CREATE USER <user_name>
WITH PASSWORD <user_password>
NAME <name> SCHEMA <schema_name";

ALTER USER <user_name>
[WITH PASSWORD <user_password>]
[SET SCHEMA <new_schema_name>];

DROP USER <user_name>;
```

It can also be used to modify the user password or/and the default schema.

3.1.14 Catalog Inspection

The meta-information about the database objects is stored in a number of tables and internal data structures. They are updated through the CREATE, ALTER and DROP statements. They are available for inspection using a readonly copy or a materialized view.

Each session has a private copy of the catalog. Updates become effective at transaction commit only. <?niels when are the results visible to other users?>

3.1.14.1 Session Variables

MonetDB/SQL supports session variables declared by the user. They are indistinguishable from table and column names and can be used anywhere a literal constant is allowed.

```
sql>DECLARE high integer;
sql>DECLARE n varchar(256);
sql>SET high=4000;
sql>SET n='monetdb';
sql>SELECT count(*) from tables where id > high;
+--------+
| count_ |
+========+
|      2 |
+--------+
```

The SQL variables (and environment variables) can be accessed through predefined table returning functions VAR() and ENV(). The initial state is something like:

```
>select * from var() as v;
TO BE FILLED
```

- The DEBUG variable takes an integer and sets the server global debug flag (See MonetDB configuration documentation file). It also activates the debugger when the query is being executed.
- The PROFILE variable takes a boolean and when set collects execution statistics on the SQL queries executed in the table PROFILE.
- The TRACE variable takes a boolean and when set lists the execution timing of all MAL instructions to solve the SQL query.
- The CACHE variable takes a boolean and when set SQL uses a query cache to speed up subsequent calls to identical (up to constants) queries.
- The OPTIMIZER variable takes a string. It controls the query optimizers, see Chapter 3 [SQL Reference], page 40.

3.1.14.2 Environment Tables and Views

All the information of the catalog can is stored in tables. There are also some views over these tables to join some useful information about the several database objects existent in a MonetDB database.

To see all the environment and views tables you can consult the view tables.

```
for all tables and views:
>SELECT name from tables;

to select only the tables:
>SELECT name from tables where type = 0;
[ "schemas"     ] Table with all information about schema (name, authorization, owner).
[ "types"       ] Table with all the types existent and created by the user.
[ "functions"   ] Table with all functions existent and created by the user.
[ "args"        ] Table wich contains all the arguments for all the functions in the system.
[ "sequences"   ] Table with all information about the sequences created by the user.
[ "_tables"     ] Table with all information about tables and views on the system (there is an-
other for the temporary schema).
[ "_columns"    ] Table with all information about the columns for the existent tables (they also ex-
ist for the temporary schema).
[ "keys"        ] This table together with the table idx and keycolumns contain all the information
[ "idxs"        ] about the constraints created over tables or columns  (there is another for the tem-
porary schema).
[ "triggers"    ] Table with all information about the existent triggers (there is an-
other for the temporary schema).
[ "keycolumns"  ] Table with columns which are a Primary key or Foreign Key (there is an-
other for the temporary schema.
[ "dependencies"] This table contains all the dependencies between several database objects.
[ "connections" ] Table with all information about connection to remote catalogs.
[ "db_user_info"] This table together with user_role table contains all the information
[ "user_role"   ] about all users.
```

```
[ "auths"       ] This table together with privileges contains all the information about
[ "privileges"  ] privilegies and authorizations for all users.
[ "profile"     ] This table contains profile information about query executions for each user.


to select only the views:
>SELECT name from tables where type = 1;
[ "tables"      ] View over the system and the temporary tables table.
[ "columns"     ] View over the system and the temporary columns table.
[ "users"       ] View over the system and the temporary users table.
```

3.1.14.3 Environment Functions

3.1.14.4 Query Cache

SQL statements are translated a query cache retained for re-use. The cache can be inspected and manipulate with the following predefined SQL functions.

show_cache()	lists the content of the query cache and statistics of the call behavior
explain_cache(qryname)	list the query plan for a given query
drop_cache(qryname)	remove a specific plan from the cache
clear_cache()	removes all cached query plans

3.1.14.5 Administrator Functions

3.2 SQL Runtime Features

The MonetDB SQL implementation provides a few primitives to simplify profiling and debugging of SQL queries. They are described separately.

3.2.1 EXPLAIN Statement

The intermediate code produced by the SQL compiler can be made visible using the EXPLAIN statement modifier. It gives a detailed description of the actions taken to produce the answer. The example below illustrates what you can expect when a simple query is prepended by the EXPLAIN modifier. The details of this program are better understood when you have read the Chapter on Chapter 5 [MAL Reference], page 97.

```
sql>select count(*) from tables;
+--------+
| count_ |
+========+
|     27 |
+--------+
sql>explain select count(*) from tables;
+------------------------------------------------------------------------------+
| function user.s1_1():void;                                                   |
|     _1:bat[:oid,:int]{notnil=true,rows=0:lng,bid=1728}  :=                   |
| sql.bind("tmp","_tables","id",0);                                            |
|     constraints.emptySet(_1);                                                |
|     _10:bat[:oid,:sht]{notnil=true,rows=0:lng,bid=1732}  :=                  |
| sql.bind("tmp","_tables","type",0);                                          |
|     _13:bat[:oid,:int]{notnil=true,rows=30:lng,bid=854}  :=                   |
| sql.bind("sys","_tables","id",0);                                             |
|     _15:bat[:oid,:int]{notnil=true,rows=0:lng,bid=1588}  :=                   |
| sql.bind("sys","_tables","id",1);                                             |
|     constraints.emptySet(_15);                                                |
|     _15:bat[:oid,:int]{notnil=true,rows=0:lng,bid=1588}  := nil;              |
|     _16:bat[:oid,:int]{notnil=true,rows=0:lng,bid=1589}  :=                   |
| sql.bind("sys","_tables","id",2);                                             |
|     constraints.emptySet(_16);                                                |
|     _16:bat[:oid,:int]{notnil=true,rows=0:lng,bid=1589}  := nil;              |
|     _20:bat[:oid,:sht]{notnil=true,rows=30:lng,bid=860}  :=                   |
| sql.bind("sys","_tables","type",0);                                           |
|     _8{rows=0:lng} := algebra.markT(_1,0@0);                                  |
|     _9{rows=0:lng} := bat.reverse(_8);                                        |
|     _8{rows=0:lng} := nil;                                                    |
|     _12{rows=0:lng} := algebra.join(_9,_10);                                  |
|     _10:bat[:oid,:sht]{notnil=true,rows=0:lng,bid=1732}  := nil;              |
|     _18{rows=30:lng} := algebra.markT(_13,0@0);                               |
|     _19{rows=30:lng} := bat.reverse(_18);                                     |
|     _18{rows=30:lng} := nil;                                                  |
|     _21{rows=30:lng} := algebra.join(_19,_20);                                |
|     _20:bat[:oid,:sht]{notnil=true,rows=30:lng,bid=860}  := nil;              |
|     _22{rows=31:lng} := bat.setWriteMode(_21);                                |
|     _21{rows=30:lng} := nil;                                                  |
|     bat.append(_22,_12,true);                                                 |
|     _12{rows=0:lng} := nil;                                                   |
|     _26{rows=31:lng} := algebra.uselect(_22,nil:sht,2,false,false);           |
|     _22{rows=31:lng} := nil;                                                  |
|     _29{rows=31:lng} := algebra.markT(_26,0@0);                               |
|     _26{rows=31:lng} := nil;                                                  |
|     _30{rows=31:lng} := bat.reverse(_29);                                     |
|     _29{rows=31:lng} := nil;                                                  |
|     _31{rows=0:lng} := algebra.join(_9,_1);                                   |
|     _9{rows=0:lng} := nil;                                                    |
|     _1:bat[:oid,:int]{notnil=true,rows=0:lng,bid=1728}  := nil;               |
|     _32{rows=30:lng} := algebra.join(_19,_13);                                |
|     _19{rows=30:lng} := nil;                                                  |
|     _13:bat[:oid,:int]{notnil=true,rows=30:lng,bid=854}  := nil;              |
|     _33{rows=31:lng} := bat.setWriteMode(_32);                                |
|     _32{rows=30:lng} := nil;                                                  |
|     bat.append(_33,_31,true);                                                 |
|     _31{rows=0:lng} := nil;                                                   |
|     _35{rows=31:lng} := algebra.join(_30,_33);                                |
|     _30{rows=31:lng} := nil;                                                  |
|     _33{rows=31:lng} := nil;                                                  |
|     _36{rows=31:lng} := algebra.markT(_35,0@0);                               |
|     _35{rows=31:lng} := nil;                                                  |
```

```
|     _37{rows=31:lng} := bat.reverse(_36);                                     |
|     _36{rows=31:lng} := nil;                                                  |
|     _38{rows=1:lng} := aggr.count(_37);                                       |
|     _37{rows=31:lng} := nil;                                                  |
|     sql.exportValue(1,"sys.","count_","int",32,0,6,_38,"");                   |
| end s1_1;                                                                     |
+-------------------------------------------------------------------------------+
sql>
```

The SQL compiler keeps a limited cache of queries. Each query is looked up in this cache based on an expression pattern match where the constants may take on different values. If it doesn't exist, the query is converted into a code block and stored in the module USER.S0.

The call to the cached function is included in a wrapper function MAIN, which is the only piece of code produced if the query is used more than once. The query cache disappears when the server is brought to a halt.

```
+----------------------------+
| function user.main():void; |
|     mdb.start();           |
|     user.s3_1();           |
|     mdb.stop();            |
| end main;                  |
+----------------------------+
```

3.2.2 PROFILE Statement

The SQL implementation comes with a simple profiler to learn about the expensive queries. The profiler is controlled by the boolean session variable PROFILE. The snippet below illustrates its use:

```
sql> set profile= true;
  ...
sql>select * from profile;
+----------------------+--------------------+------+------+-----+------+--------+
| start                |query               |parse |optim |exec |total |user    |
|                      |                    |      |ize   |     |      |        |
+======================+====================+======+======+=====+======+========+
| 2007-08-14           |select * from       |  305 |  738 | 161 | 1204 |monetdb |
| 19:38:23.000000      |profile;            |      |      |     |      |        |
| 2007-08-14           |select * from       |  306 |  728 | 233 | 1267 |monetdb |
| 19:38:28.000000      |profile;            |      |      |     |      |        |
| 2007-08-14           |select count(*)     |  779 |  802 | 200 | 1781 |monetdb |
| 19:38:39.000000      |from tables;        |      |      |     |      |        |
+----------------------+--------------------+------+------+-----+------+--------+
```

It illustrates the wall-clock time the query was started, the query itself, followed by the wall-clock timing obtained for the parser, the optimizer, the execution phase and the total execution time (in microseconds). The final column indicates the user responsible for this request.

3.2.3 DEBUG Statement

The SQL statements are translated into MAL programs, which are optimized and stored away in an USER module. The generated code can be inspected with the MAL debugger. It provides a simple mechanism to trace the execution, hunting for possible errors and detect performance bottlenecks (Section 8.5 [Runtime Inspection], page 144).

The example below illustrates the start of such a session:

```
>debug select count(*) from tables;
#    mdb.start()
mdb>next
#    user.s1_0()
mdb>next
#    _2:bat[:oid,:int]  := sql.bind(_3="sys", _4="ptables", _5="id", _6=0)
mdb>next
#    _8:bat[:oid,:int]  := sql.bind(_3="sys", _4="ptables", _5="id", _9=1)
mdb> ...
```

3.2.4 TRACE Statement

Inspection of the execution time of the query plan uses the debugger facilities to time each request. The example below illustrates the TRACE statement modifier to obtain a first glimpse on the expensive components of the query plan.

```
sql>trace select count(*) from tables;
+-------------+-------------------------------------------------------------------+
|      2 usec |    mdb.setTimer(_2=true)                                          |
|     22 usec |    _1:bat[:oid,:int]  := sql.bind(_2="tmp", _3="_tables",         |
|             |_4="id", _5=0)                                                     |
|     13 usec |    constraints.emptySet(_1=<tmp_3300>bat[:oid,:int]{0})           |
|     11 usec |    _10:bat[:oid,:sht]  := sql.bind(_2="tmp", _3="_tables",        |
|             |_11="type", _5=0)                                                  |
|     12 usec |    _13:bat[:oid,:int]  := sql.bind(_14="sys", _3="_tables",       |
|             |_4="id", _5=0)                                                     |
|     11 usec |    _15:bat[:oid,:int]  := sql.bind(_14="sys", _3="_tables",       |
|             |_4="id", _6=1)                                                     |
|     10 usec |    constraints.emptySet(_15=<tmp_3064>bat[:oid,:int]{0})          |
|     11 usec |    _15:bat[:oid,:int]  := nil;                                    |
|     10 usec |    _16:bat[:oid,:int]  := sql.bind(_14="sys", _3="_tables",       |
|             |_4="id", _17=2)                                                    |
|      9 usec |    constraints.emptySet(_16=<tmp_3065>bat[:oid,:int]{0})          |
|      9 usec |    _16:bat[:oid,:int]  := nil;                                    |
|     11 usec |    _20:bat[:oid,:sht]  := sql.bind(_14="sys", _3="_tables",       |
|             |_11="type", _5=0)                                                  |
|     39 usec |    _8 := algebra.markT(_1=<tmp_3300>bat[:oid,:int]{0}, _7=0@0)    |
|     13 usec |    _9 := bat.reverse(_8=<tmp_3512>bat[:oid,:oid]{0})              |
|     10 usec |    _8 := nil;                                                     |
|     28 usec |    _12 := algebra.join(_9=<~tmp_3512>bat[:oid,:oid]{0},           |
|             |_10=<tmp_3304>bat[:oid,:sht]{0})                                   |
|     10 usec |    _10:bat[:oid,:sht]  := nil;                                    |
```

```
|     15 usec |    _18 := algebra.markT(_13=<tmp_1526>bat[:oid,:int]{30},         |
|             |_7=0@0)                                                            |
|      9 usec |    _19 := bat.reverse(_18=<tmp_3520>bat[:oid,:oid]{30})           |
|      9 usec |    _18 := nil;                                                    |
|     55 usec |    _21 := algebra.join(_19=<~tmp_3520>bat[:oid,:oid]{30},         |
|             |_20=<tmp_1534>bat[:oid,:sht]{30})                                  |
|     10 usec |    _20:bat[:oid,:sht]  := nil;                                    |
|     42 usec |    _22 := bat.setWriteMode(_21=<tmp_3506>bat[:oid,:sht]{30})      |
|      9 usec |    _21 := nil;                                                    |
|     11 usec |    bat.append(_22=<tmp_3506>bat[:oid,:sht]{30},                   |
|             |_12=<tmp_3412>bat[:oid,:sht]{0}, _24=true)                         |
|     15 usec |    _12 := nil;                                                    |
|     32 usec |    _26 := algebra.uselect(_22=<tmp_3506>bat[:oid,:sht]{30},       |
|             |_27=nil:sht, _25=2, _28=false, _28=false)                          |
|     16 usec |    _22 := nil;                                                    |
|     15 usec |    _29 := algebra.markT(_26=<tmp_3453>bat[:oid,:oid]{27},         |
|             |_7=0@0)                                                            |
|     10 usec |    _26 := nil;                                                    |
|      9 usec |    _30 := bat.reverse(_29=<tmp_3506>bat[:oid,:oid]{27})           |
|      8 usec |    _29 := nil;                                                    |
|     15 usec |    _31 := algebra.join(_9=<~tmp_3512>bat[:oid,:oid]{0},           |
|             |_1=<tmp_3300>bat[:oid,:int]{0})                                    |
|     13 usec |    _9 := nil;                                                     |
|      9 usec |    _1:bat[:oid,:int]  := nil;                                     |
|     30 usec |    _32 := algebra.join(_19=<~tmp_3520>bat[:oid,:oid]{30},         |
|             |_13=<tmp_1526>bat[:oid,:int]{30})                                  |
|     12 usec |    _19 := nil;                                                    |
|      8 usec |    _13:bat[:oid,:int]  := nil;                                    |
|     26 usec |    _33 := bat.setWriteMode(_32=<tmp_3522>bat[:oid,:int]{30})      |
|      9 usec |    _32 := nil;                                                    |
|     10 usec |    bat.append(_33=<tmp_3522>bat[:oid,:int]{30},                   |
|             |_31=<tmp_3366>bat[:oid,:int]{0}, _24=true)                         |
|     11 usec |    _31 := nil;                                                    |
|     22 usec |    _35 := algebra.join(_30=<~tmp_3506>bat[:oid,:oid]{27},         |
|             |_33=<tmp_3522>bat[:oid,:int]{30})                                  |
|     17 usec |    _30 := nil;                                                    |
|     14 usec |    _33 := nil;                                                    |
|     15 usec |    _36 := algebra.markT(_35=<tmp_3366>bat[:oid,:int]{27},         |
|             |_7=0@0)                                                            |
|     10 usec |    _35 := nil;                                                    |
|     10 usec |    _37 := bat.reverse(_36=<tmp_3522>bat[:oid,:oid]{27})           |
|      9 usec |    _36 := nil;                                                    |
|      9 usec |    _38 := aggr.count(_37=<~tmp_3522>bat[:oid,:oid]{27})           |
|     17 usec |    _37 := nil;                                                    |
+-------------+-------------------------------------------------------------------+
|  27                                                                             |
+-------------+-------------------------------------------------------------------+
|     21 usec |    sql.exportValue(_6=1, _40="sys.", _41="count_", _42="int",  |
|             |_43=32, _5=0, _44=6, _38=27, _45="")                            |
|   1646 usec |    user.s3_1()                                                 |
+-------------+----------------------------------------------------------------+
```

3.2.5 Optimizer Control

The code produced by MonetDB/SQL is massaged by several code optimizers to arrive at the best possible plan for evaluation. However, for development purposes and the rare case that more control is needed, the SQL session variable OPTIMIZER can be set to a list of optimizers to identify the steps needed.

```
sql>set optimizer='costModel,coercions,emptySet';
sql>select optimizer;
+------------------------------+
| single_value                 |
+==============================+
| costModel,coercions,emptySet |
+------------------------------+
```

The default SQL optimization sequence is defined as:

inline	Inline functions identified as such.
remap	Locate hardwired multiplex operations.
costModel	Inspects the SQL catalog for size information.
coercions	Performs static coercions.
emptySet	Removes all empty set expressions
accessmodes	Ensures that BATs for update are writeable.
aliases	Remove alias assignments.
commonterms	Hunts for common terms and retains one only.
accumulators	Re-uses BATs to hold the result of an arithmetic expression.
joinPath	Searchs multiple joins and glues them together for better optimization.
deadcode	Remove all code not leading to used results.
reduce	Reduces the stack space for faster calls.
garbageCollector	Injects calls to the garbage collector to free up space.
multiplex	Expand all remaining multiplex operations to iterators.

The final result of the optimizer steps become visible using the EXPLAIN statement modifier. Alternatively, the DEBUG statement modifier in combination with the 'o' command provides access to the intermediate optimizer results.

3.3 SQL programming

SQL has grown to become a full-fledged programming language, which permits applications to safely run complex code within server. The downside of this flexibility is an increase in server load.

MonetDB/SQL supports server side programming through the standards for SQL persistent modules, triggers, and direct binding to external functions.

3.3.1 Persistent Stored Modules

The persistent stored modules definitions are collected in text files. They should be loaded through the client interface as part of a session start.

The distribution comes with a few sample modules

query_cache()	table	produces a list of all cached queries.
bbp()	table	produces a list of all known BATs.

3.3.2 Triggers

Triggers are a convenient programming abstraction. They are activated at transaction commit based on updates to the base tables.

The following example provides a glimpse of their functionality:

```
create table t1 (id int, name varchar(1024));

--test FOR EACH STATEMENT (default one)
insert into t1 values(10, 'monetdb');
insert into t1 values(20, 'monet');
create trigger test5
        after update on t1
        for each statement
        when id >0 insert into t1 values(4, 'update_when_statement_true');
```

All trigger definitions are considered together at the transaction commit. There is no a priori defined order in which they run. Each may in turn activate new triggers, but each trigger definition is also executed only once per transaction commit.

3.3.3 External Functions

External functions provide a convenient bridge between SQL and MAL. This way also a bridge can be established with dynamically loaded functions written in C. Any SQL function signature can be bound to MAL function or command.

The example below illustrates a binding to the a function that produces a tabular overview of the BAT catalog.

```
CREATE FUNCTION bbp ()
    RETURNS TABLE (id int, name string, htype string,
        ttype string, count BIGINT, refcnt int, lrefcnt int,
        location string, heat int, dirty string, status string,
        kind string)
            EXTERNAL NAME sql.bbp;
```

A table returning function should be implemented as MAL function that returns a series of synchronized BATs.

```
function bbp():bat[:str,:bat];
    b := bat.new(:str,:bat,12);
    ns := bbp.getNames();
    ri := algebra.markT(ns, 0:oid);
...
kind := algebra.markH(ikind, 0:oid);
```

```
        bat.insert(b, "kind", kind);
        return b;
    end bbp;
```

Two MsSQL functions are also supported by MonetDB SQL. The functions supported are 'STUFF' and 'ROUND'. The functionality of both functions can be achived by calling the function with the prefix MS_.

```
>MS_ROUND(10.0, 1, 0);
```

The SQL definition for MS_ROUND is:

```
CREATE FUNCTION MS_ROUND(num float, precision int, truncat int)
RETURNS float
BEGIN
IF (truncat = 0)
THEN RETURN ROUND(num, precision);
ELSE RETURN TRUNCATE(num, precision);
END IF;
END;
```

The SQL definition for MS_STUFF is:

```
CREATE FUNCTION MS_STUFF( s1 varchar(32), st int, len int, s3 varchar(32))
RETURNS varchar(32)
BEGIN
DECLARE res varchar(32), aux varchar(32);
DECLARE ofset int;
SET ofset = 0;
SET res = SUBSTRING(s1,ofset,st-1);
SET res = res || s3;
SET ofset = LENGTH(s1)-len;
SET aux = SUBSTRING(s1,ofset, len);
SET res = res || aux;
RETURN res;
END;
```

4 XQuery

MonetDB/XQuery provides a full-fledged XQuery implementation, which adheres to all the typing rules prescribed in the W3C standard. It is constructed as an independent compiler, producing code for the MonetDB server back-end. Preliminary experiments against a well-known benchmark show its supreme performance and scalability in processing individual documents. Interactive query processing of large XML documents has finally become within reach.

The following W3C material is recommended:

- W3C XQuery 1.0 now full standard recommendation.
- W3C XPath 2.0 the XML navigation sub-language, common to both XQuery and XSLT.
- W3C XQuery Functions and Operators provides an overview of all built-in functions.
- W3C XQuery Update Facility is a proposal in-the-works for updating XML documents.

Additional W3C reference material:

W3C XML Schema 2.0 note that schema support in MonetDB/XQuery is untested at this point.

- W3C XQuery Serialization defines how XML documents should be converted to text.
- W3C XQuery Formal Semantics formally defines the behavior of all XQuery language constructs.

MonetDB/XQuery is based on technology originally developed within the Pathfinder project, at the University of Konstanz and is now pursued at the Technische Universit Munich, CWI Amsterdam and the University of Twente. A detailed account of this technology can be found on the Pathfinder website.

The documentation provided here is organized by feature area:

- Section 4.3 [Document Management], page 71 section Adding XML documents to a database, as well as deleting them, is not yet covered by the W3C. Thus, MonetDB/XQuery provides its own interface for such tasks.
- Section 4.4 [Supported Functions], page 75 section describes which features from the various W3C standard recommendations have been implemented, and which not. You will find that MonetDB/XQuery is very standard-conformant already at this stage.
- Section 4.5 [Prepared Queries], page 77 to accelerate repetitive fast-running queries, are a well-known feature in SQL databases. MonetDB/XQuery uses XQuery user-defined functions, defined in library modules (standard features in W3C XQuery 1.0) to bring the benefits of prepared queries also to XML database systems.
- Section 4.6 [StandOff Extensions], page 79 introduce new XPath steps to ease querying of XML documents with so-called **region-annotations**. The decision to extend a W3C standard is not taken lightly, and was only done after users from widely different backgrounds found these extensions critical. Also, these extensions are **turned off by default** and need to be actived explicitly by starting the MonetDB/XQuery server with the –SET STANDOFF=ENABLE command-line switch.

- ⟨undefined⟩ [XRPC Extension], page ⟨undefined⟩ is a minimal yet orthogonal extension for the XQuery language that adds the **function shipping** facility to XQuery by intoducing a single statement.
- Section 13.5 [The JDBC Library], page 242 describes how users and application programmers can interact with MonetDB/XQuery, either by invoking the the `mclient` utility on XQuery scripts, or by querying the MonetDB/XQuery server from a client program. We also describe how to use MonetDB/XQuery as XML database backbone behind a web server.

4.1 XQuery Overview

The success of XML(Extensible Markup Language) as a universal file format is still unbroken. Processing high volumes of XML documents, however, requires efficient means to store and query the data. XQuery, derived from both Xpath and SQL technology, has become the target for unlocking their potential for XML-based applications.

An XQuery processor is either developed from scratch, or exploits the decades of expertise in managing relational databases. The latter road has been followed in the Pathfinder research project, in which techniques are developed to benefit from modern *database technology*.

The vehicle for this approach is the MonetDB system with a proven track record in high performance database processing. MonetDB/XQuery is constructed as a front end solution that can handle multi-gigabyte XML> documents. Its implementation supports a substantial subset of XQuery's functionality; remaining features will follow.</p>

The MonetDB/XQuery system requires the installation of just two components: the MonetDB server and the XQuery compiler, which translates the expressions into MonetDB's intermediate language (MIL). The download section contains prepackaged versions for a variety of platforms.

Quality assurance is guaranteed using the stringent and automated testing of all products in the MonetDB <!–#include virtual="_ROOTDIR_/Development/TestWeb/link.shtml" –>. The testing section provides an outlook on the results of last night's builds, but is also a good source for small examples of the system's functionality.</p>

MonetDB/XQuery is a joint-effort of University of Munich (T.Grust, J. Teubner, J. Rittinger), University of Twente (M. van Keulen, J. Flokstra), and CWI (P. Boncz, S. Manegold, S. Mullender, Y. Zhang, N. Nes). Detailed technology information is organized in the separate Pathfinder project website. Contributions to maintenance and further improvements are highly appreciated.

The research focus and development efforts in the Pathfinder project are *scalability* and *standards compliance*. The packaging of the software aims at sharing our experiences and speeding up further research in this key technology area. MonetDB/XQuery is released under the Pathfinder Public License Version 1.1.

4.1.1 Database Scalability

The last decades of research brought us database management systems that store, retrieve and process huge amounts of data efficiently. They have been developed and implemented for various data models, though the relational model is by far the most popular and successful one.

The Pathfinder research effort pushes this development a step further and provides efficient relational database support for XML data (and its query language XQuery). The benchmark section illustrates the story in more detail.

4.1.2 Standard Compliance

The XQuery language has been defined without a strong existing user base or existing software solutions. This led to a clean design, but also one where many typing features greatly impact the implementation effort. The nature of the XQuery language is quite different from, e.g., the relational database processing model, which makes this translation a challenging task. Many research and commercial XQuery prototypes sacrifice some features to allow for an efficient translation.

Through careful design, our compiler still sticks to the XQuery working drafts</a>. The heart of the Pathfinder compiler is a translation approach that correctly copes with all important aspects of XQuery. The Research section contains more information on our compilation approach.

4.2 Tractable Interfaces

An XQuery implementation based on database back-end technology calls for a translation of the expressions into a low-level or algebraic intermediate language. This translation, and subsequent parsing and optimization, brings a fixed cost per query and can certainly be improved. But in the current state of technology, it provides a clear handle on debugging. Query caching often amortizes the cost of this performance-wise less preferable coupling.

4.2.1 Pathfinder Project

The *Pathfinder* research project has been initiated by the Database & Information Systems Group at the University of Konstanz, Germany.

All key people (the group of prof. dr. Torsten Grust) relocated in 2005 to the Technical University of Munich where the project continues unabated.

It is now a joint project with the database groups at CWI, Amsterdam, and the University of Twente, Enschede, The Netherlands.

The project strives for a full XQuery implementation, backed by modern database technology. As such, the techniques allow for effective optimizations and scalability to multi-gigabyte volumes of XML data.

The software implementation of these techniques is the open-source system MonetDB/XQuery. As an XQuery front end to MonetDB, the implementation *compiles* queries in the XQuery language into a relational equivalent. After applying various optimization techniques, the relational query is then sent as a MIL program (the low-level interface language to MonetDB) to the back-end database. After execution on the MonetDB back-end, the result is serialized back into an XML format to be returned to the user.

This XQuery compiler is distributed as a separate software package. The software includes the stand-alone XQuery compiler PF, as well as a server plug-in for MonetDB (the PATHFINDER *extension module* that contains the compiler and thus allows for server-side XQuery processing on MonetDB. The stand-alone compiler is mainly targeted people from the research community that want to look "under the hood" of MonetDB/XQuery. Users that just want to have a functional XQuery implementation will typically stick to the instant XQuery compilation provided by the pathfinder extension module.

4.3 Document Management

The MonetDB/XQuery database system manages a Document Collection. Users can Add Documents and Remove Documents to this collection by hand, giving them a logical name. Even without this, MonetDB/XQuery can query any XML document on the web for which you know a working URL. In that case, the document is automatically imported at query-time, and a Cache Policy determines whether and how long it stays in the database.

This page describes the interface for managing the document collection and the caching policy. These features are not covered by the W3C XQuery language itself, however. Therefore, for the moment, the management of these features is handled in MonetDB/XQuery through some special commands added to the MIL algebra, that can either be typed into the Mserver console directly, or using the mclient interface.

The upcoming release contains new XQuery **builtin functions** integrated in the update framework to allow document management to be performed using XQuery statements.

4.3.0.1 Document Collection

we can print the current document collection by:

```
mil> xmldb_print();
#---------------------------------------------------------------------------------#
# alias          URI                                                size            # name
# str            str                                                lng             # type
#---------------------------------------------------------------------------------#
[ "auctions.xml", "http://www.cwi.nl/~boncz/xmark//auctions.xml", 589824          ]
[ "music.xml",    "http://www.cwi.nl/~boncz/music/music.xml",     3276800         ]
mil>
```

In this case, two XML documents happen to be loaded. All documents have an alias (the logical name) and an uri (the physical name). The size printed is the database file storage space it occupies in bytes.

Documents are shredded either

- explicitly (as explained at the start), using the MIL command shred_doc("URI","alias");
- or because a query executed the XQuery function fn:doc("uri") and the Section 4.3 [Cache Policy], page 73 of MonetDB/XQuery determined that this document should stay in the cache; so it stayed there even after query execution ended (for automatically cached documents, the alias is identical to the URI).

In the former case, the document stays in the database until the point that it is explicitly removed with delete_doc(name), where name may be either the URI or the alias. The delete_all_docs(cachedOnly) deletes all documents if passed false and only the automatically cached ones if passed true.

4.3.0.2 Add Document

For example, to add the document HelloWorld.xml, to the database try the following:

```
shell> mclient -lmil
# Monet Database Server V4.8.2
# Copyright (c) 1993-2005, CWI. All rights reserved.
# Compiled for i686-redhat-linux-gnu/32bit; dynamically linked.
# Visit http://monetdb.cwi.nl/ for further information.
mil>shred_doc("/.../HelloWorld.xml","HelloWorld.xml");
mil>\q
```

Using the MIL language switch of `mclient` you directly interact with the server using the proprietary MonetDB server language. The command line returns with a language specific prompt. On this prompt you can type the MIL-code and to quit `mclient`, you simply press the Ctrl-D key combination or backslash-q.

Note: all statements in MIL have to be terminated with a ';' (semi-colon).

Note: you don't have to start a new MAPI session for each document you want to shred. You can simply call shred_doc(physical_name, logical_name) again during the same session.

A copy of the original document is now part of your persistent collection. The second argument of shred_doc is its logical name, which should be unique within the collection. Now we are able to issue XQuery queries on the document in our collection using its logical name:

```
doc("HelloWorld.xml")
```

Note: that issueing this XQuery should be done with `mclient` in XQuery mode (i.e. with option -lxquery instead of -lmil).

4.3.0.3 Remove Document

You can delete the "HelloWorld.xml" document from the collection using its logical name.

```
shell> mclient -lmil
# Monet Database Server V4.8.2
# Copyright (c) 1993-2005, CWI. All rights reserved.
# Compiled for i686-redhat-linux-gnu/32bit; dynamically linked.
# Visit http://monetdb.cwi.nl/ for further information.
>delete_doc("HelloWorld.xml");
>\q
```

The query doc("HelloWorld.xml") won't work anymore and returns an error message accordingly.

If you wish to delete all documents from the collection, type:

```
> delete_all_docs(true);
>
```

to delete only cached documents, or

```
> delete_all_docs(false);
>
```

to delete also the (explicitely) shredded documents.

4.3.0.4 Cache Policy

The **document cache** holds recently accessed XML documents. It is persistent accross Mserver sessions. There is a **caching policy** that determines whether and until when a document that is loaded by an executing XQuery will stay in the cache.

An important parameter is the size of the cache. It is controlled by the XQUERY_CACHEMB variable in the MONETDB.CONF file. This value is in megabytes (MB). From MIL, you can inspect (but not modify) these settings using:

```
mil> env();
#-------------------------------------------------------------#
# h               t                                           # name
# str             str                                         # type
#-------------------------------------------------------------#
[ "config",        "/home/boncz/MonetDB/Linux/etc/MonetDB.conf" ]

...lines omitted...

[ "mapi_port",     "50000"                                    ]

...lines omitted...

[ "xquery_output", "dm"                                       ]
[ "xquery_cacheMB","100"                                      ]
mil>
```

To modify the settings, edit the MONETDB.CONF file. Its exact filepath location is listed under "CONFIG" in the same table.

Apart from the cache size, there are **rules** that control the behavior of the document cache. A general rule is that MonetDB/XQuery **always** caches file URIs. The system uses the last-modified timestamp of the filesystem to guarantee that when a query is run, the cached document is actual (if it has changed on disk, the document is removed from the cache automatically, and is shredded anew).

Other caching **lifetime-rules** can be added explicitly with the XMLCACHE_ADD_RULE("URI-PREFIX", SECONDS); command. Such a rule states that all documents whose URI starts with the specified prefix should be cached for a maximum amount of seconds (in case of multiple lifetime rules prefix-matching and URI, the longets match counts).

Such rules are persistent and remain in place until explicitly revoked with XMLCACHE_DEL_RULE("URI-PREFIX");. The currently active set of lifetime rules can be inspected with:

```
mil> xmlcache_print_rules();
#--------------------------------#
# URI-prefix    liftime-secs      # name
# str           lng               # type
#--------------------------------#
mil>
```

no rules, currently!

Below an example of how to add two lifetime rules:

- cache all HTTP URIs for 5 minutes.
- cache all documents from HTTP://WWW.CWI.NL for a month.

```
mil> xmlcache_add_rule("http://", 300);
mil> xmlcache_add_rule("http://www.cwi.nl", 30*24*60*60);
mil> xmlcache_print_rules();
#-----------------------------------------#
# URI-prefix            liftime-secs      # name
# str                   lng               # type
#-----------------------------------------#
[ "http://",              300             ]
[ "http://www.cwi.nl",    2592000         ]
mil>
```

4.3.0.5 Miscellaneous

MonetDB/XQuery offers a few other functions that might turn out to be useful. The function signatures are shown using SIGS("PATHFINDER");.

Sigs:

```
mil> sigs("pathfinder");
#--------------------------------------------------------#
# signature                                              # name
# str                                                    # type
#--------------------------------------------------------#
[ "shred_doc(str, str) : void"                           ]
[ "xmldb_print() : void"                                 ]
[ "delete_doc(str) : void"                               ]
[ "delete_all_docs(bit) : void"                          ]

.. lines omitted ...

[ "xmlcache_print() : void"                              ]
[ "xmlcache_print_rules() : void"                        ]
[ "xmlcache_add_rule(str, any) : void"                   ]
[ "xmlcache_del_rule(str) : void"                        ]
mil>
```

The above MIL commands are the interface for managing the document collection, as explained below. You may also try HELP("COMMAND"); to get a summary for each command:

```
mil> help("xmldb_print");
PROC:      xmldb_print() : void
MODULE:    pathfinder
COMPILED:  by boncz on May 2005
DESCRIPTION:
shows the actual content of the persistent XML document database (not the XML document ca
This consists of all documents explicitly shredded with shred_doc(URI, alias).█
mil>
```

4.4 Supported Functions

MonetDB/XQuery supports a wide range of the built-in functions described in the W3C specifications. Here is a list of functions that MonetDB/XQuery currently supports. Some functions are not yet implemented, while others probably won't be implemented in the near future.</p>

yes	**Order Awareness** MonetDB/XQuery correctly implements document and sequence orders, as well as node identity.
yes	**XPath Location Steps** MonetDB/XQuery implements XQuery's *full axis feature*, i.e., we support all 12 XPath axes. Note, however, that we do not support node tests on *type annotations* as introduced with newer XQuery drafts.
yes	**FLWOR clauses** We support FLWOR clauses with full generality and arbitrary nesting, including positional variables.
yes	**Arithmetics, Logics, Conditionals** We support arithmetics, logics (AND, OR), and conditionals (SOME/EVERY, IF-THEN-ELSE).
partly	**Node Construction** We fully support element, attribute, and text constructors, at arbitrary nesting depth. We have not implemented document, comment and processing-instruction constructors, yet.
yes	**Namespaces** MonetDB/XQuery fully supports namespaces.
yes	**Schema Import** MonetDB/XQuery implements the *Schema Import* feature. This is not well tested, though.
partly	**Module Import** MonetDB/XQuery implements the *Module Import* feature, however its handling of recursive module imports is non-standard. Most importantly, the W3C formal semantics specify that if a query (or a module) imports a module, only the variables and functions directly defined by that module will become visible to it. In MonetDB/XQuery, though, **all** namespaces, variables and functions that were imported by that module, will become visible. Thus MonetDB/XQuery may encounter clashes (leading to errors), e.g. in namespace identifiers, or funtion signature or variable names, that should not occur according to the formal semantics.
partly	**XQuery Built-In Functions** We support a large set of built-in functions, as listed in our <a href='../Functions/'>Function Library</a>. We currently do *not* support functions that involve specific collation orders, as well as functions that involve date/time conversions.

yes	**User-Defined Functions** MonetDB/XQuery correctly deals with user-defined functions, with or without recursion. Current development efforts will lead to an even more efficient implementation that completely eliminates function call overhead.
partly	**Simple Types** We currently support the XQuery simple types <tt>xs:integer</tt>, <tt>xs:decimal</tt>, <tt>xs:double</tt>, <tt>xs:string</tt>, and <tt>xs:boolean</tt>. Note that we currently implement <tt>xs:decimal</tt> as a floating point number which may lead to rounding errors.
yes	**Static Typing** MonetDB/XQuery supports the *Static Typing* feature. (For experts: we actually support *structural* typing here, using Antimirov's algorithm.)
no	**Dynamic Typing** Our implementation of XQuery's TYPESWITCH clause is still very limited. We currently allow type tests that can be decided at compile time, and tests for *atomic* types. Note that the lack of dynamic typing may also limit XQuery's casting functionalities.
no	**Validation**<br/>MonetDB/XQuery does not yet support validation, though work is underway to close that gap.

MonetDB/XQuery supports a wide range of the built-in functions described in the W3C specifications. This section provides an overview is of functions that MonetDB/XQuery currently supports.

The function library has been organized as follows:

AGGREGATION FUNCTIONS (MIN,MAX, ..).
: **supported: 6/8** Missing functions are all collation specific, See ⟨undefined⟩ [XQuery Aggregation], page ⟨undefined⟩.

NUMERIC COMPUTATION FUNCTIONS (+ -, ..).
: **supported: 18/20** No support yet of ROUND-HALF-TO-EVEN, See ⟨undefined⟩ [XQuery Numeric], page ⟨undefined⟩.

BOOLEAN LOGIC OR COMPARISON FUNCTIONS (AND, &GT;, ..).
: **supported: 9/13** We miss DEEP-EQUAL and `binaryEqual` variants, See ⟨undefined⟩ [XQuery Boolean], page ⟨undefined⟩.

STRING FUNCTIONS (SUBSTRING, ..).
: **supported: 21/32** We miss collation-speific versions as well as NORMALIZE-UNICODE and `tokenize`, See ⟨undefined⟩ [XQuery String], page ⟨undefined⟩.

NODE FUNCTIONS (ROOT, DOC, ..).
: **supported: 17/23**, See ⟨undefined⟩ [XQuery Node], page ⟨undefined⟩.

SEQUENCE FUNCTIONS (FIRST, LAST, ..).
: **supported: 11/19** No support for COLLECTION yet, See ⟨undefined⟩ [XQuery Sequence], page ⟨undefined⟩.

URI FUNCTIONS (BASE-URI, ESCAPE-URI, ..).
: **supported: 2/10** We only have NAMESPACE-URI, See ⟨undefined⟩ [XQuery URI], page ⟨undefined⟩.

QName functions (node-name, resolve-QName, ..).
supported: 0/6 Not supported yet, See ⟨undefined⟩ [XQuery QName], page ⟨undefined⟩.

Runtime functions(error, trace ..).
supported: 0/5 Not supported yet, See ⟨undefined⟩ [XQuery Runtime], page ⟨undefined⟩.

Date/Time functions(date-equal, time-less-than, ..).
supported: 0/75 Not supported yet, See ⟨undefined⟩ [XQuery Date/Time], page ⟨undefined⟩.

We encourage new Developers to volunteer and help fill in these blanks. Work has started on adding date/time functions but help is surely welcome.

4.5 Prepared Queries.

MonetDB/XQuery has support for modules. It helps XQuery users to structure their query code, but are also the instrument for MonetDB/XQuery to implement **canned queries**.

4.5.1 XQuery Modules

The below shows a simple example of an XQuery module test.xq, that just defines a single function countDescendants("uri"):

```
module namespace test = "http://monetdb.cwi.nl/XQuery/Documentation/Language/Modules/";

declare function test:countDescendants($doc as xs:string) as xs:integer
{
  count(doc($doc)//*)
};
```

You may type import module inside an XQuery query, after which you can use the functions (and variables) defined in it:

```
import module namespace test = "http://monetdb.cwi.nl/XQuery/Documentation/Language/Modul
                               at "http://monetdb.cwi.nl/XQuery/Documentation/Language/Modul
test:countDescendants("http://monetdb.cwi.nl/xmark/auctions.xml")
```

which basically does the same as the ad-hoc query, namely counting how many nodes the XMark document has:

```
count(doc("http://monetdb.cwi.nl/xmark/auctions.xml")//*)
```

Warning: while highly similar, the module feature as implemented by MonetDB/XQuery deviates in the following respects from the XQuery formal semantics:

- You *must* give a location hint in the "import module" statement. Each file hinted there will be loaded as a module. It has to match the namespace given in the "import module" statement, though.
- Modules cannot see variables declared in other modules, regardless if they imported the module theirselves or not. A module is not allowed, though, to override variable declarations of other modules (conforming to the specs).
- Modules will see functions defined in other modules. They are not allowed to override them, though.

- All modules and the main query share the same type definitions. So modules will see XML Schema definitions imported by the main query. (see also below for XML Schema import)
- Pathfinder does allow cyclic importing of modules, regardless of their namespace.
- The XQuery specifications state that two module import statements that use the same target namespace should produce an error. This is not the case in MonetDB/XQuery: the module will be loaded once, but its functions and variables will be available under both namespace identifiers.

4.5.2 Prepared Queries using Functions from Modules

In SQL databases, interfaces like ODBC or JDBC allow to prepare **prepared queries** for faster execution of frequently used query patterns. For example, web-sites powered by a database usually generate a **fixed** set of queries to that database; each page fires off the same (set of) queries, possibly though with different parameters (that contain e.g. a customer identity or cookie). SQL systems can **prepare** for such prepared queries in advance, such that when a web-page is served out and the query result is needed quick, only the actual parameters need to be fed (**bound**) into a ready-to-run query (saving query parsing and optimization time, which is unavoidable for ad-hoc queries).

Such prepared queries can be seen as a queries whose result is a function of a number of parameters. MonetDB/XQuery takes this definition literally in its implementation of prepared queries:

prepared query = XQuery function

XQuery modules allow to define XQuery functions (and variables, but that's beside the point now). So:

- if a XQuery consists of only a function call, and
- that function is defined not inside the query itself, but inside a module (so there's an import module statement preceding the function call), and
- the query parameters are simple constants (atomic values of type xs:integer, xs:decimal, xs:double, or xs:string)

then MonetDB/XQuery will execute this query much faster, making use of a **prepared query plan**.

To put the caching mechanism to the test, first import the XMark document into the database by typing in the MonetDB console:

```
mil&gt; shred_doc("http://monetdb.cwi.nl/xmark/auctions.xml",
               "http://monetdb.cwi.nl/xmark/auctions.xml");
```

then run the queries q.xq (standard), m.xq (function from module) while monitoring elapsed time:

```
shell> mclient -lx -t q.xq
1729

Trans      18.000 msec
Shred       0.000 msec
Query       5.000 msec
Print       0.000 msec
```

```
Timer      24.436 msec

shell> mclient -lx -t m.xq
1729

Trans       0.000 msec
Shred       0.000 msec
Query       2.000 msec
Print       0.000 msec
Timer       2.778 msec
```

We see that the latter "prepared" query (that uses a function defined in a module) performs almost **ten times** faster than the ad-hoc query! The translation time has completely disappeared for the prepared query M.XQ and the query execution also improved by a factor of two.

This performance trend was confirmed on the XMark benchmark. We developed prepared queries for the benchmark, which produced a factor 10 (small 116KB documents) to 2 (larger 10MB documents) performance increase over the ad-hoc benchmark queries. Obviously, complex queries or queries on huge (>=GB) documents, which take many seconds to run anyway, do not benefit from prepared query mechanism (but neither are hindered by it).

As a general rule, the benefit of canned queries is larger for short-running than for long-running queries, because for short-running queries the XQuery translation and optimization time weighs in more heavily. We especially recommend the use of canned queries when MonetDB/XQuery is used to power web-sites.

4.6 StandOff Extensions

XML is often used to store annotations (i.e. meta-data, data describing other data). In particular, XML **Standoff Annotation**, concern annotations that annotate some object that itself is **not** included in the XML document. Such StandOff annotation often refers to *regions* in this object. We support a form of XML annotations that denotes these regions as XML node attributes called START and END.

Consider, for example, a video file (documentary) annotated as follows:

```
<sample>
 <video>
  <scene id="Intro" start="0" end="800"/>
  <scene id="Interview" start="801" end="10400"/>
  <scene id="Outro" start="10401" end="13400"/>
 </video>
 <music>
  <song artist="Beatles" start="0" end="4500"/>
  <song artist="Bach" start="10000" end="13000"/>
 </music>
</sample>
```

On the above example XML file (multimedia case), one may want to ask which music was played during the interview. In that case, we want SONG elements whose regions *overlap* with the Interview SHOT. Without StandOff extensions, such queries are tedious to express in XQuery, and perform very slowly.

with StandOff extensions, the query can be posed as follows:

```
doc("example.xml")//scene[@id="Interview"]/select-wide::song
```

Note the SELECT-WIDE is an extension of the XPath (and thus XQuery) syntax.

4.6.1 Extended XPath Steps for Region Querying

The StandOff axis steps, similar in behavior to the standard XPath steps (e.g. CHILD::*, DESCENDANT::*) have been added to MonetDB/XQuery to make querying concurrent such region really easy.

- /SELECT-NARROW::
- /SELECT-WIDE::
- /REJECT-NARROW::
- /REJECT-WIDE::
- Axis steps will always be 'local' e.g. will only yield matches from the same document (fragment).
- Each node is only returned once (no duplicates) and in document order.

4.6.1.1 context/select-narrow::nodename

From the set of nodes with nodename 'nodename', say: {n1, n2...}, return only the nodes contained in the context nodes (e.g. return n if there is a context node for which holds: context_start <= n_start and n_end <= context_end)

4.6.1.2 context/select-wide::nodename

From the set of nodes with nodename 'nodename', say: {n1, n2...}, return only the nodes overlapping with the context nodes (e.g. return n if there is a context node for which holds: context_start <= n_end and n_start <= context_end)

4.6.1.3 context/reject-narrow::nodename

From the set of nodes with nodename 'nodename', say: {n1, n2...}, return all BUT the nodes contained in the context nodes (e.g. return n if there is NO context node for which holds: context_start <= n_start and n_end <= context_end)

4.6.1.4 context/reject-wide::nodename

From the set of nodes with nodename 'nodename', say: {n1, n2...}, return all BUT the nodes overlapping with the context nodes (e.g. return n if there is NO context node for which holds: context_start <= n_end and n_start <= context_end)

4.6.2 Availability

The steps have been made available in MonetDB/XQuery next to the regular XPath axis. The StandOff steps have been **turned off by default** as they do not follow the XQuery recommendation as set by the W3C. To *enable* the steps you need to start the database server (Mserver) with the option –SET STANDOFF=ENABLE.

Inside the server, the StandOff steps are implemented efficiently using sophisticated *interval-join* algorithms, as well as a *temporal index*. Both are employed automatically by MonetDB/XQuery, without need of user or DBA intervention. The `http://www.cwi.nl/htbin/ins1/publications?request=intabstract&key=AlBhVrBo:XIMEP:06">XIME-P` 2006 paper from our scientific library gives technical background on these StandOff extensions.

4.6.3 Why Extending the XPath Standard?

We have found a surprising wide variety of XML data owners to have region annotations:

- Standoff In Multimedia: XML that holds the output of video scene detection or speech recognition tools (etc.). Used in various kinds of content-based multimedia search/browsing systems.
- Standoff In Forensic: XML describing interesting features discovered on confiscated hard drives (e.g. person names, addresses, emails, recovered file hierarchies, etc..). The regions refer to the positions on disk where the features where found. Used in computer-assisted crime scene investigations (CSI).
- Standoff In NLP: XML describing the grammatical structure of natural texts. Inline annotation cannot be used because natural language is ambiguous, and multiple parses are often possible. Thus structure is separated from content, and refers to it by word position. Used in automatic question answering systems.
- Standoff In Bio-Informatics: XML storing DNA sequences annotated by genome research groups. The regions refer by position in the DNA strands. The annotations may contain clinical characteristics of patients or hold additional biomolecular data on those genes. Used in collaborative genome research efforts.

If you have similar XML data and use MonetDB/XQuery to manage this, please contact us on the mailing list.

For XQueries with such region overlap/containment conditions, other XML database systems resort to query plans that have to compare all pairs of regions ("quadratic complexity"). On XML data sizes above a few hundred KB, this quickly systems become unusably slow. In contrast, MonetDB/XQuery with StandOff extensions runs bio-informatics queries on gigabytes of XML annotations within a few seconds.

4.7 XRPC Extension

The XQuery 1.0 language only provides a *data shipping* model for querying XML documents distributed on the Internet. The built-in function `fn:doc()` fetches an XML document from a remote peer to the local server, where it subsequently can be queried. The recently published W3C working draft of XQuery Update Facility (XQUF) introduces the built-in function `fn:put()` for remote storage of an updated document, which again implies data shipping.

To equip XQuery with *function shipping* style distributed querying abilities, we introduce **XRPC**. XRPC is a minimal yet powerful XQuery extension that enables efficient distributed querying of heterogeneous XQuery data sources. XRPC enhances the existing concept of XQuery functions with the Remote Procedure Call (RPC) paradigm. By calling out of an XQuery `for`-loop to multiple destinations, and by calling functions that themselves perform XRPC calls, complex P2P communication patterns can be achieved.

The XRPC extension is orthogonal to all XQuery features, including XQUF. Hence, in all places where a function application is allowed by the XQuery 1.0 language, an XRPC function call can be placed. All functions defined in an XQuery module can be called remotely, provided that both the caller and the callee of the function have access to the same module definition file. All XQUF updating expressions[1] can be included in the definition of an updating XQuery module function, which then can be called with XRPC.

The XRPC extension is enabled by default in MonetDB/XQuery. It is compiled together with the `pathfinder` module. XRPC has two major components, a request handler (module `xrpc_server`) and a message sender (module `xrpc_client`). Both module are loaded when the module `pathfinder` is loaded in `Mserver`.

4.7.1 XRPC Syntax

Remote function applications take the XQuery syntax:

`execute at` $\{Expr\}\{FunApp(ParamList)\}$

where *Expr* is an XQuery expression that specified the URI of the peer on which the function *FunApp* is to be executed.

For a precise syntax definition, we show the rules of the XQuery 1.0 grammar that were changed:

```
PrimaryExpr  ::= ... | FunctionCall | XRPCCall | ...                    XRPCCall     ::= "execu
```

We restrict the function application *FunApp* to user-defined functions that are defined in a module. Thus, the defining parameters of an XRPC call are: *(i)* a module URI, *(ii)* a function name, and *(iii)* the actual parameters (passed by value). The module URI is the one bound to the namespace identifier in the function application. The module URI *must* be supplemented by a so-called *at*-hint, which also is a URI.

The current choice to allow functions defined in XQuery modules is due to efficiency and security reasons. MonetDB/XQuery has the mechanism of caching the query plan of a module. For all subsequent use of the functions in a cached module, only the function parameters need to be extracted to be feed directly into the query plan. For security reason, by allowing only modules, it is trivial to specify which modules are allowed to be executed or not.

It is **important** to know that actual parameters of the called function are passed **by value** (in contrary to *by reference*), which implies that if an XML node is passed as a parameter of an XRPC call, only its subtree is serialised in the request message and sent to the remote site.

4.7.2 XRPC Call Examples

As a running example, we assume a set of XQuery database systems (peers) that each store a film database document "*filmDB.xml*" with contents similar to:

```
<films>
  <film>
    <name>The Rock</name>
    <actor>Sean Connery</actor>
  </film>
```

[1] The `transform` expression is not supported yet.

```
    <film>
      <name>Goldfinger</name>
      <actor>Sean Connery</actor>
    </film>
    <film>
      <name>Green Card</name>
      <actor>Gerard Depardieu</actor>
    </film>
  </films>
```

We assume an XQuery module "*film.xq*" stored at "*example.org*", that defines a function `filmsByActor()`:

```
module namespace film="films";

declare function film:filmsByActor($actor as xs:string) as node()*
{ doc("filmDB.xml")//name[../actor=$actor] };
```

We can execute this function on remote peer "*x.example.org*" to get a sequence of films in which Sean Connery plays in the remote movie database:

```
import module namespace f="films" at "http://example.org/film.xq";

<films> {                                                          (Q1)
  execute at {"x.example.org"} {f:filmsByActor("Sean Connery")}
} </films>
```

Above example yields (white spaces have been added for readability):

```
<films>
  <name>The Rock</name>
  <name>Goldfinger</name>
</films>
```

All functions defined in an XQuery module can be called remotely, *provided* that both the XRPC client and the XPRC server can access the *same* module definition file. **Beware** that the XRPC server does not check if it is indeed accessing the *same* module definition file as meant by the caller. Hence, if the XRPC client uses a local file, and the XRPC server happens to have a file on the server's local system with the same name but different contents, the query can produce unexpected results. It is up to the query writer to prevent this problem from happening.

More examples. A more elaborate example demonstrates the possibility of multiple remote function calls to a peer:

```
import module namespace f="films" at "http://example.org/film.xq";

<films> {
  for $actor in ("Julie Andrews", "Sean Connery")                  (Q2)
  return
    execute at {"x.example.org"} {f:filmsByActor($actor)}
} </films>
```

To make it a bit more complex, we could do multiple function calls to multiple remote peers:

```
import module namespace f="films" at "http://example.org/film.xq";

<films> {
  for $actor in ("Julie Andrews", "Sean Connery")                  (Q3)
  for $dst in ("x.example.org", "y.example.org")
  return execute at {$dst} {f:filmsByActor($actor)}
} </films>
```

Complex communication patterns may be programmed with XRPC, especially if recursive functions are used:

```
module namespace film="filmdb";

declare function
film:recursiveActor($dsts as xs:string*, $actor as xs:string) as node()*
{
    let $cnt := fn:count($destinations)
    let $pos := ($cnt / 2) cast as xs:integer
    let $dsts1 := fn:subsequence($destinations, 1, $pos)
    let $dsts2 := fn:subsequence($destinations, $pos+1)
    let $peer1 := $destinations[1]                                 (Q4)
    let $peer2 := $destinations[$pos]
    return (
        if ($cnt > 1) then
            execute at {$peer1} {film:recursiveActor($dsts1, $actor)}
        else (),
        doc("filmDB.xml")//name[../actor=$actor],
        if ($cnt > 2) then
            execute at {$peer2} {film:recursiveActor($dsts2, $actor)}
        else ())
};
```

The above function executes the function `recursiveActor` on a set of destination peers, uniting all results, and does so by constructing an binary spanning tree of recursive XRPC calls.

4.7.3 XRPC Server

As from version 0.16, the module `xrpc_server` is loaded automatically when loading the module `pathfinder`:

```
$ Mserver --dbinit="module(pathfinder);"
...
# XRPC administrative console at http://localhost:50001/admin
MonetDB>
```

By default, the XRPC server listens to the port number (`mapi_port + 1`). As the default `mapi_port` number is 50000, the default `xrpc_port` number is thus 50001 (please note the port number of the XRPC administrative console above).

Change `mapi_port` will change de XRPC server port as well (please not the different port number of the XRPC administrative console below):

```
$ Mserver --set mapi_port=60000 --dbinit="module(pathfinder);"
...
# XRPC administrative console at http://localhost:60001/admin
MonetDB>
```

Of course, the `xrpc_port` can be also set explicitly with the `--set var=value` option and this will not affect the value of `mapi_port`:

```
$ Mserver --set xrpc_port=60000 --dbinit="module(pathfinder);"
...
# XRPC administrative console at http://localhost:60000/admin
MonetDB>monet_environment.find("mapi_port").print();
[ "50000" ]
MonetDB>monet_environment.find("xrpc_port").print();
[ "60000" ]
MonetDB>
```

Trusted Modules. For security reasons, the XRPC server won't execute an arbitrary module, instead, it will only execute those modules which location (given by the `at`-hint) has the same prefix as one of the values listed in the MIL variable `xrpc_trusted`:

```
MonetDB>xrpc_trusted.print();
#-----------------------------------------------#
# h                                     t       # name
# str                                   void    # type
#-----------------------------------------------#
[ "http://monetdb.cwi.nl",              nil     ]
[ "${prefix}/share/MonetDB/xrpc/export", nil     ]
```

Hence, by default, only modules stored in the MonetDB domain and in the `export` directory can be called. Calls to functions in un-trusted modules will be rejected with an HTTP response code 403.

The default value of `xrpc_trusted` can be overwritten or extended:

```
$ Mserver --set xrpc_trusted="http://example.org/" --dbinit="module(pathfinder);"
...
MonetDB>xrpc_trusted.print();
#--------------------------------#
# h                     t         # name
# str                   void      # type
#--------------------------------#
[ "http://example.org/",  nil     ]
MonetDB>xrpc_trusted.insert("http://monetdb.cwi.nl", nil);
MonetDB>xrpc_trusted.print();
#----------------------------------------#
# h                             t         # name
# str                           void      # type
#----------------------------------------#
[ "http://example.org/",          nil     ]
[ "http://monetdb.cwi.nl",        nil     ]
```

By removing the MIL varialbe `xrpc_trusted`, one can instruct the XRPC server to accept any modules:

```
MonetDB>xrpc_trusted.delete();
```

The XRPC Administrative Console. The XRPC extention comes with an adminstrative console, which provides web interface to the XQuery funcions defined in:

`${prefix}`/share/MonetDB/xrpc/admin/admin.xq

The admin console is available via the URL:

http://localhost:`<xrpc_port>`/admin

By default, the admin console can only be accessed via the localhost. This is guarded by the MIL variable `xrpc_admin`:

```
$ Mserver --dbinit="module(pathfinder);"# Monet Database Server V4.18.1
...
# XRPC administrative console at http://localhost:50001/admin
MonetDB>xrpc_admin.print();
#-------------------------#
# h             t         # name
# str           void      # type
#-------------------------#
[ "127.0.0.1",    nil     ]
```

To access the admin console from a remote host, the IP address must be listed in `xrpc_admin`. Again, this can be done by overwriting or extending the value of `xrpc_admin`:

```
$ Mserver --set xrpc_admin="123.123.123.123" --dbinit="module(pathfinder);"
...
# XRPC administrative console at http://localhost:50001/admin
MonetDB>xrpc_admin.print();
#--------------------------------#
# h                     t         # name
# str                   void      # type
#--------------------------------#
[ "123.123.123.123",      nil     ]
MonetDB>xrpc_admin.insert("123.123.123.124",nil);
MonetDB>xrpc_admin.print();
#--------------------------------#
# h                     t         # name
# str                   void      # type
#--------------------------------#
[ "123.123.123.123",      nil     ]
[ "123.123.123.124",      nil     ]
```

File Serving. The XRPC server is a simple HTTP server as well. It serves all files stored in:

`${prefix}`/share/MonetDB/xrpc

For example, the dummy XQuery module "*export.xq*" that is standard installed in:

`${prefix}`/share/MonetDB/xrpc/export

can be retrieved using the URL:

http://`<yourhost>`:`<xrpc_port>`/export/export.xq

Directory listing is turned off and it can only be turned on by changing the XRPC source code.

4.7.4 The SOAP XRPC Message Format

The design goal of XRPC is to create a distributed XQuery mechanism with which different XQuery processors at different sites can jointly execute queries. This implies that our XRPC extension also encompasses a *network protocol*.

Network comminicatin in XRPC uses the Simple Object Access Protocol (SOAP), i.e. XML messages over HTTP. The SOAP XRPC message format is defined in **XRPC.xsd**. According to the classification in the article "Discover SOAP encoding's impact on Web service performance", the SOAP XRPC protocol belongs to the family of "*document/literal*". Note that SOAP XRPC should not be confused with SOAP RPC, a sub-protocol defined by the SOAP 1.2 standard[2].

XRPC Request Message. SOAP messages consist of an envelope, with an optional `Header` element and a `Body` element. Inside the body, we define a `request` element with several attributes:

- required attributes
  - `module`: namespace of the XQuery module
  - `method`: name of the called function
  - `arity`: the number of parameters the called method has
  - `location`: the *at*-hint, i.e. the location where the module file is stored.
- optional attributes
  - `iter-cnt`: number of iterations included in this request
  - `updCall`: is the called function an updating function (as defined by XQUF) or not. Note that the pathfinder document management functions (e.g. `pf:add-doc()`) are also considered to be updating functions by XRPC.

The actual parameter values of a single function call are enclosed by a `call` element. Each individual parameter consists of a `sequence` element, that contains zero or more values.

Below we show the SOAP XRPC request message generated for the first example query (Q2) that looks for films played by Sean Connery:

[2] SOAP RPC is oriented towards binding with programming languages such as C++ and Java, and specifies parameter marshaling of a certain number of simple (atomic) data type. However, its supported atomic data types do not match directly those of the XQuery Data Model (XDM), and the support for arrays and structs is not relevant in XRPC, where there rather is a need for supporting arbitrary-shaped XML nodes as parameters as well as sequences of heterogeneously typed items. This is the reason why SOAP XRPC message format, while supporting the general SOAP standard over HTTP with the purpose of RPC, implements a new parameter passing sub-format, hence *SOAP XRPC != SOAP RPC.*

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope
    xmlns:xrpc="http://monetdb.cwi.nl/XQuery"
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://monetdb.cwi.nl/XQuery http://monetdb.cwi.nl/XQuery/X
  <env:Body>
    <xrpc:request module="filmdb"
                  method="filmsByActor"
                  arity="1"
                  location="http://example.org/film.xq"
                  iter-cnt="1"
                  updCall="false">
      <xrpc:call>
        <xrpc:sequence>
          <xrpc:atomic-value xsi:type="xs:string">Sean Connery</xrpc:atomic-value>
        </xrpc:sequence>
      </xrpc:call>
    </xrpc:request>
  </env:Body>
</env:Envelope>
```

- *Atomic values* are represented with `atomic-value`, and are annotated with their (simple) XML Schema Type in the `xsi:type` attribute. Thus, the heterogeneously typed sequence consisting on a string `"abc"` and a double `3.1` would become:

```
<xrpc:sequence>
  <xrpc:atomic-value xsi:type="xs:string">abc</xrpc:atomic-value>
  <xrpc:atomic-value xsi:type="xs:double">3.1</xrpc:atomic-value>
</xrpc:sequence>
```

- *XML nodes* are passed by value, enclosed by an `element` element:

```
<xrpc:sequence>
  <xrpc:element>
    <filmName>The Rock</filmName>
  </xrpc:element>
  <xrpc:element>
    <filmName>Goldfinger</filmName>
  </xrpc:element>
</xrpc:sequence>
```

  Similarly, the XML Schema XRPC.xsd defines enclosing elements for document, attribute, text, processing instruction, and comment nodes. Document nodes are represented in the SOAP message as a `document` element that contains the serialized document root. Text, comment and processing instruction nodes are serialized textually inside the respective elements `text`, `comment` and `processing-instruction`. Attribute nodes are serialized *inside* the `attribute` element: `<xrpc:attribute x="y">`.
- *User-defined types:* XRPC fully supports the XQuery Data Model, a requirement for making it an orthogonal language feature. This implies XRPC also supports pass-

ing of values of user-defined XML Schema types, including the ability to validate SOAP messages. XQuery already allows importing XML Schema files that contain such definitions. Values of user-defined types are enclosed in SOAP messages by `element` elements, with a `xsi:type` attribute annotating their type. The XQuery system implementing XRPC should include a `xmlns` namespace definition as well as a `xsi:schemaLocation` declaration inside the `Envelope` element when values of such imported element types occur in the SOAP message.

- *Multi-parameter functions:* for functions with more than one parameters, the value of each parameter is enclosed in a separat `sequence` element. For example, to call the function

```
declare function add ($v1 as xs:integer, $v2 as xs:integer) as xs:integer
```

  with the parameters `10` and `20`, the values are serialised as the following:

```
<xrpc:sequence>
  <xrpc:atomic-value xsi:type="xs:integer">10</xrpc:atomic-value>
<xrpc:sequence>
<xrpc:sequence>
  <xrpc:atomic-value xsi:type="xs:integer">20</xrpc:atomic-value>
<xrpc:sequence>
```

- *Loop-lifting:* on of the main feature of the SOAP XRPC protocol is the support for loop-lifting, that is, *all* iterations in a `for`-loop that containing the applications of the *same* function (but usually with different parameter values) on the *same* remote peer, are serialised in *one* XRPC request message. The parameter values of each iteration is enclosed in a separate `call` element. The execution results of all those iterations will also be serialised into *one* XRPC response message. For example, the example query (Q2) above contains two iterations that call the same function on the same remote peer. For this query, the following request message (only the main part is shown) will be generated:

```
<xrpc:request module="filmdb"
              method="filmsByActor"
              arity="1"
              location="http://example.org/film.xq"
              iter-cnt="2"
              updCall="false">
  <xrpc:call>
    <xrpc:sequence>
      <xrpc:atomic-value xsi:type="xs:string">Julie Andrews</xrpc:atomic-value>
    </xrpc:sequence>
  </xrpc:call>
  <xrpc:call>
    <xrpc:sequence>
      <xrpc:atomic-value xsi:type="xs:string">Sean Connery</xrpc:atomic-value>
    </xrpc:sequence>
  </xrpc:call>
</xrpc:request>
```

XRPC Response Messages follow the same principles, e.g.:

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope
  xmlns:xrpc="http://monetdb.cwi.nl/XQuery"
  xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://monetdb.cwi.nl/XQuery
                      http://monetdb.cwi.nl/XQuery/XRPC.xsd">
  <env:Body>
    <xrpc:response module="filmdb" method="filmsByActor">
      <xrpc:sequence>
        <xrpc:element><filmName>The Rock</filmName></xrpc:element>
        <xrpc:element><filmName>Goldfinger</filmName></xrpc:element>
      </xrpc:sequence>
    </xrpc:response>
  </env:Body>
</env:Envelope>
```

Inside the body is now a XRPC:RESPONSE element that contains the result SEQUENCE of the remote function call.

XRPC Error Message. Whenever an XRPC server discovers an error during the processing of an XRPC request, it immediately stops execution and sends back an XRPC error message, using the format of the SOAP Fault message (see SOAP Version 1.2 Part 0: Primer and Part 1: Messaging Framework). For example, the following SOAP Fault message indicates that a required module could not be loaded:

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Receiver</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en">could not load module!</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

4.8 Frequently Asked Questions

4.8.1 General

Q: How powerful is MonetDB/XQuery? Which features of XQuery does it support? A: The Language page lists features and limitations of the current MonetDB/XQuery implementation.

Q: I want to learn more about the research behind MonetDB/XQuery. A: MonetDB/XQuery is powered by the research project <em>Pathfinder</em>. Information about this project is available on the Pathfinder Project site, along with a publication list and a list of persons and institutions involved.

4.8.2 Installation

Q: How do I install MonetDB/XQuery on my system? A: The Installation Guide section gives download and installation instructions for various platforms with various degrees of detail.

4.8.3 XQuery

Q: I get the message <code>!ERROR mapi_reconnect: connect: Connection refused</code>, when trying to execute a statement using mclient. A: Before you can start executing XQuery statements, you need to start the MonetDB Server. Details on how to start the server can be found here. If you have a MonetDB Server running and `mclient` cannot connect, you might (in case of a Linux/Unix environment) have forgotten to load the pathfinder module: module(pathfinder);.

Q: What XQuery Language is supported? A: The XQuery language is very powerful, but it is sometimes difficult to grasp. If you have problems formulating your queries, please make sure that the functions you use are supported by MonetDB/XQuery. A detailed list of the supported functions please check this page.

Q: I want to learn XQuery

A: XQuery is a standard set by the W3C (World Wide Web Consortium). It is written down as a specification. This specification is not very explanatory. If you would like an introduction to XQuery, there are several websites available You can find them using Google searching for "XQuery tutorial".

Q: I get the type error type error: no variant of function op:plus accepts the given argument type(s): double*; integer, *when trying to execute the following query: doc("foo")/val + 1* MonetDB/XQuery uses static typing which prohibits that operations on atomic values (like value comparisons or arithmetics) work on sequence arguments. To avoid this compile time error, the functions fn:zero-or-one() or fn:exactly-one() can be used. Note that these functions will cause a runtime error if `doc("foo")/val` evaluates to more than one item.

4.8.4 Miscellaneous

Q: How do I add/delete a document to my collection? A: Check the Document collection page, it will explain how this is done.

Q: I cannot find an answer to my problem. Where can I get help?

A: If this FAQ does not help you out (and the problem is XQuery related), you still have a few options left to get assistance:

- The MonetDB Server pages.
- The MonetDB/XQuery mailing list
- The buglist maintained at Sourceforge.

4.9 Design Considerations

Redesign of the MonetDB software stack was driven by the need to reduce the effort to extend the system into novel directions and to reduce the Total Execution Cost (TEC). The TEC is what an end-user or application program will notice. The TEC is composed on several cost factors:

- A) API message handling
- P) Parsing and semantic analysis
- O) Optimization and plan generation
- D) Data access to the persistent store
- E) Execution of the query terms
- R) Result delivery to the application

Choosing an architecture for processing database operations pre-supposes an intuition on how the cost will be distributed. In an OLTP setting you expect most of the cost to be in (P,O), while in OLAP it will be (D,E,R). In a distributed setting the components (O,D,E) are dominant. Web-applications would focus on (A,E,R).

Such a simple characterization ignores the wide-spread differences that can be experienced at each level. To illustrate, in D) and R) it makes a big difference whether the data is already in the cache or still on disk. With E) it makes a big difference whether you are comparing two integers, evaluation of a mathematical function, e.g., Gaussian, or a regular expression evaluation on a string. As a result, intense optimization in one area may become completely invisible due to being overshadowed by other cost factors.

The Version 5 infrastructure is designed to ease addressing each of these cost factors in a well-defined way, while retaining the flexibility to combine the components needed for a particular situation. It results in an architecture where you assemble the components for a particular application domain and hardware platform.

The primary interface to the database kernel is still based on the exchange of text in the form of queries and simply formatted results. This interface is designed for ease of interpretation, versatility and is flexible to accommodate system debugging and application tool development. Although a textual interface potentially leads to a performance degradation, our experience with earlier system versions showed that the overhead can be kept within acceptable bounds. Moreover, a textual interface reduces the programming effort otherwise needed to develop test and application programs. The XML trend as the language for tool interaction supports our decision.

4.10 Architecture Overview

The architecture is built around a few independent components: the MonetDB server, the MonetDB guardian, and the client application. The MonetDB server is the heart of the system, it manages a single physical database on one machine for all (concurrent) applications. The guardian program works along side a single server, keeping an eye on its behavior. If the server accidently crashes, it is this program that will attempt an automatic restart. The server and the guardian are managed with the MONETDB script, introduced in Section 1.8.1 [Start and Stop], page 21.

The top layer consists of applications written in your favorite language. They provide both specific functionality for a particular product, e.g., Proximity, and generic functionality, e.g., the Aquabrowser or Dbvisualizer. The applications communicate with the server using de-facto standard interface packaged, i.e., JDBC, ODBC, Perl, PHP, etc.

The middle layer consists of query language processors such as SQL and XQuery. The former supports the core functionality of SQL'99 and extends into SQL'03. The latter is based on the W3C standard and includes the XUpdate functionality. The query language processors each manage their own private catalog structure. Software bridges, e.g., import/export routines, are used to share data between language paradigms.

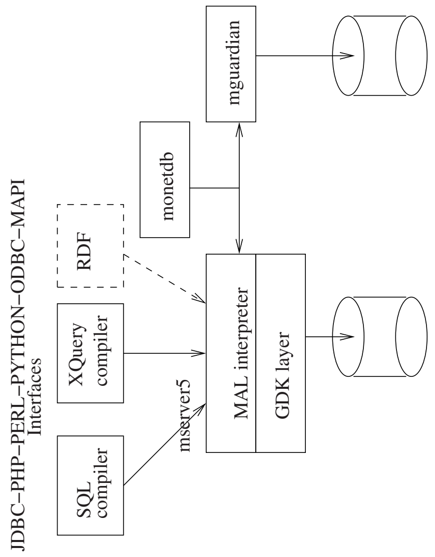

Figure 2.1

4.11 MonetDB Assembly Language (MAL)

The target language for a query compiler is the MonetDB Assembly Language (MAL). It was designed to ease code generation and fast interpretation by the server. The compiler produces algebraic query plans, which are turned into physical execution plans by the MAL optimizers.

The output of a compiler is either an ASCII representation of the MAL program or the compiler is tightly coupled with the server to save parsing and communication overhead.

A snippet of the MAL code produced by the SQL compiler for the query SELECT COUNT(*) FROM TABLES is shown below. It illustrates a sequences of relational operations against a table column and producing a partial result.

```
...
    _22:bat[:oid,:oid]  := sql.bind_dbat("tmp","_tables",0);
    _23 := bat.reverse(_22);
    _24 := algebra.kdifference(_20,_23);
    _25 := algebra.markT(_24,0:oid);
    _26 := bat.reverse(_25);
    _27 := algebra.join(_26,_20);
    _28 := bat.setWriteMode(_19);
    bat.append(_28,_27,true);
...
```

MAL supports the full breath of computational paradigms deployed in a database setting. It is language framework where the execution semantics is determined by the code transformations and the final engine choosen.

The design and implementation of MAL takes the functionality offered previously a significant step further. To name a few:

- All instructions are strongly typed before being executed.
- It supports polymorphic functions. They act as templates that produce strongly typed instantiations when needed.
- Function style expressions where each assignment instruction can receive multiple target results; it forms a point in the dataflow graph.
- It supports co-routines (Factories) to build streaming applications.
- Properties are associated with the program code for ease of optimization and scheduling.
- It can be readily extended with user defined types and function modules.

4.12 Execution Engine

The execution engine comes in several flavors. The default is a simple, sequential MAL interpreter. For each MAL function call it creates a stack frame, which is initialized with all constants found in the function body. During interpretation the garbage collector ensures freeing of space consumptive tables (BATs) and strings. Furthermore, all temporary structures are garbage collected before the funtion returns the result.

This simple approach leads to an accumulation of temporary variables. They can be freed earlier in the process using an explicit garbage collection command, but the general intend is to leave such decisions to an optimizer or scheduler.

The execution engine is only called when all MAL instructions can be resolved against the available libraries. Most modules are loaded when the server starts using a bootstrap script MAL_INIT.MX Failure to find the startup-file terminates the session. It most likely points to an error in the MonetDB configuration file.

During the boot phase, the global symbol table is initialized with MAL function and factory definitions, and loading the pre-compiled commands and patterns. The libraries are dynamically loaded by default. Expect tens of modules and hundreds of operations to become readily available.

Modules can not be dropped without restarting the server. The rational behind this design decision is that a dynamic load/drop feature is often hardly used and severely complicates the code base. In particular, upon each access to the global symbol table we have to be prepared that concurrent threads may be actively changing its structure. Especially, dropping modules may cause severe problems by not being able to detect all references kept around. This danger required all accesses to global information to be packaged in a critical section, which is known to be a severe performance hindrance.

4.13 Session Scenarios

In MonetDB multiple languages, optimizers, and execution engines can be combined at run time to satisfy a wide user-community. Such an assemblage of components is called a *scenario* and consists of a *reader*, *parser*, *optimizer*, *tactic scheduler* and *engine*. These hooks allow for both linked-in and external components.

The languages supported are SQL, XQuery, and the Monet Assembly Language (MAL). The default scenario handles MAL instructions, which is used to illustrate the behavior of the scenario steps.

The MAL reader component handles interaction with a front-end to obtain a string for subsequent compilation and execution. The reader uses the common stream package to read data in large chunks, if possible. In interactive mode the lines are processed one at a time.

The MAL parser component turns the string into an internal representation of the MAL program. During this phase semantic checks are performed, such that we end up with a type correct program.

The code block is subsequently sent to an MAL optimizer. In the default case the program is left untouched. For other languages, the optimizer deploys language specific code transformations, e.g., foreign-key optimizations in joins and remote query execution. All optimization information is statically derived from the code blocks and possible catalogues maintained for the query language at hand. Optimizers leave advice and their findings in properties in the symbol table, see Section 5.9 [Property Management], page 109.

Once the program has thus been refined, the MAL scheduler prepares for execution using tactical optimizations. For example, it may parallelize the code, generate an ad-hoc user-defined function, or prepare for efficient replication management. In the default case, the program is handed over to the MAL interpreter without any further modification.

The final stage is to choose an execution paradigm, i.e. interpretative (default), compilation of an ad-hoc user defined function, dataflow driven interpretation, or vectorized pipe-line execution by a dedicated engine.

A failure encountered in any of the steps terminates the scenario cycle. It returns to the user for a new command.

4.14 Scenario management

Scenarios are captured in modules; they can be dynamically loaded and remain active until the system is brought to a halt. The first time a scenario xyz is used, the system looks for a scenario initialization routine xyzinitSystem() and executes it. It is typically used to prepare the server for language specific interactions. Thereafter its components are set to those required by the scenario and the client initialization takes place.

When the last user interested in a particular scenario leaves the scene, we activate its finalization routine calling xyzexitSystem(). It typically perform cleanup, backup and monitoring functions.

A scenario is interpreted in a strictly linear fashion, i.e. performing a symbolic optimization before scheduling decisions are taken. The routines associated with each state in the scenario may patch the code so as to assure that subsequent execution can use a different scenario, e.g., to handle dynamic code fragments.

The building blocks of scenarios are routines obeying a strict name signature. They require exclusive access to the client record. Any specific information should be accessible from there, e.g., access to a scenario specific state descriptor. The client scenario initialization and finalization brackets are xyzinitClient() and xyzexitClient().

The xyzparser(Client c) contains the parser for language XYZ and should fill the MAL program block associated with the client record. The latter may have been initialized with variables. Each language parser may require a catalog with information on the translation of language specific datastructures into their BAT equivalent.

The xyzoptimizer(Client c) contains language specific optimizations using the MAL intermediate code as a starting point.

The xyztactics(Client c) synchronizes the program execution with the state of the machine, e.g., claiming resources, the history of the client or alignment of the request with concurrent actions (e.g., transaction coordination).

The xyzengine(Client c) contains the applicable back-end engine. The default is the MAL interpreter, which provides good balance between speed and ability to analysis its behavior.

5 MonetDB Assembly Language (MAL)

The primary textual interface to the Monetdb kernel is a simple, assembly-like language, called MAL. The language reflects the virtual machine architecture around the kernel libraries and has been designed for speed of parsing, ease of analysis, and ease of target compilation by query compilers. The language is not meant as a primary programming language, or scripting language. Such use is even discouraged.

Furthermore, a MAL program is considered a specification of intended computation and data flow behavior. It should be understood that its actual evaluation depends on the execution paradigm choosen in the scenario. The program blocks can both be interpreted as ordered sequences of assembler instructions, or as a representation of a data-flow graph that should be resolved in a dataflow driven manner. The language syntax uses a functional style definition of actions and mark those that affect the flow explicitly. Flow of control keywords identify a point to chance the interpretation paradigm and denote a synchronization point.

MAL is the target language for query compilers, such as the SQL and XQuery front-ends. Even simple SQL queries generate a long sequence of MAL instructions. They represent both the administrative actions to ensure binding and transaction control, the flow dependencies to produce the query result, and the steps needed to prepare the result set for delivery to the front-end.

Only when the algebraic structure is too limited (e.g. updates), or the database back-end lacks feasible builtin bulk operators, one has to rely on more detailed flow of control primitives. But even in that case, the basic blocks to be processed by a MAL back-end are considered large, e.g. tens of simple bulk assignment instructions.

The remainder of this chapter provide a concise overview of the language features and illustrative examples.

5.1 MAL Literals

Literals in MAL follow the lexical conventions of the programming language C. A default type is attached, e.g. the literal 1 is typed as an INT value. Likewise, the literal 3.14 is typed FLT rather than DBL.

A literal can be coerced to another type by tagging it with a type classifier, provided a coercion operation is defined. For example, 1:LNG marks the literal as of type LNG. and `"1999-12-10"`:DATE creates a DATE literal.

MonetDB comes with the hardwired types BIT, BTE, CHR, WRD, SHT, INT, LNG, OID, FLT, DBL, STR and BAT, the bat identifier. The kernel code has been optimized to deal with these types efficiently, i.e. without unnecessary function call overheads. In addition, the system supports temporal types DATE, DAYTIME, TIME, TIMESTAMP, TIMEZONE, extensions to deal with IPv4 addresses and URLs using INET, URL, and several types to interact more closely with the kernel LOCK, SEMPHORE. This list can be extended with user defined types.

5.2 MAL Variables

Variables are denoted by identifers and implicitly defined upon first use. They take on a type through a type classifier or inherit it from the context in which they are first used, see ⟨undefined⟩ [MAL Type System], page ⟨undefined⟩.

Variables are organized into two classes, starting with and without an underscore. The latter are reserved as MAL parser tempoaries, whose name aligns with an entry in the symbol table. In general they can not be used in MAL programs, but they may become visible in MAL program listings or during debugging.

5.3 Instructions

A MAL instruction has purposely a simple format. It is syntactically represented by an assignment, where an expression (function call) delivers results to multiple target variables. The assignment patterns recognized are illustrated below.

```
(t1,..,t32) := module.fcn(a1,..,a32);
t1 := module.fcn(a1,..,a32);
t1 := v1 operator v2;
t1 := literal;
(t1,..,tn) := (a1,..,an);
```

Operators are grouped into user defined modules. Ommission of the module name is interpreter as the USER module.

Simple binary arithmetic operations are merely provided as a short-hand, e.g. the expression T:=2+2 is converted directly into T:= CALC.+(2,2).

Target variables are optional. The compiler introduces temporary variables to hold the result of the expression upon need. They won't show up when you list the MAL program unless it is used elsewhere.

For parsing simplicity, each instruction fits on a single line. Comments start with a sharp '#' and continues to the end of the line. They are retained in the internal code representation to ease debugging of compiler generated MAL programs.

The data structure to represent a MAL block is kept simple. It contains a sequence of MAL statements and a symbol table. The MAL instruction record is a code byte string overlaid with the instruction pattern, which contains references into the symbol tables and administrative data for the interpreter.

This method leads to a large allocated block, which can be easily freed. Variable- and statement- block together describe the static part of a MAL procedure. It carries enough information to produce a listing and to aid symbolic debugging.

5.4 MAL Flow-of-control

The flow of control within a MAL program block can be changed by tagging a statement with either RETURN, YIELD, BARRIER, CATCH, LEAVE, REDO, or EXIT.

The flow modifiers RETURN and YIELD mark the end of a call and return one or more results to the calling environment. The RETURN and YIELD are followed by a target list or an assignment, which is executed first.

The BARRIER (CATCH) and EXIT pair mark a guarded statement block. They may be nested to form a proper hierarchy identified by their primary target variable, also called the control variable.

The LEAVE and REDO are conditional flow modifiers. The control variable is used after the assignment statement has been evaluated to decide on the flow-of-control action to be taken. Built-in controls exists for booleans and numeric values. The barrier block is opened when the control variable holds true, when its numeric value >= 0, or when it is a non-empty string. The NIL value blocks entry in all cases.

Once inside the barrier you have an option to prematurely LEAVE it at the exit statement or to REDO interpretation just after the corresponding barrier statement. Much like 'break' and 'continue' statements in the programming language C. The action is taken when the condition is met.

The EXIT marks the exit for a block. Its optional assignment can be used to re-initialize the barrier control variables or wrap-up any related administration.

The barrier blocks can be properly nested to form a hierarchy of basic blocks. The control flow within and between blocks is simple enough to deal with during an optimizer stage. The REDO and LEAVE statements mark the partial end of a block. Statements within these blocks can be re-arranged according to the data-flow dependencies. The order of partial blocks can not be changed that easily. It depends on the mutual exclusion of the data flows within each partial block.

Common guarded blocks in imperative languages are the for-loop and if-then-else constructs. They can be simulated as follows.

Consider the statement FOR(I=1;I<10;I++) PRINT(I). The (optimized) MAL block to implement this becomes:

```
        i:= 1;
    barrier B:= i<10;
        io.print(i);
        i:= i+1;
    redo B:= i<10;
    exit B;
```

Translation of the statement IF(I<1) PRINT("OK"); ELSE PRINT("WRONG"); becomes:

```
        i:=1;
    barrier ifpart:= i<1;
        io.print("ok");
    exit ifpart;
    barrier elsepart:= i>=1;
        io.print("wrong");
    exit elsepart;
```

Note that both guarded blocks can be interchanged without affecting the outcome. Moreover, neither block would have been entered if the variable happens to be assigned NIL.

The primitives are sufficient to model a wide variety of iterators, whose pattern look like:

```
    barrier i:= M.newIterator(T);
        elm:= M.getElement(T,i);
        ...
        leave i:= M.noMoreElements(T);
        ...
        redo i:= M.hasMoreElements(T);
    exit i:= M.exitIterator(T);
```

The semantics obeyed by the iterator implementations is as follows. The redo expression updates the target variable *i* and control proceeds at the first statement after the barrier when the barrier is opened by *i*. If the barrier could not be re-opened, execution proceeds with the first statement after the redo. Likewise, the leave control statement skips to the exit when the control variable *i* shows a closed barrier block. Otherwise, it continues with the next instruction. Note, in both failed cases the control variable is possibly changed.

A recurring situation is to iterate over the elements in a BAT. This is supported by an iterator implementation for BATs as follows:

```
    barrier (idx,hd,tl):= bat.newIterator(B);
        ...
        redo (idx,hd,tl):= bat.hasMoreElements(B);
    exit (ids,hd,tl);
```

Where idx is an integer to denote the row in the BAT, hd and tl denote values of the current element.

5.5 Functions

MAL comes with a standard functional abstraction scheme. Functions are represented by MAL instruction lists, enclosed by a FUNCTION signature and END statement. The FUNCTION signature lists the arguments and their types. The END statement marks the end of this sequence. Its argument is the function name.

An illustrative example is:

```
    function user.helloWorld(msg:str):str;
        io.print(msg);
        msg:= "done";
        return msg;
    end user.hellowWorld;
```

The module name designates the collection to which this function belongs. All user defined functions are assembled in the module USER by default. It may be dropped without affecting the intended semantics.

The functional abstraction scheme comes with several variations: COMMANDS, PATTERNS, and FACTORIES. They are discussed shortly.

5.5.1 Polymorphic Functions

Polymorphic functions are characterised by type variables denoted by :ANY and an optional index. Each time a polymorphic MAL function is called, the symbol table is first inspected for the matching strongly typed version. If it does not exists, a copy of the MAL program is generated, whereafter the type variables are replaced with their concrete types. The new MAL program is immediately type checked and, if no errors occured, added to the symbol table.

The generic type variable :ANY designates an unknown type, which may be filled at type resolution time. Unlike indexed polymorphic type arguments, :ANY type arguments match possibly with different concrete types.

An example of a parameterised function is shown below:

```
function user.helloWorld(msg:any_1):any_1;
    io.print(msg);
    return user.helloWorld;
end helloWorld;
```

The type variables ensure that the return type equals the argument type. Type variables can be used at any place where a type name is permitted. Beware that polymorphic typed variables are propagated throughout the function body. This may invalidate type resolutions decisions taken earlier (See ⟨undefined⟩ [MAL Type System], page ⟨undefined⟩).

This version of HELLOWORLD can also be used for other arguments types, i.e. BIT,SHT,LNG,FLT,DBL,.... For example, calling HELLOWORLD(3.14:FLT) echoes a float value.

5.5.2 C functions

The MAL function body can also be implemented with a C-function. They are introduced to the MAL type checker by providing their signature and an ADDRESS qualifier for linkage.

We distinguish both COMMAND and PATTERN C-function blocks. They differ in the information accessible at run time. The COMMAND variant calls the underlying C-function, passing pointers to the arguments on the MAL runtime stack. The PATTERN command is passed pointers to the MAL definition block, the runtime stack, and the instruction itself. It can be used to analyse the types of the arguments directly.

For example, the definitions below link the kernel routine BKCINSERT_BUN with the function BAT.INSERT(). It does not fully specify the result type. The IO.PRINT() pattern applies to any BAT argument list, provided they match on the head column type. Such a polymorphic type list may only be used in the context of a pattern.

```
command bat.insert(b:bat[:any_1,:any_2], ht:any_1, tt:any_2)
:bat[:any_1,:any_2]
address BKCinsert_bun;

pattern io.print(b1:bat[:any_1,:any]...):int
address IOtable;
```

5.6 Factories

A convenient programming construct is the co-routine, which is specified as an ordinary function, but maintains its own state between calls, and permits re-entry other than by the first statement.

The random generator example is used to illustrate its definition and use.

```
factory random(seed:int,limit:int):int;
    rnd:=seed;
    lim:= limit;
barrier lim;
    leave lim:= lim-1;
    rnd:= rnd*125;
    yield rnd:= rnd % 32676;
    redo lim;
exit lim;
```

```
end random;
```

The first time this factory is called, a *plant* is created in the local system to handle the requests. The plant contains the stack frame and synchronizes access.

In this case it initializes the generator. The random number is generated and YIELD as a result of the call. The factory plant is then put to sleep. The second call received by the factory wakes it up at the point where it went to sleep. In this case it will find a REDO statement and produces the next random number. Note that also in this case a seed and limit value are expected, but they are ignored in the body. This factory can be called upon to generate at most 'limit' random numbers using the 'seed' to initialize the generator. Thereafter it is being removed, i.e. reset to the original state.

A cooperative group of factories can be readily constructed. For example, assume we would like the random factories to respond to both RANDOM(SEED,LIMIT) and RANDOM(). This can be defined as follows:

```
factory random(seed:int,limit:int):int;
    rnd:=seed;
    lim:= limit;
barrier lim;
    leave lim:= lim-1;
    rnd:= rnd*125;
    yield rnd:= rnd % 32676;
    redo lim;
exit lim;
end random;

factory random():int;
barrier forever:=true;
    yield random(0,0);
    redo forever;
exit forever;
end random;
```

5.6.1 Factory Ownership

For simple cases, e.g. implementation of a random function, it suffices to ensure that the state is secured between calls. But, in a database context there are multiple clients active. This means we have to be more precise on the relationship between a co-routine and the client for which it works.

The co-routine concept researched in Monet 5 is the notion of a 'factory', which consists of 'factory plants' at possibly different locations and with different policies to handle requests. Factory management is limited to its owner, which is derived from the module in which it is placed. By default Admin is the owner of all modules.

The factory produces elements for multiple clients. Sharing the factory state or even remote processing is up to the factory owner. They are set through properties for the factory plant.

The default policy is to instantiate one shared plant for each factory. If necessary, the factory can keep track of a client list to differentiate the states. A possible implementation would be:

```
factory random(seed:int,clientid:int):int;
    clt:= bat.new(:int,:int);
    bat.insert(clt,clientid,seed);
barrier always:=true;
    rnd:= algebra.find(clt,clientid);
catch   rnd; #failed to find client
    bat.insert(clt,clientid,seed);
    rnd:= algebra.find(clt,clientid);
exit    rnd;
    rnd:= rnd * 125;
    rnd:= rnd % 32676;
    algebra.replace(clt,clientid,rnd);
    yield rnd;
    redo always;
exit always;
end random;
```

The operators to built client aware factories are, FACTORY.GETCALLER(), which returns a client index, FACTORY.GETMODULE() and FACTORY.GETFUNCTION(), which returns the identity of scope enclosed.

To illustrate, the client specific random generator can be shielded using the factory:

```
factory random(seed:int):int;
barrier always:=true;
    clientid:= factory.getCaller();
    yield user.random(seed, clientid);
    redo always;
exit always;
end random;
```

5.6.2 Complex Factories

The factory scheme can be used to model a volcano-style query processor. Each node in the query tree is an iterator that calls upon the operands to produce a chunk, which are combined into a new chunk for consumption of the parent. The prototypical join(R,S) query illustrates it. The plan does not test for all boundary conditions, it merely implements a nested loop. The end of a sequence is identified by a NIL chunk.

```
factory query();
    Left:= sql.bind("relationA");
    Right:= sql.bind("relationB");
    rc:= sql.joinStep(Left,Right);
barrier rc!= nil;
    io.print(rc);
    rc:= sql.joinStep(Left,Right);
    redo rc!= nil;
exit rc;
```

```
end query;

#nested loop join
factory sql.joinStep(Left:bat[:any,:any],Right:bat[:any,:any]):bat[:any,:any];
    lc:= bat.chunkStep(Left);
barrier outer:= lc != nil;
    rc:= bat.chunkStep(Right);
    barrier inner:= rc != nil;
        chunk:= algebra.join(lc,rc);
        yield chunk;
        rc:= bat.chunkStep(Right);
        redo inner:= rc != nil;
    exit inner;
    lc:= bat.chunkStep(Left);
    redo outer:= lc != nil;
exit outer;
    # we have seen everything
    return nil;
end joinStep;

#factory for left branch
factory chunkStepL(L:bat[:any,:any]):bat[:any,:any];
    i:= 0;
    j:= 20;
    cnt:= algebra.count(L);
barrier outer:= j<cnt;
    chunk:= algebra.slice(L,i,j);
    i:= j;
    j:= i+ 20;
    yield chunk;
    redo loop:= j<cnt;
exit outer;
    # send last portion
    chunk:= algebra.slice(L,i,cnt);
    yielD chunk;
    return nil;
end chunkStep;

#factory for right leg
factory chunkStepR(L:bat[:any,:any]):bat[:any,:any];
```

So far we haven't re-used the pattern that both legs are identical. This could be modeled by a generic chunk factory. Choosing a new factory for each query steps reduces the administrative overhead.

5.6.3 Materialized Views

An area where factories might be useful are support for materialized views, i.e. the result of a query is retained for ease of access. A simple strategy is to prepare the result once and return it on each successive call. Provided the arguments have not been changed. For example:

```
factory view1(l:int, h:int):bat[:oid,:str];
    a:bat[:oid,:int]:= bbp.bind("emp","age");
    b:bat[:oid,:str]:= bbp.bind("emp","name");
barrier always := true;
    lOld := l;
    hOld := h;
    c := algebra.select(a,l,h);
    d := algebra.semijoin(b,c);
barrier available := true;
    yield d;
    leave available := calc.!=(lOld,l);
    leave available := calc.!=(hOld,h);
    redo available := true;
exit available;
    redo always;
exit always;
end view1;
```

The code should be extended to also check validity of the BATs. It requires a check against the last transaction identifier known.

The Factory concept is still rather experimental and many questions should be considered, e.g. What is the lifetime of a factory? Does it persists after all clients has disappeared? What additional control do you need? Can you throw an exception to a Factory?

5.7 Type Resolution

Given the interpretative nature of many of the MAL instructions, when and where type resolution takes place is a critical design issue. Performing it too late, i.e. at each instruction call, leads to performance problems if we derive the same information over and over again. However, many built-in operators have polymorphic typed signatures, so we cannot escape it altogether.

Consider the small illustrative MAL program:

```
function sample(nme:str, val:any_1):bit;
   c := 2 * 3;
   b := bbp.bind(nme);  #find a BAT
   h := algebra.select(b,val,val);
   t := aggr.count(h);
   x := io.print(t);
   y := io.print(val);
end sample;
```

The function definition is polymorphic typed on the 2nd argument, it becomes a concrete type upon invocation. The system could attempt a type check, but quickly runs into

assumptions that generally do not hold. The first assignment can be type checked during parsing and a symbolic optimizer could even evaluate the expression once. Looking up a BAT in the buffer pool leads to an element :BAT[*ht*,*tt*] where *ht* and *tt* are runtime dependent types, which means that the selection operation can not be type-checked immediately. It is an example of an embedded polypmorphic statement, which requires intervention of the user/optimizer to make the type explicit before the type resolver becomes active. The operation COUNT can be checked, if it is given a BAT argument. This assumes that we can infer that 'h' is indeed a BAT, which requires assurance that ALGEBRA.SELECT produces one. However, there are no rules to avoid addition of new operators, or to differentiate among different implementations based on the argument types. Since PRINT(T) contains an undetermined typed argument we should postpone typechecking as well. The last print statement can be checked upon function invocation.

Life becomes really complex if the body contains a loop with variable types. For then we also have to keep track of the original state of the function. Or alternatively, type checking should consider the runtime stack rather than the function definition itself.

These examples give little room to achieve our prime objective, i.e. a fast and early type resolution scheme. Any non-polymorphic function can be type checked and marked type-safe upon completion. Type checking polymorphic functions are post-poned until a concrete type instance is known. It leads to a clone, which can be type checked and is entered into the symbol table.

5.7.1 User Defined Types

MonetDB supports an extensible type system to accomodate a wide spectrum of database kernels and application needs. The type administration keeps track of their properties and provides access to the underlying implementations.

MAL recognizes the definition of a new type by replacing the MODULE keyword with ATOM. Atoms definitions require special care, because their definition and properties should be communicated with the kernel library. The commands defined in an ATOM block are screened as of interest to the library.

MonetDB comes with the hardwired types BIT, CHR, SHT, INT, LNG, OID, FLT, DBL, STR and BAT, the representation of a bat identifier. The kernel code has been optimized to deal with these types efficiently, i.e. without unnecessary function call overheads.

A small collection of user-defined ATOM types is shipped with the sysem. They implement types considered essential for end-user applications, such as COLOR, DATE, DAYTIME, TIME, TIMESTAMP, TIMEZONE, BLOB, and INET, URL. They are implemented using the type extension mechanism described below. As such, they provide examples for future extensions. A concrete example is the 'blob' datatype in the MonetDB atom module library(see ../modules/atoms/blob.mx)

5.7.2 Defining your own types

For the courageous at heart, you may enter the difficult world of extending the kernel library. The easiest way is to derive the atom modules from one shipped in the source distributed. More involved atomary types require a study of the documentation associated with the atom structures (gdk_atoms), because you have to develop a handful routines complying with the signatures required in the kernel library. They are registered upon loading the ATOM module.

5.8 Boxed Variables

Clients sessions often come with a global scope of variable settings. Access to these global variables should be easy, but they should also provide protection against concurrent update when the client wishes to perform parallel processing. Likewise, databases, query languages, etc. may define constants and variables accessible, e.g., relational schemas, to a selected user group.

The approach taken is to rely on persistent object spaces as pioniered in Lynda and -later- JavaSpaces. They are called boxes in MonetDB and act as managed containers for persistent variables.

Before a client program can interact with a box, it should open it, passing qualifying authorization information and parameters to instruct the box-manager of the intended use. A built-in box is implicitly opened when you request for its service.

At the end of a session, the box should be closed. Some box-managers may implement a lease-scheme to automatically close interaction with a client when the lease runs out. Likewise, the box can be notified when the last reference to a leased object ceases to exist.

A box can be extended with a new object using the function deposit(name) with name a local variable. The default implementation silently accepts any new definition of the box. If the variable was known already in the box, its value is overwritten.

A local copy of an object can be obtained using the pattern 'take(name,[param])', where name denotes the variable of interest. The type of the receiving variable should match the one known for the object. Whether an actual copy is produced or a reference to a shared object is returned is defined by the box manager.

The object is given back to the box manager calling 'release(name)'. It may update the content of the repository accordingly, release locks, and move the value to persistent store. Whatever the semantics of the box requires. [The default implementation is a no-op]

Finally, the object manager can be requested to 'discard(name)' a variable completely. The default implementation is to reclaim the space in the box.

Concurrency control, replication services, as well as access to remote stores may be delegated to a box manager. Depending on the intended semantics, the box manager may keep track of the clients holding links to this members, provide a traditional 2-phase locking scheme, optimistic control, or check-out/check-in scheme. In all cases, these management issues are transparant to the main thread (=client) of control, which operates on a temporary snapshot. For the time being we realize the managers as critical code sections, i.e. one client is permitted access to the box space at a time.

Fo example, consider the client function:

```
function myfcn():void;
    b:bat[:oid,:int] := bbp.take("mytable");
    c:bat[:int,:str] := sql.take("person","age");
    d:= intersect(b,c);
    io.print(d);
    u:str:= client.take(user);
    io.print(u);
    client.release(user);
end function;
```

The function binds to a copy from the local persistent BAT space, much like bat-names are resolved in earlier MonetDB versions. The second statement uses an implementation of take that searches a variable of interest using two string properties. It illustrates that a box manager is free to extend/overload the predefined scheme, which is geared towards storing MAL variables.

The result bat C is temporary and disappears upon garbage collection. The variable U is looked up as the string object user.

Note that BATs B and C need be released at some point. In general this point in time does not coincide with a computational boundary like a function return. During a session, several bats may be taken out of the box, being processed, and only at the end of a session being released. In this example, it means that the reference to b and c is lost at the end of the function (due to garbarge collection) and that subsequent use requires another take() call. The box manager bbp is notified of the implicit release and can take garbage collection actions.

The box may be inspected at several times during a scenario run. The first time is when the MAL program is type-checked for the box operations. Typechecking a take() function is tricky. If the argument is a string literal, the box can be queried directly for the objects' type. If found, its type is matched against the lhs variable. This strategy fails in the situation when at runtime the object is subsequently replaced by another typed-instance in the box. We assume this not to happen and the exceptions it raises a valuable advice to reconsider the programming style.

The type indicator for the destination variable should be provided to proceed with proper type checking. It can resolve overloaded function selection.

Inspection of the Box can be encoded using an iterator at the MAL layer and relying on the functionality of the box. However, to improve introspection, we assume that all box implementations provide a few rudimentary functions, called objects(arglist) and dir(arglist). The function objects() produces a BAT with the object names, possibly limited to those identified by the arglist.

The world of boxes has not been explored deeply yet. It is envisioned that it could play a role to import/export different objects, e.g., introduce xml.take() which converts an XML document to a BAT, jpeg.take() similer for an image.

Nesting boxes is possible. It provides a simple containment scheme between boxes, but in general will interfere with the semantics of each box.

Each box has (should) have an access control list, which names the users having permission to read/write its content. The first one to create the box becomes the owner. He may grant/revoke access to the box to users on a selective basis.

5.8.1 Session Box

Aside from box associated with the modules, a session box is created dynamically on behalf of each client. Such boxes are considered private and require access by the user name (and password). At the end of a session they are closed, which means that they are saved in persistent store until the next session starts. For example:

```
function m():void;
    box.open("client_name");
    box.deposit("client_name","pi",3.417:flt);
```

```
        f:flt := box.take("client_name","pi");
        io.print(t);
        box.close("client_name");
    end function;
```

In the namespace it is placed subordinate to any space introduced by the system administrator. It will contain global client data, e.g., user, language, database, port, and any other session parameter. The boxes are all collected in the context of the database directory, i.e. the directory <dbfarm>/box

5.8.2 Garbage Collection

The key objects managed by MonetDB are the persistent BATs, which call for an efficient scheme to make them accessible for manipulation in the MAL procedures taking into account a possibly hostile parallel access.

Most kernel routines produce BATs as a result, which will be referenced from the runtime stack. They should be garbage collected as soon as deemed possible to free-up space. By default, temporary results are garbage collected before returning from a MAL function.

5.8.3 Globale Environment

The top level interaction keeps a 'box' with global variables, i.e. each MAL statement is interpreted in an already initialized stack frame. This causes the following problems: 1) how to get rid of global variables and 2) how to deal with variables that can take 'any' type. It is illustrated as follows:

```
f:= const.take("dbname");
io.print(f);
```

When executed in the context of a function, the answer will be simple [nil]. The reason is that the expecteed type is not known at compilation time. The correct definition would have been

```
f:str:= const.take("dbname");
io.print(f);
```

5.9 Property Management

Properties come in several classes, those linked with the symbol table and those linked with the runtime environment. The former are determined once upon parsing or catalog lookup. The runtime properties have two major subclasses, i.e. reflective and prescriptive. The reflective properties merely provide a fast cache to information aggregated from the target. Prescriptive properties communicate desirable states, leaving it to other system components to reach this state at the cheapest cost possible. This multifacetted world makes it difficult to come up with a concise model for dealing with properties. The approach taken here is an experimental step into this direction.

This MAL_PROPERTIES module provides a generic scheme to administer property sets and a concise API to manage them. Its design is geared towards support of MAL optimizers, which typically make multiple passes over a program to derive an alternative, better version. Such code-transformations are aided by keeping track of derived information, e.g. the expected size of a temporary result or the alignment property between BATs.

Properties capture part of the state of the system in the form of an simple term expression (NAME, OPERATOR, CONSTANT). The property model assumes a namespace built around Identifiers. The operator satisfy the syntax rules for MAL operators. Conditional operators are quite common, e.g. the triple (count, <, 1000) can be used to denote a small table.

The property bearing objects in the MAL setting are variables (symbol table entries). The direct relationship between instructions and a target variable, make it possible to keep the instruction properties in the corresponding target variable.

Variables properties The variables can be extended at any time with a property set. Properties have a scope identical to the scope of the corresponding variable. Ommision of the operator and value turns it into a boolean valued property, whose default value is TRUE.

```
b{count=1000,sorted}:= mymodule.action("table");
name{aligngroup=312} := bbp.take("person_name");
age{aligngroup=312} := bbp.take("person_age");
```

The example illustrates a mechanism to maintain alignment information. Such a property is helpful for optimizers to pick an efficient algorithm.

MAL function signatures. A function signature contains a description of the objects it is willing to accept and an indication of the expected result. The arguments can be tagged with properties that 'should be obeyed, or implied' by the actual arguments. It extends the typing scheme used during compilation/optimization. Likewise, the return values can be tagged with properties that 'at least' exist upon function return.

```
function test(b:bat[:oid,:int]{count<1000}):bat[:oid,:int]{sorted}
   #code block
end test
```

These properties are informative to optimizers. They can be enforced at runtime using the operation OPTIMIZER.ENFORCERULES() which injects calls into the program to check them. An assertion error is raised if the property does not hold. The code snippet

```
z:= user.test(b);
```

is translated into the following code block;

```
mal.assert(b,"count","<",1000);
z:= user.test(b);
mal.assert(z,"sorted");
```

How to propagate properties? Property inspection and manipulation is strongly linked with the operators of interest. Optimizers continuously inspect and update the properties, while kernel operators should not be bothered with their existence. Property propagation is strongly linked with the actual operator implementation. We examine a few recurring cases.

V:=W; Both V and W should be type compatible, otherwise the compiler will already complain.(Actually, it requires V.type()==W.type() and ~V.isaConstant()) But what happens with all others? What is the property propagation rule for the assignment? Several cases can be distinguished:

I) W has a property P, unknown to V. II) V has a propery P, unknown to W. III) V has property P, and W has property Q, P and Q are incompatible. IV) V and W have a property P, but its value disaggrees.

case I). If the variable V was not initialized, we can simply copy or share the properties. Copying might be too expensive, while shareing leads to managing the dependencies. case II) It means that V is re-assigned a value, and depending on its type and properties we may have to 'garbage collect/finalize' it first. Alternatively, it could be interpreted as a property that will hold after assignment which is not part of the right-hand side expression. case III) if P and Q are type compatible, it means an update of the P value. Otherwise, it should generates an exception. case IV) this calls for an update of V.P using the value of W.P. How this should be done is property specific.

Overall, the policy would be to 'disgard' all knowledge from V first and then copy the properties from W.

[Try 1] V:= fcn(A,B,C) and signature fcn(A:int,B:int,C:int):int The signature provides several handles to attach properties. Each formal parameter could come with a list of 'desirable/necessary' properties. Likewise, the return values have a property set. This leads to the extended signature function fcn(A:T,....,B:T): (C:T...D:T) where each Pi denotes a property set. Properties P1..Pn can be used to select the proper function variant. At its worst, several signatures of fcn() should be inspected at runtime to find one with matching properties. To enable analysis and optimization, however, it should be clear that once the function is finished, the properties Pk..Pm exist.

[Try 2] V:= fcn(A,B,C) and signature fcn(A:int,B:int,C:int):int The function is applicable when a (simple conjuntive) predicate over the properties of the actual arguments holds. A side-effect of execution of the function leads to an update of the property set associated with the actual arguments. An example:

```
function fcn (A:int,B:bat[int,int],C:int):int
        ?
```

[Try 3] Organize property management by the processor involved, e.g. a cost-based optimizer or a access control enforcer. For each optimizer we should then specify the 'symbolic' effect of execution of instructions of interest. This means 'duplication' of the instruction set.

Can you drop properties? It seems possible, but since property operations occur before actual execution there is no guarantee that they actually take place.

[case: how to handle sort(b:bat):bat as a means to propagate] [actually we need an expression language to indicate the propety set, e.g. sort(b:bat):bat which first obtains the properties of b and extends it with sorted. A nested structure emerge

Is it necessary to construct the property list intersection? Or do we need a user defined function to consolidate property lists?]

Aside, it may be valuable to collect information on e.g. the execution time of functions as a basis for future optimizations. Rather then cluttering the property section, it makes sense to explicitly update this information in a catalog.

5.10 Properties at the MAL level

Aside from routines targeted as changing the MAL blocks, it should be possible to reason about the properties within the language itself. This calls for gaining access and update. For example, the following snippet shows how properties are used in a code block.

```
B := bbp.new(int,int);
I := properties.has(B,);
J := properties.get(B,);
print(J);

properties.set(B,,2315);
barrier properties.has(B,);
exit;
```

These example illustrate that the property manipulations are executed throug patterns, which also accept a stack frame.

Sample problem with dropping properties:

```
    B := bbp.new(int,int);
barrier tst:= randomChoice()
    I := properties.drop(B,);
exit tst;
```

5.11 The cost model problem

An important issue for property management is to be able to pre-calculate a cost for a MAL block. This calls for an cost model implementation that recognizes instructions of interest, understands and can deal with the dataflow semantics, and

For example, selectivity estimations can be based on a histogram associated with a BAT. The code for this could look like

```
B:= new(int,int);
properties.add(B,);
Z:= select(B,1,100);
```

Addition of a property may trigger its evaluation, provided enough information is available (e.g. catalog). The instruction triggers the calls properties.set(B,), properties.set(B,), and properties.set(B,) once a property evaluation engine is ran against the code block. After assignment to Z, we have to propagate properties properties.update(B,).

5.12 SQL case

To study the use of properties in the complete pipeline SQL-execution we contrive a small SQL problem. The person table is sorted by name, the car table is unsorted.

```
create table person(name varchar not null,
address varchar);
create table car(name varchar,
model varchar,
milage int not null);
select distinct name, model, milage
from person, car
where car.name= person.name
  and milage>60000;
```

5.13 Implementation rules

Properties can be associated with variables, MAL blocks, and MAL instructions. The property list is initialized upon explicit request only, e.g. by the frontend parser, a box manager, or as a triggered action.

Every property should come with a function that accepts a reference to the variable and updates the property record. This function is activated either once or automatically upon each selection.

5.14 Property ADT implementation

addProperty(O,P) adds property P to the list associated with O. If O represents a compound structure, e.g. a BAT, we should indicate the component as well. For example, addProperty(O,P,Ia,...Ib) introduces a property shared by the components Ia..Ib (indicated with an integer index.

hasProperty(O,P) is a boolean function that merely checks existence hasnotProperty(O,P) is the dual operation.

setProperty(O,P,V) changes the propety value to V. It may raise a PropertyUpdateViolation exception when this can not be realized. Note, the property value itself is changed, not the object referenced.

getProperty(O,P) retrieves the current value of a property. This may involve calling a function or running a database query.

setPropertyAttribute(O,P,A) changes the behavior of the property. For example, the attribute 'freeze' will result in a call to the underlying function only once and to cache the result for the remainder of the objects life time.

5.15 Predefined properties

The MAL language uses a few defaults, recognizable as properties

unsafe	function has side effects.Default, unsafe=off
read	data can be read but not updated
append	data can be appended

6 The MAL Interpreter

The MAL interpreter always works in the context of a single user session, which provides for storage access to global variables and modules.

6.1 MAL API

The linkage between MAL interpreter and compiled C-routines is kept as simple as possible. Basically we distinguish four kinds of calling conventions: CMDcall, FCNcall, FACcall, and PATcall. The FCNcall indicates calling a MAL procedure, which leads to a recursive call to the interpreter.

CMDcall initiates calling a linked function, passing pointers to the parameters and result variable, i.e. f(ptr a0,..., ptr aN) The function returns a MAL-SUCCEED upon success and a pointer to an exception string upon failure. Failure leads to raise-ing an exception in the interpreter loop, by either looking up the relevant exception message in the module administration or construction of a standard string.

The PATcall initiates a call which contains the MAL context, i.e. f(MalBlkPtr mb, MalStkPtr stk, InstrPtr pci) The mb provides access to the code definitions. It is primarilly used by routines intended to manipulate the code base itself, such as the optimizers. The Mal stack frame pointer provides access to the values maintained. The arguments passed are offsets into the stack frame rather than pointers to the actual value.

6.2 Exception handling

Calling a built-in or user-defined routine may lead to an error or a cached status message to be dealt with in MAL. To improve error handling in MAL, an exception handling scheme based on CATCH-EXIT blocks. The CATCH statement identifies a (string-valued) variable, which carries the exception message from the originally failed routine or RAISE exception assignment. During normal processing CATCH-EXIT blocks are simply skipped. Upon receiving an exception status from a function call, we set the exception variable and skip to the first associated CATCH-EXIT block. MAL interpretation then continues until it reaches the end of the block. If no exception variable was defined, we should abandon the function alltogether searching for a catch block at a higher layer.

Exceptions raised within a linked-in function requires some care. First, the called procedure does not know anything about the MAL interpreter context. Thus, we need to return all relevant information upon leaving the linked library routine.

Second, exceptional cases can be handled deeply in the recursion, where they may also be handled, i.e. by issueing an GDKerror message. The upper layers merely receive a negative integer value to indicate occurrence of an error somewhere in the calling sequence. We then have to also look into GDKerrbuf to see if there was an error raised deeply inside the system.

The policy is to require all C-functions to return a string-pointer. Upon a successfull call, it is a NULL string. Otherwise it contains an encoding of the exceptional state encountered. This message starts with the exception identifer, followed by contextual details.

6.3 Garbage collection

Garbage collection is relatively straightforward, because most values are retained on the stackframe of an interpreter call. However, two storage types and possibly user-defined type garbage collector definitions require attention: BATs and strings.

A key issue is to deal with temporary BATs in an efficient way. References to bats in the buffer pool may cause dangling references at the language level. This appears as soons as your share a reference and delete the BAT from one angle. If not carefull, the dangling pointer may subsequently be associated with another BAT

All string values are private to the VALrecord, which means they have to be freed explicitly before a MAL function returns. The first step is to always safe the destination variable before a function call is made.

All operations are responsible to properly set the reference count of the BATs being produced or destroyed. The libraries should not leave the physical reference count being set. This is only allowed during the execution of a GDK operation. All references should be logical.

6.4 MAL runtime stack

The runtime context of a MAL procedure is allocated on the runtime stack of the corresponding interpreter. Access to the elements in the stack are through index offsets, determined during MAL procedure parsing.

The scope administration for MAL procedures is decoupled from their actual runtime behavior. This means we are more relaxed on space allocation, because the size is determined by the number of MAL procedure definitions instead of the runtime calling behavior. (See mal_interpreter for details on value stack management)

The variable names and types are kept in the stack to ease debugging. The underlying string value need not be garbage collected. Runtime storage for variables are allocated on the stack of the interpreter thread. The physical stack is often limited in size, which calls for safeguarding their value and garbage collection before returning. A malicious procedure or implementation will lead to memory leakage.

A system command (linked C-routine) may be interested in extending the stack. This is precluded, because it could interfere with the recursive calling sequence of procedures. To accommodate the (rare) case, the routine should issue an exception to be handled by the interpreter before retrying. All other errors are turned into an exception, followed by continuing at the exception handling block of the MAL procedure.

7 The MAL Optimizer

One of the prime reasons to design the MAL intermediate language is to have a high-level description for database queries, which is easy to generate by a front-end compiler and easy to decode, optimize and interpret.

An optimizer needs several mechanisms to be effective. It should be able to perform a symbolic evaluation of a code fragment and collect the result in properties for further decision making. The prototypical case is where an optimizer estimates the result size of a selection.

Another major issue is to be able to generate and explore a space of alternative evaluation plans. This exploration may take place up front, but can also be ran at runtime for query fragments.

7.1 The Optimizer Landscape

A query optimizer is often a large and complex piece of code, which enumerates alternative evaluation plans from which 'the best' plan is selected for evaluation. Limited progress has been made sofar to decompose the optimizer into (orthogonal) components, because it is a common believe in research that a holistic view on the problem is a prerequisite to find the best plan. Conversely, commercial optimizers use a cost-model driven approach, which explores part of the space using a limited (up to 300) rewriting rules.

Our hypothesis is that query optimization should be realized with a collection of query optimizer transformers (QOT), each dedicated to a specific task. Furthermore, they are assembled in scenarios to support specific application domains or achieve a desired behavior. Such scenarios are selected on a session basis, a query basis, or dynamically at runtime; they are part of the query plan.

The query transformer list below is under consideration for development. For each we consider its goal, approach, and expected impact. Moreover, the minimal prerequisites identify the essential optimizers that should have done their work already. For example, it doesn't make sense to perform a static evaluation unless you have already propagated the constants using Alias Removal.

Constant expressions Goal: to remove scalar expressions which need be evaluated once during the query lifetime. Rationale: static expressions appear when variables used denote literal constants (e.g. 1+1), when catalog information can be merged with the plan (e.g. max(B.salary)), when session variables are used which are initialized once (e.g. user()). Early evaluation aids subsequent optimization. Approach: inspect all instructions to locate static expressions. Whether they should be removed depends on the expected re-use, which in most cases call for an explicit request upon query registration to do so. The result of a static evaluation provides a ground for alias removal. Impact: relevant for stored queries (MAL functions) Prereq: alias removal, common terms

Relational Expression Optimizer Goal: to evaluate a relational plan using properties of BATs, such as being empty or forming an aligned group. These optimizations assume that the code generator can detect properties while compiling e.g. an SQL query. Impact: high Prereq:

Alias Removal Goal: to reduce the number of variables referenceing the same value, thereby reducing the analysis complexity. Rationale: query transformations often result in replacing the right-hand side expression with a result variable. This pollutes the code block with simple assignments e.g. V:=T. Within the descendant flow the occurrence of V could be replaced by T, provided V is never assigned a new value. Approach: literal constants within a MAL block are already recognized and replaced by a single variable. Impact: medium.

Common Term Optimizer Goal: to reduce the amount of work by avoiding calculation of the same operation twice. Rationale: to simplify code generation for front-ends, they do not have to remember the subexpressions already evaluated. It is much easier to detect at the MAL level. Approach: simply walk through the instruction sequence and locate identical patterns. (Enhance is with semantic equivalent instructions) Impact: High Prereq: Alias Removal

Dead Code Removal Goal: to remove all instructions whose result is not used Rationale: due to sloppy coding or alternative execution paths dead code may appear. Als XML Pathfinder is expected to produce a large number of simple assignments. Approach: Every instruction should produce a value used somewhere else. Impact: low

Heuristic Rule Rewrites Goal: to reduce the volume as quick as possible. Rationale: most queries are focussed on a small part of the database. To avoid carrying too many intermediates, the selection should be performed as early as possible in the process. This assumes that selectivity factors are known upfront, which in turn depends on histogram of the value distribution. Approach: locate selections and push them back/forth through the flow graph. Impact: high

Join Path Optimizer Goal: to reduce the volume produced by a join sequence Rationale: join paths are potentially expensive operations. Ideally the join path is evaluated starting at the smallest component, so as to reduce the size of the intermediate results. Approach: to successfully reduce the volume we need to estimate their processing cost. This calls for statistics over the value distribution, in particular, correlation histograms. If statistics are not available upfront, we have to restore to an incremental algorithm, which decides on the steps using the size of the relations. Impact: high

Operator Sort Goal: to sort the dataflow graph in such a way as to reduce the cost, or to assure locality of access for operands. Rationale: A simple optimizer is to order the instructions for execution by permutation of the query components Approach: Impact:

Singleton Set Goal: to replace sets that are known to produce precisely one tuple. Rationale: Singleton sets can be represented by value pairs in the MAL program, which reduces to a scalar expression. Approach: Identify a set variable for replacement. Impact:

Range Propagation Goal: look for constant ranges in select statements and propagate them through the code. Rationale: partitioned tables and views may give rise to expressions that contain multiple selections over the same BAT. If their arguments are constant, the result of such selects can sometimes be predicted, or the multiple selections can be cascaded into a single operation. Impact: high, should be followed by alias removal and dead code removal

Result Cacher Goal: to reduce the processing cost by keeping track of expensive to compute intermediate results Rationale: Approach: result caching becomes active after an instruction has been evaluated. The result can be cached as long as its underlying operands remain unchanged. Result caching can be made transparent to the user, but affects the other quer optimizers. Impact: high

Vector Execution Goal: to rewrite a query to use a cache-optimal vector implementation Rationale: processing in the cache is by far the best you can get. However, the operands may far exceed the cache size and should be broken into pieces followed by a staged execution of the fragments involved. Approach: replace the query plan with fragment streamers Impact:

Staged Execution Goal: to split a query plan into a number of steps, such that the first response set is delivered as quickly as possible. The remainder is only produced upon request. Rationale: interactive queries call for quick response and an indication of the processing time involved to run it too completion. Approach: staged execution can be realized using a fragmentation scheme over the database, e.g. each table is replaced by a union of fragments. This fragmentation could be determined upfront by the user or is derived from the query and database statistics. impact: high

Code Parallizer Goal: to exploit parallel IO and cpu processing in both SMP and MPP settings. Rationale: throwing more resources to solve a complex query helps, provided it is easy to determine that parallel processing recovers the administrative overhead Approach: every flow path segment can be handled by an independent process thread. Impact: high

Query Evaluation Maps Goal: to avoid touching any tuple that is not relevant for answering a query. Rationale: the majority of work in solving a query is to disgard tuples of no interest and to find correlated tuples through join conditions. Ideally, the database learns these properties over time and re-organizes (or builts a map) to replace disgarding by map lookup. Approach: piggyback selection and joins as database fragmentation instructions Impact: high

MAL Compiler (tactics) Goal: to avoid interpretation of functional expressions Rationale: interpretation of arithmetic expressions with an interpreter is always expensive. Replacing a complex arithmetic expressin with a simple dynamically compiled C-functions often pays off. Especially for cached (MAL) queries Approach: Impact: high

Dynamic Query Scheduler (tactics) Goal: to organize the work in a way so as to optimize resource usage Rationale: straight interpretation of a query plan may not lead to the best use of the underlying resources. For example, the content of the runtime cache may provide an opportunity to safe time by accessing a cached source Approach: query scheduling is the last step before a relation algebra interpreter takes over control. The scheduling step involves a re-ordering of the instructions within the boundaries imposed by the flow graph. impact: medium

Aggregate Groups Goal: to reduce the cost of computing aggregate expressions over times Rationale: many of our applications call for calculation of aggregates over dynamically defined groupings. They call for lengtly scans and it pays to piggyback all aggregate calculates, leaving their result in the cache for later consumption (eg the optimizers) Approach: Impact: High

Data Cube optimizer Goal: to recognize data cube operations Rationale: Approach: Impact:

Demand Driven Interpreter (tactics) Goal: to use the best interpreter and libraries geared at the task at hand Rationale: Interpretation of a query plan can be based on different computational models. A demand driven interpretation starts at the intended output and 'walks' backward through the flow graph to collect the pieces, possibly in a pipelined fashion. (Vulcano model) Approach: merely calls for a different implementation of the core operators Impact: high

Iterator Strength Reduction Goal: to reduce the cost of iterator execution by moving instructions out of the loop. Rationale: although iteration at the MAL level should be avoided due to the inherent low performance compared to built-in operators, it is not forbidden. In that case we should confine the iterator block to the minimal work needed. Approach: inspect the flowgraph for each iterator and move instructions around. Impact: low

Accumulator Evaluation Goal: to replace operators with cheaper ones. Rationale: based on the actual state of the computation and the richness of the supporting libraries there may exists alternative routes to solve a query. Approach: Operator rewriting depends on properties. No general technique. The first implementation looks at calculator expressions such as they appear frequently in the RAM compiler. Impact: high Prerequisite: should be called after common term optimizer to avoid clashes. Status: Used in the SQL optimizer.

Code Inliner Goal: to reduce the calling depth of the interpreter and to obtain a better starting point for code squeezing Rationale: substitution of code blocks (or macro expansion) leads to longer linear code sequences. This provides opportunities for squeezing. Moreover, at runtime building and managing a stackframe is rather expensive. This should be avoided for functions called repeatedly. Impact: medium Status: Used in the SQL optimizer to handle SQL functions.

Code Outliner Goal: to reduce the program size by replacing a group with a single instruction Rationale: inverse macro expansion leads to shorter linear code sequences. This provides opportunities for less interpreter overhead, and to optimize complex, but repetative instruction sequences with a single hardwired call Approach: called explicitly to outline a module (or symbol) Impact: medium

Garbage Collector Goal: to release resources as quickly as possible Rationale: BATs referenced from a MAL program keep resources locked. Approach: In cooperation with a resource scheduler we should identify those that can be released quickly. It requires a forced gargabe collection call at the end of the BAT's lifespan. Impact: large Status: Implemented. Algorithm based on end-of-life-span analysis.

Foreign Key replacements Goal: to improve multi-attribute joins over foreign key constraints Rationale: the code produced by the SQL frontend involves foreign key constraints, which provides many opportunities for speedy code using a join index. Impact: large Status: Implemented in the SQL strategic optimizer.

7.1.1 Optimizer Dependencies

The optimizers are highly targeted to a particular problem. Aside from the resources available to invest in plan optimization, optimizers are partly dependent and may interfere.

To aid selection of the components of interest, we have grouped them in a preferred order of deployment.

Group A:	Code Inliner.
	Constant Expression Evaluator.
	Relational Expression Evaluator.
	Strength Reduction.
Group B:	Common Term Optimizer.
	Query Evaluation Maps.

Group C:	Join Path Optimizer.
	Ranges Propagation.
	Operator Cost Reduction.
	Operator Sort.
	Foreign Key handling.
	Aggregate Groups.
	Data Cube optimizer.
	Heuristic Rule Rewrite.
group D:	Code Parallizer.
	Accumulator Evaluations.
	Result Cacher.
	Replication Manager.
group E:	MAL Compiler.
	Dynamic Query Scheduler.
	Vector Execution.
	Staged Execution.
group F:	Alias Removal.
	Dead Code Removal.
	Garbage Collector.

The interaction also excludes combinations. For example, the Accumulator should be used after the Partition optimizer.

7.1.2 Optimizer Building Blocks

Some instructions are independent of the execution context. In particular, expressions over side-effect free functions with constant parameters could be evaluated before the program block is considered further.

A major task for an optimizer is to select instruction (sequences) which can and should be replaced with cheaper ones. The cost model underlying this decision depends on the processing stage and the overall objective. For example, based on a symbolic analysis their may exist better implementations within the interpreter to perform the job (e.g. hashjoin vs mergejoin). Alternative, expensive intermediates may be cached for later use.

Plan enumeration is often implemented as a Memo structure, which designates alternative sub-plans based on a cost metric. Perhaps we can combine these memo structures into a large table for all possible combinations encountered for a user.

The MAL language does not imply a specific optimizer to be used. Its programs are merely a sequence of specifications, which is interpreted by an engine specific to a given task. Activation of the engine is controlled by a scenario, which currently includes two hooks for optimization; a strategic optimizer and a tactical optimizer. Both engines take a MAL program and produce a (new/modified) MAL program for execution by the lower layers.

MAL programs end-up in the symbol table linked to a user session. An optimizer has the freedom to change the code, provided it is known that the plan derived is invariant to changes in the environment. All others lead to alternative plans, which should be collected as a trail of MAL program blocks. These trails can be inspected for a posteriori analysis, at least in terms of some statistics on the properties of the MAL program structures automatically. Alternatively, the trail may be pruned and re-optimized when appropriate from changes in the environment.

The rule applied for all optimizers is to not-return before checking the state of the MAL program, and to assure the dataflow and variable scopes are properly set. It costs some performance, but the difficulties that arise from optimizer interference are very hard to debug. One of the easiest pitfalls is to derive an optimized version of a MAL function while it is already referenced by or when polymorphic typechecking is required afterwards.

7.1.3 Building Your Own Optimizer

Implementation of your own MAL-MAL optimizer can best be started from refinement of one of the examples included in the code base. Beware that only those used in the critical path of SQL execution are thorouhly tested. The others are developed up to the point that the concept and approach can be demonstrated.

The general structure of most optimizers is to actively copy a MAL block into a new program structure. At each step we determine the action taken, e.g. replace the instruction or inject instructions to achieve the desired goal.

A tally on major events should be retained, because it gives valuable insight in the effectiveness of your optimizer. The effects of all optimizers is collected in a system catalog.

Each optimizer ends with a strong defense line, `optimizerCheck()` It performs a complete type and data flow analysis before returning. Moreover, if you are in debug mode, it will keep a copy of the plan produced for inspection. Studying the differences between optimizer steps provide valuable information to improve your code.

The functionality of the optimizer should be clearly delineated. The guiding policy is that it is always safe to not apply an optimizer step. This helps to keep the optimizers as independent as possible.

It really helps if you start with a few tiny examples to test your optimizer. They should be added to the Tests directory and administered in Tests/All.

Breaking up the optimizer into different components and grouping them together in arbitrary sequences calls for careful programming.

One of the major hurdles is to test interference of the optimizer. The test set is a good starting point, but does not garantee that all cases have been covered.

In principle, any subset of optimizers should work flawlessly. With a few tens of optimizers this amounts to potential millions of runs. Adherence to a partial order reduces the problem, but still is likely to be too resource consumptive to test continously.

7.1.4 Optimizer framework

The large number of query transformers calls for a flexible scheme for the deploy them. The approach taken is to make all optimizers visible at the language level as a signature `optimizer.F()` and `optimizer.F(mod,fcn)`. The latter designates a target function to be inspected by the optimizer `F()`. Then (semantic) optimizer merely inspects a MAL block for their occurrences and activitates it.

The optimizer routines have access to the client context, the MAL block, and the program counter where the optimizer call was found. Each optimizer should remove itself from the MAL block.

The optimizer repeatedly runs through the program until no optimizer call is found.

Note, all optimizer instructions are executed only once. This means that the instruction can be removed from further consideration. However, in the case that a designated function is selected for optimization (e.g., commonTerms(user,qry)) the pc is assumed 0. The first instruction always denotes the signature and can not be removed.

To safeguard against incomplete optimizer implementations it is advisable to perform an optimizerCheck at the end. It takes as arguments the number of optimizer actions taken and the total cpu time spent. The body performs a full flow and type check and re-initializes the lifespan administration. In debugging mode also a copy of the new block is retained for inspection.

7.1.5 Lifespan analysis

Optimizers may be interested in the characteristic of the barrier blocks for making a decision. The variables have a lifespan in the code blocks, denoted by properties beginLifespan,endLifespan. The beginLifespan denotes the intruction where it receives its first value, the endLifespan the last instruction in which it was used as operand or target.

If, however, the last use lies within a BARRIER block, we can not be sure about its end of life status, because a block redo may implictly revive it. For these situations we associate the endLifespan with the block exit.

In many cases, we have to determine if the lifespan interferes with a optimization decision being prepared. The lifespan is calculated once at the beginning of the optimizer sequence. It should either be maintained to reflect the most accurate situation while optimizing the code base. In particular, it means that any move/remove/addition of a MAL instruction calls for either a recalculation or further propagation. Unclear what will be the best strategy. For the time being we just recalc.

The safety property should be relatively easy to determine for each MAL function. This calls for accessing the function MAL block and to inspect the arguments of the signature.

Any instruction may block identification of a common subexpression. It suffices to stumble upon an unsafe function whose parameter lists has a non-empty intersection with the targeted instruction. To illustrate, consider the sequence

```
L1 := f(A,B,C);
...
G1 := g(D,E,F);
...
l2:= f(A,B,C);
...
L2:= h()
```

The instruction G1:=g(D,E,F) is blocking if G1 is an alias for {A,B,C}. Alternatively, function g() may be unsafe and {D,E,F} has a non-empty intersection with {A,B,C}. An alias can only be used later on for readonly (and not be used for a function with side effects).

7.1.6 Flow analysis

In many optimization rules, the data flow dependency between statements is of crucial importance. The MAL language encodes a multi-source, multi-sink dataflow network. Optimizers typically extract part of the workflow and use the language properties to enumerate semantic equivalent solutions, which under a given cost model turns out to result in better performance.

The flow graph plays a crucial role in many optimization steps. It is unclear as yet what primitives and what storage structure is most adequate. For the time being we introduce the operations needed and evaluate them directly against the program

For each variable we should determine its scope of stability. End-points in the flow graph are illustrative as dead-code, that do not produce persistent data. It can be removed when you know there are no side-effect.

Side-effect free evaluation is a property that should be known upfront. For the time being, we assume it for all operations known to the system. The property "unsafe" is reserved to identify cases where this does not hold. Typically, a bun-insert operation is unsafe, as it changes one of the parameters.

7.2 Optimizer Toolkit

In this section, we introduce the collection of MAL optimizers included in the code base. The tool kit is incrementally built, triggered by experimentation and curiousity. Several optimizers require further development to cope with the many features making up the MonetDB system. Such limitations on the implementation are indicated where appropriate.

Experience shows that construction and debugging of a front-end specific optimizer is simplified when you retain information on the origin of the MAL code produced as long as possible. For example, the snippet `sql.insert(col, 12@0, "hello")` can be the target of simple SQL rewrites using the module name as the discriminator.

7.2.1 Access mode optimization

The routine OPTIMIZER.ACCESSMODE() reduces the number of read/write mode changes of variables to a minimum. Especially setting a BAT to write mode is expensive, because it often implies creation of a private copy first.

A full implementation is delayed until really needed.

7.2.2 Accumulator Evaluations

Bulk arithmetic calculations are pretty expensive, because new BATs are created for each expression. This memory hunger can be reduced by detecting opportunities for accummulator processing, i.e. where a (temporary) variable is overwritten. For example, consider the program snippet

```
    t3:= batcalc.*(64,t2);
    t4:= batcalc,+(t1,t3);
    optimizer.accumulators();
```

If variable t2 is a temporary variable and not used any further in the program block, we can re-use its storage space and propagate its alias through the remainder of the code.

```
    batcalc.*(t2,64,t2);
    t4:= batcalc.+(t2,t1,t2);
```

The implementation is straight forward. It only deals with the arithmetic operations available in BATCALC right now. This set will be gradually be extended. The key decision is to determine whether we may overwrite any of the arguments. This is hard to detect at compile time, e.g. the argument may be the result of a binding operation or represent a view over a persistent BAT. Therefore, the compiler injects the call ALGEBRA.REUSE(), which avoids overwriting persistent BATs by taking a copy.

7.2.3 Alias Removal

The routine OPTIMIZER.ALIASREMOVAL() walks through the program looking for simple assignment statements, e.g. V:=W. It replaces all subsequent occurrences of V by W, provided V is assigned a value once and W does not change in the remainder of the code. Special care should be taken for iterator blocks as illustrated in the case below:

```
        i:=0;
        b:= "done";
barrier go:= true;
        c:=i+1;
        d:="step";
        v:=d;
        io.print(v);
        i:=c;
redo go:= i<2;
exit go;
        io.print(b);
        optimizer.aliasRemoval();
```

The constant strings are propagated to the PRINT() routine, while the initial assigment I:=0 should be retained. The code block becomes:

```
        i:=0;
barrier go:= true;
        c:=i+1;
        io.print("step");
        i:=c;
redo go:= i<2;
exit go;
        io.print("done");
```

7.2.4 Code Factorization

In most real-life situations queries are repeatedly called with only slight changes in their parameters. This situation can be captured by the query compilers by keeping a cache of recent query plans. In MonetDB context such queries are represented as parameterized MAL programs.

To further optimize the cached functions it might help to split the query plan into two sections. One section with those actions that do not depend on the arguments given and another section that contains the heart of the query using all information. Such a program can be represented by a MAL factory, which is a re-entrend query plan.

An example of how factorize changes the code is shown below:

```
function test(s:str):lng;
    b:= bat.new(:int,:str);
    bat.insert(b,1,"hello");
    z:= algebra.select(b,s,s);
    i:= aggr.count(z);
    return i;
end test;
optimizer.factorize("user","test");
```

which translates into the following block:

```
factory user.test(s:str):lng;
    b := bat.new(:int,:str);
    bat.insert(b,1,"hello");
barrier always := true;
    z := algebra.select(b,s,s);
    i := aggr.count(z);
    yield i;
    redo always;
exit always;
end test;
```

The factorizer included is a prototype implementation of MAL factorization. The approach taken is to split the program into two pieces and wrap it as a MAL factory. The optimization assumes that the database is not changed on tables accessed only once during the factory lifetime. Such changes should be detected from the outside and followed by re-starting the factory.

A refined scheme where the user can identify the 'frozen' parameters is left for the future. As the mapping of a query to any of the possible available factories to deal with the request. For the time being we simple reorganize the plan for all parameters

The factorize operation interferes with OPTIMIZER.EXPRESSIONACCUMULATION() because that may overwrite the arguments. For the time being, this is captured in a local routine.

7.2.5 Coercion Removal

A simple optimizer that removes coercions that are not needed. They may result from a sloppy code-generator or function call resolution decision. For example:

```
v:= calc.int(23);
```

becomes a single assignment without function call.

The primary role is a small illustration of coding an optimizer algorithm.

7.2.6 Common Subexpression Elimination

Common subexpression elimination merely involves a scan through the program block to detect re-curring statements. The key problem to be addressed is to make sure that the parameters involved in the repeatative instruction are invariant.

The analysis of OPTIMIZER.COMMONTERMS() is rather crude. All functions with possible side-effects on their arguments should have been marked as 'unsafe'. Their use within a MAL block breaks the dataflow graph for all objects involved (BATs, everything kept in boxes).

The common subexpression optimizer locates backwards the identical instructions. It stops as soon as it has found an identical one. Before we can replace the expression with the variable(s) of the previous one, we should assure that we haven;t passed a safety barrier.

```
b:= bat.new(:int,:int);
c:= bat.new(:int,:int);
d:= algebra.select(b,0,100);
e:= algebra.select(b,0,100);
k1:= 24;
k2:= 27;
l:= k1+k2;
l2:= k1+k2;
l3:= l2+k1;
optimizer.commonTerms();
```

is translated into the code block where the first two instructions are not common, because they have side effects.

```
b := bat.new(:int,:int);
c := bat.new(:int,:int);
d := algebra.select(b,0,100);
e := d;
l := calc.+(24,27);
l3 := calc.+(l,24);
```

7.2.7 Constant Expression Evaluation

Expressions produced by compilers involving only constant arguments can be evaluated once. It is particular relevant in functions that are repeatably called. One time queries would not benefit from this extra step.

Consider the following snippet, which contains recursive use of constant arguments

```
a:= 1+1;        io.print(a);
b:= 2;           io.print(b);
c:= 3*b;        io.print(c);
d:= calc.flt(c);io.print(d);
e:= mmath.sin(d);io.print(e);
optimizer.aliasRemoval();
optimizer.evaluate();
```

The code produced by the optimizer would be

```
io.print(2);
io.print(2);
io.print(6);
io.print(6);
io.print(-0.279415488);
```

7.2.8 Costmodel Approach

Cost models form the basis for many optimization decisions. The cost parameters are typically the size of the (intermediate) results and response time. Alternatively, they are running aggregates, e.g. max memory and total execution time, obtained from a simulated run. The current implementation contains a framework and an example for building your own cost-based optimized.

The OPTIMIZER.COSTMODEL() works its way through a MAL program in search for relational operators and estimates their result size. The estimated size is left behind as the property ROWS.

```
r{rows=100} := bat.new(:oid,:int);
s{rows=1000}:= bat.new(:oid,:int);
rs:= algebra.select(s,1,1);
rr:= bat.reverse(r);
j:= algebra.join(rs,rr);
optimizer.costModel();
```

changes the properties of the instructions as follows:

```
r{rows=100} := bat.new(:oid,:int);
s{rows=1000} := bat.new(:oid,:int);
rs{rows=501} := algebra.select(s,1,1);
rr{rows=100} := bat.reverse(r);
j{rows=100} := algebra.join(rs,rr);
```

The cost estimation does not use any statistics on the actual data distribution yet. It relies on the ROWS property provided by the front-end or other optimizers. It just applies a few heuristic cost estimators. However, it ensures that empty results are only tagged with ROWS=0 if the estimate is accurate, otherwise it assumes at least one result row. This property makes it possible to safely pass the result of the cost estimation to the EMPTYSET optimizer for code reduction.

7.2.9 Dead Code Removal

Dead code fragments are recognized by assignments to variables whose value is not consumed any more. It can be detected by marking all variables used as arguments as being relevant. In parallel, we built a list of instructions that should appear in the final result. The new code block is than built in one scan, discarding the superflous instructions.

Instructions that produce side effects to the environment, e.g., printing and BAT updates, should be taken into account. Such (possibly recursive) functions should be marked with a property (UNSAFE). For now we recognize a few important ones Likewise, instructions marked as control flow instructions should be retained.

An illustrative example is the following MAL snippet:

```
V7 := bat.new(:oid,:int);
V10 := bat.new(:int,:oid);
V16 := algebra.markH(V7);
V17 := algebra.join(V16,V7);
V19 := bat.new(:oid,:int);
V22 := bat.new(:oid,:int);
V23 := algebra.join(V16,V22);
```

```
io.print("done");
optimizer.deadCodeRemoval();
```

The dead code removal trims this program to the following short block:

```
io.print("done");
```

A refinement of the dead code comes from using arguments that ceased to exist due to actions taken by an optimizer. For example, in the snippet below the PUSHRANGES optimizer may conclude that variable V31 becomes empty and simply injects a 'dead' variable by dropping the assignment statement. This makes other code dead as well.

```
V30 := algebra.select( V7, 10,100);
V31 := algebra.select(V30,-1,5);
V32 := aggr.sum(V31);
io.print(V32);
```

[implementation pending]

7.2.10 Emptyset Reduction

One of the key decisions during MAL optimization is to estimate the size of the BATs produced and consumed. Two cases are of interest for symbolic processing. Namely, when a BAT is known to contain no tuples and those that have precisely one element. Such information may come from application domain knowledge or as a side effect from symbolic evaluation. It is associated with the program under inspection as properties.

The empty set property is used by the reduction algorithm presented here. Any empty set is propagated through the program to arrive at a smaller and therefore faster evaluation.

For example, consider the following MAL test:

```
    V1 := bat.new(:oid,:int);
    V7 := bat.new(:oid,:int);
    V10{rows=0} := bat.new(:int,:oid);
    V11 := bat.reverse(V10);
    V12 := algebra.kdifference(V7,V11);
    V16 := algebra.markH(V12);
    V17 := algebra.join(V16,V7);
    bat.append(V1,V17);
optimizer.costModel();
optimizer.emptySet();
```

Calling the optimizers replaces this program by the following code snippet.

```
V1 := bat.new(:oid,:int);
V7 := bat.new(:oid,:int);
V10{rows=0} := bat.new(:int,:oid);
V11{rows=0} := bat.new(:oid,:int);
V12 := V7;
V16 := algebra.markH(V12);
V17 := algebra.join(V16,V7);
bat.append(V1,V17);
```

This block can be further optimized using alias propagation and dead code removal. The final block becomes:

```
        V1 := bat.new(:oid,:int);
        V7 := bat.new(:oid,:int);
        V16 := algebra.markH(V7);
        V17 := algebra.join(V16,V7);
        bat.append(V1,V17);
```

During empty set propagation, new candidates may appear. For example, taking the intersection with an empty set creates a target variable that is empty too. It becomes an immediate target for optimization. The current implementation is conservative. A limited set of instructions is considered. Any addition to the MonetDB instruction set would call for assessment on their effect.

7.2.11 SQL specifics

The `bind` operations of SQL requires special care, because they refer to containers that might initially be empty, but aren't upon a second call. This calls for a defensive approach, where a constraint check is left behind to detect a plan whose conditions are not met anymore. Of course, we can drop the constraint if we know that a plan is used onlye once (and not recursively). This can be marked by the SQL compiler, who is in control over the query cache.

7.2.12 Garbage Collection

Garbage collection of temporary variables, such as strings and BATs, takes place upon returning from a function call. Especially for BATs this may keep sizable resources locked longer than strictly necessary. Although the programmer can influence their lifespan by assignment of the NIL, thereby triggering the garbage collector, it is more appropriate to rely on an optimizer to inject these statements. For, it keeps the program smaller and a better target for code-optimizations.

The operation OPTIMIZER.GARBAGECOLLECTOR() removes all BAT references that are at their end of life to make room for new ones. It is typically called as one of the last optimizer steps. A snippet of a the effect of the garbage collector:

```
    t1 := bat.new(:oid,:int);
    t2 := array.grid(132000,8,1,0);
    t3 := array.grid(1,100,10560,0);
    t4 := array.grid(1,100,10560,0,8);
    t5 := batcalc.+(t2,t4);
    t6 := batcalc.oid(t5);
    t7 := algebra.join(t6,t1);
    optimizer.garbageCollector();
```

is translated into the following code block:

```
    t1 := bat.new(:oid,:int);
    t2 := array.grid(132000,8,1,0);
    t3 := array.grid(1,100,10560,0);
    t4 := array.grid(1,100,10560,0,8);
    t5 := batcalc.+(t2,t4);
    bat.setGarbage(t2);
    bat.setGarbage(t4);
```

```
    t6 := batcalc.oid(t5);
    bat.setGarbage(t5);
    t7 := algebra.join(t6,t1);
    bat.setGarbage(t6);
    bat.setGarbage(t1);
```

The current algorithm is straight forward. After each instruction, we check whether its BAT arguments are needed in the future. If not, we inject a garbage collection statement to release them, provided there are no other reasons to retain it. This should be done carefully, because the instruction may be part of a loop. If the variable is defined inside the loop, we can safely remove it.

7.2.13 Heuristic rewrites rules

One of the oldest optimizer tricks in relational query processing is to apply heuristic rules to reduce the processing cost. For example, a selection predicate is pushed through another operator to reduce the number of tuples to consider. Heuristic rewrites are relatively easy to apply in a context where the expression is still close to a relational algebra tree. Therefore, many of the standard rewrite rules are already applied by the SQL front-end as part of its strategic optimization decisions.

Finding rewrite opportunities within a linear MAL program may be more difficult. For example, the pattern should respect the flow of control that may already be introduced. The last resort for the optimizer builder is to write a C-function to look for a pattern of interest and transform it. The code base contains an example how to built such user specific optimizer routines. It translates the pattern:

```
    y:= reverse(R);
    z:= select(y,l,h);
```

into the statement:

```
    z:= selectHead(x,R,l,h)
```

7.2.14 Join Paths

The routine OPTIMIZER.JOINPATH() walks through the program looking for join operations and cascades them into multiple join paths.

```
        a:= bat.new(:oid,:oid);
        b:= bat.new(:oid,:oid);
        c:= bat.new(:oid,:str);
        j1:= algebra.join(a,b);
        j2:= algebra.join(j1,c);
        j3:= algebra.join(b,b);
        j4:= algebra.join(b,j3);
```

The result includes the expanded join expressions. The deadcode optimizer should take care of superflous paths.

```
        a:= bat.new(:oid,:oid);
        j1:= algebra.join(a,b);
        j2:= algebra.joinPath(a,b,c);
        j3:= algebra.join(b,b);
        j4:= algebra.joinPath(b,b,b);
```

7.2.15 Macro and Orcam Processing

The optimizers form the basis for replacing code fragments. Two optimizers are focused on code expansion and contraction. The former involves replacing individual instructions by a block of MAL code, i.e. a MACRO call. The latter depicts the inverse operation, a group of instructions is replaced by a single MAL assignment statement, i.e. a ORCAM call.

The macro facility is limited to type-correct MAL functions, which means that replacement is not essential from a semantic point of view. They could have been called, or the block need not be compressed. It is equivalent to inline code expansion.

The MACRO and ORCAM transformations provide a basis to develop front-end specific code generation templates. The prototypical test case is the following template:

```
function user.joinPath( a:bat[:any_1,:any_2],
                b:bat[:any_2,:any_3],
                c:bat[:any_3,:any_4]):bat[:any_1,:any_4]
address fastjoinpath;
    z:= join(a,b);
    zz:= join(z,c);
    return zz;
end user.joinPath;
```

The call OPTIMIZER.MACRO("USER", "JOINPATH") hunts for occurrences of the instruction call in the block in which it is called and replaces it with the body, i.e. it in-lines the code. Conversely, the OPTIMIZER.ORCAM("USER", "JOINPATH") attempts to localize a block of two join operations and, when found, it is replaced by the direct call to JOINPATH. In turn, type resolution then directs execution to a built-in function FASTJOINPATH.

The current implementation is limited to finding a consecutive sequence, ending in a return-statement. The latter is needed to properly embed the result in the enclosed environment. It may be extended in the future to consider the flow of control as well.

7.2.16 Known issues

The functions subject to expansion or contraction should be checked on 'proper' behavior.

The current implementation is extremely limited. The MACRO optimizer does not recognize use of intermediate results outside the block being contracted. This should be checked and it should block the replacement, unless the intermediates are part of the return list. Likewise, we assume here that the block has a single return statement, which is also the last one to be executed.

The MACRO optimizer can only expand functions. Factories already carry a significant complex flow of control that is hard to simulate in the nested flow structure of an arbitrary function.

The ORCAM optimizer can not deal with calls controlled by a barrier. It would often require a rewrite of several other statements as well.

```
pattern optimizer.macro(targetmod:str,targetfcn:str):void
address OPTmacro
comment "Inline the code of the target function.";
pattern optimizer.macro(mod:str,fcn:str,targetmod:str,targetfcn:str):void
address OPTmacro
```

```
comment "Inline a target function used in a specific function.";

pattern optimizer.orcam(targetmod:str,targetfcn:str):void
address OPTorcam
comment "Inverse macro processor for current function";
pattern optimizer.orcam(mod:str,fcn:str,targetmod:str,targetfcn:str):void
address OPTorcam
comment "Inverse macro, find pattern and replace with a function call.";
```

7.2.17 Merge Tables

A merge association table (MAT) descriptor defines an ordered collection of type compatible BATs, whose union represents a single (virtual) BAT. The MAT may represent a partitioned BAT (see BPM), but could also be an arbitrary collection of temporary BATs within a program fragment.

The MAT definition lives within the scope of a single block. The MAT optimizer simply expands the plan to deal with its components on an instruction basis. Only when a blocking operator is encounted, the underlying BAT is materialized.

The MAT object can not be passed as an argument to any function without first being materialized. Simply because the MAT is not known by the type system and none of the lower level operations is aware of their existence.

In the first approach of the MAT optimizer we assume that the first BAT in the MAT sequence is used as an accumulator. Furthermore, no semantic knowledge is used to reduce the possible superflous (semi)joins. Instead, we limit expansion to a single argument. This is changed at a later stage when a cost-based evaluation be decide differently.

To illustrate, consider:

```
m0:= bat.new(:oid,:int);
m1:= bat.new(:oid,:int);
m2:= bat.new(:oid,:int);
b := mat.new(m0,m1,m2);
s := algebra.select(b,1,3);
i := algebra.count(s);
io.print(s);
io.print(i);
c0 := bat.new(:int,:int);
c1 := bat.new(:int,:int);
c := mat.new(c0,c1);
j := algebra.join(b,s);
io.print(j);
```

The selection and aggregate operations can simply be rewritten using a MAT:

```
_33 := algebra.select(m0,1,3);
_34 := algebra.select(m1,1,3);
_35 := algebra.select(m2,1,3);
    s := mat.new(_33,_34,_35);
    i := 0:int;
```

```
          _36 := aggr.count(_33);
          i := calc.+(i,_36);
          _37 := aggr.count(_34);
          i := calc.+(i,_37);
          _38 := aggr.count(_35);
          i := calc.+(i,_38);
          io.print(i);
```

The print operation does not have MAT semantics yet. It requires a function that does not produce the header with each call. Instead, we can also pack the elements before printing.

```
          s := mat.pack(_33,_34,_35);
          io.print(s);
```

For the join we have to generate all possible combinations, not knowing anything about the properties of the components. The current heuristic is to limit expansion to a single argument. This leads to

```
          b := mat.pack(m0,m1,m2);
          _39 := algebra.join(b,c0);
          _40 := algebra.join(b,c1);
          j := mat.new(_39,_40);
```

The drawback of the scheme is the potential explosion in MAL statements. A challenge of the optimizer is to find the minimum by inspection of the properties of the MAT elements. For example, it might attempt to partially pack elements before proceding. This would be a runtime scheduling decision.

Alternative, the system could use MAT iterators to avoid it. At the cost of more complex program analysis afterwards.

```
    ji:= bat.new(:oid,:int);
    barrier b:= mat.newIterator(m0,m1,m2);
    barrier c:= mat.newIterator(c0,c1);
    ji := algebra.join(b,c);
    bat.insert(j,ji);
    redo c:= mat.newIterator(c0,c1);
    redo b:= mat.newIterator(m0,m1,m2);
    exit c;
    exit b;
```

7.2.18 Multiplex Compilation

The MonetDB operator multiplex concept has been pivotal to easily apply any scalar function to elements in a containers. Any operator CMD came with its multiplex variant [CMD]. Given the signature CMD(T1,..,TN) : TR, it could be applied also as [CMD](BAT[:ANY_1,:T1],...,BAT[ANY_1,TN]) :BAT[ANY_1,TR].

The semantics of the multiplex is to perform the positional join on all bat-valued parameters, and to execute the CMD for each combination of matching tuples. All results are collected in a result BAT. All but one argument may be replaced by a scalar value.

The generic solution to the multiplex operators is to translate them to a MAL loop. A snippet of its behaviour:

```
    b:= bat.new(:int,:int);
    bat.insert(b,1,1);
    c:bat[:int,:int]:= mal.multiplex("calc.+",b,1);
        optimizer.multiplex();
```

The current implementation requires the target type to be mentioned explicitly. The result of the optimizer is:

```
    b := bat.new(:int,:int);
    bat.insert(b,1,1);
    _8 := bat.new(:int,:int);
barrier (_11,_12,_13):= bat.newIterator(b);
    _15 := calc.+(_13,1);
    bat.insert(_8,_12,_15);
    redo (_11,_12,_13):= bat.hasMoreElements(b);
catch MALException;
        #ignore any error
    redo (_11,_12,_13):= bat.hasMoreElements(b);
exit MALException;
exit (_11,_12,_13);
    c := _8;
```

7.2.19 BAT Partitions

Limitations on the addressing space in older PCs and the need for distributed storage makes that BATs ideally should be looked upon as a union of smaller BATs which are processed within the (memory) resource limitations given.

The PARTITION() optimizer with the supportive bat partition library bpm addresses the issue with an adaptive database segmentation algorithm. It is designed incrementally with a focus on supporting the SQL front-end. In particularly, the operators considered is a limited subset of MAL. Occurrence of an operator outside this set terminates the optimizer activities.

The operation OPTIMIZER.PARTITIONS() hunts for bindings of SQL column BATs and prepare code for using partitioned versions instead.

We use two implementations. The first one attempts to find segments of linear dependent data and builds an iterator around it. This approach is tricky, because you have to take care of special cases. In particular, the semantics of the operators on the sequence construction posed quite some problems.

The naive() approach simply looks at individual operations and surround them with an iterator. An alias table is kept around for re-use and detect already partitioned operands. The drawback is that potentially a partitioned BAT is read multiple times [it depends on the re-use of variables, which can be calculated] and write+read of intermediates. Experiments should demonstrate the optimal one.

7.2.20 Peephole optimization

Recursive descend query compilers easily miss opportunities for better code generation, because limited context is retained or lookahead available. The peephole optimizer is built around such recurring patterns and compensates for the compilers 'mistakes'. The collection of peephole patterns should grow over time and front-end specific variations are foreseen.

The SQL frontend heavily relies on a pivot table, which is a generated oid sequence. Unfortunately, this is not seen and the pattern '$i := calc.oid(0@0); $j:= algebra.markT($k,$i);' occurs often. This can be replaced with '$j:= algebra.markT($k)';

Another example of a 2-way instruction sequence produced is then '$j:= algebra.markT($k); $l:= bat.reverse($j);', which can be replaced by '$l:= algebra.markH($k);'.

The reverse-reverse operation also falls into this category. Reversal pairs may result from the processing scheme of a front-end compiler or from a side-effect from other optimization steps. Such reversal pairs should be removed as quickly as possible, so as to reduce the complexity of finding alternative optimization opportunities. As in all cases we should ensure that the intermediates dropped are not used for other purposes as well.

```
r:bat[:int,:int]:= bat.new(:int,:int);
o:= calc.oid(0@0);
z:= algebra.markT(r,o);
rr:= bat.reverse(z);
s := bat.reverse(r);
t := bat.reverse(s);
io.print(t);
optimizer.peephole();
```

which is translated by the peephole optimizer into:

```
r:bat[:int,:int] := bat.new(:int,:int);
rr := algebra.markH(r);
io.print(r);
```

7.2.21 Query Execution Plans

A commonly used data structure to represent and manipulate a query is a tree (or graph). Its nodes represent the operators and the leaves the operands. Such a view comes in handy when you have to re-organize whole sections of code or to built-up an optimized plan bottom up, e.g. using a memo structure.

The MAL optimizer toolkit provides functions to overlay the body of any MAL block with a tree (graph) structure and to linearize them back into a MAL block. The linearization order is determined by a recursive descend tree walk from the anchor points in the source program.

To illustrate, consider the code block:

```
#T1:= bat.new(:int,:int);
#T2:= bat.new(:int,:int);
#T3:= bat.new(:int,:int);
#T4:= bat.new(:int,:int);
a:= algebra.select(T1,1,3);
b:= algebra.select(T2,1,3);
c:= algebra.select(T3,0,5);
d:= algebra.select(T4,0,5);
e:= algebra.join(a,c);
f:= algebra.join(b,d);
h:= algebra.join(e,f);
```

```
optimizer.dumpQEP();
```

which produces an indented structure of the query plan.

```
h := algebra.join(e,f);
    e := algebra.join(a,c);
        a := algebra.select(T1,1,3);
            T1 := bat.new(:int,:int);
        c := algebra.select(T3,0,5);
            T3 := bat.new(:int,:int);
    f := algebra.join(b,d);
        b := algebra.select(T2,1,3);
            T2 := bat.new(:int,:int);
        d := algebra.select(T4,0,5);
            T4 := bat.new(:int,:int);
```

Any valid MAL routine can be overlayed with a tree (graph) view based on the flow dependencies, but not all MAL programs can be derived from a simple tree. For example, the code snippet above when interpreted as a linear sequence can not be represented unless the execution order itself becomes an operator node itself.

However, since we haven't added or changed the original MAL program, the routine `qep.propagate` produces the orginial program, where the linear order has priority. If, however, we had entered new instructions into the tree, they would have been placed in close proximity of the other tree nodes.

Special care is given to the flow-of-control blocks, because to produce a query plan section that can not easily be moved around. [give dot examples]

7.2.22 Range Propagation

Almost all queries are interested in a few slices of the table. If applied to a view, the query plans often contain multiple selections over the same column. They may also have fixed range arguments comming from fragmentation criteria.

The purpose of the PUSHRANGES optimizer is to minimize the number of table scans by cascading the range terms as much as possible. Useless instructions are removed from the plan.

```
b := bat.new(:oid,:int);
s1:= algebra.select(b,1,100);
s2:= algebra.select(s1,5,95);
s3:= algebra.select(s2,50,nil);
s4:= algebra.select(s3,nil,75);
optimizer.pushranges();
```

This lengthly sequence can be compressed into a single one:

```
b := bat.new(:oid,:int);
s1:= algebra.select(b,50,75);
```

A union over two range selections from a single source could also be a target.

```
t1:= algebra.select(b,1,10);
t2:= algebra.select(b,0,5);
t3:= algebra.union(t1,t2);
```

would become

```
        t3:= algebra.select(0,10);
```

7.2.23 Remote Queries

MAL variables may live at a different site from where they are used. In particular, the SQL front-end uses portions of remote BATs as replication views. Each time such a view is needed, the corresponding BAT is fetched and added to the local cache.

Consider the following snippet produced by a query compiler,

```
mid:= mserver.reconnect("s0_0","localhost",50000,"monetdb","monetdb","mal");
b:bat[:oid,:int] := mserver.bind(mid,"rvar");
c:=algebra.select(b,0,12);
io.print(c);
d:=algebra.select(b,5,10);
low:= 5+1;
e:=algebra.select(d,low,7);
i:=aggr.count(e);
io.printf(" count %d\n",i);
io.print(d);
```

which uses a BAT RVAR stored at the remote site DB1.

There are several options to execute this query. The remote BAT can be fetched as soon as the bind operation is executed, or a portion can be fetched after a remote select, or the output for the user could be retrieved. An optimal solution depends on the actual resources available at both ends and the time to ship the BAT.

The remote query optimizer assumes that the remote site has sufficient resources to handle the instructions. For each remote query it creates a private connection. It is re-used in subsequent calls .

The remote environment is used to execute the statements. The objects are retrieved just before they are locally needed.

```
mid:= mserver.reconnect("s0_0","localhost",50000,"monetdb","monetdb","mal");
mserver.rpc(mid,"b:bat[:oid,:int] :=bbp.bind(\"rvar\");");
mserver.rpc(mid,"c:=algebra.select(b,0,12);");
c:bat[:oid,:int]:= mserver.rpc(mid, "io.print(c);");
io.print(c);
mserver.rpc(mid,"d:=algebra.select(b,5,10);");
low:= 5+1;
mserver.put(mid,"low",low);
mserver.rpc(mid,"e:=algebra.select(d,low,7);");
mserver.rpc(mid,"i:=aggr.count(d);");
i:= mserver.rpc(mid,"io.print(i);");
io.printf(" count %d\n",i);
io.print(d);
```

To reduce the number of interprocess communications this code can be further improved by glueing the instructions together when until the first result is needed.

7.2.24 Singleton Set Reduction

Application semantics and precise cost analysis may identify the result of an operation to produce a BAT with a single element. Such variables can be tagged with the property SINGLETON, whereafter the operation OPTIMIZER.SINGLETON() derives an MAL program using a symbolic evaluation as far as possible.

During its evaluation, more singleton sets can be created, leading to a ripple effect through the code. A non-optimizable instruction leads to a construction of a new table with the single instance.

```
        b:= bat.new(:int,:int);
        bat.insert(b,1,2);
        c{singleton}:= algebra.select(b,0,4);
        d:= algebra.markH(c);
        io.print(d);
        optimizer.singleton();
```

is translated by into the code block

```
        b := bat.new(:int,:int);
        bat.insert(b,1,2);
        c{singleton} := algebra.select(b,0,4);
        (_15,_16):= bat.unpack(c{singleton});
        d := bat.pack(nil,_16);
        io.print(d);
```

7.2.25 Stack Reduction

The compilers producing MAL may generate an abundance of temporary variables to hold the result of expressions. This leads to a polution of the runtime stack space, because space should be allocated and garbage collection tests should be performed.

Likewise, constant duplicates are scattered around the stack. They are located and merged.

The routine OPTIMIZER.REDUCE() reduces the number of scratch variables to a minimum. All scratch variables of the same underlying type share the storage space. The result of this optimizer can be seen using the MonetDB debugger, which marks unused variables explicitly. Experience with the SQL front-end shows, that this optimization step easily reduces the stack consumption by over 20\%.

This optimizer needs further testing. Furthermore, the other optimizers should be careful in setting the isused property, or this property can again be easily derived.

7.2.26 Strength Reduction

An effective optimization technique in compiler construction is to move invariant statements out of the loops. The equivalent strategy can be applied to the guarded blocks in MAL programs. Any variable introduced in a block and assigned a value using a side-effect free operation is a candidate to be moved. Furthermore, it may not be used outside the block and the expression may not depend on variables assigned a value within the same block.

```
        j:= "hello world";
barrier go:=true;
```

```
        i:= 23;
        j:= "not moved";
        k:= j;
        io.print(i);
        redo go:= false;
exit go;
        z:= j;
optimizer.strengthReduction();
```

which is translated into the following code:

```
        j := "hello world";
        i := 23;
barrier go := true;
        j := "not moved";
        k := j;
        io.print(i);
        redo go:= false;
exit go;
        z:= j;
```

Application is only applicable to loops and not to guarded blocks in general, because execution of a statement outside the guarded block consumes processing resources which may have been prohibited by the block condition.

For example, it doesn't make sense to move creation of objects outside the barrier.

8 The MAL Debugger

In practice it is hard to write a correct MAL program the first time around. Instead, it is more often constructed by trial-and-error. As long as there are syntax and semantic errors the MAL compiler provides a sufficient handle to proceed. Once it passes the compiler we have to resort to a debugger to assess its behavior.

Note, the MAL debugger described here can be used in conjunction with the textual interface client *mclient* only. The JDBC protocol does not permit passing through information that 'violates' the protocol.

8.1 Program Debugging

To ease debugging and performance monitoring, the MAL interpreter comes with a gdb-like debugger. An illustrative session elicits the functionality offered.

```
>function test(i:int):str;
> io.print(i);
> i:= i*2;
> b:= bat.new(:int,:int);
> bat.insert(b,1,i);
> io.print(b);
> return test:= "ok";
>end test;
>user.test(1);
[ 1 ]
#-----------------#
# h     t         # name
# int   int       # type
#-----------------#
[ 1,      2       ]
```

The debugger can be entered at any time using the call mdb.start(). An overview of the available commands is readily available.

```
>mdb.start();
#mdb !end main;
mdb>help
next             -- Advance to next statement
continue         -- Continue program being debugged
catch            -- Catch the next exception
break [<var>]    -- set breakpoint on current instruction or <var>
delete [<var>]   -- remove break/trace point <var>
debug <int>      -- set kernel debugging mask
dot [<int>] [<file>]  -- generate the dependency graph
step             -- advance to next MAL instruction
module           -- display a module signatures
atom             -- show atom list
finish           -- finish current call
exit             -- terminate executionr
```

```
quit             -- turn off debugging
list <obj>       -- list current program block
List <obj>       -- list with type information
var  <obj>       -- print symbol table for module
optimizer <obj>  -- display program after optimizer step
print <var>      -- display value of a variable
print <var> <cnt>[<first>] -- display BAT chunk
info <var>       -- display bat variable properties
run              -- restart current procedure
where            -- print stack trace
down             -- go down the stack
up               -- go up the stack
trace <var>      -- trace assignment to variables
set {timer,flow,io,memory,bigfoot} -- set trace switches
unset            -- turn off switches
help             -- this message
mdb>
```

The term <OBJ> is an abbreviation for a MAL operation <MOD>.<FCN>, optionally extended with a version number, i.e. [<NR>]. The VAR denotes a variable in the current stack frame. Debugger commands may be abbreviated.

We walk our way through a debugging session, highlighting the effects of the debugger commands. The call to mdb.start() has been encapsulated in a complete MAL function, as shown by issuing the list command. A more detailed listing shows the binding to the C-routine and the result of type resolution.

```
>mdb.start();
#end main;
mdb>l
function user.main():int;
mdb.start();
end main;
mdb>L
function user.main():int;       # 0  (main:int)
mdb.start();        # 1 MDBstart (_1:void)
end main;       # 2
```

The user module is the default place for function defined at the console. The modules loaded can be shown typeing the command 'module' (or 'm' for short). The function signatures become visible using the module and optionally the function name.

```
mdb>m alarm
#command alarm.alarm(secs:int,action:str):void address ALARMsetalarm;
#command alarm.ctime():str address ALARMctime;
#command alarm.epilogue():void address ALARMepilogue;
#command alarm.epoch():int address ALARMepoch;
#command alarm.prelude():void address ALARMprelude;
#command alarm.sleep(secs:int):void address ALARMsleep;
#command alarm.time():int address ALARMtime;
#command alarm.timers():bat[:str,:str] address ALARMtimers;
```

```
#command alarm.usec():lng address ALARMusec;
mdb>m alarm.sleep
#command alarm.sleep(secs:int):void address ALARMsleep;
mdb>
```

The debugger mode is left with a <return>. Any subsequent MAL instruction re-activates the debugger to await for commands. The default operation is to step through the execution using the 'next' ('n') or 'step' ('s) commands, as shown below.

```
>user.test(1);
#    user.test(1);
mdb>n
#    io.print(i);
mdb>
[ 1 ]
#    i := calc.*(i,2);
mdb>
#    b := bat.new(:int,:int);
mdb>
```

The last instruction shown is next to be executed. The result can be shown using a print statement, which contains the location of the variable on the stack frame, its name, its value and type. The complete stack frame becomes visible with 'values' ('v') command:

```
#    bat.insert(b,1,i);
mdb>
#    io.print(b);
mdb>v
#Stack for 'test' size=32 top=11
#[0] test        = nil:str
#[1] i   = 4:int
#[2] _2  = 0:int   unused
#[3] _3  = 2:int  constant
#[4] b   = <tmp_1226>:bat[:int,:int]   count=1 lrefs=1 refs=0
#[5] _5  = 0:int   type variable
#[6] _6  = nil:bat[:int,:int]   unused
#[7] _7  = 1:int  constant
#[8] _8  = 0:int   unused
#[9] _9  = "ok":str  constant
```

The variables marked 'unused' have been introduced as temporary variables, but which are not referenced in the remainder of the program. It also illustrates basic BAT properties, a complete description of which can be obtained using the 'info' ('i') command. A sample of the BAT content can be printed passing tuple indices, e.g. 'print b 10 10' prints the second batch of ten tuples.

8.2 Handling Breakpoints

A powerful mechanism for debugging a program is to set breakpoints during the debugging session. The breakpoints are designated by a target variable name, a [module.]function name, or a MAL line number (#<number>).

The snippet below illustrates the reaction to set a break point on assignment to variable 'i'.

```
>mdb.start();
#end main;
mdb>
>user.test(1);
#    user.test(1);
mdb>break i
breakpoint on 'i' not set
mdb>n
#    io.print(i);
mdb>break i
mdb>c
[ 1 ]
#    i := calc.*(i,2);
mdb>
```

The breakpoints remain in effect over multiple function calls. They can be removed with the DELETE statement. A list of all remaining breakpoints is obtained with BREAKPOINTS.

The interpreter can be instructed to call the debugger as soon as an exception is raised. Simply add the instruction MDB.SETCATCH(TRUE).

8.3 Profile Switches

Switches control the level of detail output shown while debugging or tracing program execution. They are toggled with the SET and UNSET command. The following switches are currently supported:

TIMER	activates a listing of all instructions being executed. It is measured in wall-clock time.
FLOW	shows the total byte size of all BAT target results and input arguments. It is a good indicator on the amount of data being processed.
MEMORY	keeps track on growing memory needs.
IO	keeps track on the amount of physical IO and is used to detect operators consuming excessive amounts of space.
BIGFOOT	keeps track of the current and maximum virtual memory footprint of the BATs.[incomplete]

The snippet below shows setting the MEMORY and TIMER switch. The switches take effect at the next instruction.

```
mdb>set timer
mdb>set flow
mdb>c
[ 3 ]
#    26 usec#   0  0#    io.print(i=3)
#     6 usec#   0  0#    i := calc.*(i=6, _3=2)
#    10 usec#   0  0#    b := bat.new(_5=0, _6=0)
```

```
#     7 usec#   0  8#    bat.insert(b=<tmp_167>bat[:int,:int]{1}, _8=1, i=6)
#-----------------#
# h     t         # name
# int   int       # type
#-----------------#
[ 1,      6       ]
#    41 usec#   0  8#    io.print(b=<tmp_167>bat[:int,:int]{1})
#     7 usec#   0  0#    return test := "ok";
#   211 usec#   0  0#    user.test(_2=3)
```

8.4 Program Inspection

The debugger commands available for inspection of the program and symbol tables are:

LIST (LIST) [<MOD>.<FCN>['['<NR>']']]
: A listing of the current MAL block, or one designated by the <mod>.<fcn> is produced. The [<NR>] extension provides access to an element in the MAL block history. The alternative name 'List' also produces the type information.

OPTIMIZER [<MOD>.<FCN>['['<NR>']']]
: Gives an overview of the optimizer actions in the history of the MAL block. Intermediate results can be accessed using the list command.

ATOMS Lists the atoms currently known

MODULES [<MOD>]
: Lists the modules currently known. An optional <mod> argument produces a list of all signatures within the module identified.

DOT [<MOD>.<FCN>['['<NR>']']] [<FILE>]
: A dataflow diagram can be produced using the DOT command. It expects a function identifier with an optional history index and produces a file for the Linux program DOT, which can produce a nice, multi-page graph to illustrate plan complexity.

```
mdb>dot user.test
```

This example produces the USER-TST.DOT in the current working directory. The program call

```
dot -Tps user-tst-0.dot -o user-tst-0.ps
```

creates a postscript file with the graphs. With the Adobe reader professional you can break it up into multiple pages. An alternative is the program available from HTTP://WWW.TUG.ORG/TEX-ARCHIVE/SUPPORT/POSTER/POSTER.C The result is shown in the figure below:

Since the flow graphs become rather complex, an optional variable list limits its size.[TODO]

8.5 Runtime Inspection and Reflection

Part of the debugger functionality can also be used directly with MAL instructions. The execution trace of a snippet of code can be visualized encapsulation with mdb.setTrace(true) and mdb.setTrace(false). Likewise, the performance can be monitored with the command mdb.setTimer(on/off). Using a boolean argument makes it easy to control the (performance) trace at run time. The following snippet shows the effect of patching the test case.

```
>function test(i:int):str;
> mdb.setTrace(true);
> io.print(i);
> i:= i*2;
> b:= bat.new(:int,:int);
> bat.insert(b,1,i);
> io.print(b);
> mdb.setTrace(false);
> return test:= "ok";
>end test;
>user.test(1);
#    mdb.setTrace(_3=true)
[ 1 ]
#    io.print(i=1)
#    i := calc.*(i=2, _5=2)
#    b := bat.new(_7=0, _8=0)
#    bat.insert(b=<tmp_1226>, _10=1, i=2)
#-----------------#
# h     t         # name
# int   int       # type
#-----------------#
[ 1,      2       ]
#    io.print(b=<tmp_1226>)
#   261 usec!    user.test(_2=1)
>
```

The command mdb.setTimer() toggles the performance traceing flag. The argument is a boolen to designate its state. The primary output of the timer switch is statistics in micro-seconds, the memory tracer shows the arena increment, and the IO tracer shows in- and out-blocks. The time spent on preparing the trace information is excluded from the report. For more detailed timing information the Linux tool *valgrind* may be of help.

The routines mdb.setFlow(), mdb.setMemory(), and mdb.setIO() (de-)activate the other switches.

```
>function test(i:int):str;
> mdb.setTimer(true);
> io.print(i);
> i:= i*2;
> b:= bat.new(:int,:int);
> bat.insert(b,1,i);
> io.print(b);
> mdb.setTimer(false);
> return test:= "ok";
>end test;
```

```
>user.test(1);
#     6 usec#    mdb.setTimer(_3=true)
[ 1 ]
#    43 usec#    io.print(i=1)
#     5 usec#    i := calc.*(i=2, _5=2)
#    24 usec#    b := bat.new(_7=0, _8=0)
#    10 usec#    bat.insert(b=<tmp_1226>, _10=1, i=2)
#-----------------#
# h     t         # name
# int   int       # type
#-----------------#
[ 1,      2       ]
#   172 usec#    io.print(b=<tmp_1226>)
#   261 usec#    user.test(_2=1)
```

It is also possible to activate the debugger from within a program using mdb.start(). It remains in this mode until you either issue a quit command, or the command mdb.stop() instruction is encountered. The debugger is only activated when the user can direct its execution from the client interface. Otherwise, there is no proper input channel and the debugger will run in trace mode.

The program listing functionality of the debugger is also captured in the MAL debugger module. The current code block can be listed using mdb.list() and mdb.List(). An arbitrary code block can be shown with mdb.list(*module,function*) and mdb.List(*module,function*). A BAT representation of the current function is return by mdb.getDefinition().

The symbol table and stack content, if available, can be shown with the operations mdb.var() and mdb.list(*module,function*) Access to the stack frames may be helpful in the context of exception handling. The operation mdb.getStackDepth() gives the depth and individual elements can be accessed as BATs using mdb.getStackFrame(*n*). The top stack frame is accessed using mdb.getStackFrame().

8.6 Debugger Attachment

Debugging a running MAL process is simplified with a few hooks in the kernel. It is illustrated with a short example.

First open a client connection with the user using MAL as preferred language. Then the state of the system can be inspected, in particular, the clients active can be looked up.

```
> b:= clients.getLogins();
> c:= clients.getUsers();
> io.print(b,c);
```

Locate the process you are interested in and obtain its identifier, say N (the first column in the list above). The next step is to gracefully put the running process into debugging mode without jeopardizing the application running.

```
> clients.suspend(N);
```

As soon as the next MAL instruction of process N starts the target process is put to sleep and the debugger gains control. You can now inspect the execution context. The control ends when you leave the debugger with a 'quit' command.

9 The MAL Profiler

A key issue in the road towards a high performance implementation is to understand where resources are being spent. This information can be obtained using different tools and at different levels of abstraction. A coarse grain insight for a particular application can be obtained using injection of the necessary performance capturing statements in the instruction sequence. Fine-grain, platform specific information can be obtained using existing profilers, like valgrind (http://www.valgrind.org), or hardware performance counters.

The MAL profiler collects detailed performance information, such as cpu, memory and statement information. It is optionally extended with IO activity, which is needed for coarse grain profiling only, and estimated bytes read/written by an instruction.

The execution profiler is supported by hooks in the MAL interpreter. The default strategy is to ship an event record immediately over a stream to a separate performance monitor, formatted as a tuple. An alternative strategy is preparation for off-line performance analysis.

Reflective performance analysis is supported by an event cache, the event log becomes available as a series of BATs.

9.1 Event Filtering

The profiler supports selective retrieval of such information by tagging the instructions of interest. This means that a profiler call has a global effect, all concurrent users are affected by the performance overhead. Therefore, it is of primary interest to single user sessions.

The example below illustrates how the different performance counter groups are activated, instructions are filtered for tracking, and where the profile information is retained for a posteriori analysis.

```
    #profiler.activate("event");
    #profiler.activate("pc");
    #profiler.activate("operation");
    profiler.activate("time");
    profiler.activate("ticks");
    #profiler.activate("cpu");
    #profiler.activate("memory");
    #profiler.activate("io");
    #profiler.activate("bytes");
    #profiler.activate("diskspace");
    profiler.activate("statement");
    profiler.setFilter("*","insert");
    profiler.setFilter("*","print");

    profiler.openStream("/tmp/MonetDBevents");
    profiler.start();
    b:= bbp.new(:int,:int);
    bat.insert(b,1,15);
    bat.insert(b,2,4);
    bat.insert(b,3,9);
```

```
    io.print(b);
    profiler.stop();
    profiler.closeStream();
```

We are interested in all functions name INSERT and PRINT. A wildcard can be used to signify any name, e.g. no constraints are put on the module in which the operations are defined. Several profiler components are ignored, shown by commenting out the code line.

Execution of the sample leads to the creation of a file with the following content. The ticks are measured in micro-seconds.

```
# time, ticks,  stmt  # name
[ "15:17:56",   12,   "_27 := bat.insert(<tmp_15>{3},1,15);" ]
[ "15:17:56",   2,    "_30 := bat.insert(<tmp_15>{3},2,4);"  ]
[ "15:17:56",   2,    "_33 := bat.insert(<tmp_15>{3},3,9);"  ]
[ "15:17:56",   245,  "_36 := io.print(<tmp_15>{3});",   ]
```

9.2 Event Caching

Aside from shipping events to a separate process, the profiler can keep the events in a local BAT group. It is the default when no target file has been opened to collect the information.

Ofcourse, every measurement scheme does not come for free and may even obscure performance measurements obtained through e.g. valgrind. The separate event caches can be accessed using the operator PROFILER.GETTRACE(*name*). The current implementation only supports access to TIME,TICKS,PC,STATEMENT. The event cache can be cleared with PROFILER.CLEARTRACE().

Consider the following MAL program snippet:

```
    profiler.setAll();
    profiler.start();
    b:= bbp.new(:int,:int);
    bat.insert(b,1,15);
    io.print(b);
    profiler.stop();
    s:= profiler.getTrace("statement");
    t:= profiler.getTrace("ticks");
    io.print(s,t);
```

The performance result of the program execution becomes:

```
#-----------------------------------------------------------#
# h     t                                       t         # name
# int   str                                     int       # type
#-----------------------------------------------------------#
[ 1,      "b := bbp.new(0,0);",                   51      ]
[ 2,      "$6 := bat.insert(<tmp_22>,1,15);",     16      ]
[ 3,      "$9 := io.print(<tmp_22>);",            189     ]
```

9.3 Monitoring Variables

The easiest scheme to obtain performance data is to retrieve the performance properties of an instruction directly after it has been executed using getEvent(). It reads the profiling stack maintained, provided you have started monitoring.

```
profiler setFilter(b);
profiler.start();
....
b:= algebra.select(a,0 1000); # some expensive operation
(clk, memread, memwrite):= profiler.getEvent();
...
profiler.stop();
```

9.3.1 The Stethoscope

The performance profiler infrastructure provides precisely control through a MAL program. Often, however, inclusion of profiling statements is an afterthought.

The program `stethoscope` addresses this situation by providing a simple application that can attach itself to a running server and extracts the profiler events from concurrent running queries.

The arguments to `stethoscope` are the profiler properties to be traced and the applicable filter expressions. For example,

```
stethoscope -t bat.insert algebra.join
```

tracks the microsecond ticks of two specific MAL instructions. A synopsis of the calling conventions:

```
    stethoscope [options] +[aefoTtcmibds] @verb{ { }<mod>.<fcn> @verb{ } }
    -d | --dbname=<database_name>
    -u | --user=<user>
    -P | --password=<password>
    -p | --port=<portnr>
    -g | --gnuplot=<boolean>
    -h | --host=<hostname>

Event selector:
        a =aggregates
    e =event
    f =function
    o =operation called
    T =time
    t =ticks
    c =cpu statistics
    m =memory resources
    i =io resources
    b =bytes read/written
    d =diskspace needed
    s =statement
```

Ideally, the stream of events should be piped into a 2D graphical tool, like xosview (Linux). A short term solution is to generate a gnuplot script to display the numerics organized as time lines. With a backup of the event lists give you all the information needed for a descent post-mortem analysis.

10 The MAL Modules

10.1 Module Loading

The server is bootstrapped by processing a MAL script with module definitions or extensions. For each module file encountered, the object library lib_<modulename>.so is searched for in .../lib(64)/MonetDB. The corresponding signature are defined in .../lib(64)/<modulename>.mal.

The default bootstrap script is called .../lib(64)/MonetDB/mal_init.mal and it is designated in the configuration file as the mal_init property. The rationale for this set-up is that database administrators can extend/overload the bootstrap procedure without affecting the distributed software package. It merely requires a different direction for the mal_init property.

The scheme also isolates the functionality embedded in modules from inadvertise use on non-compliant databases. [access control issue, how to limit what a user can do on a database?]

Unlike previous versions of MonetDB, modules can not be unloaded. Dynamic libraries are always global and, therefore, it is best to load them as part of the server initialization phase.

The MAL program should be compiled with -rdynamic and -ldl. This enables loading the routines and finding out the address of a particular routine

The mapping from MAL module.function() identifier to an address is resolved in the function getAddress. Since all modules libraries are loaded completely with GLOBAL visibility, it suffices to provide the internal function name. In case an attempt to link to an address fails, a final attempt is made to locate the *.o file in the current directory.

10.2 Module file loading

The default location to search for the module is in monet_mod_path unless an absolute path is given.

Module loading relies on the operating system's abilities to load shared libraries dynamically at runtime.

To speedup restart and to simplify debugging, the MonetDB server can be statically linked with some (or all) of the modules libraries. A complicating factor is then to avoid users to initiate another load of the module file, because it would lead to a `dlopen()` error.

The partial way out of this dilema is to administer somewhere the statically bound modules, or to enforce that each module comes with a known routine for which we can search. In the current version we use the former approach.

This section contains a synopsis of the modules being shipped and which use knowledge of the MAL runtime context. They are sorted by module name and repetitive reading may be required to understand all details.

batExtensions	Extensions to the kernel/bat module.

BBP	BAT buffer pool interface.
Box	Box variable interface.
Chopper	Break a collection into chunks.
Clients	Client record inspection.
Constants	Global system defined values.
Factory	Factory management interface.
Inspect	Inspect the runtime symbol table(s).
I/O	The input/output interface.
Language	MAL language extension features.
MDB	MAL debugger interface.
Manual	Online manual material.
Mserver	MonetDB server interface.
MAT	Multiple association tables.
PBM	Dealing with partitioned BATS.
Profiler	Performance profiler.
PCRE	Regular expression handling over strings.
Statistics	A server-side statistics catalog.
Table	Table output interface.
Transactions	Transaction interface

10.3 BAT Extensions

The kernel libraries are unaware of the MAL runtime semantics. This calls for declaring some operations in the MAL module section and register them in the kernel modules explicitly.

A good example of this borderline case are BAT creation operations, which require a mapping of the type identifier to the underlying implementation type.

Another example concerns the (un)pack operations, which direct access the runtime stack to (push)pull the values needed.

`pattern bat.new(ht:any_1, tt:any_2, b:bat[:any_3,:any_4]) :bat[:any_1,:any_2]`
address CMDBATclone comment "Creates a new empty transient BAT by cloning another";

`pattern bat.new(ht:any_1, tt:any_2) :bat[:any_1,:any_2]`
address CMDBATnew comment "Creates a new empty transient BAT, with head- and tail-types as indicated.";

`pattern bat.new(ht:any_1, tt:any_2, size:int) :bat[:any_1,:any_2]`
address CMDBATnewint comment "Creates a new BAT with sufficient space.";

`pattern bat.new(ht:any_1, tt:any_2, size:lng) :bat[:any_1,:any_2]`
address CMDBATnew comment "Creates a new BAT and allocate space.";

`pattern bat.new(ht:oid, tt:any_2, size:int) :bat[:oid,:any_2]`
address CMDBATnewint;

`pattern bat.new(ht:oid, tt:any_2, size:lng) :bat[:oid,:any_2]`
address CMDBATnew;

`pattern bat.new(b:bat[:any_1,:any_2] ) :bat[:any_1,:any_2]`
address CMDBATnewDerived;

`pattern bat.new(b:bat[:any_1,:any_2], size:lng) :bat[:any_1,:any_2]`
address CMDBATnewDerived;

`command bat.new(nme:str):bat[:any_1,:any_2]`
address CMDBATderivedByName comment "Localize a bat by name and produce a variant";

`command bat.reduce(b:bat[:any_1,:any_2]):bat[:any_1,:any_2]`
address CMDBATreduce comment "Designate a BAT for which auxillary structures can be dropped";

`command bat.flush(b:bat[:any_1,:any_2]):void`
address CMDBATflush comment "Designate a BAT as not needed anymore";

`pattern bat.setGarbage(b:bat[:any_1,:any_2]):void`
address CMDBATsetGarbage comment "Designate a BAT as garbage";

`pattern bat.unpack(b:bat[:any_1,:any_2])(h:any_1,t:any_2)`
address CMDbatunpack comment "Extract the first tuple from a bat";

`pattern bat.pack(h:any_1,t:any_2):bat[:any_1,:any_2]`
address CMDbatpack comment "Pack a pair into a BAT";

`pattern bat.setBase(b:bat[:any_1,:any_2],c:bat[:any_1,:any_2]...):void`
address CMDsetBase comment "Give the non-empty BATs consecutive oid bases";

10.4 BAT Buffer Pool

The BBP module implements a box interface over the BAT buffer pool. It is primarilly meant to ease inspection of the BAT collection managed by the server.

The two predominant approaches to use bbp is to access the BBP with either *bind* or *take*. The former merely maps the BAT name to the object in the bat buffer pool. A more controlled scheme is to *deposit*, *take*, *release* and *discard* elements. Any BAT B created can be brought under this scheme with the name N. The association N->B is only maintained in the box administration and not reflected in the BAT descriptor. In particular, taking a BATobject out of the box leads to a private copy to isolate the user from concurrent updates on the underlying store. Upon releasing it, the updates are merged with the master copy [todo].

The remainder of this module contains operations that rely on the MAL runtime setting, but logically belong to the kernel/bat module.

module bbp;

`command open():void`
address CMDbbpopen comment "Locate the bbp box and open it.";

`command close():void`
address CMDbbpclose comment "Close the bbp box.";

`command destroy():void`
address CMDbbpdestroy comment "Destroy the box";

`pattern take(name:str) :bat[:any_1,:any_2]`
address CMDbbptake comment "Load a particular bat";

`pattern deposit(name:str,v:bat[:any_1,:any_2]) :void`
address CMDbbpdeposit comment "Enter a new bat into the bbp box.";

`pattern deposit(name:str,loc:str) :bat[:any_1,:any_2]`
address CMDbbpbindDefinition comment "Relate a logical name to a physical BAT in the buffer pool.";

`pattern commit():void`
address CMDbbpReleaseAll comment "Commit updates for this client";

`pattern releaseAll():void`
address CMDbbpReleaseAll comment "Commit updates for this client";

`pattern release(name:str,val:bat[:any_1,:any_2]) :void`
address CMDbbprelease comment "Commit updates and release this BAT.";

`pattern release(b:bat[:any_1,:any_2]):void`
address CMDbbpreleaseBAT comment "Remove the BAT from further consideration";

`pattern destroy(b:bat[:any_1,:any_2]):void`
address CMDbbpdestroyBAT1 comment "BAT removal at session end";

`pattern destroy(b:bat[:any_1,:any_2],immediate:bit)`
address CMDbbpdestroyBAT comment "Schedule a BAT for removal at session end or immediately";

`pattern toString(name:str):str`
address CMDbbptoStr comment "Get the string representation of an element in the box";

`pattern discard(name:str):void`
address CMDbbpdiscard comment "Remove the BAT from the box";

`pattern iterator(nme:str):lng`
address CMDbbpiterator comment "Locates the next element in the box";

`pattern prelude():void`
address CMDbbpprelude comment "Initialize the bbp box";

`pattern bind(name:str):bat[:any_1,:any_2]`
address CMDbbpbind comment "Locate the BAT using its logical name";

`command find(name:str):bat[:any_1,:any_2]`
address CMDbbpfind3 comment "Locate the BAT using its logical name in the BAT buffer pool");

`command find(head:str,tail:str):bat[:any_1,:any_2]`
address CMDbbpfind2 comment "Locate the BAT using the head and tail names in the BAT buffer pool");

`command find(idx:BAT):bat[:any_1,:any_2]`
address CMDbbpbindindex comment "Locate the BAT using its BBP index in the BAT buffer pool";

`pattern getObjects():bat[:int,:str]`
address CMDbbpGetObjects comment "View of the box content.";

`command getHeadType() :bat[:int,:str]`
address CMDbbpHeadType comment "Map a BAT into its head type";

`command getTailType() :bat[:int,:str]`
address CMDbbpTailType comment "Map a BAT into its tail type";

`command getNames() :bat[:int,:str]`
address CMDbbpNames comment "Map BAT into its bbp name";

`command getRNames() :bat[:int,:str]`
address CMDbbpRNames comment "Map a BAT into its bbp physical name";

`command getName( b:bat[:any_1,:any_2]):str`
address CMDbbpName comment "Map a BAT into its internal name";

`command getCount() :bat[:int,:lng]`
address CMDbbpCount comment "Create a BAT with the cardinalities of all known BATs";

`command getRefCount() :bat[:int,:int]`
address CMDbbpRefCount comment "Create a BAT with the (hard) reference counts";

`command getLRefCount() :bat[:int,:int]`
address CMDbbpLRefCount comment "Create a BAT with the logical reference counts";

`command getLocation() :bat[:int,:str]`
address CMDbbpLocation comment "Create a BAT with their disk locations";

`command getHeat() :bat[:int,:int]`
address CMDbbpHeat comment "Create a BAT with the heat values";

`command getDirty() :bat[:int,:str]`
address CMDbbpDirty comment "Create a BAT with the dirty/ diffs/clean status";

`command getStatus() :bat[:int,:str]`
address CMDbbpStatus comment "Create a BAT with the disk/load status";

`command getKind():bat[:int,:str]`
address CMDbbpKind comment "Create a BAT with the persistency status";

`command getRefCount(b:bat[:any_1,:any_2]) :int`
address CMDgetBATrefcnt comment "Utility for debugging MAL interpreter";

`command getLRefCount(b:bat[:any_1,:any_2]) :int`
address CMDgetBATlrefcnt comment "Utility for debugging MAL interpreter";

`command getDiskSpace() :int`
address CMDbbpDiskSpace comment "Estimate the amount of diskspace occupied by dbfarm";

10.5 Constants

The `const` module provides a box abstraction store for global constants. Between sessions, the value of the constants is saved on disk in the form of a simple MAL program, which is scanned and made available by opening the box. A future implementation should provide transaction support over the box, which would permit multiple clients to exchange (scalar) information easily.

The default constant box is initialized with session variables, such as 'user','dbname', 'dbfarm', and 'dbdir'. These actions are encapsulated in the prelude routine called.

A box should be opened before being used. It is typically used to set-up the list of current users and to perform authorization. The constant box is protected with a simple authorization scheme, prohibiting all updates unless issued by the system administrator.

module const;

`pattern open():void`
address CSTopen comment "Locate and open the constant box";

`pattern close():void`
address CSTclose comment "Close the constant box ";

`pattern destroy():void`
address CSTdestroy comment "Destroy the box";

`pattern take(name:str):any_1`
address CSTtake comment "Take a variable out of the box";

`pattern deposit(name:str,val:any_1) :void`
address CSTdeposit comment "Enter a new variable into the box";

`pattern releaseAll():void`
address CSTreleaseAll comment "Release all variables in the box";

`pattern release(name:str) :void`
address CSTrelease comment "Release a new constant value";

`pattern release(name:any_1):void`
address CSTrelease comment "Release a new constant value";

`pattern toString(name:any_1):str`
address CSTtoString comment "Get the string representation of an element in the box";

`pattern discard(name:any_1) :void`
address CSTdiscard comment "Release the const from the box";

`pattern newIterator()(:lng,:str)`
address CSTnewIterator comment "Locate next element in the box";

`pattern hasMoreElements()(:lng,:str)`
address CSThasMoreElements comment "Locate next element in the box";

10.6 BAT Iterators

Many low level algorithms rely on an iterator to break a collection into smaller pieces. Each piece is subsequently processed by a block.

For very large BATs it may make sense to break it into chunks and process them separately to solve a query. An iterator pair is provided to chop a BAT into fixed size elements. Each chunk is made available as a BATview. It provides read-only access to an underlying BAT. Adjusting the bounds is cheap, once the BATview descriptor has been constructed.

The smallest granularity is a single BUN, which can be used to realize an iterator over the individual BAT elements. For larger sized chunks, the operators return a BATview.

All iterators require storage space to administer the location of the next element. The BAT iterator module uses a simple lng variable, which also acts as a cursor for barrier statements.

The larger chunks produced are currently static, i.e. their size is a parameter of the call. Dynamic chunk sizes are interesting for time-series query processing. (See another module)

`command bat.newIterator(b:bat[:any_1,:any_2], size:lng)`
(:lng,:bat[:any_1,:any_2]) address CHPnewChunkIterator comment "Create an iterator with fixed granule size. The result is a view.";

`command bat.hasMoreElements(b:bat[:any_1,:any_2], size:lng)`
(:lng, :bat[:any_1,:any_2]) address CHPhasMoreChunks comment "Produce the next chunk for processing.";

`pattern bat.newIterator(b:bat[:any_1,:any_2]) (:lng, h:any_1, t:any_2)`
address CHPbunIterator comment "Process the buns one by one extracted from a void table.";

`pattern bat.newIterator(b:bat[:any_1,:bat]) (:lng, h:any_1, t:any_2)`
address CHPbunIterator comment "Process the buns one by one extracted from a void table.";

`pattern bat.hasMoreElements(b:bat[:any_1,:any_2]) (:lng, h:any_1, t:any_2)`
address CHPbunHasMoreElements;

`pattern bat.hasMoreElements(b:bat[:oid,:any_2]) (:lng, h:oid, t:any_2)`
address CHPbunHasMoreElements comment "Produce the next bun for processing.";

`pattern bat.hasMoreElements(b:bat[:any_1,:bat]) (:lng, h:any_1, t:any_2)`
address CHPbunHasMoreElements comment "Produce the next bun for processing.";

The head and tail values can also be extracted using the cursor. It points to the first bun in the chunk under consideration. It is often more effective due to use the iterator with automatic extraction of head and tail value; the overhead involved is much less.

`pattern bat.getHead(b:bat[:any_1,:any],i:lng):any_1`
address CHPgetHead comment "return the BUN head value using the cursor.";

`pattern bat.getTail(b:bat[:any_2,:any_1],i:lng):any_1`
address CHPgetTail comment "return the BUN tail value using the cursor.";

10.7 Box definitions

This module shows the behavior of a simple box of objects. Objects are stored into the box using *deposit* and taken out with *take*. Once you are done, elements can be removed by name or reference using *discard*.

A box should be opened before being used. It is typically used to set-up the list of current users and to perform authorization.

module box;

`pattern open(nme:str):any_1`
address BOXopen comment "Locate the box and open it.";

`pattern close(bname:str):void`
address BOXclose comment "Close the box.";

`pattern destroy(bname:str):void`
address BOXdestroy comment "Destroy the box.";

`pattern take(bnme:str, vnme:str):any_1`
address BOXtake comment "Locates the typed value in the box.";

`pattern deposit(bname:str,name:str,v:any_1):void`
address BOXdeposit comment "Enter a new value into the box.";

`pattern releaseAll(bname:str) :void`
address BOXreleaseAll comment "Release all objects for this client.";

`pattern release(bname:str,nme:str,val:any_1):void`
address BOXrelease comment "Release the BAT from the client pool.";

`pattern toString(bname:str,name:str) :str`
address BOXtoString comment "Get the string representation of the i-th element in the box.";

`pattern discard(bname:str,name:str) :void`
address BOXdiscard comment "Release the BAT from the client pool.";

`pattern iterator(nme:str):lng`
address BOXiterator comment "Locates the next element in the box.";

`command getBoxNames():bat[:int,:str]`
address BOXgetBoxNames comment "Retrieve the names of all boxes.";

10.8 Client Management

Each online client is represented with an entry in the clients table. The client may inspect his record at run-time and partially change its properties. The administrator sees all client records and has the right to adjust global properties.

module clients;

#Operations for all clients

`command setListing(flag:int):int`
: address CLTsetListing comment "Turn on/off echo of MAL instructions: 2 - show mal instruction, 4 - show details of type resolutoin, 8 - show binding information.";

`command setHistory(s:str)`
: address CLTsetHistory comment "Designate console history file for readline.";

`command getId():int`
: address CLTgetClientId comment "Return a number that uniquely represents the current client.";

`command getInfo( ):bat[:str,:str]`
: address CLTInfo comment "Pseudo bat with client attributes.";

`command getScenario():str`
: address CLTgetScenario comment "Retrieve current scenario name.";

`command setScenario(msg:str):str`
: address CLTsetScenario comment "Switch to other scenario handler, return previous one.";

`command quit():void`
: address CLTquit comment "Terminate the server. This command can only be initiated from the console.";

`command exit():void`
: address CLTexit comment "Terminate the session for a single client using a soft error.";

#Administrator operations

`command getLogins( ):bat[:int,:str]`
: address CLTLogin comment "Pseudo bat of client login time.";

`command getLastCommand( ):bat[:int,:str]`
: address CLTLastCommand comment "Pseudo bat of client's last command time.";

`command getActions( ):bat[:int,:int]`
: address CLTActions comment "Pseudo bat of client's command counts.";

`command getTime( ):bat[:int,:lng]`
: address CLTTime comment "Pseudo bat of client's total time usage(in usec).";

`command getUsers( ):bat[:int,:str]`
: address CLTusers comment "Pseudo bat of users logged in.";

`command suspend(id:int):void`
: address CLTsuspend comment "Put a client process to sleep for some time. It will simple sleep for a second at a time, until the awake bit has been set in its descriptor";

`command wakeup(id:int):void`
: address CLTwakeup comment "Wakeup a client process";

`command shutdown(forced:bit):void`
: address CLTshutdown comment "Close all client connections. If forced=false the clients are moved into FINISHING mode, which means that the process stops at the next cycle of the scenario. If forced=true all client processes are immediately killed";

10.9 Factory management

The factory infrastructure can be inspected and steered with the commands provided here.

module factory;

`command getPlants()(mod:bat[:oid,:str], fcn:bat[:oid,:str])`
: address FCTgetPlants comment "Retrieve the names for all active factories";

`command getCaller():int`
: address FCTgetCaller comment "Retrieve the unique identity of the factory caller";

`command getOwners():bat[:oid,:str]`
: address FCTgetOwners comment "Retrieve the factory owners table";

`command getArrival():bat[:oid,:timestamp]`
: address FCTgetArrival comment "Retrieve the time stamp the last call was made";

`command getDeparture():bat[:oid,:timestamp]`
: address FCTgetDeparture comment "Retrieve the time stamp the last answer was returned";

`command shutdown(m:str, f:str):void`
: address FCTshutdown comment "Close a factory";

10.10 Inspection

This module introduces a series of commands that provide access to information stored within the interpreter data structures. In all cases, the pseudo BAT operation is returned that should be garbage collected after being used.

The main performance drain would be to use a pseudo BAT directly to successively access it components. This can be avoided by first assigning the pseudo BAT to a variable.

Inspection is organized for two purposes, for interactive use and as a bases for reflective code. The latter needs BATs to work with. The former merely dumps the result in a table for the front-ends to consume.

module inspect;

`command getDefinition(mod:str,fcn:str) :bat[:str,:str]`
address INSPECTgetDefinition comment "Returns a string representation of a specific function.";

`command getSignature(mod:str,fcn:str) :bat[:str,:str]`
address INSPECTgetSignature comment "Returns the function signature(s).";

`command getAddress(mod:str,fcn:str) :bat[:str,:str]`
address INSPECTgetAddress comment "Returns the function signature(s).";

`command getComment(mod:str,fcn:str) :bat[:str,:str]`
address INSPECTgetComment comment "Returns the function help information.";

`command getSource(mod:str,fcn:str):str`
address INSPECTgetSource comment "Return the original input for a function.";

`command getKind():bat[:oid,:str]`
address INSPECTgetkind comment "Obtain the instruction kind.";

`command getModule():bat[:oid,:str]`
address INSPECTgetAllModules comment "Obtain the function name.";

`command getFunction():bat[:oid,:str]`
address INSPECTgetAllFunctions comment "Obtain the function name.";

`command getSignatures():bat[:oid,:str]`
address INSPECTgetAllSignatures comment "Obtain the function signatures.";

`command getAddresses():bat[:oid,:str]`
address INSPECTgetAllAddresses comment "Obtain the function address.";

`pattern getSize():lng`
address INSPECTgetSize comment "Return the storage size for the current function (in bytes).";

`command getSize(mod:str):bat[:str,:lng]`
address INSPECTgetModuleSize comment "Return the storage size for a module (in bytes).";

`command getSize(mod:str,fcn:str):lng`
address INSPECTgetFunctionSize comment "Return the storage size for a function (in bytes).";

`pattern getType(v:bat[:any_1,:any_2]) (ht:str, tt:str)`
address INSPECTtypeName comment "Return the internal type of a variable (expression).";

`pattern getType(v:any_1) :str`
address INSPECTtypeName comment "Return the type of a variable (expression).";

`command getTypeName(v:int):str`
address INSPECTtypename comment "Get the type name associated with a type id.";

`pattern getTypeIndex(v:any_1):int`
address INSPECTtypeIndex comment "Return the type index of a variable.";

`command getAtomNames():bat[:int,:str]`
address INSPECTatom_names comment "Collect a BAT with the atom names.";

`command getAtomSuper():bat[:int,:str]`
address INSPECTatom_sup_names comment "Collect a BAT with the atom names.";

`command getAtomSizes():bat[:int,:int]`
address INSPECTatom_sizes comment "Collect a BAT with the atom sizes.";

`command getEnvironment():bat[:str,:str]`
address INSPECTgetEnvironment comment "Collect the environment variables.";

10.11 Input/Output module

The IO module provides simple ASCII-IO rendering options. It is modeled after the tuple formats, but does not attempt to outline the results. Instead, it is geared at speed, which also means that some functionality regarding the built-in types is duplicated from the atoms definitions.

A functional limited form of formatted printf is also provided. It accepts at most one variable. A more complete approach is the tablet module.

The commands to load and save a BAT from/to an ASCII dump are efficient, but work only for binary tables.

module io;

`command stdin():bstream`
address io_stdin comment "return the input stream to the database client";

`command stderr():streams`
address io_stderr comment "return the error stream for the database console";

`command stdout():streams`
address io_stdout comment "return the output stream for the database client";

`pattern print(val:any_1,lst:any...):void`
address IOprint_val comment "Print a MAL value tuple .";

`pattern print(b1:bat[:any_1,:any]...):void`
address IOtable comment "BATs are printed with '#' for legend lines, and the BUNs on seperate lines between brackets, containing each to comma separated values (head and tail). If multiple BATs are passed for printing, print() performs an implicit natural join, producing a multi attribute table.";

`pattern ftable( filep:streams, b1:bat[:any_1,:any], b:bat[:any_1,:any]... ):void`
address IOftable comment "Print an n-ary table to a file.";

`pattern print(order:int,b:bat[:any_1,:any], b2:bat[:any_1,:any]...):void`
address IOotable comment "The same as normal table print, but enforces to use the order of BAT number [1..argc] to do the printing.";

`pattern table(b1:bat[:any_1,:any], b2:bat[:any_1,:any]...):void`
address IOttable comment "Print an n-ary table. Like print, but does not print oid column";

`pattern table(order:int, b1:bat[:any_1,:any], b2:bat[:any_1,:any]...):void`
address IOtotable comment "Print an n-ary table.";

`pattern ftable(fp:streams, order:int, b1:bat[:any_1,:any], b:bat[:any_1,:any]...):void`
address IOfotable comment "Print an n-ary table to a file.";

`pattern print(val:any_1):void`
address IOprint_val comment "Print a MAL value tuple .";

`pattern print(val:bat[:any_1,:any_2]):void`
address IOprint_val comment "Print a MAL value tuple .";

`pattern prompt(val:any_1):void`
address IOprompt_val comment "Print a MAL value without brackets.";

`pattern printft(val:any_1):void`
address IOprint_ft comment "Select default format ";

`command printf(format:str,val:bit):void`
address IOprint_formatted_bit;

`command printf(format:str,val:chr):void`
address IOprint_formatted_chr;

`command printf(format:str,val:int):void`
address IOprint_formatted_int;

`command printf(format:str,val:sht):void`
address IOprint_formatted_sht;

`command printf(format:str,val:oid):void`
address IOprint_oid;

`command printf(format:str,val:lng):void`
address IOprint_formatted_lng;

`command printf(format:str,val:flt):void`
address IOprint_formatted_flt;

`command printf(format:str,val:dbl):void`
address IOprint_formatted_dbl;

`command printf(val:str):void`
address IOprint_str;

`command printf(format:str,val:str):void`
address IOprint_formatted_str;

`command export(b:bat[:any_1,:any_2], filepath:str):bit`
address IOexport comment "Export a BAT as ASCII to a file. If the 'filepath' is not absolute, it is put into the .../dbfarm/$DB directory. Success of failure is indicated.";

`command import(b:bat[:any_1,:any_2], filepath:str) :bat[:any_1,:any_2]`
address IOimport comment "Import a BAT from an ASCII dump. The new tuples are *inserted* into the parameter BAT. You have to create it! Its signature must match the dump, else parsing errors will occur and FALSE is returned.";

10.12 Language Extensions

Iterators over scalar ranges are often needed, also at the MAL level. The barrier and control primitives are sufficient to mimic them directly.

The modules located in the kernel directory should not rely on the MAL datastructures. That's why we have to deal with some bat operations here and delegate the signature to the proper module upon loading.

Running a script is typically used to initialize a context. Therefore we need access to the runtime context. For the call variants we have to determine an easy way to exchange the parameter/return values.

module language;

`command newRange(v:oid)(:bit,:oid)`
address RNGnewRange_oid;

`command newRange(v:sht)(:bit,:sht)`
address RNGnewRange_sht;

`command newRange(v:int)(:bit,:int)`
address RNGnewRange_int;

`command newRange(v:lng)(:bit,:lng)`
address RNGnewRange_lng;

`command newRange(v:flt)(:bit,:flt)`
address RNGnewRange_flt;

`command newRange(v:dbl)(:bit,:dbl)`
address RNGnewRange_dbl comment "This routine introduces an iterator over a scalar domain.";

`command nextElement(step:oid,last:oid)(:bit,:oid)`
address RNGnextElement_oid;

`command nextElement(step:sht,last:sht)(:bit,:sht)`
address RNGnextElement_sht;

`command nextElement(step:int,last:int)(:bit,:int)`
address RNGnextElement_int;

`command nextElement(step:lng,last:lng)(:bit,:lng)`
address RNGnextElement_lng;

`command nextElement(step:flt,last:flt)(:bit,:flt)`
address RNGnextElement_flt;

`command nextElement(step:dbl,last:dbl)(:bit,:dbl)`
address RNGnextElement_dbl comment "Advances the iterator with a fixed value until it becomes >= last.";

`command raise(msg:str) :str`
address CMDraise comment "Raise an exception labeled with a specific message.";

`command assert(v:bit,term:str):void`
address MALassertBit;

`command assert(v:sht,term:str):void`
address MALassertSht;

`command assert(v:int,term:str):void`
address MALassertInt;

`command assert(v:lng,term:str):void`
address MALassertLng;

`command assert(v:str,term:str):void`
address MALassertStr;

`command assert(v:oid,term:str):void`
address MALassertOid;

`pattern assert(v:any_1,pname:str,oper:str,val:any_2):void`
address MALassertTriple comment "Assertion test";

`pattern assertSpace(depth:int)`
address MALsafeguardStack comment "Ensures that the current call does not consume more then depth*vtop elements on the stack";

`pattern dataflow():int`
address MALstartDataflow comment "The current guarded block is executed using dataflow control ";

`command register(m:str,f:str,code:str,help:str):void`
address CMDregisterFunction comment"Compile the code string and register it as a MAL function";

`pattern call(s:str):void`
address CMDcallString comment "Evaluate a MAL string program";

`pattern call(s:bat[:oid,:str]):void`
address CMDcallBAT comment "Evaluate a program stored in a BAT";

`command source(f:str):void`
address CMDevalFile comment "Merge the instructions stored in the file with the current program";

10.13 MAL debugger interface

This module provides access to the functionality offered by the MonetDB debugger and interpreter status. It is primarilly used in interactive sessions to activate the debugger at a given point. Furthermore, the instructions provide the necessary handle to generate information for post-mortum analysis.

To enable ease of debugging and performance monitoring, the MAL interpreter comes with a hardwired gdb-like text-based debugger. A limited set of instructions can be included in the programs themselves, but beware that debugging has a global effect. Any concurrent user will be affected by breakpoints being set.

The prime scheme to inspect the MAL interpreter status is to use the MAL debugger directly. However, in case of automatic exception handling it helps to be able to obtain BAT versions of the critical information, such as stack frame table, stack trace, and the instruction(s) where an exception occurred. The inspection typically occurs in the exception handling part of the MAL block.

Beware, a large class of internal errors can not easily captured this way. For example, bus-errors and segmentation faults lead to premature termination of the process. Similar, creation of the post-mortum information may fail due to an inconsistent state or insufficient resources.

module mdb;

`pattern start():void`
address MDBstart comment "Start interactive debugger";

`pattern start(clientid:int):void`
address MDBstart comment "Start interactive debugger on a client";

`pattern start(mod:str,fcn:str):void`
address MDBstartFactory comment "Start interactive debugger on a running factory";

`pattern stop():void`
address MDBstop comment "Stop the interactive debugger";

`pattern inspect(mod:str,fcn:str):void`
address MDBinspect comment "Run the debugger on a specific function";

`pattern setTrace(b:bit):void`
address MDBsetTrace comment "Turn on/off tracing of current routine";

`pattern setTrace(b:str):void`
address MDBsetVarTrace comment "Turn on/off tracing of a variable ";

`pattern setCatch(b:bit):void`
address MDBsetCatch comment "Turn on/off catching exceptions";

`command setTimer(b:bit):void`
address MDBsetTimer comment "Turn on/off performance timer for debugger";

`command setBigfoot(b:bit):void`
address MDBsetBigfoot comment "Turn on/off memory foot print for debugger";

`command setFlow(b:bit):void`
address MDBsetFlow comment "Turn on/off memory flow debugger";

`command setMemory(b:bit):void`
address MDBsetMemory comment "Turn on/off memory statistics tracing.";

`command setIO(b:bit):void`
address MDBsetIO comment "Turn on/off io statistics tracing";

`command getDebug():int`
address MDBgetDebug comment "Get the kernel debugging bit-set. See the MonetDB configuration file for details";

`command setDebug(flg:str):int`
address MDBsetDebugStr comment "Set the kernel debugging bit-set and return its previous value. The recognized options are: threads, memory, properties, io, transactions, modules, algorithms, estimates, xproperties";

`command setDebug(flg:int):int`
address MDBsetDebug comment "Set the kernel debugging bit-set and return its previous value.";

`command getException(s:str):str`
address MDBgetExceptionVariable comment "Extract the variable name from the exception message";

`command getReason(s:str):str`
address MDBgetExceptionReason comment "Extract the reason from the exception message";

`command getContext(s:str):str`
address MDBgetExceptionContext comment "Extract the context string from the exception message";

`pattern list():void`
address MDBlist comment "Dump the current routine on standard out.";

`pattern listMapi():void`
address MDBlistMapi comment "Dump the current routine on standard out with Mapi prefix.";

`pattern list(M:str,F:str):void`
address MDBlist3 comment "Dump the routine M.F on standard out.";

`pattern List():void`
address MDBlistDetail comment "Dump the current routine on standard out.";

`pattern List(M:str,F:str):void`
address MDBlist3Detail comment "Dump the routine M.F on standard out.";

`pattern var():void`
address MDBvar comment "Dump the symboltable of current routine on standard out.";

`pattern var(M:str,F:str):void`
address MDBvar3 comment "Dump the symboltable of routine M.F on standard out.";

`command grab(id:int):void`
address MDBgrabClient comment "Stop and debug another client process.";

`pattern dot(M:str,F:str,s:str):void`
address MDBshowFlowGraph comment "Dump the data flow of the function M.F in a format recognizable by the command 'dot' on the file s";

`pattern getStackDepth():int`
address MDBStkDepth comment "Return the depth of the calling stack.";

`pattern getStackFrame(i:int):bat[:str,:str]`
address MDBgetStackFrameN;

`pattern getStackFrame():bat[:str,:str]`
address MDBgetStackFrame comment "Collect variable binding of current (n-th) stack frame.";

`pattern getStackTrace():bat[:void,:str]`
address MDBStkTrace;

`pattern dump()`
address MDBdump comment "Dump instruction, stacktrace, and stack";

`pattern getDefinition():bat[:void,:str]`
address MDBgetDefinition comment "Returns a string representation of the current function with typing information attached";

10.14 Manual Inspection

This module introduces a series of commands that provide access to the help information stored in the runtime environment.

The manual bulk operations ease offline inspection of all function definitions. It includes an XML organized file, because we expect external tools to massage it further for presentation.

module manual;

`command help(text:str)`
address MANUALhelp comment "Produces a list of all <module>.<function> that match the text pattern. The wildcard '*' can be used for <module> and <function>. Using the '(' asks for signature information and using ')' asks for the complete help record.";

`command search(text:str)`
address MANUALsearch comment "Search the manual for command descriptions that match the regular expression 'text'";

`command createXML(mod:str):void`
address MANUALcreate1 comment "Generate a synopsis of a module";

`command createXML():void`
    address MANUALcreate0 comment "Produces a XML-formatted manual over all modules loaded.";

`command section(mod:str):void`
    address MANUALcreateSection1 comment "Generate a synopsis of a module for the reference manual";

`command index():void`
    address MANUALcreateIndex comment "Produces an overview of all names grouped by module.";

`command summary():void`
    address MANUALcreateSummary comment "Produces a manual with help lines grouped by module.";

`command completion(pat:str):bat[:int,:str]`
    address MANUALcompletion comment "Produces the wordcompletion table.";

10.15 MonetDB server interface

This module contains a simple interface for setting up internet connections and to initialize a client session.

Clients may initialize a private listener to implement specific services. For example, in an OLTP environment it may make sense to have a listener for each transaction type, which simply parses a sequence of transaction parameters.

Authorization of access to the server is handled as part of the client record initialization phase.

The complete Mapi library is also available to setup communication with another Mserver. This library internally uses pointer handles, which we replace with an index in a locally maintained table. It provides a handle to easily detect havoc clients.

module mserver;

`command listen():int`
    address SERVERlisten_default comment "Start a Mapi server with the default settings.";

`command listen(port:int):int`
    address SERVERlisten_port comment "Start a Mapi server on the given port";

`command listen(port:int, maxusers:int):int`
    address SERVERlisten2 comment "Start a Mapi server";

`command listen(port:int, maxusers:int, cmd:str):int`
    address SERVERlisten3 comment "Start the Mapi listener on <port> for <maxusers>. For a new client connection MAL procedure <cmd>(Stream s_in, Stream s_out) is called.If no <cmd> is specified a new client thread is forked.";

`command listen_ssl(port:int, maxusers:int,`
    keyfile:str,certfile:str, cmd:str):int address SERVERlistenSSL comment "Start the Mapi listener on <port> for <maxusers> using SSL. <keyfile> and <certfile> give the path names for files with the server key and certificates in PEM format. For a new client connection MAL procedure <cmd>(Stream s_in, Stream s_out) is called. If no <cmd> is specified a new client thread is forked.";

`command stop():void`
    address SERVERstop comment "Terminate connection listeners";

`command suspend():void`
    address SERVERsuspend comment "Suspend accepting connections";

`command resume():void`
    address SERVERresume comment "Resume connection listeners";

`command malclient(in:streams, out:streams):void`
    address SERVERclient comment "Start a Mapi client.";

`command trace(mid:int,flag:int):void`
    address SERVERtrace comment "Toggle the Mapi library tracer";

`command reconnect(host:str, port:int, usr:str, passwd:str,lang:str):int`
    address SERVERreconnectWithoutAlias comment "Re-establish connection with a remote mserver";

`command reconnect(host:str, port:int, db_alias:str, usr:str, passwd:str,lang:str):int`
    address SERVERreconnectAlias comment "Re-establish connection with a remote mserver";

`command reconnect(mid:int):void`
    address SERVERreconnect comment "Re-establish a connection";

`command connect(host:str, port:int, usr:str, passwd:str,lang:str):int`
    address SERVERconnect comment "Establish connection with a remote mserver";

`command connect_ssl(host:str, port:int, usr:str, passwd:str,lang:str):int`
    address SERVERconnectssl comment "Establish connection with a remote mserver using the secure socket layer";

`command disconnect(dbalias:str):int`
    address SERVERdisconnectWithAlias comment "Close connection with a remote mserver";

`command disconnect():int`
    address SERVERdisconnectALL comment "Close connections with all remote mserver";

`command setAlias(dbalias:str)`
    address SERVERsetAlias comment "Give the channel a logical name";

`command lookup(dbalias:str):int`
    address SERVERlookup comment "Retrieve the connection identifier";

`command disconnect(mid:int):void`
    address SERVERdisconnect comment "Terminate the session";

`command destroy(mid:int):void`
address SERVERdestroy comment "Destroy the handle";

`command ping(mid:int):int`
address SERVERping comment "Test availability of server";

`command query(mid:int, qry:str):int`
address SERVERquery comment "Sent the query for execution";

`command query_handle(mid:int, qry:str):int`
address SERVERquery_handle comment "Sent the query for execution";

`pattern query_array(mid:int, qry:str, arg:str...):int`
address SERVERquery_array comment "Sent the query for execution replacing '?' by arguments";

`command prepare(mid:int, qry:str):int`
address SERVERprepare comment "Prepare a query for execution";

`command finish(hdl:int):int`
address SERVERfinish comment "Remove all remaining answers";

`command get_field_count(hdl:int):int`
address SERVERget_field_count comment "Return number of fields";

`command get_row_count(hdl:int):int`
address SERVERget_row_count comment "Return number of rows";

`command fetch_row(hdl:int):int`
address SERVERrows_affected comment "Return number of affected rows";

`command fetch_row(hdl:int):int`
address SERVERfetch_row comment "Retrieve the next row for analysis";

`command fetch_all_rows(hdl:int):int`
address SERVERfetch_all_rows comment "Retrieve all rows into the cache";

`command fetch_field(hdl:int,fnr:int):str`
address SERVERfetch_field_str comment "Retrieve a single field";

`command fetch_field(hdl:int,fnr:int):int`
address SERVERfetch_field_int comment "Retrieve a single int field";

`command fetch_field(hdl:int,fnr:int):lng`
address SERVERfetch_field_lng comment "Retrieve a single lng field";

`command fetch_field(hdl:int,fnr:int):sht`
address SERVERfetch_field_sht comment "Retrieve a single sht field";

`command fetch_field(hdl:int,fnr:int):void`
address SERVERfetch_field_void comment "Retrieve a single void field";

`command fetch_field(hdl:int,fnr:int):oid`
address SERVERfetch_field_oid comment "Retrieve a single void field";

`command fetch_field(hdl:int,fnr:int):chr`
address SERVERfetch_field_chr comment "Retrieve a single chr field";

`command fetch_field_array(hdl:int):bat[:int,:str]`
address SERVERfetch_field_bat comment "Retrieve all fields for a row";

`command fetch_line(hdl:int):str`
address SERVERfetch_line comment "Retrieve a complete line";

`command fetch_reset(hdl:int):int`
address SERVERfetch_reset comment "Reset the cache read line.";

`command next_result(hdl:int):int`
address SERVERnext_result comment "Go to next result set";

`command error(mid:int):int`
address SERVERerror comment "Check for an error in the communication";

`command getError(mid:int):str`
address SERVERgetError comment "Get error message";

`command explain(mid:int):str`
address SERVERexplain comment "Turn the error seen into a string";

`pattern put(mid:int, nme:str, val:any_1):void`
address SERVERput comment "Send a value to a remote site";

`pattern put(nme:str, val:any_1):str`
address SERVERputLocal comment "Prepare sending a value to a remote site";

`pattern rpc(key:int,qry:str...):any`
address SERVERmapi_rpc_single_row comment "Sent a simple query for execution and fetch result";

`pattern rpc(key:int,qry:str):bat[:any_1,:any_2]`
address SERVERmapi_rpc_bat;

`command rpc(key:int,qry:str):void`
address SERVERquery comment "Sent a simple query for execution";

`pattern bind(key:int,rschema:str,rtable:str,rcolumn:str,i:int):bat[:any_1,:any_2]`
address SERVERbindBAT comment "Bind a remote variable to a local one";

`pattern bind(key:int,rschema:str,rtable:str,i:int):bat[:any_1,:any_2]`
address SERVERbindBAT comment "Bind a remote variable to a local one";

`pattern bind(key:int,remoteName:str):bat[:any_1,:any_2]`
address SERVERbindBAT comment "Bind a remote variable to a local one";
mserver.listen();

10.16 Multiple association tables

A MAT is a convenient way to deal with scaleability. It combines the definitions of several, type compatible BATs under a single name. This view is only materialized when the operations can not deal with the components individually, or the incremental operation is not supported.

The module is supported by the mal_mergetable optimizer.

The primitives below are chosen to accomodate the SQL front-end to produce reasonable efficient code.

module mat;

`pattern new(b:bat[:any_1,:any_2]...):bat[:any_1,:any_2]`
address MATnew comment "Define a Merge Association Table (MAT) ";

`pattern pack(b:bat[:any_1,:any_2]...):bat[:any_1,:any_2]`
address MATpack comment "Materialize the MAT into the first BAT";

`pattern print(b:bat[:any_1,:any_2]...):void`
address MATprint;

`pattern newIterator(grp:bat[:any_1,:any_2]...):bat[:any_1,:any_2]`
address MATnewIterator comment "Create an iterator over a MAT";

`pattern hasMoreElements(grp:bat[:any_1,:any_2]...):bat[:any_1,:any_2]`
address MAThasMoreElements comment "Find the next element in the merge table";

`command info(g:str, e:str):bat[:any_1,:any_2]`
address MATinfo comment "retrieve the definition from the partition catalogue";

10.17 BAT Partition Manager

In real-life database applications the BATs tend to grow beyond the memory size. This leads to a heavy IO dominated behavior, which can partly be avoided by breaking up the query into a sequence of subqueries using a map-reduce strategy. The BAT partition manager (BPM) module is designed to support this strategy using range- and hash-partitioning.

Consider we want to reorganize R:bat[:oid,:int] into two partitions, based on splitting by tail value. The following MAL program illustrates the snippet of actions needed:

bpm.open(); Ralias:= bpm.deposit("myversion",R:bat[:oid,:int]); bpm.rangePartition(Ralias,nil:int,100); bpm.rangePartition(Ralias,101,200); bpm.close();

The command BPM.DEPOSIT registers a BAT as one for which a partitioned copy is required. The first partition call breaks the orginal BAT into two pieces: (nil:int,100) and (101,nil:int). The second call breaks the latter into (101,200) and (201,nil:int). The BAT partitions share the persistency properties. Partitioning on the head simple calls for a reverse operation on the source BAT first.

The partition manager also supports hash-based partitioning. Its argument is the number of hash bucket bits.

bpm.open(); Rev:= bat.reverse(R:bat[:oid,:int]); Ralias:= bpm.deposit("myHashVersion",Rev); # creates side effects bpm.hashPartition(Ralias,2); bpm.close();

This example creates a hash-partition based on the head.

The design is based on the assumption that partitions are reasonably large. This helps to limit plan explosion. (or a scheduler should step in)

10.17.1 Derived partitioning

A relational front-end would benefit from derived horizontal fragmentation. It would enable grouping together related fragments on the same site. Assume a relation R(A,B) which is already partitioned on A the derived fragmentation on the head is enforced with

bpm.derivePartition(B,A);

10.17.2 Using partitions

The partitioned BAT can be used in two ways. A query plan can be rewritten into a generator over the partitions, or it can be used by optimizers to derived all subqueries first for symbolic evaluation.

The former is illustrated with the snippet to select part of a partitioned BAT. In this example we collect the partial results in the accumulator BAT U.

bpm.open(); Ralias:bat[:oid,:int]:= bpm.take("myversion"); U:= bat.new(:oid,:int); barrier Rp:= bpm.newIterator(Ralias); ... t:= algebra.select(Rp,0,100); U:= algebra.union(tu,t); ... redo Rp:= bpm.hasMoreElements(Ralias); exit Rp; bpm.close();

The properties of the partitioned BATs are particularly useful during query optimization. However, it only works if the BAT identifier can be determined at compile time. For SQL it can be simply looked up in the catalog as part of a preparatory optimizer step.

To illustrate, the same problem handled by an optimizer that produces the plan based on a known number of partitions:

bpm.open(); R:bat[:oid,:int]:= bpm.take("myversion"); # get the partition alias optimizer.mergetable(); T:= algebra.select(R,0,100);

is translated to the plan:

bpm.open(); R:bat[:oid,:int]:= bpm.take("myversion"); # get the partition alias R0:bat[:oid,:int]:= bpm.take(R,0, nil:oid,nil:oid, 0,100); R1:bat[:oid,:int]:= bpm.take(R,1, nil:oid,nil:oid, 101,200); R2:bat[:oid,:int]:= bpm.take(R,2, nil:oid,nil:oid, 201,nil:int); R:= mat.new(R0,R1,R2); T:= algebra.select(R,0,100); optimizer.multitable();

In this translation Ri also gets the properties of the BATs. It is now up to the MAT optimizer to decide about further plan expansion or an iterator approach.

10.17.3 Partition updates

The content of the partitions is preferrable updated in bulk. This calls for accumulation of insertions/deletions in pending update BATs, as already performed in the SQL code generator. Once the transaction is commited, the updates are propagated (in parallel) to all partitions.

bpm.open(); Ralias:bat[:oid,:int] := bpm.take("myversion"); bpm.insert(Ralias, Rinsert); # handle pending inserts bpm.delete(Ralias, Rdelete); # handle pending deletes bpm.replace(Ralias, Rold, Rnew); # handle pending updates bpm.close();

The REPLACE operator works on the assumption that the head of ROLD and RNEW is unique.

It remains possible to retrieve a partition and directly insert elements, but then it is up to the compiler to ensure that the boundery conditions are met.

10.17.4 Partitioned results

In many situations, you would like to keep the partial results as a partitioned BAT again. The easiest solution is to create a partitioned BAT, whose partitions are empty. Subsequently, we insert the temporary results. Depending on the fragmentation criteria, pieces may align with the pieces known, or lead to a redistribution of the buns to the correct bats.

In the previous plan for this becomes

> bpm.open(); Tmp := bpm.deposit("tmp",:bat[:oid,:int]); bpm.rangePartition(tmp,nil:int,100); bpm.rangePartition(tmp,101,nil:int);
>
> Ralias:bat[:oid,:int]:= bpm.take("myversion"); # get the partition alias R0:bat[:oid,:int]:= bpm.take("myversion", nil:oid,nil:oid, 0,100); T0:= algebra.select(R0,0,100); bpm.insert(Tmp,T0);
>
> R1:bat[:oid,:int]:= bpm.take("myversion", nil:oid,nil:oid, 101,200); T1:= algebra.select(R1,0,100); bpm.insert(Tmp,T1);
>
> R2:bat[:oid,:int]:= bpm.take("myversion", nil:oid,nil:oid, 201,nil:int); T2:= algebra.select(R2,0,100); bpm.insert(Tmp,T2);

Note that a symbolic optimizer can reduce this plan to a small snippet.

The rationale for the update approach is that re-distribution of temporary results are hidden behind the BPM.INSERT() interface. The only decision that should be taken by the optimizer is the fragmentation criteria for the temporary results.

For temporary results the range bounds need not be stored in the BPM catalog. Instead, the mat approach could be used to reduce the plan size.

> bpm.open(); Ralias:= bpm.take("myversion",:bat[:oid,:int]); # get the partition alias R0hlow=nil:oid,hhigh=nil:oid,tlow=nil:int,thigh=100:= bpm.take(Ralias, 0); T0:=algebra.select(R0,0,100);
>
> R1hlow=nil:oid,hhigh=nil:oid,tlow=101,thigh=200:= bpm.take(Ralias, 1); T1:= algebra.select(R1,0,100);
>
> R2hlow=nil:oid,hhigh=nil:oid,tlow=201,thigh=nil:int:= bpm.take(Ralias, 2); R:= mat.new(T0,T1,T2); T2:=algebra.select(R2,0,100);

10.17.5 Partition iterators

The default strategy for an optimizer is to replace a reference to a partitioned BAT by an iterator.

> l:= bpm.new(); barrier Elm:bat[:oid,:int]:= bpm.newIterator(Ralias); t:= algebra.select(Elm,0,20); bpm.addPartition(l,t); redo Elm:bat[:oid,:int]:= bpm.newIterator(Ralias); exit Elm;

Variations on this theme are iterators that search for partitions overlapping a range or those that are not empty.

10.17.6 Partition selection

Partition aware relational operators further reduce the performance overhead and at the same time avoid cluttering the MAL plans with too much flow of control constructs. A few operators relevant for the SQL environment will be added.

The select operation can be overloaded in the BPM to improve processing further. For example, the operation

> t := bpm.select(Ralias,0,100);

extracts portions of all three partitions and creates a non-partitioned result BAT. If the partition bounds align with the selection criteria this operation becomes cheap. It can be used to convey information on the bounds to optimizers.

The lifetime of a partitioned table is inherited from its components. How to detect that a temporary BAT is removed from the BBP? Currently we have to explicitly call the bpm.garbage() on those partitioned BATs.

At the end of a query plan we have to garbage collect any of the left-over partitioned temporary tables.

10.18 Performance profiler

A key issue in developing fast programs using the Monet database back-end requires a keen eye on where performance is lost. Although performance tracking and measurements are highly application dependent, a simple to use tool makes life a lot easier.

Activation of the performance monitor has a global effect, i.e. all concurrent actions on the kernel are traced, but the events are only sent to the client initiated the profiler thread.

10.18.1 Monet Event Logger

The Monet Event Logger generates records of each event of interest indicated by a log filter, i.e. a pattern over module and function names.

The log record contents is derived from counters being (de-)activated. A complete list of recognized counters is shown below.

10.18.2 Execution tracing

Tracing is a special kind of profiling, where the information gathered is not sent to a remote system, but stored in the database itself. Each profile event is given a separate BAT

```
# thread and time since start
profiler.activate("tick");
# cpu time in nano-seconds
profiler.activate("cpu");
# memory allocation information
profiler.activate("memory");
# IO activity
profiler.activate("io");
# Module,function,program counter
profiler.activate("pc");
# actual MAL instruction executed
```

`profiler.activate("statement");`

The profiler event can be handled in several ways. The default strategy is to ship the event record immediately over a stream to a performance monitor. An alternative strategy is preparation of off-line performance analysis.

To reduce the interference of performance measurement with the experiments, the user can use an event cache, which is emptied explicitly upon need.

module profiler;

`command activate(name:str):void`
address CMDactivateProfiler comment "Make the specified counter active.";

`command deactivate(name:str):void`
address CMDdeactivateProfiler comment "Deactivate the counter";

`command openStream():void`
address CMDopenProfilerStream comment "Sent the events to output stream";

`command openStream(fnme:str):void`
address CMDsetProfilerFile comment "Send the log events to a file ";

`command openStream(host:str, port:int):void`
address CMDsetProfilerStream comment "Send the log events to a stream ";

`command closeStream():void`
address CMDcloseProfilerStream comment "Stop sending the event records";

`command setAll():void`
address CMDsetAllProfiler comment "Short cut for setFilter(*,*).";

`command setNone():void`
address CMDsetNoneProfiler comment "Short cut for clrFilter(*,*).";

`command setFilter(mod:str,fcn:str):void`
address CMDsetFilterProfiler comment "Generate an event record for all function calls that satisfy the regular expression mod.fcn. A wildcard (*) can be used as name to identify all";

`pattern setFilter(v:any):void`
address CMDsetFilterVariable comment "Generate an event record for every instruction where v is used;"

`command clrFilter(mod:str,fcn:str):void`
address CMDclrFilterProfiler comment "Clear the performance trace bit of the selected functions.";

`pattern clrFilter(v:any):void`
address CMDsetFilterVariable comment "Stop tracing the variable" ;

`command setStartPoint(mod:str,fcn:str):void`
address CMDstartPointProfiler comment "Start performance tracing at mod.fcn";

`command setEndPoint(mod:str,fcn:str)`
address CMDendPointProfiler comment "End performance tracing after mod.fcn";

`command start():void`
address CMDstartProfiler comment "Start performance tracing";

`command noop():void`
address CMDnoopProfiler comment "Fetch any pending performance events";

`command stop():void`
address CMDstopProfiler comment "Stop performance tracing";

`command reset():void`
address CMDclearTrace comment "Clear the profiler traces";

`command dumpTrace():void`
address CMDdumpTrace comment "List the events collected";

`command getTrace(e:str):bat[:int,:any_1]`
address CMDgetTrace comment "Get the trace details of a specific event";

`pattern getEvent()(:lng, :int,:int)`
address CMDgetEvent comment "Retrieve the performance indicators of the previous instruction";

`command cleanup():void`
address CMDcleanup comment "Remove the temporary tables for profiling";

10.19 PCRE library interface

The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl, with just a few differences. The current implementation of PCRE (release 4.x) corresponds approximately with Perl 5.8, including support for UTF-8 encoded strings. However, this support has to be explicitly enabled; it is not the default.

10.20 Statistics box.

Most optimizers need easy access to key information for proper plan generation. Amongst others, this volatile information consists of the tuple count, size, min- and max-value, the null-density, and a histogram of the value distribution.

The statistics are management by a Box, which gives a controlled environment to manage a collection of BATs and system variables.

BATs have to be deposit into the statistics box separately, because the costs attached maintaining them are high. The consistency of the statistics box is partly the responsibility of the upper layers. There is no automatic triggering when the BATs referenced are heavily modified or are being destroyed. They disappear from the statistics box the first time an invalid access is attempted or during system reboot.

The staleness of the information can be controlled in several ways. The easiest, and most expensive, is to assure that the statistics are updated when you start the server. Alternative, you can set a expiration interval, which will update the information only when it is considered expired. This test will be triggered either at server restart or your explicit call to update the statistics tables. The statistics table is commited each time you change it.

A forced update can be called upon when the front-end expects the situation to be changed drastically.

The statistics table is mostly used internally, but once in a while you need a dump for closed inspection. in your MAL program for inspection. Just use the BBP bind operation to locate them in the buffer pool.

module statistics;

`pattern open():void`
address STATopen comment "Locate and open the statistics box";

`pattern close():void`
address STATclose comment "Close the statistics box ";

`pattern destroy():void`
address STATdestroy comment "Destroy the statistics box";

`pattern take(name:any_1):any_2`
address STATtake comment "Take a variable out of the statistics box";

`command deposit(name:str) :void`
address STATdepositStr comment "Enter a new BAT into the statistics box";

`command deposit(name:bat[:any_1,:any_2]) :void`
address STATdeposit comment "Enter a new BAT into the statistics box";

`pattern releaseAll():void`
address STATreleaseAll comment "Release all variables in the box";

`pattern release(name:str) :void`
address STATreleaseStr comment "Release a single BAT from the box";

`command release(name:bat[:any_1,:any_2]):void`
address STATrelease comment "Release a single BAT from the box";

`pattern toString(name:any_1):str`
address STATtoString comment "Get the string representation of an element in the box";

`pattern discard(name:str) :void`
address STATdiscard comment "Release a BAT by name from the box";

`command discard(name:bat[:any_1,:any_2]) :void`
address STATdiscard2 comment "Release a BAT variable from the box";

`pattern newIterator()(:lng,:str)`
address STATnewIterator comment "Locate next element in the box";

`pattern hasMoreElements()(:lng,:str)`
address STAThasMoreElements comment "Locate next element in the box";

`command update()`
address STATupdate comment "Check for stale information";

`command forceUpdate()`
address STATforceUpdateAll comment "Bring all information up to date";

`command forceUpdate(bnme:str)`
address STATforceUpdate comment "Bring the statistics up to date for one BAT";

`command prelude() :void`
address STATprelude comment "Initialize the statistics package";

`command epilogue() :void`
address STATepilogue comment "Release the resources of the statistics package";

`command dump() :void`
address STATdump comment "Display the statistics table";

`command getObjects():bat[:int,:str]`
address STATgetObjects comment "Return a table with BAT names managed";

`command getHotset():bat[:int,:str]`
address STATgetHotset comment "Return a table with BAT names that have been touched since the start of the session";

`command getCount(nme:str):lng`
address STATgetCount comment "Return latest stored count information";

`command getSize(nme:str):lng`
address STATgetSize comment "Return latest stored count information";

`command getMin(nme:str):lng`
address STATgetMin comment "Return latest stored minimum information";

`command getMax(nme:str):lng`
address STATgetMax comment "Return latest stored maximum information";

`command getHistogram(nme:str):bat[:any_1,:any_2]`
address STATgetHistogram comment "Return the latest histogram");

10.21 The table interface

A database cannot live without ASCII tabular print/dump/load operations. It is needed to produce reasonable listings, to exchange answers with a client, and to keep a database version for backup. This is precisely where the tablet module comes in handy. [This module should replace all other table dump/load functions]

We start with a simple example to illustrate the plain ASCII representation and the features provided. Consider the relational table answer(name:str, age:int, sex:chr, address:str, dob:date) obtained by calling the routine tablet.page(B1,...,Bn) where the Bi represent BATS.

```
[ "John Doe",         25,    'M',    "Parklane 5",   "25-12-1978" ]
[ "Maril Streep",     23,    'F',    "Church 5",     "12-07-1980" ]
[ "Mr. Smith",        53,    'M',    "Church 1",     "03-01-1950" ]
```

The lines contain the representation of a list in Monet tuple format. This format has been chosen to ease parsing by any front-end. The scalar values are represented according to their type. For visual display, the columns are aligned by placing enough tabs between columns based on sampling the underlying bat to determine a maximal column width. (Note,actual commas are superfluous).

The arguments to the command can be any sequence of BATs, but which are assumed to be aligned. That is, they all should have the same number of tuples and the j-th tuple tail of Bi is printed along-side the j-th tuple tail of Bi+1.

Printing both columns of a single bat is handled by tablet as a print of two columns. This slight inconvenience is catch-ed by the io.print(b) command, which resolves most back-ward compatibility issues.

In many cases, this output would suffice for communication with a front-end. However, for visual inspection the user should be provided also some meta information derived from the database schema. Likewise, when reading a table this information is needed to prepare a first approximation of the schema namings. This information is produced by the command tablet.header(B1,...,Bn), which lists the column role name. If no role name is give, a default is generated based on the BAT name, e.g. B1_tail.

```
#------------------------------------------------------#
# name,           age, sex, address,       dob         #
#------------------------------------------------------#
[ "John Doe",      25, ’M’, "Parklane 5", "25-12-1978" ]
[ "Maril Streep",  23, ’F’, "Church 5",   "12-07-1980" ]
[ "Mr. Smith",     53, ’M’, "Church 1",   "03-01-1950" ]
```

The command tablet.display(B1,...,Bn) is a contraction of tablet.header(); tablet.page().

In many cases, the `tablet` produced may be too long to consume completely by the front end. In that case, the user needs page size control, much like the more/less utilities under Linux. However, no guarantee is given for arbitrarily going back and forth. [but works as long as we materialize results first]. A portion of the tablet can be printed by identifying the rows of interest as the first parameter(s) in the page command, e.g.

```
tablet.page(nil,10,B1,...,Bn);  #prints first 10 rows
tablet.page(10,20,B1,...,Bn);   #prints next 10 rows
tablet.page(100,nil,B1,...,Bn); #starts printing at tuple 100 until end
```

A paging system also provides the commands tablet.firstPage(), tablet.nextPage(), tablet.prevPage(), and tablet.lastPage() using a user controlled tablet size tablet.setPagesize(L).

The tablet display operations use a client (thread) specific formatting structure. This structure is initialized using either tablet.setFormat(B1,...,Bn) or tablet.setFormat(S1,...,Sn) (Bi is a BAT, Si a scalar). Subsequently, some additional properties can be set/modified, column width and brackets. After printing/paging the BAT resources should be freed using the command tablet.finish().

Any access outside the page-range leads to removal of the report structure. Subsequent access will generate an error. To illustrate, the following code fragment would be generated by the SQL compiler

```
        tablet.setFormat(B1,B2);
        tablet.setDelimiters("|","\t","|\n");
        tablet.setName(0, "Name");
        tablet.setNull(0, "?");
```

```
        tablet.setWidth(0, 15);
        tablet.setBracket(0, " ", ",");
        tablet.setName(1, "Age");
        tablet.setNull(1, "-");
        tablet.setDecimal(1, 9,2);
        tablet.SQLtitle("Query: select * from tables");
        tablet.page();
        tablet.SQLfooter(count(B1),cpuTicks);
```

This table is printed with tab separator(s) between elements and the bar (|) to mark begin and end of the string. The column parameters give a new title, a null replacement value, and the preferred column width. Each column value is optionally surrounded by brackets. Note, scale and precision can be applied to integer values only. A negative scale leads to a right adjusted value.

The title and footer operations are SQL specific routines to decorate the output.

Another example involves printing a two column table in XML format. [Alternative, tablet.XMLformat(B1,B2) is a shorthand for the following:]

```
        tablet.setFormat(B1,B2);
        tablet.setTableBracket("<rowset>","</rowset>");
        tablet.setRowBracket("<row>","</row>");
        tablet.setBracket(0, "<name>", "</name>");
        tablet.setBracket(1, "<age>", "</age>");
        tablet.page();
```

10.21.1 Tablet properties

More detailed header information can be obtained with the command tablet.setProperties(S), where S is a comma separated list of properties of interest, followed by the tablet.header(). The properties to choose from are: bat, name, type, width, sorted, dense, key, base, min, max, card,....

```
#--------------------------------------#
# B1,   B2,    B3,    B4,      B5    # BAT
# str,  int,   chr,   str,     date  # type
# true, false, false, false,   false # sorted
# true, true,  false, false,   false # key
# ,     23,    ’F’,   ,              # min
# ,     53,    ’M’,   ,              # max
# 4,     4,    4,     4,       4     # count
# 4,i    3,    2,     2,       3     # card
# name, age,   sex,   address, dob   # name
#--------------------------------------#
```

10.21.2 Scalar tablets

In line with the 10-year experience of Monet, printing scalar values follow the tuple layout structure. This means that the header() command is also applicable. For example, the sequence "i:=0.2;v:=sin(i); tablet.display(i,v);" produces the answer:

```
#----------------#
```

```
# i,    v        #
#----------------#
[ 0.2,  0.198669 ]
#----------------#
```

All other formatted printing should be done with the printf() operations contained in the module IO.

10.21.3 Tablet dump/restore

Dump and restore operations are abstractions over sequence of tablet commands. The command tablet.dump(stream,B1,...,Bn) is a contraction of the sequence tablet.setStream(stream); tablet.setProperties("name,type,dense,sorted,key,min,max"); tablet.header(B1,..,Bn); tablet.page(B1,..,Bn). The result can be read by tablet.load(stream,B1,..,Bn) command. If loading is successful, e.g. no parsing errors occurred, the tuples are appended to the corresponding BATs.

10.21.4 Front-end extension

A general bulk loading of foreign tables, e.g. CSV-files and fixed position records, is not provided. Instead, we extend the list upon need. Currently, the routines tablet.SQLload(stream,delim1,delim2, B1,..,Bn) reads the files using the Oracle(?) storage. The counterpart for dumping is tablet.SQLdump(stream,delim1,delim2);

10.21.5 The commands

The load operation is for bulk loading a table, each column will be loaded into its own bat. The arguments are void-aligned bats describing the input, ie the name of the column, the tuple separator and the type. The nr argument can be -1 (The input (datafile) is read until the end) or a maximum.

The dump operation is for dumping a set of bats, which are aligned. Again with void-aligned arguments, with name (currently not used), tuple separator (the last is the record separator) and bat to be dumped. With the nr argument the dump can be limited (-1 for unlimited).

The output operation is for ordered output. A bat (possibly form the collection) gives the order. For each element in the order bat the values in the bats are searched, if all are found they are output in the datafile, with the given separators.

The scripts from the tablet.mil file are all there too for backward compatibility with the old Mload format files.

The load_format loads the format file, since the old format file was in a table format it can be loaded with the load command.

The result from load_format can be used with load_data to load the data into a set of new bats.

These bats can be made persistent with the make_persistent script or merge with existing bats with the merge_data script.

The dump_format scripts dump a format file for a given set of to be dumped bats. These bats can be dumped with dump_data.

# the major tablet shuffling routines module tablet;

`command load( names:bat[:oid,:str], seps:bat[:oid,:str],`
types:bat[:oid,:str], datafile:str, nr:int) :bat[:str,:bat] address CMDtablet_load comment "Load a bat using specific format.";

`command input( names:bat[:oid,:str], seps:bat[:oid,:str],`
types:bat[:oid,:str], s:streams, nr:int) :bat[:str,:bat] address CMDtablet_input comment "Load a bat using specific format.";

`command dump(names:bat[:oid,:str], seps:bat[:oid,:str],`
bats:bat[:oid,:bat], datafile:str, nr:int) :void address CMDtablet_dump comment "Dump the bat in ASCII format";

`command output(order:bat[:any_1,:any_2], seps:bat[:oid,:str],`
bats:bat[:oid,:bat], s:streams) :void address CMDtablet_output comment "Send the bat to an output stream.";

`pattern display(v:any...):int`
address TABdisplayRow comment "Display a formatted row";

`pattern display(v:bat[:any_1,:any]...):int`
address TABdisplayTable comment "Display a formatted table";

`pattern page(b:bat[:any_1,:any]...):int`
address TABpage comment "Display all pages at once without header";

`pattern header(b:any...):int`
address TABheader comment "Display the minimal header for the table";

`command setProperties(prop:str):int`
address TABsetProperties comment "Define the set of properties";

`pattern dump(s:streams,b:bat[:any,:any]...):int`
address TABdump comment "Print all pages with header to a stream";

`pattern setFormat(b:any...):void`
address TABsetFormat comment "Initialize a new reporting structure.";

`command finish():void`
address TABfinishReport comment "Free the storage space of the report descriptor";

`command setStream(s:streams):void`
address TABsetStream comment "Redirect the output to a stream.";

`command setPivot(b:bat[:void,:oid]) :void`
address TABsetPivot comment "The pivot bat identifies the tuples of interest. The only requirement is that all keys mentioned in the pivot tail exist in all BAT parameters of the print comment. The pivot also provides control over the order in which the tuples are produced.";

`command setDelimiter(sep:str):void`
address TABsetDelimiter comment "Set the column separator.";

`command setTableBracket(lbrk:str,rbrk:str)`
address TABsetTableBracket comment "Format the brackets around a table";

`command setRowBracket(lbrk:str,rbrk:str)`
address TABsetRowBracket comment "Format the brackets around a row";
# Set the column properties

`pattern setColumn(idx:int, v:any_1)`
address TABsetColumn comment "Bind i-th output column to a variable";

`command setName(idx:int, nme:str)`
address TABsetColumnName comment "Set the display name for a given column";

`command setBracket(idx:int,lbrk:str,rbrk:str)`
address TABsetColumnBracket comment "Format the brackets around a field";

`command setNull(idx:int, fmt:str)`
address TABsetColumnNull comment "Set the display format for a null value for a given column";

`command setWidth(idx:int, maxwidth:int)`
address TABsetColumnWidth comment "Set the maximal display witdh for a given column. All values exceeding the length are simple shortened without any notice.";

`command setPosition(idx:int,f:int,i:int)`
address TABsetColumnPosition comment "Set the character position to use for this field when loading according to fixed (punch-card) layout.";

`command setDecimal(idx:int,s:int,p:int)`
address TABsetColumnDecimal comment "Set the scale and precision for numeric values";

`command firstPage():void`
address TABfirstPage comment "Produce the first page of output";

`command lastPage():void`
address TABlastPage comment "Produce the last page of output";

`command nextPage():void`
address TABnextPage comment "Produce the next page of output";

`command prevPage():void`
address TABprevPage comment "Produce the prev page of output";

`command getPageCnt():void`
address TABgetPageCnt comment "Return the size in number of pages";

`command getPage(i:int):void`
address TABgetPage comment "Produce the i-th page of output";

10.22 Transaction management

In the philosophy of Monet, transaction management overhead should only be paid when necessary. Transaction management is for this purpose implemented as a module. This code base is largely absolute and should be re-considered when serious OLTP is being supported. Note, however, the SQL front-end obeys transaction semantics.

module transaction;

`command sync() :bit`
address TRNglobal_sync comment "Save all persistent BATs";

`command commit() :bit`
address TRNglobal_commit comment "Global commit on all BATs";

`command abort() :bit`
address TRNglobal_abort comment "Global abort on all BATs";

`command subcommit(b:bat[:any_1,:str]):bit`
address TRNsubcommit comment "commit only a set of BATnames, passed in the tail (to which you must have exclusive access!)";

`pattern commit(c:any...)`
address TRNtrans_commit comment "Commit changes in certain BATs.";

`pattern abort(c:any...)`
address TRNtrans_abort comment "Abort changes in certain BATs.";

`pattern clean(c:any...)`
address TRNtrans_clean comment "Declare a BAT clean without flushing to disk.";

`command prev(b:bat[:any_1,:any_2]):bat[:any_1,:any_2]`
address TRNtrans_prev comment "The previous stae of this BAT";

`command alpha(b:bat[:any_1,:any_2]) :bat[:any_1,:any_2]`
address TRNtrans_alpha comment "List insertions since last commit.";

`command delta(b:bat[:any_1,:any_2]) :bat[:any_1,:any_2]`
address TRNtrans_delta comment "List deletions since last commit.";

11 The Inner Core

The innermost library of the MonetDB database system is formed by the library called GDK, an abbreviation of Goblin Database Kernel. Its development was originally rooted in the design of a pure active-object-oriented programming language, before development was shifted towards a re-usable database kernel engine.

GDK is a C library that provides ACID properties on a DSM model [*Copeland85*] , using main-memory database algorithms [*Garcia-Molina92*] built on virtual-memory OS primitives and multi-threaded parallelism. Its implementation has undergone various changes over its decade of development, many of which were driven by external needs to obtain a robust and fast database system.

The coding scheme explored in GDK has also laid a foundation to communicate over time experiences and to provide (hopefully) helpful advice near to the place where the code-reader needs it. Of course, over such a long time the documentation diverges from reality. Especially in areas where the environment of this package is being described. Consider such deviations as historic landmarks, e.g. crystallization of brave ideas and mistakes rectified at a later stage.

11.1 Short Outline

The facilities provided in this implementation are:

- GDK or Goblin Database Kernel routines for session management
- BAT routines that define the primitive operations on the database tables (BATs).
- BBP routines to manage the BAT Buffer Pool (BBP).
- ATOM routines to manipulate primitive types, define new types using an ADT interface.
- HEAP routines for manipulating heaps: linear spaces of memory that are GDK's vehicle of mass storage (on which BATs are built).
- DELTA routines to access inserted/deleted elements within a transaction.
- HASH routines for manipulating GDK's built-in linear-chained hash tables, for accelerating lookup searches on BATs.
- TM routines that provide basic transaction management primitives.
- TRG routines that provided active database support. [DEPRECATED]
- ALIGN routines that implement BAT alignment management.

The Binary Association Table (BAT) is the lowest level of storage considered in the Goblin runtime system [*Goblin*] . A BAT is a self-descriptive main-memory structure that represents the **binary relationship** between two atomic types. The association can be defined over:

`void:`	virtual-OIDs: a densely ascending column of OIDs (takes zero-storage).
`bit:`	Booleans, implemented as one byte values.
`chr:`	A single character (8 bits **integer**s). DEPRECATED for storing text (Unicode not supported).

`bte:`	Tiny (1-byte) integers (8-bit **integer**s).
`sht:`	Short integers (16-bit **integer**s).
`int:`	This is the C **int** type (32-bit).
`oid:`	Unique **long int** values uses as object identifier. Highest bit cleared always. Thus, oids-s are 31-bit numbers on 32-bit systems, and 63-bit numbers on 64-bit systems.
`wrd:`	Machine-word sized integers (32-bit on 32-bit systems, 64-bit on 64-bit systems).
`ptr:`	Memory pointer values. DEPRECATED. Can only be stored in transient BATs.
`flt:`	The IEEE **float** type.
`dbl:`	The IEEE **double** type.
`lng:`	Longs: the C **long long** type (64-bit integers).
`str:`	UTF-8 strings (Unicode). A zero-terminated byte sequence.
`bat:`	Bat descriptor. This allows for recursive adminstered tables, but severely complicates transaction management. Therefore, they CAN ONLY BE STORED IN TRANSIENT BATs. [on the list to become depreciated,

This model can be used as a back-end model underlying other -higher level- models, in order to achieve **better performance** and **data independence** in one go. The relational model and the object-oriented model can be mapped on BATs by vertically splitting every table (or class) for each attribute. Each such a column is then stored in a BAT with type **bat[oid,attribute]**, where the unique object identifiers link tuples in the different BATs. Relationship attributes in the object-oriented model hence are mapped to **bat[oid,oid]** tables, being equivalent to the concept of *join indexes* [*Valduriez87*] .

The set of built-in types can be extended with user-defined types through an ADT interface. They are linked with the kernel to obtain an enhanced library, or they are dynamically loaded upon request.

Types can be derived from other types. They represent something different than that from which they are derived, but their internal storage management is equal. This feature facilitates the work of extension programmers, by enabling reuse of implementation code, but is also used to keep the GDK code portable from 32-bits to 64-bits machines: the **oid** and **ptr** types are derived from **int** on 32-bits machines, but is derived from **lng** on 64 bits machines. This requires changes in only two lines of code each.

To accelerate lookup and search in BATs, GDK supports one built-in search accelerator: hash tables. We choose an implementation efficient for main-memory: bucket chained hash [*LehCar86,Analyti92*] . Alternatively, when the table is sorted, it will resort to merge-scan operations or binary lookups.

BATs are built on the concept of heaps, which are large pieces of main memory. They can also consist of virtual memory, in case the working set exceeds main-memory. In this case, GDK supports operations that cluster the heaps of a BAT, in order to improve performance of its main-memory.

11.1.1 Rationale

The rationale for choosing a BAT as the building block for both relational and object-oriented system is based on the following observations:

- - Given the fact that CPU speed and main-memory increase in current workstation hardware for the last years has been exceeding IO access speed increase, traditional disk-page oriented algorithms do no longer take best advantage of hardware, in most database operations.

  Instead of having a disk-block oriented kernel with a large memory cache, we choose to build a main-memory kernel, that only under large data volumes slowly degrades to IO-bound performance, comparable to traditional systems [*boncz95,boncz96*] .
- - Traditional (disk-based) relational systems move too much data around to save on (main-memory) join operations.

  The fully decomposed store (DSM [*Copeland85*)] assures that only those attributes of a relation that are needed, will have to be accessed.
- - The data management issues for a binary association is much easier to deal with than traditional *struct*-based approaches encountered in relational systems.
- - Object-oriented systems often maintain a double cache, one with the disk-based representation and a C pointer-based main-memory structure. This causes expensive conversions and replicated storage management.\\ GDK does not do such 'pointer swizzling'. It used virtual-memory (**mmap()**) and buffer management advice (**madvise()**) OS primitives to cache only once. Tables take the same form in memory as on disk, making the use of this technique transparent [*oo7*] .

A RDBMS or OODBMS based on BATs strongly depends on our ability to efficiently support tuples and to handle small joins, respectively.

The remainder of this document describes the Goblin Database kernel implementation at greater detail. It is organized as follows:

GDK Interface:
It describes the global interface with which GDK sessions can be started and ended, and environment variables used.

Binary Association Tables:
As already mentioned, these are the primary data structure of GDK. This chapter describes the kernel operations for creation, destruction and basic manipulation of BATs and BUNs (i.e. tuples: Binary UNits).

BAT Buffer Pool:
All BATs are registered in the BAT Buffer Pool. This directory is used to guide swapping in and out of BATs. Here we find routines that guide this swapping process.

GDK Extensibility:
Atoms can be defined using a unified ADT interface. There is also an interface to extend the GDK library with dynamically linked object code.

GDK Utilities:
Memory allocation and error handling primitives are provided. Layers built on top of GDK should use them, for proper system monitoring. Thread management is also included here.

Transaction Management:
For the time being, we just provide BAT-grained concurrency and global transactions. Work is needed here.

BAT Alignment:
Due to the mapping of multi-ary datamodels onto the BAT model, we expect many correspondences among BATs, e.g. *bat(oid,attr1),.. bat(oid,attrN)* vertical decompositions. Frequent activities will be to jump from one attribute to the other ('bunhopping'). If the head columns are equal lists in two BATs, merge or even array lookups can be used instead of hash lookups. The alignment interface makes these relations explicitly manageable.

In GDK, complex data models are mapped with DSM on binary tables. Usually, one decomposes *N*-ary relations into *N* BATs with an **oid** in the head column, and the attribute in the tail column. There may well be groups of tables that have the same sets of **oid**s, equally ordered. The alignment interface is intended to make this explicit. Implementations can use this interface to detect this situation, and use cheaper algorithms (like merge-join, or even array lookup) instead.

BAT Iterators:
Iterators are C macros that generally encapsulate a complex for-loop. They would be the equivalent of cursors in the SQL model. The macro interface (instead of a function call interface) is chosen to achieve speed when iterating main-memory tables.

Common BAT Operations:
These are much used operations on BATs, such as aggregate functions and relational operators. They are implemented in terms of BAT- and BUN-manipulation GDK primitives.

11.2 Interface Files

In this section we summarize the user interface to the GDK library. It consist of a header file (gdk.h) and an object library (gdklib.a), which implements the required functionality. The header file must be included in any program that uses the library. The library must be linked with such a program.

11.2.1 Database Context

The MonetDB environment settings are collected in a configuration file. Amongst others it contains the location of the database directory. First, the database directory is closed for other servers running at the same time. Second, performance enhancements may take effect, such as locking the code into memory (if the OS permits) and preloading the data dictionary. An error at this stage normally lead to an abort.

11.2.2 GDK session handling

int	GDKinit (char *db, char *dbfarm, int allocmap)
int	GDKexit (int status)

The session is bracketed by *GDKinit* and *GDKexit*. Initialization involves setting up the administration for database access, such as memory allocation for the database buffer pool. During the exit phase any pending transaction is aborted and the database is freed for access by other users. A zero is returned upon encountering an erroneous situation.

11.3 Binary Association Tables

Having gone to the previous preliminary definitions, we will now introduce the structure of Binary Association Tables (BATs) in detail. They are the basic storage unit on which GDK is modelled.

The BAT holds an unlimited number of binary associations, called BUNs (**Binary UNits**). The two attributes of a BUN are called **head** (left) and **tail** (right) in the remainder of this document.

The above figure shows what a BAT looks like. It consists of two columns, called head and tail, such that we have always binary tuples (BUNs). The overlooking structure is the **BAT record**. It points to a heap structure called the **BUN heap**. This heap contains the atomic values inside the two columns. If they are fixed-sized atoms, these atoms reside directly in the BUN heap. If they are variable-sized atoms (such as string or polygon), however, the columns has an extra heap for storing those (such **variable-sized atom heaps** are then referred to as **Head Heap**s and **Tail Heap**s). The BUN heap then contains integer byte-offsets (fixed-sized, of course) into a head- or tail-heap. The BUN heap contains a contiguous range of BUNs. It starts after the **first** pointer, and finishes at the end in the **free** area of the BUN. All BUNs after the **inserted** pointer have been added in the last transaction (and will be deleted on a transaction abort). All BUNs between the **deleted** pointer and the **first** have been deleted in this transaction (and will be reinserted at a transaction abort).

The location of a certain BUN in a BAT may change between successive library routine invocations. Therefore, one should avoid keeping references into the BAT storage area for long periods. Passing values between the library routines and the enclosing C program is primarily through value pointers of type *ptr*. Pointers into the BAT storage area should only be used for retrieval. Direct updates of data stored in a BAT is forbidden. The user should adhere to the interface conventions to guarantee the integrity rules and to maintain the (hidden) auxiliary search structures.

11.3.1 GDK variant record type

When manipulating values, MonetDB puts them into value records. The built-in types have a direct entry in the union. Others should be represented as a pointer of memory in pval or as a string, which is basically the same. In such cases the *len* field indicates the size of this piece of memory.

11.3.2 The BAT record

The elements of the BAT structure are introduced in the remainder. Instead of using the underlying types hidden beneath it, one should use a *BAT* type that is supposed to look like this:

```
typedef struct {
          /* static BAT properties */
          bat    batCacheid;       /* bat id: index in BBPcache */
          int    batPersistence;   /* persistence mode */
          bit    batCopiedtodisk;  /* BAT is saved on disk? */
          bit    batSet;           /* all tuples in the BAT are unique? */
          sht    batElmshift       /* log2 of width  */
          sht    batElmsize        /* width in byte of the BUNs */
          /* dynamic BAT properties */
          int    batHeat;          /* heat of BAT in the BBP */
          sht    batDirty;         /* BAT modified after last commit? */
          bit    batDirtydesc;     /* BAT descriptor specific dirty flag */
          bit    batDirtybuns;     /* BUN heap specific dirty flag */
          Heap*  batBuns;          /* Heap where the buns are stored */
          /* DELTA status */
          BUN    batDeleted;       /* first deleted BUN */
          BUN    batFirst;         /* empty BUN before the first alive BUN */
          BUN    batInserted;      /* first inserted BUN */
          size_t batCount;         /* Tuple count */
          /* Head properties */
          int    htype;            /* Head type number */
          str    hident;           /* name for head column */
          bit    hkey;             /* head values should be unique? */
          bit    hsorted;          /* are head values currently ordered? */
          bit    hvarsized;        /* for speed: head type is varsized? */
          oid    halign;           /* alignment OID for head. */
          bit    hheapdirty;       /* head heap specific dirty flag */
          /* Head storage */
          int    hloc;             /* byte-offset in BUN for head elements */
          Heap   *hheap;           /* heap for varsized head values */
          Hash   *hhash;           /* linear chained hash table on head */
          /* Tail properties */
          int    ttype;            /* Tail type number */
          str    tident;           /* name for tail column */
          bit    tkey;             /* tail values should be unique? */
          bit    tsorted;          /* are tail values currently ordered? */
          bit    tvarsized;        /* for speed: tail type is varsized? */
          oid    talign;           /* alignment OID for head. */
          bit    theapdirty;       /* tail heap specific dirty flag */
          /* Tail storage */
          int    tloc;             /* byte-offset in BUN for tail elements */
          Heap   theap;            /* heap for varsized tail values */
```

```
        Hash   thash;            /* linear chained hash table on tail */
 } BAT;
```

The internal structure of the **BAT** record is in fact much more complex, but GDK programmers should refrain of making use of that.

The reason for this complex structure is to allow for a BAT to exist in two incarnations at the time: the *normal view* and the *reversed view*. Each bat *b* has a BATmirror(*b*) which has the negative **cacheid** of b in the BBP.

Since we don't want to pay cost to keep both views in line with each other under BAT updates, we work with shared pieces of memory between the two views. An update to one will thus automatically update the other. In the same line, we allow **synchronized BATs** (BATs with identical head columns, and marked as such in the **BAT Alignment** interface) now to be clustered horizontally.

11.3.3 Heap Management

Heaps are the low-level entities of mass storage in BATs. Currently, they can either be stored on disk, loaded into memory, or memory mapped.

int	HEAPalloc (Heap *h, size_t nitems, size_t itemsize);
int	HEAPfree (Heap *h);
int	HEAPextend (Heap *h, size_t size);
int	HEAPload (Heap *h, str nme,ext, int trunc);
int	HEAPsave (Heap *h, str nme,ext);
int	HEAPcopy (Heap *dst,*src);
int	HEAPdelete (Heap *dst, str o, str ext);
int	HEAPwarm (Heap *h);
int	

These routines should be used to alloc free or extend heaps; they isolate you from the different ways heaps can be accessed.

11.3.4 Internal HEAP Chunk Management

Heaps are used in BATs to store data for variable-size atoms. The implementor must manage malloc()/free() functionality for atoms in this heap. A standard implementation is provided here.

`void`	HEAP_initialize (Heap* h, size_t nbytes, size_t nprivate, int align)
`void`	HEAP_destroy (Heap* h)
`int`	HEAP_malloc (Heap* heap, size_t nbytes)
`void`	HEAP_free (Heap *heap, size_t block)
`int`	HEAP_private (Heap* h)
`void`	HEAP_printstatus (Heap* h)
`void`	HEAP_check (Heap* h)

The heap space starts with a private space that is left untouched by the normal chunk allocation. You can use this private space e.g. to store the root of an rtree *HEAP_malloc* allocates a chunk of memory on the heap, and returns an index to it. *HEAP_free* frees a previously allocated chunk *HEAP_private* returns an integer index to private space.

11.3.5 BAT construction

`BAT*`	BATnew (int headtype, int tailtype, size_t cap)
`BAT*`	BATclone (BAT *c, size_t cap)
`BAT*`	BATextend (BAT *b, size_t newcap)

A temporary BAT is instantiated using *BATnew* with the type aliases of the required binary association. The aliases include the built-in types, such as *TYPE_int....TYPE_ptr*, and the atomic types introduced by the user. The initial capacity to be accommodated within a BAT is indicated by *cap*. Their extend is automatically incremented upon storage overflow. Failure to create the BAT results in a NULL pointer.

The routine *BATclone* creates an empty BAT storage area with the properties inherited from its argument.

11.3.6 BUN manipulation

BAT*	BATins (BAT *b, BAT *c, bit force)
BAT*	BATappend (BAT *b, BAT *c, bit force)
BAT*	BATdel (BAT *b, BAT *c, bit force)
BAT*	BUNins (BAT *b, ptr left, ptr right, bit force)
BAT*	BUNappend (BAT *b, ptr right, bit force)
BAT*	BUNreplace (BAT *b, ptr left, ptr right, bit force)
int	BUNdel (BAT *b, ptr left, ptr right, bit force)
int	BUNdelHead (BAT *b, ptr left, bit force)
BUN	BUNfnd (BAT *b, ptr head)
void	BUNfndOID (BUN result, BAT *b, oid *head)
void	BUNfndSTD (BUN result, BAT *b, ptr head)
BUN	BUNlocate (BAT *b, ptr head, ptr tail)
ptr	BUNhead (BAT *b, BUN p)
ptr	BUNtail (BAT *b, BUN p)

The BATs contain a number of fixed-sized slots to store the binary associations. These slots are called BUNs or BAT units. A BUN variable is a pointer into the storage area of the BAT, but it has limited validity. After a BAT modification, previously obtained BUNs may no longer reside at the same location.

The association list does not contain holes. This density permits users to quickly access successive elements without the need to test the items for validity. Moreover, it simplifies transport to disk and other systems. The negative effect is that the user should be aware of the evolving nature of the sequence, which may require copying the BAT first.

The update operations come in three flavors. Element-wise updates can use *BUNins*, *BUNappend*, *BUNreplace*, *BUNdel*, and *BUNdelHead*. The batch update operations are *BATins*, *BATappend* and *BATdel*.

Only experts interested in speed may use *BUNfastins*, since it skips most consistency checks, does not update search accelerators, and does not maintain properties such as the *hsorted* and *tsorted* flags. Beware!

The routine *BUNfnd* provides fast access to a single BUN providing a value for the head of the binary association. A very fast shortcut for *BUNfnd* if the selection type is known to be integer or OID, is provided in the form of the macro *BUNfndOID*.

To select on a tail, one should use the reverse view obtained by *BATmirror*.

The routines *BUNhead* and *BUNtail* return a pointer to the first and second value in an association, respectively. To guard against side effects on the BAT, one should normally copy this value into a scratch variable for further processing.

Behind the interface we use several macros to access the BUN fixed part and the variable part. The BUN operators always require a BAT pointer and BUN identifier.

- BAThtype(b) and BATttype(b) find out the head and tail type of a BAT.
- BUNfirst(b) returns a BUN pointer to the first BUN as a BAT.
- BUNlast(b) returns the BUN pointer directly after the last BUN in the BAT.
- BUNsize(b) gives the size in bytes of each BUN.
- BUNindex(b, p) computes the index number of a given BUN.
- BUNptr(b, i) computes the address of the i-th BUN in the BAT.
- BUNhead(b, p) and BUNtail(b, p) return pointers to the head-value and tail-value in a given BUN.
- BUNhloc(b, p) and BUNtloc(b, p) do the same thing, but knowing in advance that the head-atom resp. tail-atom of a BAT is fixed size.
- BUNhvar(b, p) and BUNtvar(b, p) do the same thing, but knowing in advance that the head-atom resp. tail-atom of a BAT is variable sized.

11.3.7 BAT properties

size_t	BATcount (BAT *b)
size_t	BATbuncount (BAT *b)
str	BATrename (BAT *b, str nme)
BAT *	BATkey (BAT *b, int onoff)
BAT *	BATset (BAT *b, int onoff)
BAT *	BATmode (BAT *b, int mode)
BAT *	BATsetaccess (BAT *b, int mode)
int	BATdirty (BAT *b)
int	BATgetaccess (BAT *b)
int	BATversion (BAT *b)

The function *BATcount* returns the number of associations stored in the BAT.

The function *BATbuncount* returns the space that is occupied in associations in the BAT. This is not the same as *BATcount*, since the first N associations may be unused or delta data.

The BAT is given a new logical name using *BATrename*.

The integrity properties to be maintained for the BAT are controlled separately. A key property indicates that duplicates in the association dimension are not permitted. The BAT is turned into a set of associations using *BATset*. Key and set properties are orthogonal integrity constraints. The strongest reduction is obtained by making the BAT a set with key restrictions on both dimensions.

The persistency indicator tells the retention period of BATs. The system support three modes: PERSISTENT, TRANSIENT, and SESSION. The PERSISTENT BATs are automatically saved upon session boundary or transaction commit. TRANSIENT BATs are removed upon transaction boundary. SESSION BATs are removed at the end of a session.

They are normally used to maintain temporary results. All BATs are initially TRANSIENT unless their mode is changed using the routine *BATmode*.

The BAT properties may be changed at any time using *BATkey*, *BATset*, and *BATmode*.

Valid BAT access properties can be set with *BATsetaccess* and *BATgetaccess*: BAT_READ, BAT_APPEND, and BAT_WRITE. BATs can be designated to be read-only. In this case some memory optimizations may be made (slice and fragment bats can point to stable subsets of a parent bat). A special mode is append-only. It is then allowed to insert BUNs at the end of the BAT, but not to modify anything that already was in there.

11.3.8 BAT manipulation

BAT *	BATclear (BAT *b)
BAT *	BATcopy (BAT *b, int ht, int tt, int writeable)
BAT *	BATmark (BAT *b, oid base)
BAT *	BATmark_grp (BAT *b, BAT *g, oid *s)
BAT *	BATnumber (BAT *b)
BAT *	BATmirror (BAT *b)
BAT *	BATreverse (BAT *b)
BAT *	BATreset (BAT *b)

The routine *BATclear* removes the binary associations, leading to an empty, but (re-)initialized BAT. Its properties are retained. A temporary copy is obtained with *BATcopy*. The new BAT has an unique name. The routine *BATmark* creates a binary association that introduces a new tail column of fresh densely ascending OIDs. The base OID can be given explicitly, or if oid_nil is passed, is chosen as a new unique range by the system. A similar routine is *BATnumber*, which copies the heads and assigns an integer index to the tail. It plays a crucial role in administration of query results.

The routine *BATmirror* returns the mirror image BAT (where tail is head and head is tail) of that same BAT. This does not involve a state change in the BAT (as previously): both views on the BAT exist at the same time.

11.3.9 BAT Input/Output

BAT *	BATload (str name)
BAT *	BATsave (BAT *b)
int	BATmmap (BAT *b, int bn, int hh, int th)
int	BATmadvise (BAT *b, int bn, int hh, int th)
int	BATmmap_pin (BAT *b)
int	BATmmap_unpin (BAT *b)
int	BATdelete (BAT *b)
BAT *	BATconvert (str name, int direction)

A BAT created by *BATnew* is considered temporary until one calls the routine *BATsave* or *BATmode*. This routine reserves disk space and checks for name clashes in the BAT directory. It also makes the BAT persistent. The empty BAT is initially marked as ordered on both columns. Failure to read or write the BAT results in a NULL, otherwise it returns the BAT pointer.

The *BATconvert* converts a BAT that contains data saved on hardware with a different byte order (little endian vs big endian), into the correct format. Direction is either

CONV_NTOH or CONV_HTON, indication conversion from network to host or host to network format.

MonetDB now has a mmap trim thread that takes care of flushing the memory mapped regions when MonetDB starts to consume too much main memory. Heaps (that are randomly accessed) can be excluded from this mechanism, by pinning them. BATmmap_pin/unpin do this for all heaps of a BAT.

11.3.10 Heap Storage Modes

The discriminative storage modes are memory-mapped, compressed, or loaded in memory. The **BATmmap()** changes the storage mode of each heap associated to a BAT. As can be seen in the bat record, each BAT has one BUN-heap (*bn*), and possibly two heaps (*hh* and *th*) for variable-sized atoms. The *BATmadvise* call works in the same way. Using the *madvise()* system call it issues buffer management advise to the OS kernel, as for the expected usage pattern of the memory in a heap.

11.3.11 Printing

int	BATprintf (stream *f, BAT *b)
int	BATmultiprintf (stream *f, int argc, BAT *b[], int printoid, int order, int printorderby)

The functions to convert BATs into ASCII and the reverse use internally defined formats. They are primarily meant for ease of debugging and to a lesser extent for output processing. Printing a BAT is done essentially by looping through its components, printing each association. If an index is available, it will be used. The *BATmultiprintf* command assumes a set of BATs with corresponding oid-s in the head columns. It performs the multijoin over them, and prints the multi-column result on the file.

11.3.12 BAT clustering

BAT *	BATsort (BAT *b)
BAT *	BATsort_rev (BAT *b)
BAT *	BATorder (BAT *b)
BAT *	BATorder_rev (BAT *b)
BAT *	BATrevert (BAT *b)
int	BATordered (BAT *b)

When working in a main-memory situation, clustering of data on disk-pages is not important. Whenever mmap()-ed data is used intensively, reducing the number of page faults is a hot issue.

The above functions rearrange data in MonetDB heaps (used for storing BUNs var-sized atoms, or accelerators). Applying these clusterings will allow that MonetDB's main-memory oriented algorithms work efficiently also in a disk-oriented context.

The *BATsort* functions return a copy of the input BAT, sorted in ascending order on the head column. *BATordered* starts a check on the head values to see if they are ordered. The result is returned and stored in the *hsorted* field of the BAT. *BATorder* is similar to *BATsort*, but sorts the BAT itself, rather than returning a copy (BEWARE: this operation destroys the delta information. TODO:fix). The *BATrevert* puts all the live BUNs of a BAT in reverse order. It just reverses the sequence, so this does not necessarily mean that they are sorted in reverse order!

11.4 BAT Buffer Pool

int	BBPfix (bat bi)
int	BBPunfix (bat bi)
int	BBPincref (bat bi, int logical)
int	BBPdecref (bat bi, int logical)
void	BBPhot (bat bi)
void	BBPcold (bat bi)
str	BBPname (bat bi)
bat	BBPindex (str nme)
BAT*	BATdescriptor (bat bi)
bat	BBPcacheid (BAT *b)

The BAT Buffer Pool module contains the code to manage the storage location of BATs. It uses two tables *BBPlogical* and *BBphysical* to relate the BAT name with its corresponding file system name. This information is retained in an ASCII file within the database home directory for ease of inspection. It is loaded upon restart of the server and saved upon transaction commit (if necessary).

The remaining BBP tables contain status information to load, swap and migrate the BATs. The core table is *BBPcache* which contains a pointer to the BAT descriptor with its heaps. A zero entry means that the file resides on disk. Otherwise it has been read or mapped into memory.

BATs loaded into memory are retained in a BAT buffer pool. They retain their position within the cache during their life cycle, which make indexing BATs a stable operation. Their descriptor can be obtained using *BBPcacheid*.

The *BBPindex* routine checks if a BAT with a certain name is registered in the buffer pools. If so, it returns its BAT id. The *BATdescriptor* routine has a BAT id parameter, and returns a pointer to the corresponding BAT record (after incrementing the reference count). The BAT will be loaded into memory, if necessary.

11.5 GDK Extensibility

GDK can be extended with new atoms, search accelerators and storage modes.

11.5.1 Atomic Type Descriptors

The atomic types over which the binary associations are maintained are described by an atom descriptor.

void	ATOMproperty (str nme, char *property, int (*fcn)());
int	ATOMindex (char *nme);
int	ATOMdump ();
void	ATOMdelete (int id);
str	ATOMname (int id);
int	ATOMsize (int id);
int	ATOMalign (int id);
int	ATOMvarsized (int id);
ptr	ATOMnilptr (int id);
int	ATOMfromstr (int id, str s, int* len, ptr* v_dst);
int	ATOMtostr (int id, str s, int* len, ptr* v_dst);

int	ATOMhash (int id, ptr val, in mask);
int	ATOMcmp (int id, ptr val_1, ptr val_2);
int	ATOMconvert (int id, ptr v, int direction);
int	ATOMfix (int id, ptr v);
int	ATOMunfix (int id, ptr v);
int	ATOMheap (int id, Heap *hp, size_t cap);
void	ATOMheapconvert (int id, Heap *hp, int direction);
int	ATOMheapcheck (int id, Heap *hp, HeapRepair *hr);
int	ATOMput (int id, Heap *hp, BUN pos_dst, ptr val_src);
int	ATOMdel (int id, Heap *hp, BUN v_src);
int	ATOMlen (int id, ptr val);
ptr	ATOMnil (int id);
int	ATOMformat (int id, ptr val, char** buf);
int	ATOMprint (int id, ptr val, stream *fd);
ptr	ATOMdup (int id, ptr val);

11.5.2 Atom Definition

User defined atomic types can be added to a running system with the following interface:.

- *ATOMproperty()* registers a new atom definition, if there is no atom registered yet under that name. It then installs the attribute of the named property. Valid names are `"size"`, `"align"`, `"null"`, `"fromstr"`, `"tostr"`, `"cmp"`, `"hash"`, `"put"`, `"get"`, `"del"`, `"length"` and `"heap"`.
- *ATOMdelete()* unregisters an atom definition.
- *ATOMindex()* looks up the atom descriptor with a certain name.

11.5.3 Atom Manipulation

- The *ATOMname()* operation retrieves the name of an atom using its id.
- The *ATOMsize()* operation returns the atoms fixed size.
- The *ATOMalign()* operation returns the atoms minimum alignment. If the alignment info was not specified explicitly during atom install, it assumes the maximum value of `{1,2,4,8}` smaller than the atom size.
- The *ATOMnilptr()* operation returns a pointer to the nil-value of an atom. We usually take one dedicated value halfway down the negative extreme of the atom range (if such a concept fits), as the nil value.
- The *ATOMnil()* operation returns a copy of the nil value, allocated with GDKmalloc().
- The *ATOMheap()* operation creates a new var-sized atom heap in 'hp' with capacity 'cap'.
- The *ATOMhash()* computes a hash index for a value. 'val' is a direct pointer to the atom value. Its return value should be an integer between 0 and 'mask'.
- The *ATOMcmp()* operation computes two atomic values. Its parameters are pointers to atomic values.
- The *ATOMlen()* operation computes the byte length for a value. 'val' is a direct pointer to the atom value. Its return value should be an integer between 0 and 'mask'.
- The *ATOMdel()* operation deletes a var-sized atom from its heap 'hp'. The integer byte-index of this value in the heap is pointed to by 'val_src'.

- The *ATOMput()* operation inserts an atom 'src_val' in a BUN at 'dst_pos'. This involves copying the fixed sized part in the BUN. In case of a var-sized atom, this fixed sized part is an integer byte-index into a heap of var-sized atoms. The atom is then also copied into that heap 'hp'.
- The *ATOMfix()* and *ATOMunfix()* operations do bookkeeping on the number of references that a GDK application maintains to the atom. In MonetDB, we use this to count the number of references directly, or through BATs that have columns of these atoms. The only operator for which this is currently relevant is BAT. The operators return the POST reference count to the atom. BATs with fixable atoms may not be stored persistently.
- The *ATOMfromstr()* parses an atom value from string 's'. The memory allocation policy is the same as in *ATOMget()*. The return value is the number of parsed characters.
- The *ATOMprint()* prints an ASCII description of the atom value pointed to by 'val' on file descriptor 'fd'. The return value is the number of parsed characters.
- The *ATOMformat()* is similar to *ATOMprint()*. It prints an atom on a newly allocated string. It must later be freed with **GDKfree**. The number of characters written is returned. This is minimally the size of the allocated buffer.
- The *ATOMdup()* makes a copy of the given atom. The storage needed for this is allocated and should be removed by the user.

These wrapper functions correspond closely to the interface functions one has to provide for a user-defined atom. They basically (with exception of *ATOMput()*, *ATOMprint()* and *ATOMformat()*) just have the atom id parameter prepended to them.

11.5.4 Unique OIDs

oid	OIDseed (oid seed);
oid	OIDnew (size_t inc);

OIDs are special kinds of unsigned integers because the system guarantees uniqueness. For system simplicity and performance, OIDs are now represented as (signed) integers; however this is hidden in the system internals and shouldn't affect semantics.

The *OIDnew(N)* claims a range of N contiguous unique, unused OIDs, and returns the starting value of this range. The highest OIDBITS designate site. [DEPRECATED]

11.5.5 Built-in Accelerator Functions

BAT*	BAThash (BAT *b, size_t masksize)
BAT *	BAThashsplit (BAT *b, size_t n, int unary)
BAT *	BATrangesplit (BAT *b, int n)

The current BAT implementation supports one search accelerator: hashing. The routine *BAThash* makes sure that a hash accelerator on the head of the BAT exists. A zero is returned upon failure to create the supportive structures.

The hash data structures are currently maintained during update operations.

A *BAT* can be redistributed over *n* buckets using a hash function with *BAThashsplit*. The return value is a list of BAT pointers. Similarly, a range partitioning based is supported.

11.5.6 Multilevel Storage Modes

We should bring in the compressed mode as the first, maybe built-in, mode. We could than add for instance HTTP remote storage, SQL storage, and READONLY (cd-rom) storage.

11.6 GDK Utilities

Interfaces for memory management, error handling, thread management and system information.

11.6.1 GDK memory management

void*	GDKmalloc (size_t size)
void*	GDKzalloc (size_t size)
void*	GDKmallocmax (size_t size, size_t *maxsize, int emergency)
void*	GDKrealloc (void* pold, size_t size)
void*	GDKreallocmax (void* pold, size_t size, size_t *maxsize, int emergency)
void	GDKfree (void* blk)
str	GDKstrdup (str s)
void*	GDKvmalloc (size_t size, size_t *maxsize, int emergency)
void*	GDKvmrealloc (void* pold, size_t oldsize, size_t newsize, size_t oldmax, size_t *maxsize, int emergency)
void	GDKvmfree (void* blk, size_t size, size_t maxsize)

These utilities are primarily used to maintain control over critical interfaces to the C library. Moreover, the statistic routines help in identifying performance and bottlenecks in the current implementation.

Compiled with -DMEMLEAKS the GDK memory management log their activities, and are checked on inconsistent frees and memory leaks.

11.6.2 GDK error handling

str	GDKmessage
bit	GDKsilent
int	GDKfatal(str msg)
int	GDKwarning(str msg)
int	GDKerror (str msg)
int	GDKgoterrors ()
int	GDKsyserror (str msg)
str	GDKerrbuf
	GDKsetbuf (str buf)

The error handling mechanism is not sophisticated yet. Experience should show if this mechanism is sufficient. Most routines return a pointer with zero to indicate an error.

The error messages are also copied to standard output unless *GDKsilent* is set to a non-zero value. The last error message is kept around in a global variable.

Error messages can also be collected in a user-provided buffer, instead of being echoed to a stream. This is a thread-specific issue; you want to decide on the error mechanism on a thread-specific basis. This effect is established with *GDKsetbuf*. The memory (de)allocation

of this buffer, that must at least be 1024 chars long, is entirely by the user. A pointer to this buffer is kept in the pseudo-variable *GDKerrbuf*. Normally, this is a NULL pointer.

The GDKembedded variable is a property set in the configuration file to indicate that the kernel is only allowed to run as a single process. This can be used to remove all locking overhead. The actual state of affairs is maintained in GDKprotected, which is set when locking is required, e.g. when multiple threads become active.

The kernel maintains a central table of all active threads. They are indexed by their tid. The structure contains information on the input/output file descriptors, which should be set before a database operation is started. It ensures that output is delivered to the proper client. The Thread structure should be ideally made directly accessible to each thread. This speeds up access to tid and file descriptors.

11.7 Transaction Management

int	TMcommit ()
int	TMabort ()
int	TMsubcommit ()

MonetDB by default offers a global transaction environment. The global transaction involves all activities on all persistent BATs by all threads. Each global transaction ends with either *TMabort* or *TMcommit*, and immediately starts a new transaction. *TMcommit* implements atomic commit to disk on the collection of all persistent BATs. For all persistent BATs, the global commit also flushes the delta status for these BATs (see *BATcommit/BATabort*). This allows to perform *TMabort* quickly in memory (without re-reading all disk images from disk). The collection of which BATs is persistent is also part of the global transaction state. All BATs that where persistent at the last commit, but were made transient since then, are made persistent again by *TMabort*. In other words, BATs that are deleted, are only physically deleted at *TMcommit* time. Until that time, rollback (*TMabort*) is possible.

Use of *TMabort* is currently NOT RECOMMENDED due to two bugs:

- *TMabort after a failed* %TMcommit does not bring us back to the previous committed state; but to the state at the failed *TMcommit*.
- At runtime, *TMabort* does not undo BAT name changes, whereas a cold MonetDB restart does.

In effect, the problems with *TMabort* reduce the functionality of the global transaction mechanism to consistent checkpointing at each *TMcommit*. For many applications, consistent checkpointingis enough.

Extension modules exist that provide fine grained locking (lock module) and Write Ahead Logging (sqlserver). Applications that need more fine-grained transactions, should build this on top of these extension primitives.

TMsubcommit is intended to quickly add or remove BATs from the persistent set. In both cases, rollback is not necessary, such that the commit protocol can be accelerated. It comes down to writing a new BBP.dir.

Its parameter is a BAT-of-BATs (in the tail); the persistence status of that BAT is committed. We assume here that the calling thread has exclusive access to these bats. An error is reported if you try to partially commit an already committed persistent BAT (it needs the rollback mechanism).

11.7.1 Delta Management

BAT *	BATcommit (BAT *b)
BAT *	BATfakeCommit (BAT *b)
BAT *	BATundo (BAT *b)
BAT *	BATprev (BAT *b)
BAT *	BATalpha (BAT *b)
BAT *	BATdelta (BAT *b)

The BAT keeps track of updates with respect to a 'previous state'. Do not confuse 'previous state' with 'stable' or 'commited-on-disk', because these concepts are not always the same. In particular, they diverge when BATcommit, BATfakecommit, and BATundo are called explictly, bypassing the normal global *TMcommit* protocol (some applications need that flexibility).

BATcommit make the current BAT state the new 'stable state'. This happens inside the global *TMcommit* on all persistent BATs previous to writing all bats to persistent storage using a BBPsync[?].

EXPERT USE ONLY: The routine *BATfakeCommit* updates the delta information on BATs and clears the dirty bit. This avoids any copying to disk. Expert usage only, as it bypasses the global commit protocol, and changes may be lost after quitting or crashing MonetDB.

BATabort undo-s all changes since the previous state. The global TMabort[?] achieves a rollback to the previously committed state by doing BATabort on all persistent bats.

BUG: after a failed *TMcommit*, *TMabort* does not do anything because *TMcommit* does the *BATcommit*s *before* attempting to sync to disk instead of *after& doing this.

The previous state can also be queried. *BATprev* is a view on the current BAT as it was in the previous state. *BATalpha* shows only the BUNs inserted since the previous state, and *BATdelta* the deleted buns.

CAVEAT: *BATprev*, *BATalpha* and *BATdelta* only return views if the underlying BATs are read-only (often not the case when BATs are being updated). Otherwise, copies must be made anyway.

11.8 BAT Alignment and BAT views

int	ALIGNsynced (BAT* b1, BAT* b2)
int	ALIGNsync (BAT *b1, BAT *b2)
int	ALIGNrelated (BAT *b1, BAT *b2)
int	ALIGNsetH ((BAT *dst, BAT *src) \hline
BAT *	BATpropcheck (BAT *b, int mode) \hline
BAT*	VIEWcreate (BAT *b)
bat	VIEWparent (BAT *b)
BAT*	VIEWhead (BAT *b)
BAT*	VIEWcombine (BAT *b)
BAT*	VIEWreset (BAT *b)
BAT*	BATmaterialize (BAT *b, size_t size)

Alignments of two columns of a BAT means that the system knows whether these two columns are exactly equal. Relatedness of two BATs means that one pair of columns (either head or tail) of both BATs is aligned. The first property is checked by *ALIGNsynced*, the latter by *ALIGNrelated*.

The *BATpropcheck* examines a BAT and tries to set all applicable properties (key,sorted,align,dense).

All algebraic BAT commands propagate the properties - including alignment properly on their results.

VIEW BATs are BATs that lend their storage from a parent BAT. They are just a descriptor that points to the data in this parent BAT. A view is created with *VIEWcreate*. The cache id of the parent (if any) is returned by *VIEWparent* (otherwise it returns 0).

VIEW bats are read-only!!

The *VIEWcombine* gives a view on a BAT that has two head columns of the parent. The *VIEWhead* constructs a BAT view that has the same head column as the parent, but has a void column with seqbase=nil in the tail. *VIEWreset* creates a normal BAT with the same contents as its view parameter (it converts void columns with seqbase!=nil to materialized oid columns).

The *BATmaterialize* materializes a VIEW (TODO) or void bat inplace. The size parameter can be used to hint the new size. This is useful as materialization is usually needed for updates.

11.9 BAT Iterators

BATloop	(BAT *b; BUN p, BUN q)
BATloopDEL	(BAT *b; BUN p; BUN q; int dummy)
DELloop	(BAT *b; BUN p, BUN q, int dummy)
HASHloop	(BAT *b; Hash *h, size_t dummy; ptr value)
HASHloop_bit	(BAT *b; Hash *h, size_t idx; bit *value, BUN w)
HASHloop_chr	(BAT *b; Hash *h, size_t idx; char *value, BUN w)
HASHloop_bte	(BAT *b; Hash *h, size_t idx; bte *value, BUN w)
HASHloop_sht	(BAT *b; Hash *h, size_t idx; sht *value, BUN w)
HASHloop_bat	(BAT *b; Hash *h, size_t idx; bat *value, BUN w)
HASHloop_ptr	(BAT *b; Hash *h, size_t idx; ptr *value, BUN w)
HASHloop_int	(BAT *b; Hash *h, size_t idx; int *value, BUN w)
HASHloop_oid	(BAT *b; Hash *h, size_t idx; oid *value, BUN w)
HASHloop_wrd	(BAT *b; Hash *h, size_t idx; wrd *value, BUN w)
HASHloop_flt	(BAT *b; Hash *h, size_t idx; flt *value, BUN w)
HASHloop_lng	(BAT *b; Hash *h, size_t idx; lng *value, BUN w)
HASHloop_dbl	(BAT *b; Hash *h, size_t idx; dbl *value, BUN w)
HASHloop_str	(BAT *b; Hash *h, size_t idx; str value, BUN w)
HASHloploc	(BAT *b; Hash *h, size_t idx; ptr value, BUN w)
HASHloopvar	(BAT *b; Hash *h, size_t idx; ptr value, BUN w)
SORTloop	(BAT *b,p,q,tl,th,s)

The *BATloop()* looks like a function call, but is actually a macro. The following example gives an indication of how they are to be used:

```
void
print_a_bat(BAT *b)
{
        BUN p, q;
        BATloop(b, p, q)
                printf("Element %3d has value %d\n",
                           *(int*) BUNhead(b, p), *(int*) BUNtail(b, p));
}
```

11.9.1 simple sequential scan

The first parameter is a BAT, the p and q are BUN pointers, where p is the iteration variable.

```
#define BATloop(r, p, q) \
for(q = BUNlast(r), p = BUNfirst(r);p < q; p = BUNnext(r, p))
#define BATloopFast(r, p, q, x) \
for(q = BUNlast(r), x = BUNsize(r), p = BUNfirst(r); p < q; p += x)
```

11.9.2 batloop where the current element can be deleted/updated

Normally it is *strictly forbidden* to update the BAT over which is being iterated, or delete the current element. This can only be done with the specialized batloop below. When doing a delete, do not forget to update the current pointer with a p = *BUNdelete*(b,p) (the delete may modify the current pointer p). After the delete/update has taken place, the pointer p is in an inconsistent state till the next iteration of the batloop starts.

```
#define BATloopDEL(r, p, q, x) \
for(p = BUNfirst(r), q = BUNlast(r), x = BUNsize(r); p < q; \
    q=MIN(q,BUNlast(r)), p += x)
```

11.9.3 sequential scan over deleted BUNs

Stable BUNS that were deleted, are conserved to transaction end. You may inspect these data items. Again, the b is a BAT, p and q are BUNs, where p is the iteration variable.

```
#define DELloop(b, p, q, x) \
for (q = (b)->batFirst, p = (b)->batDeleted, x=BUNsize(b); p < q; p += x)
```

11.9.4 hash-table supported loop over BUNs

The first parameter 'b' is a BAT, the second ('h') should point to 'b->hhash', and 'v' a pointer to an atomic value (corresponding to the head column of 'b'). The 'hb' is an integer index, pointing out the 'hb'-th BUN. You should translate this index with *BUNptr(b,hb)* to obtain a BUN from the iteration variable.

```
#define GDK_STREQ(l,r) (*(char*) (l) == *(char*) (r) && !strcmp(l,r))

#define HASHloop(b, h, hb, v) \
for (hb = h->hash[HASHprobe(h, v)]; hb != HASH_MAX; hb = h->link[hb]) \
```

```
if (ATOMcmp(h->type, v, BUNhead(b, BUNptr(b, hb))) == 0)
#define HASHloop_str(b, h, hb, v) \
for (hb = h->hash[strHash(v)&h->mask]; hb != HASH_MAX; hb = h->link[hb])
\
if (GDK_STREQ(v, BUNhvar(b, BUNptr(b, hb))))
```

For string search, we can optimize if the string heap has eliminated all doubles. This is the case when not too many different strings are stored in the heap. You can check this with the macro *strElimDoubles()* If so, we can just compare integer index numbers instead of strings:

```
#define HASHloop_fstr(b, h, hb, idx, v) \
for (hb = h->hash[strHash(v)&h->mask], idx = strLocate((b)->hheap,v); \
     hb != HASH_MAX; hb = h->link[hb]) \
     if (*(var_t*) BUNhloc(b, BUNptr(b, hb)) == idx)
```

The following example shows how the hashloop is used:

```
void
print_books(BAT *author\_books, str author)
{
        BAT *b = author\_books;
        size_t i;

        printf("%s\n==================\n", author);
        HASHloop(b, (b)->hhash, i, author)
                        printf("%s\n", ((str) BUNtail(b, BUNptr(b,i)));
}
```

Note that for optimization purposes, we could have used a *HASHloop_str* instead, and also a *BUNtvar* instead of a *BUNtail* (since we know the tail-type of *author_books* is string, hence variable-sized). However, this would make the code less general.

11.9.5 specialized hashloops

HASHloops come in various flavors, from the general *HASHloop*, as above, to specialized versions (for speed) where the type is known (e.g. *HASHloop_int*), or the fact that the atom is fixed-sized (*HASHlooploc*) or variable-sized (*HASHloopvar*). These hashloops have one extra parameter, a BUN pointer 'w' (again for speed) which already points at the current BUN (i.e. w == *BUNptr(b,hb)*).

```
#define HASHlooploc(b, h, hb, v, w) \
for (hb = h->hash[HASHprobe(h, v)]; hb != HASH_MAX; hb = h->link[hb]) \
if ((w=BUNptr(b, hb)) != NULL && ATOMcmp(h->type, v, BUNhloc(b, w)) == 0)
#define HASHloopvar(b, h, hb, v, w) \
for (hb = h->hash[HASHprobe(h, v)]; hb != HASH_MAX; hb = h->link[hb]) \
if ((w=BUNptr(b, hb)) != NULL && ATOMcmp(h->type, v, BUNhvar(b, w)) == 0)
```

11.9.6 loop over a BAT with ordered tail

Here we loop over a BAT with an ordered tail column (see for instance *BATsort*). Again, 'p' and 'q' are iteration variables, where 'p' points at the current BUN. 'tl' and 'th' are pointers to atom corresponding to the minimum (included) and maximum (included) bound in the selected range of BUNs. A nil-value means that there is no bound. The 's' finally is an integer denoting the bunsize, used for speed.

```
#define SORTloop(b,p,q,tl,th,s) \
if (!(BATtordered(b)&1)) GDKerror("SORTloop: BAT not sorted.\n"); \
else for (p = (ATOMcmp((b)->ttype,tl,ATOMnilptr((b)->ttype))?  \
        SORTfndfirst(b,tl):BUNfirst(b)), \
  q = (ATOMcmp((b)->ttype,th,ATOMnilptr((b)->ttype))?  \
        SORTfndlast(b,th):BUNlast(b)), \
  s = BUNsize(b); p < q; p += s)


/* OIDDEPEND */
#if SIZEOF_OID == SIZEOF_INT
#define SORTfnd_oid(b,v) SORTfnd_int(b,v)
#define SORTfndfirst_oid(b,v) SORTfndfirst_int(b,v)
#define SORTfndlast_oid(b,v) SORTfndlast_int(b,v)
#define SORTloop_oid(b,p,q,tl,th,s) SORTloop_int(b,p,q,tl,th,s)
#else
#define SORTfnd_oid(b,v) SORTfnd_lng(b,v)
#define SORTfndfirst_oid(b,v) SORTfndfirst_lng(b,v)
#define SORTfndlast_oid(b,v) SORTfndlast_lng(b,v)
#define SORTloop_oid(b,p,q,tl,th,s) SORTloop_lng(b,p,q,tl,th,s)
#endif
#if SIZEOF_WRD == SIZEOF_INT
#define SORTfnd_wrd(b,v) SORTfnd_int(b,v)
#define SORTfndfirst_wrd(b,v) SORTfndfirst_int(b,v)
#define SORTfndlast_wrd(b,v) SORTfndlast_int(b,v)
#define SORTloop_wrd(b,p,q,tl,th,s) SORTloop_int(b,p,q,tl,th,s)
#else
#define SORTfnd_wrd(b,v) SORTfnd_lng(b,v)
#define SORTfndfirst_wrd(b,v) SORTfndfirst_lng(b,v)
#define SORTfndlast_wrd(b,v) SORTfndlast_lng(b,v)
#define SORTloop_wrd(b,p,q,tl,th,s) SORTloop_lng(b,p,q,tl,th,s)
#endif
#define SORTloop_bit(b,p,q,tl,th,s) SORTloop_chr(b,p,q,tl,th,s)
```

11.10 Common BAT Operations

Much used, but not necessarily kernel-operations on BATs.

11.10.1 BAT aggregates

BAT*	BAThistogram(BAT *b)
BAT*	BATsample(BAT* b,size_t n)

The routine *BAThistogram* produces a new BAT with a frequency distribution of the tail of its operand.

The routine *BATsample* returns a random sample on n BUNs of a BAT.

For each BAT we maintain its dimensions as separately accessible properties. They can be used to improve query processing at higher levels.

11.10.2 Alignment transformations

Some classes of algebraic operators transform a sequence in an input BAT always in the same way in the output result. An example are the () function (including histogram(b), which is identical to (b.reverse)). That is to say, if synced(b2,b2) => synced((b1),(b2))

Another example is b.fetch(position-bat). If synced(b2,b2) and the same position-bat is fetched with, the results will again be synced. This can be mimicked by transforming the *alignment-id* of the input BAT with a one-way function onto the result.

We use **output->halign = NOID_AGGR(input->halign)** for the **output = (input)** case, and **output->align = NOID_MULT(input1->align,input2->halign)** for the fetch.

11.10.3 BAT relational operators

BAT *	BATjoin (BAT *l, BAT *r, size_t estimate)
BAT *	BATouterjoin (BAT *l, BAT *r, size_t estimate)
BAT *	BATbandjoin (BAT *l, BAT *r, ptr c1, ptr c2)
BAT *	BATthetajoin (BAT *l, BAT *r, int mode, size_t estimate)
BAT *	BATsemijoin (BAT *l, BAT *r)
BAT *	BATselect (BAT *b, ptr tl, ptr th)
BAT *	BATfragment (BAT *b, ptr l, ptr h, ptr L, ptr H)
\hline	
BAT *	BATsunique (BAT *b)
BAT *	BATkunique (BAT *b)
BAT *	BATsunion (BAT *b, BAT *c)
BAT *	BATkunion (BAT *b, BAT *c)
BAT *	BATsintersect (BAT *b, BAT *c)
BAT *	BATkintersect (BAT *b, BAT *c)
BAT *	BATsdiff (BAT *b, BAT *c)
BAT *	BATkdiff (BAT *b, BAT *c)

The BAT library comes with a full-fledged collection of relational operators. The two selection operators *BATselect* and *BATfragment* produce a partial copy of the BAT. The former performs a search on the tail; the latter considers both dimensions. The *BATselect* operation takes two inclusive ranges as search arguments. Interpretation of a NULL argument depends on the position, i.e. a domain lower or upper bound.

The operation *BATsort* sorts the BAT on the header and produces a new BAT. A side effect is the clustering of the BAT store on the sort key.

The *BATjoin* over R[A, B] and S[C, D] performs an equi-join over B and C. It results in a BAT over A and D. The *BATouterjoin* implements a left outerjoin over the BATs involved. The *BATbandjoin* produces the associations [A, D] such that S.C-c1 <= R.b <= S.C + c2. The special case c1 = 0 and c2 = infinite leads to a thetajoin. The *BATsemijoin* over R[A, B] and S[C, D] produces the subset of R[A, B] that satisfies the semijoin over A and C.

The full-materialization policy intermediate results in MonetDB means that a join can produce an arbitrarily large result and choke the system. The Data Distilleries tool therefore first computes the join result size before the actual join (better waste time than crash the server). To exploit that perfect result size knowledge, an result-size estimate parameter was added to all equi-join implementations. TODO: add this for semijoin/select/unique/diff/intersect

The routine *BATsunique* considers both dimensions in the double elimination it performs; it produces a set. The routine *BATtunique* considers only the head column, and produces a unique head column.

BATs that satisfy the set property can be further processed with the set operations *BATsunion*, *BATsintersect*, and *BATsdiff*. The same operations are also available in versions that only look at the head column:*BATkunion*, *BATkdiff*, and *BATkintersect* (which shares its implementation with *BATsemijoin*).

The kernel code modules are encapsulated with MAL wrappers. A synopsis of their functionality is described below. The signature details can be found in the appendix.

11.11 Aggregates Module

This module contains some efficient aggregate functions that compute their result in one scan, rather than in the iterative manner of the generic MIL aggregrate implementations.

The implementation code is derived from the original 'aggr' module. It uses a complete type-specific code expansion to avoid any type-checking in the inner-most loops. Where feasible, it replaced (expansive) hash-lookup by significantly cheaper positional void-lookups (if the head-column of the group-extend BAT (`"e"`) is `"void"`) or at least by (also positional) array lookups (in case the group-ids span a reasonably small range);

In addition to the 2-parameter

11.12 Timers and Timed Interrupts

This module handles various signaling/timer functionalities. The Monet interface supports two timer commands: *alarm* and *sleep*. Their argument is the number of seconds to wait before the timer goes off. The *sleep* command blocks till the alarm goes off. The *alarm* command continues directly, executes off a MIL string when it goes off. The parameterless routines *time* and *ctime* provide access to the cpu clock.They return an integer and string, respectively.

11.13 BAT Algebra

This modules contains the most common algebraic BAT manipulation commands. We call them *algebra*, because all operations take values as parameters, and produce new result values, but *do not modify their parameters*. Unlike the previous Monet versions, we reduce the number of functions returning a BAT reference. This was previously needed to simplify recursive bat-expression and manage reference counts. In the current version we return only a BAT identifier when a new bat is being created.

All parameters to the modules are passed by reference. In particular, this means that string values are passed to the module layer as (str *) and we have to de-reference them before entering the gdk library. This calls for knowlegde on the underlying BAT typs's

11.14 Basic array support

The array support library constructs the index arrays essential for the Relational Algebra Model language. The grid filler operation assumes that there is enough space. The shift variant multiplies all elements with a constant factor. It is a recurring operation for the RAM front-end and will save an additional copying.

The optimization is captured in a contraction macro.

11.15 Binary Association Tables

This module contains the commands and patterns to manage Binary Association Tables (BATs). The relational operations you can execute on BATs have the form of a neat algebra, described in algebra.mx

But a database system needs more that just this algebra, since often it is crucial to do table-updates (this would not be permitted in a strict algebra).

All commands needed for BAT updates, property management, basic I/O, persistence, and storage options can be found in this module.

All parameters to the modules are passed by reference. In particular, this means that string values are passed to the module layer as (str *) and we have to de-reference them before entering the gdk library. (Actual a design error in gdk to differentiate passing int/str) This calls for knowledge on the underlying BAT types's

11.16 BAT calculator

Many applications require extension of the basic calculator and mathematic functions to work on BAT arguments. Although the MAL multiplex optimizer contains a command ('optimizer.multiplex') to generate the necessary code, it is often much more efficient to use one of the dedidacted batcalc functions introduced below.

The operators supported are limited to the built-in fixed length atoms, because they permit ease of storage of the operation result. Variable sized atoms, especially user defined, may require more administrative activities. Furthermore, the operands involved are assumed to be aligned to assure the fastest possible join evaluation.

Optimal processing performance is further obtained when the operators can work as 'accumulators', for then we do not pay the price of space allocation for a new intermediate. It may imply a BATcopy before the accummulator function is being called. A new BAT is of course created when the result of a function does not fit the accumulator.

The implementation does not take into account possible overflows caused by the operators. However, the operators respect the NIL semantics and division by zero produces a NIL.

In addition to arithmetic and comparison operators, casting and mathematical functions are directly supported.

11.17 NULL semantics

The batcalc arithmetic is already constraint to BATs of equal size. Another improvement can be obtained when we do not have to check for NULLs in each and every basic operation +,-,/,* and comparisons. This variant can be obtained using a compile time flag (NULLTST) for the time being. If it turns out to be effective, we will derive a new version of batcalc. Experiments show that ignoring the NULLS saves about 15\% for larger instructions.

11.18 BAT Coercion Routines

The coercion routines over BATs can not easily be speed up using an accumulator approach, because they often require different storage space. Nevertheless, the implementation provided here are much faster compared to the Version 4.* implementation.

The coercion routines are build for speed. They are not protected against overflow.

11.19 BAT if-then-else multiplex expressions.

The assembled code for IF-THEN-ELSE multiplex operations. Again we assume that the BAT arguments are aligned.

11.20 Color multiplexes

[TODO: property propagations and general testing] The collection of routines provided here are map operations for the color string primitives.

In line with the batcalc module, we assume that if two bat operands are provided that they are already aligned on the head. Moreover, the head of the BATs are limited to :oid, which can be cheaply realized using the GRPsplit operation.

11.21 String multiplexes

[TODO: property propagations] The collection of routines provided here are map operations for the atom string primitives.

In line with the batcalc module, we assume that if two bat operands are provided that they are already aligned on the head. Moreover, the head of the BATs are limited to :void, which can be cheaply realized using the GRPsplit operation.

11.22 BAT math calculator

11.23 The math module

This module contains the math commands. The implementation is very simply, the c math library functions are called. See for documentation the ANSI-C/POSIX manuals of the equaly named functions.

NOTE: the operand itself is being modified, rather than that we produce a new BAT. This to save the expensive copying.

11.24 Time/Date multiplexes

[TODO: property propagations] The collection of routines provided here are map operations for the atom time and date primitives.

In line with the batcalc module, we assume that if two bat operands are provided that they are already aligned on the head. Moreover, the head of the BATs are limited to :void, which can be cheaply realized using the GRPsplit operation.

11.25 Basic arithmetic

This module is an extended version of the V4 arithmetic module. It implements the arithmetic operations on the built-in types, *chr*, *bte*, *sht*, *int*, *flt*, *dbl* and *lng*. All combinations are implemented. Limited combinations are implemented for *bit*, *oid* and *str*.

`[binary operators]`
: The implemented operators are first of all all comparison that return a TRUE/FALSE value (*bit* values), i.e. `<=`, `<`, *==*, *!=*, *>=*, and *>=*.

  The module also implements the operators `+`, `-`, *** and */*. The rules for the return types operators is as follows. If one of the input types is a floating point the result will be a floating point. The largest type of the input types is taken.

  The *max* and *min* functions return the maximum and minimum of the two input parameters.

`[unary operators]`
: This module also implements the unary *abs*() function, which calculates the absolute value of the given input parameter, as well as the `-` unary operator.

  The *inv* unary operation calculates the inverse of the input value. An error message is given when the input value is zero.

`[bitwise operators]`
: For integers there are some additional operations. The \% operator implements the congruent modulo operation. The `<<` and `>>` are the left and right bit shift. The *or*, *and*, *xor* and *not* for integers are implemented as bitwise boolean operations.

`[boolean operators]`
: The *or*, *and*, *xor* and *not* for the bit atomic type in MIL (this corresponds to what is normally called boolean) are implemented as the logic operations.

`[random numbers]`
: This module also contains the rand and srand functions. The *srand*() function initializes the random number generator using a seed value. The subsequent calls to *rand*() are pseudo random numbers (with the same seed the sequence can be repeated).

The general interpretation for the NIL value is `"unknown"`. This semantics mean that any operation that receives at least one NIL value, will produce a NIL value in the output for sure.

The only exception to this rule are the `"=="` and `"!="` equality test routines (it would otherwise become rather difficult to test whether a value is nil).

11.26 Performance Counters

This is a memory/cpu performance measurement tool for the following processor (families).

- MIPS R10000/R12000 (IP27)
- Sun UltraSparcI/II (sun4u)
- Intel Pentium (i586/P5)
- Intel PentiumPro/PentiumII/PentiumIII/Celeron (i686/P6)

- AMD Athlon (i686/K7)
- Intel Itanium/Itanium2 (ia64)

It uses

- libperfmon libperfex (IRIX) for R10000/R12000,
- (Solaris <= 7) by Richard Enbody, libcpc (Solaris >= 8) for UltraSparcI/II,
- libpperf (Linux-i?86 <= 2.2), by M. Patrick Goda and Michael S. Warren,
- libperfctr (Linux-i?86 >= 2.4), by M. Pettersson for Pentiums \& Athlons.
- libpfm (Linux-ia64 >= 2.4), by HP for Itanium[2].

Module counters provides similar interface and facilities as Peter's R10000 perfex module, but it offers no multiplexing of several events; only two events can be monitored at a time. On non-Linux/x86, non-Solaris/UltraSparc, and non-IRIX/R1x000 systems, only the elapsed time in microseconds is measured.

11.27 The group module

This module contains the primitives to construct, derive, and perform statistical operations on BATs representing groups. The default scheme in Monet is to assume the head to represent the group identifier and the tail an element in the group.

Groups play an important role in datamining, where they are used to construct cross-tables. Such cross tables over a single BAT are already supported by the histogram function. This module provides extensions to support identification of groups in a (multi-)dimensional space.

The module implementation has a long history. The first implementation provided several alternatives to produce/derive the grouping. A more complete (and complex) scheme was derived during its extensive use in the context of the Data Distilleries product. The current implementation is partly a cleanup of this code-base, but also enables provides better access to the intermediate structures produced in the process, i.e. the histogram and the sub-group mapping. They can be used for various optimization schemes at the MAL level.

The prime limitation of the current implementation is that an underlying database of BATs is assumed. This enables representation of each group using an oid, and the value representation of the group can be accordingly be retrieved easily. An optimized implementation in which we use positional integer id's (as embodied by Monet's void type) is also available.

This limitation on (v)oid-headers is marginal. The primitive GRPsplit produces for any BAT two copies with both a (v)oid header.

11.27.1 Algorithms

There are several approaches to build a cross table. The one chosen here is aimed at incremental construction, such that re-use of intermediates becomes possible. Starting with the first dimension, a BAT is derived to represent the various groups, called a *GRP BAT* or cross-table BAT.

11.27.2 Cross Table (GRP)

A cross table is an <oid,oid> BAT where the first (head) denotes a tuple in the cross table and the second (tail) marks all identical lists. The tail-oids contain group identifiers; that is, *this value is equal* **iff** *two tuples belong to the same group.* The group identifiers are chosen from the domain of the tuple-identifiers. This simplifies getting back to the original tuples, when talking about a group. If the tuple-oid of 'John' is chosen as a group-id, you might view this as saying that each member of the group is 'like John' with respect to the grouping-criterion.

Successively the subgroups can be identified by modifying the GRP BAT or to derive a new GRP BAT for the subgroups. After all groups have been identified this way, a BAT histogram operation can be used to obtain the counts of each data cube. Other aggregation operations using the MIL set aggregate construct (bat) can be used as well; note for instance that histogram == (b.reverse()).

The Monet interface module specification is shown below. Ideally we should defined stronger type constraints, e.g. command group.new(attr:bat[,:any_1]

The group macro is split along three dimensions:

[type:]	Type specific implementation for selecting the right hash function and data size etc.;
[clustered:]	The select the appropriate algorithm, i.e., with or without taking advantage of an order of values in the parent groups;
[physical properties:]	Values , choosing between a fixed predefined and a custom hashmask. Custom allows the user to determine the size of the hashmask (and indirectly the estimated size of the result). The hashmask is $2^n - 1$ where n is given by the user, or 1023 otherwise, and the derived result size is $4 \cdot 2^n$.

Further research should point out whether fitting a simple statistical model (possibly a simple mixture model) can help choose these parameters automatically; the current idea is that the user (which could be a domain-specific extension of the higher-level language) knows the properties of the data, especially for IR in which the standard grouping settings differ significantly from the original datamining application.

11.28 Lightweight Lock Module

This module provides simple SMP lock and thread functionality as already present in the MonetDB system.

This module provides simple SMP lock and thread functionality as already present in the MonetDB system.

11.29 The Transaction Logger

In the philosophy of MonetDB, transaction management overhead should only be paid when necessary. Transaction management is for this purpose implemented as a separate module and applications are required to obey the transaction policy, e.g. obtaining/releasing locks.

This module is designed to support efficient logging of the SQL database. Once loaded, the SQL compiler will insert the proper calls at transaction commit to include the changes in the log file.

The logger uses a directory to store its log files. One master log file stores information about the version of the logger and the transaction log files. This file is a simple ascii file with the following format: 6DIGIT-VERSION\n[log file number \n]*]* The transaction log files have a binary format, which stores fixed size logformat headers (flag,nr,bid), where the flag is the type of update logged. The nr field indicates how many changes there were (in case of inserts/deletes). The bid stores the bid identifier.

The key decision to be made by the user is the location of the log file. Ideally, it should be stored in fail-safe environment, or at least the log and databases should be on separate disk columns.

This file system may reside on the same hardware as the database server and therefore the writes are done to the same disk, but could also reside on another system and then the changes are flushed through the network. The logger works under the assumption that it is called to safeguard updates on the database when it has an exclusive lock on the latest version. This lock should be guaranteed by the calling transaction manager first.

Finding the updates applied to a BAT is relatively easy, because each BAT contains a delta structure. On commit these changes are written to the log file and the delta management is reset. Since each commit is written to the same log file, the beginning and end are marked by a log identifier.

A server restart should only (re)process blocks which are completely written to disk. A log replay therefore ends in a commit or abort on the changed bats. Once all logs have been read, the changes to the bats are made persistent, i.e. a bbp sub-commit is done.

11.30 Multi-Attribute Equi-Join

11.31 Priority queues

This module includes functions for accessing and updating a pqueue. A pqueue is an (oid,any) bat. The tail is used as a comparison key. The first element of the pqueue is the smallest one in a min-pqueue or the largest one in a max-pqueue. Each element is larger than (smaller than) or equal to its parent which is defined by (position/2) if position is odd or (position-1)/2 if position is even (positions are from 0 to n-1). The head of the bat is used to keep track of the object-ids which are organized in the heap with respect to their values (tail column).

11.32 System state information

This document introduces a series of bats and operations that provide access to information stored within the Monet Version 5 internal data structures. In all cases, pseudo BAT operation returns a transient BAT that should be garbage collected after being used.

The main performance drain would be to use a pseudo BAT directly to successively access it components. This can be avoided by first assigning the pseudo BAT to a variable.

11.33 Unix standard library calls

The unix module is currently of rather limited size. It should include only those facilities that are UNIX specific, i.e. not portable to other platforms. Similar modules may be defined for Windows platforms.

12 SQL/XML

This section is under development

The SQL/XML standard defines the mechanisms to produce XML formatted results from relational queries and to import relational data from XML documents. The SQL/XML 2003 definition and its enhancements in SQL/XML 2005 are the frame of reference.

The implementation in MonetDB/SQL is initially geared at providing the basic functionality for publishing XML and simple querying. A more complete and optimized implementation of XQuery is already available with MonetDB/XQuery. However, data between the two systems is not shared. They are managed by a different server implementation.

SQL/XML introduces a new datatype `xml`. It can be used as a column type in table, view definitions, parameters, and variables. The datatype tells the system that values are properly structured XML objects.

The SQL/XML implementation uses the `string` type as the carrier for XML values and relies on the widely available library LIBXML2 for additional functionality. It is a poor man's approach compared to MonetDB/XQuery.

12.1 XML Import

XML data can be inserted into a database column using any of the available APIs. A single document is added to a XML column using the blob operation `file()`.

```
insert into dossier(id,doc) values(20070831, file("/tmp/letter"));
```

Often an XML document contains a heterogenous collection of objects, which should be broken into pieces before they are stored in the database. There are many ways to shred a document and often an application specific front-end application is needed. However, MonetDB SQL/XML provides a few simple operations to cope with the majority of situations.

```
copy into database from '/tmp/jacktheripper.xml';
copy into victim(name,hair) from '/tmp/jacktheripper.xml'
delimiter 'victim(name,hair)'
```

The first statements reads the XML document and breaks it into a series of relational tables with foreign key references. It results in a structured object representation. A XML view named after the top level element is automatically defined to rebuild the original document.

The second example performs a top-down parse of the XML document. Within every victim element it extracts the sub-trees named `name` and `hair`. This scheme is equivalent to using XPath expressions VICTIM[//NAME,//HAIR].

12.2 XML Publishing

The XML publishing functions are designed to construct XML values from data stored in the database. They can be used anywhere a string value is allowed.

```
XML_value_expression : XML_primary
XML_primary : value expression primary
    | XML_value_function

XML_value function : XML_comment
    | XML_concatenation
    | XML_element
    | XML_forest
    | XML_parse
    | XML_serialize
    | XML_PI

XML_comment : XMLCOMMENT '(' <string_value_expression> ')'

XML_concatenation : XMLCONCAT '(' XML_value_expression
    { ',' XML_value_expression }... ')'

XML_element : XMLELEMENT '(' NAME identifier
    [ ',' XML_namespace_declaration ] [ ',' XML_attributes ]
    [ { ',' XML_element_content }... [ OPTION XML_content_option]
    [ XML_returning_clause ] ')'
XML_attributes : XMLATTRIBUTES '(' XML_attribute_list ')'
XML_attribute_list : XML_attribute [ { ',' XML_attribute }... ]
XML_attribute : value_expression [ AS identifier ]
XML_content_option : NULL ON NULL
    | EMPTY ON NULL
    | ABSENT ON NULL
    | NIL ON NULL
| NIL ON NO CONTENT
XML_element_content : value_expression
XML_returning_clause :
    RETURNING { CONTENT | SEQUENCE }

XML_forest : XMLFOREST '(' [ XML_namespace_declaration> ',' ]
    forest_element list ')'
forest_element list : forest_element [ { ',' forest_element }... ]
forest_element : forest_element_value [ AS identifier ]
forest_element value : value_expression

XML_aggregate : XMLAGG '(' XML_value_expression
    [ ORDER BY sort_specification_list ]
    [ XML_returning_clause ] ')'

XML_PI : XMLPI '(' NAME identifier
    [ ',' string_value_expression ] ')'

XML_parse : XMLPARSE '(' document_or_content_string_value_expression
[ XML_whitespace_option ] ')'
XML_whitespace_option : { PRESERVE | STRIP } WHITESPACE

XML_serialize:
XMLSERIALIZE '(' {DOCUMENT | CONTENT} value AS data_type
    [ VERSION string_literal ]
    [ ENCODING SQL_language_identifier]
    [ [INCLUDING | EXCLUDING] XMLDECLARATIONS] ')'

XML_document predicate : XML_value_expression IS [ NOT ] DOCUMENT
```

The XML table construct provides a mechanism to extract a table from an XML document using simple path expression to the elements of interest.

```
XML_iterate : XMLITERATE '(' XML_value_expression ')'

XML_table : XMLTABLE '('
    [ XML_namespace_declaration ',' ]
    XML_table_row_pattern
    [ XML_table_argument_list ]
    COLUMNS XML_table_column_definitions ')'

XML_table_row_pattern :
    character_string_literal

XML_table_argument_list :
    PASSING XML_table_argument_passing_mechanism

XML_query_argument
    [ { ',' XML_query_argument }... ]

XML_table_argument_passing mechanism : XML_passing_mechanism

XML_table_column_definitions :
XML_table_column_definition
[ { ',' XML_table_column_definition }... ]

XML_table_column_definition_:
    XML_table_ordinality_column_definition
    | XML_table_regular_column_definition

XML_table_ordinality_column_definition :
column_name FOR ORDINALITY

XML_table_regular_column_definition :
   column_name_data_type [ XML_passing_mechanism ]
   [ default_clause ]
   [ PATH XML_table_column_pattern ]

XML_table_column_pattern : character_string_literal
```

Declare one or more XML namespaces and the encoding to use for binary strings.

```
XML_lexically_scoped_options :
    XML_lexically_scoped _option [ ',' XML_lexically_scoped_option ]
XML_lexically_ scoped_option : XML_namespace_declaration
    | XML_binary_encoding
XML_namespace_declaration :
    XMLNAMESPACES '(' XML_namespace_declaration item
    [ { ',' XML_namespace_declaration_item }... ] ')'
XML_namespace_declaration_item :
    XML_regular_namespace_declaration_item
    | XML_default_namespace_declaration_item
XML_namespace_prefix :
    identifier
XML_namespace_URI :
    character_string_literal
XML_regular_namespace_declaration_item :
    XML_namespace URI AS XML_namespace_prefix
XML_default_namespace_declaration_item :
    DEFAULT XML_namespace_URI
    | NO DEFAULT
XML_binary_encoding :
    XMLBINARY [ USING ] { BASE64 | HEX }
```

12.3 XPath and XQuery

The SQL/XML module aims to support XQuery through a well-defined (and narrow) interface.

It relies on linkage of the libxml2 library to explore the rich world of XPath processing.

12.4 XML Schema

Indicate a registered XML Schema, and (optionally) an XML namespace of that registered XML Schema, and (optionally) a global element declaration schema component of that registered XML Schema.

```
XML_valid_according_to_clause:
    ACCORDING_TO_XMLSCHEMA_XML_valid_according_to_what
    [_XML_valid_element_clause_]
XML_valid_according_to_what:
    XML_valid_according_to_URI
    |_XML_valid_according_to_identifier
XML_valid_according_to_URI:
    URI_XML_valid_target_namespace_URI_[_<XML_valid_schema_location>_]
    |_NO_NAMESPACE_[_XML_valid_schema_location_]
XML_valid_target_namespace_URI: XML_URI
XML_URI: character_string_literal
XML_valid_schema_location:
    LOCATION_XML_valid_schema_location_URI
XML_valid_schema_location_URI: XML_URI
XML_valid_according_to_identifier:
    ID_registered_XML_Schema_name
XML_valid_element_clause:
    XML_valid_element_name_specification
    |_XML_valid_element_namespace_specification
    [_XML_valid_element_name_specification_]
XML_valid_element_name_specification:
    ELEMENT_XML_valid_element_name
XML_valid_element_namespace_specification:
    NO NAMESPACE
    | NAMESPACE <XML valid element namespace URI>
```

```
XML_character_string_serialization:
    XMLSERIALIZE '(' [ document_or_content ]
    XML_value_expression_AS_data_type
    [ XML_serialize_version ]
    [ XML_declaration_option ] ')'
XML_declaration_option:
    INCLUDING XMLDECLARATION
    |_EXCLUDING XMLDECLARATION
document_or_content: DOCUMENT | CONTENT
XML_serialize_version:
    VERSION character_string_literal
blob_value_function:
|_XML_binary_string_serialization
XML_binary_string_serialization:
XMLSERIALIZE'(' [_document_or_content ]
    XML_value_expression_AS_data_type
    [ ENCODING_XML_encoding_specification ]
    [ XML_serialize_version ]
    [ XML_declaration_option ] right_paren
XML_encoding_specification:
    XML_encoding_name
XML_encoding_name:
    SQL_language_identifier
```

Global variables are used to control choices.

```
SET XML OPTION { DOCUMENT | CONTENT };
```

12.4.1 Spatial support

This section is under development

MonetDB/SQL comes with an interface to the Simple Feature Specification of OpenGIS which opens the route to develop GIS applications

The MonetDB/SQL/GIS module supports all objects and functions specified in the OGC "Simple Features for SQL" specification. Spatial objects can, however, for the time being only expressed in the Well-Known Text (WKT) format. WKT includes information about the type of the object and the object's coordinates.

12.4.2 Get Going

We assume you have an sql client connected to a MonetDB server with the spatial extension being loaded. How to do this is described in the section 'Installation' below.

An illustrative example script is given below. It creates and populates a 'forest' table and a 'buildings' table followed by a spatial query in this fictive landscape.

```
CREATE TABLE forests(id INT,name TEXT,shape MULTIPOLYGON);
CREATE TABLE buildings(id INT,name TEXT,location POINT,outline POLYGON);

INSERT INTO forests VALUES(109, 'Green Forest',
'MULTIPOLYGON( ((28 26,28 0,84 0,84 42,28 26), (52 18,66 23,73 9,48 6,52 18)), ((59 18,67 1

INSERT INTO buildings VALUES(113, '123 Main Street',
        'POINT( 52 30 )',
        'POLYGON( ( 50 31, 54 31, 54 29, 50 29, 50 31) )');
INSERT INTO buildings VALUES(114, '215 Main Street',
        'POINT( 64 33 )',
        'POLYGON( ( 66 34, 62 34, 62 32, 66 32, 66 34) )');

SELECT forests.name,buildings.name
FROM forests,buildings
WHERE forests.name = 'Green Forest' and
    Overlaps(forests.shape, buildings.outline) = true;
```

12.4.3 Acceleration Spatial Operations

There are no special accelerators to speed up access to Spatial Objects yet. However, we can use the Minimum Bounding Rectangle (mbr) datatype to accelerate operations considerably. This requires a small query rewrite. In the example above the performance of the query can be improved in the following manner:

```
ALTER TABLE forests ADD bbox mbr;
UPDATE forests SET bbox = mbr(shape);
ALTER TABLE buildings ADD bbox mbr;
UPDATE forests SET bbox = mbr(outline);

SELECT forests.name,buildings.name
FROM forests,buildings
WHERE forests.name = 'Green Forest' AND
```

```
    mbroverlaps(forests.bbox,buildings.bbox) = TRUE AND
    Overlaps(forests.shape, buildings.outline) = TRUE;
```

In this way the mbr operation acts as a filter. Upon request, and availabilty of resources, we will develop MAL optimizers to automate this process.

12.4.4 Installation

The GIS functionality is packaged as a separate MonetDB module called `geom`. It requires installation of MonetDB/SQL version 2.20 and up.

To benefit from the geometry functionality you first have to download and install `GEOS` from http://geos.refractions.net/. It is a well-known and sound library to built upon.

The next step is to check out the MonetDB/SQL geom module from our Sourceforge location and install it following the guidelines for MonetDB/SQL.

Once the geom module has been installed you can start the MonetDB/SQL server with geometry support using the following command:

```
mserver5 --dbinit="include geom;include sql;sql.start();"
```

Now on a client run the geom initialization script once:

```
mclient -lsq < [PATH]/src/sql/geom.sql
```

[NIELS/WILKO This should be simplified. Can't we make it part of the sql startup scripts] Please note that running this script changes the database in such a way that from now on the server should always be started with the geom module included.

How about the binary installations? Can we prepackage the stuff? or at least make sure that the SQL interface is accepted, even if when geos functions can not be bound.

12.4.5 Bulk loading

The MonetDB distribution includes the program 'shp2monetdb' to convert a shapefile into SQL insert statements. This is a port from the shp2pgsql program that is shipped with PostGIS.

The functionality has been tested with OpenJUMP, an open-source Geographic Information System application available at http://openjump.org/wiki/show/HomePage.

12.4.6 Limitations

This is the first implementation of OpenGIS functionality in MonetDB. Many issues require our attention, but priority will be derived from concrete external requests and availability of manpower. The shortlist of open issues is:

- development of a JDBC extension to map the geometry datatypes to their Java counterparts.
- support for 3D types.
- spatial optimizers in the MAL optimizer toolkit.

12.4.7 Spatial Types

MonetDB supports the OpenGIS types: Point, Curve, LineString, Surface, Polygon, MultiPoint, MultiCurve, MultiLineString, MultiSurface, MultiPolygon, Geometry and GeomCollection.

One non-OpenGIS type for fast access is used. This type 'mbr' is used for storing a 2D box. Functions to create these boxes are specified in following sections.

12.4.8 SQL functions on spatial objects

12.4.8.1 Functions on mbr's

The following functions return the mbr of a given geometry:

```
CREATE FUNCTION mbr (p Point) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (c Curve) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (l LineString) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (s Surface) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (p Polygon) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (m multipoint) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (m multicurve) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (m multilinestring) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (m multisurface) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (m multipolygon) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (g Geometry) RETURNS mbr external name geom.mbr;
CREATE FUNCTION mbr (g GeomCollection) RETURNS mbr external name geom.mbr;
```

This function returns true iff two mbrs overlap:

```
CREATE FUNCTION mbroverlaps(a mbr, b mbr) RETURNS BOOLEAN external name geom."mbroverlaps";
```

12.4.8.2 Conversion from and to Well-known Text

Convert a Well-Known-Text string to a spatial object. The SRID parameter is a reference to the Spatial Reference System in which the coordinates are expressed.

```
CREATE FUNCTION GeomFromText(wkt string, srid SMALLINT) RETURNS Geometry external name geom
CREATE FUNCTION PointFromText(wkt string, srid SMALLINT) RETURNS Point external name geom."
CREATE FUNCTION LineFromText(wkt string, srid SMALLINT) RETURNS LineString external name ge
CREATE FUNCTION PolyFromText(wkt string, srid SMALLINT) RETURNS Polygon external name geom.
CREATE FUNCTION MPointFromText(wkt string, srid SMALLINT) RETURNS MultiPoint external name
CREATE FUNCTION MLineFromText(wkt string, srid SMALLINT) RETURNS MultiLineString external n
CREATE FUNCTION MPolyFromText(wkt string, srid SMALLINT) RETURNS MultiPolygon external name
CREATE FUNCTION GeomCollectionFromText(wkt string, srid SMALLINT) RETURNS MultiPolygon exte
-- alias
CREATE FUNCTION PolygonFromText(wkt string, srid SMALLINT) RETURNS Polygon external name ge
```

Return Well-know Text representation of spatial objects.

```
CREATE FUNCTION AsText(p Point) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(c Curve) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(l LineString) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(s Surface) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(p Polygon) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(p MultiPoint) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(c MultiCurve) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(l MultiLineString) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(s MultiSurface) RETURNS STRING external name geom."AsText";
```

```
CREATE FUNCTION AsText(p MultiPolygon) RETURNS STRING external name geom."AsText";
CREATE FUNCTION AsText(g Geometry) RETURNS STRING external name geom."AsText";
```

12.4.8.3 Analysis functions on Geometry

The following functions perform analysis operations on geometries:

```
CREATE FUNCTION Area(g Geometry) RETURNS FLOAT external name geom."Area";
CREATE FUNCTION Length(g Geometry) RETURNS FLOAT external name geom."Length";
CREATE FUNCTION Distance(a Geometry, b Geometry) RETURNS FLOAT external name geom."Distance
CREATE FUNCTION Buffer(a Geometry, distance FLOAT) RETURNS Geometry external name geom."Buf
CREATE FUNCTION ConvexHull(a Geometry) RETURNS Geometry external name geom."ConvexHull";
CREATE FUNCTION Intersection(a Geometry, b Geometry) RETURNS Geometry external name geom."I
CREATE FUNCTION "Union"(a Geometry, b Geometry) RETURNS Geometry external name geom."Union"
CREATE FUNCTION Difference(a Geometry, b Geometry) RETURNS Geometry external name geom."Dif
CREATE FUNCTION SymDifference(a Geometry, b Geometry) RETURNS Geometry external name geom."

CREATE FUNCTION Dimension(g Geometry) RETURNS integer external name geom."Dimension";
CREATE FUNCTION GeometryTypeId(g Geometry) RETURNS integer external name geom."GeometryType
CREATE FUNCTION SRID(g Geometry) RETURNS integer external name geom."SRID";
CREATE FUNCTION Envelope(g Geometry) RETURNS Geometry external name geom."Envelope";
CREATE FUNCTION IsEmpty(g Geometry) RETURNS BOOLEAN external name geom."IsEmpty";
CREATE FUNCTION IsSimple(g Geometry) RETURNS BOOLEAN external name geom."IsSimple";
CREATE FUNCTION Boundary(g Geometry) RETURNS Geometry external name geom."Boundary";

CREATE FUNCTION Equals(a Geometry, b Geometry) RETURNS BOOLEAN external name geom."Equals";
CREATE FUNCTION Disjoint(a Geometry, b Geometry) RETURNS BOOLEAN external name geom."Disjoi
CREATE FUNCTION "Intersect"(a Geometry, b Geometry) RETURNS BOOLEAN external name geom."Int
CREATE FUNCTION Touches(a Geometry, b Geometry) RETURNS BOOLEAN external name geom."Touches
CREATE FUNCTION Crosses(a Geometry, b Geometry) RETURNS BOOLEAN external name geom."Crosses
CREATE FUNCTION Within(a Geometry, b Geometry) RETURNS BOOLEAN external name geom."Within";
CREATE FUNCTION Contains(a Geometry, b Geometry) RETURNS BOOLEAN external name geom."Contai
CREATE FUNCTION Overlaps(a Geometry, b Geometry) RETURNS BOOLEAN external name geom."Overla
CREATE FUNCTION Relate(a Geometry, b Geometry, pattern STRING) RETURNS BOOLEAN external nam
```

12.4.8.4 SQL functions on Point

```
CREATE FUNCTION X(g Geometry) RETURNS double external name geom."X";
CREATE FUNCTION Y(g Geometry) RETURNS double external name geom."Y";

CREATE FUNCTION Point(x double,y double) RETURNS Point external name geom.point;
```

12.4.8.5 SQL functions on Curve

```
CREATE FUNCTION IsRing(l LineString) RETURNS BOOLEAN external name geom."IsRing";
CREATE FUNCTION StartPoint(l LineString) RETURNS Point external name geom."StartPoint"; --
CREATE FUNCTION EndPoint(l LineString) RETURNS Point external name geom."EndPoint"; -- not
```

12.4.8.6 SQL functions on LineString

```
CREATE FUNCTION NumPoints(l LineString) RETURNS integer external name geom."NumPoints"; --
CREATE FUNCTION PointN(l LineString,i integer) RETURNS Point external name geom."PointN"; -
```

12.4.8.7 SQL functions on Surface

```
CREATE FUNCTION PointOnSurface(s Surface) RETURNS Point external name geom."PointOnSurface"
CREATE FUNCTION Centroid(s Surface) RETURNS Point external name geom."Centroid"; -- not yet
```

12.4.8.8 SQL functions on Polygon

```
CREATE FUNCTION ExteriorRing(s Surface) RETURNS LineString external name geom."ExteriorRing
CREATE FUNCTION NumInteriorRing(s Surface) RETURNS integer external name geom."NumInteriorR
CREATE FUNCTION InteriorRingN(s Surface,n integer) RETURNS LineString external name geom."I
```

12.4.8.9 SQL functions on GeomCollection

```
-- Unimplemented Documentation
CREATE FUNCTION NumGeometries(GeomCollection c) RETURNS integer external name geom."NumGeom
CREATE FUNCTION GeometryN(GeomCollection c,n integer) RETURNS Geometry external name geom."
```

13 Application Programming Interfaces

MonetDB comes with a complete set of programming libraries. Their basis is the MonetDB application programming interface (Mapi), which describes the protocol understood by the server. The Perl, PHP, and Python libraries are mostly wrappers around the Mapi routines.

The programming interface is based on a client-server architecture, where the client program connects to a server using a TCP/IP connection to exchange commands and receives answers. The underlying protocol uses plain UTF-8 data for ease of use and debugging. This leads to publicly visible information exchange over a network, which may be undesirable. Therefore, a private and secure channel can be set up with the Secure Socket Layer functionality.

A more tightly connection between application logic and database server is described in Section 1.8.7 [Embedded Server], page 25.

13.1 The Mapi Library

The easiest way to extend the functionality of MonetDB is to construct an independent application, which communicates with a running server using a database driver with a simple API and a textual protocol. The effectiveness of such an approach has been demonstrated by the wide use of database API implementations, such as Perl DBI, PHP, ODBC,...

13.1.1 Sample MAPI Application

The database driver implementation given in this document focuses on developing applications in C. The command collection has been chosen to align with common practice, i.e. queries follow a prepare, execute, and fetch_row paradigm. The output is considered a regular table. An example of a mini application below illustrates the main operations.

```
#include <Mapi.h>
#include <stdio.h>
#include <stdlib.h>

#define die(dbh,hdl) (hdl?mapi_explain_query(hdl,stderr):               \
                          dbh?mapi_explain(dbh,stderr):                 \
                              fprintf(stderr,"command failed\n"),       \
                      exit(-1))

int main(int argc, char **argv)
{
    Mapi dbh;
    MapiHdl hdl = NULL;

    dbh = mapi_connect("localhost", 50000, "monetdb", "monetdb", "sql", NULL);
    if (mapi_error(dbh))
        die(dbh, hdl);

    if ((hdl = mapi_query(dbh, "create table emp(name varchar(20), age int)")) == NULL
        || mapi_error(dbh) != MOK)
```

```
            die(dbh, hdl);
        if (mapi_close_handle(hdl) != MOK)
            die(dbh, hdl);
        if ((hdl = mapi_query(dbh, "insert into emp values('John', 23)")) == NULL
            || mapi_error(dbh) != MOK)
            die(dbh, hdl);
        mapi_close_handle(hdl);
        if (mapi_error(dbh) != MOK)
            die(dbh, hdl);
        if ((hdl = mapi_query(dbh, "insert into emp values('Mary', 22)")) == NULL
            || mapi_error(dbh) != MOK)
            die(dbh, hdl);
        mapi_close_handle(hdl);
        if (mapi_error(dbh) != MOK)
            die(dbh, hdl);
        if ((hdl = mapi_query(dbh, "select * from emp")) == NULL
            || mapi_error(dbh) != MOK)
            die(dbh, hdl);

        while (mapi_fetch_row(hdl)) {
            char *nme = mapi_fetch_field(hdl, 0);
            char *age = mapi_fetch_field(hdl, 1);
            printf("%s is %s\n", nme, age);
        }
        if (mapi_error(dbh) != MOK)
            die(dbh, hdl);
        mapi_close_handle(hdl);
        if (mapi_error(dbh) != MOK)
            die(dbh, hdl);
        mapi_destroy(dbh);

        return 0;
    }
```

The MAPI_CONNECT() operation establishes a communication channel with a running server. The query language interface is either "sql", "mil" or "xquery".

Errors on the interaction can be captured using MAPI_ERROR(), possibly followed by a request to dump a short error message explanation on a standard file location. It has been abstracted away in a macro.

Provided we can establish a connection, the interaction proceeds as in many similar application development packages. Queries are shipped for execution using MAPI_QUERY() and an answer table can be consumed one row at a time. In many cases these functions suffice.

The Mapi interface provides caching of rows at the client side. MAPI_QUERY() will load tuples into the cache, after which they can be read repeatedly using MAPI_FETCH_ROW() or directly accessed (MAPI_SEEK_ROW()). This facility is particularly handy when small, but stable query results are repeatedly used in the client program.

To ease communication between application code and the cache entries, the user can bind the C-variables both for input and output to the query parameters, and output columns, respectively. The query parameters are indicated by '?' and may appear anywhere in the query template.

The Mapi library expects complete lines from the server as answers to query actions. Incomplete lines leads to Mapi waiting forever on the server. Thus formatted printing is discouraged in favor of tabular printing as offered by the TABLE.PRINT() commands.

The following action is needed to get a working program. Compilation of the application relies on the *monetdb-config* program shipped with the distribution. It localizes the include files and library directories. Once properly installed, the application can be compiled and linked as follows:

```
cc sample.c `monetdb-clients-config --cflags --libs` -lMapi -o sample
./sample
```

It assumes that the dynamic loadable libraries are in public places. If, however, the system is installed in your private environment then either one of the two following options can be used.

```
cc sample.c `monetdb-clients-config --cflags --libs` -lMapi -o sample \
-Wl,rpath,`monetdb-clients-config --libdir`
./sample
export LD_LIBRARY_PATH=`monetdb-clients-config --libdir`
cc sample.c `monetdb-clients-config --cflags --libs` -lMapi -o sample
./sample
```

The compilation on Windows is slightly more complicated. It requires more attention towards the location of the include files and libraries.

13.1.2 Command Summary

The quick reference guide to the Mapi library is given below. More details on their constraints and defaults are given in the next section.

mapi_bind()	Bind string C-variable to a field
mapi_bind_numeric()	Bind numeric C-variable to field
mapi_bind_var()	Bind typed C-variable to a field
mapi_cache_freeup()	Forcefully shuffle fraction for cache refreshment
mapi_cache_limit()	Set the tuple cache limit
mapi_cache_shuffle()	Set shuffle fraction for cache refreshment
mapi_clear_bindings()	Clear all field bindings
mapi_clear_params()	Clear all parameter bindings
mapi_close_handle()	Close query handle and free resources
mapi_connect()	Connect to a mserver
mapi_connect_ssl()	Connect to a mserver using Secure Socket Layer (SSL)
mapi_destroy()	Free handle resources
mapi_disconnect()	Disconnect from server
mapi_error()	Test for error occurrence
mapi_execute()	Execute a query
mapi_execute_array()	Execute a query using string arguments
mapi_explain()	Display error message and context on stream

mapi_explain_query()	Display error message and context on stream
mapi_fetch_all_rows()	Fetch all answers from server into cache
mapi_fetch_field()	Fetch a field from the current row
mapi_fetch_field_array()	Fetch all fields from the current row
mapi_fetch_line()	Retrieve the next line
mapi_fetch_reset()	Set the cache reader to the beginning
mapi_fetch_row()	Fetch row of values
mapi_finish()	Terminate the current query
mapi_get_dbname()	Database being served
mapi_get_field_count()	Number of fields in current row
mapi_get_host()	Host name of server
mapi_get_language()	Query language name
mapi_get_mapi_version()	Mapi version name
mapi_get_monet_versionId()	MonetDB version identifier
mapi_get_monet_version()	MonetDB version name
mapi_get_motd()	Get server welcome message
mapi_get_row_count()	Number of rows in cache or -1
mapi_get_trace()	Get trace flag
mapi_get_user()	Current user name
mapi_next_result()	Go to next result set
mapi_log()	Activate logging file client/server interaction
mapi_ping()	Test server for accessibility
mapi_prepare()	Prepare a query for execution
mapi_prepare_array()	Prepare a query for execution using arguments
mapi_query()	Send a query for execution
mapi_query_array()	Send a query for execution with arguments
mapi_query_handle()	Send a query for execution
mapi_quick_query_array()	Send a query for execution with arguments
mapi_quick_query()	Send a query for execution
mapi_quick_response()	Quick pass response to stream
mapi_quote()	Escape characters
mapi_reconnect()	Reconnect with a clean session context
mapi_rows_affected()	Obtain number of rows changed
mapi_seek_row()	Move row reader to specific location in cache
mapi_setAutocommit()	Set auto-commit flag
mapi_stream_query()	Send query and prepare for reading tuple stream
mapi_table()	Get current table name
mapi_timeout()	Set timeout for long-running queries[TODO]
mapi_trace()	Set trace flag
mapi_virtual_result()	Submit a virtual result set
mapi_unquote()	remove escaped characters

13.1.3 Mapi Library

The routines to build a MonetDB application are grouped in the library MonetDB Programming Interface, or shorthand Mapi.

The protocol information is stored in a Mapi interface descriptor (mid). This descriptor can be used to ship queries, which return a MapiHdl to represent the query answer. The application can set up several channels with the same or a different `mserver`. It is the programmer's responsibility not to mix the descriptors in retrieving the results.

The application may be multi-threaded as long as the user respects the individual connections represented by the database handlers.

The interface assumes a cautious user, who understands and has experience with the query or programming language model. It should also be clear that references returned by the API point directly into the administrative structures of Mapi. This means that they are valid only for a short period, mostly between successive MAPI_FETCH_ROW() commands. It also means that it the values are to retained, they have to be copied. A defensive programming style is advised.

Upon an error, the routines MAPI_EXPLAIN() and MAPI_EXPLAIN_QUERY() give information about the context of the failed call, including the expression shipped and any response received. The side-effect is clearing the error status.

13.1.4 Error Message

Almost every call can fail since the connection with the database server can fail at any time. Functions that return a handle (either MAPI or MAPIHDL) may return NULL on failure, or they may return the handle with the error flag set. If the function returns a non-NULL handle, always check for errors with mapi_error.

Functions that return MapiMsg indicate success and failure with the following codes.

MOK	No error
MERROR	Mapi internal error.
MTIMEOUT	Error communicating with the server.

When these functions return MERROR or MTIMEOUT, an explanation of the error can be had by calling one of the functions MAPI_ERROR_STR(), MAPI_EXPLAIN(), or MAPI_EXPLAIN_QUERY().

To check for error messages from the server, call MAPI_RESULT_ERROR(). This function returns NULL if there was no error, or the error message if there was. A user-friendly message can be printed using MAP_EXPLAIN_RESULT(). Typical usage is:

```
do {
    if ((error = mapi_result_error(hdl)) != NULL)
        mapi_explain_result(hdl, stderr);
    while ((line = mapi_fetch_line(hdl)) != NULL)
        /* use output */;
} while (mapi_next_result(hdl) == 1);
```

13.1.5 Mapi Function Reference

13.1.6 Connecting and Disconnecting

- Mapi mapi_connect(const char *host, int port, const char *username, const char *password, const char *lang, const char *dbname)

  Setup a connection with a `mserver` at a *host:port* and login with *username* and *password*. If host == NULL, the local host is accessed. If host starts with a '/' and the system supports it, host is actually the name of a UNIX domain socket, and port is ignored. If port == 0, a default port is used. If username == NULL, the username of the owner of the client application containing the Mapi code is used. If password == NULL, the password is omitted. The preferred query language is any of {sql,mil,mal,xquery }. On success, the function returns a pointer to a structure with administration about the connection.

- Mapi mapi_connect_ssl(const char *host, int port, const char *username, const char *password, const char *lang, const char *dbname)

  Setup a connection with a `mserver` at a *host*:*port* and login with *username* and *password*. The connection is made using the Secure Socket Layer (SSL) and hence all data transfers to and from the server are encrypted. The parameters are the same as in MAPI_CONNECT().
- MapiMsg mapi_disconnect(Mapi mid)

  Terminate the session described by *mid*. The only possible uses of the handle after this call is *mapi_destroy()* and MAPI_RECONNECT(). Other uses lead to failure.
- MapiMsg mapi_destroy(Mapi mid)

  Terminate the session described by *mid* if not already done so, and free all resources. The handle cannot be used anymore.
- MapiMsg mapi_reconnect(Mapi mid)

  Close the current channel (if still open) and re-establish a fresh connection. This will remove all global session variables.
- MapiMsg mapi_ping(Mapi mid)

  Test availability of the server. Returns zero upon success.

13.1.7 Sending Queries

- MapiHdl mapi_query(Mapi mid, const char *Command)

  Send the Command to the database server represented by mid. This function returns a query handle with which the results of the query can be retrieved. The handle should be closed with MAPI_CLOSE_HANDLE(). The command response is buffered for consumption, c.f. mapi_fetch_row().
- MapiMsg mapi_query_handle(MapiHdl hdl, const char *Command)

  Send the Command to the database server represented by hdl, reusing the handle from a previous query. If Command is zero it takes the last query string kept around. The command response is buffered for consumption, e.g. MAPI_FETCH_ROW().
- MapiHdl mapi_query_array(Mapi mid, const char *Command, char **argv)

  Send the Command to the database server replacing the placeholders (?) by the string arguments presented.
- MapiHdl mapi_quick_query(Mapi mid, const char *Command, FILE *fd)

  Similar to MAPI_QUERY(), except that the response of the server is copied immediately to the file indicated.
- MapiHdl mapi_quick_query_array(Mapi mid, const char *Command, char **argv, FILE *fd)

  Similar to MAPI_QUERY_ARRAY(), except that the response of the server is not analyzed, but shipped immediately to the file indicated.
- MapiHdl mapi_stream_query(Mapi mid, const char *Command, int windowsize)

  Send the request for processing and fetch a limited number of tuples (determined by the window size) to assess any erroneous situation. Thereafter, prepare for continual reading of tuples from the stream, until an error occurs. Each time a tuple arrives, the cache is shifted one.
- MapiHdl mapi_prepare(Mapi mid, const char *Command)

  Move the query to a newly allocated query handle (which is returned). Possibly interact with the back-end to prepare the query for execution.
- MapiMsg mapi_execute(MapiHdl hdl)

  Ship a previously prepared command to the back-end for execution. A single answer is pre-fetched to detect any runtime error. MOK is returned upon success.
- MapiMsg mapi_execute_array(MapiHdl hdl, char **argv)

  Similar to mapi_execute but replacing the placeholders for the string values provided.
- MapiMsg mapi_finish(MapiHdl hdl)

  Terminate a query. This routine is used in the rare cases that consumption of the tuple stream produced should be prematurely terminated. It is automatically called when a new query using the same query handle is shipped to the database and when the query handle is closed with MAPI_CLOSE_HANDLE().
- MapiMsg mapi_virtual_result(MapiHdl hdl, int columns, const char **columnnames, const char **columntypes, const int *columnlengths, int tuplecount, const char ***tuples)

  Submit a table of results to the library that can then subsequently be accessed as if it came from the server. columns is the number of columns of the result set and must be greater than zero. columnnames is a list of pointers to strings giving the names of the individual columns. Each pointer may be NULL and columnnames may be NULL if there are no names. tuplecount is the length (number of rows) of the result set. If tuplecount is less than zero, the number of rows is determined by a NULL pointer in the list of tuples pointers. tuples is a list of pointers to row values. Each row value is a list of pointers to strings giving the individual results. If one of these pointers is NULL it indicates a NULL/nil value.

13.1.8 Getting Results

- int mapi_get_field_count(Mapi mid)

  Return the number of fields in the current row.
- int mapi_get_row_count(Mapi mid)

  If possible, return the number of rows in the last select call. A -1 is returned if this information is not available.
- int mapi_rows_affected(MapiHdl hdl)

  Return the number of rows affected by a database update command such as SQL's INSERT/DELETE/UPDATE statements.

- int mapi_fetch_row(MapiHdl hdl)

  Retrieve a row from the server. The text retrieved is kept around in a buffer linked with the query handle from which selective fields can be extracted. It returns the number of fields recognized. A zero is returned upon encountering end of sequence or error. This can be analyzed in using MAPI_ERROR().
- int mapi_fetch_all_rows(MapiHdl hdl)

  All rows are cached at the client side first. Subsequent calls to MAPI_FETCH_ROW() will take the row from the cache. The number or rows cached is returned.
- int mapi_quick_response(MapiHdl hdl, FILE *fd)

  Read the answer to a query and pass the results verbatim to a stream. The result is not analyzed or cached.
- MapiMsg mapi_seek_row(MapiHdl hdl, int rownr, int whence)

  Reset the row pointer to the requested row number. If whence is MAPI_SEEK_SET (0), rownr is the absolute row number (0 being the first row); if whence is MAPI_SEEK_CUR (1), rownr is relative to the current row; if whence is MAPI_SEEK_END (2), rownr is relative to the last row.
- MapiMsg mapi_fetch_reset(MapiHdl hdl)

  Reset the row pointer to the first line in the cache. This need not be a tuple. This is mostly used in combination with fetching all tuples at once.
- char **mapi_fetch_field_array(MapiHdl hdl)

  Return an array of string pointers to the individual fields. A zero is returned upon encountering end of sequence or error. This can be analyzed in using mapi_error().
- char *mapi_fetch_field(MapiHdl hdl, int fnr)

  Return a pointer a C-string representation of the value returned. A zero is returned upon encountering an error or when the database value is NULL; this can be analyzed in using mapi_error().
- MapiMsg mapi_next_result(MapiHdl hdl)

  Go to the next result set, discarding the rest of the output of the current result set.

13.1.9 Errors

- MapiMsg mapi_error(Mapi mid)

  Return the last error code or 0 if there is no error.
- char *mapi_error_str(Mapi mid)

  Return a pointer to the last error message.
- char *mapi_result_error(MapiHdl hdl)

  Return a pointer to the last error message from the server.
- MapiMsg mapi_explain(Mapi mid, FILE *fd)

  Write the error message obtained from `mserver` to a file.
- MapiMsg mapi_explain_query(MapiHdl hdl, FILE *fd)

  Write the error message obtained from `mserver` to a file.
- MapiMsg mapi_explain_result(MapiHdl hdl, FILE *fd)

  Write the error message obtained from `mserver` to a file.

13.1.10 Parameters

- MapiMsg mapi_bind(MapiHdl hdl, int fldnr, char **val)

  Bind a string variable with a field in the return table. Upon a successful subsequent MAPI_FETCH_ROW() the indicated field is stored in the space pointed to by val. Returns an error if the field identified does not exist.
- MapiMsg mapi_bind_var(MapiHdl hdl, int fldnr, int type, void *val)

  Bind a variable to a field in the return table. Upon a successful subsequent MAPI_FETCH_ROW(), the indicated field is converted to the given type and stored in the space pointed to by val. The types recognized are `{` MAPI_TINY, MAPI_UTINY, MAPI_SHORT, MAPI_USHORT, MAPI_INT, MAPI_UINT, MAPI_LONG, MAPI_ULONG, MAPI_LONGLONG, MAPI_ULONGLONG, MAPI_CHAR, MAPI_VARCHAR, MAPI_FLOAT, MAPI_DOUBLE, MAPI_DATE, MAPI_TIME, MAPI_DATETIME `}`. The binding operations should be performed after the mapi_execute command. Subsequently all rows being fetched also involve delivery of the field values in the C-variables using proper conversion. For variable length strings a pointer is set into the cache.
- MapiMsg mapi_bind_numeric(MapiHdl hdl, int fldnr, int scale, int precision, void *val)

  Bind to a numeric variable, internally represented by MAPI_INT Describe the location of a numeric parameter in a query template.
- MapiMsg mapi_clear_bindings(MapiHdl hdl)

  Clear all field bindings.
- MapiMsg mapi_param(MapiHdl hdl, int fldnr, char **val)

  Bind a string variable with the n-th placeholder in the query template. No conversion takes place.
- MapiMsg mapi_param_type(MapiHdl hdl, int fldnr, int ctype, int sqltype, void *val)

  Bind a variable whose type is described by ctype to a parameter whose type is described by sqltype.
- MapiMsg mapi_param_numeric(MapiHdl hdl, int fldnr, int scale, int precision, void *val)

  Bind to a numeric variable, internally represented by MAPI_INT.
- MapiMsg mapi_param_string(MapiHdl hdl, int fldnr, int sqltype, char *val, int *sizeptr)

  Bind a string variable, internally represented by MAPI_VARCHAR, to a parameter. The sizeptr parameter points to the length of the string pointed to by val. If sizeptr == NULL or *sizeptr == -1, the string is NULL-terminated.
- MapiMsg mapi_clear_params(MapiHdl hdl)

  Clear all parameter bindings.

13.1.11 Miscellaneous

- MapiMsg mapi_setAutocommit(Mapi mid, int autocommit)

  Set the autocommit flag (default is on). This only has an effect when the language is SQL. In that case, the server commits after each statement sent to the server.

- MapiMsg mapi_cache_limit(Mapi mid, int maxrows)

  A limited number of tuples are pre-fetched after each EXECUTE(). If maxrows is negative, all rows will be fetched before the application is permitted to continue. Once the cache is filled, a number of tuples are shuffled to make room for new ones, but taking into account non-read elements. Filling the cache quicker than reading leads to an error.
- MapiMsg mapi_cache_shuffle(MapiHdl hdl, int percentage)

  Make room in the cache by shuffling percentage tuples out of the cache. It is sometimes handy to do so, for example, when your application is stream-based and you process each tuple as it arrives and still need a limited look-back. This percentage can be set between 0 to 100. Making shuffle= 100% (default) leads to paging behavior, while shuffle==1 leads to a sliding window over a tuple stream with 1% refreshing.
- MapiMsg mapi_cache_freeup(MapiHdl hdl, int percentage)

  Forcefully shuffle the cache making room for new rows. It ignores the read counter, so rows may be lost.
- char * mapi_quote(const char *str, int size)

  Escape special characters such as \n, \t in str with backslashes. The returned value is a newly allocated string which should be freed by the caller.
- char * mapi_unquote(const char *name)

  The reverse action of MAPI_QUOTE(), turning the database representation into a C-representation. The storage space is dynamically created and should be freed after use.
- MapiMsg mapi_trace(Mapi mid, int flag)

  Set the trace flag to monitor interaction with the server.
- int mapi_get_trace(Mapi mid)

  Return the current value of the trace flag.
- MapiMsg mapi_trace_log(Mapi mid, const char *fname)

  Log the interaction between the client and server for offline inspection. Beware that the log file overwrites any previous log. It is not intended for recovery.

The remaining operations are wrappers around the data structures maintained. Note that column properties are derived from the table output returned from the server.

- char *mapi_get_name(MapiHdl hdl, int fnr)
- char *mapi_get_type(MapiHdl hdl, int fnr)
- char *mapi_get_table(MapiHdl hdl, int fnr)
- int mapi_get_len(Mapi mid, int fnr)
- char *mapi_get_dbname(Mapi mid)
- char *mapi_get_host(Mapi mid)
- char *mapi_get_user(Mapi mid)
- char *mapi_get_lang(Mapi mid)
- char *mapi_get_motd(Mapi mid)

- char **mapi_tables(Mapi mid)

  Return a list of accessible database tables.
- char **mapi_fields(Mapi mid)

  Return a list of accessible tables fields. This can also be obtained by inspecting the field descriptor returned by MAPI_FETCH_FIELD().

13.2 MonetDB Perl Library

Perl is one of the more common scripting languages for which a 'standard' database application programming interface is defined. It is called DBI and it was designed to protect you from the API library details of multiple DBMS vendors. It has a very simple interface to execute SQL queries and for processing the results sent back. DBI doesn't know how to talk to any particular database, but it does know how to locate and load in DBD ('Database Driver') modules. The DBD modules encapsulate the interface library's intricacies and knows how to talk to the real databases.

MonetDB comes with its own DBD module which is included in both the source and binary distribution packages. The module is also available via CPAN.

Two sample Perl applications are included in the source distribution; a MIL session and a simple client to interact with a running server.

For further documentation we refer to the Perl community home page.

13.2.1 A Simple Perl Example

```
use strict;
use warnings;
use DBI();

print "\nStart a simple Monet MIL interaction\n\n";

# determine the data sources:
my @ds = DBI->data_sources('monetdb');
print "data sources: @ds\n";

# connect to the database:
my $dsn = 'dbi:monetdb:database=test;host=localhost;port=50000;language=mil';
my $dbh = DBI->connect( $dsn,
  undef, undef,  # no authentication in MIL
  { PrintError => 0, RaiseError => 1 }  # turn on exception handling
);
{
  # simple MIL statement:
  my $sth = $dbh->prepare('print(2);');
  $sth->execute;
  my @row = $sth->fetchrow_array;
  print "field[0]: $row[0], last index: $#row\n";
}
{
  my $sth = $dbh->prepare('print(3);');
  $sth->execute;
  my @row = $sth->fetchrow_array;
  print "field[0]: $row[0], last index: $#row\n";
}
{
  # deliberately executing a wrong MIL statement:
  my $sth = $dbh->prepare('( xyz 1);');
  eval { $sth->execute }; print "ERROR REPORTED: $@" if $@;
}
$dbh->do('var b:=new(int,str);');
$dbh->do('insert(b,3,"three");');
{
  # variable binding stuff:
  my $sth = $dbh->prepare('insert(b,?,?);');
  $sth->bind_param( 1,     7 , DBI::SQL_INTEGER() );
  $sth->bind_param( 2,'seven' );
  $sth->execute;
}
{
  my $sth = $dbh->prepare('print(b);');
  # get all rows one at a time:
  $sth->execute;
  while ( my $row = $sth->fetch ) {
    print "bun: $row->[0], $row->[1]\n";
  }
  # get all rows at once:
  $sth->execute;
  my $t = $sth->fetchall_arrayref;
  my $r = @$t;         # row count
  my $f = @{$t->[0]};  # field count
  print "rows: $r, fields: $f\n";
  for my $i ( 0 .. $r-1 ) {
    for my $j ( 0 .. $f-1 ) {
      print "field[$i,$j]: $t->[$i][$j]\n";
    }
  }
}
{
  # get values of the first column from each row:
  my $row = $dbh->selectcol_arrayref('print(b);');
  print "head[$_]: $row->[$_]\n" for 0 .. 1;
}
{
  my @row = $dbh->selectrow_array('print(b);');
  print "field[0]: $row[0]\n";
  print "field[1]: $row[1]\n";
```

```
}
{
  my $row = $dbh->selectrow_arrayref('print(b);');
  print "field[0]: $row->[0]\n";
  print "field[1]: $row->[1]\n";
}
$dbh->disconnect;
print "\nFinished\n";
```

13.3 MonetDB PHP Library

The MonetDB distribution comes with a MAPI based PHP interface For general compilation of MonetDB see the howto's for Unix and Linux. The unix configure process normally tries to detect if you have PHP including developer packages installed and builds the PHP module only if you have it. With the –WITH-PHP option you could tell 'configure' where to find the PHP installation.

When the build process is complete you should have a PHP extension_dir under your MonetDB prefix directory. Usually this is PREFIX/LIB(64)/PHP5. It contains the loadable php module, *lib/php5/monetdb.dll*.

Depending on your local setup you could now use these files by coping them into the system extension_dir or with a private webserver you could simply reset the environment variables, include_path and extension_dir. For example you could have a php.ini file which has the following php section.

```
[PHP]
safe_mode = Off
safe_mode_gid = Off
extension_dir = /opt/MonetDB-4.6/lib(64)/php4
```

13.3.1 A Simple PHP Example

A tiny example of the use the MonetDB PHP module follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"" xml:lang="en" lang="en">

<head>
        <title>MonetDB Query</title>
</head>

<body>
<?php
        if ( isset($_POST['query']) )
        {
                $db = monetdb_connect('sql', 'localhost', 50000, 'monetdb',
'monetdb')
                                    or die('Failed to connect to MonetDB<br>');
                $sql = stripslashes($_POST['query']);
                $res = monetdb_query($sql);
                while ( $row = monetdb_fetch_assoc($res) )
                {
                        print "<pre>\n";
                        print_r($row);
                        print "</pre>\n";
                }
        }

        print "<form method=\"post\" action=\"{$_SERVER['PHP_SELF']}\">\n";
        print "<label for=\"query\">SQL Query:</label>\n";
        print "<input type=\"text\" name=\"query\" id=\"query\"
value=\"{$_POST['query']}\" />\n";
        print "<input type=\"submit\" value=\"Execute\" />\n";
        print "</form>\n";
?>
</body>

</html>
```

More examples can be found in the sources.

The PHP module is aligned with the PostgreSQL implementation. A synopsis of the operations provided:

- proto resource monetdb_connect([string host [, string port [, string username [, string password [, string language]]]]]) Open a MonetDB connection
- proto resource monetdb_pconnect([string host [, string port [, string username [, string password [, string language]]]]]) Open a persistent MonetDB connection
- proto bool monetdb_close([resource connection]) Close a MonetDB connection
- proto string monetdb_dbname([resource connection]) Get the database name
- proto string monetdb_last_error([resource connection]) Get the error message string
- proto string monetdb_host([resource connection]) Returns the host name associated with the connection
- proto array monetdb_version([resource connection]) Returns an array with client, protocol and server version (when available)
- proto bool monetdb_ping([resource connection]) Ping database. If connection is bad, try to reconnect.
- proto resource monetdb_query([resource connection,] string query) Execute a query
- proto resource monetdb_query_params([resource connection,] string query, array params) Execute a query
- proto resource monetdb_prepare([resource connection,] string stmtname, string query) Prepare a query for future execution
- proto resource monetdb_execute([resource connection,] string stmtname, array params) Execute a prepared query

- proto int monetdb_num_rows(resource result) Return the number of rows in the result
- proto int monetdb_num_fields(resource result) Return the number of fields in the result
- proto int monetdb_affected_rows(resource result) Returns the number of affected tuples
- proto string pg_last_notice(resource connection) Returns the last notice set by the back-end
- proto string monetdb_field_name(resource result, int field_number) Returns the name of the field
- proto string monetdb_field_table(resource result, int field_number) Returns the name of the table field belongs to
- proto string monetdb_field_type(resource result, int field_number) Returns the type of the field
- proto int monetdb_field_num(resource result, string field_name) Returns the field number of the named field
- proto mixed monetdb_fetch_result(resource result, [int row_number,] mixed field_name) Returns values from a result identifier
- proto array monetdb_fetch_row(resource result [, int row [, int result_type]]) Get a row as an enumerated array
- proto array monetdb_fetch_assoc(resource result [, int row]) Fetch a row as an assoc array
- proto array monetdb_fetch_array(resource result [, int row [, int result_type]]) Fetch a row as an array
- proto object monetdb_fetch_object(resource result [, int row [, string class_name [, NULL|array ctor_params]]]) Fetch a row as an object
- proto bool monetdb_result_seek(resource result, int offset) Set internal row offset
- proto int monetdb_field_prtlen(resource result, [int row,] mixed field_name_or_number) Returns the printed length
- proto int monetdb_field_is_null(resource result, [int row,] mixed field_name_or_number) Test if a field is NULL
- proto bool monetdb_free_result(resource result) Free result memory
- proto string monetdb_escape_string(string data) Escape string for text/char type
- proto int monetdb_connection_status(resource connnection) Get connection status
- proto bool monetdb_connection_reset(resource connection) Reset connection (reconnect)
- proto bool monetdb_put_line([resource connection,] string query) Send null-terminated string to back-end server
- proto bool monetdb_end_copy([resource connection]) Sync with back-end. Completes the Copy command
- proto array monetdb_copy_to(resource connection, string table_name [, string delimiter [, string null_as]]) Copy table to array
- proto bool monetdb_copy_from(resource connection, string table_name , array rows [, string delimiter [, string null_as]]) Copy table from array

- proto bool monetdb_connection_busy(resource connection) Get connection is busy or not
- proto bool monetdb_send_query(resource connection, string query) Send asynchronous query
- proto bool monetdb_send_query_params(resource connection, string query) Send asynchronous parameterized query
- proto bool monetdb_send_prepare(resource connection, string stmtname, string query) Asynchronously prepare a query for future execution
- proto bool monetdb_send_execute(resource connection, string stmtname, array params) Executes prevriously prepared stmtname asynchronously
- proto resource monetdb_get_result(resource connection) Get asynchronous query result
- proto mixed monetdb_result_status(resource result[, long result_type]) Get status of query result
- proto array monetdb_meta_data(resource db, string table) Get meta_data
- proto array monetdb_convert(resource db, string table, array values[, int options]) Check and convert values for MonetDB SQL statement
- proto mixed monetdb_insert(resource db, string table, array values[, int options]) Insert values (filed=>value) to table
- proto mixed monetdb_update(resource db, string table, array fields, array ids[, int options]) Update table using values (field=>value) and ids (id=>value)
- proto mixed monetdb_delete(resource db, string table, array ids[, int options]) Delete records has ids (id=>value)
- proto mixed monetdb_select(resource db, string table, array ids[, int options]) Select records that has ids (id=>value)

13.4 MonetDB Python Library

The MonetDB distribution comes with a Python DB-API driver. This is build on top of the MAPI interface for Python.

13.4.1 Installation

The unix configure process normally tries to detect if you have python including developer packages installed and builds the python module only if you have it. With the *–with-python* option you could tell 'configure' where to find the python installation.

When the build process is complete you should have a python library directory under your MonetDB prefix directory. Usually this is *prefix/lib(64)/python2.4/site-packages*. Now add this directory to your PYTHONPATH.

For windows setups the story is a little bit more complex. TODO

13.4.2 A Simple Python Example

A small example to access the SQL functionality is shown below.

```
import MonetSQLdb

# Let the cursor return dictionaries instead of tuples:
```

```
# x = MonetSQLdb.connect(cursorclass = MonetSQLdb.cursors.DictCursor)

x = MonetSQLdb.connect(host = 'localhost', user = 'monetdb',
                       password = 'monetdb', lang = 'sql')

c = x.cursor()
c.execute('select * from tables')

print c.fetchone()
# print c.fetchall()

x.close()
```

13.4.3 Python MonetDB functions

13.5 MonetDB JDBC Driver

The most obvious way to connect to a data source using the Java programming language is by making use of the in Java defined JDBC framework.

MonetDB has a native Java JDBC driver type 4 which allows use of the MonetDB database in a Java alike way.

It is quite difficult to have a fully complete JDBC implementation. Also this driver isn't complete in that sense. However, it is believed that the most prominent parts of the JDBC interface are implemented, and in such a way that they adhere to the specifications. If you make extensive use of JDBC semantics and rely on many of its features, please read the release notes which are to be found in the SRC/JDBC directory of the sql CVS tree.

This document will give a short description how to use the MonetDB JDBC driver in Java applications. A familiarity with the Java JDBC API is required to fully understand this document. Please note that you can find the complete JDBC API on Sun's web site HTTP://JAVA.SUN.COM/.

13.5.1 Requirements

In order to use the MonetDB JDBC driver in Java applications you need (of course) a running MonetDB instance with the SQL module loaded.\\ The SQL module is a separate module in the CVS tree, and should be installed separately, since it is not included in the main MonetDB module.

You should be able to issue the following command on the MonetDB prompt without errors:

```
include sql;
```

13.5.2 Getting the driver Jar

If you (compiled and) installed MonetDB and the SQL module the regular way, you can find a two jar-archives in the directory SHARE/MONETDB/LIB called MONETDB-X.Y-JDBC.JAR and JDBCCLIENT.JAR in the installation root (chosen during the configure process). For regular applications that only need the JDBC driver, the first jar-file is sufficient. When the JdbcClient utility is required, the second is required. Applications that require the

JDBC driver will, however, work with both jar-files. The X and Y in the names represent the major and minor version numbers of the JDBC driver.

13.5.3 Compiling the driver (using ant, optional)

The directory SRC/JDBC of the sql CVS tree contains all the files to build the driver using Apache's Ant tool. Simply issuing the command ANT should be sufficient to build the driver jar-archive in the subdirectory JARS. See the ant web site for more documentation on the ant build-tool. HTTP://ANT.APACHE.ORG/ The Java sources require at least a Java 2 platform 1.4 compatible compiler.

13.5.4 Testing the driver using the JdbcClient utility

Before you start developing your programs which use the MonetDB JDBC driver it is generally a good idea to check if the driver actually works in your environment.\\ Follow the steps below to assure your setup is complete:

1. start MonetDB
2. load the SQL module
3. run the JdbcClient utility using JAVA -JAR {PATH/TO/JDBCCLIENT.JAR} -UMONETDB (with password monetdb)

The last step should give you something like this:

```
% java -jar jars/jdbcclient.jar -umonetdb
password:

Welcome to the MonetDB interactive JDBC terminal!
Database: MonetDB 5.0.0
Driver: MonetDB Native Driver 1.5 (Steadfast_pre4 20061124)
Type \q to quit, \h for a list of available commands
auto commit mode: on
monetdb->
```

13.5.5 Using the driver in your Java programs

In order to use the MonetDB JDBC driver, the jar-archive has to be in the Java classpath. Make sure this is actually the case.

Loading the driver in your Java program requires two lines of code:

```
// make sure the ClassLoader has the MonetDB JDBC driver loaded
Class.forName("nl.cwi.monetdb.jdbc.MonetDriver");
// request a Connection to a MonetDB server running on 'localhost'
Connection con = DriverManager.getConnection("jdbc:monetdb://localhost/database", "monetdb"
```

The first line makes sure the Java ClassLoader has initialised (and loaded) the Driver class of the MonetDB JDBC package, so it is registered with the DriverManager. The second line requests a Connection object from the DriverManager which is suitable for MonetDB.

The string passed to the "GETCONNECTION()"method is defined as "JDBC:MONETDB://<HOST>[:<PORT>]/<DATABASE>" where elements between "<" and ">" are required and elements between "[" and "]" are optional. Note that even though the database can be specified, it usually is not supported by the database. A warning is issued each time that this value is being ignored.

13.5.6 A sample Java program

```
import java.sql.*;

/**
 * This example assumes there exist tables a and b filled with some data.
 * On these tables some queries are executed and the JDBC driver is tested
 * on it's accuracy and robustness against 'users'.
 *
 * @author Fabian Groffen
 */
public class MJDBCTest {
        public static void main(String[] args) throws Exception {
                // make sure the driver is loaded
                Class.forName("nl.cwi.monetdb.jdbc.MonetDriver");
                Connection con = DriverManager.getConnection("jdbc:monetdb://localhost/data
                Statement st = con.createStatement();
                ResultSet rs;

                rs = st.executeQuery("SELECT a.var1, COUNT(b.id) as total FROM a, b WHERE a
                // get meta data and print columns with their type
                ResultSetMetaData md = rs.getMetaData();
                for (int i = 1; i <= md.getColumnCount(); i++) {
                        System.out.print(md.getColumnName(i) + ":" +
                                md.getColumnTypeName(i) + "\t");
                }
                System.out.println("");
                // print the data: only the first 5 rows, while there probably are
                // a lot more. This shouldn't cause any problems afterwards since the
                // result should get properly discarded on the next query
                for (int i = 0; rs.next() && i < 5; i++) {
                        for (int j = 1; j <= md.getColumnCount(); j++) {
                                System.out.print(rs.getString(j) + "\t");
                        }
                        System.out.println("");
                }

                // tell the driver to only return 5 rows, it can optimize on this
                // value, and will not fetch any more than 5 rows.
                st.setMaxRows(5);
                // we ask the database for 22 rows, while we set the JDBC driver to
                // 5 rows, this shouldn't be a problem at all...
                rs = st.executeQuery("select * from a limit 22");
                // read till the driver says there are no rows left
                for (int i = 0; rs.next(); i++) {
                        System.out.print("[" + rs.getString("var1") + "]");
                        System.out.print("[" + rs.getString("var2") + "]");
                System.out.print("[" + rs.getInt("var3") + "]");
                System.out.println("[" + rs.getString("var4") + "]");
        }

        // this close is not needed, should be done by next execute(Query) call
        // however if there can be some time between this point and the next
        // execute call, it is from a resource perspective better to close it.
        //rs.close();

        // unset the row limit; 0 means as much as the database sends us
        st.setMaxRows(0);
        // we only ask 10 rows
        rs = st.executeQuery("select * from b limit 10;");
        // and simply print them
        while (rs.next()) {
                System.out.print(rs.getInt("rowid") + ", ");
                System.out.print(rs.getString("id") + ", ");
                System.out.print(rs.getInt("var1") + ", ");
                System.out.print(rs.getInt("var2") + ", ");
                System.out.print(rs.getString("var3") + ", ");
                System.out.println(rs.getString("var4"));
        }

        // this close is not needed, as the Statement will close the last
        // ResultSet around when it's closed
        // again, if that can take some time, it's nicer to close immediately
        // the reason why these closes are commented out here, is to test if
        // the driver really cleans up it's mess like it should
        //rs.close();

        // perform a ResultSet-less query (with no trailing ; since that should
        // be possible as well and is JDBC standard)
        // Note that this method should return the number of updated rows. This
        // method however always returns -1, since Monet currently doesn't
        // support returning the affected rows.
        st.executeUpdate("delete from a where var1 = 'zzzz'");

        // closing the connection should take care of closing all generated
        // statements from it...
        // don't forget to do it yourself if the connection is reused or much
        // longer alive, since the Statement object contains a lot of things
        // you probably want to reclaim if you don't need them anymore.
        //st.close();
        con.close();
    }
}
```

13.6 MonetDB ODBC Driver

Short for Open DataBase Connectivity, a standard database access method developed by the SQL Access group in 1992. The goal of ODBC is to make it possible to access any data from any application, regardless of which database management system (DBMS) is handling the data. ODBC manages this by inserting a middle layer, called a database driver, between an application and the DBMS. The purpose of this layer is to translate the application's data queries into commands that the DBMS understands. For this to work, both the application and the DBMS must be ODBC-compliant – that is, the application must be capable of issuing ODBC commands and the DBMS must be capable of responding to them.

The ODBC driver for MonetDB is included in the Windows installer and Linux RPMs. The source can be found in the *SQL* CVS tree.

To help you setup your system to use the ODBC driver with MonetDB, two how-tos are available, one for Windows users and one for Linux/UNIX users.

1. Windows Excel import demo A little demo showing how to import data from a MonetDB server into Microsoft Excel.
2. Unix/Linux ODBC guide An example of how to configure ODBC on Unix/Linux.

Appendix A SQL Features

The table below illustrates the features supported (S) and not supported (N) in the MonetDB/SQL distribution. Beware, some of the features are technically impossible to support without major code changes or excessive performance consequences.

Feature ID	Feature name	S/N
B011-B017	Embedded Language support. Core SQL:1999 says that at least one of Embedded Ada, Embedded C, Embedded Cobol, Embedded Fortran, Embedded MUMPS, Embedded Pascal or Embedded PL/I 1 should be supported.	N
E011	Numeric data types (FULL support)	S
E011-01	INTEGER and SMELLIEST data types (including all spellings)	S
E011-02	REAL, DOUBLE PRECISON, and FLOAT data types	S
E011-03	DECIMAL and NUMERIC data types	S
E011-04	Arithmetic operators	S
E011-05	Numeric comparison	S
E011-06	Implicit casting among the numeric data types	S
E021	Character data types	S(PARTIAL support)
E021-01	CHARACTER data type (including all its spellings)	S
E021-02	CHARACTER VARYING data type (including all its spellings)	S
E021-03	Character literals	
E021-04	CHARACTER_LENGTH function	S
E021-05	OCTET_LENGTH function	S
E021-06	SUBSTRING function	S
E021-07	Character concatenation	S
E021-08	UPPER and LOWER functions	S
E021-09	TRIM function	S
E021-10	Implicit casting among the character data types	S
E021-11	POSITION function	S
E021-12	Character comparison	S
E031	Identifiers (FULL support)	S
E031-01	Delimited identifiers	S
E031-02	Lower case identifiers	S
E031-03	Trailing underscore	S
E051	Basic query specification (FULL support)	S
E051-01	SELECT DISTINCT	S
E051-02	GROUP BY clause	S
E051-04	GROUP BY can contain columns not in select-list	S
E051-05	Select list items can be renamed	S
E051-06	HAVING clause	S
E051-07	Qualified * in select list	S
E051-08	Correlation names in the FROM clause	S
E051-09	Rename columns in the FROM clause	S
E061	Basic predicates and search conditions (FULL support)	S
E061-01	Comparison predicate	S
E061-02	BETWEEN predicate	S
E061-03	IN predicate with list of values	S
E061-04	LIKE predicate	S
E061-05	LIKE predicate: ESCAPE clause	S
E061-06	NULL predicate	S
E061-07	Quantified comparison predicate	S
E061-08	EXISTS predicate	S
E061-09	Subqueries in comparison predicate	S
E061-11	Subqueries in IN predicate	S
E061-12	Subqueries in quantified comparison predicate	S
E061-13	Correlated subqueries	S
E061-14	Search condition	S
E071	Basic query expressions	S (FULL support)
E071-01	UNION DISTINCT table operator	S
E071-02	UNION ALL table operator	S
E071-03	EXCEPT DISTINCT table operator	S
E071-05	Columns combined via table operators need not have exactly the same data type	S
E071-06	Table operators in subqueries	S
E081	Basic Privileges	S
E081-01	SELECT privilege at the table level	S
E081-02	DELETE privilege	S
E081-03	INSERT privilege at the table level	S
E081-04	UPDATE privilege at the table level	S
E081-05	UPDATE privilege at the column level	S
E081-06	REFERENCES privilege at the table level	N (SELECT privilege is used)
E081-07	REFERENCES privilege at the column level	N (SELECT privilege is used)
E081-08	WITH GRANT OPTION	S
E081-09	USAGE privilege	N
E081-10	EXECUTE privilege	N
E091	Set functions (FULL support)	S
E091-01	AVG	S
E091-02	COUNT	S
E091-03	MAX	S
E091-04	MIN	S
E091-05	SUM	S
E091-06	ALL quantifier	S

E091-07	DISTINCT quantifier	S
E101	Basic data manipulation (FULL support)	S
E101-01	INSERT statement	S
E101-03	Searched UPDATE statement	S
E101-04	Searched DELETE statement	S
E111	Single row SELECT statement	S
E121	Basic cursor support	N (Cursors are not supported)
E121-01	DECLARE CURSOR	
E121-02	ORDER BY columns need not be in select list	S
E121-03	Value expressions in ORDER BY clause	N
E121-04	OPEN statement	N
E121-06	Positioned UPDATE statement	N
E121-07	Positioned DELETE statement	N
E121-08	CLOSE statement	N
E121-10	FETCH statement: implicit NEXT	N
E121-17	WITH HOLD cursors	N
E131	Null value support (nulls in lieu of values)	S
E141	Basic integrity constraints	S
E141-01	NOT NULL constraints	S
E141-02	UNIQUE constraints of NOT NULL columns	S
E141-03	PRIMARY KEY constraints	S
E141-04	Basic FOREIGN KEY constraint with the NO ACTION default	S
E141-06	CHECK constraints	N
E141-07	Column defaults	S
E141-08	NOT NULL inferred on PRIMARY KEY	S
E141-10	Names in a foreign key can be specified in any order (columns should be in the proper order)	N
E151	Transaction support	S
E151-01	COMMIT statement	S
E151-02	ROLLBACK statement	S
E152	Basic SET TRANSACTION statement	S
E152-01	SET TRANSACTION statement: ISOLATION LEVEL SERIALIZABLE clause	S
E152-02	SET TRANSACTION statement: READ ONLY and READ WRITE clauses	N
E153	Updatable queries with subqueries	S
E161	SQL comments using leading double minus	S
E171	SQLSTATE support	N
F021	Basic information schema	N
F021-01	COLUMNS view	
F021-02	TABLES view	
F021-03	VIEWS view	
F021-04	TABLE_CONSTRAINTS view	

F021-05	REFERENTIAL_CONSTRAINTS view	
F021-06	CHECK_CONSTRAINTS view	
F031	Basic schema manipulation	S
F031-01	CREATE TABLE statement to create persistent base tables	S
F031-02	CREATE VIEW statement	S
F031-03	GRANT statement	S
F031-04	ALTER TABLE statement: ADD COLUMN clause	S
F031-13	DROP TABLE statement: RESTRICT clause	S
F031-16	DROP VIEW statement: RESTRICT clause	S
F031-19	REVOKE statement: RESTRICT clause	S
F041	Basic joined table	S
F041-01	Inner join (but not necessarily the INNER keyword)	S
F041-02	INNER keyword	S
F041-03	LEFT OUTER JOIN	S
F041-04	RIGHT OUTER JOIN	S
F041-05	Outer joins can be nested	S
F041-07	The inner table in a left or right outer join can also be used in an inner join	S
F041-08	All comparison operators are supported (rather than just =)	S
F051	Basic date and time	S
F051-01	DATE data type (including DATE literal)	S
F051-02	TIME data type (including TIME literal) with fractional seconds precision of 0	S
F051-03	TIMESTAMP data type (including TIMESTAMP literal) with fractional seconds precision of 0 and 6	S
F051-04	Comparison predicate on DATE, TIME, and TIMESTAMP data types	S
F051-05	Explicit CAST between datetime types and character types	S
F051-06	CURRENT_DATE	S
F051-07	LOCALTIME	S
F051-08	LOCALTIMESTAMP	S
F081	UNION and EXCEPT in views	S
F131	Grouped operations	S
F131-01	WHERE, GROUP BY, and HAVING clauses supported in queries with grouped views	S
F131-02	Multiple tables supported in queries with grouped views	S
F131-03	Set functions supported in queries with grouped views	S
F131-04	Subqueries with GROUP BY and HAVING clauses and grouped views	S
F131-05	Single row SELECT with GROUP BY and HAVING clauses and grouped views	S
F181	Multiple module support	S (limited support)
F201	CAST function	S
F221	Explicit defaults	S
F261	CASE expression	S

F261-01	Simple CASE	S
F261-02	Searched CASE	N
F261_03	NULLIF	S
F261-04	COALESCE	S
F311	Schema definition statement	S
F311-01	CREATE SCHEMA	S
F311-02	CREATE TABLE for persistent base tables	S
F311-03	CREATE VIEW	S
F311-04	CREATE VIEW: WITH CHECK OPTION	N
F311-05	GRANT statement	S
F471	Scalar subquery values	S
F481	Expanded NULL predicate	S
F501	Features and conformance views	N
F501-01	SQL_FEATURES view	
F501-02	SQL_SIZING view	
F501-03	SQL_LANGUAGES view	
F812	Basic flagging	N
S011	Distinct data types	S
S011-01	USER_DEFINED_TYPES view	N
T321	Basic SQL-invoked routines	S
T321-01	User-defined functions with no overloading	S
T321-02	User-defined stored procedures with no overloading	S
T321-03	Function invocation	S
T321-04	CALL statement	N
T321-05	RETURN statement	S
T321-06	ROUTINES view	N
T321-07	PARAMETERS view	N

Appendix B MAL Instruction Summary

The table below gives a condensed overview of the operations defined in each of the modules.

aggr.avg	aggr.count_no_nil	aggr.min	aggr.size
aggr.cardinality	aggr.histogram	aggr.prod	aggr.sum
aggr.count	aggr.max	aggr.product	
alarm.alarm	alarm.epoch	alarm.time	
alarm.ctime	alarm.prelude	alarm.timers	
alarm.epilogue	alarm.sleep	alarm.usec	
algebra.antijoin	algebra.joinPath	algebra.position	algebra.sortReverseTail
algebra.bandjoin	algebra.kdifference	algebra.project	algebra.sortTH
algebra.copy	algebra.kintersect	algebra.rangesplit	algebra.sortTail
algebra.crossproduct	algebra.kunion	algebra.reuse	algebra.split
algebra.difference	algebra.kunique	algebra.revert	algebra.ssort
algebra.exist	algebra.leftfetchjoin	algebra.sample	algebra.ssort_rev
algebra.fetch	algebra.leftjoin	algebra.sdifference	algebra.sunion
algebra.fetchjoin	algebra.like	algebra.select	algebra.sunique
algebra.find	algebra.markH	algebra.selectH	algebra.thetajoin
algebra.fragment	algebra.markT	algebra.selectNotNil	algebra.topN
algebra.groupby	algebra.mark_grp	algebra.semijoin	algebra.tunique
algebra.hashjoin	algebra.materialize	algebra.sintersect	algebra.uhashsplit
algebra.hashsplit	algebra.merge	algebra.slice	algebra.union
algebra.indexjoin	algebra.mergejoin	algebra.sort	algebra.unique
algebra.intersect	algebra.number	algebra.sortHT	algebra.urangesplit
algebra.join	algebra.outerjoin	algebra.sortReverse	algebra.uselect
array.grid	array.print	array.product	array.project
bat.append	bat.hasAppendMode	bat.new	bat.setHot
bat.delete	bat.hasMoreElements	bat.newIterator	bat.setKey
bat.flush	bat.hasReadMode	bat.order	bat.setMemoryAdvise
bat.getAccess	bat.hasWriteMode	bat.orderReverse	bat.setMemoryMap
bat.getAlpha	bat.info	bat.pack	bat.setName
bat.getCapacity	bat.inplace	bat.reduce	bat.setPersistent
bat.getDelta	bat.insert	bat.replace	bat.setReadMode
bat.getHead	bat.isCached	bat.reverse	bat.setRole
bat.getHeadType	bat.isPersistent	bat.revert	bat.setSet
bat.getHeat	bat.isSorted	bat.save	bat.setSorted
bat.getName	bat.isSortedReverse	bat.setAccess	bat.setTransient
bat.getRole	bat.isSynced	bat.setAppendMode	bat.setWriteMode
bat.getSequenceBase	bat.isTransient	bat.setBase	bat.unload
bat.getSpaceReserved	bat.isaKey	bat.setCold	bat.unpack
bat.getSpaceUsed	bat.isSet	bat.setColumn	
bat.getTail	bat.load	bat.setGarbage	
bat.getTailType	bat.mirror	bat.setHash	
batcalc.!=	batcalc.and	batcalc.like	batcalc.str
batcalc.%	batcalc.bit	batcalc.lng	batcalc.string
batcalc.*	batcalc.bte	batcalc.ltrim	batcalc.substitute
batcalc.+	batcalc.chr	batcalc.match	batcalc.substring

batcalc.++	batcalc.chrAt	batcalc.nbytes	batcalc.toLower
batcalc.-	batcalc.dbl	batcalc.not	batcalc.toUpper
batcalc.–	batcalc.endsWith	batcalc.oid	batcalc.trim
batcalc./	batcalc.flt	batcalc.or	batcalc.unicodeAt
batcalc.<	batcalc.ifthen	batcalc.r_search	batcalc.xml
batcalc.<=	batcalc.ifthenelse	batcalc.rtrim	batcalc.xor
batcalc.==	batcalc.int	batcalc.search	
batcalc.>	batcalc.isnil	batcalc.sht	
batcalc.>=	batcalc.length	batcalc.startsWith	
batcolor.blue	batcolor.green	batcolor.red	batcolor.value
batcolor.cb	batcolor.hsv	batcolor.rgb	
batcolor.color	batcolor.hue	batcolor.saturation	
batcolor.cr	batcolor.luminance	batcolor.str	
batmath.acos	batmath.cos	batmath.fmod	batmath.sinh
batmath.asin	batmath.cosh	batmath.log	batmath.sqrt
batmath.atan	batmath.exp	batmath.log10	batmath.tan
batmath.atan2	batmath.fabs	batmath.pow	batmath.tanh
batmath.ceil	batmath.floor	batmath.sin	
batmtime.day	batmtime.milliseconds	batmtime.seconds	
batmtime.hours	batmtime.month	batmtime.year	
bbp.bind	bbp.getDirty	bbp.getNames	bbp.prelude
bbp.close	bbp.getDiskSpace	bbp.getObjects	bbp.release
bbp.commit	bbp.getHeadType	bbp.getRNames	bbp.releaseAll
bbp.deposit	bbp.getHeat	bbp.getRefCount	bbp.take
bbp.destroy	bbp.getKind	bbp.getStatus	bbp.toString
bbp.discard	bbp.getLRefCount	bbp.getTailType	
bbp.find	bbp.getLocation	bbp.iterator	
bbp.getCount	bbp.getName	bbp.open	
blob.blob	blob.prelude	blob.tostring	
blob.nitems	blob.toblob		
box.close	box.discard	box.open	box.take
box.deposit	box.getBoxNames	box.release	box.toString
box.destroy	box.iterator	box.releaseAll	
bpm.adapt	bpm.fold	bpm.mapNxt	bpm.rangePartition
bpm.addPartition	bpm.garbage	bpm.mapPrv	bpm.rangePartitionSort
bpm.close	bpm.getDimension	bpm.mapThghDbl	bpm.replace
bpm.count	bpm.getNames	bpm.mapThghLng	bpm.saveCatalog
bpm.delete	bpm.getNumberOfPartitions	bpm.mapTlowDbl	bpm.select
bpm.deposit	bpm.hasMoreElements	bpm.mapTlowLng	bpm.sortPartitions
bpm.derivePartition	bpm.hashPartition	bpm.new	bpm.sortTail
bpm.destroy	bpm.hashPartitions	bpm.newIterator	bpm.splitquant
bpm.discard	bpm.insert	bpm.open	bpm.take
bpm.dump	bpm.mapAlias	bpm.partition	bpm.unfold
bpm.emptySet	bpm.mapBid	bpm.pieces	
bpm.epilogue	bpm.mapName	bpm.prelude	
bstream.create	bstream.destroy	bstream.read	
calc.!=	calc.abs	calc.getBATidentifier	calc.or

calc.%	calc.and	calc.ifthenelse	calc.ptr
calc.*	calc.bat	calc.int	calc.setoid
calc.+	calc.between	calc.inv	calc.sht
calc.-	calc.bit	calc.isnil	calc.sqlblob
calc./	calc.blob	calc.isnotnil	calc.str
calc.<	calc.bte	calc.length	calc.timestamp
calc.<<	calc.chr	calc.lng	calc.void
calc.<=	calc.date	calc.max	calc.xml
calc.==	calc.daytime	calc.min	calc.xor
calc.>	calc.dbl	calc.newoid	
calc.>=	calc.flt	calc.not	
calc.>>	calc.getBAT	calc.oid	
clients.addScenario	clients.getActions	clients.getTime	clients.setListing
clients.addUser	clients.getId	clients.getUsers	clients.setPassword
clients.changePassword	clients.getInfo	clients.quit	clients.setScenario
clients.changeUsername	clients.getLastCommand	clients.removeScenario	clients.shutdown
clients.checkPermission	clients.getLogins	clients.removeUser	clients.suspend
clients.exit	clients.getScenario	clients.setHistory	clients.wakeup
color.blue	color.green	color.print	color.saturation
color.cb	color.hsv	color.printf	color.str
color.color	color.hue	color.red	color.value
color.cr	color.luminance	color.rgb	color.ycc
const.close	const.epiloque	const.prelude	const.toString
const.deposit	const.hasMoreElements	const.release	
const.destroy	const.newIterator	const.releaseAll	
const.discard	const.open	const.take	
constraints.emptySet			
crackers.DeleteMap	crackers.dselect	crackers.joinselect	crackers.sizeCrackerDeletions
crackers.InsertAVLIndex	crackers.extendCrackerBAT	crackers.joinuselect	crackers.sizeCrackerInsertions
crackers.buildAVLIndex	crackers.extendCrackerMap	crackers.printAVLTree_int	crackers.sizePendingInsertions
crackers.crackOrdered	crackers.getCrackerBAT	crackers.printCrackerBAT	crackers.tselect
crackers.crackOrdered_validate	crackers.hselect	crackers.printCrackerDeletions	crackers.uselect
crackers.crackUnordered_validate	crackers.insert	crackers.printCrackerIndexBATpart	crackers.verifyCrackerIndex
crackers.deleteAVL	crackers.insertionsBForce	crackers.printCrackerInsertions	crackers.zcrackOrdered
crackers.deletionsOnNeed	crackers.insertionsBOnNeed	crackers.printPendingInsertions	crackers.zcrackOrdered_validate
crackers.deletionsOnNeedGradually	crackers.insertionsBOnNeedGradually	crackers.project	crackers.zcrackUnordered
crackers.deletionsOnNeedGraduallyRipple	crackers.insertionsBOnNeedGraduallyRipple	crackers.projectR	crackers.zcrackUnordered_validate
crackers.djoinselect	crackers.insertionsForget	crackers.select	
crackers.dproject	crackers.insertionsPartiallyForget	crackers.selectAVL	
date.!=	date.<=	date.>	date.date
date.<	date.==	date.>=	date.isnil
daytime.!=	daytime.<=	daytime.>	daytime.isnil
daytime.<	daytime.==	daytime.>=	
factory.getArrival	factory.getDeparture	factory.getPlants	
factory.getCaller	factory.getOwners	factory.shutdown	
group.avg	group.max	group.prelude	group.size
group.count	group.min	group.refine	group.sum

group.derive	group.new	group.refine_reverse	
inet.!=	inet.>	inet.host	inet.new
inet.<	inet.>=	inet.hostmask	inet.setmasklen
inet.<<	inet.>>	inet.isnil	inet.text
inet.<<=	inet.>>=	inet.masklen	
inet.<=	inet.abbrev	inet.netmask	
inet.=	inet.broadcast	inet.network	
inspect.getAddress	inspect.getComment	inspect.getModule	inspect.getStatistics
inspect.getAddresses	inspect.getDefinition	inspect.getSignature	inspect.getType
inspect.getAtomNames	inspect.getEnvironment	inspect.getSignatures	inspect.getTypeIndex
inspect.getAtomSizes	inspect.getFunction	inspect.getSize	inspect.getTypeName
inspect.getAtomSuper	inspect.getKind	inspect.getSource	
io.export	io.print	io.prompt	io.stdout
io.ftable	io.printf	io.stderr	io.table
io.import	io.printft	io.stdin	
language.assert	language.dataflow	language.raise	
language.assertSpace	language.newRange	language.register	
language.call	language.nextElement	language.source	
lock.create	lock.set	lock.try	
lock.destroy	lock.tostr	lock.unset	
mal.multiplex			
manual.completion	manual.help	manual.search	manual.summary
manual.createXML	manual.index	manual.section	
mat.hasMoreElements	mat.new	mat.pack	
mat.info	mat.newIterator	mat.print	
mdb.List	mdb.getException	mdb.list	mdb.setMemory
mdb.collect	mdb.getReason	mdb.listMapi	mdb.setTimer
mdb.dot	mdb.getStackDepth	mdb.setBigfoot	mdb.setTrace
mdb.dump	mdb.getStackFrame	mdb.setCatch	mdb.start
mdb.getContext	mdb.getStackTrace	mdb.setDebug	mdb.stop
mdb.getDebug	mdb.grab	mdb.setFlow	mdb.var
mdb.getDefinition	mdb.inspect	mdb.setIO	
mkey.bulk_rotate_xor_hash	mkey.hash	mkey.rotate	
mmath.acos	mmath.exp	mmath.log	mmath.sinh
mmath.asin	mmath.fabs	mmath.log10	mmath.sqrt
mmath.atan	mmath.finite	mmath.pi	mmath.srand
mmath.atan2	mmath.floor	mmath.pow	mmath.tan
mmath.ceil	mmath.fmod	mmath.rand	mmath.tanh
mmath.cos	mmath.isinf	mmath.round	
mmath.cosh	mmath.isnan	mmath.sin	
mserver.bind	mserver.fetch_field_array	mserver.listen_ssl	mserver.query_handle
mserver.connect	mserver.fetch_line	mserver.lookup	mserver.reconnect
mserver.connect_ssl	mserver.fetch_reset	mserver.malclient	mserver.resume
mserver.destroy	mserver.fetch_row	mserver.next_result	mserver.rpc
mserver.disconnect	mserver.finish	mserver.ping	mserver.setAlias
mserver.error	mserver.getError	mserver.prepare	mserver.stop
mserver.explain	mserver.get_field_count	mserver.put	mserver.suspend

mserver.fetch_all_rows	mserver.get_row_count	mserver.query	mserver.trace
mserver.fetch_field	mserver.listen	mserver.query_array	
mtime.add	mtime.dayname	mtime.month	mtime.time_sub_sec_interval
mtime.adddays	mtime.daynum	mtime.monthname	mtime.time_synonyms
mtime.addmonths	mtime.dayofweek	mtime.monthnum	mtime.timestamp
mtime.addyears	mtime.dayofyear	mtime.msec	mtime.timestamp_add_month_interval
mtime.compute	mtime.daytime	mtime.msecs	mtime.timestamp_add_sec_interval
mtime.current_date	mtime.diff	mtime.olddate	mtime.timestamp_sub_month_interval
mtime.current_time	mtime.dst	mtime.oldduration	mtime.timestamp_sub_sec_interval
mtime.current_timestamp	mtime.end_dst	mtime.prelude	mtime.timezone
mtime.date	mtime.epilogue	mtime.rule	mtime.timezone_local
mtime.date_add_month_interval	mtime.hours	mtime.seconds	mtime.weekday
mtime.date_add_sec_interval	mtime.local_timezone	mtime.setTimezone	mtime.weekofyear
mtime.date_sub_sec_interval	mtime.milliseconds	mtime.start_dst	mtime.year
mtime.day	mtime.minutes	mtime.time_add_sec_interval	
optimizer.accessmode	optimizer.dumpQEP	optimizer.macro	optimizer.reduce
optimizer.accumulators	optimizer.emptySet	optimizer.mergetable	optimizer.remap
optimizer.aliases	optimizer.evaluate	optimizer.multiplex	optimizer.remoteQueries
optimizer.clrDebug	optimizer.factorize	optimizer.optimize	optimizer.setDebug
optimizer.coercions	optimizer.garbageCollector	optimizer.orcam	optimizer.showFlowGraph
optimizer.commonTerms	optimizer.heuristics	optimizer.partitions	optimizer.showPlan
optimizer.costModel	optimizer.inline	optimizer.peephole	optimizer.singleton
optimizer.crack	optimizer.joinPath	optimizer.prelude	optimizer.strengthReduction
optimizer.deadcode	optimizer.joinselect	optimizer.pushranges	optimizer.trace
pcre.compile	pcre.patindex	pcre.replace	
pcre.like	pcre.pcre_quote	pcre.select	
pcre.match	pcre.prelude	pcre.uselect	
pqueue.dequeue_max	pqueue.enqueue_min	pqueue.topn_min	
pqueue.dequeue_min	pqueue.init	pqueue.topreplace_max	
pqueue.enqueue_max	pqueue.topn_max	pqueue.topreplace_min	
profiler.activate	profiler.dumpTrace	profiler.reset	profiler.setStartPoint
profiler.cleanup	profiler.getEvent	profiler.setAll	profiler.start
profiler.closeStream	profiler.getTrace	profiler.setEndPoint	profiler.stop
profiler.clrFilter	profiler.noop	profiler.setFilter	
profiler.deactivate	profiler.openStream	profiler.setNone	
scheduler.choice	scheduler.isolation	scheduler.volumeCost	
scheduler.costPrediction	scheduler.pick		
sema.create	sema.destroy	sema.down	sema.up
sqlblob.sqlblob			
statistics.close	statistics.forceUpdate	statistics.getObjects	statistics.release
statistics.deposit	statistics.getCount	statistics.getSize	statistics.releaseAll
statistics.destroy	statistics.getHistogram	statistics.hasMoreElements	statistics.take
statistics.discard	statistics.getHotset	statistics.newIterator	statistics.toString
statistics.dump	statistics.getMax	statistics.open	statistics.update
statistics.epilogue	statistics.getMin	statistics.prelude	
status.batStatistics	status.getThreads	status.mem_cursize	status.vm_maxsize
status.cpuStatistics	status.ioStatistics	status.mem_maxsize	

status.getDatabases	status.memStatistics	status.vmStatistics	
status.getPorts	status.memUsage	status.vm_cursize	
str.+	str.length	str.rtrim	str.substitute
str.STRepilogue	str.like	str.search	str.substring
str.STRprelude	str.locate	str.space	str.suffix
str.ascii	str.ltrim	str.startsWith	str.toLower
str.chrAt	str.nbytes	str.str	str.toUpper
str.codeset	str.prefix	str.string	str.trim
str.endsWith	str.r_search	str.stringleft	str.unicode
str.iconv	str.repeat	str.stringlength	str.unicodeAt
str.insert	str.replace	str.stringright	
streams.blocked	streams.openReadBytes	streams.readStr	streams.socketWriteBytes
streams.close	streams.openWrite	streams.socketRead	streams.writeInt
streams.flush	streams.openWriteBytes	streams.socketReadBytes	streams.writeStr
streams.openRead	streams.readInt	streams.socketWrite	
tablet.display	tablet.input	tablet.setBracket	tablet.setPivot
tablet.dump	tablet.lastPage	tablet.setColumn	tablet.setPosition
tablet.finish	tablet.load	tablet.setDecimal	tablet.setProperties
tablet.firstPage	tablet.nextPage	tablet.setDelimiter	tablet.setRowBracket
tablet.getPage	tablet.output	tablet.setFormat	tablet.setStream
tablet.getPageCnt	tablet.page	tablet.setName	tablet.setTableBracket
tablet.header	tablet.prevPage	tablet.setNull	tablet.setWidth
timestamp.!=	timestamp.==	timestamp.epoch	
timestamp.<	timestamp.>	timestamp.isnil	
timestamp.<=	timestamp.>=	timestamp.unix_epoch	
timezone.str	timezone.timestamp		
transaction.abort	transaction.clean	transaction.delta	transaction.subcommit
transaction.alpha	transaction.commit	transaction.prev	transaction.sync
unix.getenv	unix.setenv		
url.getAnchor	url.getDomain	url.getProtocol	url.isaURL
url.getBasename	url.getExtension	url.getQuery	url.new
url.getContent	url.getFile	url.getQueryArg	url.url
url.getContext	url.getHost	url.getRobotURL	
url.getDirectory	url.getPort	url.getUser	
user.main			
xml.agg	xml.document	xml.parse	xml.str
xml.attribute	xml.element	xml.pi	xml.tag
xml.comment	xml.forest	xml.prelude	xml.text
xml.concat	xml.isdocument	xml.root	xml.xml
xml.content	xml.options	xml.serialize	xml.xquery
zrule.define			

Appendix C MAL Instruction Help

The table below summarizes the commentary lines encountered in the system associated with a MAL kernel modules.

aggr.avg	Grouped tail average on dbl
aggr.cardinality	Return the cardinality of the BAT tail values.
aggr.count	Grouped count
aggr.count_no_nil	Return the number of elements currently in a BAT ignoring BUNs with nil-tail
aggr.histogram	Produce a BAT containing the histogram over the tail values.
aggr.max	Give the highest tail value.
aggr.min	Give the lowest tail value.
aggr.prod	Gives the product of all tail values.
aggr.product	Product over grouped tail on dbl
aggr.size	Grouped count of true values
aggr.sum	Grouped tail sum on dbl
alarm.alarm	execute action in X secs
alarm.ctime	current time as a string
alarm.epilogue	Finalize alarm module
alarm.epoch	current time as unix epoch
alarm.prelude	Initialize alarm module
alarm.sleep	sleep X secs
alarm.time	time in millisecs
alarm.timers	give a list of all active timers
alarm.usec	return cpu microseconds info
algebra.antijoin	Returns the antijoin
algebra.bandjoin	This is a join() for which the predicate is that two BUNs match if the left-tail value is within the range [right-head - minus, right-head + plus]. Works only for the builtin numerical types, and their derivates.
algebra.copy	Returns physical copy of a BAT.
algebra.crossproduct	Returns the cross product
algebra.difference	
algebra.exist	Returns true when 'h,t' occurs as a bun in b.
algebra.fetch	Returns a positional selection of b by the oid head values of s
algebra.fetchjoin	Hook directly into the fetch implementation of the join.
algebra.find	Returns the tail value 't' for which some [h,t] BUN exists in b. If no such BUN exists, an error occurs.
algebra.fragment	Select both on head and tail range.
algebra.groupby	Produces a new BAT with groups identified by the head column. The result contains tail times the head value, ie the tail contains the result group sizes.
algebra.hashjoin	Hook directly into the hash implementation of the join.

algebra.hashsplit	Split a BAT on tail column according (hash-value MOD buckets). Returns a recursive BAT, containing the fragments in the tail, their bucket number in the head.
algebra.indexjoin	Hook directly into the index implementation of the join.
algebra.intersect	
algebra.join	Returns all BUNs, consisting of a head-value from 'left' and a tail-value from 'right' for which there are BUNs in 'left' and 'right' with equal tail- resp. head-value (i.e. the join columns are projected out).
algebra.joinPath	internal routine to handle join paths. The type analysis is rather tricky.
algebra.kdifference	Returns the difference taken over only the *head* columns of two BATs. Results in all BUNs of 'left' that are *not* in 'right'. It does *not* do double-elimination over the 'left' BUNs. If you want this, use: 'kdifference(left.kunique,right.kunique)' or: 'kdifference(left,right).kunique'.
algebra.kintersect	Returns the intersection taken over only the *head* columns of two BATs. Results in all BUNs of 'left' that are also in 'right'. Does *not* do double-elimination over the 'left' BUNs. If you want this, use: 'kintersect(kunique(left),kunique(right))' or: 'kunique(kintersect(left,right))'.
algebra.kunion	Returns the union of two BATs; looking at head-columns only. Results in all BUNs of 'left' that are not in 'right', plus all BUNs of 'right'. *no* double-elimination is done. If you want this, do: 'kunion(left.kunique,right.kunique)' or: 'sunion(left,right).kunique'.
algebra.kunique	Select unique tuples from the input BAT. Double elimination is done only looking at the head column. The result is a BAT with property hkeyed() == true.
algebra.leftfetchjoin	Hook directly into the left fetch join implementation.
algebra.leftjoin	
algebra.like	Selects all elements that have 'substr' as in the tail.
algebra.markH	Produces a new BAT with fresh unique dense sequense of OIDs in the head that starts at base (i.e. [base,..base+b.count()-1]).
algebra.markT	Produces a BAT with fresh unique OIDs in the tail starting at 0.
algebra.mark_grp	"grouped mark": Produces a new BAT with per group a locally unique dense ascending sequense of OIDs in the tail. The tail of the first BAT (b) identifies the group that each BUN of b belongs to. The second BAT (g) represents the group extend, i.e., the head is the unique list of group IDs from b's tail. The third argument (s) gives the base value for the new OID sequence of each group.

algebra.materialize	Materialize the void column
algebra.merge	Merge head and tail into a single value
algebra.mergejoin	Hook directly into the merge implementation of the join.
algebra.number	Produces a new BAT with identical head column, and consecutively increasing integers (start at 0) in the tail column.
algebra.outerjoin	Returns all the result of a join, plus the BUNS formed NIL in the tail and the head-values of 'outer' whose tail-value does not match an head-value in 'inner'.
algebra.position	Returns the position of the value pair It returns an error if 'val' does not exist.
algebra.project	Fill the tail column with a constant.
algebra.rangesplit	Split a BAT on tail column in 'ranges' equally sized consecutive ranges. Returns a recursive BAT, containing the fragments in the tail, the higher-bound of the range in the head. The higher bound of the last range is 'nil'.
algebra.reuse	Reuse a temporary BAT if you can. Otherwise, allocate enough storage to accept result of an operation (not involving the heap)
algebra.revert	Returns a BAT copy with buns in reverse order
algebra.sample	Produce a random selection of size 'num' from the input BAT.
algebra.sdifference	Returns the difference taken over *both* columns of two BATs. Results in all BUNs of 'left' that are *not* in 'right'. Does *not* do double-elimination over the 'left' BUNs. If you want this, use: 'sdifference(left.sunique,right.sunique)' or: 'sdifference(left,right).sunique'.
algebra.select	Select all BUNs of a BAT with a certain tail value. Selection on NIL is also possible (it should be properly casted, e.g.:int(nil)).
algebra.selectH	
algebra.selectNotNil	Select all not-nil values
algebra.semijoin	Returns the intersection taken over only the *head* columns of two BATs. Results in all BUNs of 'left' that are also in 'right'. Does *not* do double-elimination over the 'left' BUNs. If you want this, use: 'kintersect(kunique(left),kunique(right))' or: 'kunique(kintersect(left,right))'.
algebra.sintersect	Returns the intersection taken over *both* columns of two BATs. Results in all BUNs of 'left' that are also in 'right'. Does *not* do double-elimination over the 'left' BUNs, If you want this, use: 'sintersect(sunique(left),sunique(right))' or: 'sunique(sintersect(left,right))'.
algebra.slice	Return the slice with the BUNs at position x till y.
algebra.sort	Returns a BAT copy sorted on the head column.
algebra.sortHT	Returns a lexicographically sorted copy on head,tail.

algebra.sortReverse	Returns a BAT copy reversely sorted on the tail column.
algebra.sortReverseTail	Returns a BAT copy reversely sorted on the tail column.
algebra.sortTH	Returns a lexicographically sorted copy on tail,head.
algebra.sortTail	Returns a BAT copy sorted on the tail column.
algebra.split	Split head into two values
algebra.ssort	Returns copy of a BAT with the BUNs sorted on ascending head values. This is a stable sort.
algebra.ssort_rev	Returns copy of a BAT with the BUNs sorted on descending head values. This is a stable sort.
algebra.sunion	Returns the union of two BATs; looking at both columns of both BATs. Results in all BUNs of 'left' that are not in 'right', plus all BUNs of 'right'. *no* double-elimination is done. If you want this, do: 'sunion(left.sunique,right.sunique)' or: 'sunion(left,right).sunique'.
algebra.sunique	Select unique tuples from the input BAT. Double elimination is done over BUNs as a whole (head and tail). Result is a BAT with real set() semantics.
algebra.thetajoin	Theta join on for 'mode' in LE, LT, EQ, GT, GE . JOIN_EQ is just the same as join(). All other options do merge algorithms. Either using the fact that they are ordered() already (left on tail, right on head), or by using/creating binary search trees on the join columns.
algebra.topN	Trim all but the top N tuples.
algebra.tunique	Select unique tuples from the input BAT. Double elimination is done over the BUNs tail. The result is a BAT with property tkeyd()== true
algebra.uhashsplit	Same as hashsplit, but only collect the head values in the fragments
algebra.union	
algebra.unique	
algebra.urangesplit	Same as rangesplit, but only collect the head values in the fragments
algebra.uselect	Value select, but returning only the head values. SEE ALSO:select(bat,val)
array.grid	Fills an index BAT, (grpcount,grpsize,clustersize,offset) and shift all elemenets with a factor s
array.print	Prints an array, using 1 value bat and N aligned index bats
array.product	Produce an array product
array.project	Fill an array representation with constants
bat.append	append the value u to i
bat.delete	Delete from the first BAT all BUNs with a corresponding BUN in the second.
bat.flush	Designate a BAT as not needed anymore
bat.getAccess	return the access mode attached to this BAT as a character.
bat.getAlpha	Obtain the list of BUNs added

bat.getCapacity	Returns the current allocation size (in max number of elements) of a BAT.
bat.getDelta	Obtain the list of BUNs deleted
bat.getHead	return the BUN head value using the cursor.
bat.getHeadType	Returns the type of the head column of a BAT, as an integer type number.
bat.getHeat	Return the current BBP heat (LRU stamp)
bat.getName	Gives back the logical name of a BAT.
bat.getRole	Returns the rolename of the head column of a BAT.
bat.getSequenceBase	Get the sequence base for the void column of a BAT.
bat.getSpaceReserved	Determine the total space (in bytes) reserved for a BAT.
bat.getSpaceUsed	Determine the total space (in bytes) occupied by a BAT.
bat.getTail	return the BUN tail value using the cursor.
bat.getTailType	Returns the type of the tail column of a BAT, as an integer type number.
bat.hasAppendMode	return true if to this BAT is append only.
bat.hasMoreElements	Produce the next bun for processing.
bat.hasReadMode	return true if to this BAT is read only.
bat.hasWriteMode	return true if to this BAT is read and write.
bat.info	Produce a BAT containing info about a BAT in [attribute,value] format. It contains all properties of the BAT record. See the BAT documentation in GDK for more information.
bat.inplace	inplace replace values on the given locations
bat.insert	Insert one BUN[h,t] in a BAT.
bat.isCached	Bat is stored in main memory.
bat.isPersistent	
bat.isSorted	Returns whether a BAT is ordered on head or not.
bat.isSortedReverse	Returns whether a BAT is ordered on head or not.
bat.isSynced	Tests whether two BATs are synced or not.
bat.isTransient	
bat.isaKey	return whether the head column of a BAT is unique (key).
bat.isaSet	return whether the BAT mode is set to unique.
bat.load	Load a particular BAT from disk
bat.mirror	Returns the head-mirror image of a BAT (two head columns).
bat.new	Localize a bat by name and produce a variant
bat.newIterator	Process the buns one by one extracted from a void table.
bat.order	Sorts the BAT itself on the head, in place.
bat.orderReverse	Reverse sorts the BAT itself on the head, in place.
bat.pack	Pack a pair into a BAT
bat.reduce	Designate a BAT for which auxillary structures can be dropped
bat.replace	Replace the tail value of one BUN that has some head value.

bat.reverse	Returns the reverse view of a BAT (head is tail and tail is head). BEWARE no copying is involved; input and output refer to the same object!
bat.revert	Puts all BUNs in a BAT in reverse order. (Belongs to the BAT sequence module)
bat.save	Save a BAT to storage, if it was loaded and dirty. Returns whether IO was necessary. Please realize that calling this function violates the atomic commit protocol!!
bat.setAccess	Try to change the update access priviliges to this BAT. Mode: r[ead-only] - allow only read access. a[append-only] - allow reads and update. w[riteable] - allow all operations. BATs are updatable by default. On making a BAT read-only, all subsequent updates fail with an error message.Returns the BAT itself.
bat.setAppendMode	Change access privilige of BAT to append only
bat.setBase	Give the non-empty BATs consecutive oid bases
bat.setCold	Makes a BAT very cold for the BBP. The chance of being choses for swapout is big, afterwards.
bat.setColumn	Give both columns of a BAT a new name.
bat.setGarbage	Designate a BAT as garbage
bat.setHash	
bat.setHot	Makes a BAT very hot for the BBP. The chance of being chosen for swapout is small, afterwards.
bat.setKey	Sets the 'key' property of the head column to 'mode'. In 'key' mode, the kernel will silently block insertions that cause a duplicate entries in the head column. KNOWN BUG:when 'key' is set to TRUE, this function does not automatically eliminate duplicates. Use b := b.kunique;
bat.setMemoryAdvise	alias for madvise(b,mode,mode,mode, mode,mode)
bat.setMemoryMap	Alias for mmap(b,mode,mode,mode, mode,mode)
bat.setName	Give a logical name to a BAT.
bat.setPersistent	Make the BAT persistent. Returns boolean which indicates if the BAT administration has indeed changed.
bat.setReadMode	Change access privilige of BAT to read only
bat.setRole	Give a logical name to the columns of a BAT.
bat.setSet	Sets the 'set' property on this BAT to 'mode'. In 'set' mode, the kernel will silently block insertions that cause a duplicate BUN [head,tail] entries in the BAT. KNOWN BUG:when 'set' is set to TRUE, this function does not automatically eliminate duplicates. Use b := b.sunique; Returns the BAT itself.
bat.setSorted	Assure BAT is ordered on the head.
bat.setTransient	Make the BAT transient. Returns boolean which indicates if the BAT administration has indeed changed.
bat.setWriteMode	Change access privilige of BAT to read and write

bat.unload	Swapout a BAT to disk. Transient BATs can also be swapped out. Returns whether the unload indeed happened.
bat.unpack	Extract the first tuple from a bat
batcalc.!=	Equate a bat of strings against a singleton
batcalc.%	Binary BAT calculator function with new BAT result
batcalc.*	Binary BAT calculator function with new BAT result
batcalc.+	Concatenate two strings.
batcalc.++	Unary minus over the tail of the bat
batcalc.-	Unary minus over the tail of the bat
batcalc.–	Unary minus over the tail of the bat
batcalc./	Binary BAT calculator function with new BAT result
batcalc.<	Compare a bat of timestamp against a singleton
batcalc.<=	Compare a bat of timestamp against a singleton
batcalc.==	Equate a bat of strings against a singleton
batcalc.>	Compare a bat of timestamp against a singleton
batcalc.>=	Compare a bat of timestamp against a singleton
batcalc.and	Binary BAT calculator function with new BAT result
batcalc.bit	Coerce an str tail to a bat with bit tail.
batcalc.bte	Coerce an bit tail to a bat with bte tail.
batcalc.chr	
batcalc.chrAt	String array lookup operation.
batcalc.dbl	Coerce an flt tail to a bat with dbl tail.
batcalc.endsWith	Suffix check.
batcalc.flt	Coerce an dbl tail to a bat with flt tail.
batcalc.ifthen	Ifthen operation to assemble a conditional result
batcalc.ifthenelse	If-then-else operation to assemble a conditional result
batcalc.int	Coerce an str tail to a bat with int tail.
batcalc.isnil	Unary check for nil over the tail of the bat
batcalc.length	Return the length of a string.
batcalc.like	Perform SQL like operation against a string bat
batcalc.lng	Coerce an bit tail to a bat with lng tail.
batcalc.ltrim	Strip whitespaces from start of a string.
batcalc.match	POSIX pattern matching against a string BAT
batcalc.nbytes	Return the string length in bytes.
batcalc.not	Return a BAT with the negated tail
batcalc.oid	Coerce an lng tail to a bat with oid tail.
batcalc.or	Binary BAT calculator function with new BAT result
batcalc.r_search	Reverse search for a substring. Returns position, -1 if not found.
batcalc.rtrim	Strip whitespaces from end of a string.
batcalc.search	Search for a substring. Returns position, -1 if not found.
batcalc.sht	Coerce an bit tail to a bat with sht tail.
batcalc.startsWith	Prefix check.
batcalc.str	
batcalc.string	Return the tail s[offset..n] of a string s[0..n].

batcalc.substitute	Substitute first occurrence of 'src' by 'dst'. Iff repeated = true this is repeated while 'src' can be found in the result string. In order to prevent recursion and result strings of unlimited size, repeating is only done iff src is not a substring of dst.
batcalc.substring	Substring extraction using [start,start+length]
batcalc.toLower	Convert a string to lower case.
batcalc.toUpper	Convert a string to upper case.
batcalc.trim	Strip whitespaces around a string.
batcalc.unicodeAt	get a unicode character (as an int) from a string position.
batcalc.xml	
batcalc.xor	Binary BAT calculator function with new BAT result
batcolor.blue	Extracts blue component from a color atom
batcolor.cb	Extracts Cb(blue color) component from a color atom
batcolor.color	Converts string to color
batcolor.cr	Extracts Cr(red color) component from a color atom
batcolor.green	Extracts green component from a color atom
batcolor.hsv	Converts an HSV triplets to a color atom
batcolor.hue	Extracts hue component from a color atom
batcolor.luminance	Extracts Y(luminance) component from a color atom
batcolor.red	Extracts red component from a color atom
batcolor.rgb	Converts an RGB triplets to a color atom
batcolor.saturation	Extracts saturation component from a color atom
batcolor.str	Identity mapping for string bats
batcolor.value	Extracts value component from a color atom
batmath.acos	
batmath.asin	
batmath.atan	
batmath.atan2	
batmath.ceil	
batmath.cos	
batmath.cosh	
batmath.exp	
batmath.fabs	
batmath.floor	
batmath.fmod	
batmath.log	
batmath.log10	
batmath.pow	
batmath.sin	
batmath.sinh	
batmath.sqrt	
batmath.tan	
batmath.tanh	
batmtime.day	
batmtime.hours	

batmtime.milliseconds	
batmtime.month	
batmtime.seconds	
batmtime.year	
bbp.bind	Locate the BAT using its logical name
bbp.close	Close the bbp box.
bbp.commit	Commit updates for this client
bbp.deposit	Relate a logical name to a physical BAT in the buffer pool.
bbp.destroy	Schedule a BAT for removal at session end or immediately
bbp.discard	Remove the BAT from the box
bbp.find	Locate the BAT using its BBP index in the BAT buffer pool
bbp.getCount	Create a BAT with the cardinalities of all known BATs
bbp.getDirty	Create a BAT with the dirty/ diffs/clean status
bbp.getDiskSpace	Estimate the amount of diskspace occupied by dbfarm
bbp.getHeadType	Map a BAT into its head type
bbp.getHeat	Create a BAT with the heat values
bbp.getKind	Create a BAT with the persistency status
bbp.getLRefCount	Utility for debugging MAL interpreter
bbp.getLocation	Create a BAT with their disk locations
bbp.getName	Map a BAT into its internal name
bbp.getNames	Map BAT into its bbp name
bbp.getObjects	View of the box content.
bbp.getRNames	Map a BAT into its bbp physical name
bbp.getRefCount	Utility for debugging MAL interpreter
bbp.getStatus	Create a BAT with the disk/load status
bbp.getTailType	Map a BAT into its tail type
bbp.iterator	Locates the next element in the box
bbp.open	Locate the bbp box and open it.
bbp.prelude	Initialize the bbp box
bbp.release	Remove the BAT from further consideration
bbp.releaseAll	Commit updates for this client
bbp.take	Load a particular bat
bbp.toString	Get the string representation of an element in the box
blob.blob	Noop routine.
blob.nitems	get the number of bytes in this blob.
blob.prelude	
blob.toblob	store a string as a blob.
blob.tostring	get the bytes from blob as a string, starting at byte 'index' till the first 0 byte or the end of the blob.
box.close	Close the box.
box.deposit	Enter a new value into the box.
box.destroy	Destroy the box.
box.discard	Release the BAT from the client pool.
box.getBoxNames	Retrieve the names of all boxes.
box.iterator	Locates the next element in the box.

box.open	Locate the box and open it.
box.release	Release the BAT from the client pool.
box.releaseAll	Release all objects for this client.
box.take	Locates the typed value in the box.
box.toString	Get the string representation of the i-th element in the box.
bpm.adapt	Re-organize segment s using the selection (val1,val2) stored in bat rs
bpm.addPartition	Add a partition to a fragmented temporary table
bpm.close	Save and close the BAT partition box
bpm.count	
bpm.delete	Delete elements from the BAT partitions
bpm.deposit	Create a new partitioned BAT by name
bpm.derivePartition	Create a derived fragmentation over the head using src.
bpm.destroy	Destroy the BAT partition box
bpm.discard	Release all partitioned BATs
bpm.dump	Give the details of the partition tree
bpm.emptySet	Implement the constraints test efficiently
bpm.epilogue	
bpm.fold	Collapse the partitioned BAT into a single BAT
bpm.garbage	Remove a temporary partitioned table
bpm.getDimension	Obtain the partition boundary values.
bpm.getNames	Retrieve the names of all known partitioned BATs
bpm.getNumberOfPartitions	Return the number of partitions known
bpm.hasMoreElements	Localize the next partition for processing.
bpm.hashPartition	Create a hash partition on a BAT
bpm.hashPartitions	Ensure all partitions have a hash in the head
bpm.insert	Insert elements into the BAT partitions
bpm.mapAlias	
bpm.mapBid	
bpm.mapName	
bpm.mapNxt	
bpm.mapPrv	
bpm.mapThghDbl	
bpm.mapThghLng	
bpm.mapTlowDbl	
bpm.mapTlowLng	
bpm.new	Create a temporary partitioned table
bpm.newIterator	Create an iterator over the BAT partitions.
bpm.open	Locate and open the BAT partition box
bpm.partition	Split all partitions that cover the split value
bpm.pieces	Count the number of partitions
bpm.prelude	
bpm.rangePartition	Create the partitions based on a range vector
bpm.rangePartitionSort	Create the partitions based on a range vector
bpm.replace	Replace the content of the BAT partitions
bpm.saveCatalog	

bpm.select	Partitioned based selection
bpm.sortPartitions	Sort all partitions of alias b on the tail
bpm.sortTail	Implement the sort on tail for partitioned BAT efficiently
bpm.splitquant	Split all partitions to fit into a memory bound in KB
bpm.take	Retrieve a single component of a partitioned BAT by index
bpm.unfold	Unfold a BAT into a partitioned one
bstream.create	create a buffered stream
bstream.destroy	destroy bstream
bstream.read	read at least size bytes into the buffer of s
calc.!=	Equality of two timestamps
calc.%	
calc.*	
calc.+	Concatenate two strings
calc.-	negative value
calc./	
calc.<	Equality of two timestamps
calc.<<	
calc.<=	Equality of two timestamps
calc.==	Equality of two timestamps
calc.>	Equality of two timestamps
calc.>=	Equality of two timestamps
calc.>>	
calc.abs	absolute value
calc.and	
calc.bat	
calc.between	
calc.bit	coercion dbl to bit
calc.blob	
calc.bte	coercion lng to bte
calc.chr	coercion lng to chr
calc.date	
calc.daytime	
calc.dbl	coercion lng to dbl
calc.flt	coercion lng to flt
calc.getBAT	Coerce bat to BAT identifier
calc.getBATidentifier	Coerce bat to BAT identifier
calc.ifthenelse	
calc.int	coercion dbl to int
calc.inv	inverse value (1/x)
calc.isnil	Nil test for timestamp value
calc.isnotnil	is a value not equal to nil?
calc.length	
calc.lng	coercion dbl to lng
calc.max	
calc.min	
calc.newoid	Reserves a range of consecutive unique OIDs; returns the lowest in range. equivalent to newoid(0,incr)

calc.not	
calc.oid	coercion dbl to oid
calc.or	
calc.ptr	
calc.setoid	Equivalent to setoid(1:oid).
calc.sht	coercion dbl to sht
calc.sqlblob	
calc.str	coercion dbl to str
calc.timestamp	
calc.void	
calc.xml	
calc.xor	
clients.addScenario	add the given scenario to the allowed scenarios for the given user
clients.addUser	Allow user with password access to the given scenarios
clients.changePassword	Change the password for the current user
clients.changeUsername	Change the username of the user into the new string
clients.checkPermission	Check permission for a user
clients.exit	Terminate the session for a single client using a soft error.
clients.getActions	Pseudo bat of client's command counts.
clients.getId	Return a number that uniquely represents the current client.
clients.getInfo	Pseudo bat with client attributes.
clients.getLastCommand	Pseudo bat of client's last command time.
clients.getLogins	Pseudo bat of client login time.
clients.getScenario	Retrieve current scenario name.
clients.getTime	Pseudo bat of client's total time usage(in usec).
clients.getUsers	return a BAT with user id and name available in the system with access to the given scenario(s)
clients.quit	Terminate the server. This command can only be initiated from the console.
clients.removeScenario	remove the given scenario from the allowed scenarios for the given user
clients.removeUser	Remove the given user from the system
clients.setHistory	Designate console history file for readline.
clients.setListing	Turn on/off echo of MAL instructions: 2 - show mal instruction, 4 - show details of type resolutoin, 8 - show binding information.
clients.setPassword	Set the password for the given user
clients.setScenario	Switch to other scenario handler, return previous one.
clients.shutdown	Close all client connections. If forced=false the clients are moved into FINISHING mode, which means that the process stops at the next cycle of the scenario. If forced=true all client processes are immediately killed

clients.suspend	Put a client process to sleep for some time. It will simple sleep for a second at a time, until the awake bit has been set in its descriptor
clients.wakeup	Wakeup a client process
color.blue	Extracts blue component from a color atom
color.cb	Extracts Cb(blue color) component from a color atom
color.color	Converts string to color
color.cr	Extracts Cr(red color) component from a color atom
color.green	Extracts green component from a color atom
color.hsv	Converts an HSV triplets to a color atom
color.hue	Extracts hue component from a color atom
color.luminance	Extracts Y(luminance) component from a color atom
color.print	
color.printf	
color.red	Extracts red component from a color atom
color.rgb	Converts an RGB triplets to a color atom
color.saturation	Extracts saturation component from a color atom
color.str	Converts color to string
color.value	Extracts value component from a color atom
color.ycc	Converts an YCC triplets to a color atom
const.close	Close the constant box
const.deposit	Enter a new variable into the box
const.destroy	Destroy the box
const.discard	Release the const from the box
const.epiloque	Cleanup the const box
const.hasMoreElements	Locate next element in the box
const.newIterator	Locate next element in the box
const.open	Locate and open the constant box
const.prelude	Initialize the const box
const.release	Release a new constant value
const.releaseAll	Release all variables in the box
const.take	Take a variable out of the box
const.toString	Get the string representation of an element in the box
constraints.emptySet	Check if the bat is empty
crackers.DeleteMap	Throw away a certain map
crackers.InsertAVLIndex	Insert u in the AVL tree index of BAT b
crackers.buildAVLIndex	Create an AVL tree index for this BAT
crackers.crackOrdered	Break a BAT into three pieces with tail<mid, tail==mid, tail>mid, respectively; maintaining the head-oid order within each piece.
crackers.crackOrdered_validate	Validate whether a BAT is correctly broken into five pieces with tail<low, tail==low, low<tail<hgh, tail==hgh, tail>hgh, respectively; maintaining the head-oid order within each piece.

crackers.crackUnordered_validate	Validate whether a BAT is correctly broken into five pieces with tail<low, tail==low, low<tail<hgh, tail==hgh, tail>hgh, respectively.
crackers.deleteAVL	Delete a collection of values from the index
crackers.deletionsOnNeed	Keep the deletions BAT separatelly and do a complete merge only if a relevant query arrives in the future
crackers.deletionsOnNeedGradually	Keep the deletions BAT separatelly and merge only what is needed if a relevant query arrives in the future
crackers.deletionsOnNeedGraduallyRipple	Keep the deletions BAT separatelly and merge only what is needed using ripple if a relevant query arrives in the future
crackers.djoinselect	Use the pivot. For each tuple in pivot with a 0, check if the respective tuple (in the same position) in the tail of cpair satisfies the range restriction. If yes mark the pivot BUN as 1.
crackers.dproject	Sync the cracking pair and project the tail. Use for disjunctive queries that require a larger bit vector
crackers.dselect	Crack based on date and evaluate the date disjunctive predicate outside the cracked area. Return a bit vector.
crackers.extendCrackerBAT	Extend the cracker column by P positions
crackers.extendCrackerMap	Extend the cracker map by P positions
crackers.getCrackerBAT	Get the cracker BAT of b
crackers.hselect	Retrieve the subset head using a cracker index producing preferably a BATview.
crackers.insert	Keep the insertions BAT separatelly and merge in the future on demand with the Ripple
crackers.insertionsBForce	Merge the insertions BAT with the cracker bat and update the cracker index
crackers.insertionsBOnNeed	Keep the insertions BAT separatelly and do a complete merge only if a relevant query arrives in the future
crackers.insertionsBOnNeedGradually	Keep the insertions BAT separatelly and merge only what is needed if a relevant query arrives in the future
crackers.insertionsBOnNeedGraduallyRipple	Keep the insertions BAT separatelly and merge only what is needed using the ripple strategy if a relevant query arrives in the future
crackers.insertionsForget	Append c to the cracked BAT of b and completelly forget the cracker index
crackers.insertionsPartiallyForget	Append c to the cracked BAT of b and partially forget the cracker index, i.e., forget only what is affected
crackers.joinselect	Use the pivot. For each tuple in pivot with a 1, check if the respective tuple (in the same position) in the tail of cpair satisfies the range restriction. If not mark the pivot BUN as 0.

crackers.joinuselect	Join left and right on head-OIDs. From right, only those BUNs qualify that satisfy the range-restriction on the tail. If inPlace is TRUE (and left has an OID head and is not a BAT-view), we operate in-place, overwriting left and returning it as result. Otherwise, the result is a new [:oid,:void] BAT. If isForeignKey is TRUE, we assume that each tuple from left finds a match in right, and hence skip the respective check. (NOTE: This may lead to CRASHES, if isForeignKey is incorrectly passed as TRUE!)
crackers.printAVLTree_int	Print the AVL Tree of the cracker index (for debugging purposes)
crackers.printCrackerBAT	Print the cracker BAT of b
crackers.printCrackerDeletions	Print the pending deletions of the cracker BAT of b
crackers.printCrackerIndexBATpart	Print the cracker index of b
crackers.printCrackerInsertions	Print the pending insertions of the cracker BAT of b
crackers.printPendingInsertions	Print the pending insertions
crackers.project	Sync the cracking pair and project the tail
crackers.projectH	Sync the cracking pair and project the head
crackers.select	Retrieve the subset using a cracker index producing preferably a BATview.
crackers.selectAVL	Retrieve the subset using the AVL index
crackers.sizeCrackerDeletions	Get the size of the pending deletions of the cracker BAT of b
crackers.sizeCrackerInsertions	Get the size of the pending insertions of the cracker BAT of b
crackers.sizePendingInsertions	Get the size of the pending insertions for this map
crackers.tselect	Retrieve the subset tail using a cracker index producing preferably a BATview.
crackers.uselect	Retrieve the subset using a cracker index producing preferably a BATview.
crackers.verifyCrackerIndex	Check the cracker index and column, whether each value is in the correct chunk
crackers.zcrackOrdered	Break a BAT into three pieces with tail<=low, low<tail<=hgh, tail>hgh, respectively; maintaining the head-oid order within each piece.
crackers.zcrackOrdered_validate	Validate whether a BAT is correctly broken into three pieces with tail<=low, low<tail<=hgh, tail>hgh, respectively; maintaining the head-oid order within each piece.
crackers.zcrackUnordered	Break a BAT into three pieces with tail<=low, low<tail<=hgh, tail>hgh, respectively.
crackers.zcrackUnordered_validate	Validate whether a BAT is correctly broken into three pieces with tail<=low, low<tail<=hgh, tail>hgh, respectively.
date.!=	Equality of two dates
date.<	Equality of two dates
date.<=	Equality of two dates

date.==	Equality of two dates
date.>	Equality of two dates
date.>=	Equality of two dates
date.date	Noop routine.
date.isnil	Nil test for date value
daytime.!=	Equality of two daytimes
daytime.<	Equality of two daytimes
daytime.<=	Equality of two daytimes
daytime.==	Equality of two daytimes
daytime.>	Equality of two daytimes
daytime.>=	Equality of two daytimes
daytime.isnil	Nil test for daytime value
factory.getArrival	Retrieve the time stamp the last call was made
factory.getCaller	Retrieve the unique identity of the factory caller
factory.getDeparture	Retrieve the time stamp the last answer was returned
factory.getOwners	Retrieve the factory owners table
factory.getPlants	Retrieve the names for all active factories
factory.shutdown	Close a factory
group.avg	grouped tail average
group.count	Grouped count
group.derive	Cross tabulation group extension step. Returned head values are identical as in 'ct'. Tail values are from the same domain and indicate further refinement of the groups in 'ct', taking into account also the tail-values in 'attr'.
group.max	Select the minimum element of each group
group.min	Select the minimum element of each group
group.new	Cross tabulation group initialization like GRPgroup, but with user provided #bits in hashmask and #distinct values in range.
group.prelude	
group.refine	refine the ordering of a tail-ordered BAT by sub-ordering on the values of a second bat 'a' (where the heads of a and b match 1-1). The effect of this is similar to (hash-based) GRPderive, with the distinction that the group ids respect the ordering of the group values.
group.refine_reverse	refine the ordering of a tail-ordered BAT by sub-ordering on the values of a second bat 'a' (where the heads of a and b match 1-1). The effect of this is similar to (hash-based) GRPderive, with the distinction that the group ids respect the ordering of the group values.
group.size	Grouped count of true values
group.sum	Tail sum of groups of a sliding window of fixed size
inet.!=	Inequality of two inets
inet.<	Whether v is less than w
inet.<<	Whether v is contained within w
inet.<<=	Whether v is contained within or is equal to w

inet.<=	Whether v is less than or equal to w
inet.=	Equality of two inets
inet.>	Whether v is greater than w
inet.>=	Whether v is equal to or greater than w
inet.>>	Whether v contains w
inet.>>=	Whether v contains or is equal to w
inet.abbrev	Abbreviated display format as text
inet.broadcast	Returns the broadcast address for network
inet.host	Extract IP address as text
inet.hostmask	Construct host mask for network
inet.isnil	Nil test for inet value
inet.masklen	Extract netmask length
inet.netmask	Construct netmask for network
inet.network	Extract network part of address
inet.new	Create an inet from a string literal
inet.setmasklen	Set netmask length for inet value
inet.text	Extract IP address and netmask length as text
inspect.getAddress	Returns the function signature(s).
inspect.getAddresses	Obtain the function address.
inspect.getAtomNames	Collect a BAT with the atom names.
inspect.getAtomSizes	Collect a BAT with the atom sizes.
inspect.getAtomSuper	Collect a BAT with the atom names.
inspect.getComment	Returns the function help information.
inspect.getDefinition	Returns a string representation of a specific function.
inspect.getEnvironment	Collect the environment variables.
inspect.getFunction	Obtain the function name.
inspect.getKind	Obtain the instruction kind.
inspect.getModule	Obtain the function name.
inspect.getSignature	Returns the function signature(s).
inspect.getSignatures	Obtain the function signatures.
inspect.getSize	Return the storage size for a function (in bytes).
inspect.getSource	Return the original input for a function.
inspect.getStatistics	Get optimizer property statistics such as #calls, #total actions, #total time
inspect.getType	Return the type of a variable (expression).
inspect.getTypeIndex	Return the type index of a variable.
inspect.getTypeName	Get the type name associated with a type id.
io.export	Export a BAT as ASCII to a file. If the 'filepath' is not absolute, it is put into the .../dbfarm/$DB directory. Success of failure is indicated.
io.ftable	Print an n-ary table to a file.
io.import	Import a BAT from an ASCII dump. The new tuples are *inserted* into the parameter BAT. You have to create it! Its signature must match the dump, else parsing errors will occur and FALSE is returned.
io.print	Print a MAL value tuple .

io.printf	
io.printft	Select default format
io.prompt	Print a MAL value without brackets.
io.stderr	return the error stream for the database console
io.stdin	return the input stream to the database client
io.stdout	return the output stream for the database client
io.table	Print an n-ary table.
language.assert	Assertion test
language.assertSpace	Ensures that the current call does not consume more then depth*vtop elements on the stack
language.call	Evaluate a program stored in a BAT
language.dataflow	The current guarded block is executed using dataflow control
language.newRange	This routine introduces an iterator over a scalar domain.
language.nextElement	Advances the iterator with a fixed value until it becomes >= last.
language.raise	Raise an exception labeled with a specific message.
language.register	Compile the code string and register it as a MAL function
language.source	Merge the instructions stored in the file with the current program
lock.create	Create an unset lock
lock.destroy	Destroy a lock
lock.set	Try to set a lock. If set, block till it is freed
lock.tostr	Overloaded atom function
lock.try	Try a lock. If free set it, if not return EBUSY
lock.unset	Unset a lock
mal.multiplex	
manual.completion	Produces the wordcompletion table.
manual.createXML	Produces a XML-formatted manual over all modules loaded.
manual.help	Produces a list of all <module>.<function> that match the text pattern. The wildcard '*' can be used for <module> and <function>. Using the '(' asks for signature information and using ')' asks for the complete help record.
manual.index	Produces an overview of all names grouped by module.
manual.search	Search the manual for command descriptions that match the regular expression 'text'
manual.section	Generate a synopsis of a module for the reference manual
manual.summary	Produces a manual with help lines grouped by module.
mat.hasMoreElements	Find the next element in the merge table
mat.info	retrieve the definition from the partition catalogue
mat.new	Define a Merge Association Table (MAT)
mat.newIterator	Create an iterator over a MAT
mat.pack	Materialize the MAT into the first BAT
mat.print	
mdb.List	Dump the routine M.F on standard out.

mdb.collect	Dump the previous instruction to a temporary file
mdb.dot	Dump the data flow of the function M.F in a format recognizable by the command 'dot' on the file s
mdb.dump	Dump instruction, stacktrace, and stack
mdb.getContext	Extract the context string from the exception message
mdb.getDebug	Get the kernel debugging bit-set. See the MonetDB configuration file for details
mdb.getDefinition	Returns a string representation of the current function with typing information attached
mdb.getException	Extract the variable name from the exception message
mdb.getReason	Extract the reason from the exception message
mdb.getStackDepth	Return the depth of the calling stack.
mdb.getStackFrame	Collect variable binding of current (n-th) stack frame.
mdb.getStackTrace	
mdb.grab	Stop and debug another client process.
mdb.inspect	Run the debugger on a specific function
mdb.list	Dump the routine M.F on standard out.
mdb.listMapi	Dump the current routine on standard out with Mapi prefix.
mdb.setBigfoot	Turn on/off memory foot print for debugger
mdb.setCatch	Turn on/off catching exceptions
mdb.setDebug	Set the kernel debugging bit-set and return its previous value.
mdb.setFlow	Turn on/off memory flow debugger
mdb.setIO	Turn on/off io statistics tracing
mdb.setMemory	Turn on/off memory statistics tracing.
mdb.setTimer	Turn on/off performance timer for debugger
mdb.setTrace	Turn on/off tracing of a variable
mdb.start	Start interactive debugger on a running factory
mdb.stop	Stop the interactive debugger
mdb.var	Dump the symboltable of routine M.F on standard out.
mkey.bulk_rotate_xor_hash	pre: h and b should be synced on head post: [:xor=]([:rotate=](h, nbits), [hash](b))
mkey.hash	compute a hash int number from any value
mkey.rotate	left-rotate an int by nbits
mmath.acos	The acos(x) function calculates the arc cosine of x, that is the value whose cosine is x. The value is returned in radians and is mathematically defined to be between 0 and PI (inclusive).
mmath.asin	The asin(x) function calculates the arc sine of x, that is the value whose sine is x. The value is returned in radians and is mathematically defined to be between -PI/20 and -PI/2 (inclusive).

mmath.atan	The atan(x) function calculates the arc tangent of x, that is the value whose tangent is x. The value is returned in radians and is mathematically defined to be between -PI/2 and PI/2 (inclusive).
mmath.atan2	The atan2(x,y) function calculates the arc tangent of the two variables x and y. It is similar to calculating the arc tangent of y / x, except that the signs of both arguments are used to determine the quadrant of the result. The value is returned in radians and is mathematically defined to be between -PI/2 and PI/2 (inclusive).
mmath.ceil	The ceil(x) function rounds x upwards to the nearest integer.
mmath.cos	The cos(x) function returns the cosine of x, where x is given in radians. The return value is between -1 and 1.
mmath.cosh	The cosh() function returns the hyperbolic cosine of x, which is defined mathematically as (exp(x) + exp(-x)) / 2.
mmath.exp	The exp(x) function returns the value of e (the base of natural logarithms) raised to the power of x.
mmath.fabs	The fabs(x) function returns the absolute value of the floating-point number x.
mmath.finite	The finite(x) function returns true if x is neither infinite nor a 'not-a-number' (NaN) value, and false otherwise.
mmath.floor	The floor(x) function rounds x downwards to the nearest integer.
mmath.fmod	The fmod(x,y) function computes the remainder of dividing x by y. The return value is x - n * y, where n is the quotient of x / y, rounded towards zero to an integer.
mmath.isinf	The isinf(x) function returns -1 if x represents negative infinity, 1 if x represents positive infinity, and 0 otherwise.
mmath.isnan	The isnan(x) function returns true if x is 'not-a-number' (NaN), and false otherwise.
mmath.log	The log(x) function returns the natural logarithm of x.
mmath.log10	The log10(x) function returns the base-10 logarithm of x.
mmath.pi	return an important mathematical value
mmath.pow	The pow(x,y) function returns the value of x raised to the power of y.
mmath.rand	return a random number
mmath.round	The round(n, m) returns n rounded to m places to the right of the decimal point; if m is omitted, to 0 places. m can be negative to round off digits left of the decimal point. m must be an integer.
mmath.sin	The sin(x) function returns the cosine of x, where x is given in radians. The return value is between -1 and 1.

mmath.sinh	The sinh() function returns the hyperbolic sine of x, which is defined mathematically as (exp(x) - exp(-x)) / 2.
mmath.sqrt	The sqrt(x) function returns the non-negative square root of x.
mmath.srand	initialize the rand() function with a seed
mmath.tan	The tan(x) function returns the tangent of x, where x is given in radians
mmath.tanh	The tanh() function returns the hyperbolic tangent of x, which is defined mathematically as sinh(x) / cosh(x).
mserver.bind	Bind a remote variable to a local one
mserver.connect	Establish connection with a remote mserver
mserver.connect_ssl	Establish connection with a remote mserver using the secure socket layer
mserver.destroy	Destroy the handle
mserver.disconnect	Terminate the session
mserver.error	Check for an error in the communication
mserver.explain	Turn the error seen into a string
mserver.fetch_all_rows	Retrieve all rows into the cache
mserver.fetch_field	Retrieve a single chr field
mserver.fetch_field_array	Retrieve all fields for a row
mserver.fetch_line	Retrieve a complete line
mserver.fetch_reset	Reset the cache read line.
mserver.fetch_row	Retrieve the next row for analysis
mserver.finish	Remove all remaining answers
mserver.getError	Get error message
mserver.get_field_count	Return number of fields
mserver.get_row_count	Return number of rows
mserver.listen	Start the Mapi listener on <port> for <maxusers>. For a new client connection MAL procedure <cmd>(Stream s_in, Stream s_out) is called.If no <cmd> is specified a new client thread is forked.
mserver.listen_ssl	Start the Mapi listener on <port> for <maxusers> using SSL. <keyfile> and <certfile> give the path names for files with the server key and certificates in PEM format. For a new client connection MAL procedure <cmd>(Stream s_in, Stream s_out) is called. If no <cmd> is specified a new client thread is forked.
mserver.lookup	Retrieve the connection identifier
mserver.malclient	Start a Mapi client.
mserver.next_result	Go to next result set
mserver.ping	Test availability of server
mserver.prepare	Prepare a query for execution
mserver.put	Prepare sending a value to a remote site
mserver.query	Sent the query for execution
mserver.query_array	Sent the query for execution replacing '?' by arguments
mserver.query_handle	Sent the query for execution

mserver.reconnect	Re-establish a connection
mserver.resume	Resume connection listeners
mserver.rpc	Sent a simple query for execution
mserver.setAlias	Give the channel a logical name
mserver.stop	Terminate connection listeners
mserver.suspend	Suspend accepting connections
mserver.trace	Toggle the Mapi library tracer
mtime.add	returns the timestamp that comes 'msecs' (possibly negative) after 'value'.
mtime.adddays	returns the date after a number of days (possibly negative).
mtime.addmonths	returns the date after a number of months (possibly negative).
mtime.addyears	returns the date after a number of years (possibly negative).
mtime.compute	compute the date from a rule in a given year
mtime.current_date	
mtime.current_time	
mtime.current_timestamp	
mtime.date	extracts date from timestamp in a specific timezone.
mtime.date_add_month_interval	Add months to a date
mtime.date_add_sec_interval	Add seconds to a date
mtime.date_sub_sec_interval	Subtract seconds from a date
mtime.day	extract day from rule.
mtime.dayname	Returns day name from a number between [1-7], str(nil) otherwise.
mtime.daynum	Returns number of day [1-7] from a string or nil if does not match any.
mtime.dayofweek	Returns the current day of the week where 1=sunday, .., 7=saturday
mtime.dayofyear	Returns N where d is the Nth day of the year (january 1 returns 1)
mtime.daytime	default time with zeroed components
mtime.diff	returns the number of milliseconds between 'val1' and 'val2'.
mtime.dst	return whether DST holds in the timezone at a certain point of time.
mtime.end_dst	extract rule that determines end of DST from timezone.
mtime.epilogue	
mtime.hours	extracts hour from daytime
mtime.local_timezone	get the local timezone in seconds
mtime.milliseconds	extracts milliseconds from daytime
mtime.minutes	extract minutes from rule.
mtime.month	extract month from rule.
mtime.monthname	Returns month name from a number between [1-12], str(nil) otherwise.

mtime.monthnum	Returns month number [1-12] from a string or nil if does not match any.
mtime.msec	get time of day in msec since 1-1-1970.
mtime.msecs	convert date components to milliseconds
mtime.olddate	create a date from the old instant format.
mtime.oldduration	parse the old duration format and return an (estimated) number of days.
mtime.prelude	
mtime.rule	create a DST start/end date rule.
mtime.seconds	extracts seconds from daytime
mtime.setTimezone	Test and set the timezone.
mtime.start_dst	extract rule that determines start of DST from timezone.
mtime.time_add_sec_interval	Add seconds to a time
mtime.time_sub_sec_interval	Subtract seconds from a time
mtime.time_synonyms	Allow synonyms for the parse format of date/timestamp.
mtime.timestamp	creates a timestamp from (d,00:00:00) parameters (in the local timezone).
mtime.timestamp_add_month_interval	Add months to a timestamp
mtime.timestamp_add_sec_interval	
mtime.timestamp_sub_month_interval	Subtract months from a timestamp
mtime.timestamp_sub_sec_interval	
mtime.timezone	create a timezone as an hour difference from GMT and a DST.
mtime.timezone_local	get the local timezone; which is used for printing timestamps
mtime.weekday	extract weekday from rule.
mtime.weekofyear	Returns the week number in the year.
mtime.year	extracts year from date (nonzero value between -5867411 and +5867411).
optimizer.accessmode	Reduce the number of mode changes.
optimizer.accumulators	Replace calculations with accumulator model
optimizer.aliases	Alias removal optimizer
optimizer.clrDebug	
optimizer.coercions	Handle simple type coercions
optimizer.commonTerms	Common sub-expression optimizer
optimizer.costModel	Estimate the cost of a relational expression
optimizer.crack	Replace algebra select with crackers select
optimizer.deadcode	Dead code optimizer
optimizer.dumpQEP	Produce an indented tree visualisation
optimizer.emptySet	Symbolic evaluation of empty BAT expressions
optimizer.evaluate	Evaluate constant expressions once.
optimizer.factorize	Turn function into a factory
optimizer.garbageCollector	Garbage collector optimizer
optimizer.heuristics	Handle simple replacements
optimizer.inline	Expand inline functions
optimizer.joinPath	Join path constructor

optimizer.joinselect	Replace select with join select
optimizer.macro	Inline a target function used in a specific function.
optimizer.mergetable	Resolve the multi-table definitions
optimizer.multiplex	Compiler for multiplexed instructions.
optimizer.optimize	Optimize a specific operation
optimizer.orcam	Inverse macro, find pattern and replace with a function call.
optimizer.partitions	Experiment with partitioned databases
optimizer.peephole	Perform local rewrites
optimizer.prelude	Initialize the optimizer
optimizer.pushranges	Push constant range selections through the program
optimizer.reduce	Reduce the stack space claims
optimizer.remap	Remapping function calls to a their multiplex variant
optimizer.remoteQueries	Resolve the multi-table definitions
optimizer.setDebug	
optimizer.showFlowGraph	Dump the data flow of the function M.F in a format recognizable by the command 'dot' on the file s
optimizer.showPlan	Illustrate the plan derived so far
optimizer.singleton	Perform singleton optimization
optimizer.strengthReduction	Move constant expressions out of the loop
optimizer.trace	Collect trace of a specific operation
pcre.compile	compile a pattern
pcre.like	
pcre.match	POSIX pattern matching against a string
pcre.patindex	Location of the first POSIX pattern matching against a string
pcre.pcre_quote	Return a PCRE pattern string that matches the argument exactly.
pcre.prelude	Initialize pcre
pcre.replace	
pcre.select	Select tuples based on the pattern
pcre.uselect	Select tuples based on the pattern, only returning the head
pqueue.dequeue_max	Removes top element of the max-pqueue and updates it
pqueue.dequeue_min	Removes top element of the min-pqueue and updates it
pqueue.enqueue_max	Inserts element (oid,dbl) in the max-pqueue
pqueue.enqueue_min	Inserts element (oid,dbl) in the min-pqueue
pqueue.init	Creates an empty pqueue of bat a's tailtype with maximum size maxsize
pqueue.topn_max	Return the topn elements of the bat t using a max-pqueue
pqueue.topn_min	Return the topn elements of the bat t using a min-pqueue
pqueue.topreplace_max	Replaces top element with input and updates max-pqueue
pqueue.topreplace_min	Replaces top element with input and updates min-pqueue
profiler.activate	Make the specified counter active.
profiler.cleanup	Remove the temporary tables for profiling
profiler.closeStream	Stop sending the event records
profiler.clrFilter	Stop tracing the variable

profiler.deactivate	Deactivate the counter
profiler.dumpTrace	List the events collected
profiler.getEvent	Retrieve the performance indicators of the previous instruction
profiler.getTrace	Get the trace details of a specific event
profiler.noop	Fetch any pending performance events
profiler.openStream	Send the log events to a stream
profiler.reset	Clear the profiler traces
profiler.setAll	Short cut for setFilter(*,*).
profiler.setEndPoint	End performance tracing after mod.fcn
profiler.setFilter	Generate an event record for every instruction where v is used;
profiler.setNone	Short cut for clrFilter(*,*).
profiler.setStartPoint	Start performance tracing at mod.fcn
profiler.start	Start performance tracing
profiler.stop	Stop performance tracing
scheduler.choice	Select the next step in a query memo plan
scheduler.costPrediction	A sample cost prediction function
scheduler.isolation	Run a private copy of the MAL program
scheduler.pick	Pick up the first result
scheduler.volumeCost	A sample cost function based on materialized results
sema.create	Create an unset sema, with an initial value
sema.destroy	Destroy a semaphore
sema.down	Decrement the semaphpore if >0; else block
sema.up	Increment the semaphore
sqlblob.sqlblob	Noop routine.
statistics.close	Close the statistics box
statistics.deposit	Enter a new BAT into the statistics box
statistics.destroy	Destroy the statistics box
statistics.discard	Release a BAT variable from the box
statistics.dump	Display the statistics table
statistics.epilogue	Release the resources of the statistics package
statistics.forceUpdate	Bring the statistics up to date for one BAT
statistics.getCount	Return latest stored count information
statistics.getHistogram	Return the latest histogram
statistics.getHotset	Return a table with BAT names that have been touched since the start of the session
statistics.getMax	Return latest stored maximum information
statistics.getMin	Return latest stored minimum information
statistics.getObjects	Return a table with BAT names managed
statistics.getSize	Return latest stored count information
statistics.hasMoreElements	Locate next element in the box
statistics.newIterator	Locate next element in the box
statistics.open	Locate and open the statistics box
statistics.prelude	Initialize the statistics package
statistics.release	Release a single BAT from the box

statistics.releaseAll	Release all variables in the box
statistics.take	Take a variable out of the statistics box
statistics.toString	Get the string representation of an element in the box
statistics.update	Check for stale information
status.batStatistics	Show distribution of bats by kind
status.cpuStatistics	Global cpu usage information
status.getDatabases	Produce a list of known databases in the current dbfarm
status.getPorts	Produce a list of default ports for a specific language
status.getThreads	Produce overview of active threads
status.ioStatistics	Global IO activity information
status.memStatistics	Global memory usage information
status.memUsage	Get a split-up of how much memory blocks are in use.
status.mem_cursize	the amount of physical swapspace in KB that is currently in use
status.mem_maxsize	set the maximum usable amount of physical swapspace in KB
status.vmStatistics	Get a split-up of how much virtual memory blocks are in use.
status.vm_cursize	the amount of logical VM space in KB that is currently in use
status.vm_maxsize	set the maximum usable amount of physical swapspace in KB
str.+	Concatenate two strings.
str.STRepilogue	
str.STRprelude	
str.ascii	Return unicode of head of string
str.chrAt	String array lookup operation.
str.codeset	Return the locale's codeset
str.endsWith	Suffix check.
str.iconv	String codeset conversion
str.insert	Insert a string into another
str.length	Return the length of a string.
str.like	SQL pattern match function
str.locate	Locate the start position of a string
str.ltrim	Strip whitespaces from start of a string.
str.nbytes	Return the string length in bytes.
str.prefix	Extract the prefix of a given length
str.r_search	Reverse search for a char. Returns position, -1 if not found.
str.repeat	
str.replace	Insert a string into another
str.rtrim	Strip whitespaces from end of a string.
str.search	Search for a character. Returns position, -1 if not found.
str.space	
str.startsWith	Prefix check.
str.str	Noop routine.
str.string	Return substring s[offset..offset+count] of a string s[0..n]

str.stringleft	
str.stringlength	Return the length of a right trimed string (SQL semantics).
str.stringright	
str.substitute	Substitute first occurrence of 'src' by 'dst'. Iff repeated = true this is repeated while 'src' can be found in the result string. In order to prevent recursion and result strings of unlimited size, repeating is only done iff src is not a substring of dst.
str.substring	Extract a substring from str starting at start, for length len
str.suffix	Extract the suffix of a given length
str.toLower	Convert a string to lower case.
str.toUpper	Convert a string to upper case.
str.trim	Strip whitespaces around a string.
str.unicode	convert a unicode to a character.
str.unicodeAt	get a unicode character (as an int) from a string position.
streams.blocked	open a block based stream
streams.close	close and destroy the stream s
streams.flush	flush the stream
streams.openRead	convert an ascii stream to binary
streams.openReadBytes	open a file stream for reading
streams.openWrite	convert an ascii stream to binary
streams.openWriteBytes	open a file stream for writing
streams.readInt	read integer data from the stream
streams.readStr	read string data from the stream
streams.socketRead	open ascii socket stream for reading
streams.socketReadBytes	open a socket stream for reading
streams.socketWrite	open ascii socket stream for writing
streams.socketWriteBytes	open a socket stream for writing
streams.writeInt	write data on the stream
streams.writeStr	write data on the stream
tablet.display	Display a formatted table
tablet.dump	Print all pages with header to a stream
tablet.finish	Free the storage space of the report descriptor
tablet.firstPage	Produce the first page of output
tablet.getPage	Produce the i-th page of output
tablet.getPageCnt	Return the size in number of pages
tablet.header	Display the minimal header for the table
tablet.input	Load a bat using specific format.
tablet.lastPage	Produce the last page of output
tablet.load	Load a bat using specific format.
tablet.nextPage	Produce the next page of output
tablet.output	Send the bat to an output stream.
tablet.page	Display all pages at once without header
tablet.prevPage	Produce the prev page of output
tablet.setBracket	Format the brackets around a field

tablet.setColumn	Bind i-th output column to a variable
tablet.setDecimal	Set the scale and precision for numeric values
tablet.setDelimiter	Set the column separator.
tablet.setFormat	Initialize a new reporting structure.
tablet.setName	Set the display name for a given column
tablet.setNull	Set the display format for a null value for a given column
tablet.setPivot	The pivot bat identifies the tuples of interest. The only requirement is that all keys mentioned in the pivot tail exist in all BAT parameters of the print comment. The pivot also provides control over the order in which the tuples are produced.
tablet.setPosition	Set the character position to use for this field when loading according to fixed (punch-card) layout.
tablet.setProperties	Define the set of properties
tablet.setRowBracket	Format the brackets around a row
tablet.setStream	Redirect the output to a stream.
tablet.setTableBracket	Format the brackets around a table
tablet.setWidth	Set the maximal display witdh for a given column. All values exceeding the length are simple shortened without any notice.
timestamp.!=	Equality of two timestamps
timestamp.<	Equality of two timestamps
timestamp.<=	Equality of two timestamps
timestamp.==	Equality of two timestamps
timestamp.>	Equality of two timestamps
timestamp.>=	Equality of two timestamps
timestamp.epoch	unix-time (epoch) support: seconds since epoch
timestamp.isnil	Nil test for timestamp value
timestamp.unix_epoch	The Unix epoch time (00:00:00 UTC on January 1, 1970)
timezone.str	
timezone.timestamp	Utility function to create a timestamp from a number of seconds since the Unix epoch
transaction.abort	Abort changes in certain BATs.
transaction.alpha	List insertions since last commit.
transaction.clean	Declare a BAT clean without flushing to disk.
transaction.commit	Commit changes in certain BATs.
transaction.delta	List deletions since last commit.
transaction.prev	The previous stae of this BAT
transaction.subcommit	commit only a set of BATnames, passed in the tail (to which you must have exclusive access!)
transaction.sync	Save all persistent BATs
unix.getenv	Get the environment variable string.
unix.setenv	Set the environment variable string.
url.getAnchor	Extract the URL anchor (reference)
url.getBasename	Extract the URL base file name
url.getContent	Get the URL resource in a local file

url.getContext	Get the path context of a URL
url.getDirectory	Extract directory names from the URL
url.getDomain	Extract Internet domain from the URL
url.getExtension	Extract the file extension of the URL
url.getFile	Extract the last file name of the URL
url.getHost	Extract the server name from the URL
url.getPort	Extract the port id from the URL
url.getProtocol	Extract the protocol from the URL
url.getQuery	Extract the query string from the URL
url.getQueryArg	Extract argument mappings from the URL
url.getRobotURL	Extract the location of the robot control file
url.getUser	Extract the user identity from the URL
url.isaURL	Check conformity of the URL syntax
url.new	Construct URL from protocol, host,and file
url.url	Create an URL from a string literal
user.main	
xml.agg	Aggregate the XML values over grouping specified
xml.attribute	Construct an attribute value pair
xml.comment	Construct an comment struction
xml.concat	Concatenate the xml values
xml.content	Parse the string as a document
xml.document	Parse the string as a document
xml.element	The basic building block for XML elements are namespaces, attributes and a sequence of xml elements. The name space and the attributes may be left unspecified(=nilt).
xml.forest	Construct an element list
xml.isdocument	Validate the string as a document
xml.options	Create the components including NULL conversions
xml.parse	Parse the XML document or element string values
xml.pi	Call the processing instruction
xml.prelude	
xml.root	Contruct the root nodes
xml.serialize	Serialize the XML object to a string
xml.str	Cast the string to an xml compliant string
xml.tag	Routine to put element brackets around an XML value
xml.text	Serialize the XQuery object to a string
xml.xml	Cast the string to an xml compliant string
xml.xquery	Execute the XQuery against the elements
zrule.define	Introduce a synomym timezone rule.

Appendix D Historical Background on the VOC

For your convenience, we give here a short histororical background and interpretation of what to find in this database. The introduction given below is an OCR version from the book: J.R. Bruijn, F.S. Gaastra and I. Schaar, Dutch-Asiatic Shipping in the 17th and 18th Centuries

The history of the VOC is an active area of research and a focal point for multi-country heritage projects, e.g. TANAP, which includes a short historic overview of the VOC written by world expert on the topic F. Gaastra. The archives of the VOC are spread around the world, but a large contingent still resides in the National Archive, The Hague. The archives comprise over 25 million historical records. Much of which has not (yet) been digitized.

This book presents tables which give a virtually complete survey of the direct shipping between the Netherlands and Asia between 1595-1795. This period contains, first, the voyages of the so-called Voorcompagnien and, hence, those for and under control of the Vereenigde Oostindische Compagnie (VOC). The survey ends in 1795. That year saw an end of the regular sailings of the VOC between the Netherlands and Asia, since, following the Batavian revolution in January, the Netherlands became involved in war with England. The last outward voyage left on 26 December 1794. After news of the changed situation in the Netherlands was received in Asia, the last homeward voyage took place in the spring of 1795. The VOC itself was disbanded in 1798.

In total 66 voyages of the voorcompagnien are listed, one more than the traditionally accepted number. The reconnaissance ship, POSTILJON, from the fleet of Mahu and De Cordes, that was collected en route is given its own number (0022). Since the attempt of the Australische Compagnie to circumvent the monopoly of the VOC can be considered as a continuation of the voorcompagnien the voyage of Schouten and Le Maire is also listed (0196-0197). For the rest, exclusively the outward and homeward voyages of the VOC are mentioned in the tables. Of those there were in total 4722 outward and 3359 homeward. The administration of the company was strictly followed, so that, for example, the voyage of Hudson in 1609 (0133) is listed, but not that of Roggeveen in 1721-1722. Voyages of East Indiamen that were driven off course, and arrived for instance in Surinam, or those which went no further than the Cape are mentioned, as opposed to those of warships of the five Admiralties which, from 1783, were sent to Asia to protect the fleets and possessions of the VOC.

The sources of the journeys consist primarily of the archives of the VOC in the Algemeen Rijksarchief in The Hague. They are, on the one hand, the so-called 'Uitloopboeken' and ship registers, and, on the other, the 'Overgekomen Brieven en Papieren' (OBP's). The latter contain the regular reports on the arrival and departure of ships in Batavia and other Asiatic harbors. In addition, the 'Overgekomen Brieven van de Kaap de Goede Hoop' and some other, more dispersed sources must be mentioned. The data on the voyages of the voorcompagnien derive above all from ources published by the Linschoten Vereeniging.

In volume I, the principal sources are described extensively and the origin of the information on each voyage is given. In addition, that volume contains an introduction on the organization of the VOC's shipping, which also includes an analysis and summary of the data presented in the tables. Various other supplementary information, such as the value of the export from the Netherlands, only available by year, is also published there. The tables follow closely the material presented in the major sources ('Uitloopboeken' en

OBP's). Since these sources are not uniform over a period of almost two centuries, the level of completeness of the information given for each voyage also varies.

Homeward voyage During the compilation of the tables it became necessary in a few cases to add an A to some numbers. This occurred 5 times, in the following places: 5022, 5980, 5987, 6246 and 6649. Similarly in three cases a number had to be left open. The following numbers have not been used: 4605, 5027 and 8215. The voyage number is followed by a figure which shows whether the ship is making its first, second or subsequent voyage. The outward and homeward voyages are counted separately. The first voyage from the Netherlands and the subsequent homeward voyages are both shown by a '1'. Occasionally a ship was built or acquired in Asia. The first outward voyage of such a ship is considered as its second voyage.

Ship's name A uniform spelling has been chosen for the numerous variants given in the sources. In alphabetical ordering and in the index, the most relevant word was chosen. Thus the WAPEN VAN, HOORN (0243) is given under HOORN, the HOF NIET ALTIJD ZOMER (2380) under ZOMER and the VROUWE REBECCA JACOBA (3668) under REBECCA. It should be noted that especially in the seventeenth century ships' names were frequently provided with additions which were not used in a consistent fashion. The AMSTERDAM (0431) was sometimes called NIEUWAMSTERDAM, the WITTE OLIFANT (0533), the OLIFANT. The most frequent name is given in the tables. In the eighteenth century, especially, ships' names were frequently changed, or they used each other's names. This is always mentioned under the heading Particulars and in the index.

Master's name Similarly, a uniform spelling has been chosen for the name of the master, generally schipper, but in the eighteenth century also a kapitein or kapiteinluitenant. The index is arranged by surname or patronymic.

Tonnage The volume of the ships is given in metric tons. The sources give the figures in lasten (1 last = 2 tons). After 1636, however, information in lasten is no longer of any value, as, for fiscal reasons, the VOC's figures were kept artificially low. Since then the volume has been calculated on the basis of the measurements of the ships, according to a simple formula (volume in lasten = length x breadth x depth in Amsterdam feet, divided by 200; 1 ft. = 28,3 cm). The results of this calculation have been doubled and are given in the tables. This method and the problems regarding the assessment of the ships' volume is described in Volume I. In a number of cases where inconsistent information was found, both calculations are given, thus e.g. 600/850.

Type of ship Occasionally, in those cases where this is mentioned in the sources, the type of the ship is given in the same column as the tonnage. In general, the most frequent type of ship, the retourschip (East Indiaman) is not mentioned in the sources. Therefore, where the type of ship is not mentioned, it may often be assumed that an East Indiaman is meant. The various other types - hoeker, kat, pinas, jacht, fluit, paketboot - are given in Dutch.

Built The year given in this column refers to the year in which the ship was built. If the ship was hired or bought by the VOC, then this is mentioned in the column, together with the year in which the transaction occurred.

Yard The place is given where the ship was built. The chambers of the VOC had their own yard. 'A' refers to Amsterdam, 'Z' to Zeeland, 'D' to Delft, 'R' to Rotterdam, 'H' to Hoorn, and 'E' to Enkhuizen. When a ship was hired or bought by the VOC, the letter indicates the chamber that was responsible for the transaction. The ships of the voorcompagnien did not belong to a chamber. In these cases, A' indicates that a ship was built at an Amsterdam yard. The chambers also had no part in the buying or building of ships by the Hoge Regering in Batavia. In these cases the place of building or purchase in Asia is given.

Chamber With the outward voyages, this column gives the chamber which equipped the ship; with the homeward, the chamber to which the ship was addressed. There is no entry in this column for ships organized by the voorcompagnien

Departure

Under this heading is given the date and place of departure from Europe, Asia or the Cape of Good Hope. A date like 03-02-1645 refers to 3 February 1645. Where sailings from the Republic are concerned, the date given refers to the departure from the roads. Amsterdam, Hoorn and Enkhuizen ships generally left from Texel roads, Zeeland ships from the Wielingen or the roads of Rammekens, and Rotterdam and Delft ships from Goeree. Sometimes, ships were forced by storms or damage to return to the roads for a time or they sought shelter in one of the estuaries on the coast of Holland or Zeeland. Where possible, this is mentioned under the heading Particulars. In general the first date of departure is given in the tables, but in some cases, a later date has been chosen, in deference to the sources.

As for leaving Batavia, departure from the roads of the town was decisive, and not, as is frequently described in the Company papers, the reaching of the 'open sea' after passing the Sunda Strait. Where departures from other Asian ports are concerned in general only the Company establishment from which the ship sailed is given. Thus Ceylon is mentioned in the columns, but it can be assumed that most ships left from the Bay of Galle, at the southern point of the island. China is given for ships which left from Canton, and the date refers to departure from the roads at Whampoa. Bengal is given for ships which left from the anchorage in the Ganges close to the VOC-establishment at Hughly.

Call at Cape The data in this column give the arrival at (above) and the departure (below) from the Cape of Good Hope. In general no distinction is made between Table Bay and False Bay. Mention is made, when given in the sources, of ships which put in to the more northerly Saldanha and St. Helena Bays. When a ship sailed past the Cape, this is denoted by 'no call'. When it is not known whether the ship stopped at the Cape at all - especially frequent before the foundation of the refreshment station there in 1652 - the column is left blank.

Arrival The third column contains the date and place of arrival in Asia, Europe or, when that was the destination of the journey, at the Cape of Good Hope. The place of arrival is given in the same manner as that of departure, though, in addition to the estuaries mentioned above, ships sometimes arrived in the Netherlands via the Vlie or at Delfzijl. The place of arrival in Asia refers to the establishment reached, unless the sources specify the actual port.

On board It is possible to differentiate the number of those on board into various categories. For the outward journey, these are seafarers, soldiers, craftsmen, and passengers. The craftsmen are those who were employed to perform some particular service in Asia, and are thus not part of the crew as such. 'Passengers' is in fact a residual category, including high officials of the Company, including ministers of religion with their wives and servants, but also slaves and stowaways. Whenever such a differentiation is not possible, which is

especially the case in the early years, a figure for the total is given. Italics are used for this, or when the figures refer to more than one category. Only those categories are mentioned which were on board. Therefore, when one category is mentioned, this implies that the others were not represented on board.

The sources for the return voyages are of a different kind and normally far less complete. They are totally absent for the journey between the Cape and the Netherlands. However, another category must be mentioned, namely the impotenten, who for various reasons were released from active service for the VOC and sent back to Europe. With regard to many voyages the sources only give the number of passengers and impotenten, and not the number of sailors and soldiers. Obviously, the absence of figures under these headings does not imply that there were none on board.

Information on the outward voyages is divided into six columns:

1. onboard_at_departure The number on board at departure
2. death_at_cape The number dying between the Netherlands and the Cape. Frequently this figure refers to all the categories together, even when the other information is available per category. In such cases this figure is printed in italics.
3. left_at_cape The number who leave the ship at the Cape.
4. onboard_at_cape The number who come on board at the Cape.
5. death_during_voyage The number dying on the whole voyage. Subtraction of III from V gives the number dying between the Cape and Asia.
6. onboard_at_arrival The number on board on arrival in Asia.

Three columns are given for the homeward voyages:

1. onboard_at_departure The number on board at departure
2. death_at_cape The number dying en route to the Cape
3. left_at_cape The number who went from board at the Cape

The figures in the various columns are taken from different sources which are not always consistent with each other. Therefore the figures on changes in the number of those on board during the voyages do not always tally with those on the size of the crew at departure and arrival.

Invoice value For the return voyages, the total value of the ship's cargo, according to the invoice made up in Batavia or some other establishment, is given, as is the chamber for which the cargo was destined. Generally, this was for the chamber under whose jurisdiction the ship sailed, but occasionally a proportion of the cargo was for one or more of the other chambers.

Particulars Under the last heading details deriving from the basic sources are given. They are generally incidental and as such not to be placed in one of the preceding columns. Because the sources are not the same across the whole period, and at times less complete, the extent and sort of material under this heading could not be consistent.

In so far as it is available, information deals with the ports of call on the journey, with the details of changes in the composition of the crew and with the eventual fate of the ship. For the return voyage, the name of the fleet-commander is generally given, and, after his name, the number of the ship he was on. Finally, where necessary, differences in data between various sources are indicated. Occasionally, particulars from a published source are added.

Corresponding number This number, placed at the far right of the tables, denotes the next homeward voyage of the ship in volume II (naturally absent when the ship remains in Asia), or, in volume III, for homeward voyages, the number of the ship's previous outward voyage. In those cases where the ship was acquired in Asia, no corresponding number is given for the first homeward voyage from Asia.

Due to the long duration of the preparation of these two volumes there are some inconsistencies in the text of the particulars and in the use of language.