Contents

1 MONETDB Internals **1**
1.1 Redesign considerations. . . . 1
1.2 Storage Model 2
1.3 All (relational) operators exploit a small set of properties: 2
1.4 Execution Model 2
1.5 Software Stack 2

2 Binary Association Tables **2**

3 MAL Reference (MonetDB Assembly Language) **2**
3.1 Literals (follow the lexical conventions of C) 3
3.2 Variables 3
3.3 Instructions 3
3.4 Type System 3
3.5 Flow of Control 4
3.6 Exceptions 4
3.7 Functions 4
3.8 MAL Syntax 5
3.9 MAL Interpreter 5
3.10 MAL Debugger 5
3.11 MAL Profiler 5
3.12 MAL Optimizers 6
3.13 MAL Modules 7

4 MAL Algebra **8**

1 MONETDB Internals

`http://sites.computer.org/debull/A12mar/monetdb.pdf` MonetDB Internals Source Compile

DOCUMENTATION -> monetdb source/lib/monetdb5/algebra.mal

1.1 Redesign considerations.

Redesign of the MonetDB software driven by the need to reduce the effort to extend the system into novel directions and to reduce the **Total Execution Cost (TEC)**.

TEC:

- API message handling (**A**)
- Parsing and semantic analysis (**P**)
- Optimization and plan generation (**O**)
- Data access to the persistent store (**D**)
- Execution of the query terms (**E**)
- Result delivery to the application (**R**)

OLTP -> Online Transaction Processing -> expected most of the cost to be in (P,O) OLAP -> Online Analytical Processing -> expected most of the cost to be in (D,E,R)

1.2 Storage Model

- Represents relational tables using vertical fragmentation.
- Stores each column in a separate {(OID,value)} table, called a **BAT (Binary Association Table)**
- Relies on a low-level relational algebra called the BAT algebra, which takes BATs and scalar values as input.
- The complete result is always stored in (intermediate) BATs, and the result of an SQL query is a collection of BATs.
- **BAT** is implemented as an ordinary C-array. OID maps to the index in the array.
- Persistent version of **BAT** is a **memory mapped file**.
- **O(1) positional database lookup mechanism** (MMU - memory management unit)

1.3 All (relational) operators exploit a small set of properties:

- seq - the sequence base, a mapping from array index 0 into a OID value
- key - the values in the column are unique
- nil - there is at least one NIL value
- nonil - it is unknown if there NIL values
- dense - the numeric values in the column form a dense sequence
- sorted - the column contains a sorted list for ordered domains
- revsorted - the column contains a reversed sorted list

1.4 Execution Model

- **MonetDB** kernel is an abstract machine, programmed in the **MonetDB Assemblee Language (MAL)**.
- Each relational algebra operator corresponds to a **MAL instruction** (zero degrees of freedom).
- Each **BAT algebra operator** maps to a simple **MAL instruction**.

1.5 Software Stack

Three software layers:

- **FRONT-END Query language parser and a heuristic, language - and data model - specific optimizer**. **OUTPUT** -> logical plan expressed in MAL.
- **BACK-END Collection of optimizer modules** -> assembled into an optimization pipeline
- **MAL interpreter** -> contains the library of highly optimized implementation of the binary relational algebra operators.

2 Binary Association Tables

3 MAL Reference (MonetDB Assembly Language)

- MAL program is considered a specification of intended computation and data flow behavior.
- Language syntax uses a functional style definition of actions and mark those that affect the flow explicitly.

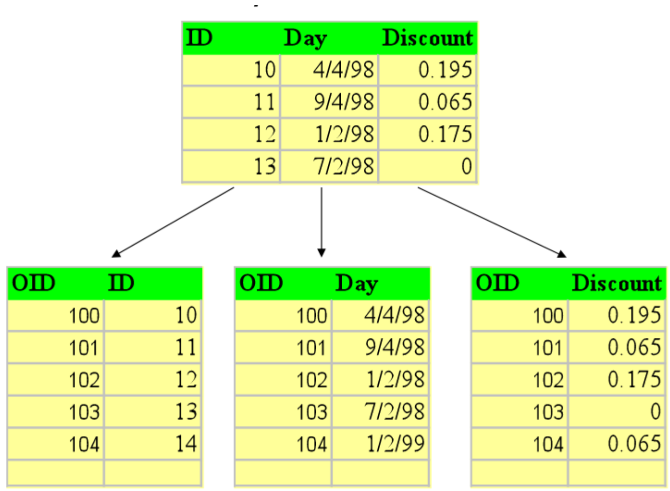

Figure 1: Bat Sample

3.1 Literals (follow the lexical conventions of C)

Hardwire Types	Temporal Types	IPv4 addresses and URLs
bit (bit)	date	inet
bte (byte)	daytime	url
chr (char)	time	UUID
wrd (word)	timestamp	json
sht (short)	-	-
int (integer)	-	-
lng (long)	-	-
oid (object id)	-	-
flt (float)	-	-
dbl (double)	-	-
str (string)	-	-

3.2 Variables

User Defined -> start with a letter **Temporary** -> start with X_ (generated internally by optimizers)

3.3 Instructions

One liners -> easy to parse

```
(t1,..,t32) := module.fcn(a1,..,a32);
t1 := module.fcn(a1,..,a32);
t1 := v1 operator v2;
t1 := literal;
(t1,..,tn) := (a1,..,an);
```

Figure 2: Instructions example

3.4 Type System

Strongly typed language

- Polymorphic given by "any".
- Type checker (intelligent type resolution).

```
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;
```

Figure 3: Polymophism example

3.5 Flow of Control

For statement implementation:

```
     i:= 1;
barrier B:= i<10;
     io.print(i);
     i:= i+1;
redo B:= i<10;
exit B;
```

Figure 4: For example

If statement implementation:

```
    i:=1;
barrier ifpart:= i<1;
    io.print("ok");
exit ifpart;
barrier elsepart:= i>=1;
    io.print("wrong");
exit elsepart;
```

Figure 5: If example

3.6 Exceptions

(**To explore.**)

3.7 Functions

Function example

Side Effects

- Functions can be pre-pended with the keyword unsafe.
- Designates that execution of the function may change the state of the database or sends information to the client.
- Unsafe functions are critical for the optimizers -> order of execution should be guaranteed.
- Functions that return **:void** -> unsafe by default.

```
function user.helloWorld(msg:str):str;
    io.print(msg);
    msg:= "done";
    return msg;
end user.helloWorld;
```

Figure 6: Function example

Inline Functions

- Functions prepended with the keyword **inline** are a target for the optimizers to be inlined. -> reduce the function call overhead.

3.8 MAL Syntax

Expressed in extended Backus–Naur form (EBNF) Wiki

Alternative constructors	(vertical bar) grouped by ()
Repetition	'+'-> at least once; '*'-> many
Lexical tokens	small capitals

```
program         : (statement ';') *
statement       : moduleStmt [helpinfo] | definition [helpinfo]
                | includeStmt | stmt
moduleStmt      : MODULE ident | ATOM ident [':'ident]
includeStmt     : INCLUDE identifier | INCLUDE string_literal
definition      : [UNSAFE] COMMAND header ADDRESS identifier
                | [UNSAFE] PATTERN header ADDRESS identifier
                | [INLINE | UNSAFE ] FUNCTION header statement* END name
                | FACTORY header statement* END name
helpinfo        : COMMENT string_literal
header          : name '(' params ')' result
name            : [ moduleName '.' ] name
result          : typeName | '(' params ')'
params          : binding [',' binding]*
binding         : identifier typeName
typeName        : scalarType | columnType | ':' any ['_' digit]
scalarType      : ':' identifier
columnType      : ':' BAT '[' ':' colElmType ']'
colElmType      : scalarType | anyType
stmt            : [flow] varlist [':=' expr ]
flow            | RETURN  | BARRIER | CATCH | LEAVE | REDO | RAISE
varlist         : variable | '(' variable [',' variable ] * ')'
variable        : identifier
expr            : fcncall | [factor operator] factor
factor          : literal_constant | NIL | var
fcncall         : moduleName '.' name '(' [args] ')'
args            : factor [',' factor]*
```

Figure 7: Syntax example

3.9 MAL Interpreter

(**To explore.**)

3.10 MAL Debugger

(**To explore.**)

3.11 MAL Profiler

The program stethoscope is a simple Linux application that can attach itself to a running MonetDB server and extracts the profiler events from concurrent running queries. Stethoscope is an online-only inspection tool, i.e., it only monitors the execution state of the current queries and outputs the information in STDOUT for immediate inspection. For example, the following command tracks the microsecond ticks for all database instructions denoted in MAL on a database called "voc":

$ stethoscope -u monetdb -P monetdb -d voc

Discontinued:

- Tachograph

- Tomograph
-

3.12 MAL Optimizers

Triggered by experimentation and curiousity

- Alias Removal
- Building Blocks -> there are examples for a user to build a Optimizer
- Coercions Removes coercions that are not needed –> v:= calc.int(23); (sloppy code-generator or function call resolution decision)
- Common Subexpressions

```
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();
```

Figure 8: Syntax example

```
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);
```

Figure 9: Syntax example 2

- Constant Expression Evaluation

```
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();
```

Figure 10: Expression example

- Cost Model
- Data Flow Query executions without side effects can be rearranged.
- Garbage Collector
- Join Paths Looks up the MAL query and "composes" multiple joins. **algebra.join -> algebra.joinPath**

```
io.print(2);
io.print(2);
io.print(6);
io.print(6);
io.print(-0.279415488);
```

Figure 11: Expression example 2

- Landscape
- Lifespans
- Macro Processing
- Memoization
- Multiplex Functions
- Remove Actions
- Stack Reduction

3.13 MAL Modules

- Alarm
- Algebra (Important)
- BAT (Important)
- BAT Extensions (Important)
- BBP
- Calculator
- Clients (Important)
- Debugger (Important)
- Factories
- Groups (Important)
- I/O
- Imprints
- Inspect
- Iterators
- Language Extension
- Logger
- MAPI Interface (Important)
- Manual
- PCRE Library

- Profiler
- Remote
- Transaction

4 MAL Algebra

Operation	MAL Cmd	C Cmd	Arguments/Return	Comment
GroupBy	groupby	ALGgroupby	gids :: bat-columntype:oid cnts :: bat-columntype:oid return :: bat-columntype:oid	Produces a new BAT with groups indentified by the head column. (The result contains tail times the head value, ie the tail contains the result group sizes.)
Find	find	ALGfind	b :: bat-columntype:any-1 t :: any-1 return :: oid	Returns the index position of a value. If no such BUN exists return OID-nil.
Fetch	fetch	ALGfetchoid	b :: bat-columntype:any-1 x :: oid return :: any-1	Returns the value of the BUN at x-th position with 0 <= x < b.count
Project	project	ALGprojecttail	b :: bat-columntype:any-1 v :: any-3 return :: bat-columntype:any-3	Fill the tail with a constant
Projection	projection	ALGprojection	left :: bat-columntype:oid rigth :: bat-columntype:any-3 return :: bat-columntype:any-3	Project left input onto right input.
Projection2	projection2	ALGprojection2	left :: bat-columntype:oid rigth1 :: bat-columntype:any-3 rigth2 :: bat-columntype:any-3 return :: bat-columntype:any-3	Project left input onto right inputs which should be consecutive.

BAT copying

Operation	MAL Cmd	C Cmd	Arguments/Return	Comment
Copy	copy	ALGcopy	b :: bat-columntype:any-1 return :: bat-columntype:any-1	Returns physical copy of a BAT.
Exist	exist	ALGexist	b :: bat-columntype:any-1 return :: bit	Returns whether 'val' occurs in b.

select ALGselect1 ALGselect2 ALGselect1nil ALGselect2nil
thetaselect ALGthetaselect1 ALGthetaselect2
selectNotNil ALGselectNotNil
sort ALGsort11 ALGsort12 ALGsort13 ALGsort21 ALGsort22 ALGsort23 ALGsort31 ALGsort32 ALGsort33
unique ALGunique2 ALGunique1
Join operations crossproduct ALGcrossproduct2
join ALGjoin ALGjoin1

leftjoing ALGleftjoin ALGleftjoin1
outerjoin ALGouterjoin
semijoin ALGsemijoin
thetajoin ALGthetajoin
band join ALGbandjoin
rangejoin ALGrangejoin
difference ALGdifference
intersect ALGintersect
Projection operations firstn ALGfirstn **reuse** ALGreuse
slice $\text{ALGslice}_{\text{oid}}$ ALGslice $\text{ALGslice}_{\text{int}}$ $\text{ALGslice}_{\text{lng}}$ **subslice** $\text{ALGsubslice}_{\text{lng}}$
Common BAT Aggregates
count $\text{ALGcount}_{\text{bat}}$ $\text{ALGcount}_{\text{nil}}$ $\textbf{count}_{\textbf{nonil}}$ $\text{ALGcount}_{\text{nonil}}$
count $\text{ALGcountCND}_{\text{bat}}$ $\text{ALGcountCND}_{\text{nil}}$ $\textbf{count}_{\textbf{nonil}}$ $\text{ALGcountCND}_{\text{nonil}}$
Default Min and Max
cardinality ALGcard **min** ALGminany $\text{ALGminany}_{\text{skipnil}}$ **max** ALGmaxany $\text{ALGmaxany}_{\text{skipnil}}$
PATTERN **avg** CMDcalcavg
Standard deviation
stdeb ALGstdev **stdevp** ALGstdevp **variance** ALGvariance **variancep** ALGvariancep **covariance** ALGcovariance **covariancep** ALGcovariancep **corr** ALGcorr