Contents

1 MONETDB Internals

`http://sites.computer.org/debull/A12mar/monetdb.pdf` MonetDB Internals Source Compile Source Documentation -> $\text{monetdb}_{\text{source}}$/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

Key-Value Pair Model

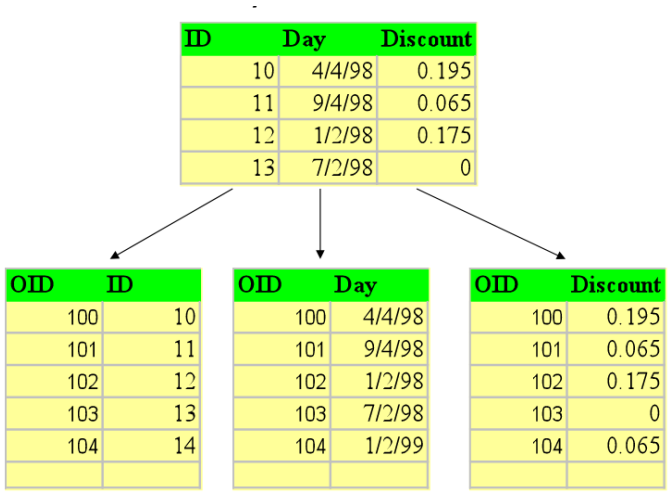

Figure 1: Bat Sample

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.

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

```
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

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

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:

3.6 Exceptions

(**To explore.**)

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

Figure 5: If example

3.7 Functions

Function example

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

Figure 6: 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.

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

3.9 MAL Interpreter

(**To explore.**)

3.10 MAL Debugger

(**To explore.**)

```
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

3.11 MAL Profiler

Stethoscope REPLACED WITH Pystethoscope

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
Check source
Cost Model, Alias Removal, Landscape, Lifespans, Macro Processing, Memoization, Multiplex Functions, Remove Actions, Stack Reduction, Garbage Collector

3.12.1 Building Blocks

There are examples for a user to build a Optimizer

3.12.2 Coercions

Removes coercions that are not needed –> v:= calc.int(23); (sloppy code-generator or function call resolution decision)

3.12.3 Common Subexpressions

3.12.4 Constant Expression Evaluation

3.12.5 Data Flow

Query executions without side effects can be rearranged.

3.12.6 Join Paths

Looks up the MAL query and "composes" multiple joins. **algebra.join -> algebra.joinPath**

```
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

```
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

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

Figure 11: Expression example 2

3.13 MAL Modules Overview

- 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

4.1 Misc

MAL	Address	Comment
groupby	ALGgroupby	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	ALGfind	Returns the index position of a value. If no such BUN exists return OID-nil.
fetch	ALGfetchoid	Returns the value of the BUN at x-th position with 0 <= x < b.count
project	ALGprojecttail	Fill the tail with a constant
projection	ALGprojection	Project left input onto right input.
projection2	ALGprojection2	Project left input onto right inputs which should be consecutive.

4.2 BAT copying

MAL	Address	Comment
copy	ALGcopy	Returns physical copy of a BAT.
exist	ALGexist	Returns whether 'val' occurs in b.

4.3 Selecting

The range selections are targeted at the tail of the BAT.

MAL	Address	Comment
select	ALGselect1	Select all head values for which the tail value is in range. Input is a dense-headed BAT, output is a dense-headed BAT with in the tail the head value of the input BAT for which the tail value is between the values low and high (inclusive if li respectively hi is set). The output BAT is sorted on the tail value.
select	ALGselect2	Select all head values of the first input BAT for which the tail value is in range and for which the head value occurs in the tail of the second input BAT. The first input is a dense-headed BAT, the second input is a dense-headed BAT with sorted tail, output is a dense-headed BAT with in the tail the head value of the input BAT for which the tail value is between the values low and high (inclusive if li respectively hi is set). The output BAT is sorted on the tail value.
select	ALGselect1nil	With unknown set, each nil != nil
select	ALGselect2nil	With unknown set, each nil != nil
selectNotNil	ALGselectNotNil	Select all not-nil values.
thetaselect	ALGthetaselect1	Select all head values for which the tail value obeys the relation value OP VAL. Input is a dense-headed BAT, output is a dense-headed BAT with in the tail the head value of the input BAT for which the relationship holds. The output BAT is sorted on the tail value.

Continued on next page

Continued from previous page

MAL	Address	Comment
thetaselect	ALGthetaselect2	Select all head values of the first input BAT for which the tail value obeys the relation value OP VAL and for which the head value occurs in the tail of the second input BAT. Input is a dense-headed BAT, output is a dense-headed BAT with in the tail the head value of the input BAT for which the relationship holds. The output BAT is sorted on the tail value.

4.4 Sort

MAL	Address	Comment
sort	ALGsort11	Returns a copy of the BAT sorted on tail values. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.
sort	ALGsort12	Returns a copy of the BAT sorted on tail values and a BAT that specifies how the input was reordered. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.
sort	ALGsort13	Returns a copy of the BAT sorted on tail values, a BAT that specifies how the input was reordered, and a BAT with group information. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.
sort	ALGsort21	Returns a copy of the BAT sorted on tail values. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.
sort	ALGsort22	Returns a copy of the BAT sorted on tail values and a BAT that specifies how the input was reordered. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.
sort	ALGsort23	Returns a copy of the BAT sorted on tail values, a BAT that specifies how the input was reordered, and a BAT with group information. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.
sort	ALGsort31	Returns a copy of the BAT sorted on tail values. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.
sort	ALGsort32	Returns a copy of the BAT sorted on tail values and a BAT that specifies how the input was reordered. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.
sort	ALGsort33	Returns a copy of the BAT sorted on tail values, a BAT that specifies how the input was reordered, and a BAT with group information. The order is descending if the reverse bit is set. This is a stable sort if the stable bit is set.

4.5 Unique

MAL	Address	Comment
unique	ALGunique2	Select all unique values from the tail of the first input. Input is a dense-headed BAT, the second input is a dense-headed BAT with sorted tail, output is a dense-headed BAT with in the tail the head value of the input BAT that was selected. The output BAT is sorted on the tail value. The second input BAT is a list of candidates.
unique	ALGunique1	Select all unique values from the tail of the input. Input is a dense-headed BAT, output is a dense-headed BAT with in the tail the head value of the input BAT that was selected. The output BAT is sorted on the tail value.

4.6 Join operations

4.6.1 Crossproduct

MAL	Address	Comment
crossproduct	ALGcrossproduct2	Returns 2 columns with all BUNs, consisting of the head-oids from 'left' and 'right' for which there are BUNs in 'left' and 'right' with equal tails

4.6.2 Joining

MAL	Address	Comment
join	ALGjoin	Join
join	ALGjoin1	Join; only produce left output
leftjoin	ALGleftjoin	Left join with candidate lists
leftjoin	ALGleftjoin1	Left join with candidate lists; only produce left output
outerjoin	ALGouterjoin	Left outer join with candidate lists
semijoin	ALGsemijoin	Semi join with candidate lists
thetajoin	ALGthetajoin	Theta join with candidate lists
bandjoin	ALGbandjoin	Band join: values in l and r match if r - c1 <[=] l <[=] r + c2
rangejoin	ALGrangejoin	Range join: values in l and r1/r2 match if r1 <[=] l <[=] r2
difference	ALGdifference	Difference of l and r with candidate lists
intersect	ALGintersect	Intersection of l and r with candidate lists (i.e. half of semi-join)

4.7 Projection operations

MAL	Address	Comment
firstn	ALGfirstn	Calculate first N values of B
reuse	ALGreuse	Reuse a temporary BAT if you can. Otherwise, allocate enough storage to accept result of an operation (not involving the heap)
slice	ALGslice\oid	Return the slice based on head oid x till y (exclusive).
slice	ALGslice	Return the slice with the BUNs at position x till y
slice	ALGslice\int	Return the slice with the BUNs at position x till y
slice	ALGslice\lng	Return the slice with the BUNs at position x till y
subslice	ALGsubslice\lng	Return the oids of the slice with the BUNs at position x till y

4.8 Common BAT Aggregates

These operations examine a BAT, and compute some simple aggregate result over it.

MAL	Address	Comment
count	ALGcount$\backslash_{\text{bat}}$	Return the current size (in number of elements) in a BAT.
count	ALGcount$\backslash_{\text{nil}}$	Return the number of elements currently in a BAT ignores BUNs with nil-tail iff ignore$_{\text{nils}}$==TRUE.
count	ALGcountCND$\backslash_{\text{bat}}$	Return the current size (in number of elements) in a BAT.
count	ALGcountCND$\backslash_{\text{nil}}$	Return the number of elements currently in a BAT ignores BUNs with nil-tail iff ignore$_{\text{nils}}$==TRUE.
count$_{\text{nonil}}$	ALGcount$_{\text{no}}\backslash_{\text{nil}}$	Return the number of elements currently in a BAT ignoring BUNs with nil-tail
count$_{\text{nonil}}$	ALGcountCND$\backslash_{\text{no}}\backslash_{\text{nil}}$	Return the number of elements currently in a BAT ignoring BUNs with nil-tail

4.9 Default Min and Max

Implementations a generic Min and Max routines get declared first. The @emph{min()} and @emph{max()} routines below catch any tail-type. The type-specific routines defined later are faster, and will override these any implementations.

cardinality - ALGcard **min** - ALGminany, ALGminany$_{\text{skipnil}}$ **max** - ALGmaxany, ALGmaxany$_{\text{skipnil}}$ PATTERN **avg** - CMDcalcavg

4.10 Standard deviation

The standard deviation of a set is the square root of its variance. The variance is the sum of squares of the deviation of each value in the set from the mean (average) value, divided by the population of the set.

stdeb - ALGstdev **stdevp** - ALGstdevp **variance** - ALGvariance **variancep** - ALGvariancep **covariance** - ALGcovariance **covariancep** - ALGcovariancep **corr** - ALGcorr