Department of Informatics, University of Zürich

MSc Basic Module

Implementing Self Addition Inside MonetDB

Timo Surbeck

Matrikelnummer: 15-701-733

Email: timo.surbeck@uzh.ch

June 23, 2019

supervised by Prof. Dr. Michael Böhlen and Oksana Dolmatova

University of Zurich UZH

Department of Informatics

D B T G

1 Introduction

1.1 MonetDB

MonetDB is a widely known open source database management system (DBMS) developed and maintained since 1993 by a team at the Centrum Wiskunde & Informatica (CWI) in the Netherlands. When it comes to DBMS featuring columnar storage (column stores), MonetDB played a pioneering role: In column stores, relations are stored and processed in terms of sets of attribute lists (relations' columns), which is clearly distinct from the tuple-wise approach implemented in traditional DBMS. For a variety of applications (*i.e.*, queries), *e.g.*, joins, column stores are exceptionally efficient, as internally it is possible to directly iterate through columns. Being written in the C programming language, on the low level, MonetDB uses the Binary Association Table (BAT) data structure to store relations' columns. Next to the actual data held in traditional C-arrays, BATs store a unique identifier (OID) for each value, such that for recovering multi-attribute relations' tuples, values residing in several BATs can be joined according to their OID [2].

Working with MonetDB involves running two separate programs: At the frontend application (`mclient`), a user can enter queries in the SQL language, or, load SQL-scripts. Entered instructions are transferred to the backend (`mserver5`), where a series of passing several layers from high to low is started off: Initially, an SQL query is parsed into a symbol tree, which – itself – is not yet related to the internals of a DBMS. However, this symbol tree is used to construct MonetDB's equivalent of a query tree, namely the relation tree. At this point, an optimizer rearranges the relation tree using heuristics and optimization rules. The efficiency-optimized relation tree is then converted into the attribute-oriented statement tree. This tree allows for defining operations applied directly on columns. Finally, the statement tree is translated to a MAL-plan; MAL stands for MonetDB Assembly Language and handles BAT-level operations [1], [2].

1.2 Problem Definition

The main purpose of this module is the preparation for a MSc project, for which I will collaborate with my fellow students Alphonse Mariyagnanaseelan and Jonathan Stahl on the implementation of *Matrix Operations with Gathering in MonetDB*. Therefor, the task of this MSc basic module was created, namely the implementation of a self addition feature for numeric single-attribute relations. This task presented itself as an ideal preparation for the upcoming project, as – except for optimization of the relation tree and modifying the MAL – every part of MonetDB's internals had to be explored and extended.

The following illustration shows self addition on the exemplary single-attribute relation `r` producing result relation `r'`:

r

A
3
9
2

r’

A
6
18
4

2 Design & Implementation

To address the implementation’s general approach, it can be said that the sequence of a query execution was directly followed, *i.e.*, proceeding from higher to lower architecture levels. Therefore, as a first step, the SQL parser was extended, such that the following self addition query became understandable for the backend (`r` is a numeric single-attribute relation):

```
SELECT * FROM ADD r;
```

It should be mentioned, that during development, MonetDB’s cross join functionality was used to gradually check advancements in all of MonetDB’s layers, *e.g.*, it was invoked from within the extended query parser as a means to verify, if the current layer extension was implemented correctly – even if self addition was not yet realized on lower levels. Furthermore, during the development, `gdb` (GNU debugger) revealed itself very useful to interim examine fields pointing at complex data structures (*e.g.* the symbol tree).

The source code containing the finished self addition feature implemented in MonetDB 11.23.13 can be obtained at: https://github.com/timolex/monetdbBasicModule

2.1 Extension Of The SQL Parser

MonetDB uses Yacc (Yet Another Compiler Compiler) as a tool to parse SQL queries [1]. Its configuration resides in file `sql_parser.y` and was extended as follows:

```
|   ADD table_ref
        { dlist *l = L();
        append_symbol(l, $2);
        $$ = _symbol_create_list( SQL_ADD, l); }
```

As apparent in the self addition SQL query, the ADD-keyword appears in the FROM-clause. Therefore, the new expression was configured in the same area as different FROM-clause expressions, *e.g.,* cross join. $2 refers to the table name of the single-attribute relation. In an early stage version, $2 was appended twice to the dlist and the SQL_CROSS token was invoked, such that by running the query, self cross join was executed.

2.2 Adding A Self Addition Relation Tree Node

The symbol tree created by the parser only contains symbols (*e.g.* relation names) and therefore might also represent a query aiming to access inexistent relations, *resp.*, attributes. It is – however – the implementation of the relation tree which considers the actual DB to verify the existence of relation names present in the query. The below illustration depicts the relation tree for self addition which is generated by the added code. $\oplus$ denotes the self addition operation:

$$\begin{tikzcd}
\pi_* \arrow[d, dash] \\
\oplus \arrow[d, dash] \\
\mathtt{r}
\end{tikzcd}$$

The following code snippet developed in file rel_select.c exhibits the translation from a symbol tree node (dnode) to a relation tree's child (*i.e.*, a relation, sql_rel):

```
dnode *n        =   q->data.lval->h;
symbol *tab1    =   n->data.sym;
sql_rel *t1     =   table_ref(sql, rel, tab1);
```

Additionally, in file rel_rel.c a function called rel_addition was implemented which returns a relation tree node holding static information about self addition, *e.g.*, its single child (relation), the no. of returned columns *etc.*

2.3 Translation: Relation Tree Node To Statement Tree

After defining a new relation tree node, the statement tree was extended to cover the self addition operation. While the relation tree holds relations as well as operators (representing the operation to be applied on child nodes), the statement tree entirely consists of statement nodes (data type stmt). Statement tree nodes are either child nodes, *i.e.*, columns of a relation, or representing tree nodes applying an operation on one or several child nodes. The translation from relation tree node to statement tree takes place in file rel_bin.c, where function rel2bin_addition takes a relation tree node as input parameter and translates it to a statement tree accordingly, as depicted in the following code snippet (left and addition are stmt-pointers):

```
left      =  subrel_bin(sql, rel->l, refs);
node *n   =  left->op4.lval->h;
stmt *c   =  n->data;
stmt *l   =  column(sql->sa, c);
addition  =  stmt_addition(sql->sa, l);
```

Field `addition` points at a statement tree node which is returned by `stmt_addition` defined in `sql_statement.c`; This function calls `stmt_create`, which in MonetDB is used to create new statement tree nodes.

2.4 Translation: Self Addition Statement To MAL plan

In file `sql_gencode.c`, function `_dumpstmt` serves the purpose of translating the statement tree into a MAL plan invoking BAT operations. This is accomplished by adding a switch-case for the statement type `st_addition` which is associated to the self addition statement tree node defined in `sql_statement.c`. In essence, the following code was necessary to let the MAL code be generated allowing to apply element-wise addition of two BATs:

```
q = newStmt(mb, batcalcRef, "+");
q = pushArgument(mb, q, l);
q = pushArgument(mb, q, l);
```

While with the first line of the above code snippet, the BAT addition function is set up, at the latter two lines, the same argument (`t`, *i.e.*, the relation's single attribute) is pushed twice to the addition function to attain self addition. Internally, this BAT function is called `batcalc.+` and is also invoked for attribute additions using the usual SQL syntax. This became clear when prepending `EXPLAIN` to the following query:

```
SELECT A + A FROM r;
```

3 Conclusion & Discussion

Overall, self addition for single-attribute relations can be described as a very simple functionality, not only in terms of the executed operation but also the underlying implementation; However, following cross join as a guideline revealed the flexibility and high extendability of all of MonetDB's backend layers. In terms of scalability, it can be stated, that with – contextually – moderately high effort, it would be possible to extend this implementation to attain the functionality of addition of two single-attribute relations: One of the necessary steps therefor would be to add a second child node to the relation tree representing the second relation.

Furthermore, with very little effort it is possible to change this implementation to realize different simple operations, such as self multiplication, division, modulo, *etc.*, as the corresponding BAT operations are present and ready to use.

Bibliography

[1] Alphonse Mariyagnanaseelan, *Optimization of Mixed Queries in MonetDB System*, `https://www.merlin.uzh.ch/contributionDocument/download/11242`, accessed 2019-06-19, Sep. 2018.

[2] Database Architectures Research Group (CWI), *MonetDB*, `https://www.monetdb.org`, accessed 2019-06-18.