Informatik-Vertiefung Report
Datenbanktechnologie

Table of Contents

1. Task

The task of this project is to install the database management system „MonetDB“ and study the execution level of it in a first step. The execution level includes the creation of so-called MAL plans and the execution of queries using MAL instructions. In a next step, the LAPACK library should be included into the compilation process of MonetDB. An external function should be included into MonetDB that calls the LAPACK library for the internal MonetDB function of vector addition.
The data from the database should be copied in this process and the executed query should have the form „SELECT x + y from t;“, where x and y are numeric values in the table t. Afterwards, the run times of the data being processed solely by MonetDB and the data being processed with a partial external execution (vector operation is delegated to LAPACK) should be compared and repeated with different number of rows (1000, 10000, 1000000).
There is an optional part of the task where the data of the database should not be copied to LAPACK but rather just passed on to the LAPACK function. Additionally, run times should be compared with this optimized function where the values are passed on to LAPACK.

2. MonetDB

For this task, the column-oriented database management system „MonetDB“ is used. MonetDB is an open source system that is primarily used to perform complex queries on large databases. Its advantage over other DBMS is the high performance, that allows processing millions of rows at an incomparable speed for applications such as data mining, online analytical processing and other fields that deal with an enormous amount of data.
What separates MonetDB from traditional DBMS is the architecture. There are three layers which process date with their own set of optimizers. The top layer is the front-end, which provides an interface for database languages like SQL. Incoming queries are parsed into specific representations like relational algebra. Afterwards, there is a translation into the MonetDB Assembly Language (MAL) that is passed to the back-end layer, which optimizes based on costs. From there, the database kernel acts as the bottom layer, providing access to the data in a special storage model. MonetDB uses vertical fragmentation, which stores every column in a so-called Binary Association Table (BAT). This table contains Binary Units (BUN) which themselves contain a head and a tail value. The head value is an OID, that stores a unique ID for the tail value. The tail value is basically the value that was inserted into the column.
For a table with the columns x and y there are two BAT’s. If the values 10 and 25 are inserted into column x, BAT x will have two BUN’s which are [OID: 1, VAL: 10] and [OID: 2, VAL: 25]. The OID is the same for all values in one tuple, this means that if the values 15 and 35 are inserted into column y, BAT y will have two BUN’s with following OID’s:
[OID: 1, VAL: 15] and [OID: 2, VAL: 35].
This architecture generally allows for a fast lookup when it comes to queries that use statements like JOIN or similar.

BAT x	
OID	VAL
1	10
2	25

BAT y	
OID	VAL
1	15
2	35

Two BAT’s filled with values

3. Structure of MonetDB-trees

As in many DBMS, one of the reasons why MonetDB performs so well is the breakdown of incoming queries into different tree-structures. The resulting structure is optimized and ensures this way a performance improvement with every breakdown-step. The first step includes the breakdown of the query into a relation tree. In this tree, all parent nodes are statements and all child nodes are tables. Optimizing can be achieved for example by changing the order of the statements.
In the next step, a statement tree is created where the structure is broken down to BAT’s and operations. This structure is the base for the conversion into the MAL-plan where the programmatic steps for the query-execution get put together. The MAL-plan gets executed and the result get computed.

4. Example: JOIN-Operation

This example demonstrates the processing of MonetDB by executing a query containing a JOIN-Operation and a condition that applies to it. Assume two relations table1 and table2 with the schema a and b for table1 and x and y for table2.

The columns are filled with the following values:

(10, 25, 40) for a,
(20, 50, 85) for b,
(15, 20, 25) for x,
(30, 50, 60) for y.

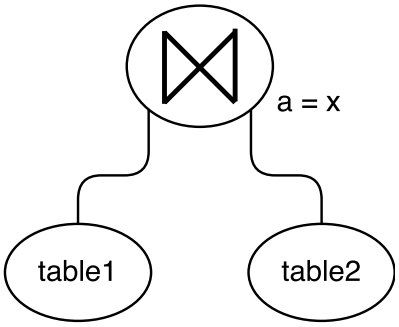
table1	
a	b
10	20
25	50
40	85

table2	
x	y
15	30
20	50
25	60

Schema for table1 and table2

The query has the following form:
SELECT * FROM table1 JOIN table2 ON a = x;

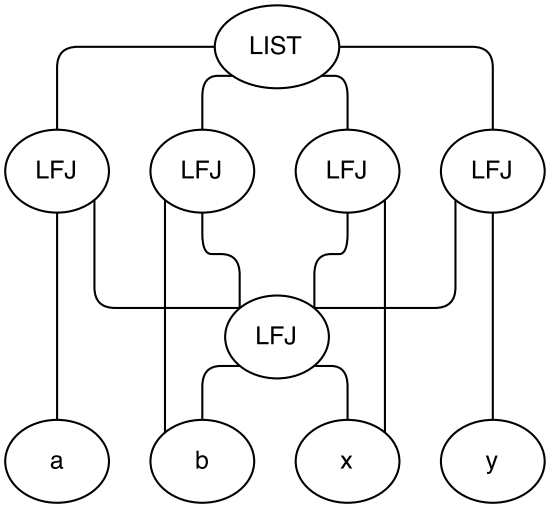
Before the query is executed, MonetDB transforms the query into a relation tree. The nodes of such a tree can either be relations or relational algebra operators. All of the relations involved are leaf-nodes in this tree. The other nodes are operators, they have child-nodes that can be relations or other operators. In this example, there is a parent-node that contains the JOIN-Statement including the condition „a = x“. It has two children, table1 and table2, which are the leaves in this tree.



Relation tree

After the creation of the relation tree, the structure is further broken down into a statement tree. The leaves of the statement tree are BAT's from the columns of all the tables that are involved in the JOIN-Operation. The BAT's that represent the columns which are mentioned in the condition (a and x in this example) are left-fetch joined and deliver the OID's from the BAT's which are „stored“ in a parent-node. For this query, „Bat a“ and „Bat x“ compare their values through iteration and find OID 2 for Bat a and OID 3 for Bat x, because both BAT's store the value 25 at this position. The resulting node plays a crucial part in the next step of the building of the statement tree. Every leaf-node now performs a left-fetch join with the node that was created in the previous step. For every BAT, the corresponding OID from the same table gets selected and put out as a result. In this example, OID 2 (from x) is used for BAT's from table1 and OID 3 (from a) is used for BAT's

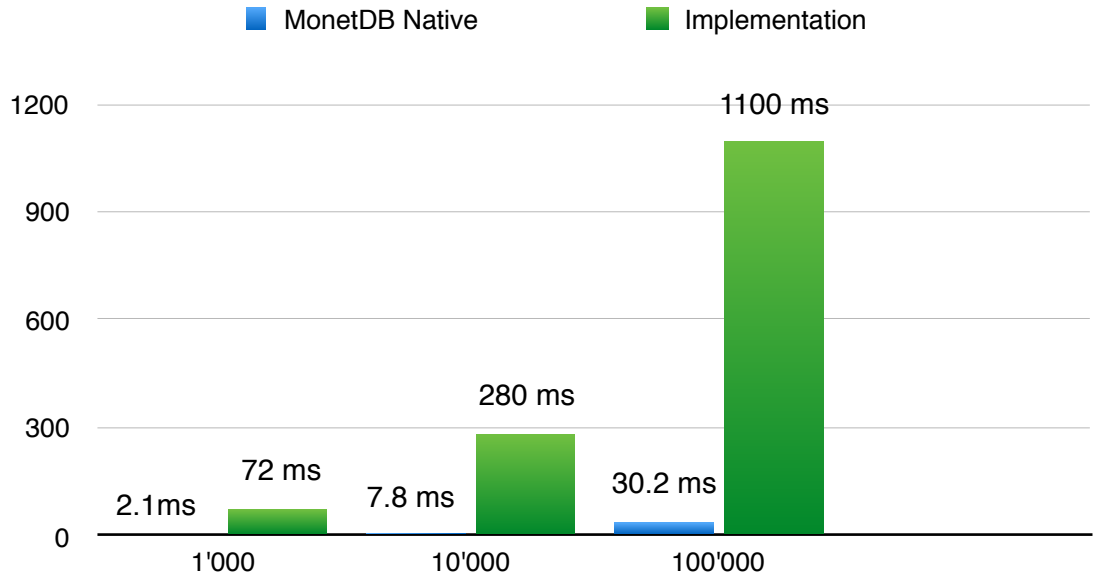
from table2. This yields the values (25, 50) for (a, b) and (25, 60) for (x, y). The result of the query is therefore the row (25, 50, 25, 60) for (a, b, x, y).



Statement tree, LFJ = left-fetch join

5. Implementation

By implementing a function that is based on copying the data into an array, adding the values and copying it back into a BAT, it should be shown that MonetDB's own implementation is faster than a simple function based on copying values.
The implemented function takes several parameters as an input, including the two BAT's that should be added and a BAT where the result should be filled in. Memory is allocated for two arrays of type double which are filled with the function „BATloop" that iterates over the values in the BAT's. Additionally, the memory for the resulting BAT is allocated and the result-array is pointed to this memory. A for-loop fills the result-array by adding the array-values of the two arrays. After that, the memory is freed.
The comparison of the runtimes for tables with 1'000, 10'000 and 100'000 yielded to the following results:



MonetDB's original implementation performed several times better than the implementation based on copying. When copying the data, there is an increased memory usage because memory needs to be allocated for two double arrays and one result-array. Assuming a size of 8 bytes for a double, the three cases need following capacity:

1'000 entries:	24'000 bytes	around 0.02 Megabytes
10'000 entries:	240'000 bytes	around 0.22 Megabytes
100'000 entries:	2'400'000 bytes	around 2.28 Megabytes

6. Conclusion

I gained some interesting insights into the inner workings of MonetDB, a column-oriented DBMS. In addition to studying the structure and how queries are broken down, I implemented a function on the lowest level in the C language and analyzed the difference of performance. Unfortunately, there was not enough time to implement the optional part that includes the LAPACK library but nevertheless, this project was an interesting experience in the big field of database systems.