MonetDB: Open-source Database Technology Beyond Textbooks

http://monetdb.cwi.nl/

Stefan Manegold
Stefan.Manegold@cwi.nl
http://homepages.cwi.nl/~manegold/
Query Processing at Light Speed

MonetDB is an open-source database system for high-performance applications in data mining, OLAP, GIS, XML Query, text and multimedia retrieval. MonetDB often achieves a significant speed improvement for SQL and XQuery over other open-source systems. Use the product strands as your roadmap for exploration.

Products

- MonetDB/SQL our relational database solution.
- MonetDB/XQuery our XML database solution.
- MonetDB Server our multi-model database server.

Science & Technology at Work

MonetDB achieves its goal by innovations at all layers of a DBMS, e.g. a storage model based on vertical fragmentation, a modern CPU-tuned query execution architecture, automatic and self-tuning indexes, run-time query optimization, and a modular software architecture. In-depth information on the technical innovations in the design and implementation of MonetDB can be found in our science library.

Commercial support

An independent company has been set up to facilitate the dissemination of the MonetDB code base as an open source solution worldwide and to provide expert consultancy, participation in projects, and joint ventures on a commercial basis to realize its mission.
Who?

Database Architecture Group @ CWI
(Centrum Wiskunde & Informatica)

- People
  - 1 Professor
  - 1 Scientific programmer
  - 3 Tenure researchers
  - 1 Tenure-track researcher
  - 3 Postdocs
  - 4 PhD students
  - Master students
  - (6 nationalities)
  - 4 Vacancies
Database Architecture Group @ CWI
(Centrum Wiskunde & Informatica)

• Research Focus
  • Database Architecture
    – Hardware Awareness, Self-*, Performance, Multi-modal

• Research Approach
  • R & D
  • Strong Engineering
  • System-oriented
  • Experimental

• Research Products
  • Scientific Papers, PhD Theses, ...
  • Open-Source MonetDB
Why?

Motivation

- Relational DBMSs dominate since the late 1970's / early 1980's
  - Ingres, Oracle, SQLserver, DB2, ...
  - Transactional workloads (OLTP, row-wise access)
  - I/O based processing
### Workload changes: Transactions (OLTP) vs ...

<table>
<thead>
<tr>
<th>contract</th>
<th>client</th>
<th>date</th>
<th>name</th>
<th>price</th>
<th>city</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>12302346</td>
<td>10042334</td>
<td>Eno</td>
<td>Redmond</td>
<td>Car</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37611373</td>
<td>10987097</td>
<td>Gotz</td>
<td>Berkeley</td>
<td>Redmond</td>
<td>update query</td>
<td></td>
</tr>
<tr>
<td>95371001</td>
<td>10032112</td>
<td>Chen</td>
<td>Seattle</td>
<td>House</td>
<td>lookup query</td>
<td></td>
</tr>
<tr>
<td>51213123</td>
<td>10032423</td>
<td>Jones</td>
<td>Washington</td>
<td>Travel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54535545</td>
<td>10087823</td>
<td>Smith</td>
<td>New York</td>
<td>House</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45447894</td>
<td>10013232</td>
<td>Doe</td>
<td>Boston</td>
<td>Car</td>
<td>insert query</td>
<td></td>
</tr>
</tbody>
</table>

**OLTP queries:** access all columns of just one row.
Why?

Workload changes: ... vs OLAP, BI, Data Mining, ...

OLAP query: accesses only a few columns of almost all rows.

select those tuples sold after march 21

sum claims

while grouping by city and product
Databases hit The Memory Wall


- CPU is 60%-90% idle, waiting for memory:
  - L1 data stalls
  - L1 instruction stalls
  - L2 data stalls
  - TLB stalls
  - Branch mispredictions
  - Resource stalls
Hardware Changes: The Memory Wall

Why?

Trip to memory = 1000s of instructions!
Hardware Changes: Memory Hierarchies

- Caches trade off capacity for speed
- Exploit instruction/data locality
- Demand fetch/wait for data

[ADH99]:
- Running top 4 database systems
- At most 50% CPU utilization

+Transition Lookaside Buffer (TLB)
Cache for VM address translation ➔ only 64 entries!
Evolution

It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to change.

Charles Darwin (1809 - 1882)
“We can't solve problems by using the same kind of thinking we used when we created them.”
What?

MonetDB

- Database kernel developed at CWI since 1993
  - *Research prototype turned into open-source product*
- Pioneering columnar database architecture
  - *Complete Relational/SQL & XML/XQuery DBMS*
- Focusing on in-memory processing
  - *Data is kept persistent on disk and can exceed memory limits*
- Aiming at OLAP, BI & Data Mining workloads (“read-dominated”)
  - *Supporting ACID transactions (WAL, optimistic CC)*
- Platform for database architecture research
  - *Used in academia (research & teaching) & commercial environments*
- Back-end for various DB research projects:
  - Multi-Media DB & IR (“Tijah”), XML/XQuery (“Pathfinder”),
  - Data Mining (“Proximity”), Digital Forensics (“XIRAF”), GIS (“OSM”), ...
How is MonetDB Different

- full vertical fragmentation: always!
  - everything in binary (2-column) tables (Binary Association Table)
  - saves you from table scan hell in OLAP and Data Mining

- RISC approach to databases
  - simple back-end data model
  - simple back-end query language (binary/columnar relational algebra)
  - don’t need (to pay for) a buffer manager => manage virtual memory
  - explicit transaction management => DIY approach to ACID

- Multiple user data models & query languages
  - SQL, XML/XQuery, (RDF/SPARQL)
  - front-ends map data models and query languages
How is MonetDB Different

- optimized for in-memory processing
  - efficiently exploit large main-memories
  - data is persistent, though
- operator-at-a-time bulk processing
  - avoids tuple-at-a-time management overhead
- CPU and memory cache optimized
  - programming team experienced in main memory DBMS techniques
  - use of scientific programming optimizations (loop unrolling)
MonetDB vs Traditional DBMS Architecture

- Architecture-Conscious Query Processing
  - vs Magnetic disk I/O conscious processing
  - Data layout, algorithms, cost models

- RISC Relational Algebra (operator-at-a-time)
  - vs Tuple-at-a-time Iterator Model
  - Faster through simplicity: no tuple expression interpreter

- Multi-Model: ODMG, SQL, XML/XQuery, ..., RDF/SPARQL
  - vs Relational with Bolt-on Subsystems
  - Columns as the building block for complex data structures

- Decoupling of Transactions from Execution/Buffering
  - vs ARIES integrated into Execution/Buffering/Indexing
  - ACID, but not ARIES. Pay as you need transaction overhead.

- Run-Time Indexing and Query Optimization
  - vs Static DBA/Workload-driven Optimization & Indexing
    - Extensible Optimizer Framework;
    - cracking, recycling, sampling-based runtime optimization
The MonetDB Software Stack

Front-ends
- XQuery
- SQL 03
- Optimizers
- RDF
- Arrays

Back-end(s)
- MonetDB 4
- MonetDB 5

Kernel
- MonetDB kernel
### Storing Relations in MonetDB

<table>
<thead>
<tr>
<th>contract</th>
<th>client</th>
<th>date</th>
<th>name</th>
<th>price</th>
<th>city</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>12302346</td>
<td>10042334</td>
<td>Eno</td>
<td></td>
<td></td>
<td>Redmond</td>
<td>Car</td>
</tr>
<tr>
<td>37611373</td>
<td>10987097</td>
<td>Gotz</td>
<td></td>
<td></td>
<td>Redmond</td>
<td>House</td>
</tr>
<tr>
<td>51213123</td>
<td>10032423</td>
<td>Jones</td>
<td></td>
<td></td>
<td>Washington</td>
<td>Travel</td>
</tr>
<tr>
<td>54535545</td>
<td>10087823</td>
<td>Smith</td>
<td></td>
<td></td>
<td>New York</td>
<td>House</td>
</tr>
<tr>
<td>45447894</td>
<td>10013232</td>
<td>Doe</td>
<td></td>
<td></td>
<td>Boston</td>
<td>Car</td>
</tr>
<tr>
<td>95371001</td>
<td>10032112</td>
<td>Chen</td>
<td></td>
<td></td>
<td>Seattle</td>
<td>House</td>
</tr>
</tbody>
</table>

**Virtual OID**: `seqbase=1000` (increment=1)
BAT Data Structure

BAT: binary association table
BUN: binary unit
Head & Tail:
- consecutive memory blocks (arrays)
- memory-mapped files
Tail Heap:
- best-effort duplicate elimination for strings (~ dictionary encoding)
BAT Property Management

type - (physical) type number
dense - dense ascending range (OID only)
(rev)sorted - column sorted ascending (descending)
constant - all equal values
align - unique sequence id
key - no duplicates on column
set - no duplicates in BAT
hash - accelerator flag
mirrored - head=tail value
count - cardinality
nonil - no NIL (NULL) values in column
RISC Relational Algebra

```c
batcalc_minus_int(int* res,
    int* col,
    int val,
    int n)
{
    for(i=0; i<n; i++)
        res[i] = col[i] - val;
}
```

**CPU 🥰?** Give it “nice” code!

- few dependencies (control, data)
- CPU gets out-of-order execution
- compiler can e.g. generate SIMD

**One loop for an entire column**

- no per-tuple interpretation
- arrays: no record navigation
- better instruction cache locality

**SELECT** id, name, (age-30)*50 as bonus
**FROM** people
**WHERE** age > 30

Simple, hard-coded semantics in operators

MATERIALIZED intermediate results
The MonetDB Software Stack

Front-ends
- XQuery
- SQL 03
- Optimizers

Back-end(s)
- MonetDB 4
- MonetDB 5

Kernel
- MonetDB kernel

Runtime operational optimization
Processing Model (MonetDB Kernel)

- **Bulk processing:**
  - full materialization of all intermediate results
- **Binary (i.e., 2-column) algebra core:**
  - select, join, semijoin, outerjoin
  - union, intersection, diff (BAT-wise & column-wise)
  - group, count, max, min, sum, avg
  - reverse, mirror, mark
- **Runtime operational optimization:**
  - Choosing optimal algorithm & implementation according to input properties and system status
Processing Model (MonetDB Kernel)

- Heavy use of code expansion to reduce cost

1 algebra operator

3 overloaded operators

10 operator algorithms

~1500(!) routines (macro expansion)

- ~1500 selection routines
- 149 unary operations
- 335 join/group operations
- ...

select()

select("=" value)

select("between", L, H)

select("fcn", parm)

scan hash-lookup bin-search bin-tree pos-lookup

scan_range_select_oid_int(),
hash_equi_select_void_str(), ...
The MonetDB Software Stack

Front-ends
- XQuery
- SQL 03

Back-end(s)
- MonetDB 4
- MonetDB 5

Kernel
- MonetDB kernel

MAL

Tactical optimization: MAL -> MAL rewrites
MonetDB/5 Back-end: MAL

- **MAL**: Monet Assembly Language
  - textual interface
  - Interpreted language
- **Designed as system interface language**
  - Reduced, concise syntax
  - Strict typing
  - Meant for automatic generation and parsing/rewriting/processing
  - Not meant to be typed by humans
- **Efficient parser**
  - Low overhead
  - Inherent support for *tactical optimization*: MAL -> MAL
  - Support for optimizer plug-ins
  - Support for runtime schedulers
- **Binary-algebra core**
- **Flow control** (MAL is computational complete)
MonetDB/5 Back-end: MAL Optimizers

- General front-end independent MAL -&gt; MAL rewriting
  - Implemented once, shared by all (future) front-ends
- Examples:
  - Constant propagation
  - Scalar expression evaluation
  - Dead-code elimination
  - Common sub-expression elimination
  - Empty result set removal
  - Reordering to optimize intermediate result usage
  - Reordering of linear (projection-) join chains
- Parallelization:
  - Dataflow analysis
  - Horizontal partitioning
  - Remote execution
- Cracking
- Recycling
- ...
The MonetDB Software Stack

Front-ends
- XQuery
- SQL 03

Back-end(s)
- MonetDB 4
- MonetDB 5

Kernel
- MonetDB kernel

Strategic optimization
- MAL

Optimizers
MonetDB Front-end: SQL

- SQL 2003
  - Stick to the standard; avoid “private” extensions, renaming, etc.
- Parse SQL into logical n-ary relational algebra tree
- Translate n-ary relational algebra into logical 2-ary relational algebra
- Turn logical 2-ary plan into physical 2-ary plan (MAL program)
  - Generate internal tree representation, not textual MAL program
- Front-end specific strategic optimization:
  - Heuristic optimization during all three previous steps
- Primary key and distinct constraints:
  - Create and maintain hash indices
- Foreign key constraints
  - Create and maintain foreign key join indices
- Exploit both above indices during query evaluation
PLAN SELECT a FROM t WHERE c < 10;

project (  
  select (  
    table(sys.t) [ t.a, t.c, t.%TID% NOT NULL ]  
  ) [ t.c < convert(10) ]  
) [ t.a ]
EXPLAIN SELECT a FROM t WHERE c < 10;

function user.s1_1{autoCommit=true}():void;
barrier _55 := language.dataflow();
  _2:bat[:oid,:int] := sql.bind("sys","t","c",0);
  _7 := algebra.thetauselect(_2,10,"<");
  _2:bat[:oid,:int] := nil:BAT;
  _7 := algebra.markT(_7,0@0);
  _7 := nil:BAT;
  _11 := bat.reverse(_10);
  _10 := nil:BAT;
  _12:bat[:oid,:int] := sql.bind("sys","t","a",0);
  _14 := algebra.leftjoin(_11,_12);
  _11 := nil:BAT;
  _12:bat[:oid,:int] := nil:BAT;
exit _55;
  _15 := sql.resultSet(1,1,_14);
sql.rsColumn(_15,"sys.t","a","int",32,0,_14);
  _14 := nil:BAT;
  _21 := io.stdout();
sql.exportResult(_21,_15);
end s1_1;
PLAN SELECT a, z FROM t, s WHERE t.c = s.x;

project (  
join (  
  table(sys.t) [ t.a, t.c, t.%TID% NOT NULL ],  
  table(sys.s) [ s.x, s.z, s.%TID% NOT NULL ]  
) [ t.c = s.x ]  
) [ t.a, s.z ]
EXPLAIN SELECT a, z FROM t, s WHERE t.c = s.x;

% .explain # table_name
% mal # name
% clob # type
% 0 # length
function user.s2_1{autoCommit=true}():void;
barrier _73 := language.dataflow();
_2:bat[oid,:int] := sql.bind("sys","t","c",0);
_7:bat[oid,:int] := sql.bind("sys","s","x",0);
_10 := bat.reverse(_7);
_7:bat[oid,:int] := nil:BAT;
_11 := algebra.join(_2,_10);
_2:bat[oid,:int] := nil:BAT;
_10 := nil:BAT;
_13 := algebra.markT(_11,0@0);
_14 := bat.reverse(_13);
_13 := nil:BAT;
_15:bat[oid,:int] := sql.bind("sys","t","a",0);
_17 := algebra.leftjoin(_14,_15);
_14 := nil:BAT;
_15:bat[oid,:int] := nil:BAT;
_18 := bat.reverse(_11);
_11 := nil:BAT;
_19 := algebra.markT(_18,0@0);
_18 := nil:BAT;
_20 := bat.reverse(_19);
_19 := nil:BAT;
_21:bat[oid,:int] := sql.bind("sys","s","z",0);
_23 := algebra.leftjoin(_20,_21);
_20 := nil:BAT;
_21:bat[oid,:int] := nil:BAT;
exit _73;
_24 := sql.resultSet(2,1,_17);
sql.rsColumn(_24,"sys.t","a","int",32,0,_17);
_17 := nil:BAT;
sql.rsColumn(_24,"sys.s","z","int",32,0,_23);
_23 := nil:BAT;
_33 := io.stdout();
sql.exportResult(_33,_24);
end s2_1;
MonetDB Front-end: SQL

- **Updates / transactions:**
  - Write ahead logging
  - Optimistic concurrency control
  - “delta” BATs
<table>
<thead>
<tr>
<th>SF</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1097</td>
<td>1734</td>
<td>4409</td>
</tr>
</tbody>
</table>

TPC-H

60K rows line_item table
Comfortably fit in memory
Performance in milliseconds

ATHLON X2 3800+ (2000mhz) 2 disks in raid 0, 2G main memory
### TPC-H

<table>
<thead>
<tr>
<th>Load</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1097</td>
<td>1734</td>
<td>4409</td>
<td>96466</td>
<td>342103</td>
<td>140888</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
<td>558</td>
<td>366</td>
<td>4004</td>
<td>61253</td>
<td>36305</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>15</td>
<td>35</td>
<td>104</td>
<td>3344</td>
<td>152870</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>39</td>
<td>141</td>
<td>935</td>
<td>18504</td>
<td>34137</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>18</td>
<td>34</td>
<td>663</td>
<td>3273</td>
<td>3187</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>21</td>
<td>223</td>
<td>867</td>
<td>758</td>
<td>19931</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>44</td>
<td>156</td>
<td>171</td>
<td>14011</td>
<td>6897</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>31</td>
<td>59</td>
<td>902</td>
<td>17395</td>
<td>5967</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>35</td>
<td>30</td>
<td>353</td>
<td>17101</td>
<td>2940</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>124</td>
<td>75</td>
<td>855</td>
<td>49769</td>
<td>9233</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>20</td>
<td>132</td>
<td>692</td>
<td>1014</td>
<td>12424</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>21</td>
<td>27</td>
<td>65</td>
<td>2491</td>
<td>39164</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>61</td>
<td>74</td>
<td>404</td>
<td>14303</td>
<td>6023</td>
</tr>
<tr>
<td>13</td>
<td>47</td>
<td>51</td>
<td>112</td>
<td>5532</td>
<td>6461</td>
<td>8128</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>43</td>
<td>301</td>
<td>350</td>
<td>13129</td>
<td>32832</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>67</td>
<td>152</td>
<td>487</td>
<td>8972</td>
<td>9086</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>46</td>
<td>80</td>
<td>176</td>
<td>&gt; 1 hour</td>
<td>1042</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>92</td>
<td>18</td>
<td>698</td>
<td>35273</td>
<td>&gt; 1 hour</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>64</td>
<td>10</td>
<td>1060</td>
<td>15208</td>
<td>297</td>
</tr>
<tr>
<td>19</td>
<td>12</td>
<td>1810</td>
<td>11</td>
<td>538</td>
<td>&gt; 1 hour</td>
<td>16936</td>
</tr>
<tr>
<td>20</td>
<td>26</td>
<td>199</td>
<td>27</td>
<td>2483</td>
<td>47391</td>
<td>99153</td>
</tr>
<tr>
<td>21</td>
<td>26</td>
<td>199</td>
<td>27</td>
<td>2483</td>
<td>47391</td>
<td>99153</td>
</tr>
<tr>
<td>22</td>
<td>8</td>
<td>420</td>
<td>13</td>
<td>246</td>
<td>&gt; 1 hour</td>
<td>718</td>
</tr>
</tbody>
</table>

**ATHLON X2 3800+ (2000mhz) 2 disks in raid 0, 2G main memory**

**Scale-factor 1**

**6M row line-item table**

**Out of the box performance**

**Queries produce empty or erroneous results**
<table>
<thead>
<tr>
<th>SF</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1097</td>
<td>1734</td>
<td>4409</td>
<td>96466</td>
<td>342103</td>
<td>140888</td>
<td>211</td>
<td>733</td>
<td>303</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
<td>558</td>
<td>366</td>
<td>4004</td>
<td>61253</td>
<td>36305</td>
<td>6</td>
<td>121</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>15</td>
<td>35</td>
<td>104</td>
<td>3344</td>
<td>152870</td>
<td>1</td>
<td>6</td>
<td>308</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>39</td>
<td>141</td>
<td>935</td>
<td>18504</td>
<td>34137</td>
<td>3</td>
<td>31</td>
<td>91</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>18</td>
<td>34</td>
<td>663</td>
<td>3273</td>
<td>3187</td>
<td>1</td>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>21</td>
<td>223</td>
<td>867</td>
<td>758</td>
<td>19931</td>
<td>3</td>
<td>2</td>
<td>5755</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>44</td>
<td>156</td>
<td>171</td>
<td>14011</td>
<td>6897</td>
<td>0</td>
<td>23</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>31</td>
<td>59</td>
<td>902</td>
<td>17395</td>
<td>5967</td>
<td>2</td>
<td>37</td>
<td>2655</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>35</td>
<td>30</td>
<td>335</td>
<td>17101</td>
<td>2940</td>
<td>121</td>
<td>32</td>
<td>504</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>124</td>
<td>75</td>
<td>855</td>
<td>49769</td>
<td>9233</td>
<td>3</td>
<td>256</td>
<td>57</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>20</td>
<td>132</td>
<td>692</td>
<td>1014</td>
<td>12424</td>
<td>3</td>
<td>4</td>
<td>91</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>21</td>
<td>27</td>
<td>65</td>
<td>2491</td>
<td>39164</td>
<td>1</td>
<td>5</td>
<td>175</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>61</td>
<td>74</td>
<td>404</td>
<td>14303</td>
<td>6023</td>
<td>2</td>
<td>27</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>47</td>
<td>51</td>
<td>112</td>
<td>5532</td>
<td>6461</td>
<td>8126</td>
<td>20</td>
<td>15</td>
<td>132</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>43</td>
<td>301</td>
<td>350</td>
<td>13129</td>
<td>32832</td>
<td>2</td>
<td>27</td>
<td>15193</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>67</td>
<td>152</td>
<td>110</td>
<td>15093</td>
<td>18023</td>
<td>1</td>
<td>31</td>
<td>49</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td>46</td>
<td>80</td>
<td>487</td>
<td>8972</td>
<td>9086</td>
<td>1</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>12</td>
<td>10</td>
<td>176</td>
<td>&gt; 1 hour</td>
<td>1042</td>
<td>4</td>
<td>&gt; 1 hour</td>
<td>27</td>
</tr>
<tr>
<td>18</td>
<td>11</td>
<td>92</td>
<td>18</td>
<td>698</td>
<td>35273</td>
<td>&gt; 1 hour</td>
<td>13</td>
<td>125</td>
<td>&gt; 1 hour</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>64</td>
<td>64</td>
<td>1060</td>
<td>15208</td>
<td>297</td>
<td>4</td>
<td>31</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>1810</td>
<td>11</td>
<td>538</td>
<td>&gt; 1 hour</td>
<td>16936</td>
<td>1</td>
<td>&gt; 1 hour</td>
<td>162</td>
</tr>
<tr>
<td>21</td>
<td>26</td>
<td>199</td>
<td>27</td>
<td>2483</td>
<td>47391</td>
<td>99153</td>
<td>131</td>
<td>107</td>
<td>1492</td>
</tr>
<tr>
<td>22</td>
<td>8</td>
<td>420</td>
<td>13</td>
<td>246</td>
<td>&gt; 1 hour</td>
<td>718</td>
<td>3</td>
<td>&gt; 1 hour</td>
<td>1755</td>
</tr>
</tbody>
</table>

ATHLON X2 3800+ (2000mhz) 2 disks in raid 0, 2G main memory
**TPC-H**

<table>
<thead>
<tr>
<th>SF</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
<th>MonetDB 5.2.3</th>
<th>PostgreSQL 8.2.6</th>
<th>MySQL 5.0.45</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1097</td>
<td>1734</td>
<td>4409</td>
<td>221</td>
<td>733</td>
<td>303</td>
<td>5</td>
<td>793</td>
<td>3570</td>
</tr>
<tr>
<td>1</td>
<td>29 ms</td>
<td>558 ms</td>
<td>366 ms</td>
<td>4004</td>
<td>61253 ms</td>
<td>36305 ms</td>
<td>6</td>
<td>121 sec</td>
<td>80 sec</td>
</tr>
<tr>
<td>2</td>
<td>9 sec</td>
<td>15 sec</td>
<td>35 sec</td>
<td>104</td>
<td>3344</td>
<td>152870 sec</td>
<td>1 sec</td>
<td>6 sec</td>
<td>308 sec</td>
</tr>
<tr>
<td>3</td>
<td>12 sec</td>
<td>39 sec</td>
<td>141 sec</td>
<td>935</td>
<td>18504</td>
<td>34137 sec</td>
<td>3 sec</td>
<td>31 sec</td>
<td>91 sec</td>
</tr>
<tr>
<td>4</td>
<td>10 sec</td>
<td>18 sec</td>
<td>34 sec</td>
<td>663</td>
<td>3273</td>
<td>3187 sec</td>
<td>1 sec</td>
<td>6 sec</td>
<td>17 sec</td>
</tr>
<tr>
<td>5</td>
<td>12 sec</td>
<td>21 sec</td>
<td>223 sec</td>
<td>867</td>
<td>758</td>
<td>19931 sec</td>
<td>3 sec</td>
<td>2 sec</td>
<td>5755 sec</td>
</tr>
<tr>
<td>6</td>
<td>4 sec</td>
<td>44 sec</td>
<td>156 sec</td>
<td>171</td>
<td>14011</td>
<td>6897 sec</td>
<td>0 sec</td>
<td>23 sec</td>
<td>27 sec</td>
</tr>
<tr>
<td>7</td>
<td>13 sec</td>
<td>31 sec</td>
<td>59 sec</td>
<td>902</td>
<td>17395</td>
<td>5967 sec</td>
<td>2 sec</td>
<td>37 sec</td>
<td>2655 sec</td>
</tr>
<tr>
<td>8</td>
<td>9 sec</td>
<td>35 sec</td>
<td>30 sec</td>
<td>353</td>
<td>17101</td>
<td>2940 sec</td>
<td>121 sec</td>
<td>32 sec</td>
<td>504 sec</td>
</tr>
<tr>
<td>9</td>
<td>14 sec</td>
<td>124 sec</td>
<td>75 sec</td>
<td>855</td>
<td>49769</td>
<td>9233 sec</td>
<td>3 sec</td>
<td>256 sec</td>
<td>57 sec</td>
</tr>
<tr>
<td>10</td>
<td>12 sec</td>
<td>20 sec</td>
<td>132 sec</td>
<td>692</td>
<td>1014</td>
<td>12424 sec</td>
<td>3 sec</td>
<td>4 sec</td>
<td>91 sec</td>
</tr>
<tr>
<td>11</td>
<td>7 sec</td>
<td>21 sec</td>
<td>27 sec</td>
<td>65</td>
<td>2491</td>
<td>39164 sec</td>
<td>1 sec</td>
<td>5 sec</td>
<td>175 sec</td>
</tr>
<tr>
<td>12</td>
<td>9 sec</td>
<td>61 sec</td>
<td>74 sec</td>
<td>404</td>
<td>14303</td>
<td>6023 sec</td>
<td>2 sec</td>
<td>27 sec</td>
<td>23 sec</td>
</tr>
<tr>
<td>13</td>
<td>47 sec</td>
<td>51 sec</td>
<td>112 sec</td>
<td>5532</td>
<td>6461</td>
<td>8126 sec</td>
<td>20 sec</td>
<td>15 sec</td>
<td>132 sec</td>
</tr>
<tr>
<td>14</td>
<td>7 sec</td>
<td>43 sec</td>
<td>301 sec</td>
<td>350</td>
<td>13129</td>
<td>32832 sec</td>
<td>2 sec</td>
<td>27 sec</td>
<td>15193 sec</td>
</tr>
<tr>
<td>15</td>
<td>6 sec</td>
<td>67 sec</td>
<td>152 sec</td>
<td>110</td>
<td>15093</td>
<td>18023 sec</td>
<td>1 sec</td>
<td>31 sec</td>
<td>49 sec</td>
</tr>
<tr>
<td>16</td>
<td>9 sec</td>
<td>46 sec</td>
<td>80 sec</td>
<td>487</td>
<td>8972</td>
<td>9086 sec</td>
<td>1 sec</td>
<td>17 sec</td>
<td>19 sec</td>
</tr>
<tr>
<td>17</td>
<td>4 sec</td>
<td>12 sec</td>
<td>10 sec</td>
<td>176 sec</td>
<td>&gt; 1 hour</td>
<td>1042 sec</td>
<td>4 sec</td>
<td>&gt; 1 hour</td>
<td>27 sec</td>
</tr>
<tr>
<td>18</td>
<td>11 sec</td>
<td>92 sec</td>
<td>18 sec</td>
<td>698 sec</td>
<td>35273 sec</td>
<td>&gt; 1 hour</td>
<td>13 sec</td>
<td>125 sec</td>
<td>&gt; 1 hour</td>
</tr>
<tr>
<td>19</td>
<td>18 sec</td>
<td>64 sec</td>
<td>297 sec</td>
<td>1060</td>
<td>15208</td>
<td>297 sec</td>
<td>4 sec</td>
<td>31 sec</td>
<td>50 sec</td>
</tr>
<tr>
<td>20</td>
<td>12 sec</td>
<td>1810 sec</td>
<td>11 sec</td>
<td>538 sec</td>
<td>&gt; 1 hour</td>
<td>16936 sec</td>
<td>1 sec</td>
<td>&gt; 1 hour</td>
<td>162 sec</td>
</tr>
<tr>
<td>21</td>
<td>26 sec</td>
<td>199 sec</td>
<td>27 sec</td>
<td>2483</td>
<td>47391</td>
<td>99153 sec</td>
<td>131 sec</td>
<td>107 sec</td>
<td>1492 sec</td>
</tr>
<tr>
<td>22</td>
<td>8 sec</td>
<td>420 sec</td>
<td>13 sec</td>
<td>246 sec</td>
<td>&gt; 1 hour</td>
<td>718 sec</td>
<td>3 sec</td>
<td>&gt; 1 hour</td>
<td>1755 sec</td>
</tr>
</tbody>
</table>

ATHLON X2 3800+ (2000mhz) 2 disks in raid 0, 2G main memory
MySQL Performance Blog

- table with 200 integer columns
- random numbers between 0 and 10000
- Loading 1 million rows

![Load Time Graph]

- infobright
- monetdb
- myisam

Load Time vs Seconds
1. `SELECT sum(c19), sum(c89), sum(c129) FROM t;`
2. `SELECT sum(c19), sum(c89), sum(c129) FROM t WHERE c11 > 5;`
3. `SELECT sum(c19), sum(c89), sum(c129) FROM t WHERE c11 < 5;`

<table>
<thead>
<tr>
<th></th>
<th>myisam</th>
<th>monetdb</th>
<th>infobright</th>
</tr>
</thead>
<tbody>
<tr>
<td>size (bytes)</td>
<td>826000000</td>
<td>829946723</td>
<td>332497242</td>
</tr>
<tr>
<td>load time (seconds)</td>
<td>88</td>
<td>200</td>
<td>486</td>
</tr>
<tr>
<td>query1 time</td>
<td>3.4</td>
<td>0.012</td>
<td>0.0007</td>
</tr>
<tr>
<td>query2 time</td>
<td>3.4</td>
<td>0.15</td>
<td>1.2</td>
</tr>
<tr>
<td>query3 time</td>
<td>2.5</td>
<td>0.076</td>
<td>0.15</td>
</tr>
</tbody>
</table>
I tried to analyze a bigger dataset and I took public available data about USA domestic flights with information about flight length and delays. In total raw data is about 55GB.
Open-Source Development

- Feature releases: 3-4 per year
  - Research results
  - User requests
- Bug-fix releases: monthly
- QA
  - Automated nightly testing on >20 platforms
  - Ensure correctness & stability
  - Ensure portability
  - Bug reports become test cases
  - Semi-automatic performance monitoring
  - Passed static code verification by Coverity with only minor problems
<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
<th>Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Usage

- CWI, Amsterdam:
  - Core DBMS Research
  - TIJAH: Multi-Media IR
  - Data Mining, GIS, Astronomy, RDF/SPARQL, Streams, ...
- Data Distilleries (CWI Spin-Off, now part of SPSS -> IBM), Amsterdam:
  - Commercial Data-Mining & CRM Software
  - Many banks & insurance companies in NL
- Knowledge Discovery Lab, UMass, Amherst:
  - Proximity: OpenSource relational knowledge discovery tool
- Universität Tübingen (with UTwente & CWI):
  - Pathfinder: XQuery compiler
Usage

- Dutch National Forensics Institute (NFI):
  - XIRAF: Digital Forensics
- University of Utrecht:
  - Data Mining & Bio-Informatics
- University of Twente:
  - Distributed Object-Oriented Data Management
  - Probabilistic databases
- Open-Source Community:
  - OpenStreetMap: Highperformance API
    - MonetDB, Cherokee, Sphinx, Django
  - Active user community (mailing lists)
  - (many more “anonymously”)
- >10000 downloads per month
Welcome to the DR6 site!
The Sixth Data Release is dedicated to Jim Gray for his fundamental contribution to the SDSS project and the extraordinary energy and passion he shared with everybody!

This website presents data from the Sloan Digital Sky Survey, a project to make a map of a large part of the universe. We would like to show you the beauty of the universe, and share with you our excitement as we build the largest map in the history of the world.

News
The site hosts data from Data Release 6 (DR6). What's new in DR6, what's new on this site, and known problems.
More...

For Astronomers
A separate branch of this website for professional astronomers (English)

More...

SkyServer Tools
Famous places
Get images
Visual Tools
Explore
Search
Object upload
CasJobs

Science Projects
Basic
Advanced
Challenges
For kids
Games and contests
Teachers
Links to other projects

Info Links
About Astronomy
About the SDSS
About the SkyServer
SDSS Data Release 6
SDSS Project Website
Open SkyQuery
Images of RC3 Galaxies

Help
Getting Started
FAQ
How To
Glossary
Schema Browser
Sample SQL Queries
Details of SDSS Data

Contact Us

SDSS is supported by

Powered by
Microsoft Internet Explorer
Privacy Policy
SkyServer Schema

446 columns > 585 million rows

6 columns > 20 Billion rows
Recycler
motivation & idea

Motivation:

- scientific databases, data analytics
- Terabytes of data (observational, transactional)
- Prevailing read-only workload
- Ad-hoc queries with commonalities

Background:

- Operator-at-a-time execution paradigm
  - Automatic materialization of intermediates
- Canonical column-store organization
  - Intermediates have reduced dimensionality and finer granularity
  - Simplified overlap analysis

Recycling idea:

- instead of garbage collecting, keep the intermediates and reuse them
  - speed up query streams with commonalities
  - low cost and self-organization

Recycler
fit into MonetDB

function user.s1_2(A0:date, ...):void;
  X5 := sql.bind("sys","lineitem",...);
  X10 := algebra.select(X5,A0);
  X12 := sql.bindIdx("sys","lineitem",...);
  X15 := algebra.join(X10,X12);
  X25 := mtime.addmonths(A1,A2);
  ...

function user.s1_2(A0:date, ...):void;
  X5 := sql.bind("sys","lineitem",...);
  X10 := algebra.select(X5,A0);
  X12 := sql.bindIdx("sys","lineitem",...);
  X15 := algebra.join(X10,X12);
  X25 := mtime.addmonths(A1,A2);
  ...

Run time comparison of

- instruction types
- argument values

**Exact matching**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Data type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>10</td>
<td>:bat[:oid,:date]</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>&quot;sys&quot;</td>
<td>:str</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>&quot;orders&quot;</td>
<td>:str</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
X1 := sql.bind("sys","orders","o_orderdate",0);

Y3 := sql.bind("sys","orders","o_orderdate",0);
```

Recycler
instruction subsumption

Y3 := algebra.select(X1,10,45);

X3 := algebra.select(X1,10,80);
...

X5 := algebra.select(X1,20,60);

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Data type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>10</td>
<td>:bat[:oid,:int]</td>
<td>2000</td>
</tr>
<tr>
<td>X3</td>
<td>130</td>
<td>:bat[:oid,:int]</td>
<td>700</td>
</tr>
<tr>
<td>X5</td>
<td>150</td>
<td>:bat[:oid,:int]</td>
<td>350</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decide about storing the results

- **KEEPALL**
  - all instructions advised by the optimizer

- **CREDIT**
  - instructions supplied with credits
  - storage ‘paid’ with 1 credit
  - reuse returns credits
  - lack of reuse limits admission and resource claims


“An architecture for recycling intermediates in a column-store”.

CWI Recycler admission policies

MONETDB
Decide about eviction of intermediates

- Pick instructions with smallest utility
  - LRU: time of computation or last reuse
  - BENEFIT: estimated contribution to performance: CPU and I/O costs, recycling
- Triggered by resource limitations (memory or entries)
Sloan Digital Sky Survey / SkyServer
http://cas.sdss.org

- 100 GB subset of DR4
- 100-query batch from January 2008 log
- 1.5GB intermediates, 99% reuse
- Join intermediates major consumer of memory and major contributor to savings

RDF in MonetDB

- Triples store contain only numeric keys
- Keys point to dictionary encoding of URIs & literals
- (transitive closure) queries:
  - Selection (binary search) + series of (self-)joins
RDF in MonetDB

Gene ontology data (~19 M triples)
Idea: each query can be used to re-arrange the data

Build a partial index, tailored to the encountered workload

- Adapt to query workload as it comes
- Create indices/improve locality **during** query processing
- Make optimization decisions continuously, at run-time
- Let the system do this automatically
Cracking how it works

Each query is treated as an advice on how data should be stored.

Updates become pending updates
Applied on demand
Goal: minimum physical actions

Pieces are ordered
Values in a piece are not ordered
New Value: 11

Piece1: $A \leq 10$

Piece2: $10 < A < 14$

Piece3: $14 \leq A$

The more we crack, the more we learn, the less we touch.
Cracking sideways cracking in column stores

Tuple reconstruction adopts the self-organizing properties of cracking
Everything happens on demand and on the fly
Driven by query needs
Only the (areas of the) columns needed are aligned
Cracking TPC-H Performance

“Self-organizing tuple reconstruction in column-stores“, Idreos, Manegold, Kersten, SIGMOD’09

Presorted MonetDB
Preparation cost 3-14 minutes

MonetDB with sideways cracking

No a priori knowledge needed
No idle time needed
Works for random workloads
Works for frequent updates
Cracking for locality in large RDF graphs

- More operators, joins, aggregation, ...
- External cracking
- Concurrency control
- Compression
- Distribution