MonetDB:
Open-source Columnar Database Technology
Beyond Textbooks

http://www.monetdb.org/

Stefan Manegold
Stefan.Manegold@cwi.nl
http://homepages.cwi.nl/~manegold/
monetdb column-store pioneers

DOWNLOAD NOW

features  documentation  developers

An Open-Source Database System
Why?

Motivation (early 1990s)

- Relational DBMSs dominate since the late 1970's / early 1980's
  - IBM DB2, MS SQL Server, Oracle, Ingres, ...
  - Transactional workloads (OLTP, row-wise access)
  - I/O based processing

- But:
  - Workloads change (early 1990s)
  - Hardware changes (late 1990s)
  - Data “explodes” (early 2000s)
### Why?

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></td>
<td>Eno</td>
<td></td>
<td>Redmond</td>
<td>Car</td>
</tr>
<tr>
<td>37611373</td>
<td>10987097</td>
<td></td>
<td>Gotz</td>
<td></td>
<td>Berkeley</td>
<td>Redmond</td>
</tr>
</tbody>
</table>

- **update query**

<table>
<thead>
<tr>
<th>contract</th>
<th>client</th>
<th>name</th>
<th>city</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>95371001</td>
<td>10032112</td>
<td>Chen</td>
<td>Seattle</td>
<td>House</td>
</tr>
<tr>
<td>51213123</td>
<td>10032423</td>
<td>Jones</td>
<td>Washington</td>
<td>Travel</td>
</tr>
<tr>
<td>54535545</td>
<td>10087823</td>
<td>Smith</td>
<td>New York</td>
<td>House</td>
</tr>
</tbody>
</table>

- **lookup query**

<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>45447894</td>
<td>10013232</td>
<td></td>
<td>Doe</td>
<td></td>
<td>Boston</td>
<td>Car</td>
</tr>
</tbody>
</table>

- **insert query**

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

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

select those tuples sold after march 21 → sum claims → while grouping by city and product

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

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!
Solution

“We can't solve problems by using the same kind of thinking we used when we created them.”
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 & scientific 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”), ...
Abbreviated History:

• 1985: DSM (Copeland et al.; SIGMOD 1985)
• 1992: First ideas and kernel for MonetDB (Kersten)
• 1993: MonetDB is born
• 1995?: Sybase IQ (first commercial DSM system)
• 2002: MonetDB goes open-source
• 2004?: Stonebraker et al. start “C-Store” project and coin DSM as “Column-Store”
• 2006?: Stonebraker et al. found “Vertica”; end of “C-Store” as research project
• 2008: Zukowski, Boncz, et al. (CWI) found VectorWise (based on MonetDB/X100)
• 2010: INGRES acquires VectorWise
• 2011: HP acquires Vertica
Storing Relations in MonetDB

DSM => Column-store

<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>Redmond</td>
<td>House</td>
<td></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></td>
<td></td>
</tr>
<tr>
<td>95371001</td>
<td>10032112</td>
<td>Chen</td>
<td>Seattle</td>
<td>House</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Virtual OID: seqbase=1000 (increment=1)
Hash tables, T-trees, R-trees, ...

BAT:
- binary association table

BUN:
- binary unit

BUN Heap:
- consecutive memory block (array)
- memory-mapped files

Tail Heap:
- best-effort duplicate elimination for strings (~ dictionary encoding)
- minimal offset width; adapts automatically (1, 2, 4, 8 byte wide)

Tail:
- consecutive memory block (array)
- memory-mapped files

built automatically on-the-fly when required/beneficial
kept for (later) re-use also for intermediate results

kept for (later) re-use also for intermediate results
SELECT id, name, (age-30)*50 as bonus
FROM people
WHERE age > 30

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

Simple, hard-coded semantics in operators
Zero cost
MATERIALIZED intermediate results
MATERIALIZED intermediate results
VIEWS (not materialized)
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 (BATs)
  • simple back-end query language (binary/columnar relational algebra: MAL)
  • no need (to pay) for a buffer manager => manage virtual memory
  • admission control in scheduler to regulate memory consumption
  • 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

- operator-at-a-time bulk processing
  - avoids tuple-at-a-time management overhead
- CPU and memory cache optimized
  - Techniques adopted from scientific programming
  - Data structures:
    - Arrays
  - Code:
    - Compiler-friendly, branch-free, loop unrolling, instruction cache friendly
  - Algorithms:
    - Exploit spacial & temporal access locality
    - Multi-pass radix-cluster, partitioned hash-join, radix-decluster

=> up to 10x performance improvement
The MonetDB Software Stack

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

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

Kernel
- MonetDB kernel
The MonetDB Software Stack

Front-ends
- XQuery
- SQL 03

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

Kernel
- MonetDB kernel

Runtime operational optimization
# BAT Property Management

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>(physical) type number</td>
</tr>
<tr>
<td>dense</td>
<td>dense ascending range (OID only)</td>
</tr>
<tr>
<td>(rev)sorted</td>
<td>column sorted ascending (descending)</td>
</tr>
<tr>
<td>constant</td>
<td>all equal values</td>
</tr>
<tr>
<td>align</td>
<td>unique sequence id</td>
</tr>
<tr>
<td>key</td>
<td>no duplicates on column</td>
</tr>
<tr>
<td>set</td>
<td>no duplicates in BAT</td>
</tr>
<tr>
<td>hash</td>
<td>accelerator flag</td>
</tr>
<tr>
<td>mirrored</td>
<td>head=tail value</td>
</tr>
<tr>
<td>count</td>
<td>cardinality</td>
</tr>
<tr>
<td>nonil</td>
<td>no NIL (NULL) values in column</td>
</tr>
</tbody>
</table>
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
  - e.g.:
    - hash join vs. merge join
    - hash select vs. binary search vs. scan select
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
- ...
The MonetDB Software Stack

Front-ends
- XQuery
- SQL 03

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

Kernel
- MonetDB kernel

MAL
SELECT id, name, (age-30)*50 as bonus
FROM people
WHERE age > 30
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)
The MonetDB Software Stack

Front-ends
- XQuery
- SQL 03
- Optimizers

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

Kernel
- MonetDB kernel

Tactical optimization: MAL -> MAL rewrites

MAL
MonetDB/5 Back-end: MAL Optimizers

- General front-end independent MAL -> 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
- Optimizers

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

Kernel
- MonetDB kernel

Strategic optimization
- MAL
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
- Cache parametrized physical plans as MAL functions
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():void;
barrier _55 := language.dataflow();

_02:bat[:void,:int] := sql.bind("sys","t","c",0);
_07:bat[:oid, :int] := algebra.thetaselect(_02,10,"<");
_02:bat[:void,:int] := nil:BAT;
_10:bat[:oid,:void] := algebra.markT(_07,0@0);
_07:bat[:oid, :int] := nil:BAT;
_11:bat[:void,:oid] := bat.reverse(_10);
_10:bat[:oid,:void] := nil:BAT;
_12:bat[:oid, :int] := sql.bind("sys","t","a",0);
_14:bat[:void,:int] := algebra.leftjoin(_11,_12);
_11:bat[:void,:oid] := 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:bat[:void,:int] := 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 ]
MonetDB Front-end: SQL

EXPLAIN SELECT a, z FROM t, s WHERE t.c = s.x;

function user.s2_1():void;
barrier _73 := language.dataflow();
_02:bat[:void,:int] := sql.bind("sys","t","c",0);
_07:bat[:void,:int] := sql.bind("sys","s","x",0);
_10:bat[:int,:void] := bat.reverse(_07);
_07:bat[:void,:int] := nil:BAT;
_11:bat[:oid,:oid] := algebra.join(_02,_10);
_02:bat[:void,:int] := nil:BAT;
_10:bat[:int,:void] := nil:BAT;
_13:bat[:oid,:void] := algebra.markT(_11,0@0);
_14:bat[:void,:oid] := bat.reverse(_13);
_15:bat[:void,:int] := sql.bind("sys","t","a",0);
_17:bat[:void,:int] := algebra.leftjoin(_14,_15);
_14:bat[:void,:oid] := nil:BAT;
_15:bat[:void,:int] := nil:BAT;
_18:bat[:oid,:oid] := bat.reverse(_11);
_11:bat[:oid,:oid] := nil:BAT;
_19:bat[:oid,:void] := algebra.markT(_18,0@0);
_18:bat[:oid,:oid] := nil:BAT;
_20:bat[:void,:oid] := bat.reverse(_19);
_19:bat[:void,:void] := nil:BAT;
_21:bat[:void,:int] := sql.bind("sys","s","z",0);
_23:bat[:void,:int] := algebra.leftjoin(_20,_21);
_20:bat[:void,:oid] := nil:BAT;
exit _73;
_24 := sql.resultSet(2,1,_17);
sql.rsColumn(_24,"sys.t","a","int",32,0,_17);
_17:bat[:void,:int] := nil:BAT;
sql.rsColumn(_24,"sys.s","z","int",32,0,_23);
_23:bat[:void,:int] := 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
# TPC-H

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

<table>
<thead>
<tr>
<th>SF</th>
<th>MonetDB</th>
<th>PostgreSQL</th>
<th>MySQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1097</td>
<td>1734</td>
<td>4409</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SF</th>
<th>MonetDB</th>
<th>PostgreSQL</th>
<th>MySQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>558</td>
<td>366</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>39</td>
<td>141</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>18</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>21</td>
<td>223</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>44</td>
<td>156</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>31</td>
<td>59</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>35</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>124</td>
<td>75</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>20</td>
<td>132</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>21</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>61</td>
<td>74</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>51</td>
<td>112</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>43</td>
<td>301</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>67</td>
<td>152</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td>46</td>
<td>80</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>18</td>
<td>11</td>
<td>92</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>1810</td>
<td>11</td>
</tr>
<tr>
<td>21</td>
<td>26</td>
<td>199</td>
<td>27</td>
</tr>
<tr>
<td>22</td>
<td>8</td>
<td>420</td>
<td>13</td>
</tr>
</tbody>
</table>

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

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>
</tr>
</thead>
<tbody>
<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>9</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>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**
**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>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1097</td>
<td>1734</td>
<td>4409</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
<td>558</td>
<td>366</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>39</td>
<td>141</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>18</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>21</td>
<td>223</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>44</td>
<td>156</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>31</td>
<td>59</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>35</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>124</td>
<td>75</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>20</td>
<td>132</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>21</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>61</td>
<td>74</td>
</tr>
<tr>
<td>13</td>
<td>47</td>
<td>51</td>
<td>112</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>43</td>
<td>301</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>67</td>
<td>152</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td>46</td>
<td>80</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>18</td>
<td>11</td>
<td>92</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>64</td>
<td>1060</td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>1810</td>
<td>11</td>
</tr>
<tr>
<td>21</td>
<td>26</td>
<td>199</td>
<td>27</td>
</tr>
<tr>
<td>22</td>
<td>8</td>
<td>420</td>
<td>13</td>
</tr>
</tbody>
</table>

**ATHLON X2 3800+ (2000mhz) 2 disks in raid 0, 2G main memory**
<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>
<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>
<td>793</td>
<td>3570</td>
<td>1233</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>
<td>74</td>
<td>293</td>
<td>201</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>
<td>88</td>
<td>25</td>
<td>1028</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>
<td>29</td>
<td>79</td>
<td>421</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>3</td>
<td>17</td>
<td>12</td>
<td>21</td>
<td>13</td>
<td>112</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>44</td>
<td>156</td>
<td>867</td>
<td>758</td>
<td>19931</td>
<td>3</td>
<td>2</td>
<td>5755</td>
<td>425</td>
<td>13</td>
<td>&gt; 1 hour</td>
</tr>
<tr>
<td>6</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>
<td>47</td>
<td>99</td>
<td>&gt; 1 hour</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>30</td>
<td>30</td>
<td>353</td>
<td>17101</td>
<td>2940</td>
<td>121</td>
<td>32</td>
<td>504</td>
<td>58</td>
<td>82</td>
<td>2818</td>
</tr>
<tr>
<td>8</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>
<td>85</td>
<td>538</td>
<td>&gt; 1 hour</td>
</tr>
<tr>
<td>9</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>
<td>34</td>
<td>108</td>
<td>344</td>
</tr>
<tr>
<td>10</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>
<td>56</td>
<td>24</td>
<td>504</td>
</tr>
<tr>
<td>11</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>
<td>19</td>
<td>77</td>
<td>60</td>
</tr>
<tr>
<td>12</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>
<td>40</td>
<td>61</td>
<td>802</td>
</tr>
<tr>
<td>13</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>
<td>20</td>
<td>73</td>
<td>&gt; 1 hour</td>
</tr>
<tr>
<td>14</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>
<td>5</td>
<td>60</td>
<td>113</td>
</tr>
<tr>
<td>15</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>
<td>26</td>
<td>36</td>
<td>49</td>
</tr>
<tr>
<td>16</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>
<td>402</td>
<td>&gt; 1 hour</td>
<td>185</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>92</td>
<td>18</td>
<td>698</td>
<td>&gt; 1 hour</td>
<td>35273</td>
<td>13</td>
<td>125</td>
<td>&gt; 1 hour</td>
<td>21</td>
<td>69</td>
<td>&gt;&gt; 1 hour</td>
</tr>
<tr>
<td>18</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>
<td>14</td>
<td>571</td>
<td>&gt; 1 hour</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>
<td>1</td>
<td>&gt; 1 hour</td>
<td>162</td>
<td>17</td>
<td>&gt; 1 hour</td>
<td>489</td>
</tr>
<tr>
<td>20</td>
<td>21</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>
<td>188</td>
<td>302</td>
<td>&gt; 1 hour</td>
</tr>
<tr>
<td>21</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>
<td>84</td>
<td>&gt; 1 hour</td>
<td>4</td>
</tr>
</tbody>
</table>

ATHLON X2 3800+ (2000mhz) 2 disks in raid 0, 2G main memory
<table>
<thead>
<tr>
<th>Q</th>
<th>MonetDB</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1:02:42</td>
<td>0:53:22</td>
</tr>
<tr>
<td>Q2</td>
<td>0:01:20</td>
<td>0:15:17</td>
</tr>
<tr>
<td>Q3</td>
<td>0:09:53</td>
<td>3:39:38</td>
</tr>
<tr>
<td>Q4</td>
<td>0:05:06</td>
<td>0:02:29</td>
</tr>
<tr>
<td>Q5</td>
<td>0:16:43</td>
<td>0:02:28</td>
</tr>
<tr>
<td>Q6</td>
<td>0:06:15</td>
<td>0:10:14</td>
</tr>
<tr>
<td>Q7</td>
<td>0:14:15</td>
<td>2:22:09</td>
</tr>
<tr>
<td>Q8</td>
<td>0:08:04</td>
<td>6:15:27</td>
</tr>
<tr>
<td>Q9</td>
<td>0:31:45</td>
<td></td>
</tr>
<tr>
<td>Q10</td>
<td>0:41:49</td>
<td>0:03:18</td>
</tr>
<tr>
<td>Q11</td>
<td>0:00:36</td>
<td>0:08:02</td>
</tr>
<tr>
<td>Q12</td>
<td>0:08:39</td>
<td>0:10:13</td>
</tr>
<tr>
<td>Q13</td>
<td>0:36:36</td>
<td>1:20:03</td>
</tr>
<tr>
<td>Q14</td>
<td>0:09:31</td>
<td>0:10:53</td>
</tr>
<tr>
<td>Q15</td>
<td>0:04:43</td>
<td>0:10:16</td>
</tr>
<tr>
<td>Q16</td>
<td>0:37:14</td>
<td>0:29:30</td>
</tr>
<tr>
<td>Q17</td>
<td>0:04:47</td>
<td></td>
</tr>
<tr>
<td>Q18</td>
<td>0:37:37</td>
<td>0:34:34</td>
</tr>
<tr>
<td>Q19</td>
<td>0:03:55</td>
<td>0:38:19</td>
</tr>
<tr>
<td>Q20</td>
<td>0:05:35</td>
<td></td>
</tr>
<tr>
<td>Q21</td>
<td>0:24:15</td>
<td>1:08:10</td>
</tr>
<tr>
<td>Q22</td>
<td>0:01:25</td>
<td></td>
</tr>
</tbody>
</table>

MonetDB is somewhat slower
MonetDB is >10x faster
Takes >10hr to run
Error, empty result
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
SkyServer Schema

- 446 columns
- >585 million rows

- 6 columns
- >20 Billion rows
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
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

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

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

<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>
Recycler

instruction subsumption

```
Y3 := algebra.select(X1,20,45);
...
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
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

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
Each query is treated as an advice on how data should be stored. Updates become pending updates and are applied on demand. The goal is to minimize physical actions.

Pieces are ordered, but values in a piece are not ordered. The new value is 11.

- **Piece 1:** \( A \leq 10 \)
- **Piece 2:** \( 10 < A < 14 \)
- **Piece 3:** \( 14 \leq A \)

The more we crack, the more we learn, the less we touch.
“Self-organizing tuple reconstruction in column-stores“, Idreos, Manegold, Kersten, SIGMOD’09

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

MonetDB with sideways cracking

Presorted MonetDB
Preparation cost 3-14 minutes

No a priori knowledge needed
No idle time needed
Works for random workloads
Works for frequent updates
Ongoing Research

- DataCell:
  - efficient stream processing using DBMS technology
- DataCyclotron:
  - “let the data (hot set) spin through the network”
- Adaptive Indexing:
  - Beyond database cracking
- SciLens: http://www.scilens.org/
  - Database support for science & data intensive research
  - SciQL: Array support in SQL & DBMS
  - SciBORQ: Biased sampling
    - Scientific data management with Bounds On Runtime and Quality
- Runtime query optimization
- JIT query plan compilation
- Exploiting GPGPUs  --- memory speed, rather than compute power
- More parallelization & distribution
"The speed at which any given scientific discipline advances will depend on how researchers collaborate with one another, and with technologists, in areas of eScience such as databases..."

The Fourth Paradigm
Our new Playground & Challenge

- **128 Pebbles**: dual-core AMD bobcat, 8GB RAM, 10 TB HDD
- **128 bricks**: 8-core i7, 16GB RAM, 1 TB SSD, 2 TB HDD
- **1024 cores, 2 TB RAM, 128 TB SSD, 256 TB HDD**
- **256 cores, 1 TB RAM, 1.3 PB HDD**
- **1 Diamond**: 64 core CPU, 1TB
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
Usage

- **CWI, Amsterdam:**
  - Core DBMS Research
  - TIJAH: Multi-Media IR
  - Data Mining, GIS, Astronomy, RDF/SPARQL, Streams, ...

- **Data Distilleries, Amsterdam (CWI Spin-Off, founded in 1995; sold to SPSS in 2004; now IBM ;-) ):**
  - Commercial Data-Mining & CRM Software
  - Many banks & insurance companies in NL

- **Knowledge Discovery Lab, UMass, Amherst:**
  - Proximity: OpenSource relational knowledge discovery tool

- **Universität Tübingen:**
  - Pathfinder: XQuery compiler (with UTwente & CWI)
  - Ferry: Database-Supported Program Execution (LINQ, etc.)
Usage

- Dutch National Forensics Institute (NFI):
  - XIRAF: Digital Forensics
- University of Utrecht:
  - Data Mining & Bio-Informatics
- University of Twente:
  - Distributed Object-Oriented Data Management
  - Propabilistic databases
- Commercial:
  - Consulting, CRM, ...
  - Telco
- Open-Source Community:
  - OpenStreetMap
  - Active user community (mailing lists)
  - (many more “anonymously”)
- >10000 downloads per month
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