MonetDB: Open-source Columnar Database Technology
Beyond Textbooks

http://www.monetdb.org/

Stefan Manegold
Stefan.Manegold@cwi.nl
http://homepages.cwi.nl/~manegold/

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)
Workload changes: Transactions (OLTP) vs OLAP, BI, Data Mining, ...

Why?


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

Databases hit The Memory Wall

Hardware Changes: The Memory Wall

Trip to memory = 1000s of instructions!
Why?

Hardware Changes: Memory Hierarchies

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

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 & 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"), ...
Column-Store
formerly know as
Decomposed Storage Model

- 1985: DSM (Copeland et al.; SIGMOD 1985)
- 1992: First ideas and kernel for MonetDB (Kersten)
- 1993: MonetDB is born
- 1993: KDB (first commercial DSM system (?)
- 1995: Sybase IQ
- 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 (now called Actian) acquires VectorWise
- 2011: HP acquires Vertica
- 201?: SAP HANA, IBM BLINK -> ISAO -> BLU, Oracle Database In-Memory
Microsoft SQL Server Column-store indexes (“Apollo”), ...

How?

SIGMOD 1985

2.1 Support Of Multivalued Attributes

- A more comprehensive data model than normalized relations might allow multivalued

2.2 Support of Entities

- A more comprehensive data model than the original relational model might support the notion

2.3 Support of Multiple Parent Relations

- A data model with more generality than relations might allow multiple parent relations, where a single record can have more than one parent

2.4 Support of Heterogeneous Records

- A data model with more generality than relations might allow heterogeneous records, where records of a single relation can have different

2.5 Support of Directed Graphs

- A data model with more generality than relations might allow a directed graph structure.

Storing Relations in MonetDB

DSM => Column-store

Front-End

logical data model

mapping rules

physical data model (BATs)

Monet
Virtual OID: seqbase=1000 (increment=1)
RISC Relational Algebra

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

Zero cost

```
select(30/nb)
```

VIEWS
(not materialized)

RISC Relational Algebra

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

Simple, hard-coded semantics in operators

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

MATERIALIZED intermediate results

RISC Relational Algebra

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

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
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 to BATs and query languages to MAL

- 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

The MonetDB Software Stack

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

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

Kernel
- MonetDB kernel
The MonetDB Software Stack

Front-ends
- XQuery
- SQL 03
- Optimizers

Back-ends
- MonetDB 4
- MonetDB 5
- Kernel

Kernel
- MonetDB kernel

Strategic optimization
- Rel. Alg. -> MAL

Tactical optimization
- MAL -> MAL rewrites

Runtime operational optimization

RISC Relational Algebra

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

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.thetauselect(_02,10,"<");
    _10:bat[:oid,:void] := algebra.markT(_07,0@0);
    _11:bat[:void,:oid] := bat.reverse(_10);
    _12:bat[:oid,:int] := sql.bind("sys","t","a",0);
    _14:bat[:void,:int] := algebra.leftjoin(_11,_12);
exit _55;
_15 := sql.resultSet(1,1,_14);
sql.rsColumn(_15,"sys.t","a","int",32,0,_14);
_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 ]

Multi-core Parallelism: Mitosis

- Horizontally slice largest table
- As many slices as CPU cores
- As many slices such that #cores slices fit in memory
- Replicate query plan per slice
- As far as possible
- Evaluate replicated plan fragments concurrently
- Concatenate partial intermediate result
- Evaluate remaining query plan

EXPLAIN SELECT a FROM t WHERE c < 10;

function user.s1_1():void;
  barrier _55 := language.dataflow();
  _02a:bat[void:int] := sql.bind("sys","t","c",0,0,2);
  _07a:bat[void:int] := algebra.thetauselect(_02a,10,"<");
  _10a:bat[void:oid] := algebra.markT(_07a,0@0);
  _11a:bat[void:oid] := bat.reverse(_10a);
  _12a:bat[void:int] := sql.bind("sys","t","a",0,0,2);
  _14a:bat[void:int] := algebra.leftjoin(_11a,_12a);
  _02b:bat[void:int] := sql.bind("sys","t","c",0,1,2);
  _07b:bat[void:int] := algebra.thetauselect(_02b,10,"<");
  _10b:bat[void:oid] := algebra.markT(_07b,0@0);
  _11b:bat[void:oid] := bat.reverse(_10b);
  _12b:bat[void:int] := sql.bind("sys","t","a",0,1,2);
  _14b:bat[void:int] := algebra.leftjoin(_11b,_12b);
exit _55;
  _14 := mat.pack(_14a,_14b);
  _15 := sql.resultSet(1,1,_14);
sql.rsColumn(_15,"sys.t","a","int",32,0,_14);
  _21 := io.stdout();
sql.exportResult(_21,_15);
end s1_1;

Multi-core Parallelism: Mitosis
EXPLAIN SELECT a FROM t WHERE c < 10;

function user.s1_1():void;
barrier _55 := language.dataflow();
    ...    _21 := io.stdout();
    sql.exportResult(_21,_15);
end s1_1;

Multi-core Parallelism: Mitosis

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

MonetDB vs Traditional DBMS Architecture

- Architecture-Conscious Query Processing
  - vs Magnetic disk I/O conscious processing
- 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