Mammals Flourished Long Before Dinosaurs Became Extinct

VLDB 2009 Lyon - Ten Year Award
“Database Architecture Optimized For The New Bottleneck: Memory Access” (VLDB 1999)

Stefan Manegold (manegold@cwi.nl)
Peter Boncz (boncz@cwi.nl)
Martin Kersten (mk@cwi.nl)
MICHAEL STONEBRAKER… What I see happening is that the large database vendors, whom I'll call the elephants, are selling a one-size-fits-all, 30-year-old architecture, sold somewhere in the late 1970s...
Reptiles

200 MYA

Dinosaur

Mammals
Reptiles

Mammals

Dinosaur

200 MYA
Reptiles
200 MYA
Dinosaur
Mammals
Reptiles

Mammals

Dinosaurs

200 MYA
Reptiles

Mammals

200 MYA

Dinosaurs

60 MYA
Large mammals once dined on dinosaurs

Repenomamus giganticus

Repenomamus robustus

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)
The genes of a species

- Reptiles
  - Mammals
    - SQL86, SQL92, SQL99, SQL03
    - n-ary storage scheme
    - relational algebra + DDL
    - 5+ way indexing schemes
    - slotted pages of records
    - Volcano-style computation

- Dinosaur
The evolution of the Fox

1979-1985

Troll a relational engine to simplify relational database programming

SWI Prolog made a much better relational engine then my first system and Ingres, Oracle…
The evolution of the Fox

1979-1985
Troll a relational engine to simplify relational database programming

SWI Prolog made a much better relational engine then my first system and Ingres, Oracle…


The evolution of the Fox

1979-1985
Troll a relational engine to simplify relational database programming

Non-first-normal-form disease
Object-orientation religion
IO pages size increase
The evolution of the Fox

1979-1985
  Troll a relational engine to simplify relational database programming

Non-first-normal-form disease
Object-orientation religion
IO pages size increase
The evolution of the Fox

1979-1985
Troll a relational engine to simplify relational database programming

Non-first-normal-form disease
Object-orientation religion
IO pages size increase

512 bytes
The evolution of the Fox

1979-1985
Troll a relational engine to simplify relational database programming

Non-first-normal-form disease
Object-orientation religion
IO pages size increase
Albert Einstein

“We can't solve problems by using the same kind of thinking we used when we created them.”
A DECOMPOSITION STORAGE MODEL

SIGMOD 1985

21 Support Of Multivalued Attributes

A more comprehensive data model than normalized relations might allow multivalued attributes.

22 Support Of Entities

A more comprehensive data model than the original relational model might support the notion of entities.

23 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.

24 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 types.

25 Support Of Directed Graphs

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

The DSM offers simplicity - Simple systems

have several major advantages over complex systems. One advantage is that a set of fewer and simpler

functions, given fixed development resources, can be either further tuned in software or pushed

further into hardware to improve performance. This is similar to the RISC (Patterson and Ditzel 1980)

approach in general purpose architectures. A second advantage is that many alternative cases

with different processing strategies can less often be exploited, since the cases are not always

recognized.

Studies have compared the performance of transposed storage models with the DSM (Hoffer 1976, Batary 1976, March and Severance 1977, March and Scudder 1984). In this report, we describe the advantages of a fully decomposed storage model (DSM), which is a transposed storage model with surrogates included. The DSM pairs each attribute value with the surrogate of its conceptual schema record in a binary relation. For example, the above relation would be stored as

```
<table>
<thead>
<tr>
<th>a1</th>
<th>sur</th>
<th>va1</th>
<th>a2</th>
<th>sur</th>
<th>va1</th>
<th>a3</th>
<th>sur</th>
<th>va1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>v11</td>
<td></td>
<td>s1</td>
<td>v21</td>
<td></td>
<td>s1</td>
<td>v31</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>v12</td>
<td></td>
<td>s2</td>
<td>v22</td>
<td></td>
<td>s2</td>
<td>v32</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>v13</td>
<td></td>
<td>s3</td>
<td>v23</td>
<td></td>
<td>s3</td>
<td>v33</td>
<td></td>
</tr>
</tbody>
</table>
```
The genes of a new species

- SQL86, SQL92, SQL99, SQL03
- n-ary storage scheme
- relational algebra + DDL
- 5+ way indexing schemes
- slotted pages of records
- Volcano-style computation
The genes of a new species

- SQL86, SQL92, SQL99, SQL03
- Binary Association Tables
  - storage scheme
  - relational algebra + DDL
- Self managing Arrays
  - indexing schemes
  - of records
- Materialize operator
  - computation
The MonetDB Software Stack

- MonetDB kernel
- MonetDB 5
- Optimizers
- SQL 03
Cache-Conscious Query Processing in

Stefan Manegold (manegold@cwi.nl)
Peter Boncz (boncz@cwi.nl)
Martin Kersten (mk@cwi.nl)
Simple Scan: select max(c) from t
Evolution == Progress?

Hardware Evolution (Moore's Law)

BUT Software Stagnation!
Databases hit The Memory Wall

- Detailed and exhaustive analysis for different workloads using 4 RDBMSs by Anastassia Ailamaki et al. in “DBMSs On A Modern Processor: Where Does Time Go?” (VLDB 1999)

- CPU is 50%-90% idle, waiting for memory:
  - L1 data stalls
  - L1 instruction stalls
  - L2 data stalls
  - TLB stalls
  - Branch mispredictions
  - Resource stalls
2009:
- L2 (+L3) on CPU die
- Memory access: up to 1000 cycles
Required DBMS Evolution

- Memory access has become a significant cost factor
- Database algorithms suffer particularly from latency (due to random access patterns)

<table>
<thead>
<tr>
<th>Goal</th>
<th>Optimize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use cache lines fully</td>
<td>⇒ Data structures</td>
</tr>
<tr>
<td>Prevent cache &amp; TLB misses</td>
<td>⇒ Memory access / algorithms</td>
</tr>
<tr>
<td>Prevent CPU stalls</td>
<td>⇒ Implementation techniques</td>
</tr>
<tr>
<td>Exploit CPU-inherent parallelism</td>
<td>⇒ Implementation techniques</td>
</tr>
</tbody>
</table>
### Data Structure Evolution

**Row-storage**
- wastes bandwidth

**Column-storage**
- exploits full bandwidth

<table>
<thead>
<tr>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>...</th>
<th>A_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>...</td>
<td>●</td>
</tr>
</tbody>
</table>

**Requested attribute**

**Cache line**
Algorithm Evolution: Joins

- Nested-loop:
  + sequential access to both inner & outer input
  -- quadratic complexity

- Sort-merge:
  + single sequential scan during merge ("benefit")
  -- random access during sort ("investment")

- Hash-join:
  + sequential scan over both inputs
  -- random access to hash table (build & probe)
Algorithm Evolution: Partitioned Hash-Joins

**Phase 1:**
- Cluster both input relations
- Create clusters that fit in CPU cache
- Restrict random data access to (smallest) cache
- Avoid cache capacity misses

**Phase 2:**
- Join matching clusters
Partitioned Hash-Join: Joining (Phase 2)

=> Phase 2 solved; but what about Phase 1?

Points measured
Lines modeled [VLDB2002]
**Problem:**
- Number of clusters exceeds number of cache lines / TLB entries
- => cache / TLB thrashing

**Solution:**
- Multi-pass clustering
Algorithm Evolution: Multi-Pass Clustering

- Limit number of clusters per pass
- Avoid cache / TLB thrashing
- Trade memory cost for CPU cost
Partitioned Hash-Join: Multi-Pass Clustering

elapsed time [seconds]

Number of clusters

passes:
4  -
3  -
2  -
1  -

[Graph showing elapsed time for different numbers of passes and clusters]
Partitioned Hash-Join

The diagram shows the performance of a hash-join algorithm under different cluster sizes and number of passes. The x-axis represents the cluster size in bytes, and the y-axis represents the execution time in seconds. The graph compares default, optimized, and different pass configurations.

- **Default** (red line) shows the baseline performance.
- **Optimized** (blue line) demonstrates improved performance with optimizations.
- **1 pass** (plus symbols) indicates single-pass execution.
- **2 passes** (square symbols) shows double-pass performance.
- **3 passes** (circle symbols) and **4 passes** (triangle symbols) show performance with additional passes.

As the cluster size increases, the execution time decreases for both default and optimized configurations, indicating better performance with larger clusters. The number of passes also affects the execution time, with more passes generally increasing the time required for processing.
Joins in Column-Stores: Handling Payload

Problem:
- Join result: pairs of tuple IDs; *Out-of order*
- => random access during projection / tuple-reconstruction

Solutions:
- Jive-Join (Li, Ross; VLDB-Journal 1998)
- Flash-Join (Tsirogiannis, Harizopoulos, Shah, Wiener, Graefe; SIGMOD 2009)
- Radix-Decluster (Boncz, Manegold, Kersten; VLDB 2004)
- (Sideways Cracking (Idreos, Kersten, Manegold; SIGMOD 2009))

=> post-projection / late materialization
Algorithm Evolution: Multi-pass Clustering

1 pass  P passes  P passes, CPU optimized
Algorithm Evolution: Partitioned Hash-Join

CPU optimized
Cost Model Evolution: Data Access

- Total data access cost is sum over all cache/memory levels.
- Cost per level is number of cache misses scored by latency.
- Simple tool to measure latency per cache level ("The Calibrator").
- Few simple basic access patterns "sequential", "random", ...
- Compound access patterns: combinations of basic access patterns.
- Basic cost functions: estimate number of cache misses of basic access patterns.
- Rules how to create compound cost functions using basic cost functions.
- Describe data access of algorithms using access patterns.
The Bigger Picture:
Evolving Columnar Database Architecture

Stefan Manegold (manegold@cwi.nl)
Peter Boncz (boncz@cwi.nl)
Martin Kersten (mk@cwi.nl)
RISC Relational Algebra

**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

```c
for(i=0; i<n; i++)
    res[i] = col[i] - val;
```

Simple, hard-coded semantics in operators

**MATERIALIZED intermediate results**
Materialization vs Pipelining

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

**Materialized intermediate results**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>age</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>ivan</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>alice</td>
<td>22</td>
<td>TRUE</td>
</tr>
<tr>
<td>102</td>
<td>ivan</td>
<td>37</td>
<td>350</td>
</tr>
</tbody>
</table>
Materialization vs Pipelining
Observations:

- called much less often
- more time spent in primitives
- less overhead

Loops with:

- "Vectorized In Cache Processing"
- 100 CPUs cache resident
- for(i=0; i<n; i++) res[i] = (col[i] > x)
- for(i=0; i<n; i++) res[i] = (col[i] - x)
- for(i=0; i<n; i++) res[i] = (col[i] * x)
Vectors start to exceed the CPU cache, causing additional memory traffic and less iterator.next() primitive function calls ("interpretation overhead")
And much more..

14:00-17:00 Tutorial “Column-Oriented DB Systems”
+ Daniel Abadi (Yale) and Stavros Harizopoulos (HP Labs)
MonetDB Highlights

- **Architecture-Conscious Query Processing**
  - Data layout, algorithms, cost models
- **Multi-Model: ODMG, SQL, XQuery, .. SPARQL**
  - Columns as the building block for complex data structures
- **RISC Relational Algebra (vs CISC)**
  - Faster through simplicity: no tuple expression interpreter
- **Decoupling of Transactions from Execution/Buffering**
  - ACID, but not ARIES.. Pay as you need transaction overhead.
  - differential, lazy, optimistic, snapshot
- **Run-Time Indexing and Query Optimization**
  - Extensible Optimizer Framework
  - cracking, recycling, sampling-based runtime optimization
MonetDB vs Traditional Architecture

- Architecture-Conscious Query Processing
  - vs Magnetic disk I/O conscious processing
- Multi-Model: ODMG, SQL, XQuery, .. SPARQL
  - vs Relational with Bolt-on Subsystems
- RISC Relational Algebra
  - vs Tuple-at-a-time Iterator Model
- Decoupling of Transactions from Execution/Buffering
  - vs ARIES integrated into Execution/Buffering/Indexing
- Run-Time Indexing and Query Optimization
  - vs Static DBA/Workload-driven Optimization & Indexing
The **MonetDB** Software Stack

- **SQL 03**
- **Optimizers**
- **MonetDB 5**
- **MonetDB kernel**

- Orthogonal extension of SQL03
- Clear computational semantics
- Minimal extension to MonetDB
The MonetDB Software Stack

- XQuery
- SQL 03
- RDF
- Arrays
- Optimizers
- MonetDB 4
- MonetDB 5
- OGIS
- X100
- MonetDB kernel

compile
The MonetDB Software Stack

- Extensible query language frontend framework
- Extensible Dynamic Runtime QOPT Framework!
- SQL 03
- Optimizers
- MonetDB 4
- MonetDB 5
- RDF
- Arrays
- Extensible Architecture-Conscious Execution platform
Farming new species
Acknowledgements

Martin Kersten
Peter Boncz
Niels Nes
Stefan Manegold
Fabian Groffen
Sjoerd Mullender
Steffen Goeldner
Arjen de Vries
Menzo Windhouwer
Tim Ruhl
Romulo Goncalves

Alex van Ballegooij
Johan List
Georgina Ramirez
Marcin Zukowski
Roberto Cornacchia
Sandor Heman
Torsten Grust
Jens Teubner
Maurice van Keulen
Jan Flokstra
Milena Ivanova
Lefteris Sidirourgos

Jan Rittinger
Wouter Alink
Jennie Zhang
Stratos Idreos
Erietta Liarou
Lefteris Sidirourgos
Florian Waas
Albrecht Schmidt
Jonas Karlsson
Maurice van Keulen
Martin van Dinther
Peter Bosch
Carel van den Berg
Wilco Quak
Whoa MonetDB! Speed lines!