CedarDB

Overview

● “PostgreSQL for analytics”
  ○ Full-featured versatile database
  ○ Simultaneous high-performance analytics and operations on the same data
  ○ Several orders of magnitude speedup over existing systems
  ○ Full utilization of modern hardware capabilities (e.g. massive parallelism, RAM capacity)
  ○ Transparently and gracefully scales beyond main memory

● Started at TUM
  ○ 5 PhDs, developed the system over the last ~6 years
  ○ Bring the most efficient data processing engine to the world
Who uses a database system daily?
Who uses a database system daily?
SQL or NoSQL?
Database Systems

- Who uses a database system daily?
- SQL or NoSQL?

**Ranking scores per category in percent, February 2024**

- Relational DBMS 71.9%
- Document stores 10.3%
- Graph DBMS 1.7%
- Key-value stores 5.4%
- Multivalue DBMS 0.2%
- Time Series DBMS 1.2%
- Spatial DBMS 0.5%
- Vector DBMS 0.3%
- Native XML DBMS 0.3%
- Object oriented DBMS 0.2%
- Search engines 4.5%
- Wide column stores 2.9%
- RDF stores 0.4%
CedarDB

**SQL / Relational**
- Old

**NoSQL**
- Not so old
CedarDB

SQL / Relational

- Old
- Ancient

NoSQL

- Not so old
CedarDB

SQL / Relational

- Old
- Ancient

NoSQL

- Not so old
CedarDB

**SQL / Relational**
- Old
- Ancient

**NoSQL**
- Not so old
- High scalability
- Flexible data formats
- Simple key/value storage
<table>
<thead>
<tr>
<th>CedarDB</th>
<th>SQL / Relational</th>
<th>NoSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>● Old</td>
<td>● Not so old</td>
</tr>
<tr>
<td></td>
<td>● Ancient</td>
<td>● High scalability</td>
</tr>
<tr>
<td></td>
<td>● But an efficient model to work with data</td>
<td>● Flexible data formats</td>
</tr>
<tr>
<td></td>
<td>● Decades of experience building data processing pipelines for data-driven applications</td>
<td>● Simple key/value storage</td>
</tr>
</tbody>
</table>
Case Study: MongoDB
Case Study: MongoDB

- New query engine in MongoDB 6 (great write-up on laplab.me)
- Previous MongoDB engine was document oriented

A couple of intermediate query stages, each creating a JSON document to pass to the next one. All documents are discarded except the last one on the right.
Case Study: MongoDB

- Slot-Based Query Execution Engine

```javascript
{
  "a": 43,
  "b": "hello world",
  "c": [1, 2, 3]
}
```
Case Study: MongoDB

- Slot-Based Query Execution Engine

```json
{
    "a": 43,
    "b": "hello world",
    "c": [1, 2, 3]
}
```

`open()`, `getNext()` and `close()` interface and can be used like this:

```java
stream.open();
while (stream.getNext()) {
    // Do something with the data stream provided.
}
stream.close();
```
Volcano—An Extensible and Parallel Query Evaluation System

Goetz Graefe

Abstract—To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. The Volcano effort provides a rich environment for research and education in database systems design, heuristics for query optimization, parallel query execution, and resource allocation.

Volcano uses a standard interface between algebra operators, allowing easy addition of new operators and operator implementations. Operations on individual items, e.g., predicates, are imported into the query processing operators using support functions. The semantics of support functions is not prescribed; any data type including complex objects and any operation can be realized. Thus, Volcano is extensible with new operators, algorithms, data types, and type-specific methods.

Volcano includes two novel meta-operators. The choose-plan meta-operator supports dynamic query evaluation plans that allow delaying selected optimization decisions until run-time, e.g., for embedded queries with free variables. The exchange meta-operator supports intra-operator parallelism on sorted systems as it lacks features such as a user-friendly query language, a type system for instances (record definitions), a query optimizer, and catalogs. Because of this focus, Volcano is able to serve as an experimental vehicle for a multitude of purposes, all of them open-ended, which results in a combination of requirements that have not been integrated in a single system before. First, it is modular and extensible to enable future research, e.g., on algorithms, data models, resource allocation, parallel execution, load balancing, and query optimization heuristics. Thus, Volcano provides an infrastructure for experimental research rather than a final research prototype in itself. Second, it is simple in its design to allow student use and research.

Modularity and simplicity are very important for this purpose because they allow students to begin working on projects without an understanding of the entire design and
Case Study: MongoDB

- Volcano model query processing

Volcano—An Extensible and Parallel Query Evaluation System
Goetz Graefe

Abstract—To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. The Volcano effort provides a rich environment for research and education in database systems design, heuristics for query optimization, parallel query execution, and resource allocation.

Volcano uses a standard interface between algebra operators, allowing easy addition of new operators and operator implementations. Operations on individual items, e.g., predicates, are imported into the query processing operators using support functions. The semantics of support functions is not prescribed; any data type including complex objects and any operation can be realized. Thus, Volcano is extensible with new operators, algorithms, data types, and type-specific methods.

Volcano includes two novel meta-operators. The choose-plan meta-operator supports dynamic query evaluation plans that allow delaying selected optimization decisions until run-time, e.g., for embedded queries with free variables. The exchange meta-operator supports intra-operator parallelism on most
Volcano—An Extensible and Parallel Query Evaluation System

Goetz Graefe

Abstract—To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. The Volcano effort provides a rich environment for research and education in database systems design, heuristics for query optimization, parallel query execution, and resource allocation.

Volcano uses a standard interface between algebra operators, allowing easy addition of new operators and operator implementations. Operations on individual items, e.g., predicates, are imported into the query processing operators using support functions. The semantics of support functions is not prescribed; any data type including complex objects and any operation can be realized. Thus, Volcano is extensible with new operators, algorithms, data types, and type-specific methods.

Volcano includes two novel meta-operators. The choose-plan meta-operator supports dynamic query evaluation plans that allow delaying selected optimization decisions until run-time, e.g., for embedded queries with free variables. The exchange meta-operator supports intra-operator parallelism on rooted as it lacks features such as a user-friendly query language, a type system for instances (record definitions), a query optimizer, and catalogs. Because of this focus, Volcano is able to serve as an experimental vehicle for a multitude of purposes, all of them open-ended, which results in a combination of requirements that have not been integrated in a single system before. First, it is modular and extensible to enable future research, e.g., on algorithms, data models, resource allocation, parallel execution, load balancing, and query optimization heuristics. Thus, Volcano provides an infrastructure for experimental research rather than a final research prototype in itself. Second, it is simple in its design to allow student use and research. Modularity and simplicity are very important for this purpose because they allow students to begin working on projects without an understanding of the entire design and
State of the art

- **DBMS "X"**
  - MySQL 4.1
  - "tuple at a time"
  - "column at a time"
  - MonetDB/MIL
  - Query without selection
  - Vectors start to exceed CPU cache, causing extra memory traffic

- **Hand-Coded C Program**
  - "vector at a time"
  - Low interpretation overhead in-cache materialization

- **Time (seconds)**
  - 0.1, 0.22, 0.60, 1, 10, 100

- **Data Points**
  - 3.7
  - 2.4

- **Keynotes**
  - Interpretation dominates execution
  - Interpretation overhead decreases
  - Main-memory materialization overhead
CedarDB

State of the art

"tuple at a time"
DBMS "X"
MySQL 4.1

"column at a time"
MonetDB/MIL
main-memory materialization overhead

interpretation overhead decreases

DuckDB

"vector at a time"
MonetDB/X100
low interpretation overhead in-cache materialization

Hand-Coded C Program

query without selection

vectors start to exceed CPU cache, causing extra memory traffic

Time (seconds)
Code generation

- Query execution plan similar to hand-coded C
- Slots ≈ CPU registers

```python
for doc in docs:
    a = json::lookup(doc, "a")
    b = json::lookup(doc, "b")
    e = json::lookup(doc, "e")
    sum[b] += a
```
CedarDB

Data-centric code generation

- Build data pipelines as tight loops
- Keeps data in registers as long as possible

```python
for t in R1:
    HT1[t.a] = t

for t in R2:
    if t.y < 5:
        HT2[t.z] += t.y

for t in HT2:
    HT3[t.b] = t

for t in R3:
    if t.z > 3:
        if HT3[t.z]:
            if HT1[t.x]:
                print t
```
Expression compilation

%1 = zext i64 %int1;  Zero extend to 64 bit
%2 = zext i64 %int2;
%3 = rotr i64 %2, 32;  Rotate right
%v = or i64 %1, %3;  Combine int1 and int2
%5 = crc32 i64 6763793487589347598, %v;  First crc32
%6 = crc32 i64 4593845798347983834, %v;  Second crc32
%7 = rotr i64 %6, 32;  Shift second part
%8 = xor i64 %5, %7;  Combine hash parts
%hash = mul i64 %8, 11400714819323198485;  Mix parts

Efficient code generation for arbitrary expressions

- Example: Efficient hashing of two 32-bit columns
- Generalizes to arbitrary data types
- And to arbitrary number of columns
Roadmap

- ✅ Flexible data formats
- ✅ Efficient data pipelines
- ✅ Efficient execution with arbitrary complex expressions
- Scalability
CedarDB

Scalable execution

- Morsel-Driven Parallelism
  - Full intra-query parallelism
  - Self-adapting morsel sizes

- Scheduler
  - Adaptive scheduling of short- and long-running queries
  - Extensible task-based interface

- Adaptive Compilation
  - Re-compilation and re-optimization
  - Flying start: Directly emit x86
Scaling with remote storage

- Fast relations on external storage
  - Interleaved networking and processing
- Asynchronous networking
  - Download close to network bandwidth
  - Up to 80 Gbit/s from S3
- S3-optimized storage layout
  - Columnar format
  - Small materialized aggregates
Data Examples

- German federal register portal “Handelsregister”
- ~4GB JSON data from offeneregister.de
CedarDB

Data Examples

- German federal register portal “Handelsregister”
- ~4GB JSON data from offeneregister.de

```json
{
  "all_attributes": {
    "_registerArt": "HRB", "_registerNummer": "141703",
    "additional_data": {"AD": true, "CD": true, "DK": true, "HD": false, "SI": true, "UT": true, "VÖ": false},
    "federal_state": "Bavaria",
    "native_company_number": "München HRB 141703",
    "registered_office": "Garching", "registrar": "München",
    "company_number": "D2601V_HRB141703",
    "current_status": "currently registered",
    "jurisdiction_code": "de",
    "name": "UnternehmerTUM GmbH",
    "officers": [
      {
        "end_date": "2010-02-11",
        "name": "Bernward Doctor Jopen",
        "other_attributes": {
          "city": "Gräfelfing",
          "dismissed": true,
          "firstname": "Bernward",
        },
        "position": "Geschäftsführer",
        "type": "person"
      },
      {
        "end_date": "2018-01-22",
        "name": "Claudia Anke Frey",
        "other_attributes": {
          "city": "Neufahrn",
          "dismissed": true,
          "firstname": "Claudia Anke",
        },
        "position": "Prokurist",
        "type": "person"
      },
      {
        "name": "Andreas Doctor Liebl",
        "other_attributes": {
          "city": "Unterföhring",
          "firstname": "Andreas",
          "lastname": "Doctor Liebl",
        },
        "position": "Geschäftsführer",
        "type": "person"
      },
      {
        "name": "Claudia Anke Frey",
        "other_attributes": {
          "city": "Neufahrnb. Freising",
          "firstname": "Claudia Anke",
          "flag": "sole representation",
        },
        "position": "Geschäftsführer",
        "type": "person"
      },
      {
        "name": "Helmut Doctor Schönenberger",
        "other_attributes": {
          "city": "München",
          "firstname": "Helmut",
          "flag": "mit der Befugnis im Namen der Gesellschaft mit sich im eigenen Namen oder als Vertreter eines Dritten Rechtsgeschäfte abzuschließen",
        },
        "position": "Geschäftsführer",
        "type": "person"
      },
      {
        "name": "Stefan Drüssler",
        "other_attributes": {
          "city": "München",
          "firstname": "Stefan",
          "flag": "sole representation",
          "lastname": "Drüssler",
        },
        "position": "Geschäftsführer",
        "type": "person"
      }
    ],
    "registered_address": "Lichtenbergstr.", "retrieved_at": "2019-01-31T00:07:28Z"
}
```
Data Examples

- German federal register portal “Handelsregister”
- ~4GB JSON data from offeneregister.de
- Munich’s most wanted
- Who-knows-who of Munich
CedarDB

Six Degrees of Jan Marsalek

with execs_json as (select data->>'company_number' company_number, data->>'name' company_name,
    json_array_elements((data->>'officers')::json) officer_json
    from register_data where data->>'officers' is not null),
execs as (select company_number, company_name, officer_json->>'name' as name,
    officer_json->>'other_attributes'->>'city' city
    from execs_json),
marsalek as (select * from execs where name = 'Jan Marsalek' and city = 'München'),
marsalek_l1 as (select * from execs where company_number in
    (select company_number from marsalek)),
marsalek_l2 as (select * from execs o where exists
    (select * from marsalek_l1 m where o.name = m.name and o.city = m.city)),
marsalek_l3 as (select * from execs where company_number in
    (select company_number from marsalek_l2))
select distinct name from marsalek_l3 order by name;
### Six Degrees of Jan Marsalek

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CedarDB</td>
<td>1.5s</td>
</tr>
<tr>
<td>DuckDB</td>
<td>13s</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>15s</td>
</tr>
</tbody>
</table>

**Relational CedarDB**

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexander von Knoop</td>
</tr>
<tr>
<td>Amra Blume</td>
</tr>
<tr>
<td>Andrea Göres</td>
</tr>
<tr>
<td>Andreas Doctor Görg</td>
</tr>
<tr>
<td>Anne C. Signorino Gelo</td>
</tr>
<tr>
<td>Anne Matthias</td>
</tr>
<tr>
<td>Benjamin Aquilino</td>
</tr>
<tr>
<td>Bettina Funk</td>
</tr>
<tr>
<td>Brigitte Häuser-Axtner</td>
</tr>
<tr>
<td>Burkhard Ley</td>
</tr>
<tr>
<td>Carlos Häuser</td>
</tr>
<tr>
<td>Christian von Hammel-Bonten</td>
</tr>
<tr>
<td>Christian von von Hammel-Bonten</td>
</tr>
</tbody>
</table>
CedarDB

Try it now

- Full-featured versatile database
- Simultaneous high-performance analytics and operations on the same data

Dive deeper: 
[cedardb.com/docs](https://cedardb.com/docs)

Get in touch: 
[philipp@cedardb.com](mailto:philipp@cedardb.com)

docker pull pfent/umbra
CedarDB

Adaptive Compilation

● Multiple JIT Backends
  ○ **High Efficiency vs. Low Latency**
  ○ Modularized for different requirements & platforms

● Adaptive Query Execution
  ○ Problem: Selecting strategy upfront is hard
  ○ Solution: **Start quickly & upgrade later**
  ○ Robust decisions with runtime feedback
  ○ Worker threads don't idle during single-threaded LLVM optimisations
CedarDB

Scheduling

● Low latencies under high load
  ○ Compute burned on heavy queries
  ○ Finish light queries quickly
  ○ Example: 95% system load, 75% light + 25% heavy
    *Light queries almost not affected by load.*

● Adaptive morsel sizes
  ○ Fairness through normalized time slices
  ○ Simplifies adaptive compilation