



*One Engine, one Query Language.
Multiple Data Models.*



+



+



Hej, trevligt att träffas!

My name is Jan!

I am from Cologne, Germany

I work for ArangoDB Inc., a database vendor

There I work on ArangoDB, a native multi-model database

I am a developer and mostly do C++

Multi-model databases (with emphasis on ArangoDB)



Databases, a short history



The history of databases (in one slide)

Relational
197x - now

relational storage, SQL, transactions

NoSQL
200x - now

write scaling, high availability, low-latency operations
non-relational storage, no complex querying,
no transactions

NewSQL
201x - now

fusing relational elements with NoSQL elements,
write scaling and high availability
with complex querying and transactions

Now the database market is crowded


db-engines.com lists 343 database systems as of now:

[Ranking](#) > Complete Ranking RSS RSS Feed

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the [method](#) of calculating the scores.



trend chart

WTF?!?

343 systems in ranking, January 2019

We now got relational databases, key-value stores, document stores, time-series databases, graph databases, search engines, object databases, XML databases, RDF stores, ...

The choice is yours: polyglot persistence

- "use the right tool for the job"
- assumption: specialized products are better suited than generic products
- this translates to using a search engine for search tasks, a key-value store for fast primary key operations, graph databases for handling connected data, and relational databases for working with fixed schemas

Issues with polyglot persistence

- Needs custom scripts or application logic for shipping the data from one system to the other
- Potential atomicity and consistency issues across the different database systems (i.e. no ACID transactions possible)
- Potential data redundancy
- Requires learning, administering and maintaining multiple technologies in parallel

Multi-model database value propositions

- The main idea behind multi-model databases is to support different data models in the same database product
- For users this means increased flexibility, and less lock-in to one specific data model and its limitations
- This also removes the need for maintaining multiple datastores, plus the atomicity and consistency issues when syncing them
- The different data models in a multi-model database can ideally be easily combined in queries and even transactions

"Layered multi-model" is a trend

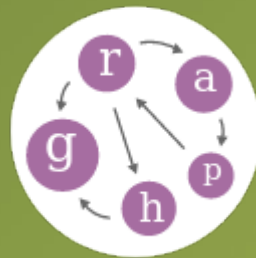
- Several databases now support different data models side-by-side
- Some traditional relational databases got extensions for key-value and document storage
- Many newer databases are very generic key-value stores under the hood, with the ability of "bolting on top" different data models

ArangoDB, the native multi-model database



At a glance

ArangoDB is a native multi-model database, with support for key-values, documents, graphs and search functionality



These data models are composable and can be combined via AQL, ArangoDB's unified query language

Databases, collections, documents

- On the highest level, ArangoDB organizes data in databases and collections
- Collections are used to store documents of similar types
- Documents can be considered JSON objects with arbitrary attributes, with optional nesting (sub-objects, sub-arrays)

Homogeneous documents

In the easiest case, documents in a collection are homogeneous (i.e. same attributes and types)

Example use case: product categories

```
{ "_key" : "books",      "title" : "Books" }  
{ "_key" : "cam",        "title" : "Camera products" }  
{ "_key" : "kitchen",    "title" : "Kitchen Appliances" }  
{ "_key" : "toys",        "title" : "Toys & Games" }  
{ "_key" : "video",       "title" : "Video games" }
```

Such data can be used much like data in a relational table

AQL queries – first examples

```
FOR c IN categories
  FILTER c._key == 'books'
  RETURN c
```

```
FOR c IN categories
  SORT c.title
  LIMIT 50, 10
  RETURN {
    _key: c._key,
    title: c.title
  }
```

Heterogeneous documents

In more complex cases, documents in the same collection are heterogeneous (i.e. different attributes or types)

This is often the case when objects have some common attributes, but there are different special attributes for sub-types

Collections in ArangoDB do not have a fixed schema, so it is possible to store somewhat-related objects in the same collection

Example use case: products

Heterogeneous documents

```
{
  "_key" : "A053720452",
  "category" : "books",
  "name" : "Harry Potter and the Cursed Child",
  "author" : "Joanne K. Rowling",
  "isbn" : "978-0-7515-6535-5", "published" : 2016
}
{
  "_key" : "ZB4061305X34",
  "category" : "toys",
  "name" : "Nerf N-Strike Elite Mega CycloneShock Blaster",
  "upc" : "630509278862",
  "colors" : [ "black", "red" ],
  "brand" : "HASBRO, INC."
}
```

Querying heterogeneous documents

```
FOR p IN products
  FILTER p.name == @search ||
        p.isbn == @search ||
        p.upc == @search
RETURN p
```

Indexes

- Documents can be accessed efficiently by their primary key, which is automatically indexed ("_key" attribute)
- Extra secondary indexes can be created as needed to support efficient querying by other attributes
- Index types: hash, skiplist, geo, fulltext

AQL – joins

```
FOR c IN categories
  FOR p IN products
    FILTER p.category == c._key
    SORT c.title, p.name
    RETURN {
      category: c
      product: p
    }
```

AQL – subqueries

```
FOR c IN categories
  RETURN {
    category: c.title,
    products: (
      FOR p IN products
        FILTER p.category == c._key
        SORT p.name
        RETURN p.name
    )
  }
```

AQL – aggregation

```
FOR s IN sales
  FILTER s.year >= 2017 && s.year <= 2019
  COLLECT   year      = s.year,
            month     = s.month,
            product   = s.product
  AGGREGATE count     = SUM(s.count),
            amount    = SUM(s.amount)
  RETURN {
    year, month, product, count, amount
  }
```

AQL – data modification

```
FOR p IN products
  FILTER !HAS(p, "lastModified")
  UPDATE p WITH {
    lastModified: DATE_ISO8601(DATE_NOW())
  } IN products
RETURN OLD._key
```

```
FOR p IN products
  FILTER p.category IN [ "cam", "video" ]
  REMOVE p IN products RETURN OLD
```

Graphs and traversals

- ArangoDB supports graph traversal queries
- Graph traversals can reveal which documents are directly or indirectly connected to which other documents via which connections
- Graph queries can be run on existing data
- Graphs are an optional feature – there is no need to model everything as a graph

Edges

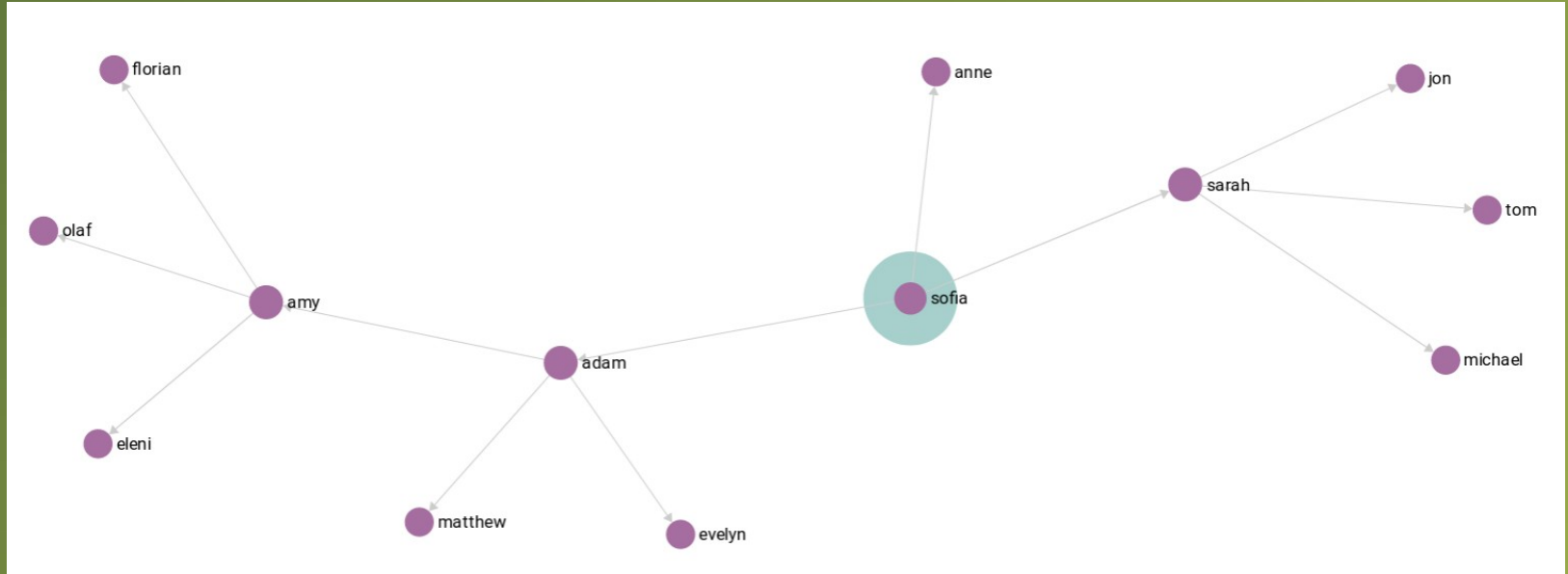
- Connections between documents are called "edges" and are stored in edge collections
- Edges have "`_from`" and "`_to`" attributes, which reference the connected vertices
- Edges are always directed (`_from -> _to`), but can also be queried in opposite order
- Edges are documents themselves and can carry arbitrary payload attributes

Edge collection example

Example use case: isManagerOf

```
{ "_from" : "employees/sofia", "_to" : "employees/adam" }  
{ "_from" : "employees/sofia", "_to" : "employees/sarah" }  
{ "_from" : "employees/sofia", "_to" : "employees/anne" }  
{ "_from" : "employees/sarah", "_to" : "employees/jon" }  
{ "_from" : "employees/sarah", "_to" : "employees/tom" }
```

Graph example, employees graph



AQL graph traversals, examples

```
FOR emp IN 1..1 OUTBOUND 'employees/sofia' isManagerOf  
  RETURN emp._key
```

```
FOR emp IN 1..1 INBOUND 'employees/olaf' isManagerOf  
  RETURN emp._key
```

```
FOR emp IN 1..2 OUTBOUND 'employees/sofia' isManagerOf  
  RETURN emp._key
```

AQL graph traversals, examples

```
FOR emp IN 2..2 ANY 'employees/olaf' isManagerOf  
  RETURN emp._key
```

```
FOR emp, edge, path IN 1..5 INBOUND 'employees/olaf'  
  isManagerOf  
  RETURN {  
    who: emp._key,  
    away: LENGTH(path.edges)  
  }
```

Advanced graph queries

- Edges can also be created to connect documents from different collections
- Many different collections together can form the graph
- While traversing the graph, it is also possible to filter on edge attributes and to not follow unwanted edges
- Graphs can contain cycles, so it is possible to stop searching already-visited documents and edges

ArangoDB Foxx, for exposing data via REST APIs



Foxx – built-in microservice framework

- ArangoDB comes with Foxx, a built-in framework for easily exposing dedicated database data via REST APIs
- Input parameter validation and API documentation included
- Combine your data with an API -> now you have a microservice!

Foxx example, custom API code

```
router.get("/managersOf/:employee", function (req, res) {
  let query = `
    FOR emp, edge, path IN 1..5 INBOUND
      CONCAT('employees/', @employee) isManagerOf
    RETURN { who: emp._key, away: LENGTH(path.edges) }'
  `;
  let result = db._query(query, {
    employee: req.params("employee")
  }).toArray();
  res.json(result);
})
.pathParam("employee", joi.string().required(), "employee name")
.summary("returns managers of the employee");
```

Foxx – use cases

- Data validation:
validate incoming data before storing it in the database
- Keeping things private:
no need to expose more data than necessary
no need to expose general-purpose querying API
- Complex permissioning:
implement permissions based on other database data or queries
- Eliminating network hops:
bundling multiple database queries in one server-side REST API

Deployment options



Deployment modes

- Single server
- Single servers with asynchronous replication (master/slave), manual failover
- Active failover: 2+ single servers with asynchronous replication and automatic failover
- Cluster: multiple query processing and storage nodes, write scaling and resilience

Officially supported platforms

- Linux, with 64 bit OS
- Windows 10, Windows 7 SP1, Windows server, with 64 bit OS
- Mac OS, Sierra and newer
- Kubernetes (including Google GKE, Amazon EKS, Pivotal PKS)
- Docker

ArangoDB clusters running on ARM64



Bonus features



Low-level operations

- For primary key CRUD operations there are also dedicated low-level interfaces
- These completely eliminate the overhead of query parsing and query optimization
- Performance improvement can be 2x to 3x

Administration via web interface

The screenshot displays the ArangoDB Enterprise Edition web interface. The top header bar is dark blue and contains the ArangoDB logo, the text "ENTERPRISE EDITION", and system status information: "USER: ROOT", "DB: _SYSTEM", and "HEALTH: GOOD". A sidebar on the left lists navigation options: DASHBOARD, COLLECTIONS (selected), VIEWS, QUERIES, GRAPHS, SERVICES, USERS, DATABASES, REPLICATION, LOGS, SUPPORT, and HELP US. The main content area shows a grid of collection tiles. Each tile includes an icon (a plus sign for 'Add Collection', a document icon for single collections, or a share icon for graph collections) and the collection name with a green 'loaded' status indicator. The collections shown are: acl, acl, pageViews, productRatings, products, purchases, ratings, relations, tags, and users. A search bar is located in the top right of the main area.

ArangoDB
ENTERPRISE EDITION

USER: ROOT DB: _SYSTEM HEALTH: GOOD

DASHBOARD

COLLECTIONS

VIEWS

QUERIES

GRAPHS

SERVICES

USERS

DATABASES

REPLICATION

LOGS

SUPPORT

HELP US

Search...

+ Add Collection

acl loaded

pageViews loaded

productRatings loaded

products loaded

purchases loaded

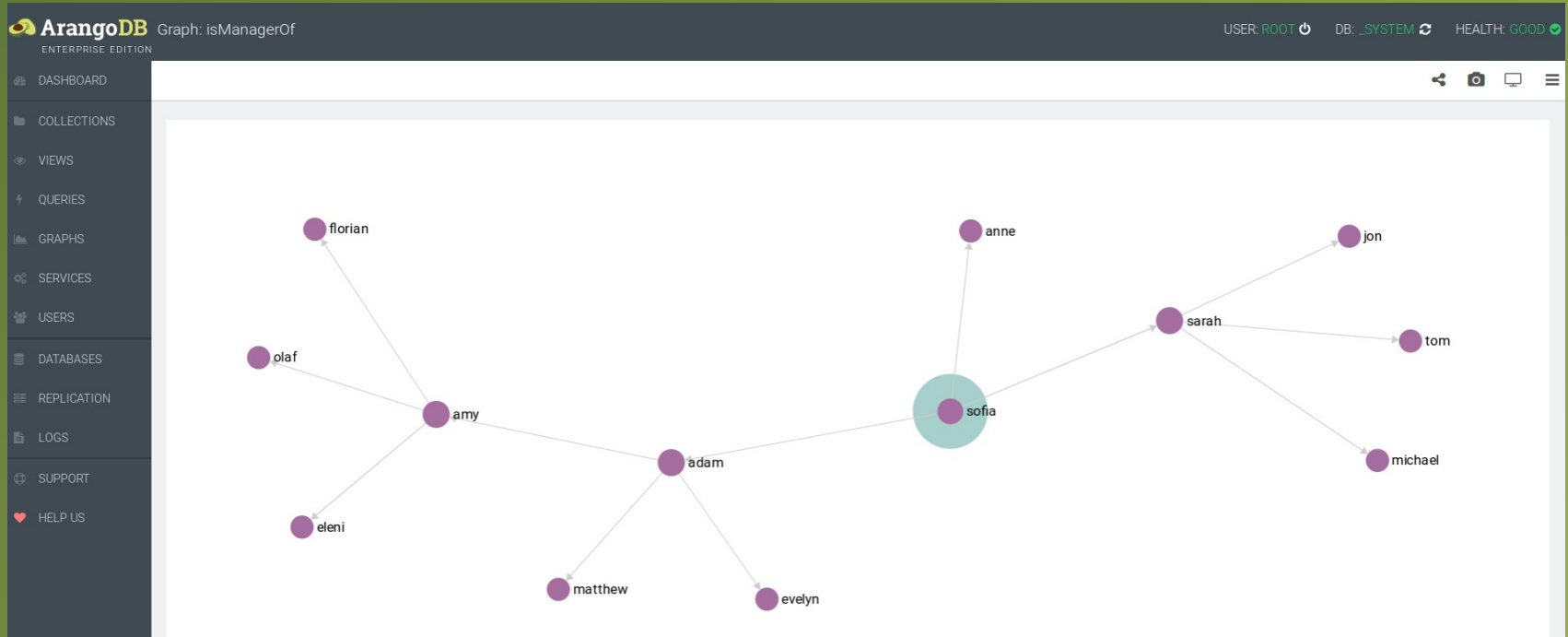
ratings loaded

relations loaded

tags loaded

users loaded

Built-in graph viewer



Bundled tools for automation

- ArangoDB comes with arangosh, a scriptable shell
- arangosh provides JavaScript database bindings for running quick tests, prototyping and automating tasks
- Release packages also provide tools for data import, backup and restore

Additional enterprise edition features

- "Smart graphs" (colocating of documents with all their outgoing edges on the same physical server)
- Datacenter-to-datacenter (DC2DC) replication
- Encryption at rest, encrypted backups
- LDAP integration for user management
- Support subscription with SLAs

Tack alla er!

Any questions?



Getting in touch

- Website: <https://www.arangodb.com/>
- Slack: <https://slack.arangodb.com/>
- Github: [arangodb/arangodb](https://github.com/arangodb/arangodb)
- Twitter: [@arangodb](https://twitter.com/arangodb)
- Stack Overflow: [#arangodb](https://stackoverflow.com/questions/tagged/arangodb)