Prof. DI Dr. Erich Gams

Einführung und Anwendung

# MongoDB

informationssysteme htl-wels

# Übersicht ☛ Was lernen wir?

- Eigenschaften
- Überblick
- Hands-on -> Tutorial

# Überblick Verbreitung DBMS

| Rang Sep 2021 | Rang Aug 2021 | Rang Sep 2020 | DBMS | Datenbankmodell |
|---|---|---|---|---|
| 1. | 1. | 1. | Oracle | Relational, Multi-Model |
| 2. | 2. | 2. | MySQL | Relational, Multi-Model |
| 3. | 3. | 3. | Microsoft SQL Server | Relational, Multi-Model |
| 4. | 4. | 4. | PostgreSQL | Relational, Multi-Model |
| 5. | 5. | 5. | MongoDB | Document, Multi-Model |
| 6. | 6. | ↑ 7. | Redis | Key-value, Multi-Model |
| 7. | 7. | ↓ 6. | IBM Db2 | Relational, Multi-Model |
| 8. | 8. | 8. | Elasticsearch | Suchmaschine, Multi-Model |
| 9. | 9. | 9. | SQLite | Relational |
| 10. | ↑ 11. | 10. | Cassandra | Wide column |

› (db-engines.com/de/ranking)

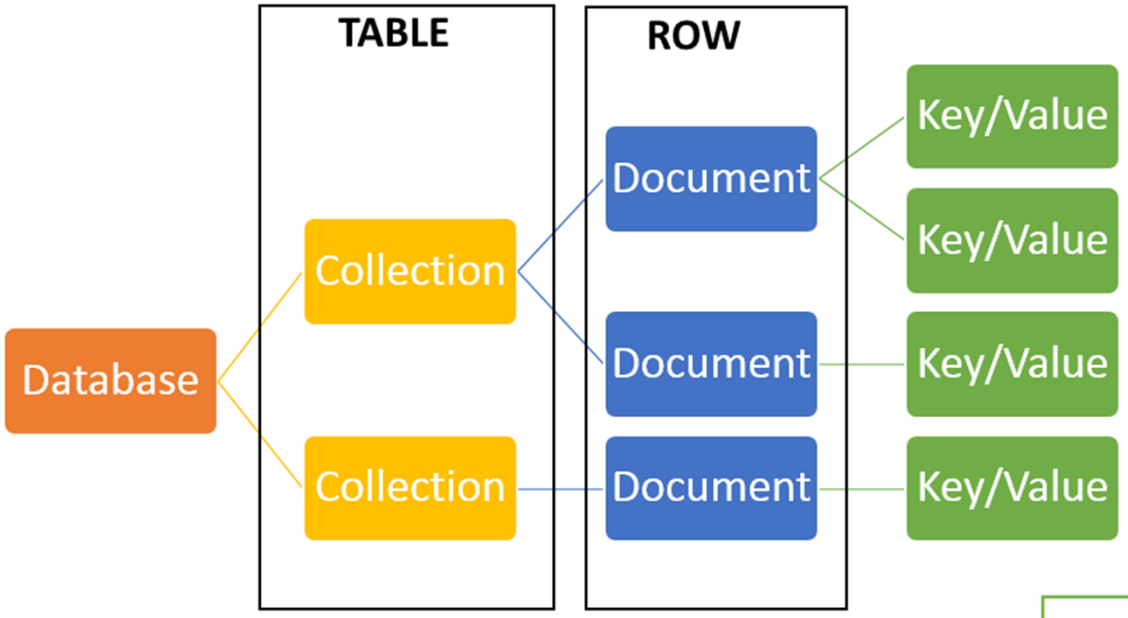
# Eigenschaften

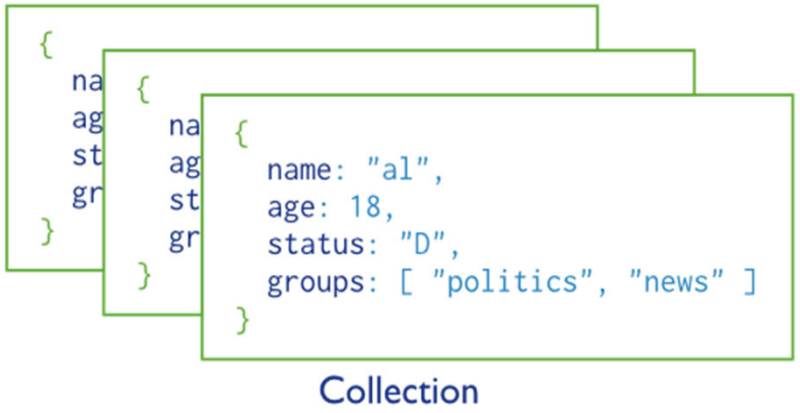
- **MongoDB** is an open-source **document store** database
  - “humongous” (riesig, enorm)
  - The leading NoSQL database
  - Schema free
  - Save JSON-style objects with dynamic schemas
- **Supports many features:**
  - Support for indices
  - Rich, document-based queries (CRUD operations)
  - Flexible aggregation and data processing
  - Map/Reduce support
  - Open Source (GNU AGPL v3.0.)

# MongoDB database structure

- A MongoDB instance can have many databases
- A database can have many collections
- A collection can have many documents

# MongoDB database structure



# Vergleich

| RDBMS | | MongoDB |
|---|---|---|
| Database | → | Database |
| Table, View | → → | Collection |
| Row | → → | Document (JSON, BSON) |
| Column | | Field |
| Index | → | Index |
| Join | → | Embedded Document |
| Foreign Key | → | Reference |
| Partition | → | Shard |

# CRUD operations - create

*Insert a new user.*

SQL

```
INSERT INTO users                          <-- table
            ( name, age, status )          <-- columns
VALUES      ( "sue", 26, "A" )             <-- values/row
```

MongoDB

```
db.users.insert (                          <-- collection
   {
     name: "sue",                          <-- field: value
     age: 26,                              <-- field: value
     status: "A"                           <-- field: value
   }                                       } document
)
```

# CRUD operations – create (cont'd)

Collection

Document

```
db.users.insert(
                  {
                    name: "sue",
                     age: 26,
                  status: "A",
                  groups: [ "news", "sports" ]
                  }
                )
```

Document

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

insert

Collection

```
{ name: "al", age: 18, ... }
{ name: "lee", age: 28, ... }
{ name: "jan", age: 21, ... }
{ name: "kai", age: 38, ... }
{ name: "sam", age: 18, ... }
{ name: "mel", age: 38, ... }
{ name: "ryan", age: 31, ... }
{ name: "sue", age: 26, ... }
```

*©MongoDB [3]*

# CRUD operations - read

```
$ db.courses.find({ level: { $gt: 2 }}, { name: true }).limit(5);
```

collection filters projection

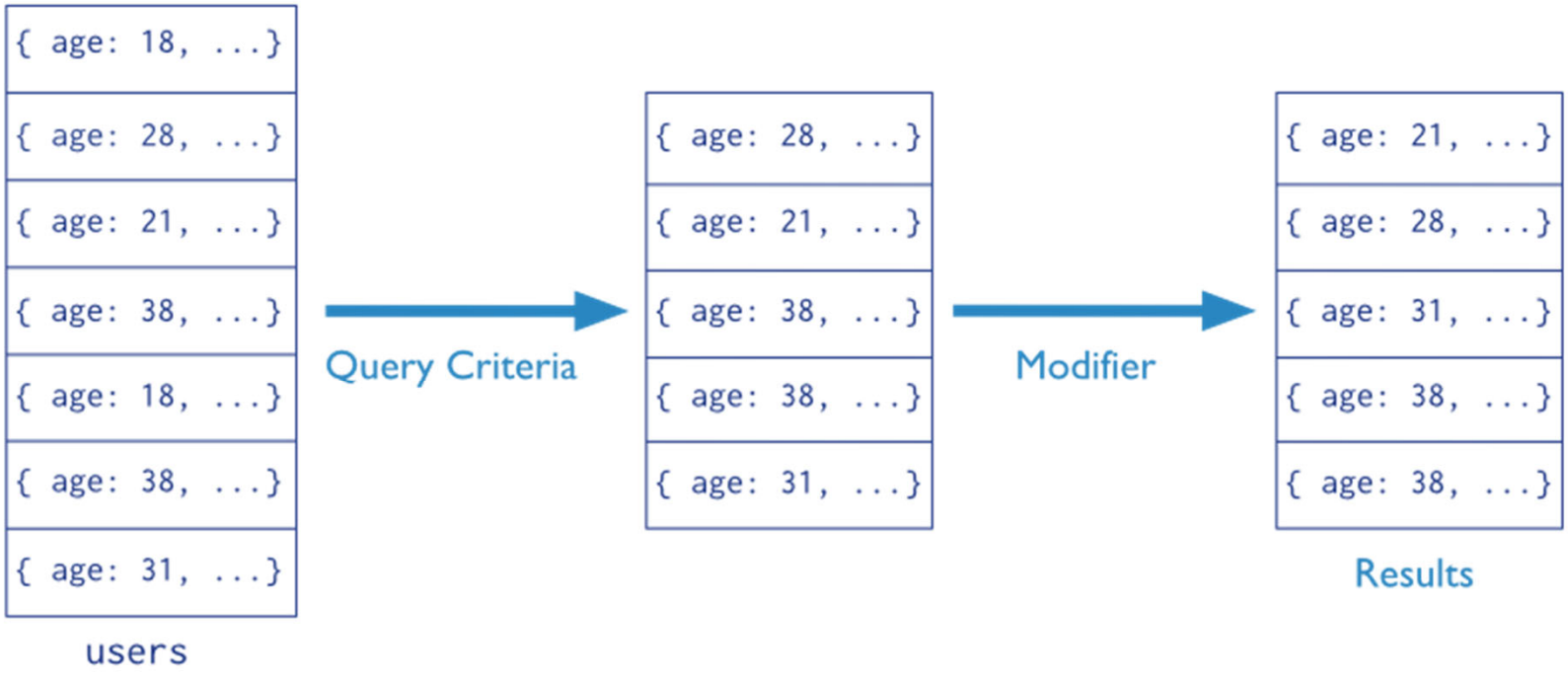
```
SELECT name
   FROM courses

WHERE level > 2

LIMIT 5
```

# CRUD operations - read

*Find the users of age greater than 18 and sort by age.*

Collection | Query Criteria | Modifier

```
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )
```




*©MongoDB [3]*

# CRUD operations – read (find)

```
$ db.courses.find({
    name: "Databases"
  });
```

```
$ db.courses.find({
    $or: [{ level: { $gt: 1 } }, { level: { $lt: 3 } }]
  });
```

- Operators:
  - Comparison – **$eq, $ne, $gt**, **$gte**, **$in**, **$lt**, **$lte**
  - Logical – **$and**, **$nor**, **$not**, **$or**
  - Element – **$exists**, **$type**

# CRUD operations - update

*Update the users of age greater than 18 by setting the status field to A.*

SQL

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

MongoDB

```
db.users.update(                    <-- collection
   { age: { $gt: 18 } },            <-- update criteria
   { $set: { status: "A" } },       <-- update action
   { multi: true }                  <-- update option
)
```

*©MongoDB [4]*

# CRUD operations - delete

*Delete the users with status equal to D.*

SQL

```
DELETE FROM users      <-- table
WHERE  status = 'D'    <-- delete criteria
```

MongoDB

```
db.users.remove(           <-- collection
   { status: "D" }         <-- remove criteria
)
```

# MongoDB Queries – Bulk Insert

```
var bulk = db.courses.initializeUnorderedBulkOp();

bulk.insert( { name: "ASP.NET MVC", level: 3 } );

bulk.insert( { item: "Web Development", level: 3 } );

bulk.insert( { item: "SPA Applications", level: 2 } );

bulk.execute();
```

# Embedding documents

- Nesting of objects and arrays inside a BSON document
- For a “contains” type of relationship
- Retrieve entire document with one call

```
{ _id: ObjectId(‘12345‘),
  author: 'joe',
  created : new Date('03/28/2009'),
  title : 'Yet another blog post',
  text : 'Here is the text...',
  tags : [ 'example', 'joe' ],
  comments : [ { author: 'jim', comment: 'I disagree' },
               { author: 'nancy', comment: 'Good post' }
  ]
}
```

# Linking documents

- “application-level relations”
- Where embedding would cause duplication of data

```
{ _id: ObjectId(‘12345‘),
  author: 'joe',
  created : new Date('03/28/2009'),
  title : 'Yet another blog post',
  text : 'Here is the text...',
  tags : [ 'example', 'joe‘ ]
}
```

```
{ author: 'jim',
  post_id: ObjectId(‘12345‘),
  comment: 'I disagree‘
}
{ author: 'nancy',
  post_id: ObjectId(‘12345‘),
  comment: 'Good post'
}
```

# Querying

- Queries return a cursor, which can be iterated to retrieve results
- Query optimizer executes new plans in parallel
- Queries are expressed as BSON documents which indicate a query pattern

```
db.users.find({'last_name': 'Smith'})

// retrieve ssn field for documents where last_name == 'Smith':
db.users.find({last_name: 'Smith'}, {'ssn': 1});

// retrieve all fields *except* the thumbnail field, for all documents:
db.users.find({}, {thumbnail:0});

// retrieve all users order by last_name:
db.users.find({}).sort({last_name: 1});

// skip and limit:
db.users.find().skip(20).limit(10);
```

# Advanced querying

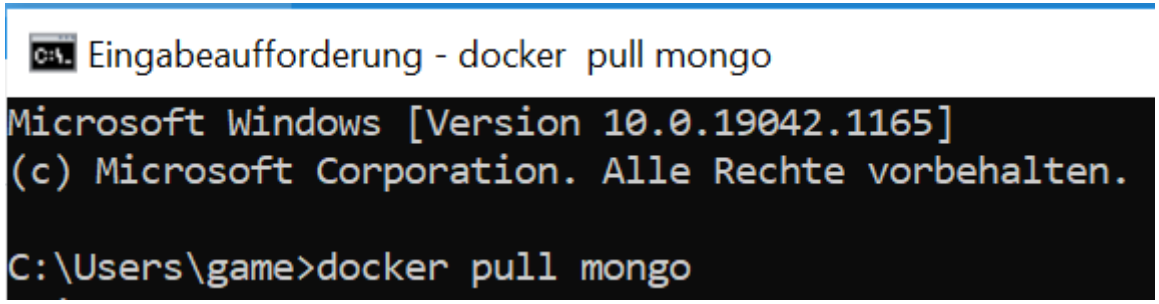
```
{ name: "Joe", address: { city: "San Francisco", state:
"CA" } , likes: [ 'scuba', 'math', 'literature' ] }

// field in sub-document:
db.persons.find( { "address.state" : "CA" } )

// find in array:
db.persons.find( { likes : "math" } )

// regular expressions:
db.persons.find( { name : /acme.*corp/i } );

// javascript where clause:
db.persons.find("this.name != 'Joe'");

// check for existence of field:
db.persons.find( { address : { $exists : true } } );
```

# Tools

- MongoDB is an open-source DB system
  - So there are many available viewers
- Some, but not all are:
  - **MongoDB CLI**
    - Comes with installation of MongoDB
    - Execute queries inside the CMD/Terminal
  - **MongoVUE** & **UMongo** & **Robomongo**
    - Provide UI to view, edit are remove DB documents
    - Execute queries inside the tool
  - **MongoDB Compass**
  - **Studio 3T**

# MongoDB und Docker

- Docker + Docker Desktop
- Load a docker image
- From https://hub.docker.com/_/mongo




- run an instance

```
C:\Users\game>docker run --name mymongo -d mongo:latest
61cdafad06ecda4f68129e5e401329fa93b14dc49ee6614bf85d7516f60a1afe
```

# MongoDB und Docker

- Run an instance with port mapping

```
C:\Users\game>docker run --name mymongo -p 27017:27017 -d mongo:latest
85d3d14b3c7ff02ddfce8892c18d1ccfb09dd672050afa789aff51c91d7a3159
```

- Open bash

```
C:\Users\game>docker exec -it mymongo bash
```

# MongoDB Atlas

- Cloud Solution
- Create Atlas Account
- Create Free Cluster
- Create DB User
- Import Sample Data

# MongoDB Docker + Atlas

- From the bash open a client for your DB on Atlas

```
root@85d3d14b3c7f:/bin# mongosh "mongodb+srv://cluster0.d0woq.mongodb.net/myFirstDatabase" --username gam
```

# Auf los geht‘s los ;-)