Q&A 2 Session

MS2 Assignment

Welcome!

Agenda

- Organisational Issues
- Final Presentation
- MS2 Details
- Q&A

universität wien

Welcome!

Agenda

- Organisational Issues
- Final Presentation
- MS2 Details
- Q&A

universität wien

Organisational Issues

- Everyone should have read and understood feedback by now

- Define your collaboration
  - Fix issues mentioned in feedback
  - Agree on a timeline, prioritize team tasks
  - Each student is responsible for his/her use cases in RDBMS as well as NoSQL!

- Milestone 2 is due till 20. January 2026, 13:00
- Use the templates provided on Moodle (Word and Latex)
- Total Points: 30

Organisational Issues

- Requirements are ...
  - defined in the Guidelines document
  - and summarized in the template

- Technical questions are best asked in the forum

- Administrative questions or personal issues are best discussed via email (see Moodle)
  - **ise@cs.univie.ac.at**

Welcome!

Agenda

- Organisational Issues
- Final Presentation
- MS2 Details
- Q&A

Final Presentation

- **Attendance is mandatory! (Will be graded!)**
- Presentations will start from the **21st of January** 2026
  - Time slots must be reserved in Moodle (Email will follow in January)
  - **Presentation slides** are to be uploaded separately (deadline same as MS2)
  - Presentation must not exceed 20 minutes + 10 minutes of discussion
- Presentation is online using BBB on Moodle
  - Audio and video are required by every participant
- These details about what to expect can be found in the **guidelines document**

Final Presentation

Presentation should contain:

- ER and an overview of the domain and system
- Summary of use cases and reports
- Ideas behind NoSQL Design
- Demo

1. Deploy system
2. Data generation
3. Use Case 1 + Report 1
4. Use Case 2 + Report 2
5. Data Migration
6. Report 1 + Report 2
7. Use Case 1 + Report 1
8. Use Case 2 + Report 2

Welcome!

Agenda

- Organisational Issues
- Final Presentation
- MS2 Details
- Q&A

universität wien

MS2 Details

MS1 Recap and Revisions

2.1 Infrastructure

Container Infrastructure

2.2 – RDBMS Implementation

Data Import and Relational Implementation

2. 3 – NoSQL Implementation

DB Design and Data Migration

Use-Case Implementation in NoSQL

Query Statements

Indexing

MS1 Recap and Revisions

Recap

- Include the ER-Diagram from MS1 at the beginning of the MS2 pdf
- If you have an updated version of the ER-Diagram, please include both the original and the updated versions

Changes

- Document any changes to MS1, including adjustments made based on our feedback or modifications necessary for accurate and efficient implementation and summarize changes

MS2 Details

MS1 Recap and Revisions

2.1 Infrastructure

Container Infrastructure

2.2 – RDBMS Implementation

Data Import and Relational Implementation

2. 3 – NoSQL Implementation

DB Design and Data Migration

Use-Case Implementation in NoSQL

Query Statements

Indexing

See Guidelines Document

NoSQL in a Nutshell
A Schema-less Database

In NoSQL not every document inside the same collection must have the same structure.

NoSQL in a Nutshell
IS-A Relationship with Merged Documents

Employee Entities

```
{
 _id: "pcarter",
 name: "P. Carter",
 user: "pc",
 pwd: "Howard<3"
},
{
 _id: "srogers",
 name: "S.Rogers",
user: "sr",
pwd: "SHIELD123"
}
```

Agent Entities

```
{
    _id: "pcarter",
    take_downs: 3,
    specialty: "kicking"
}
```

Heros Entities

```
{
    _id: "srogers",
    weapon: "Shield",
    press_name: "C. America",
    costume: "Blue-Red-White"
}
```

Resulting Documents

universität wien

NoSQL in a Nutshell
IS-A Relationship with Merged Documents

Employee Entities

```
{
 _id: "pcarter",
 name: "P. Carter",
 user: "pc",
 pwd: "Howard<3"
},
{
 _id: "srogers",
 name: "S.Rogers",
user: "sr",
pwd: "SHIELD123"
}
```

Agent Entities

```
{
    _id: "pcarter",
    take_downs: 3,
    specialty: "kicking"
}
```

Heros Entities

```
{
    _id: "srogers",
    weapon: "Shield",
    press_name: "C. America",
    costume: "Blue-Red-White"
}
```

Resulting Documents

```
{
_id: "pcarter",
name: "P. Carter",
user: "pc",
pwd: "Howard<3,
take_downs: 3,
specialty: "kicking"
},
{
_id: "srogers",
name: "S.Rogers",
user: "sr",
pwd: "SHIELD123",
weapon: "Shield",
press_name: "C. America",
costume: "Blue-Red-White"
}
```

Notice that related documents have been merged utilizing the "schemaless-ness" of NoSQL databases

universität wien

NoSQL in a Nutshell
Denormalization (aka. selective duplication)

Publisher Collection

```
{
    publisher_id: "oreilly",
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
}
```

Breaking apart the normalized model by duplicating data and sacrificing storage (and update) efficiency for increased read performance (but keeping references for details). Best applicable to data which is expected to not update frequently.

Books Collection

```
{  title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   language: "English",
   publisher: {
                name: "O'Reilly Media",
                publisher_id: "oreilly"
              }
},
{  title: "50 Tips and Tricks for MongoDB Devs",
   author: "Kristina Chodorow",
   language: "English",
   publisher: {
                name: "O'Reilly Media",
                publisher_id: "oreilly",
              }
}
```

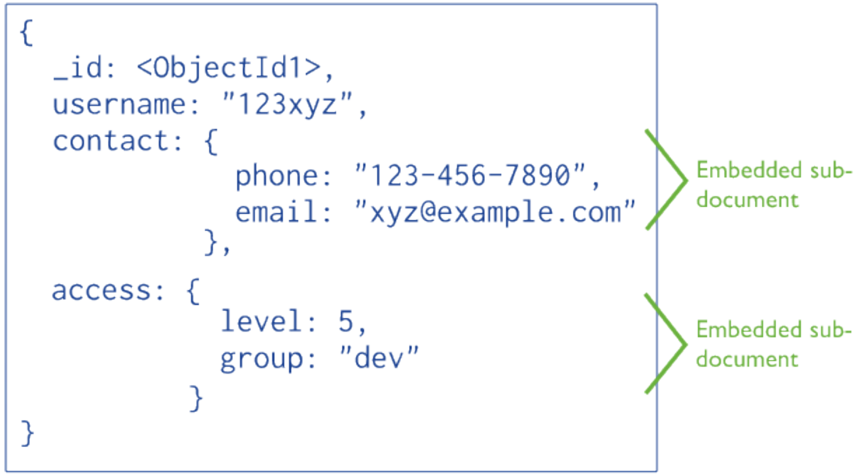
universität wien

NoSQL in a Nutshell
Embedded Documents

Favor embedding …

NoSQL in a Nutshell Embedding

- Consider Atomicity
  - Store related pieces of information in the same database record
  - Issue fewer queries and updates to complete common operations
  - Better performance for read operations
- Best suited for
  - "Contains" relationships (e.g. weak entities, IS-A relationship)
  - "One-To-One" relationships
  - "One-To-Many" where the children always appear or are viewed in the context of the parent



```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
          }
}
```

Source: https://www.mongodb.com/docs/manual/data-modeling/

NoSQL in a Nutshell
1:1 Relationship with Embedded Documents

Patron Entities

```
{
   _id: "joe",
   name: "Joe Bookreader"
}
```

Address Entities

```
{
   _id: "joe_address",
   patron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}
```

Resulting Document

universität wien

Source: https://www.mongodb.com/docs/manual/data-modeling/

NoSQL in a Nutshell
1:1 Relationship with Embedded Documents

Patron Entities

```
{
   _id: "joe",
   name: "Joe Bookreader"
}
```

Address Entities

```
{
   _id: "joe_address",
   patron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}
```

Resulting Document

```
{
   _id: "joe",
   name: "Joe Bookreader",
   address: {
     street: "123 Fake Street",
     city: "Faketon",
     state: "MA",
     zip: "12345"
   }
}
```

Notice the removal of no longer required keys (address._id) and references (address.patron_id)

universität wien

Source: https://www.mongodb.com/docs/manual/data-modeling/

NoSQL in a Nutshell
1:N Relationship with Embedded Documents

Patron Entities

```
{
   _id: "joe",
   name: "Joe Bookreader"
}
```

Address Entities

```
{
   _id: "joe_address_prim",
   patron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
},
{
   _id: "joe_address_alt",
   patron_id: "joe",
   street: "1 Other Street",
   city: "Miami",
   state: "FL",
   zip: "33164"
}
```

Resulting Document

NoSQL in a Nutshell
1:N Relationship with Embedded Documents

Patron Entities

```
{
    _id: "joe",
    name: "Joe Bookreader"
}
```

Address Entities

```
{
    _id: "joe_address_prim",
    patron_id: "joe",
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
},
{
    _id: "joe_address_alt",
    patron_id: "joe",
    street: "1 Other Street",
    city: "Miami",
    state: "FL",
    zip: "33164"
}
```

Resulting Document

```
{
 "_id": "joe",
 "name": "Joe Bookreader",
 "addresses": [
   {"street": "123 Fake Street",
    "city": "Faketon",
    "state": "MA",
    "zip": "12345"},
   {"street": "1 Other Street",
    "city": "Miami",
    "state": "FL",
    "zip": "33164"       }
   ]
}
```

Notice the array data type of addresses and FK/PK removal

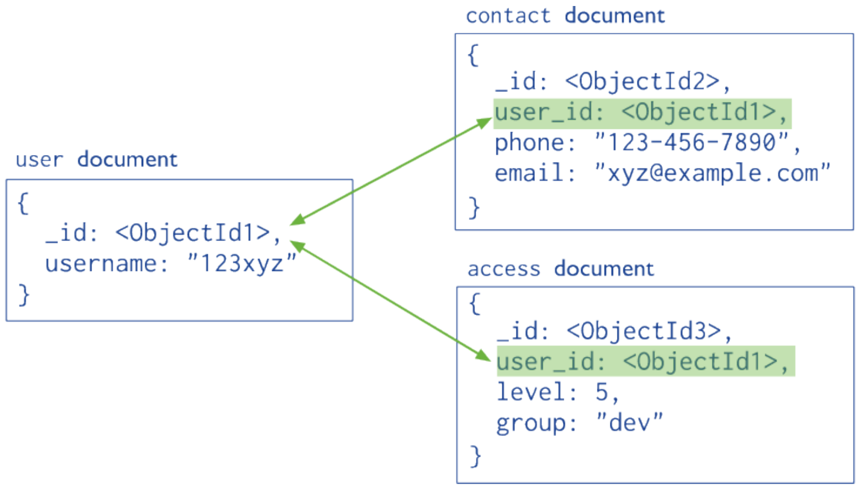
NoSQL in a Nutshell
Document
References

Favor embedding ...

... unless there is a compelling reason not to

NoSQL in a Nutshell Referencing

- Normalized Data
  - Relationships are described using references
  - Rules of database normalization are applicable
  - Consistency needs to be handled on application level
- Best suited for
  - When duplication of data but would not provide sufficient read performance advantages
  - more complex many-to-many relationships
  - to model large hierarchical data sets



```
user document
{
  _id: <ObjectId1>,
  username: "123xyz"
}

contact document
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}

access document
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Source: https://www.mongodb.com/docs/manual/data-modeling/

NoSQL in a Nutshell
1:N Relationship with Documents References

Publisher Collection

```
{
 _id: "oreilly",
 name: "O'Reilly Media",
 founded: 1980,
 location: "CA"
}
```

Like in relational, normalized models. Also known as "Referencing on the 1-side"

Books Collection

```
{
   _id: 123456789,
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher_id: "oreilly"
},
{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher_id: "oreilly"
}
```

universität wien

NoSQL in a Nutshell
1:N Relationship with Documents References

Publisher Collection

```
{
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA",
    books: [123456789, 234567890, ...]
}
```

Utilizing data type Array to implement "Referencing on the N-side"

Caveat: Publisher.books is a mutable, growing array and can lead to inefficiencies for large numbers of references and/or high update rates.

Books Collection

```
{
   _id: 123456789,
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English"
}

{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English"
}
```

NoSQL in a Nutshell
1:N Relationship with Documents References

Publisher Collection

```
{
 _id: "oreilly",
 name: "O'Reilly Media",
 founded: 1980,
 location: "CA",
 books: [123456789, 234567890, ...]
}
```

A combined solution, although more expensive/complex on updates, can yield significant performance improvements when reading data.

Books Collection

```
{
  _id: 123456789,
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",
  publisher_id: "oreilly"
},
{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher_id: "oreilly"
}
```

universität wien

NoSQL in a Nutshell
When to do What? 5 Rules of Thumb

- One: Favor embedding unless there is a compelling reason not to
- Two: The need to access an object on its own is a compelling reason not to embed it
- Three: High-cardinality arrays are a compelling reason not to embed
- Four: Consider the write/read ratio of a collection/document when denormalizing.
- Five: Structure your data to match the ways that your application queries and updates it.

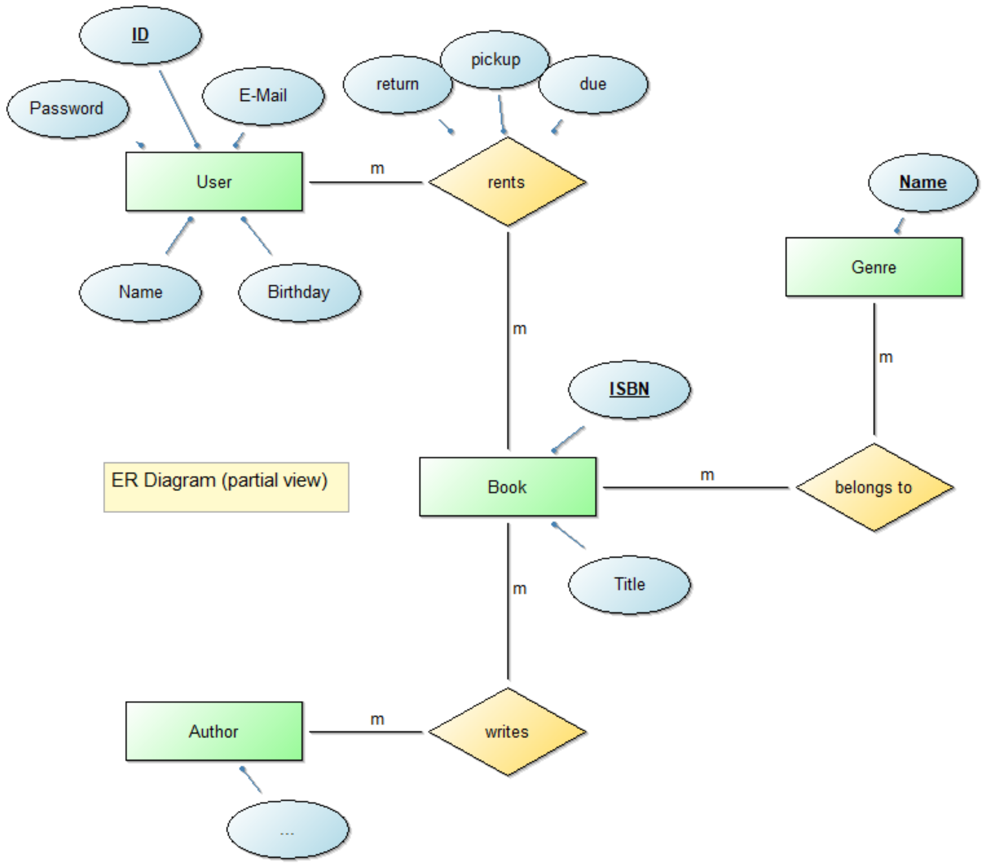
MongoDB Documentation: https://www.mongodb.com/docs/manual/

Example

Simplified Library

Some Ideas for 2.3

2.3 – NoSQL Design – Example Library



2.3 – NoSQL Design – Example Library

```
{ _id: 21,
  name: “John Dow”,
  email: john@example.com,
  password: “pasword123”,
  birthday: “2001-01-01”,
  rentals: [
    { isbn: “3442460719”,
      title: “Brief History
       of nearly Everything”,
      author: [{
        id: “42”,
        name: “B. Bryson”
      }],
      pickup: “2022-04-15”,
      returned: “2022-04-25”
  },{
      isbn: “9780857522405 ”,
      title: “The Body”,
      author: [{
        id: “42”,
        name: “B. Bryson”
      }],
      pickup: “2022-04-27”,
      due: “2022-05-27”,
  }, { … }
]}
```

User Collection Design

- *User* will be accessed almost always together with rental history, as it is included on the landing page of the user, thus embedding saves a second query or join
- *rentals* are expected be in the O(100-1000)
- *rentals* are expected be updated only every few days at best (i.e. very low update frequency)
- Duplicated (immutable) data:
  - From Books: isbn, title, author
- *returned* will only be set if the book is already returned to avoid null values
- *due* is only set for outstanding rentals and will be removed on return

2.3 – NoSQL Design – Example Library

```
{ _id: 3442460719,
  title: “Brief History of  Everything”,
  author: [
 { id: “42”,
   name: “B. Bryson”
 }],
  date: “2005-09-12”,
 genre: [“science”, “comedy”],
  rentals: [
  { id: 21,
    name: “J. Dow”,
    FromDate: “2022-04-15”,
    DueDate: “2022-05-15”,
    ReturnDate: “2022-04-25”
   },
   { id: 22,
     name: “A. Miller”,
     FromDate: “2022-04-27”,
     DueDate: “2022-05-27”
   },  …
 ]}
```

Books Collection Design

- *Books* will be accessed almost always together with rental history information (included in the *book* details) and thus save a second query or join
- Include *genre* and a subset of *author* saves one join each
- *rentals* are expected to be in the O(100-1000), thus not concerned about immutable growing
- *rentals* are expected to be updated only on return (every few days, thus very low update frequency)
- Duplicated (immutable) data:
  - Users: [id, name]
  - Authors: [id, name]
  - Genres: [name]
- *ReturnDate* will only be set if the book is already returned to avoid null values (*DueDate* is kept for statistics)

2.3 – NoSQL Indexing – Example Library

```
{ _id: 21,
  name: "John Dow",
  email: john@example.com,
  password: "pasword123",
  birthday: "2001-01-01",
  rentals: [
  { isbn: "3442460719",
     title: "Brief History
      of nearly Everything",
     author: [{
       id: "42",
       name: "B. Bryson"
     }],
     pickup: "2022-04-15",
     returned: "2022-04-25"
  },{
     isbn: "9780857522405 ",
     title: "The Body",
     author: [{
       id: "42",
       name: "B. Bryson"
     }],
     pickup: "2022-04-27",
     due: "2022-05-27",
  }, { … }
]}
```

Indices on User

- A **uniqueness index** *`email`*, to ensure uniqueness and find the user fast on login

  ```
  db.User.createIndex(
      { email: 1}, { unique: true })
  ```

- A **sparse index** on *due*, for all users with at least one outstanding rental

  ```
  db.User.createIndex(
   { rentals.due: 1},
   { sparse: true }})
  ```

2.3 – NoSQL Indexing – Example Library

```
{ _id: 3442460719,
  title: “Brief History of  Everything”,
  author: [
 { id: “42”,
   name: “B. Bryson”
 }],
  date: “2005-09-12”,
 genre: [“science”, “comedy”],
  rentals: [
  { id: 21,
    name: “J. Dow”,
    FromDate: “2022-04-15”,
    DueDate: “2022-05-15”,
    ReturnDate: “2022-04-25”
   },
   { id: 22,
     name: “A. Miller”,
     FromDate: “2022-04-27”,
     DueDate: “2022-05-27”
   },  …
]}
```

Indices on Books

- A **text index** on `title`, to ensure fast search for book titles
  `db.Books.createIndex({title: “text”})`

- A **partial index** on `ReturnDate`, report all rented books
  ```
  db.Books.createIndex(
    { rentals.ReturnDate: 1},
    { partialFilterExpression:
      { rentals.ReturnDate : {
          $exists: false
     }}})
  ```

- A **multi key index** on `genre`, to enable fast browsing
  `db.Books.createIndex({genre: 1})`

HINT: Using `$exists:false` is more efficient than using `$exists:true`, as there will be fewer outstanding rentals than returned ones

Student Presentations & Discussions