Migration of a web service back-end from a relational to a document-oriented database

Sebastian Drenckberg, Marius Politze

IT Center
RWTH Aachen University

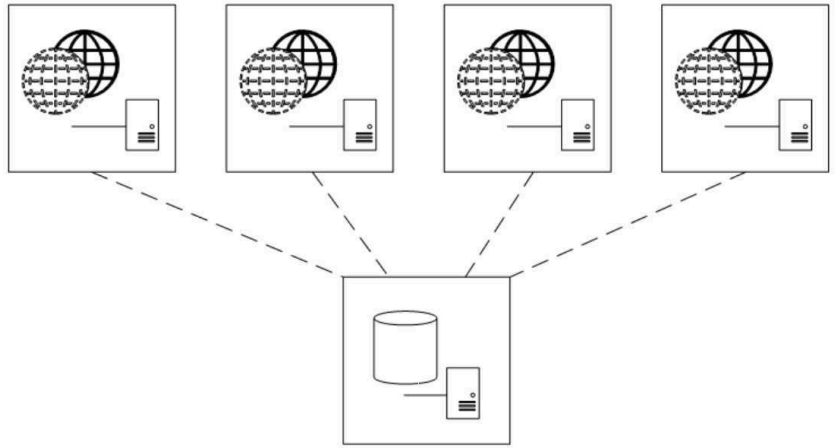
Outline

- Motivation
- From Relational to Document oriented
- Validation of Migration
- Generalization
- Conclusion

Motivation: Infrastructure

- ~20 Software developers for Process Supporting Software
  - 4-5 Agile Teams
  - Student Lifecycle, RWTHApp, eLearning, Quality Management, eScience

- Database server used for several “small” software projects
  - Agile development process → regular changes
  - Some are interactively used by 1000+ users

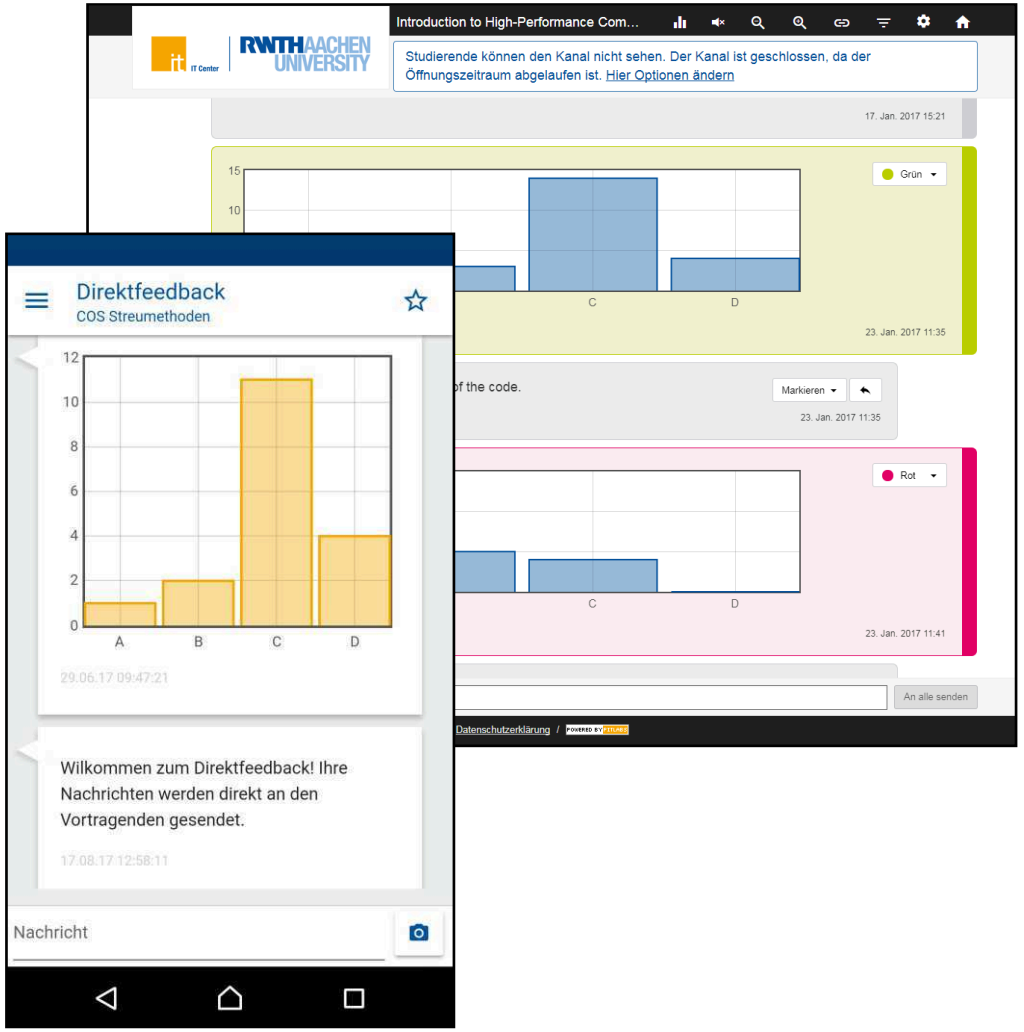


- Previously single instance of MS SQL Server 2008 R2
  - No redundancy
  - No scalability
  - Nightly Backups
  - → Single Point of Failure

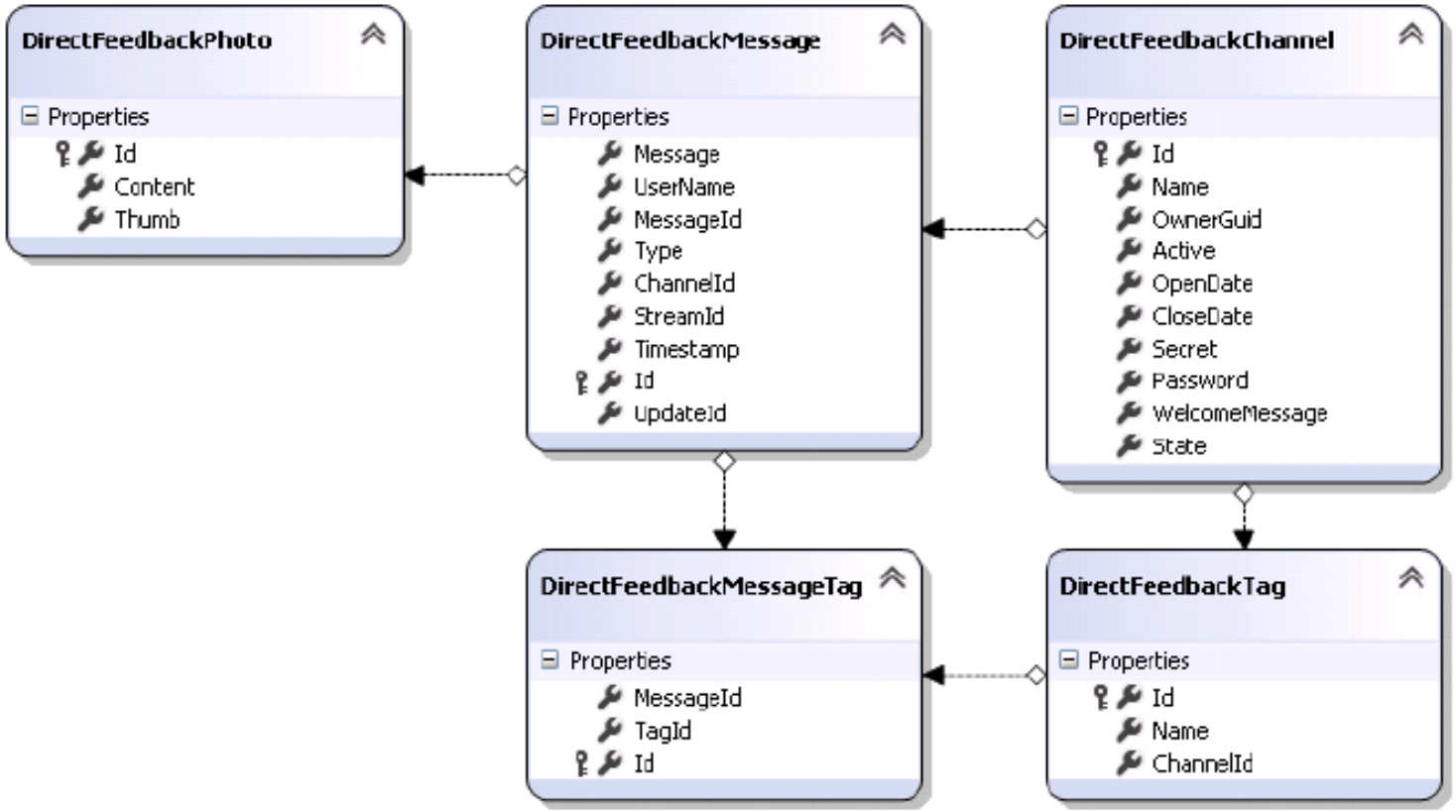
- Goal: more flexibility, scalability and redundancy
  - Consider new database systems / technologies
  - Limit migration effort / costs

Motivation: Case Study for Migration

- Audience Response System that is part of RWTHApp
  - Targeting large audiences 500-1200 students
  - Anonymous usage
- “Chat-like” 1:n communication during lectures
  - Teacher – Student
  - Teacher – all Students
- Multiple Message Types
  - Images
  - Polls
- Available via
  - RWTHApp
  - HTML5 Web Application



SQL Server: Relations



Language Integrated Query (LINQ)

- LINQ is a language extension to formulate queries on collection Classes like Lists

```
List<int> numbers = new List<int> { 3, 6, 2, 7, 9, 4, 1 };
```

- LINQ to SQL
  - Code generator to access relational databases
  - Relations can be accessed like Lists

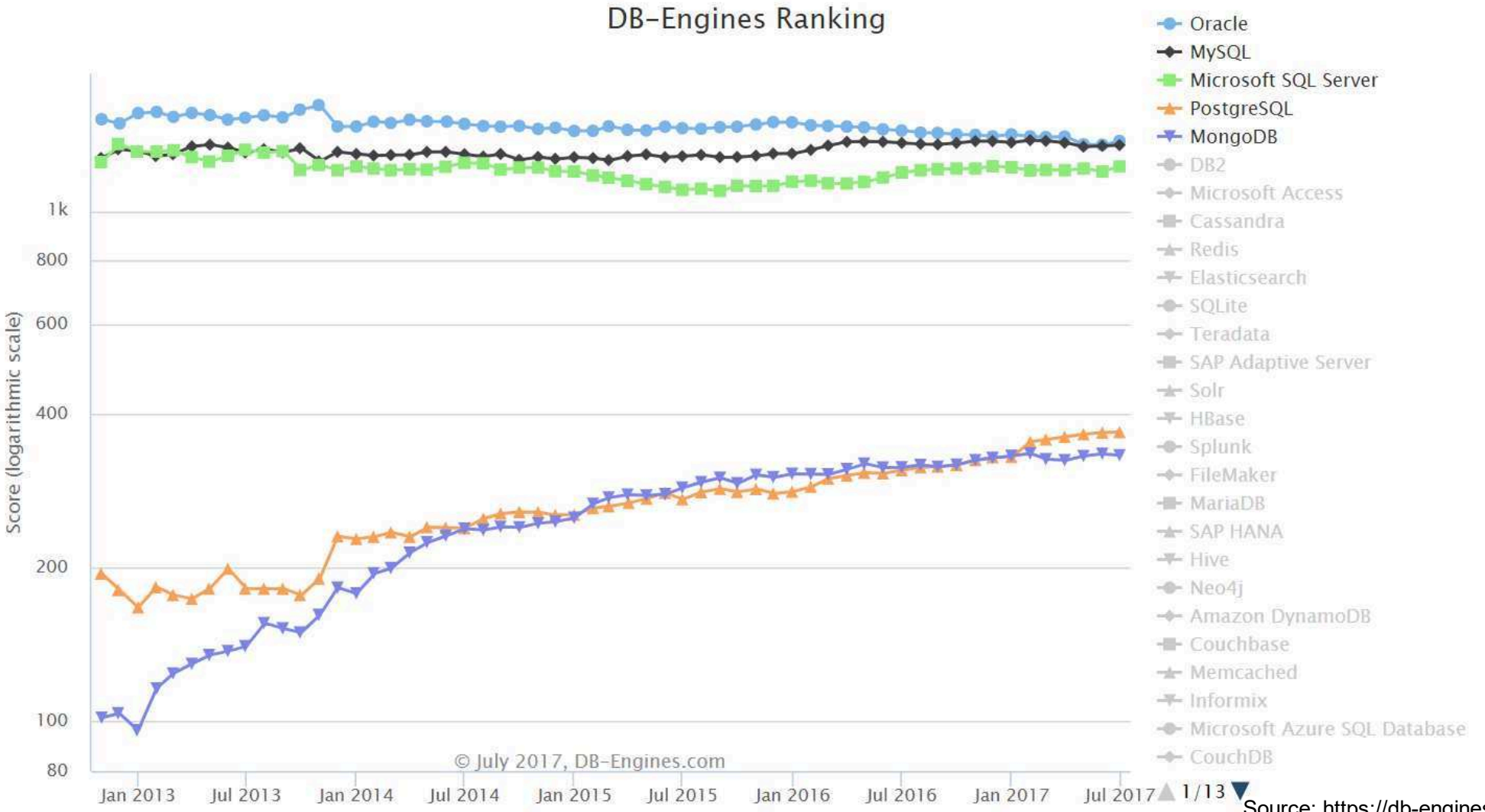
For example:

```

```

is (roughly) translated to:

```

```

Popularity of Document Oriented Databases



Source: https://db-engines.com/en/ranking

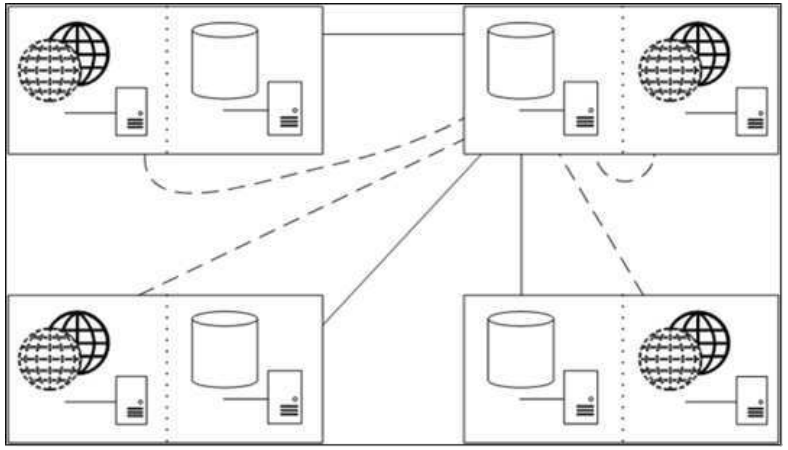
Document Oriented Databases / NoSQL

- Popularity is recently rising from niche to common use
- No predefined structure
  - Holds Documents consisting of key-value-pairs
  - Documents are organized in collections
- Common Formats:
  - JavaScript Object Notation (JSON)
  - Extended Markup Language (XML)
  - (or dialects)
- Implementation specific query languages

```
{
  "address": {
    "street": "Seffenter Weg",
    "no": 23,
    "zip": 52074,
    "city": "Aachen",
    "country": "Germany"
  }
}
```

MongoDB

- Published in 2009
- Connector library needed, available for many languages
  - C# library: `MongoDB.Driver`
  - Supports LINQ
- Multiple Collections per database process
  - DB process relatively lightweight (~300MB Disk, ~100MB Memory)
  - Allows DB process on Application servers
- Replication
  - Master-Multi Slave
  - Automatic Failover
  - “Every server-VM is equal”



Estimation of Migration Effort

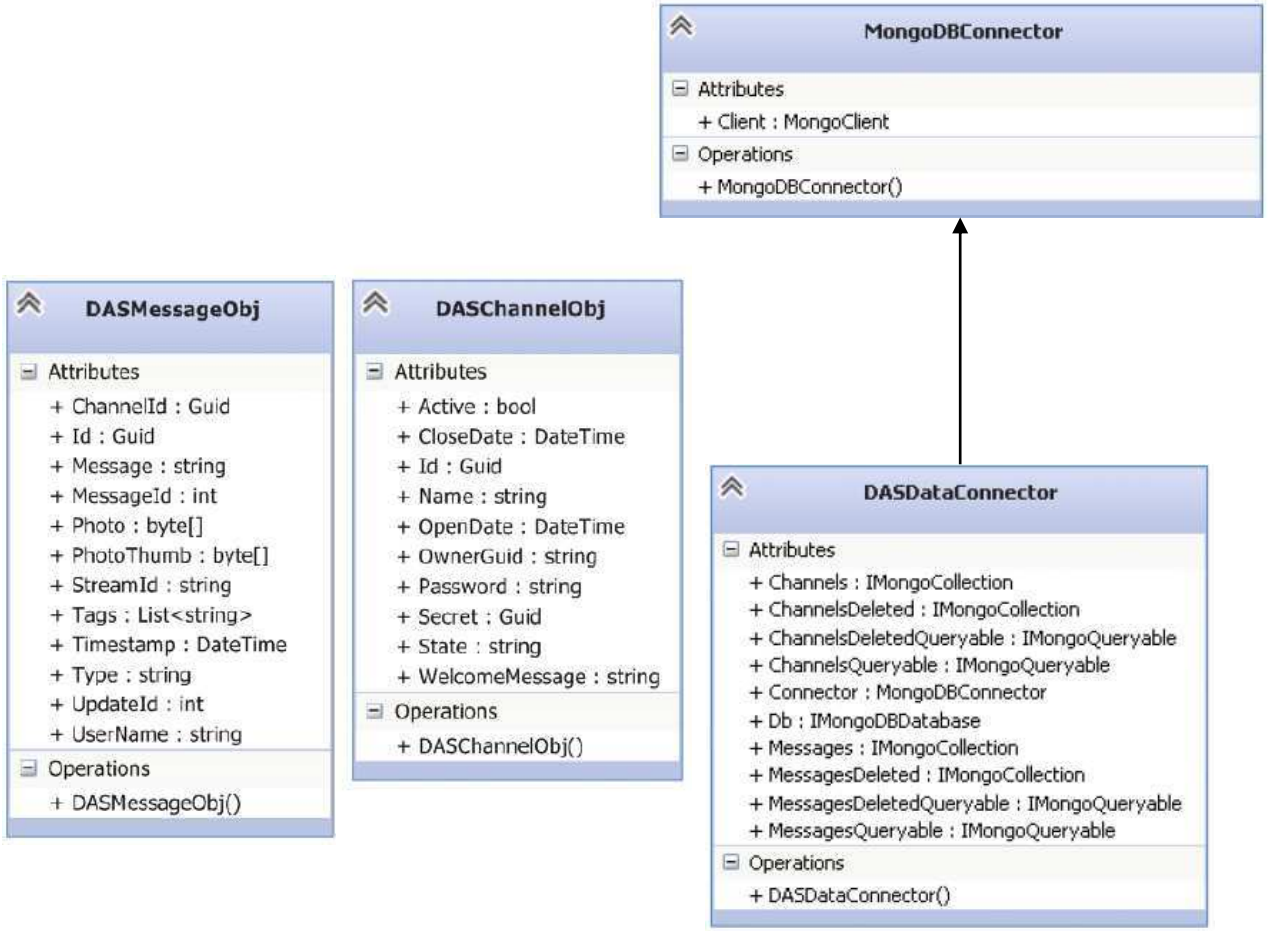
- Before

```
public static List<DASChannel> GetAllChannelsForUser(string[] personGguids) {
    using (var context = new DASDataContext()) {
```

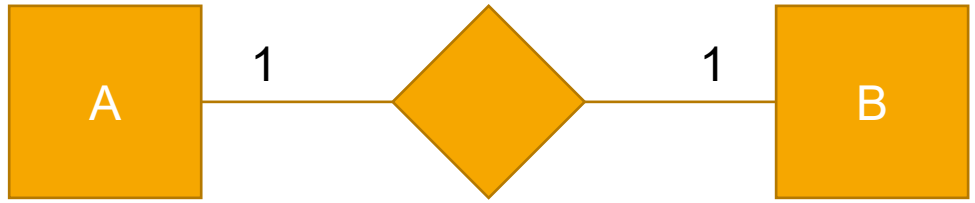
- After

Migration I: Replacing Code Generated By LINQ2SQL

- Replace generated classes by own code
  - DB connection
  - Serializable Types for stored Information
- Add explicitly typed methods for current application
- Generic connection class is reused in future migrations



Generalization: 1:1 Relation

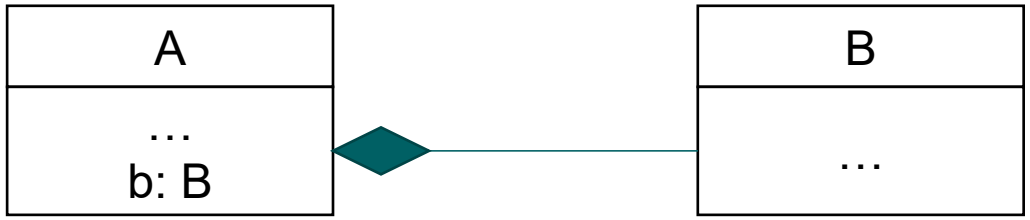


Association by reference

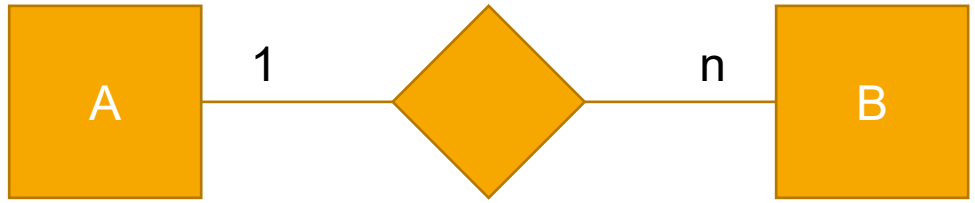
A
… b: int / gguid

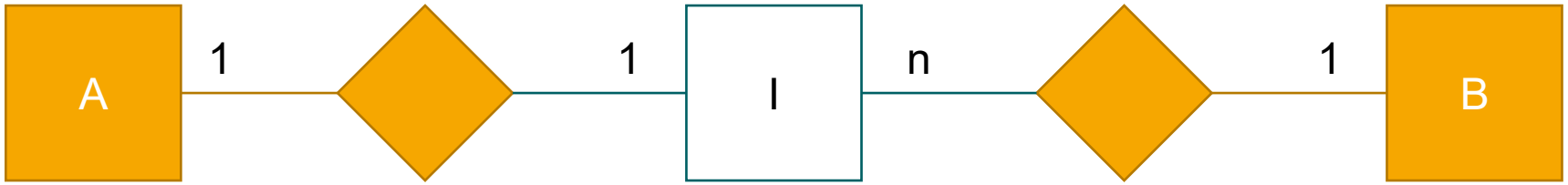
B
id: int / gguid …

Association by embedding

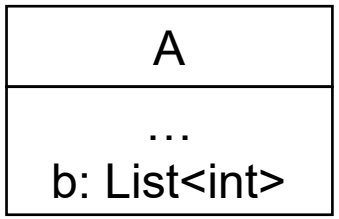


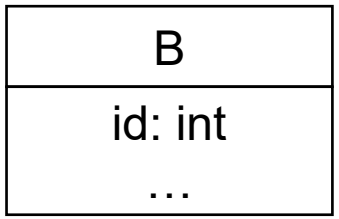
Generalization: 1:n Relation



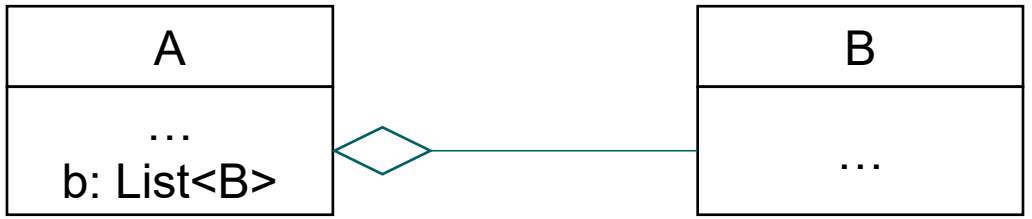


Association by reference List

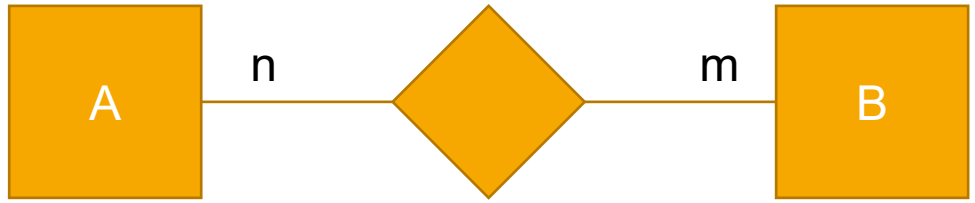


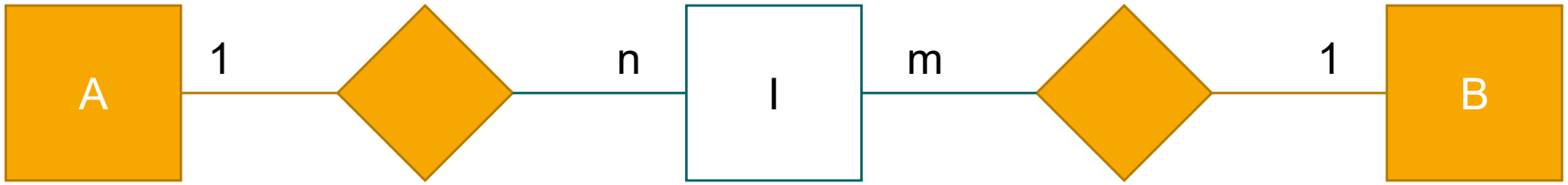


Association by embedding multiple documents

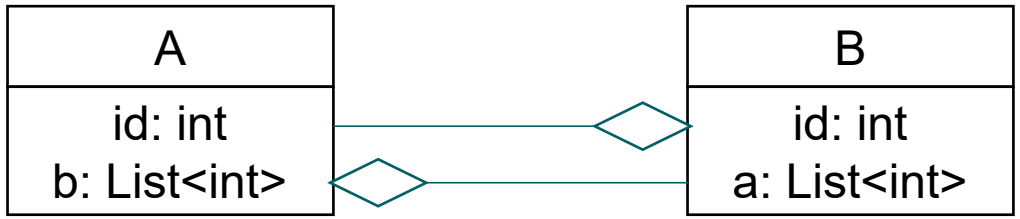


Generalization: n:m Relation





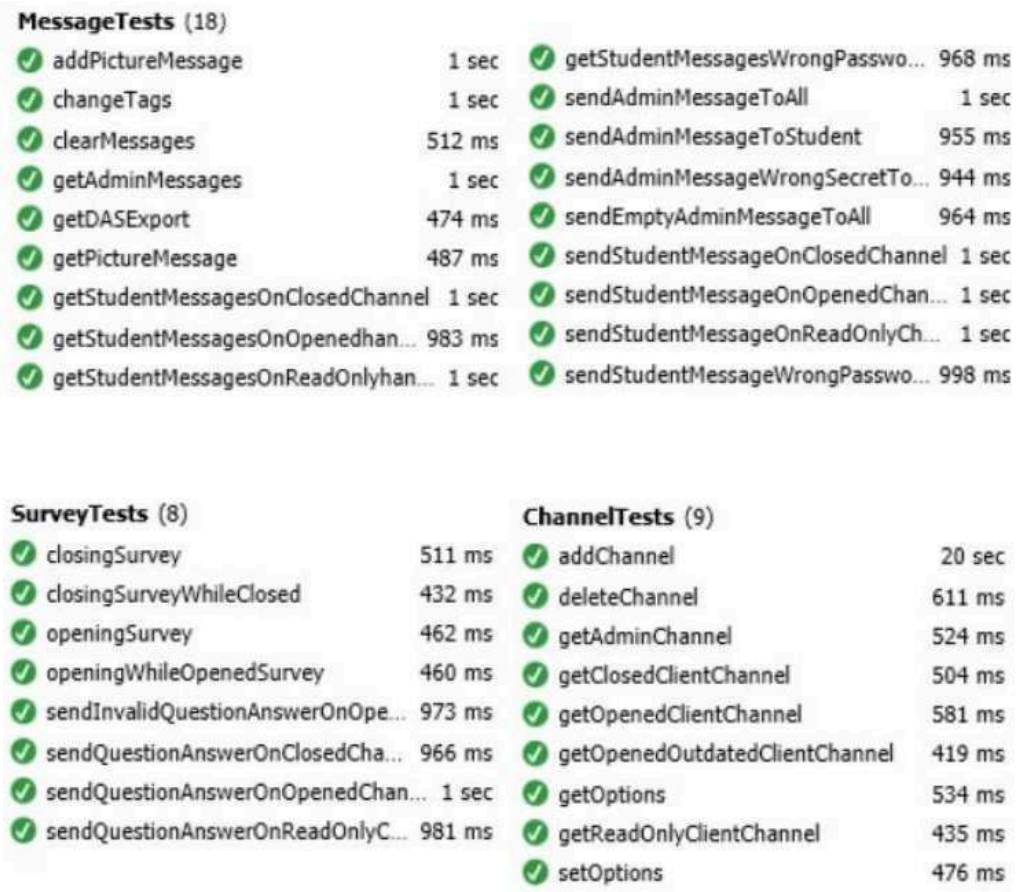
Association by reference List



Association by embedding?

Validation Using Coded Tests

- Integration Level Tests
  - API Level / Blackbox
  - Compare actual and expected results
- Independent for different use cases
  - Init and Cleanup always create the initial setup
  - Coded tests are executed on check-in
- Tests remain untouched
  - Results before and after can be compared
  - Creates a check-list during migration
- Additional Unit Tests should be considered



Conclusion

- Migration successful
  - Validation using automated integration tests
  - Generalization guides future migrations
- Production system running since Aug 2017
  - Clear documentation, steep learning curve
  - Running without significant issues
- Major Version Update was successful
  - Updating one server after another
  - 0 downtime
- More optimizations for MongoDB “native” applications
  - Server side aggregation pipelines
  - Map-Reduce
  - Sharding

Thank you for your attention

Vielen Dank für Ihre Aufmerksamkeit