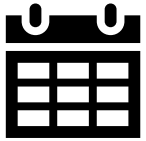
Datenbank Vertiefung

Moderne Datenbanken – Gruppe 1 - Stundenplan

Von: Nils Thorben Konopka, Lukas Meier, Rodrigo Galarza

Inhaltsverzeichnis

Aufgabenstellung

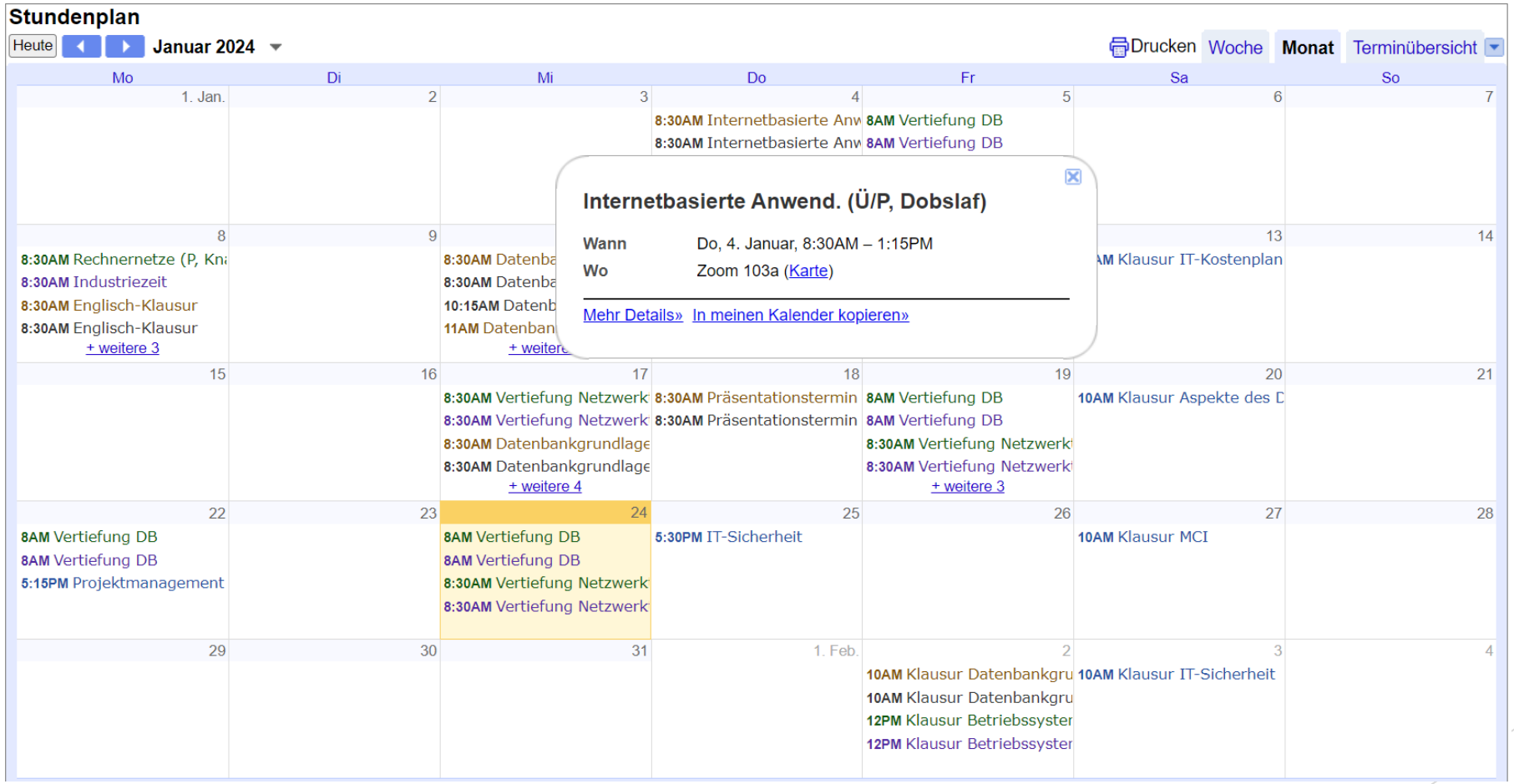


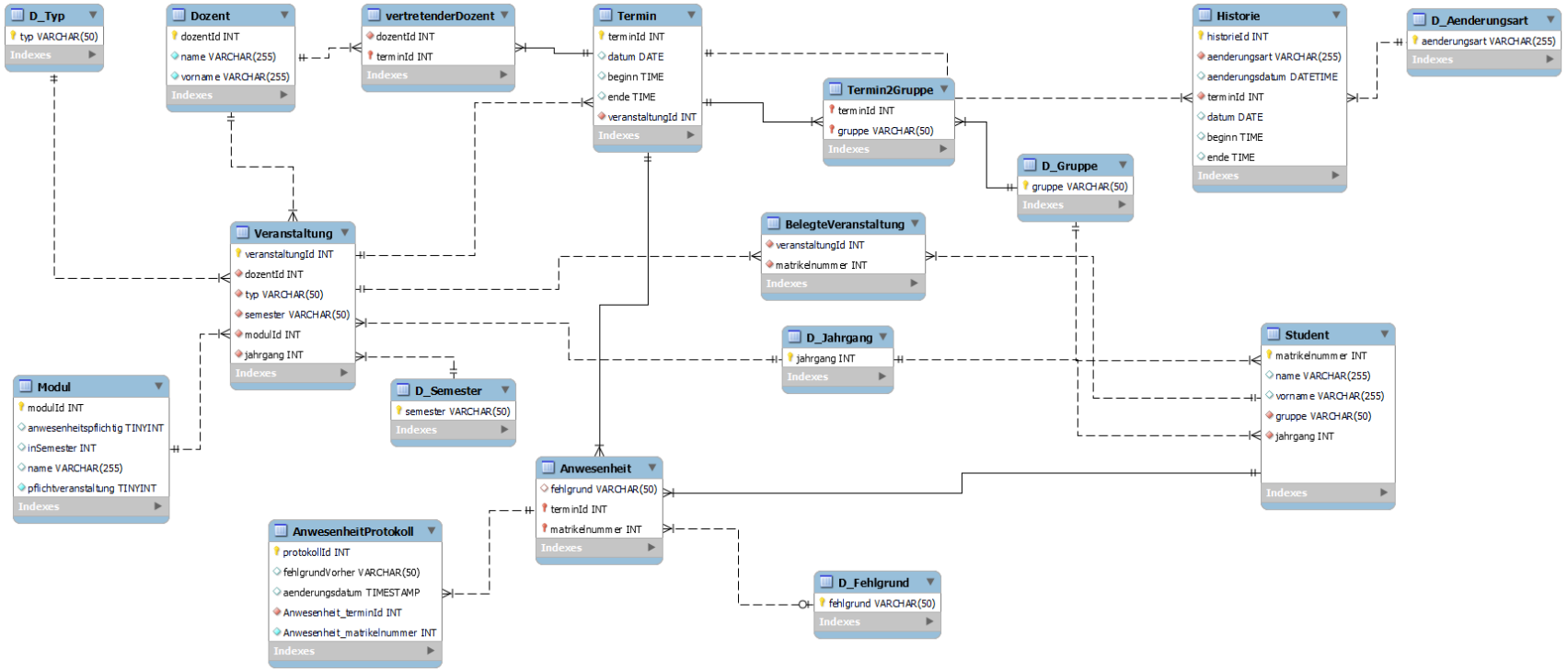
Mit Hilfe des Datenbankschemas Stundenpläne, Veranstaltungstermine und Abwesenheiten verwalten lassen.

Änderungen am Stundenplan Protokollieren.

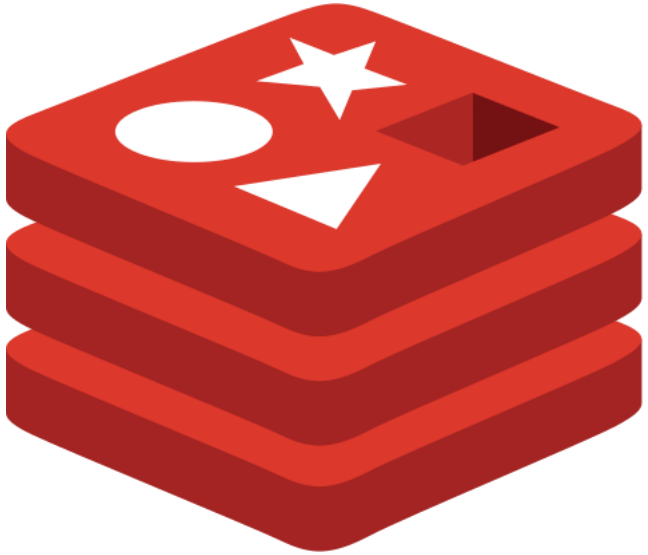
Anwesenheitslisten für Veranstaltungen generieren.

Aufgabenstellung





Datenbankmodell



redis

Redis

Key-Value Datenbanken

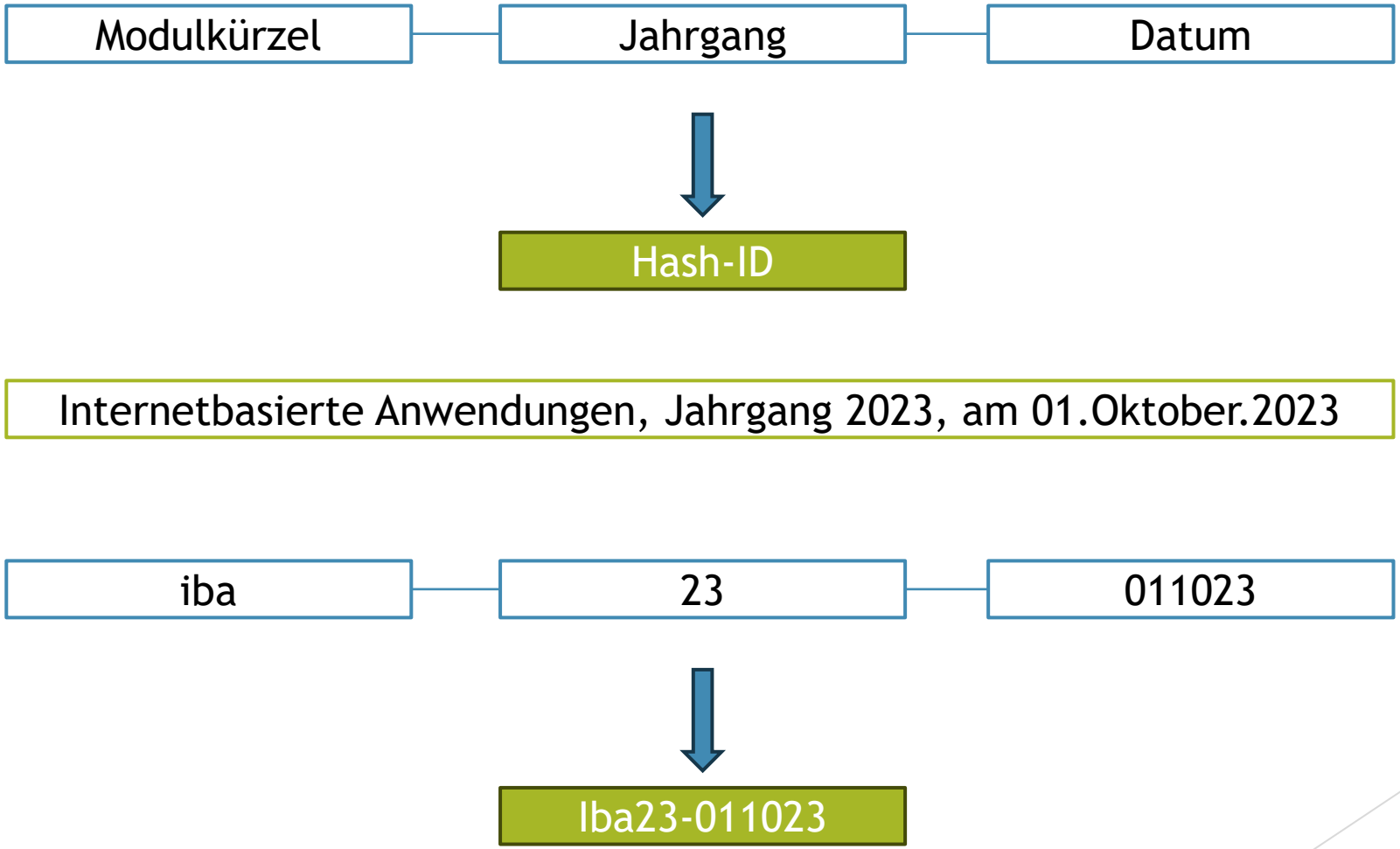
Teilbereich Umsetzung – Dozenten View

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Modul	veranstaltungid	typ	Dozent	datum	Start	Ende
Datenbanken	1	Vorlesungen	Müller	2025-10-01	08:00:00	10:00:00
Datenbanken	1	Vorlesungen	Müller	2023-10-01	08:00:00	10:00:00
Irgendwas im berufsbegleitenden Teil	2	Praktika	Schmidt	2024-04-02	10:00:00	12:00:00
Web-Engineering	3	Übungen	Fischer	2024-10-03	14:00:00	16:00:00

```
USE `Stundenplan`;
CREATE VIEW dozentView AS
SELECT modul.name AS Modul,
veranstaltung.veranstaltungid,
veranstaltung.typ,
dozent.name AS Dozent,
termin.datum, termin.beginn AS Start,
termin.ende AS Ende
FROM Termin termin
JOIN Veranstaltung veranstaltung ON veranstaltung.veranstaltungId = termin.veranstaltungId
JOIN Dozent dozent ON dozent.dozentId = veranstaltung.dozentId
JOIN Modul modul ON modul.modulId = veranstaltung.modulId;
```

Teilbereich Umsetzung – Dozenten View



Teilbereich Umsetzung – Dozenten View

Dozenten-View

```
HSET iba23-011023
id "iba23-011023"
dozentName "Anna Müller "
veranstaltungTyp "Vorlesungen"
semester "WS2023/2024 "
modulName "Internetbasierte Anwendungen"
datum "01.10.2023" beginn "08:00" ende "10:00" teilnehmer "iba23T" jahrgang 23
```

Teilnehmerliste

```
HSET iba23T
id "iba23T"
"Bubi Blauschuh" "Krank" "Thomas Koenigsmann" "Krank" "Maria Mandarina"
"Entschuldigt" "Katrin Kleeblatt" "unentschuldigt"
```

Teilbereich Umsetzung – Dozenten View

Metadaten

HSET meta
uebung "<modulkuerzel>U<datum>" teilnehmerliste "<modulkuerzel>T"
abfrage "<modulkuerzel>Abfrage"

Kürzel Map

HSET modulkuerzel "Internetbasierte Anwendungen" "iba"

Erweiterungen – Übungsveranstaltung

Abgabeliste (Übungen)

HSET iba23U-011023
id "iba23"
"Bubi Blauschuh" "Nicht abgegeben" "Thomas Koenigsmann" 10 "Maria Mandarina"
"Nicht abgegeben" "Katrin Kleeblatt" 5

Aufrufliste

LPush iba23Abfrage
"Bubi Blauschuh" "Thomas Koenigsmann" "Maria Mandarina" "Katrin Kleeblatt"

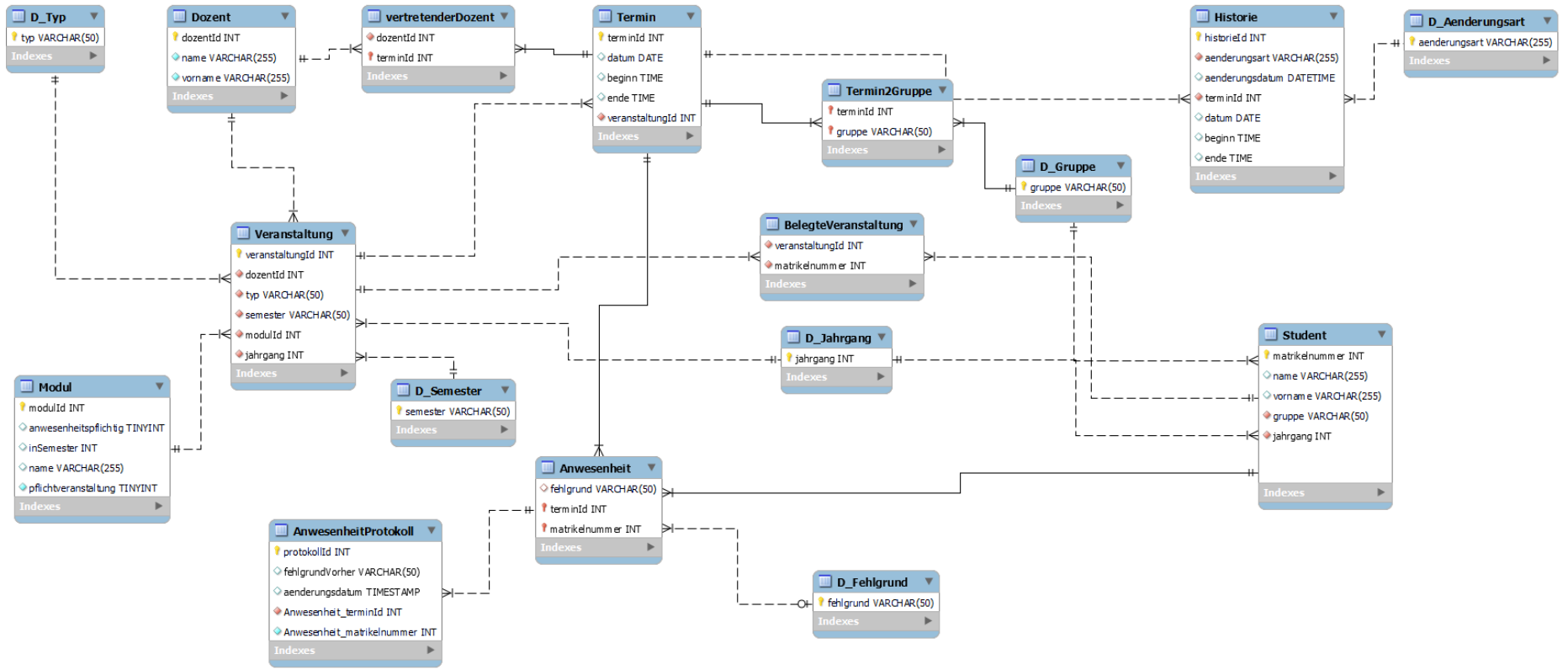
RPOP iba23Abfrage → Bubi Blauschuh

LPUSH iba23Abfrage "Bubi Blauschuh"

Neo4J

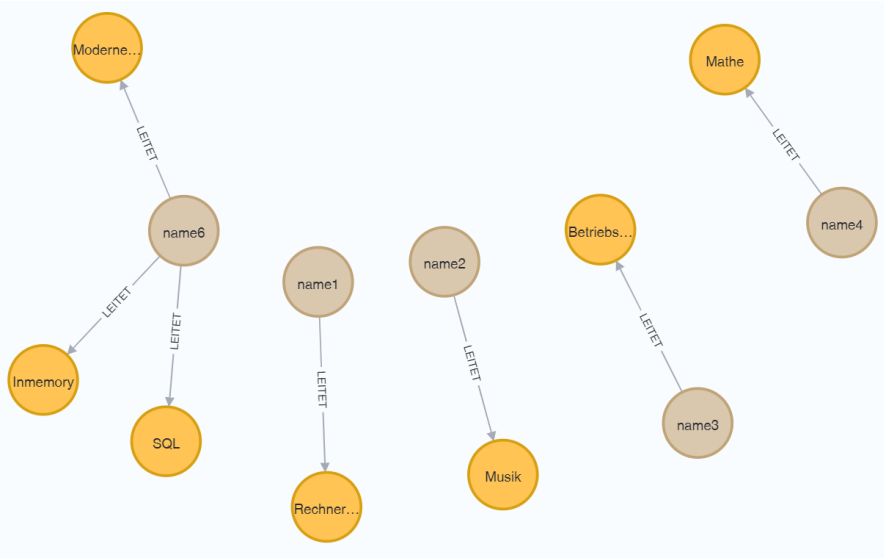
Graph-Datenbanken

Umsetzung der Datenbank auf Neo4J



Umsetzung der Datenbank auf Neo4J

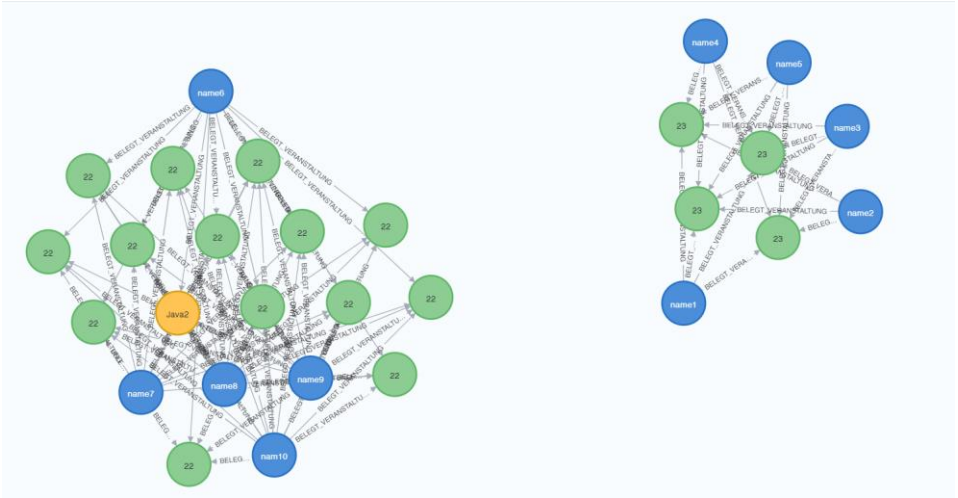
- Dozent
- Termin
- Veranstaltung
- Modul
- Student
- ~~Historie~~



Entity > Label

Umsetzung der Datenbank auf Neo4J

- VertretenderDozent
- ~~Termin2Gruppe~~
- BelegteVeranstaltung
- Anwesenheit



Releation > Beziehungen

Knoten und Beziehungen

Knoten

- Dozent
- Termin
- Modul
- Veranstaltung
- Student

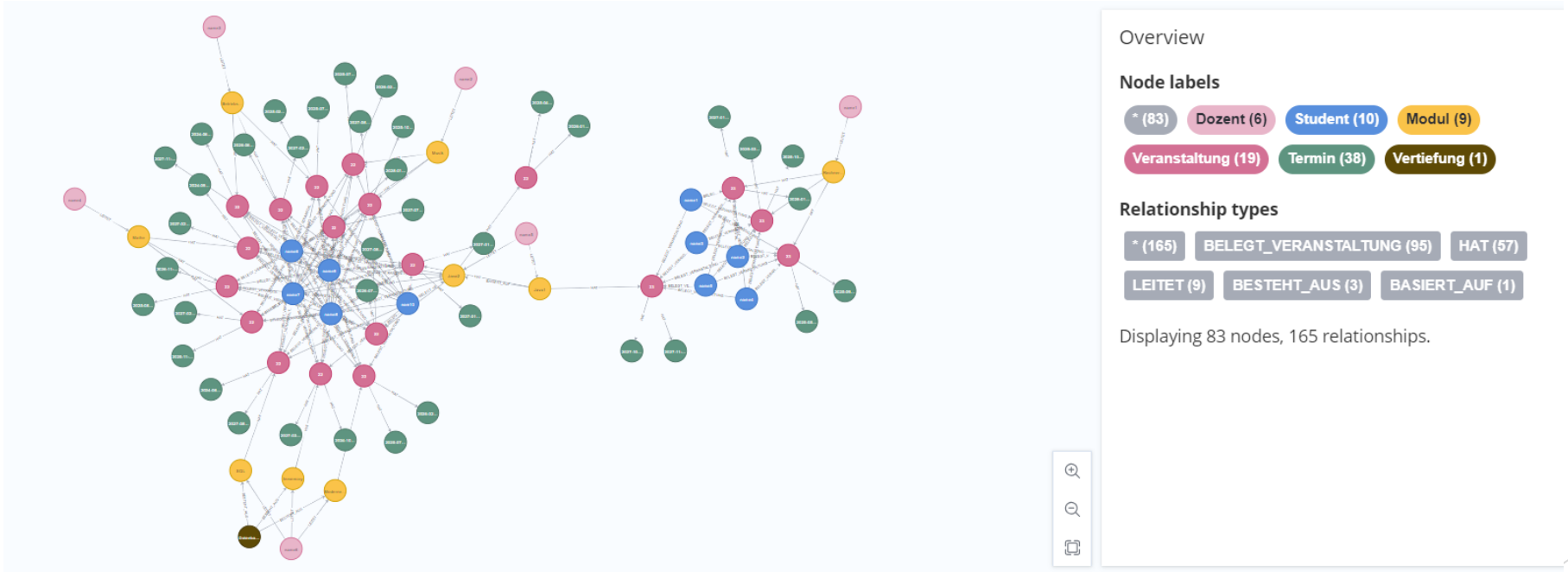
Beziehungen

- VERTRITT_IN
- BELEGT_VERANSTALTUNG
- NICHT_ANWESEND
- HAT
- LEITET

Erweiterung des Datenbankmodells

- Aufeinander aufbauende Module
  - Module bauen auf einander auf
  - Neue Beziehung: BASIERT_AUF
- Vertiefungen
  - Eine Vertiefung ist eine Gruppierung von Modulen
  - Neuer Knotentyp: Vertiefung
    - Besitzt einen Namen
  - Neue Beziehung: BESTEHT_AUS
- Hilfe unter Studenten
  - Studenten können Studenten sehen, die das Modul bereits abgeschlossen haben
  - Keine Änderung bzw. Erweiterung notwendig

Daten



Overview

Node labels

* (83) | Dozent (6) | Student (10) | Modul (9) | Veranstaltung (19) | Termin (38) | Vertiefung (1)

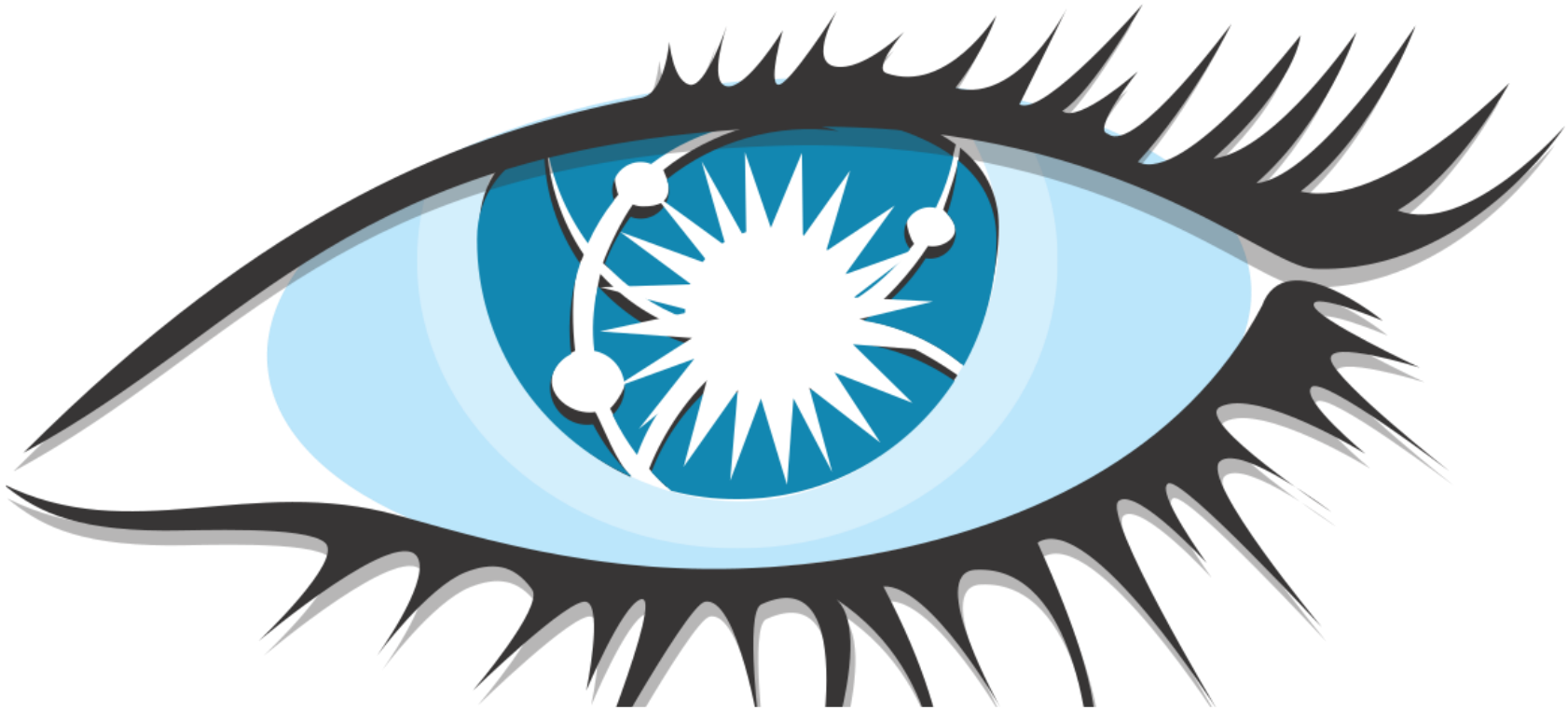
Relationship types

* (165) | BELEGT_VERANSTALTUNG (95) | HAT (57) | LEITET (9) | BESTEHT_AUS (3) | BASIERT_AUF (1)

Displaying 83 nodes, 165 relationships.

Fazit

- Wenig Anwendungsfälle für Neo4J
- Viele Beziehungen (unübersichtlich in der Darstellung)
- MySQL ist in den meisten Anwendungsfällen vermutlich schneller.
- Gefundener Anwendungsfall wurde dazu erfunden und wird wahrscheinlich gar nicht gebraucht.
- Würden wir nicht benutzen.



Cassandra

Spaltenorientierte Datenbanken

Anwendungsfälle

- Eine Tabelle wird in Cassandra um ein Query herumgebaut
- Für diese Query ist die Tabelle sehr performant
- Ein Student möchte seine Termine einsehen
  - Datum, Beginn, Ende, Bezeichnung, Typ, Dozent, Teilnahmestatus
- Ein Dozent möchte seine Termine einsehen
  - Datum, Beginn, Ende, Bezeichnung, Typ

Tabellen

- Daten über alle Knoten gleichmäßig verteilen
- Von so wenig Partitionen lesen wie möglich
- Studenten Tabelle für Termine
  - 60 Termine pro Student pro Semester
  - 10 Semester wären 600 Termine pro Student
- Dozenten Tabelle für Termine
  - Wir gehen von 300 Terminen pro Dozent pro Semester aus.
  - Daten sollen nur für 2 Semester in Cassandra liegen
  - 600 Termine pro Partition

Tabellen

Studenten-Termin Tabelle
CREATE TABLE student_termine (student_id int, termin_id int, datum date, beginn time, ende time, bezeichnung text, typ text, dozent text, teilnahmestatus text, PRIMARY KEY((student_id), termin_id));
Dozenten-Termin Tabelle:
CREATE TABLE dozent_termine (dozent_id int, termin_id int, datum date, beginn time, ende time, bezeichnung text, typ text, PRIMARY KEY((dozent_id), termin_id));

Schreib- und Leseoperation

- Redundanz
  - Schreiben ist deutlich schneller
  - Mehr schreiben um schneller zu lesen
  - Ein Jahrgang hat 30 Studenten
  - Ein Termin wird 31 mal persistiert
- ConsistencyLevel
  - ONE reicht aus, weil viel Zeit zwischen Lesen und Schreiben liegt

Konsistente Daten

- Die 31 Termine müssen konsistent persistiert werden
- BATCH Statement
  - Sammlung von Statements
  - Ganz oder gar nicht
  - Konsistenz wird sicher gestellt
  - Mehr Partitionen = langsamer
- Alternative
  - Alle Termine asynchron einzeln ausführen und bei einem Fehler erneut versuchen

Insert

```
BEGIN BATCH
INSERT INTO stundenplan.dozent_termine(dozent_id, termin_id, datum, beginn, ende, bezeichnung, typ)
VALUES(1, 1, '2024-04-15', '08:00:00', '15:30:00', 'Moderne DB', 'Vorlesung');

INSERT INTO stundenplan.student_termine(student_id, termin_id, datum, beginn, ende, bezeichnung, typ, dozent)
VALUES(1, 1, '2024-04-15', '08:00:00', '15:30:00', 'Moderne DB', 'Vorlesung', 'Königsmann');

INSERT INTO stundenplan.student_termine(student_id, termin_id, datum, beginn, ende, bezeichnung, typ, dozent)
VALUES(2, 1, '2024-04-15', '08:00:00', '15:30:00', 'Moderne DB', 'Vorlesung', 'Königsmann');

INSERT INTO stundenplan.student_termine(student_id, termin_id, datum, beginn, ende, bezeichnung, typ, dozent)
VALUES(3, 1, '2024-04-15', '08:00:00', '15:30:00', 'Moderne DB', 'Vorlesung', 'Königsmann');

APPLY BATCH;
```

Tombstone

- Cassandras Art Sachen zu löschen
- Tombstone != Time To Life
- Termine eines Dozenten werden nur 2 Semester persistiert

Performance

- Angelegt wurden:
  - 700 Termine
  - 300 Studenten
  - 7 Module
  - 3 Dozenten
- Cassandra: 21700
  - Studententabelle: 21000
  - Dozenttabelle: 700
- MySQL: 22174

Cassandra Queries

Studenten Termine
SELECT datum, beginn, ende, bezeichnung, typ, dozent, teilnahmestatus FROM stundenplan.student_termine WHERE student_id = <id>
Dozenten Termine
SELECT datum, beginn, ende, bezeichnung, typ FROM stundenplan.dozent_termine WHERE dozent_id = <id>

MySQL Queries

Studenten Termine:
SELECT DISTINCT Termin.datum, Termin.beginn, Termin.ende, Modul.name, Anwesenheit.fehlgrund, Veranstaltung.typ, Dozent.name FROM Termin JOIN Veranstaltung ON Termin.veranstaltungId = Veranstaltung.veranstaltungId JOIN Modul ON Veranstaltung.modulId = Modul.modulId JOIN BelegteVeranstaltung ON Veranstaltung.veranstaltungId = BelegteVeranstaltung.veranstaltungId JOIN Dozent ON Veranstaltung.dozentId = Dozent.dozentId LEFT JOIN Anwesenheit ON BelegteVeranstaltung.matrikelnummer = Anwesenheit.matrikelnummer AND Termin.terminId = Anwesenheit.terminId WHERE BelegteVeranstaltung.matrikelnummer = <matrikelnummer>;

MySQL Queries

Dozenten Termine:
SELECT Termin.datum, Termin.beginn, Termin.ende, Modul.name, Veranstaltung.typ FROM Termin JOIN Veranstaltung ON Termin.veranstaltungId = Veranstaltung.veranstaltungId JOIN Modul ON Veranstaltung.modulId = Modul.modulId WHERE Veranstaltung.dozentId = <dozentId>;

Performance

Test	Cassandra	MySQL
Testdaten anlegen	2500 - 6000ms	3500 - 3850ms
Alle Termine für einen Studenten abfragen (70 Termine)	3 - 5ms	5 - 15ms
Alle Termine für einen Dozenten abfragen	5 - 6ms	3- 6ms

Beobachtungen

Cassandra

- Inkonsistente Schreibgeschwindigkeit
- Batches sind langsam
- Queries sind kurz und schnell
- Cassandra ist gut skalierbar
- Das Cluster muss verwaltet werden

MySQL

- Konsistente Schreibgeschwindigkeit
- Viele Joins sind langsam
  - Geschwindigkeit im allgemeinen ist okay
- Nicht gut skalierbar

Fazit

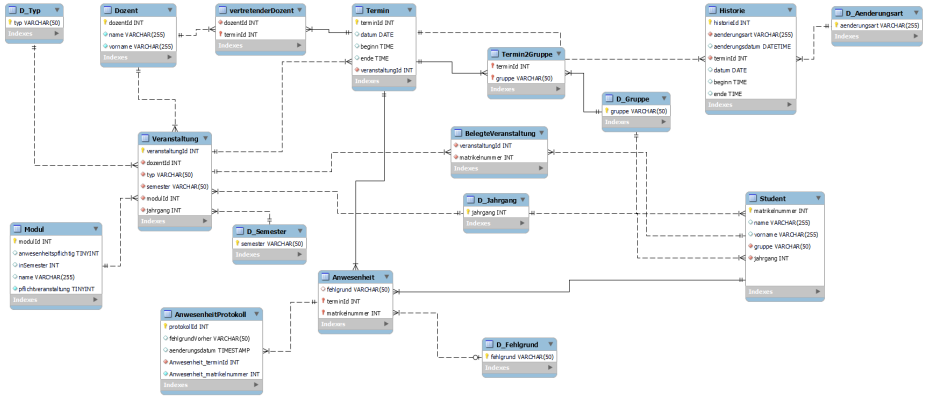
- Cassandra kann für die Anwendungsfälle gut benutzt werden
- Es gibt aber keine Notwendigkeit
- MySQL ist langsamer aber weniger aufwendig zu verwalten
- Wir würden anhand der Größe des Projektes entscheiden, ob wir Cassandra benutzen.

CouchDB

CouchDB

Dokumentorientierte Datenbanken

Ziel:



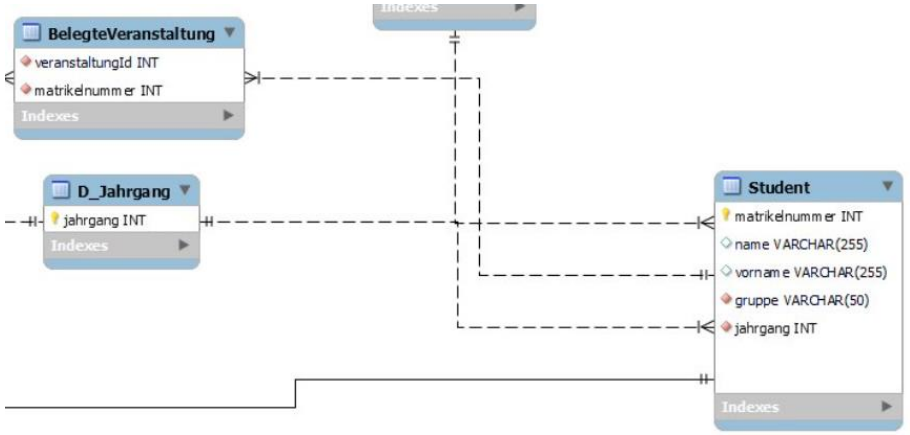
{.json}
{.json}
{.json}

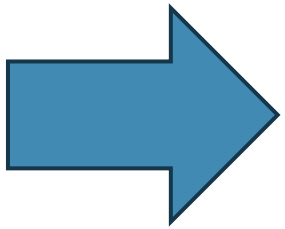
Relationales Modell in eine Menge an Dokumenten umwandeln

Prämissen:

1. Redundanzen sind nicht schlimm
2. Konsistenzen werden nicht über das Modell abgefangen
3. Da es keine Joins gibt, müssen die Daten entsprechend strukturiert sein

Beispiel: Student

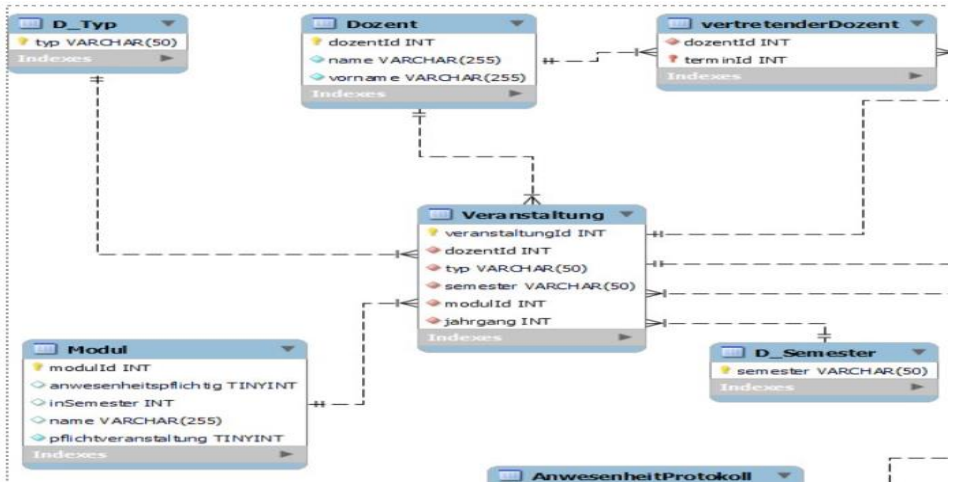


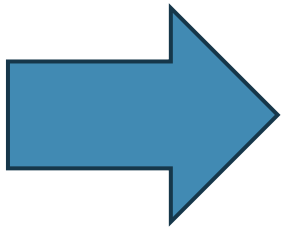



```
{
  "_id": "5ed5663ada0c0fd7dd81d8dbce12dd59",
  "_rev": "1-fa1c8e80ada453de1f211a85032f41f1",
  "typ": "student",
  "name": "Blauschuh",
  "vorname": "Bubi",
  "matrikelnummer": "123456",
  "gruppe": "A-F",
  "jahrgang": "23",
  "belegteVeranstaltungen": [
    "Internetbasierte Anwendungen"
  ]
}
```


Informationen aus 3 Tabellen werden in ein Dokument geschrieben

Beispiel: Semester

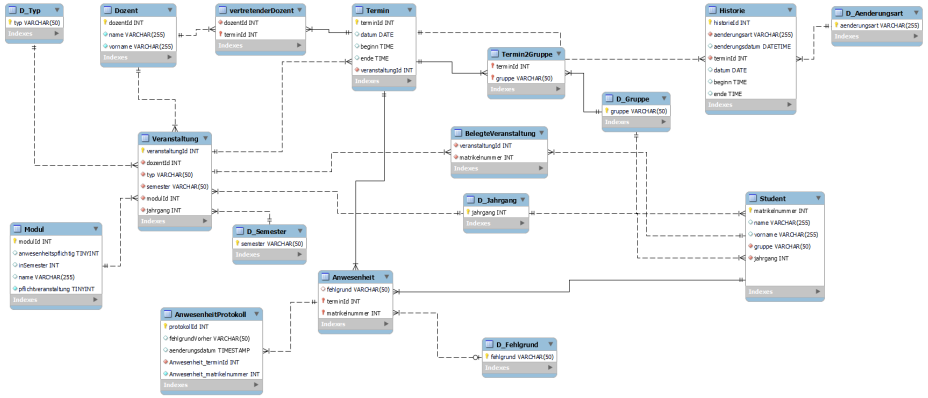





```
{
  "_id": "76ee891ed52bd004b8b677292a007df1",
  "_rev": "2-beb5e927e044400706ce82a4f9a879de",
  "typ": "semester",
  "name": "WS2023/2024",
  "veranstaltungen": [
    {
      "veranstaltungId": "1",
      "dozent": {
        "name": "Müller",
        "vorname": "Anna"
      },
      "typ": "Vorlesungen",
      "modul": {
        "modulId": "1",
        "name": "Internetbasierte Anwendungen",
        "anwesenheitspflichtig": true,
        "pflichtveranstaltung": true
      },
      "jahrgang": "23"
    },
    {
      "veranstaltungId": "4",
      "dozent": {
        "name": "Müller",
        "vorname": "Anna"
      },
      "typ": "Übungen",
      "modul": {
        "modulId": "1",
        "name": "Internetbasierte Anwendungen",
        "anwesenheitspflichtig": true,
        "pflichtveranstaltung": true
      },
      "jahrgang": "23"
    }
  ]
}
```


Informationen aus 6 Tabellen werden in ein Dokument geschrieben

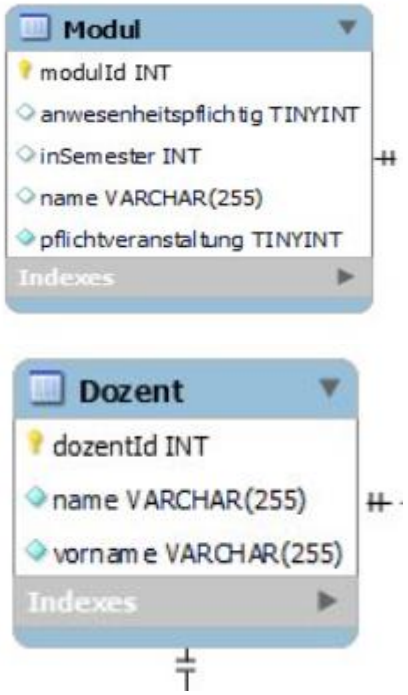
Beispiel: Veranstaltung

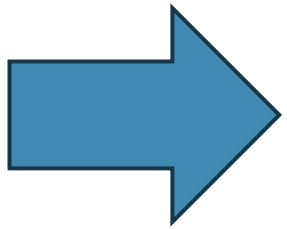



```
{
  "_id": "76ee891ed52bd004b8b677292a015a57",
  "_rev": "2-d5b2ef3818d5aebea2ece477f961f818",
  "typ": "Vorlesung",
  "datum": "1.10.2023.",
  "beginn": "08:00:00",
  "ende": "10:00:00",
  "veranstaltungname": "Internetbasierte Anwendungen",
  "dozent": {
    "name": "Anna Müller"
  },
  "gruppe": "A-F",
  "historie": [
    {
      "aenderungsart": "Verschiebung",
      "aenderungsdatum": "30.9.2023 12:00",
      "neuesDatum": "2.10.2023.",
      "neuerBeginn": "08:00:00",
      "neuesEnde": "10:00:00"
    }
  ],
  "anwesenheit": [
    {
      "matrikelnummer": "123456",
      "name": "Bubi Blauschuh",
      "status": "anwesend"
    },
    {
      "matrikelnummer": "456789",
      "name": "Thomas Königsmann"
    }
  ]
}
```


Dieses Dokument vereinigt semantische Informationen aus der gesamten Datenbank

Beispiel: Modul & Dozent





Modul wird 1:1 abgebildet, der Dozent setzt sich aus zwei Tabellen zusammen

Views und Reduce-Funktionen

View	Reduce-Funktion
Alle Studenten der Gruppe A-F	
Alle ausgefallenen Termine (zum Nachholen)	Anzahl der noch nachzuholenden Termine
Alle bislang nicht entschuldigten Fehlzeiten	Anzahl der nicht entschuldigten Fehlzeiten
Alle Studenten	Anzahl der Studenten
Alle Termine nach Dozenten	
Alle Veranstaltungen zu IBA	

Sinnvolle Erweiterungen

1. Nachhalten der Social Points und der zugehörigen Messen, Werbeaktionen, ...
2. Raumnutzungsplan
3. Erweiterungen der Termininformationen, z.B. Termine für besondere Inhalte vormerken((Probe)klausur, Wiederholung,…)

Performance - CRUD

Test	Dauer
Dokument anlegen	127 ms
Dokument lesen	61 ms
Dokument bearbeiten	38 ms
Dokument löschen	20 ms

Performance-Vergleich

Test	JDBC(mySQL)	JDBC(CouchDB)
100 Studenten einzeln erstellen	444 ms	912 ms
100 Studenten einzeln löschen	600 ms	1223 ms
100 Studenten in wenigen aber großen Statements erstellen	20 ms	28 ms
100 Studenten in wenigen aber großen Statements löschen	10 ms	17 ms

Beobachtungen:

1. In den Einzeloperationen dauert das Erstellen mit Abstand am längsten
2. Bei der Massenoperation dauert das Löschen mit Abstand am längsten
3. Im Allgemeinen ist MySQL bzw. dessen Java-Schnittstelle performanter
4. Die Schemagebundenheit von MySQL verträgt sich gut mit Javas Klassengebundenheit
5. Die Nutzung ist stark Use-Case abhängig, man muss sich anhand der Operationen orientieren.
6. CouchDB löscht nicht, es markiert als gelöscht

Impressum

Nils Thorben Konopka

nils-thorben.konopka@itc-studenten.de

Rodrigo Galarza

i.quiroz-galarza@itc-studenten.de

Lukas Cornel Meier

lukas.meier@itc-studenten.de