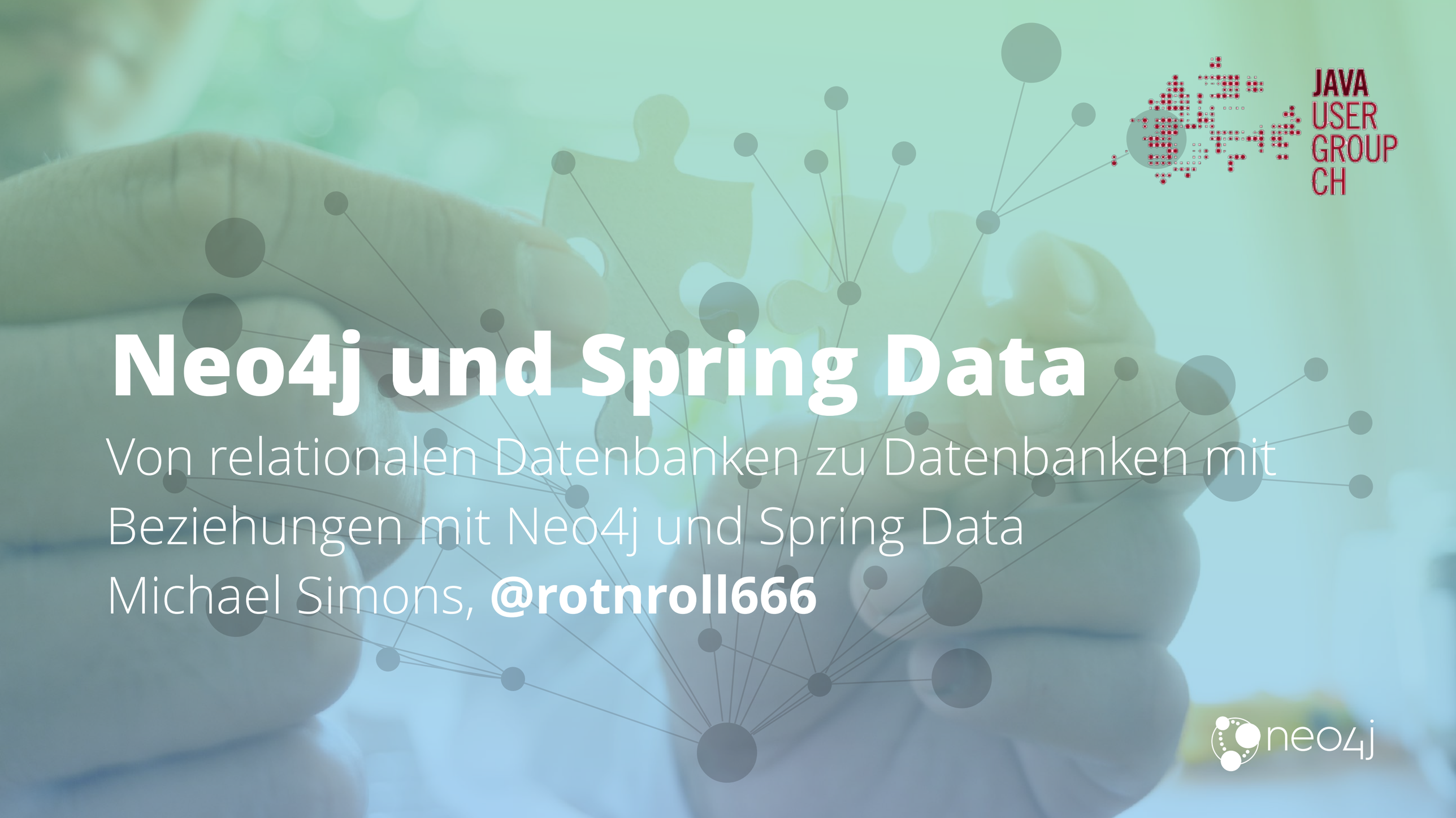


# Neo4j und Spring Data

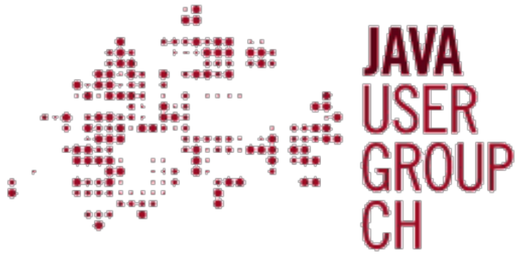
Von relationalen Datenbanken zu Datenbanken mit Beziehungen mit Neo4j und Spring Data

Michael Simons, **@rotnroll666**

JAVA USER GROUP CH

neo4j

# Agenda




- Über Neo4j
- Meine Geschäftslogik
- Neo4j mit Daten füllen
- Auf der JVM mit Neo4j zu kommunizieren
- Spring Data Neo4j
- Einige fortgeschrittene Abfragen

# Über Neo4j

neo4j

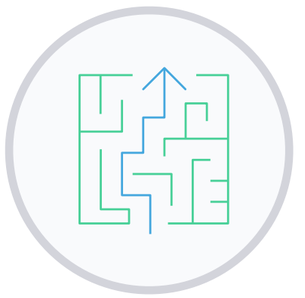
# Neo4j

## Mindset

“Graph Thinking” is all about considering connections in data as important as the data itself.

## Native Graph Platform

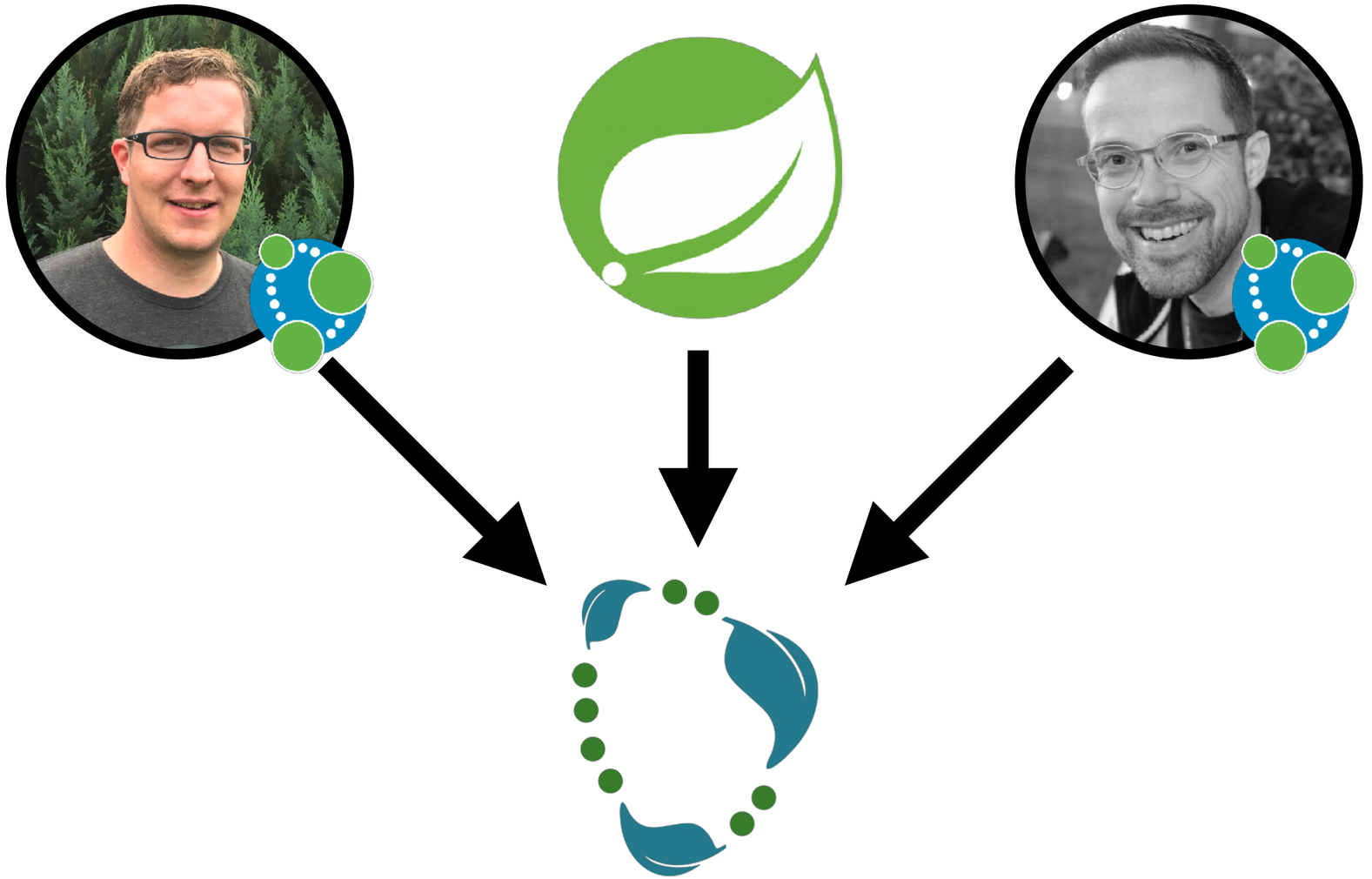
Neo4j is an internet-scale, native graph database which executes connected workloads faster than any other database management system.



## Ecosystem

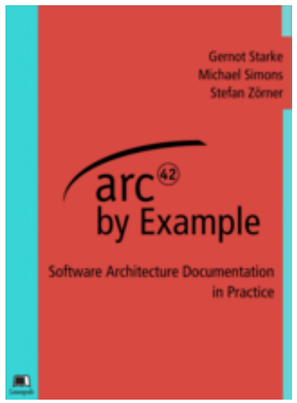
Neo4j Professional Services
300+ partners
47,000 group members
61,000 trained engineers
3.5M downloads

# Spring Data und Neo4j



# Über mich

- Neo4j seit Juli 2018
- Java Champion
- Gründer und aktueller Leiter der Java User Group **EuregJUG**
- Autor (**Spring Boot 2** und **Arc42 by example**)

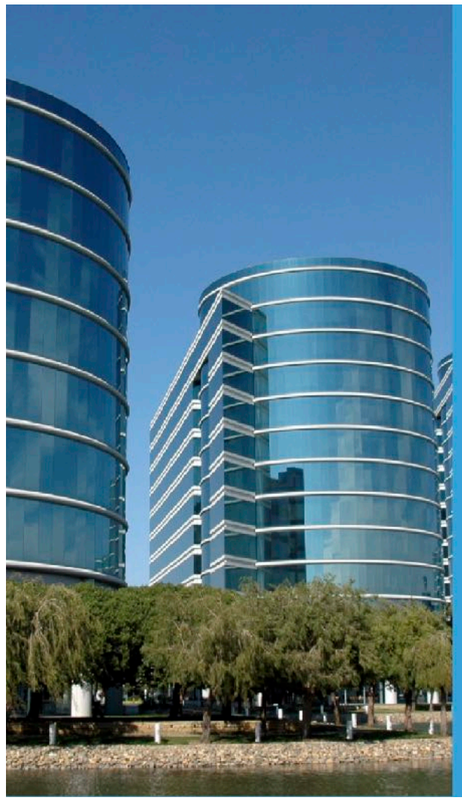

First contact to Neo4j through QAssistant

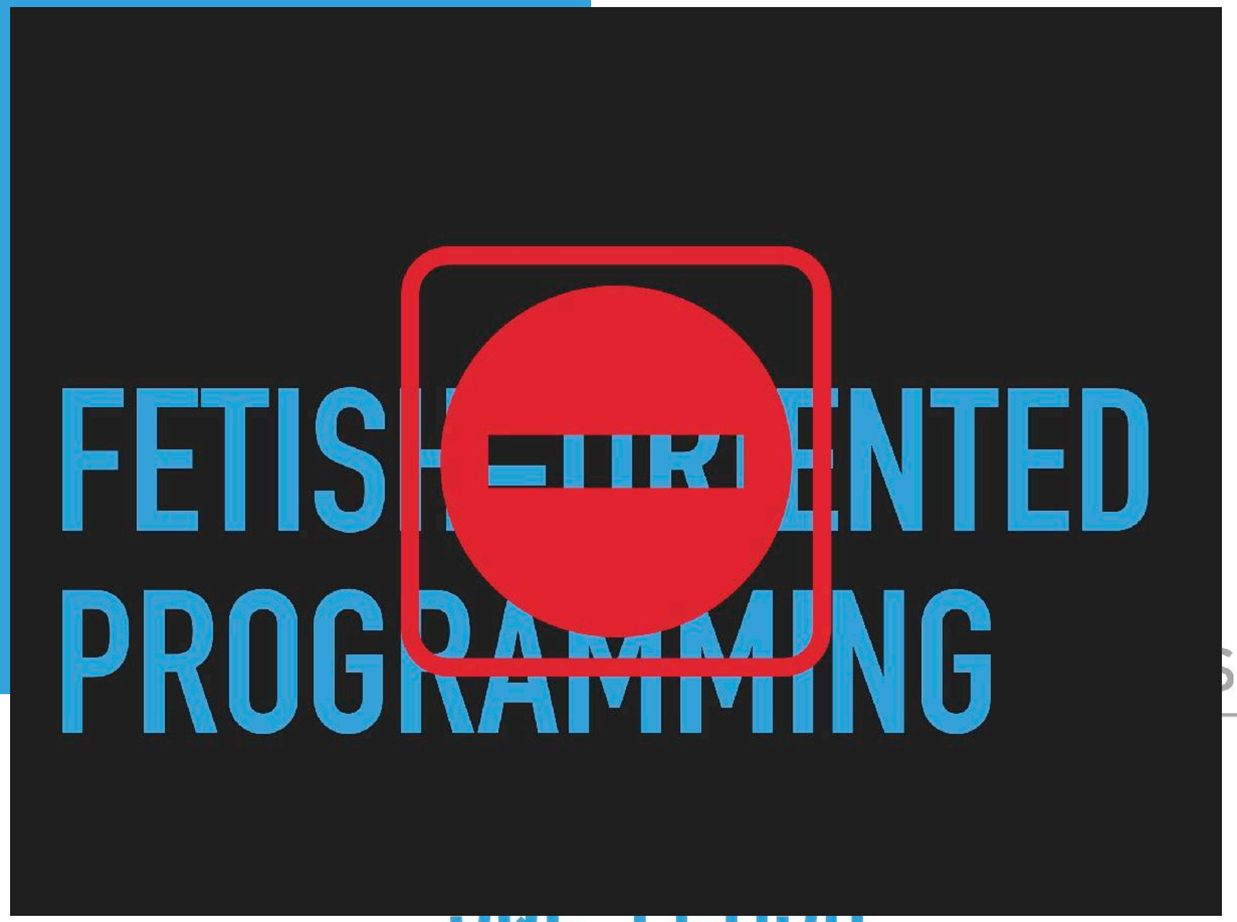
# Auch bekannt für...


JOIN 2017 AT ORDINA JWORKS

LIVE WITH YOUR
SQL-FETISH

# Auch bekannt für…

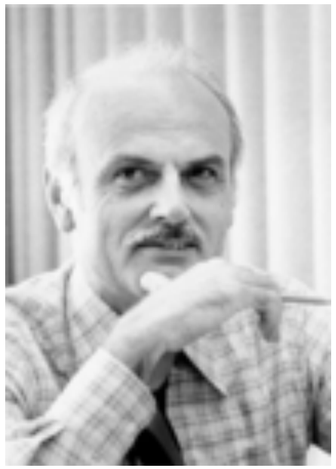


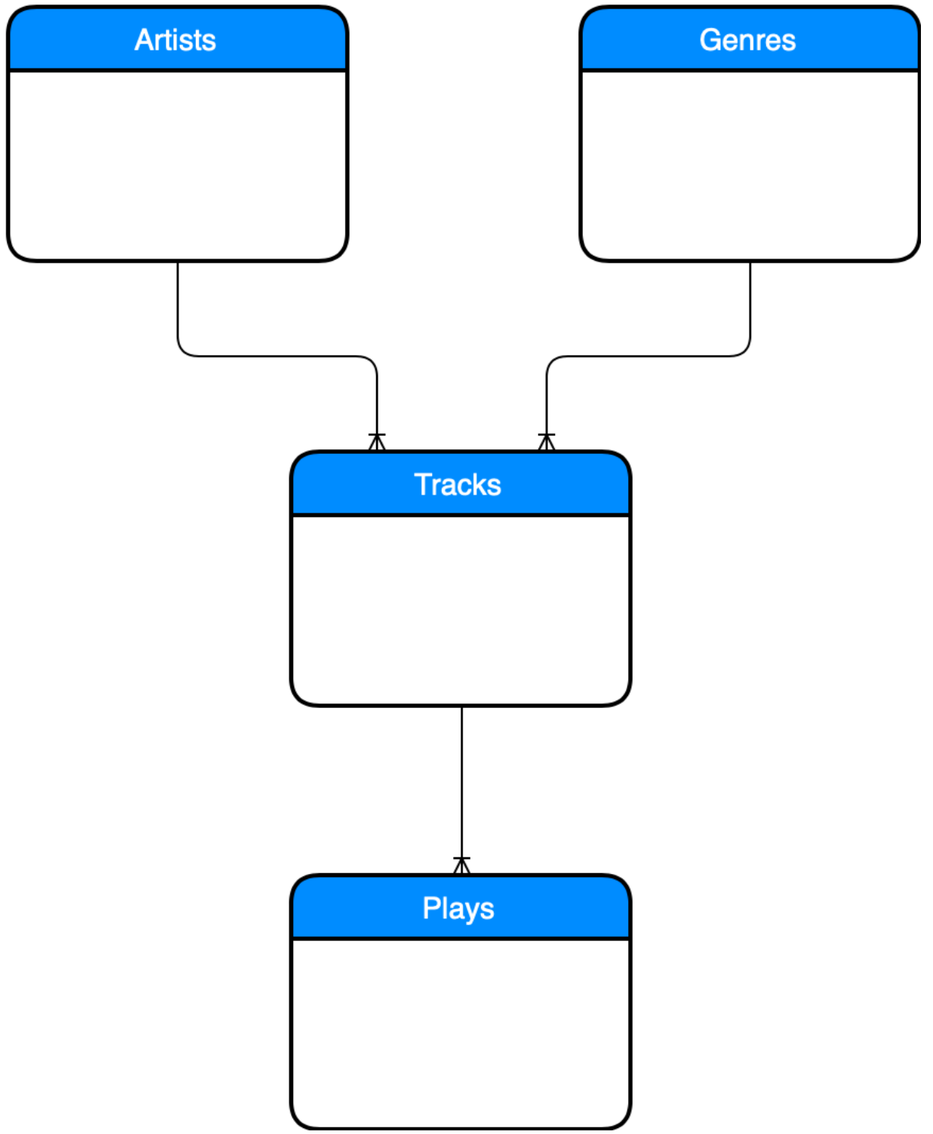


# Meine Geschäftslogik

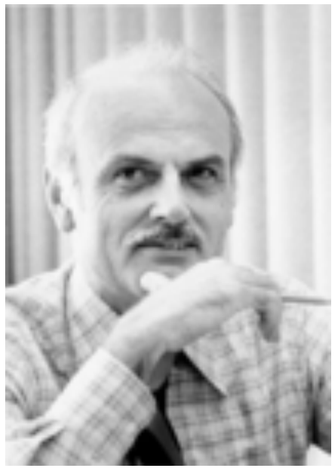
neo4j

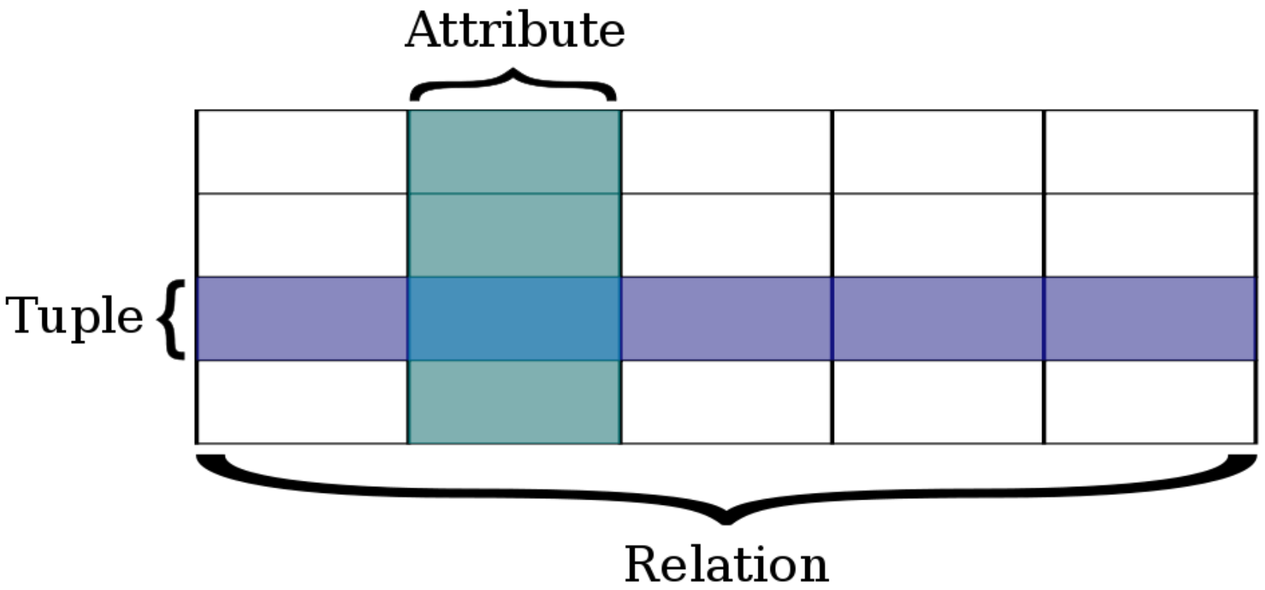
# Hörgewohnheiten





# Hörgewohnheiten





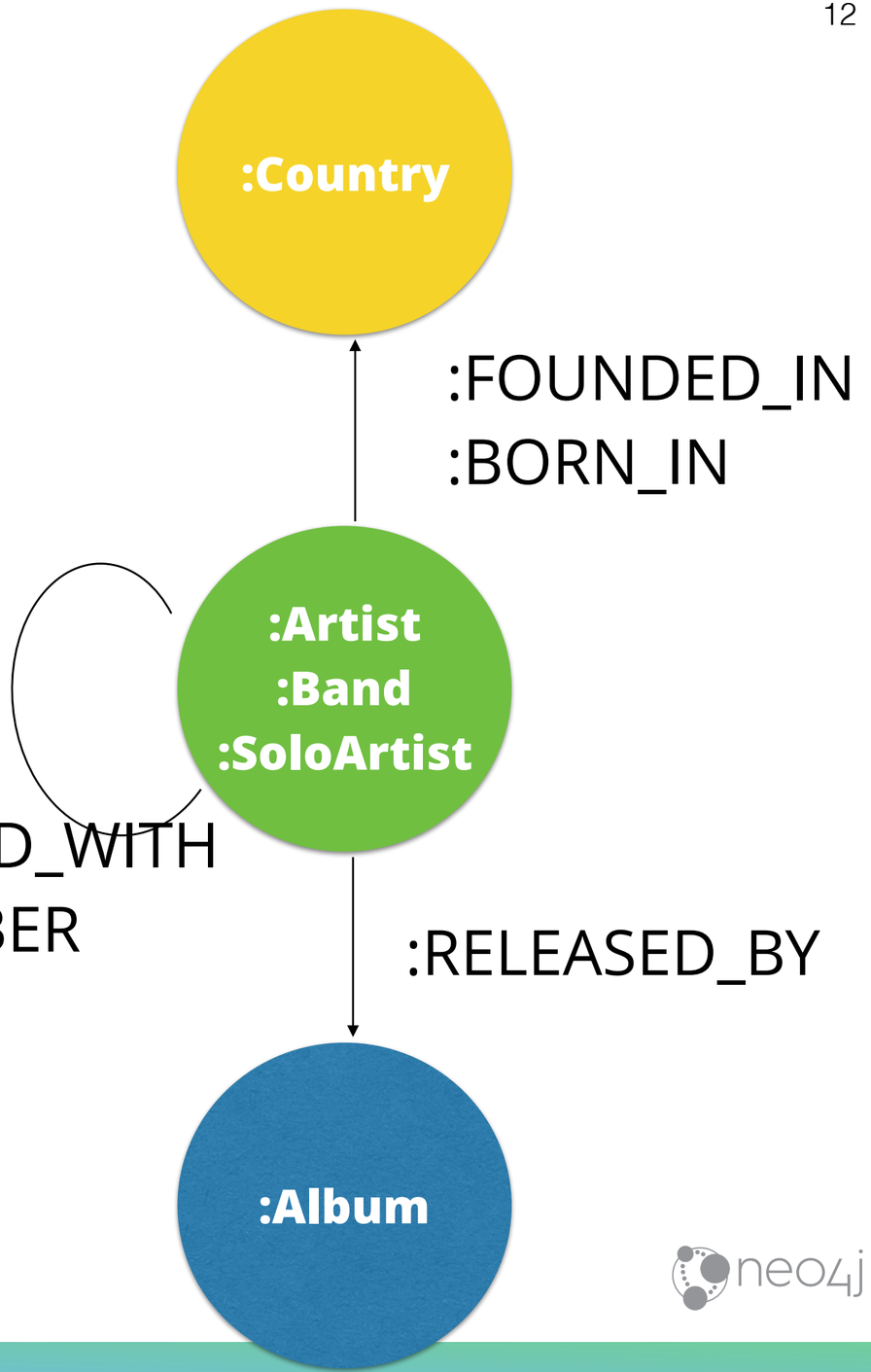
# Logisches vs physikalisches Model

- Logisches Model als ER-Diagram entworfen
- Dann beginnt die Normalisierung:
  - Redundanzfreiheit als Ziel
  - UNF (Nicht normalisiert)
  - 1NF: Atomare Spalten
  - 2NF: + Keine teilweisen Abhängigkeiten
  - 3NF: + Keine transitiven Abhängigkeiten

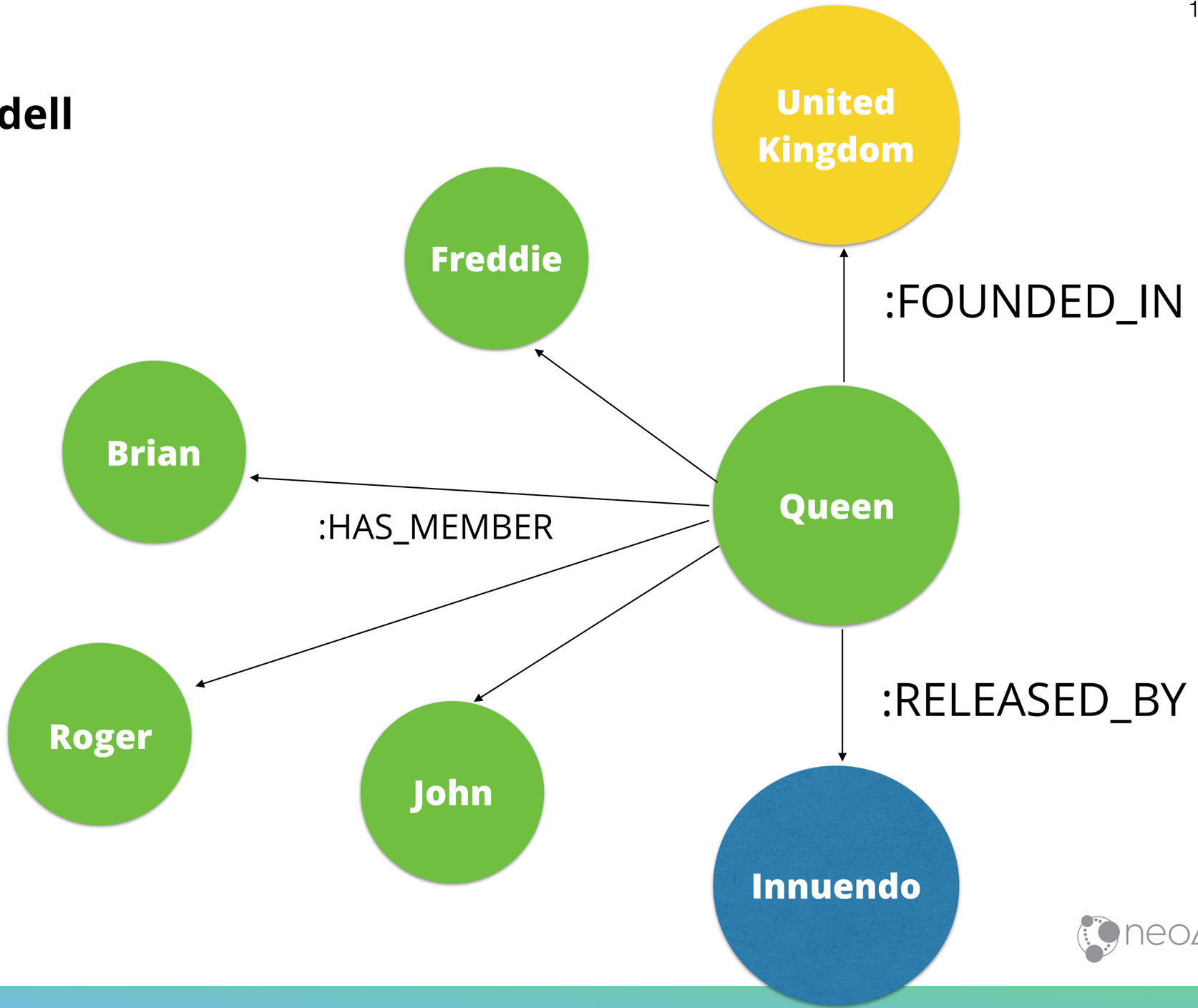
**Fremdschlüssel zwischen Tabellen sind keine Relationen!**
**Tabellen und Ergebnismengen von Abfragen sind Relationen.**

# Das „Whiteboard“ Modell entspricht dem physikalischen

- Bands wurden **in Ländern gegründet** und Solokünstler **geboren**
- Einige Künstler sind mit anderen Künstler **assoziiert** und
  Bands **haben Mitglieder**
- Künstler veröffentlichen **Alben**

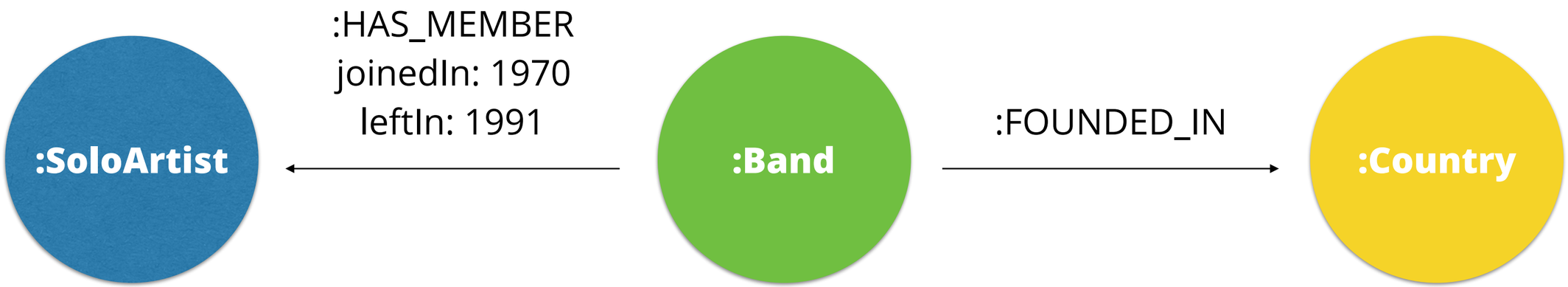


**Das „Whiteboard“ Modell entspricht dem physikalischen**



# Ein „Property Graph“

Beziehungen (Relations) verbinden Knoten und repräsentieren Handlungen (Verben)




name: Freddie
role: Lead Singer

Knoten (Nodes) repräsentieren Objekte

Knoten und Beziehungen
haben beide Eigenschaften

# Abfragen

- Cypher ist für Neo4j was SQL für relationale Datenbanken ist:
  Eine dekorative Abfragesprache
- https://www.opencypher.org / Das GQL Manifesto

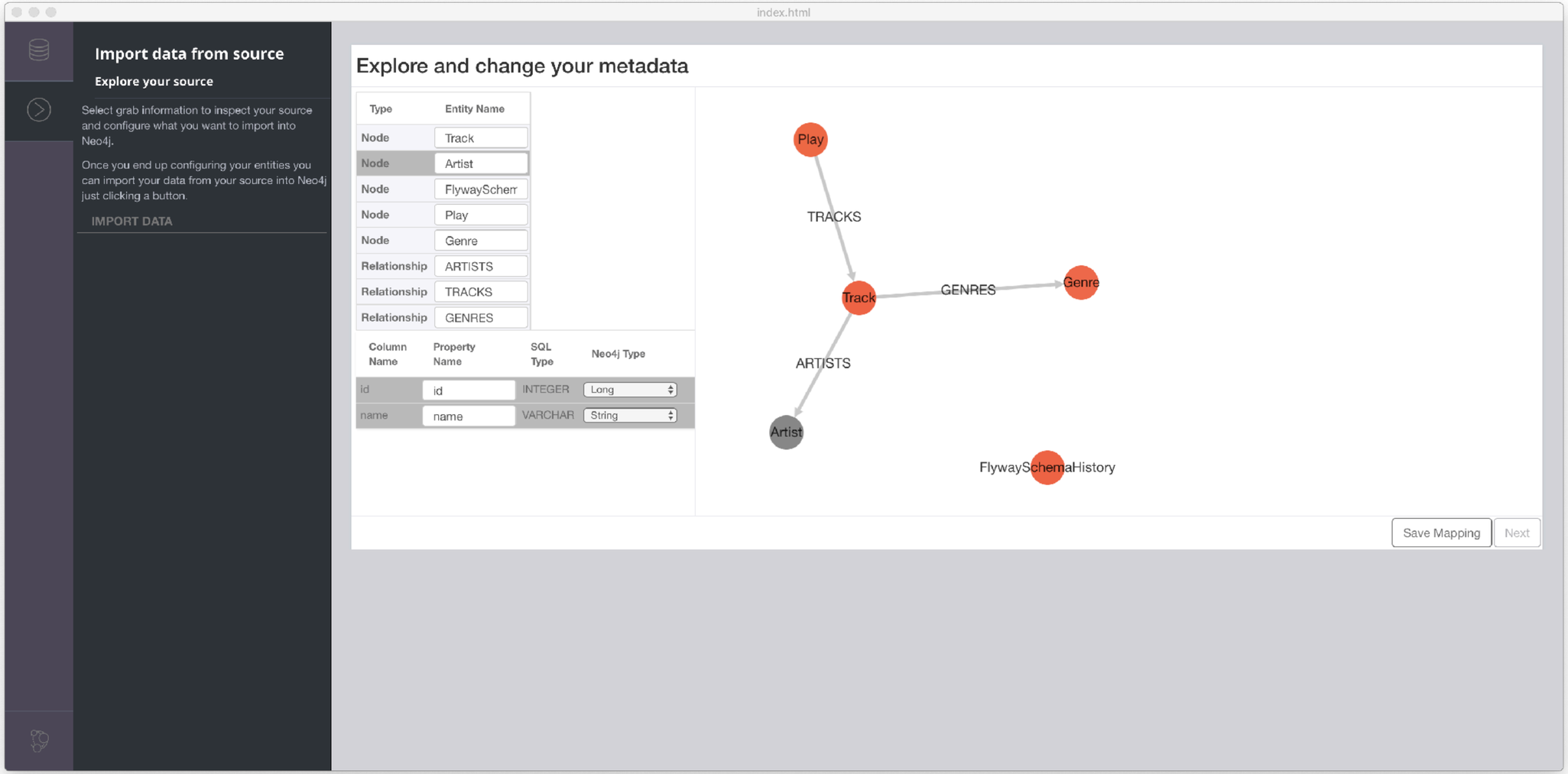
```
MATCH (a:Album) -[:RELEASED_BY]→ (b:Band),
      (c) ←[:FOUNDED_IN]- (b) -[:HAS_MEMBER]→ (m) -[:BORN_IN]→ (c2)
WHERE a.name = 'Innuendo'
RETURN a, b, m, c, c2
```

# Demo

# Neo4j mit Daten füllen

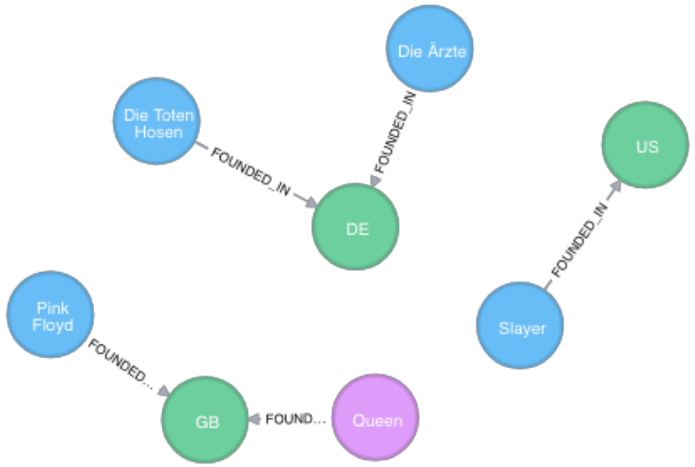
neo4j

# Das Neo4j-ETL Tool



# LOAD CSV

```
Name;Founded in
Slayer;US
Die Ärzte;DE
Die Toten Hosen;DE
Pink Floyd;GB
```




```
LOAD CSV WITH HEADERS FROM 'http://localhost:8001/data/artists.csv'
AS line FIELDTERMINATOR ';'
MERGE (a:Artist {name: line.Name})
MERGE (c:Country {code: line.`Founded in`})
MERGE (a) -[:FOUNDED_IN]→ (c)
RETURN *
```

# Eigene „stored procedures“

```
public class StatsIntegration {

  @Context public GraphDatabaseService db;

  @Procedure(name = "stats.loadArtistData", mode = Mode.WRITE)
  public void loadArtistData(
       @Name("userName") final String userName,
       @Name("password") final String password,
       @Name("url") final String url) {

    try (var connection = DriverManager.getConnection(url, userName, password);
        var neoTransaction = db.beginTx()) {

      DSL.using(connection)
           .selectFrom(ARTISTS)
           .forEach(a →
             db.execute("MERGE (artist:Artist {name: $artistName}) ", Map.of("artistName", a.getName()))
           );
      neoTransaction.success();
    } catch (Exception e) {}
  }
}
```

# APOC

- Nicht nur ein Typ aus dem Film „Matrix“

# APOC

- Nicht nur ein Typ aus dem Film „Matrix“
- Auch nicht dieser… 😉
- „A Package Of Components“ for Neo4j
- „Awesome Procedures on Cypher“

Eine Sammlung von Erweiterungen für Neo4j
https://neo4j-contrib.github.io/neo4j-apoc-procedures/

# apoc.load.jdbc

- Funktioniert für komplette Tabellen
- Oder mit eigenen SQL-Statements

# apoc.load.jdbc

```
WITH "jdbc:postgresql://localhost:5432/bootiful-music?user=statsdb-dev&password=dev" as url,
     "SELECT DISTINCT a.name as artist_name, t.album, g.name as genre_name, t.year
      FROM tracks t JOIN artists a ON a.id = t.artist_id JOIN genres g ON g.id = t.genre_id
      WHERE t.compilation = 'f'" as sql
CALL apoc.load.jdbc(url,sql) YIELD row
MERGE (decade:Decade {value: row.year-row.year%10})
MERGE (year:Year {value: row.year})
MERGE (year) -[:PART_OF]→ (decade)
MERGE (artist:Artist {name: row.artist_name})
MERGE (album:Album {name: row.album}) -[:RELEASED_BY]→ (artist)
MERGE (genre:Genre {name: row.genre_name})
MERGE (album) -[:HAS]→ (genre)
MERGE (album) -[:RELEASED_IN]→ (year)
```

# Demo

# Auf der JVM mit Neo4j zu kommunizieren

neo4j

# Verschiedene Endpunkte

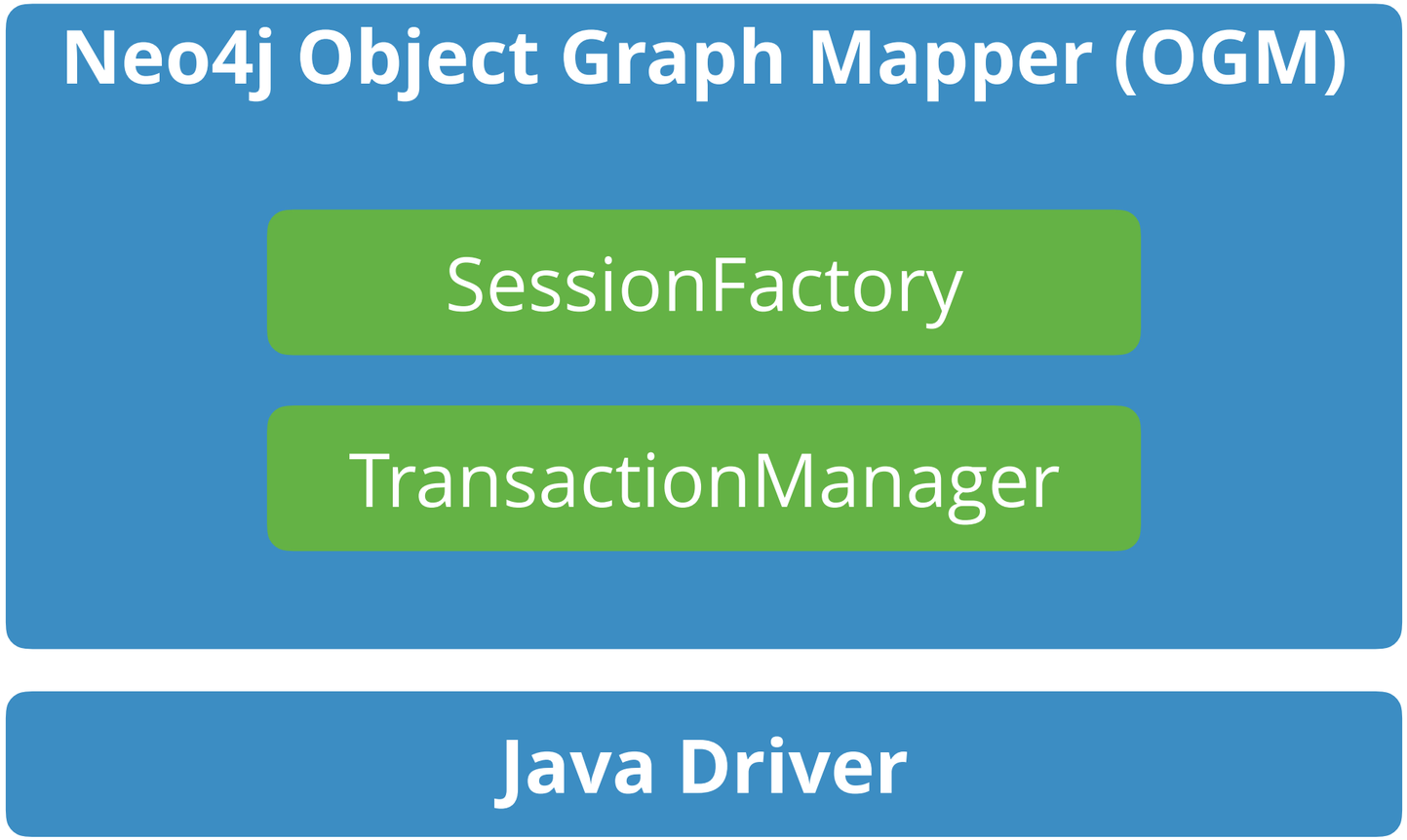
- Neo4j als eingebettete Datenbank
- Neo4j über HTTP
- Oder über das binäre **Bolt Protokoll**
  - Treiber für Java, Go, C#, Seabolt (C), Python, JavaScript

# Direkt über den Treiber

```
try (
  Driver driver = GraphDatabase.driver(uri, AuthTokens.basic(user, password));
  Session session = driver.session()
) {
  List<String> artistNames =
    session
      .readTransaction(tx → tx.run("MATCH (a:Artist) RETURN a", emptyMap()))
      .list(record → record.get("a").get("name").asString());
}
```



# Neo4j-OGM


Neo4j Object Graph Mapper (OGM)
SessionFactory
TransactionManager
Java Driver

# Neo4j-OGM

- Einheitliche Konfiguration
- Annotationen
  - Abbildung des Graphen auf die Domain
- Datenzugriff entweder
  - Domain basiert
  - Oder mit eigenen Abfragen

```java
@NodeEntity("Band")
public class BandEntity extends ArtistEntity {

  @Id @GeneratedValue
  private Long id;

  private String name;

  @Relationship("FOUNDED_IN")
  private CountryEntity foundedIn;

  @Relationship("ACTIVE_SINCE")
  private YearEntity activeSince;

  @Relationship("HAS_MEMBER")
  private List<Member> member = new ArrayList<>();
}
```

# Annotationen

```java
@RelationshipEntity("HAS_MEMBER")
public static class Member {
  @Id @GeneratedValue
  private Long memberId;

  @StartNode
  private BandEntity band;

  @EndNode
  private SoloArtistEntity artist;

  @Convert(YearConverter.class)
  private Year joinedIn;

  @Convert(YearConverter.class)
  private Year leftIn;
}
```

# Annotationen

# Zugriff über Domain-Klassen

```
var artist = new BandEntity("Queen");
artist.addMember(new SoloArtistEntity("Freddie Mercury"));

var session = sessionFactory.openSession();
session.save(artist);
```

# Zugriff über Domain-Klassen

```
var queen = session.load(BandEntity.class, 4711);

var allBands = session.loadAll(BandEntity.class);
```

# Zugriff über Domain-Klassen

```
session.delete(nickelback);

session.deleteAll(BandEntity.class);
```

# Eigene Abfragen

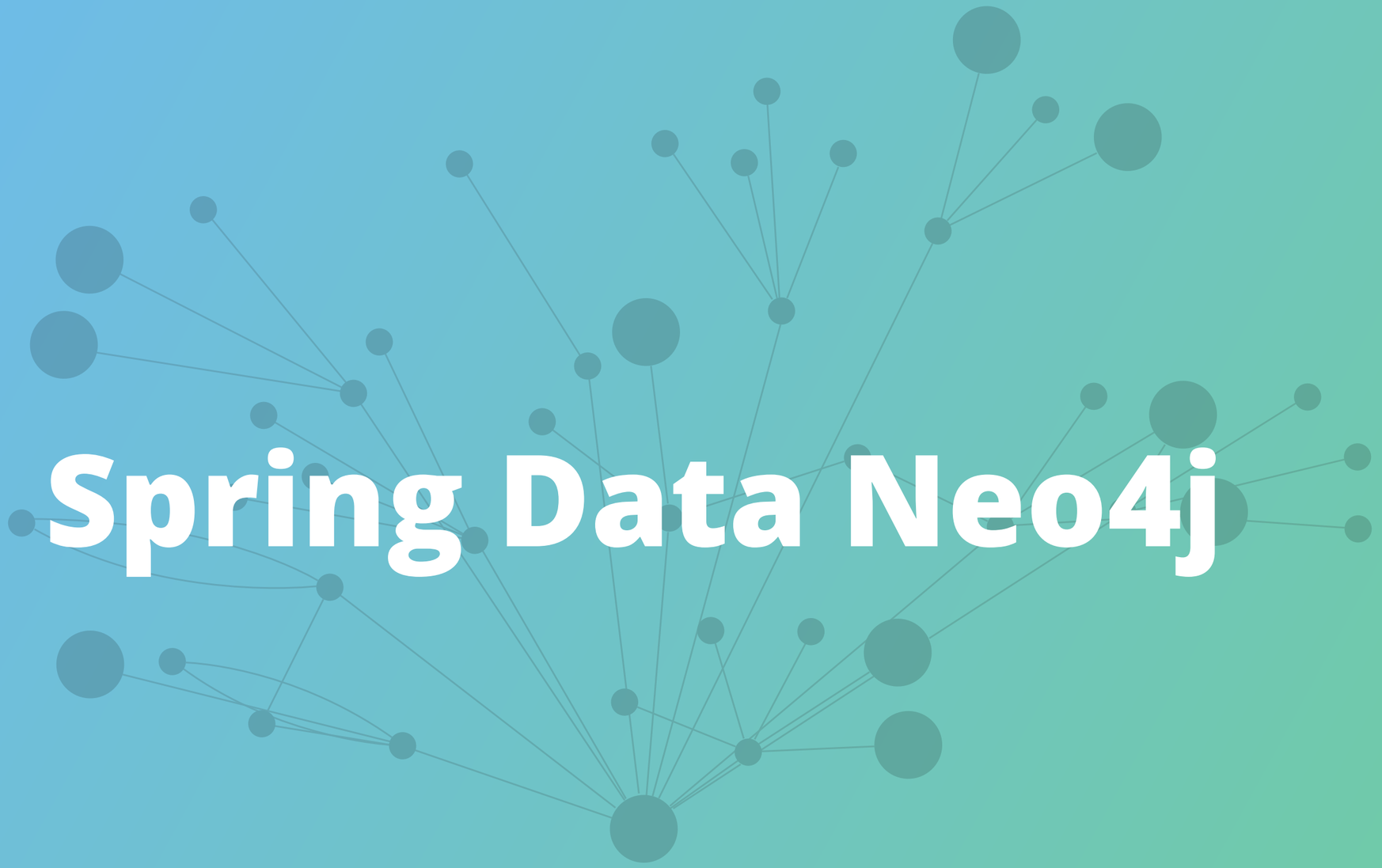
```
var britishBands = session.query(
     ArtistEntity.class,
     "MATCH (b:Band) -[:FOUNDED_IN]→ (:Country {code: 'GB'})", emptyMap());

Result result = session.query(
  "MATCH (b:Artist) ←[r:RELEASED_BY]- (a:Album) -[:RELEASED_IN]→ () -
[:PART_OF]→ (:Decade {value: $decade})"
  "WHERE b.name = $name" +
  "RETURN b, r, a",
  Map.of("decade", 1970, "name", "Queen")
);
```

# Funktioniert mit

- „Plain“ Java
- Micronaut
- Spring
- Spring Boot

# Spring Data Neo4j



neo4j

# Spring Data Neo4j

- Sehr frühes Spring Data Module
  - First Version ~2010 (Emil Eifrem, Rod Johnson)
- Basiert vollständig auf Neo4j-OGM
- Community-Modul, aber Teil des Spring Data Release-Train
- Integriert in Spring Boot

# Spring Data Neo4j

- Kann ohne
  - Wissen über den Store
  - und Cypher genutzt werden
- Oder „Graph aware“
  - Insbesondere begrenzte Fetch-Tiefe
  - Mit eigenen Cypher-Abfragen

# Zugriff über Repository-Klassen

```
interface BandRepository extends Repository<BandEntity, Long> {
}
```

# Zugriff über Repository-Klassen

```
interface BandRepository extends Neo4jRepository<BandEntity, Long> {
}
```

- CRUD Methods
  - (save, findById, delete, count)
- Supports `@Depth` annotation as well as `depth` argument

# Zugriff über Repository-Klassen

```
var artist = new BandEntity("Queen");
artist.addMember(new SoloArtistEntity("Freddie Mercury"));

artist = bandRepository.save(artist);
```

# Zugriff über Repository-Klassen

```
var artist = bandRepository.findByName("Nickelback")

artist.ifPresent(bandRepository::delete);
```

# „Derived finder“ Methoden

```
interface AlbumRepository extends Neo4jRepository<AlbumEntity, Long> {

  Optional<AlbumEntity> findOneByName(String x);

  List<AlbumEntity> findAllByNameMatchesRegex(String name);

  List<AlbumEntity> findAllByNameMatchesRegex(
    String name, Sort sort, @Depth int depth);

  Optional<AlbumEntity> findOneByArtistNameAndName(
     String artistName, String name);
}
```

# Eigene Abfragen

```
interface AlbumRepository extends Neo4jRepository<AlbumEntity, Long> {
  @Query(value
    = " MATCH (album:Album) - [:CONTAINS] → (track:Track)"
    + " MATCH p=(album) - [*1] - ()"
    + " WHERE id(track) = $trackId"
    + "   AND ALL(relationship IN relationships(p) "
    + "               WHERE type(relationship) <> 'CONTAINS')"
    + " RETURN p"
  )
  List<AlbumEntity> findAllByTrack(Long trackId);
}
```

# POJO-Results (Projektionen)

```
@QueryResult
public class AlbumTrack {
  private Long id;

  private String name;

  private Long discNumber;

  private Long trackNumber;
}
```

# POJO-Results (Projektionen)

```
interface AlbumRepository extends Neo4jRepository<AlbumEntity, Long> {
  @Query(value
    = " MATCH (album:Album) - [c:CONTAINS] → (track:Track) "
    + " WHERE id(album) = $albumId"
    + " RETURN id(track) AS id, track.name AS name, "
    + "          c.discNumber AS discNumber, c.trackNumber AS trackNumber"
    + " ORDER BY c.discNumber ASC, c.trackNumber ASC"
  )
  List<AlbumTrack> findAllAlbumTracks(Long albumId);
}
```

# Spring Transaktionen

```
public class ArtistService {

  @Transactional
  public void deleteArtist(Long id) {

    this.bandRepository.findById(id).ifPresent(a → {
      session.delete(a);
      session.query("MATCH (a:Album) WHERE size((a)-[:RELEASED_BY]→(:Artist))=0 DETACH DELETE a", emptyMap());
      session.query("MATCH (t:Track) WHERE size((:Album)-[:CONTAINS]→(t))=0 DETACH DELETE t", emptyMap());
    });
  }
}
```

# Spring Transaktionen

```
TransactionTemplate transactionTemplate;

return transactionTemplate.execute(t → {
  ArtistEntity artist = this.findArtistById(artistId).get();

  var oldLinks = artist.updateWikipediaLinks(newLinks);
  session.save(artist);
  oldLinks.forEach(session::delete);

  return artist;
});
```

# Spring Boot: Automatische Konfiguration

`org.springframework.boot:spring-boot-starter-neo4j`

```
spring.data.neo4j.username=neo4j
spring.data.neo4j.password=music
spring.data.neo4j.uri=bolt://localhost:7687

spring.data.neo4j.embedded.enabled=false
```

# Spring Boot: „Test-Slices“

```
@DataNeo4jTest
@TestInstance(Lifecycle.PER_CLASS)
class CountryRepositoryTest {

  private final Session session;

  private final CountryRepository countryRepository;

  @Autowired
  CountryRepositoryTest(Session session, CountryRepository countryRepository) {
    this.session = session;
    this.countryRepository = countryRepository;
  }

  @BeforeAll
  void createTestData() {}

  @Test
  void getStatisticsForCountryShouldWork() {}
}
```

# Spring Data Neo4j: Don'ts

- Nicht geeignet für Batch-Verarbeitung
- „Derived finder“ nicht missbrauchen!

  i.e. `Optional<AlbumEntity> findOneByArtistNameAndNameAndLiveIsTrueAndReleasedInValue(String artistName, String name, long year)`
- Nicht blindlings den Graphen in der Anwendung nachbauen
  - Das Graph-Model im Sinne der gewünschten Abfragen aufbauen
  - Das Domain-Model nach Anwendungs-Usecase

# Nicht blindlings den Graphen in der Anwendung nachbauen

```java
@NodeEntity("Artist")
public class ArtistEntity {

  private String name;

  @Relationship(
    value = "RELEASED_BY",
    direction = INCOMING)
  private List<AlbumEntity> albums;
}
```

```java
@NodeEntity("Album")
public class AlbumEntity {

  @Relationship("RELEASED_BY")
  private ArtistEntity artist;

  @Relationship("CONTAINS")
  private List<TrackEntity> tracks;
}

@NodeEntity("Track")
public class TrackEntity {

  @Relationship(
    value = "CONTAINS", direction = INCOMING)
  private List<AlbumEntity> tracks;
}
```

# Besserer Ansatz

```
@NodeEntity("Artist")
public class ArtistEntity {

  private String name;
}

@NodeEntity("Album")
public class AlbumEntity {

  @Relationship("RELEASED_BY")
  private ArtistEntity artist;
}

@QueryResult
public class AlbumTrack {

  private String name;

  private Long trackNumber;
}
```

```
interface AlbumRepository extends Repository<AlbumEntity, Long> {
  List<AlbumEntity> findAllByArtistNameMatchesRegex(
    String artistName,
    Sort sort);

  @Query(value
    = " MATCH (album:Album) - [c:CONTAINS] → (track:Track) "
    + " WHERE id(album) = $albumId"
    + " RETURN track.name AS name, c.trackNumber AS trackNumber"
    + " ORDER BY c.discNumber ASC, c.trackNumber ASC"
  )
  List<AlbumTrack> findAllAlbumTracks(long albumId);
}
```

# Demo


What are your top devices?
Where are your users?
Sessions by country
Sessions by device
COHORT ANALYSIS REPORT
Czechia
Slovakia

# Einige fortgeschrittene Abfragen

neo4j

# More Cypher

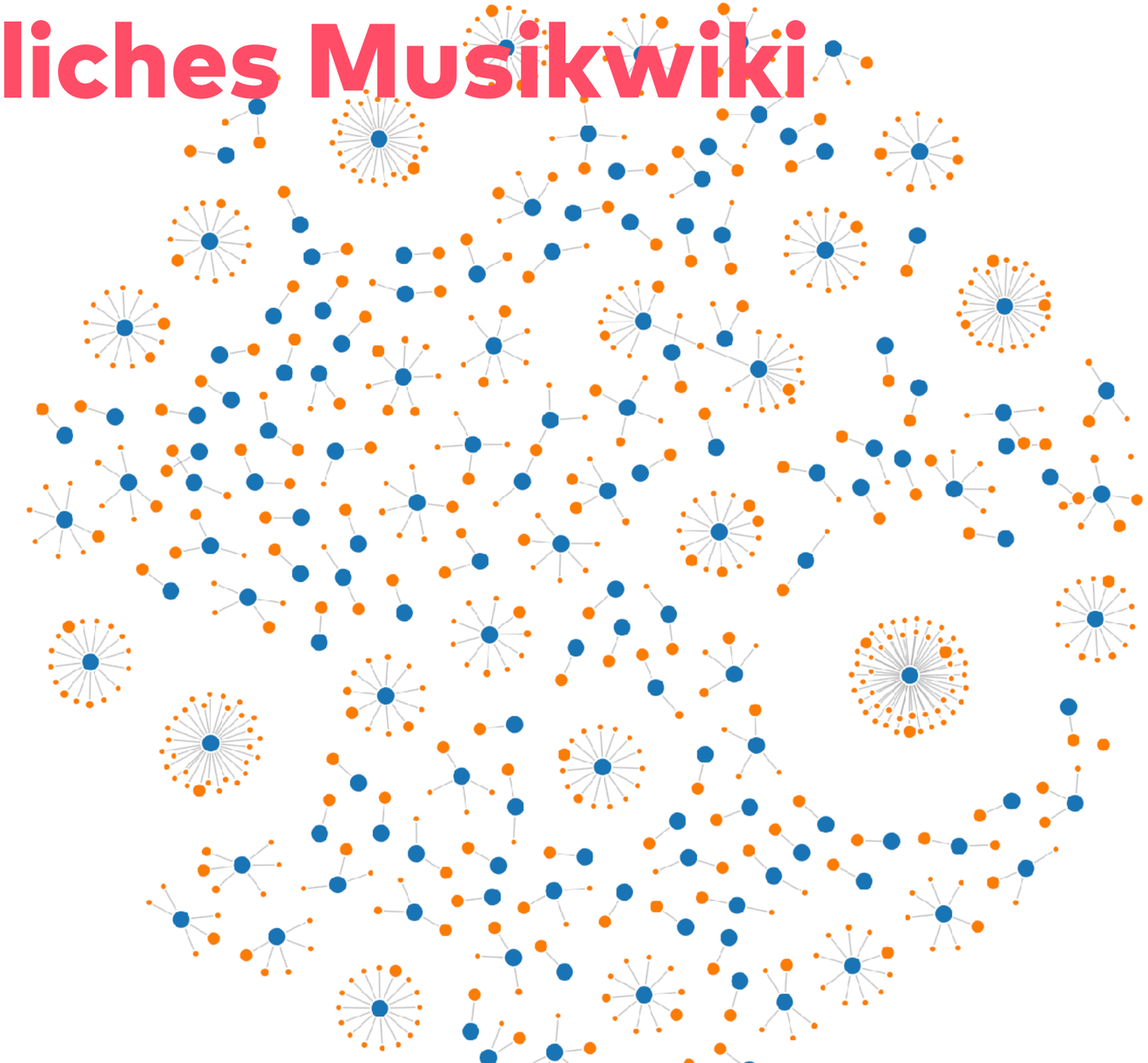
# Und nun?

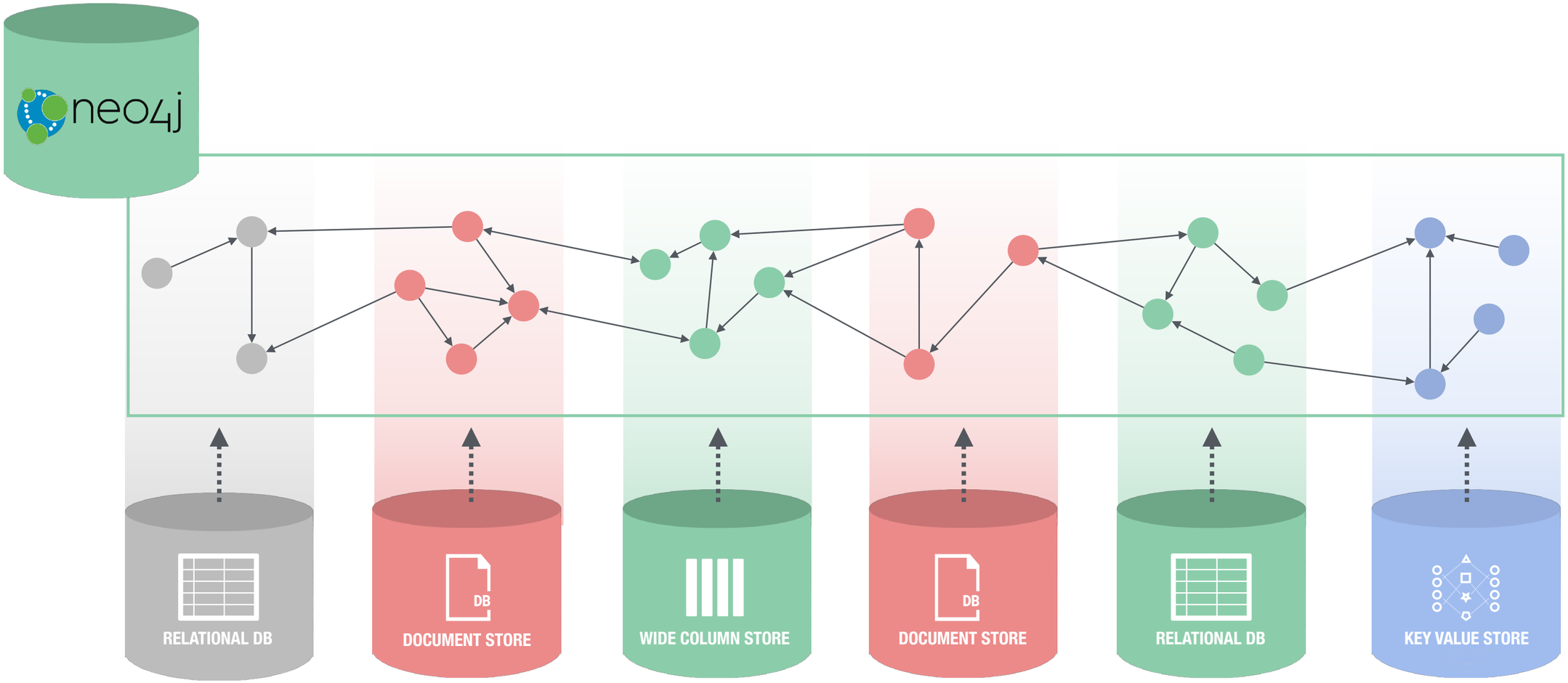
neo4j

# Genre

# Mein persönliches Musikwiki

| Name | |
|---|---|
| 200 [illegible] Roc | 1[illegible]74 |
| 2010 Germany Punk Rock | 8271 |
| 1990 Germany Punk Rock | 6919 |
| 2010 Germany Rock | 6778 |
| 2000 United Kingdom Rock | 5116 |
| 1970 United Kingdom Rock | 3554 |
| 1980 United Kingdom Rock | 3433 |
| 2010 United Kingdom Rock | 3224 |
| 2000 United Kingdom Heavy Metal | 2370 |
| 1990 United States Heavy Metal | 2314 |



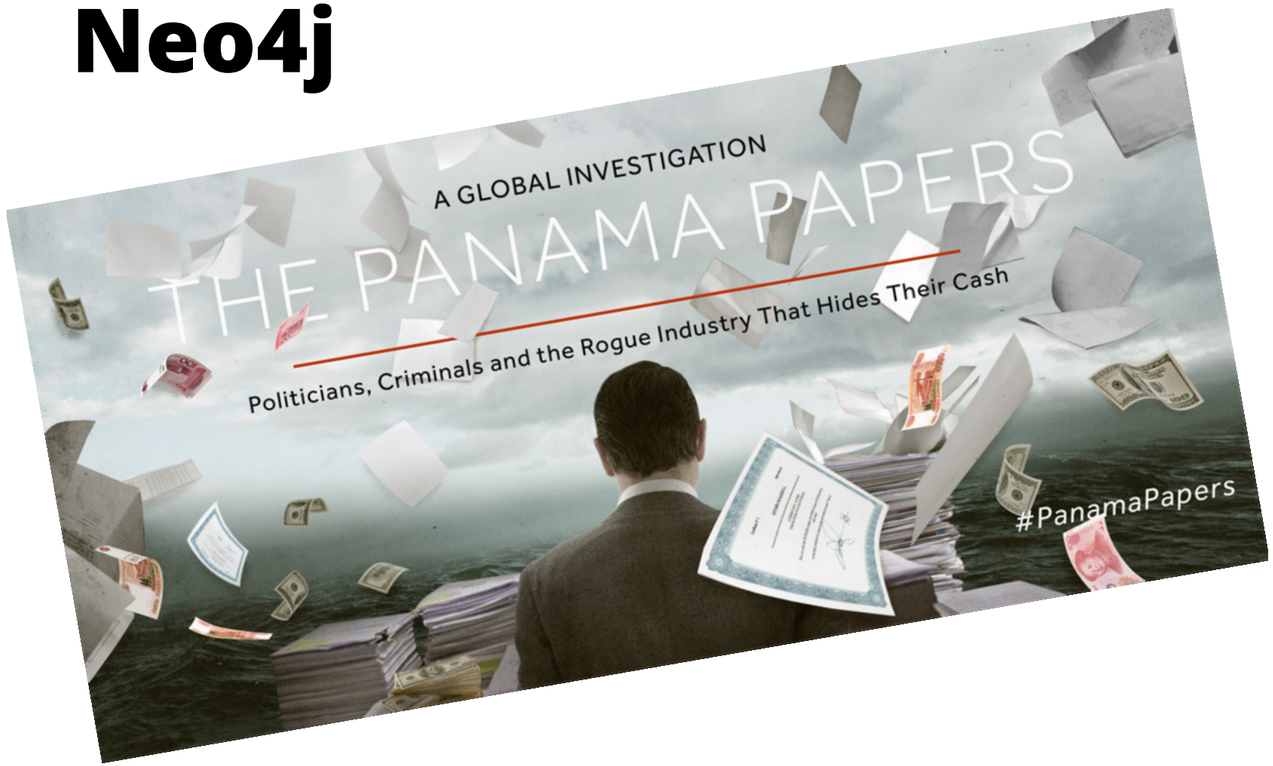



# Leveraging Cross-Silo Connections

# Echte Anwendungsfälle

neo4j

# Neo4j




**ICIJ - International Consortium of Investigative Journalists**


**https://neo4j.com/blog/icij-neo4j-unravel-panama-papers/**

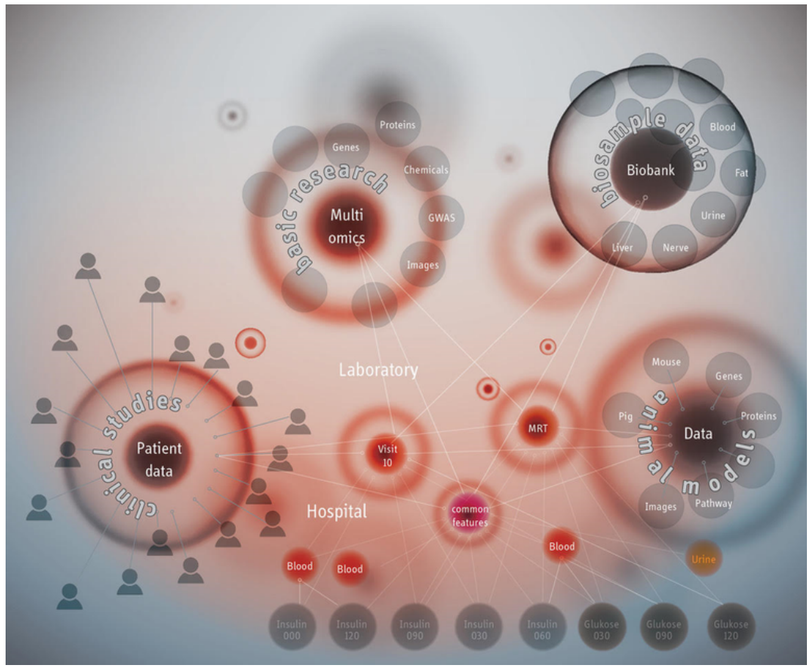
**https://neo4j.com/blog/analyzing-panama-papers-neo4j/**

**https://neo4j.com/blog/analyzing-paradise-papers-neo4j/**

# Neo4j

„In biology or medicine, data is connected. You know that entities are connected -- they are dependent on each other. The reason why we chose graph technology and Neo4j is because all the entities are connected.“

Dr Alexander Jarasch, DZD German centre of diabetic research




**https://www.zdnet.com/article/using-graph-database-technology-to-tackle-diabetes/**

# Probiert es aus!

neo4j

# neo4j.com/graphtour


@Neo4j #GraphTour

## GraphTour 2019 Brings Neo4j to a City Near You

Neo4j is hitting the road to bring a full day of content-rich sessions on how graph databases are revolutionizing the modern enterprise. This one-day event will turn you into a graph expert — no matter your technical background or familiarity with graph technology.

Meet our experts to hear first-hand about the advantages of Neo4j's native Graph Platform, which offers not just the Neo4j database, but also Analytics, Data Import and Transformation, Visualization, and Discovery capabilities.

There's a relationship-rich community waiting for you on the Neo4j GraphTour. Pick any of the cities below to find out more about these free events.

# Neo4j

- https://neo4j.com/download/
  - Neo4j Desktop (Analyst centric)
  - Neo4j Server (Community and Enterprise Edition)
    Community Edition: GPLv3
    Enterprise Edition: Proprietary

# Neo4j Datasets

- https://neo4j.com/sandbox-v2/
  - Preconfigured instance with several different datasets
- https://neo4j.com/graphgists/
  - Neo4j Graph Gists, Example Models and Cypher Queries
- https://offshoreleaks.icij.org/
  - Data convolutes mentioned early

# Mein „Bootiful Music“ Projekt

- https://github.com/michael-simons/bootiful-music
  - Beinhaltet Dockerfiles und Docker-Compose-Skripte für alle Dienste
- Zwei Spring Boot Anwendungen
  - charts: Anwendung auf Basis relationaler Daten
  - knowledge: Die gezeigte Anwendung auf Basis von Neo4j
- etl: das eigene Neo4j plugin
- Plus: Eine kleine Micronaut Demo

# Ressourcen

- Demo:
  github.com/michael-simons/bootiful-music
- Eine Reihe von Blog Posts: „From relational databases to databases with relations“
  https://info.michael-simons.eu/2018/10/11/from-relational-databases-to-databases-with-relations/
- Folien: speakerdeck.com/michaelsimons
- Kuratierte Liste von Neo4j, Neo4j-OGM und SDN Tipps:
  https://github.com/michael-simons/neo4j-sdn-ogm-tips
- GraphTour 2019: https://neo4j.com/graphtour/
- (German) Spring Boot Book
  @SpringBootBuch // springbootbuch.de

# Danke sehr!

neo4j

# Bildquellen

- Medical graph: DZD German centre of diabetic research
- Codd: Wikipedia
- Apoc and Cypher: Stills from the motion picture „The Matrix“
- Demo:
  https://unsplash.com/photos/Uduc5hJX2Ew
  https://unsplash.com/photos/FlPc9_VocJ4
  https://unsplash.com/photos/gp8BLyaTaA0