Datenbankzentrische Anwendungen mit Spring Boot und jOOQ

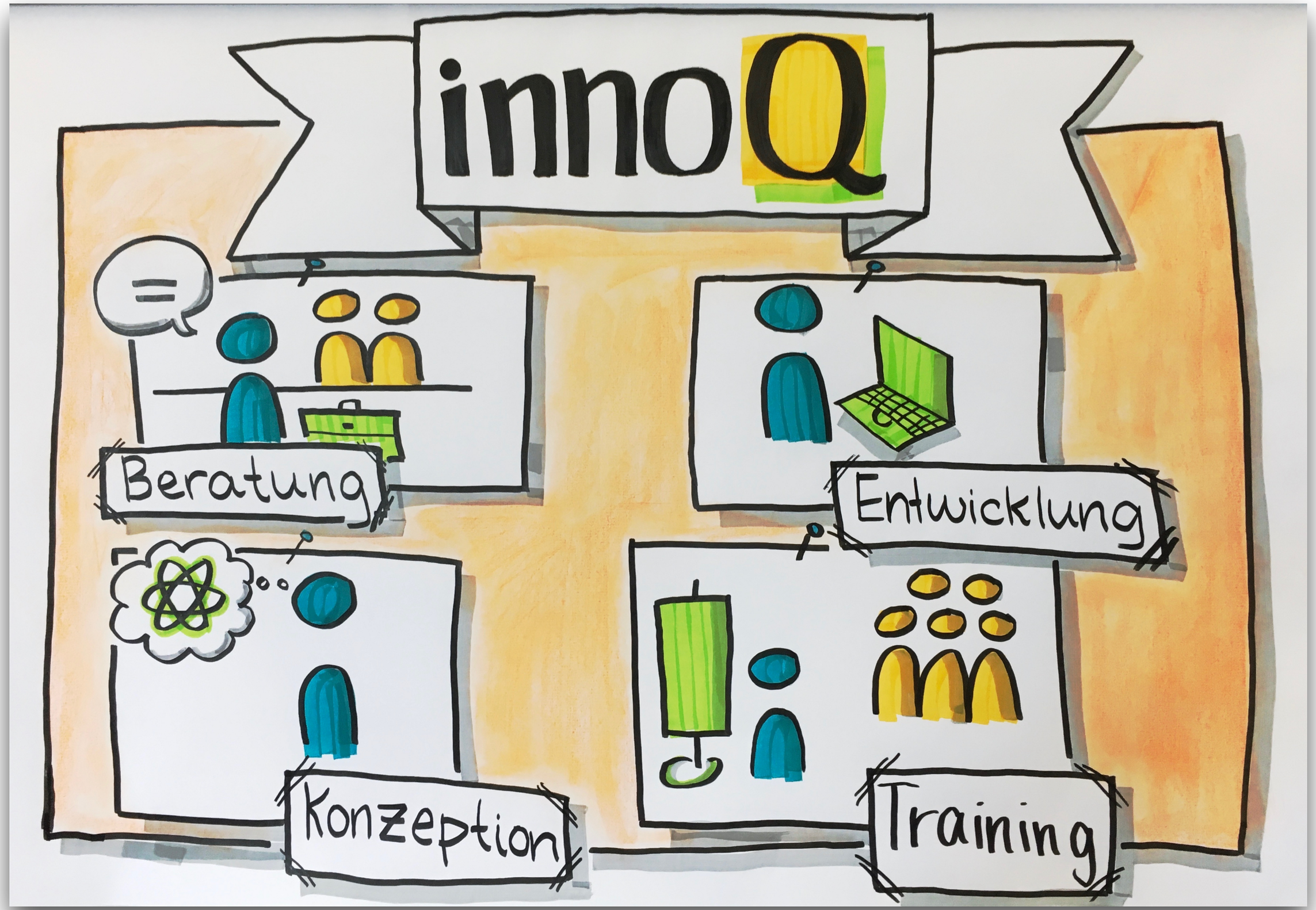
Java User Group Münster

15. November 2017

Michael Simons, @rotnroll666

Über mich

- Senior Consultant bei innoQ
- Mag relationale Datenbanken und SQL
- Bloggt zu Java, Spring und Softwarearchitektur unter info.michael-simons.eu
- Schreibt gerne ->
- Regt sich auf Twitter als @rotnroll666 über alles mögliche auf



von @iamjoyclark

Hintergrund

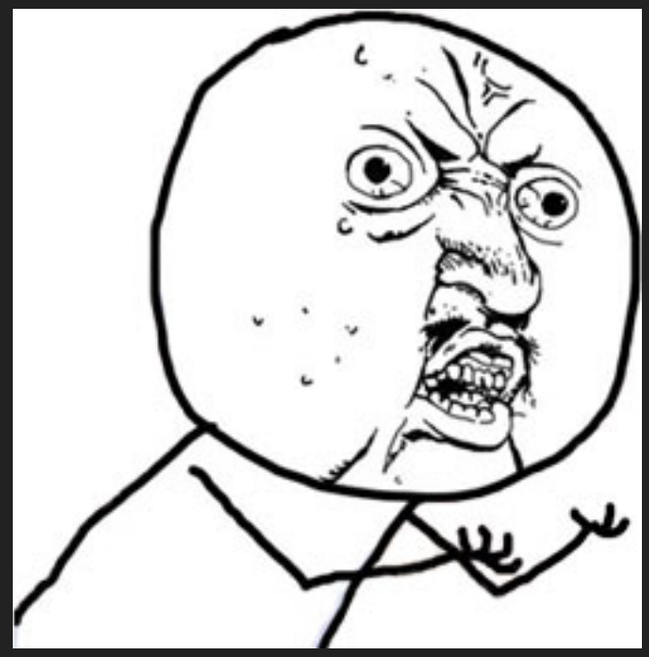
- Zeitreihenmanagement im Energiemarkt (Auswertungen Ist-Daten, Prognosen)
- GIS-Systeme auf Basis der Oracle Spatial Option

JAVA UND DATENBANKEN

Plain SQL, ORM oder etwas dazwischen?

JEDE MENGE!

- JDBC
- Springs JDBCTemplate
- JPA
  - JPQL
  - Criteria query
- MyBatis
- jOOQ
- noch einige mehr…



Y U NOT GIVE ME THE RIGHT TOOL?

NUN...

Quelle: https://www.flickr.com/photos/133021668@N04/31403416393

Quelle: https://www.flickr.com/photos/potentialpast/5582667252

✌Jochen Mader 🇪🇺

@codepitbull

Folgen

A good developer is like a werewolf: Afraid of silver bullets.

Original (Englisch) übersetzen

RETWEETS	GEFÄLLT
212	236

11:48 - 8. Okt. 2016

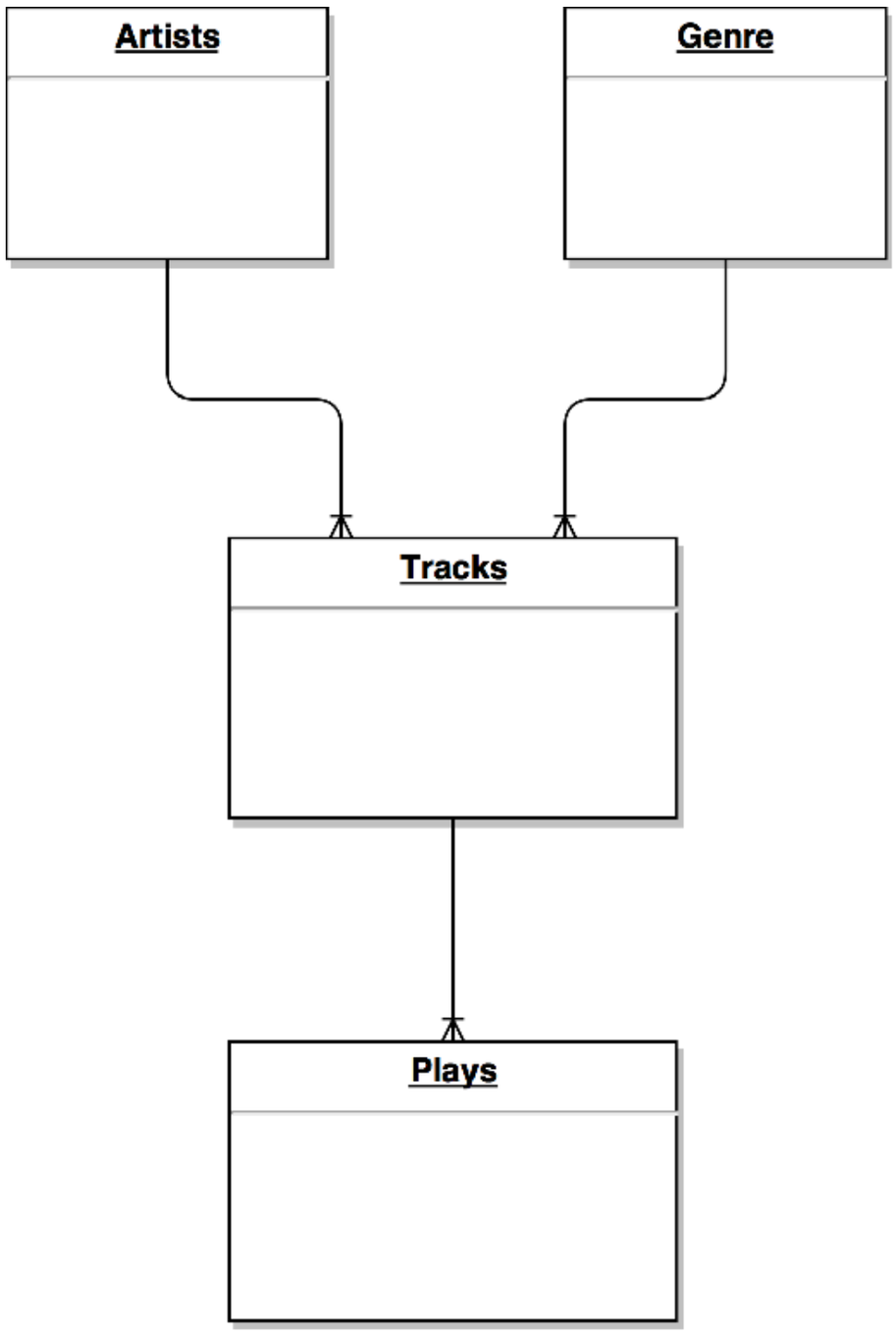
6 212 236

ZUERST EIN BEISPIEL

DIE HITPARADE*

* Ebenfalls Zeitreihen

DAS ZUGEHÖRIGE SCHEMA



PLAIN SQL

```
Select *
  from tracks
 where album = 'True Survivor';
```

PLAIN JPA

```
@Entity
@Table(name = "tracks")
public class TrackEntity implements Serializable {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    @Column
    private String album;

    public static void main(String...a) {
       final EntityManagerFactory factory =
          Persistence.createEntityManagerFactory("whatever");
       final EntityManager entityManager =
          factory.createEntityManager();

        List<Track> tracks = entityManager
          .createQuery("Select t from tracks where album = :album")
          .setParameter("album", "True Survivor")
          .getResultList();
    }
}
```

JPA + SPRING DATA

```
public interface TrackRepository extends
   JpaRepository<TrackEntity, Integer> {

    public List<Track> findAllByAlbum(final String name);

    public static void main(String...a) {
        TrackRepository trackRepository;
        final List<Track> tracks = trackRepository
           .findAllByAlbum("True Survivor");
    }
}
```

UND DANN WOLLTE JEMAND* EINE AUSWERTUNG HABEN...

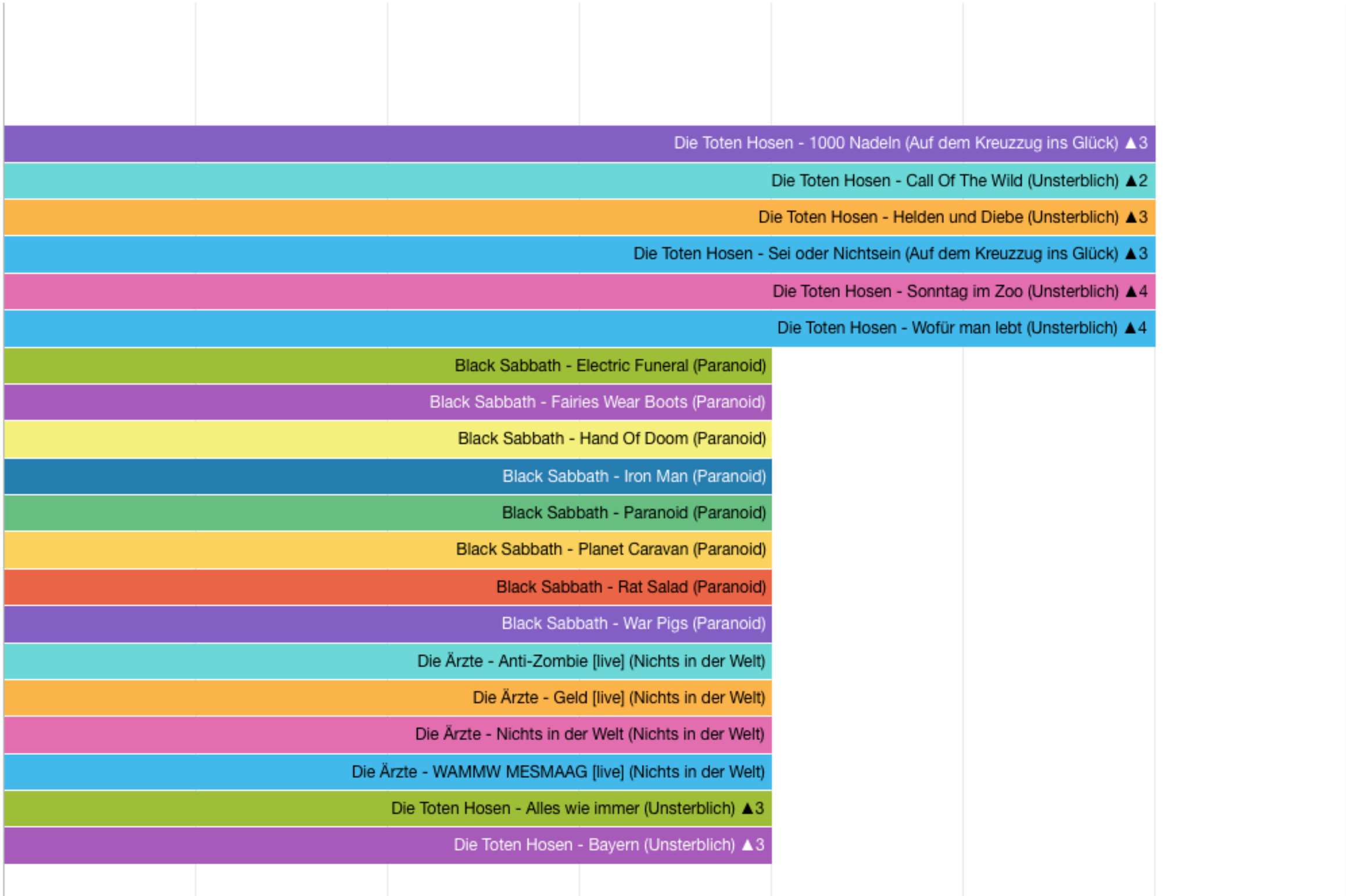
* „Business“

Charts

Select year and month

2016 May

Top 20



SELECT ALL THE STUFF...

```
WITH
  previous_month AS
   (SELECT p.track_id, count(*) as cnt,
           dense_rank() over(order by count(*) desc) as position
      FROM plays p
     WHERE trunc(p.played_on, 'DD') BETWEEN
           date'2016-04-01' and date'2016-04-30' GROUP BY p.track_id),
  current_month AS
   (SELECT p.track_id, count(*) as cnt,
           dense_rank() over(order by count(*) desc) as position
      FROM plays p
     WHERE trunc(p.played_on, 'DD') BETWEEN
           date'2016-05-01' and date'2016-05-31' GROUP BY p.track_id)
SELECT a.artist || ' - ' || t.name || ' (' || t.album || ')' as label,
       current_month.cnt,
       previous_month.position - current_month.position as change
  FROM tracks t
  JOIN artists a on a.id = t.artist_id
  JOIN current_month current_month on current_month.track_id = t.id
  LEFT OUTER join previous_month on previous_month.track_id = t.id
 ORDER BY current_month.cnt desc, label asc
 FETCH FIRST 20 ROWS ONLY;
```

ERNSTHAFT?

```
@Entity
@SqlResultSetMapping(
        name = "ChartMapping",
        columns = {
            @ColumnResult(name = "label", type = String.class),
            @ColumnResult(name = "cnt", type = Integer.class),
            @ColumnResult(name = "chage", type = Integer.class)
        })
@NamedNativeQueries(
        @NamedNativeQuery(
                name = "ChartQuery",
                resultSetMapping = "ChartMapping",
                query = ""
                + "WITH \n"
                + "  previous_month AS\n"
                + "    (SELECT p.track_id, count(*) as cnt, \n"
                + "            dense_rank() over(order by count(*) desc) as
position  \n"
                + "       FROM plays p \n"
                + "      WHERE trunc(p.played_on, 'DD') between date'2016-04-01'
and date'2016-04-30' GROUP BY p.track_id),\n"
                + "  current_month AS\n"
                + "    (SELECT p.track_id, count(*) as cnt, \n"
                + "            dense_rank() over(order by count(*) desc) as
position \n"
                + "       FROM plays p \n"
                + "     WHERE trunc(p.played_on, 'DD') between date'2016-05-01'
and date'2016-05-31' GROUP BY p.track_id)\n"
                + "SELECT a.artist || ' - ' || t.name || ' (' || t.album || ')'
as label,\n"
                + "       current_month.cnt,  \n"
                + "       previous_month.position - current_month.position as
change\n"
                + "  FROM tracks t\n"
                + "  JOIN artists a on a.id = t.artist_id\n"
                + "  JOIN current_month current_month on current_month.track_id
= t.id\n"
                + "  LEFT OUTER join previous_month on previous_month.track_id
= t.id\n"
                + " ORDER BY current_month.cnt desc, label asc"
        )
)
public class PlayEntity {
    public static void main(String... a) {
        // Don't do this at home
        EntityManager entityManager;
        List<Object[]> results =
entityManager.createNamedQuery("ChartQuery").setMaxResults(20).getResultList();
        results.stream().forEach((record) -> {
            String label = (String) record[0];
            Integer cnt = (Integer) record[1];
            Integer change = (Integer) record[2];
        });
    }
}
```

SQL TRIFFT JAVA

```
this.create
        .with(currentMonth)
        .with(previousMonth)
        .select(label,
                currentMonth.field("cnt"),
                previousMonth.field("position").minus(
                    currentMonth.field("position")
                ).as("change")
        )
        .from(TRACKS)
        .join(ARTISTS).onKey()
        .join(currentMonth)
          .on(currentMonth.field("track_id", BigDecimal.class)
             .eq(TRACKS.ID))
        .leftOuterJoin(previousMonth)
          .on(previousMonth.field("track_id", BigDecimal.class)
             .eq(TRACKS.ID))
        .orderBy(currentMonth.field("cnt").desc(), label.asc())
        .limit(n)
        .fetch()
        .formatJSON(response.getOutputStream());
```

jOOQ

WAS IST JOOQ?

- „Query builder framework“
- Java DSL zur Generierung datenbankspezifischer Statements
- Das Schema ist die „treibende Kraft“
  - Generierung eines Java-Schemas (Optional, aber empfohlen)
- Typsicher
- OpenSource für OpenSource Datenbanken, $ bis $$ für Enterprise Datenbanken

Open Source

CUBRID 8.4
Derby 10.10
Firebird 2.5
H2 1.3
HSQLDB 2.2
MariaDB 5.2
MySQL 5.5
PostgreSQL 9.0
SQLite

Express

CUBRID 8.4
Derby 10.10
Firebird 2.5
H2 1.3
HSQLDB 2.2
MariaDB 5.2
MySQL 5.5
PostgreSQL 9.0
SQLite

Microsoft Access 2013 [1]
Oracle 10g Express
SQL Server 2008 Express

Professional

CUBRID 8.4
Derby 10.10
Firebird 2.5
H2 1.3
HSQLDB 2.2
MariaDB 5.2
MySQL 5.5
PostgreSQL 9.0
SQLite

Microsoft Access 2013 [1]
Oracle 10g (All editions)
SQL Server 2008 (All editions)

Amazon Redshift [4]
SQL Azure

Enterprise

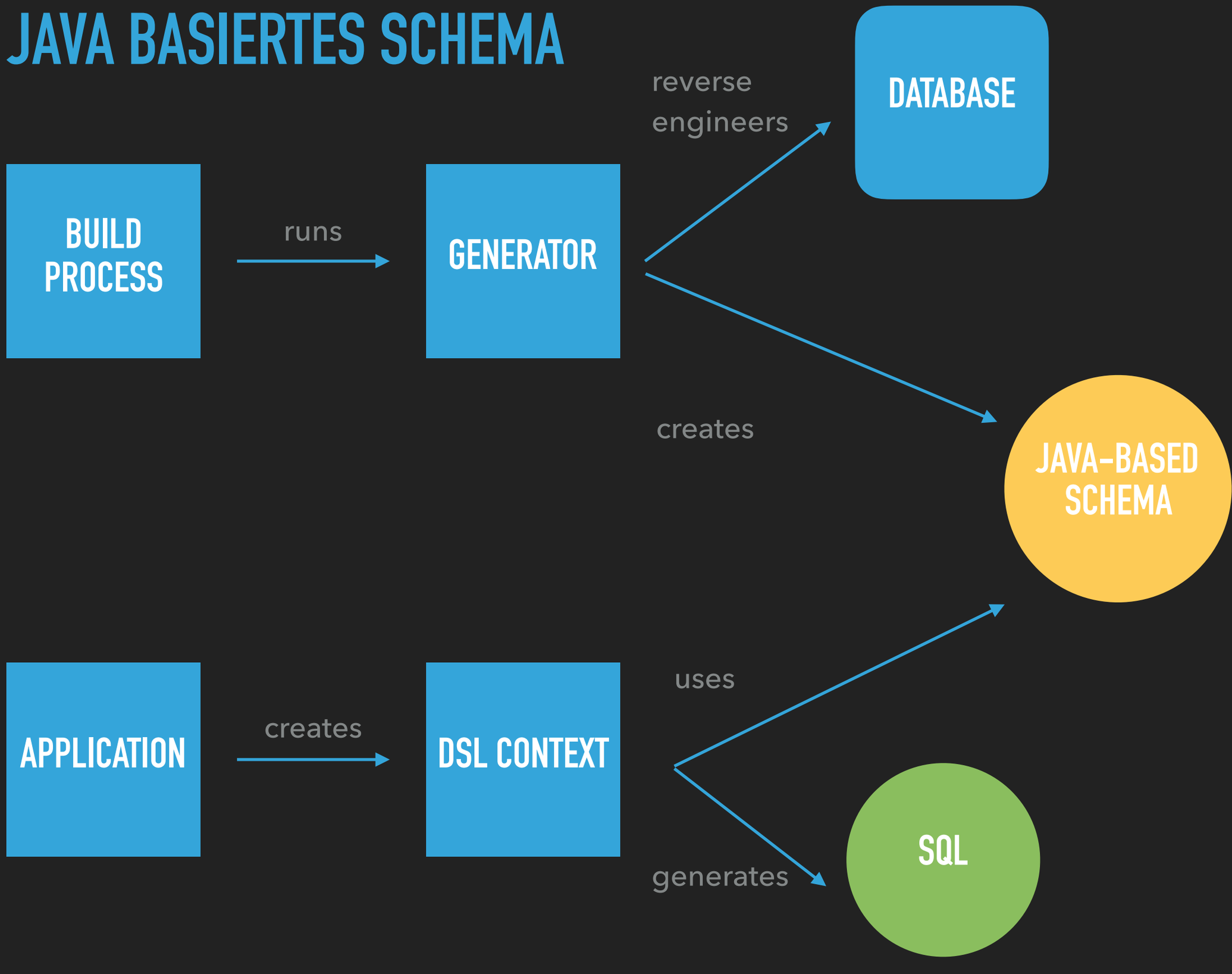
CUBRID 8.4
Derby 10.10
Firebird 2.5
H2 1.3
HSQLDB 2.2
MariaDB 5.2
MySQL 5.5
PostgreSQL 9.0
SQLite

Microsoft Access 2013 [1]
Oracle 10g (All editions)
SQL Server 2008 (All editions)

Amazon Redshift [4]
SQL Azure

DB2 LUW 9.7
HANA (All editions) [3]
Informix 12.10 [2]
Ingres 10.1
Sybase ASE 15.5
Sybase SQL Anywhere 12
Vertica 7.1 [4]

JAVA BASIERTES SCHEMA



DATENBANKMIGRATIONEN SIND ESSENTIELL

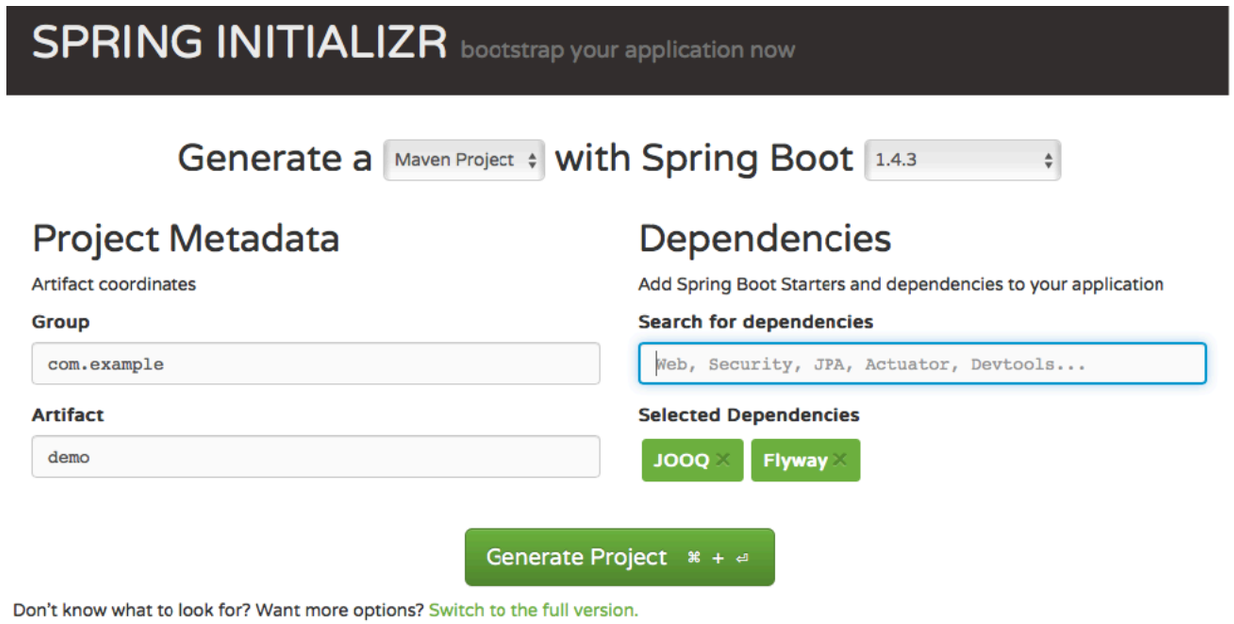
- Liquibase
- Flyway

WORKFLOW

- Build gegen Entwicklungsdatenbank
  - startet Migration
  - startet jOOQ Generator
- Anwendung gegen Produktionsdatenbank
  - startet ebenfalls Migration

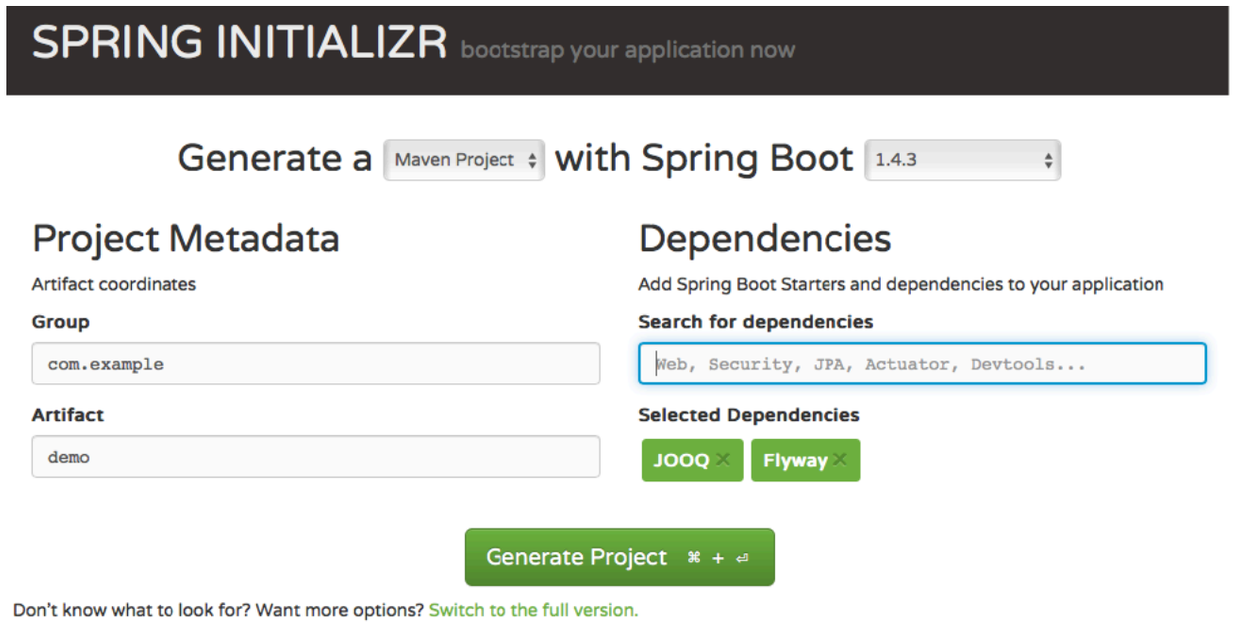
➡ Java Schema „passt“ immer zur Datenbank

WIE FUNKTIONIERT DAS MIT SPRING BOOT?




```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jooq</artifactId>
</dependency>
<dependency>
  <groupId>org.flywaydb</groupId>
  <artifactId>flyway-core</artifactId>
</dependency>
```

WIE FUNKTIONIERT DAS MIT SPRING BOOT?




```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jooq</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.jooq</groupId>
      <artifactId>jooq</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.jooq.pro</groupId>
  <artifactId>jooq</artifactId>
  <version>${jooq.version}</version>
</dependency>
<dependency>
  <groupId>org.flywaydb</groupId>
  <artifactId>flyway-core</artifactId>
</dependency>
```

DEVELOPMENT CONNECTION IM POM

```
<properties>
  <db.url>
    jdbc:oracle:thin:@//localhost:1521/ORCLPDB1
  </db.url>
  <db.username>doag2016</db.username>
  <db.password>doag2016</db.password>
  <db.schema>DOAG2016</db.schema>
</properties>
```

MAVEN PROPERTIES IN DEFAULT-KONFIGURATION NUTZEN

```
spring.datasource.url = @db.url@
spring.datasource.username = @db.username@
spring.datasource.password = @db.password@
```

BUILDTIME MIGRATION MIT FLYWAY

```
<build>
  <plugins>
    <plugin>
      <groupId>org.flywaydb</groupId>
      <artifactId>flyway-maven-plugin</artifactId>
      <version>${flyway.version}</version>
      <executions>
        <execution>
          <phase>generate-sources</phase>
          <goals>
            <goal>migrate</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <url>${db.url}</url>
        <user>${db.username}</user>
        <password>${db.password}</password>
        <locations>
          <location>filesystem:src/main/resources/db/migration</location>
        </locations>
      </configuration>
    </plugin>
  </plugins>
</build>
```

JOOQ GENERATOR IMMER NACH DER MIGRATION STARTEN...

```
<plugin>
  <groupId>org.jooq.pro</groupId>
  <artifactId>jooq-codegen-maven</artifactId>
  <version>${jooq.version}</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <jdbc>
      <driver>oracle.jdbc.OracleDriver</driver>
      <url>${db.url}</url>
      <user>${db.username}</user>
      <password>${db.password}</password>
    </jdbc>
    <generator> <!-- whatever -->
    </generator>
  </configuration>
</plugin>
```

DEMO

ZUSAMMENFASSUNG

- Direkte Abbildung von Abfragen auf URLs
- Von einfach bis kompliziert alles möglich
  - Logging der generierten Queries ist hilfreich
- Einfache Übergabe von Parametern an Queries
- Oft benutzte Fragmente können wiederverwendet werden
- Records können auf beliebige POJOs abgebildet werden
  - Spezialisiertes Domainen-Modell
- DAOs sind auch möglich

IST DAS PORTABLE?

ZUM GRÖßTEN TEIL...

- jOOQ hält sich soweit wie möglich an den SQL-Standard
- Kann verschiedene SQL-Clauses emulieren
- Beispiele:
  - LIMIT vs. OFFSET x ROWS / FETCH
  - Herstellerspezifische Funktionen (trunc vs. date_trunc etc.)

WANN BENUTZT MAN DAS?

THE PROBLEM WITH INTERNET QUOTES IS THAT YOU CANT ALWAYS DEPEND ON THEIR ACCURACY" - ABRAHAM LINCOLN, 1864

THE SQL... IT'S ALWAYS BEEN THERE, IT WILL GUIDE YOU

Lukas Skyeder

USE CASES

- Kein Interesse an Managed-Objects
- Analytic functions (Use what your paid for)
- „Upsert“ (Merge-Statements)
- Partial selects
- Aufruf von Stored Procedures

ZURÜCK ZU DEN SILVER BULLETS...


FETISH-ORIENTED
PROGRAMMING

NOTHING IS MORE DANGEROUS THAN AN IDEA, WHEN IT'S THE ONLY ONE WE HAVE.

Émile Auguste Chartier

Source: https://www.flickr.com/photos/gruenewiese/15790420752

JPA / HIBERNATE *UND* JOOQ

- Automatische Datenbankmigrationen
- JPA / Hibernate zusammen mit Spring Data JPA
- JPQL Queries falls nötig (Eventuell Criteria Queries)
  - Native Queries nicht in Annotationen verstecken!
- SQL Code komplexer Abfragen und Projektionen mit jOOQ generieren
  - An den EntityManager übergeben und Entitäten selektieren
  - **oder den DSL Context direkt benutzen**

KLASSISCHE SQL-INTERVIEW FRAGE

„DAS GENRE, DAS AM HÄUFIGSTEN GESPIELT WURDE…“

„ALS JPA-ENTITY!“

SQL:2003, WINDOW-FUNKTION UND RANK()

```
select id, genre
from (
  select g.id, g.genre,
         rank() over (order by count(*) desc) rnk
    from plays p
    join tracks t on p.track_id = t.id
    join genres g on t.genre_id = g.id
   group by g.id, g.genre
) src
where src.rnk = 1;
```

EIGENES INTERFACE, DAS „UNSERE“ METHODE DEKLARIERT

```
interface GenreRepositoryExt {
    List<GenreEntity> findWithHighestPlaycount();
}
```

IMPLEMENTIERUNG DIESES INTERFACES

```
class GenreRepositoryImpl implements GenreRepositoryExt {

    private final EntityManager entityManager;
    private final DSLContext create;

    public List<GenreEntity> findWithHighestPlaycount() {
        final SelectQuery<Record> sqlGenerator =
                this.create.select()./* Query */getQuery();

        final String sql = sqlGenerator
                .getSQL(ParamType.NAMED);

        final Query query = entityManager
                .createNativeQuery(sql, GenreEntity.class);
        return query.getResultList();
    }
}
```

DEKLARATION DES SPRING DATA JPA REPOSITORY

```
public interface GenreRepository extends
        CrudRepository<GenreEntity, Integer>,
        GenreRepositoryExt {
}
```

DEMO

DANKE FÜR EURE ZEIT!

- Demo project: github.com/michael-simons/bootiful-databases
- Slides: speakerdeck.com/michaelsimons
- Kontakt: michael-simons.eu
- Twitter: @rotnroll666

SPRING BOOT BUCH

- Q1 2018 im Dpunkt.verlag
- springbootbuch.de
- @springbootbuch
- Beispiele des Kapitels „Persistenz“:
  github.com/springbootbuch/database_examples
  Beinhaltet JDBCTemplate, JPA und JOOQ mit Pagila-Demo Datenbank (PostgreSQL)