

# Various Ways to jOOQ

Software Craftsmen @ Java Vienna

# Klaus Unger



---

CEO von Software Craftsmen

---



Nutzt




-  **SQL** seit 1988
-  **Java** seit 1997

glücklich und zufrieden!

# Christoph Kober

Software Engineer bei Software Craftsmen



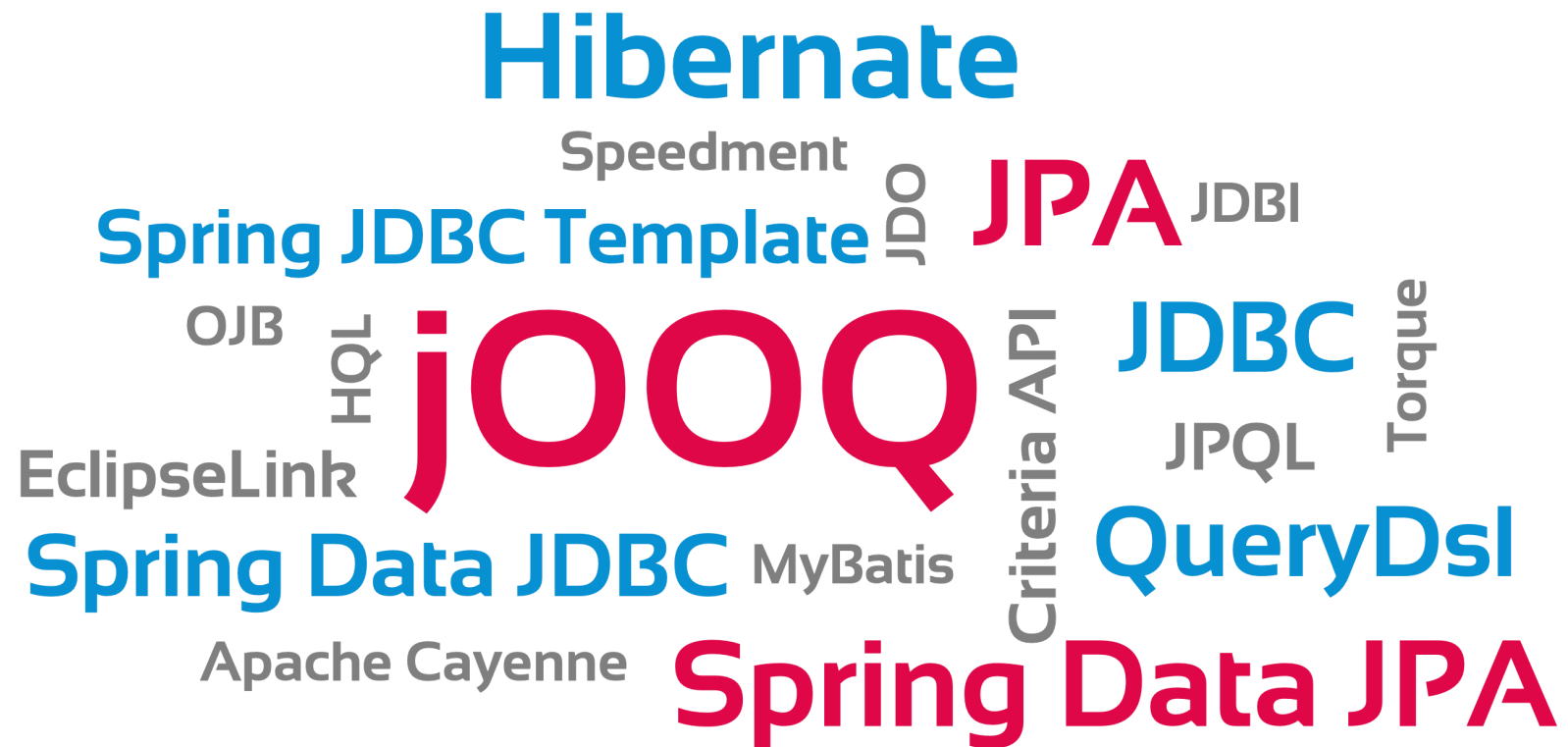
-  Fokus auf **Java** und **Kotlin**
-  liebt saubere **Architekturen**
-  entdeckt seine Liebe für **SQL**

# Persistenz-Technologien

im Java-Umfeld

# Persistenz-Technologien

im Java-Umfeld






# jOOQ im Überblick

Das Schweizer Taschenmesser für Persistence

# Was ist jOOQ?

-  **Library** + Toolset für RDBMS
-  für **Java** (+ Kotlin, Scala, Groovy)

# Lizenz

-  **Open Source** für Open Source-Datenbanken
-  **kostenpflichtig** für Enterprise-Datenbanken
-  je mehr **Enterprise**, desto **teurer**



# "SQL was never meant to be abstracted"

————— Der Grundgedanke hinter jOOQ —————

# "SQL was never meant to be abstracted"

SQL-artige, typsichere DSL für Java

```
List<PhotosRecord> photosFromAustrianStudios = dsl
    .select(PHOTOS.fields())
    .from(PHOTOS)
    .join(PHOTOGRAPHERS)
    .on(PHOTOS.PHOTOGRAPHER_ID.eq(PHOTOGRAPHERS.ID))
    .join(COUNTRIES)
    .on(PHOTOGRAPHERS.STUDIO_COUNTRY_ID.eq(COUNTRIES.ID))
    .where(COUNTRIES.ISO2CODE.eq("AT"))
    .fetchInto(PHOTOS);
```

# Code-Generierung

---

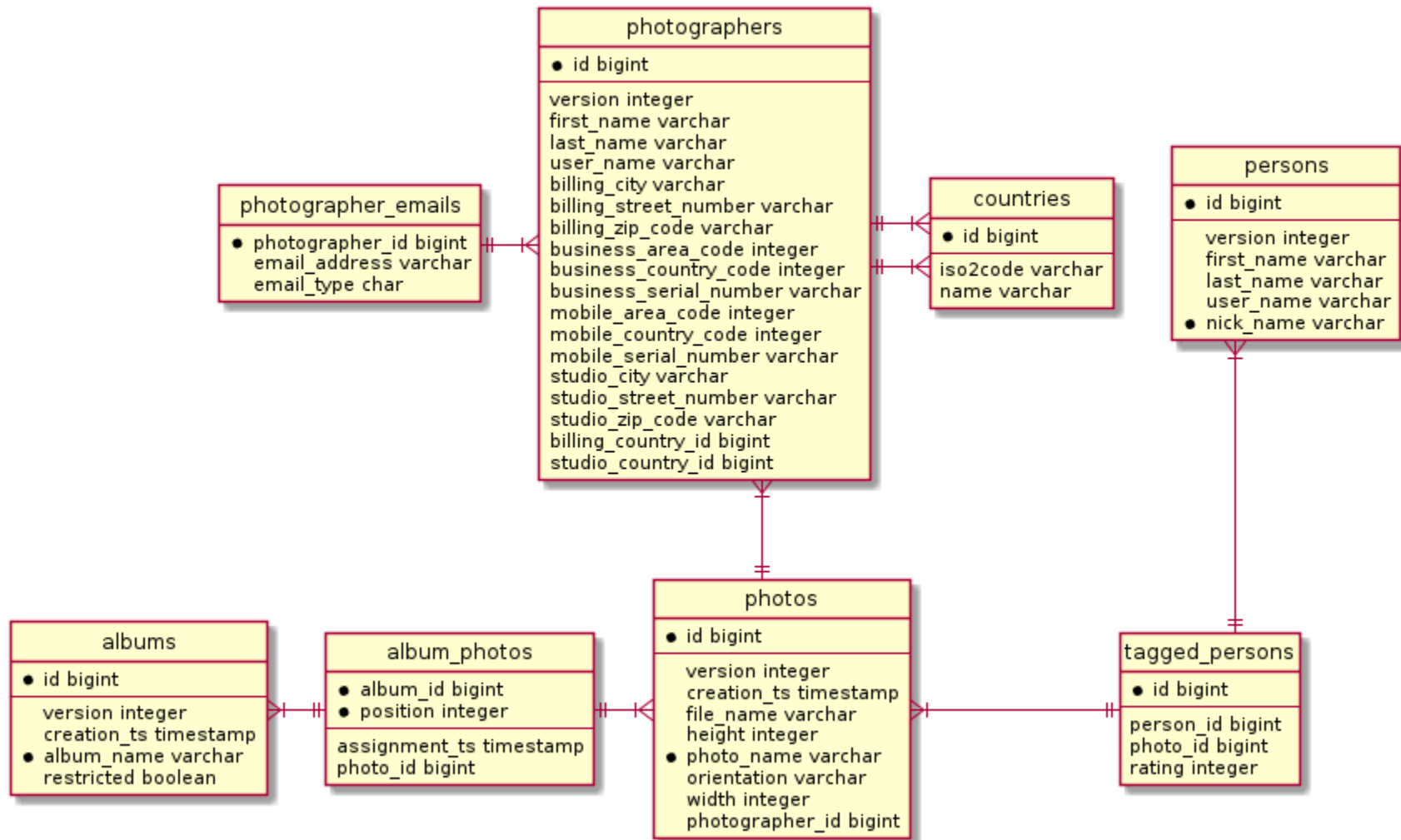
## Quellen

---



- bestehende Datenbank
- Liquibase
- DDL-Skripte (z.B. Flyway)
- XML-Konfiguration

# Beispiel-Schema



# Kombination JPA und jOOQ

## Szenario 1

# JPA und jOOQ

## Szenario

🌿 Bestandsprojekt mit **Spring Data JPA**

## Problem

🔍 **Komplexe Queries** notwendig, bestehende Tools unzureichend

## Lösung

✅ **jOOQ** nur für spezifische Queries, nebst JPA

# Existierende Entity + Repository

```
@Entity @Table(name = "albums")
public class Album extends AbstractPersistable<Long> {
    @NotNull @Version private Integer version;
    @NotBlank @NotNull @Column(name = "album_name", length = 64) private String name;
    @PastOrPresent @NotNull @Column(name = "creation_ts") private LocalDateTime creationTS;
    @NotNull private boolean restricted;

    @ElementCollection
    @CollectionTable(
        name = "album_photos",
        joinColumns = @JoinColumn(name = "album_id"))
    @OrderColumn(name = "position")
    private List<AlbumPhoto> photos = new ArrayList<>();
}
```

```
@Repository
public interface AlbumRepository extends JpaRepository<Album, Long>, CustomAlbumRepository {
    List<Album> findAllByNameLike(String name);
    List<Album> findAllByCreationTSBetween(LocalDateTime startTS, LocalDateTime endTS);
}
```

# Komplexe Query

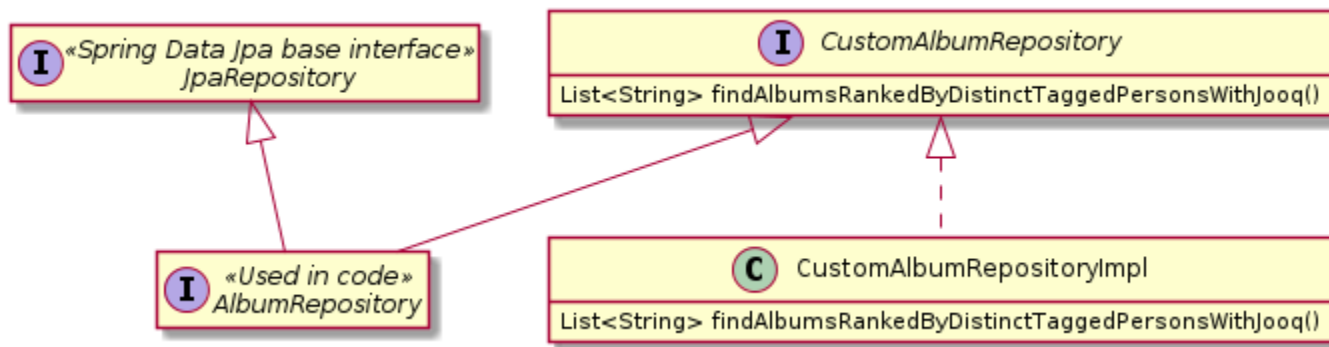
mit JPA @Query-Annotation

```
@Repository
public interface AlbumRepository extends JpaRepository<Album, Long>, CustomAlbumRepository {

    @Query(
        value = """
            select albums.album_name
            from albums
            join album_photos
            on albums.id = album_photos.album_id
            join photos
            on album_photos.photo_id = photos.id
            join tagged_persons
            on photos.id = tagged_persons.photo_id
            group by albums.id, albums.album_name
            order by count(distinct tagged_persons.person_id) desc
        """,
        nativeQuery = true
    )
    List<String> findAlbumsRankedByDistinctTaggedPersonsWithNativeQuery();
}
```



# Spring Data Custom Repository



# Implementierung im Custom Repository

mit jOOQ

```
@Override
public List<String> findAlbumsRankedByDistinctTaggedPersonsWithJooq() {
    return dsl.select(ALBUMS.ID, ALBUMS.ALBUM_NAME, countDistinct(TAGGED_PERSONS.PERSON_ID)
        .from(ALBUMS)
        .join(ALBUM_PHOTOS)
        .on(ALBUMS.ID.eq(ALBUM_PHOTOS.ALBUM_ID))
        .join(PHOTOS)
        .on(ALBUM_PHOTOS.PHOTO_ID.eq(PHOTOS.ID))
        .join(TAGGED_PERSONS)
        .on(PHOTOS.ID.eq(TAGGED_PERSONS.PHOTO_ID))
        .groupBy(ALBUMS.ID, ALBUMS.ALBUM_NAME)
        .orderBy(countDistinct(TAGGED_PERSONS.PERSON_ID).desc())
        .fetch(result -> result.get(ALBUMS.ALBUM_NAME));
}
```

# Die volle Power von SQL

## Common Table Expressions deklarieren

```
var photosWherePersonIsTagged = name("photos_where_person_is_tagged")
    .fields(photoId, rating)
    .as(select(TAGGED_PERSONS.PHOTO_ID.as(photoId), TAGGED_PERSONS.RATING.as(rating))
        .from(TAGGED_PERSONS)
        .where(TAGGED_PERSONS.PERSON_ID.eq(personId)));
```

```
var ratingsOfPhotosWherePersonIsTagged = name("ratings_of_photos_where_person_is_tagged")
    .fields(photoId, rating, avgRating)
    .as(select(
        photosWherePersonIsTagged.field(photoId),
        photosWherePersonIsTagged.field(rating),
        avg(TAGGED_PERSONS.RATING))
    .from(TAGGED_PERSONS).join(photosWherePersonIsTagged)
    .on(photosWherePersonIsTagged.field(photoId, BIGINT).eq(TAGGED_PERSONS.PHOTO_ID))
    .groupBy(photosWherePersonIsTagged.field(photoId), photosWherePersonIsTagged.field(rat
```

# Die volle Power von SQL

## Common Table Expressions verwenden

```
var ratingStatistics = dsl
    .with(photosWherePersonIsTagged)
    .with(ratingsOfPhotosWherePersonIsTagged)
    .select(
        avg(ratingsOfPhotosWherePersonIsTagged.field(rating, INTEGER))
            .cast(avgRatingInGoodPhotos)
            .as(avgRatingInGoodPhotos),
        count(ratingsOfPhotosWherePersonIsTagged.field(rating))
            .as(numberOfGoodPhotos)
    )
    .from(ratingsOfPhotosWherePersonIsTagged)
    .where(ratingsOfPhotosWherePersonIsTagged.field(avgRating, NUMERIC).ge(BigDecimal.valueOf(4)))
    .fetchSingle();
```

# Dynamische Queries

## Queries programmatisch erstellen

```
Optional<String> countryCode = ...;

var countryCondition = countryCode.map(COUNTRIES.ISO2CODE::eq).orElse(noCondition());

Result<CountriesRecord> countries = dsl
    .selectFrom(COUNTRIES)
    .where(countryCondition)
    .fetch();
```

# jOOQ im Greenfield-Einsatz

## Szenario 2

# jOOQ im Greenfield-Einsatz

-  Volle **Flexibilität** im Persistence-Layer
- Von **DB-lastig** bis **komplette Isolation** - alles ist möglich!

# Stored Procedures + Views

## Datenbanklastiger Ansatz

```
var personId = 1234L;  
var ratingStatistics = Routines.ratingStatisticsForPerson(dsl.configuration(), personId);
```

```
var photoId = 5678L;  
var photoDetails = dsl  
    .selectFrom(PHOTO_DETAILS)  
    .where(PHOTO_DETAILS.ID.eq(photoId))  
    .fetchOne();
```



# jOOQ Records

## Active Record-like

```
PhotosRecord record = new PhotosRecord();
record.setId(1L);
record.setPhotoName("My First Photo");
record.setFileName("mfp.jpg");
record.setWidth(80);
record.setHeight(80);
record.setOrientation("S");
record.setVersion(1);

record.attach(dsl.configuration());
record.insert();
```

# POJOs + DAOs

## Layered Architecture-kompatibel

```
var photoDao = new PhotosDao(dsl.configuration());
var favoritePhotographerId = 1234L;

var photos = photoDao.fetchByPhotographerId(favoritePhotographerId);
photos.forEach(photo ->
    photo.setPhotoName(photo.getPhotoName() + " - Very cool!")
);

photoDao.update(photos);
```

# Mapping von Domain Entities





## Abgekapseltes Domain Model

```
var mappedPhotos = dsl
    .selectFrom(PHOTOS)
    .where(PHOTOS.PHOTOGRAPHER_ID.in(1L, 2L, 3L, 4L))
    .fetch(photo -> new MyPhotoDomainEntity(photo.getId(), photo.getPhotoName()));
```

```
var mappedPhotosWithProjection = dsl
    .select(PHOTOS.ID, PHOTOS.PHOTO_NAME)
    .from(PHOTOS)
    .where(PHOTOS.PHOTOGRAPHER_ID.in(1L, 2L, 3L, 4L))
    .fetch(Records.mapping(MyPhotoDomainEntity::new));
```

# Wo ist mein Change Tracking?

oder: Brauche ich das wirklich?

-  JPA arbeitet mit beliebigen **Objektgraphen**
-  Transaktionen nur im **Request-Response-Zyklus**
-  Updates sind meistens **vorhersehbar**
-  **DDD Aggregates** setzen Grenzen

# Mapping aus anderen Quellen

## Abgekapseltes Domain Model

```
var photographerId = 123L;
var newPhotographerEmail = "the.photographer@example.com";
var event = new PhotographerBusinessEmailChanged(photographerId, newPhotographerEmail);

dsl.update(PHOTOGRAPHER_EMAILS)
    .set(PHOTOGRAPHER_EMAILS.EMAIL_ADDRESS, event.newEmail)
    .where(PHOTOGRAPHER_EMAILS.PHOTOGRAPHER_ID.eq(event.photographerId))
    .and(PHOTOGRAPHER_EMAILS.EMAIL_TYPE.eq("B"))
    .execute();
```

# Bedeutet Greenfield immer Solo-jOOQ?

---

Natürlich nicht!

---

- **CQRS** (JPA für Commands, jOOQ für Queries)
- jOOQ nur für **komplexe Queries** (+ JPA für den Rest)

# Arbeiten mit Legacy-DBs

## Szenario 3




# Szenarien

-  DB-Struktur **schlecht zu mappen** (Hibernate/JPA)
-  Schema-**Derivate** bei Kunden-Deployments



# Fazit

# Fazit

-  The **right tool** for the **right job**
-  jOOQ hilft in **vielen Szenarien**
-  **Good citizen**: Koexistenz mit anderen Technologien

# High Performance Java Persistence

Mit Hibernate-Experte **Vlad Mihalcea**



- **9.5. bis 11.5.** in Wien
- Tickets über **Eventbrite**

# Repository mit Beispielcode

Beispiele aus der Präsentation und mehr



# Software Craftsmen auf LinkedIn

für Rückfragen



# Fragen?



softwarecraftsmen