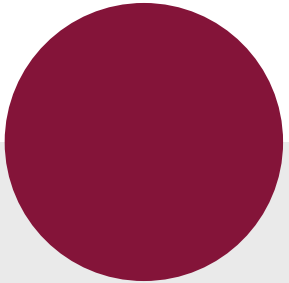
.msg .consulting .solutions .partnership



Persistenz unter Kontrolle mit JDBI für Java

DOAG Konferenz 2016 – 17. November 2016

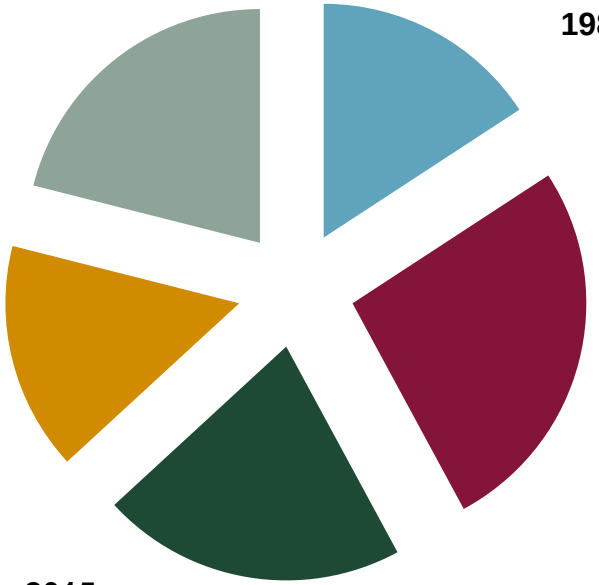
Persistenz unter Kontrolle mit JBDI für Java

1. SQL - effizienter Zugriff auf die Datenbank
2. JDBI als Abstraction für JDBC
   - Lesen von Daten
   - Schreiben von Daten
   - Transaktionen
   - Batching
   - Stored Procedures
3. Recap - JDBI als minimale Abstraction

Mein Sponsor und Arbeitgeber – msg systems ag

1980 gegründet

in 14 Städten
in Deutschland präsent



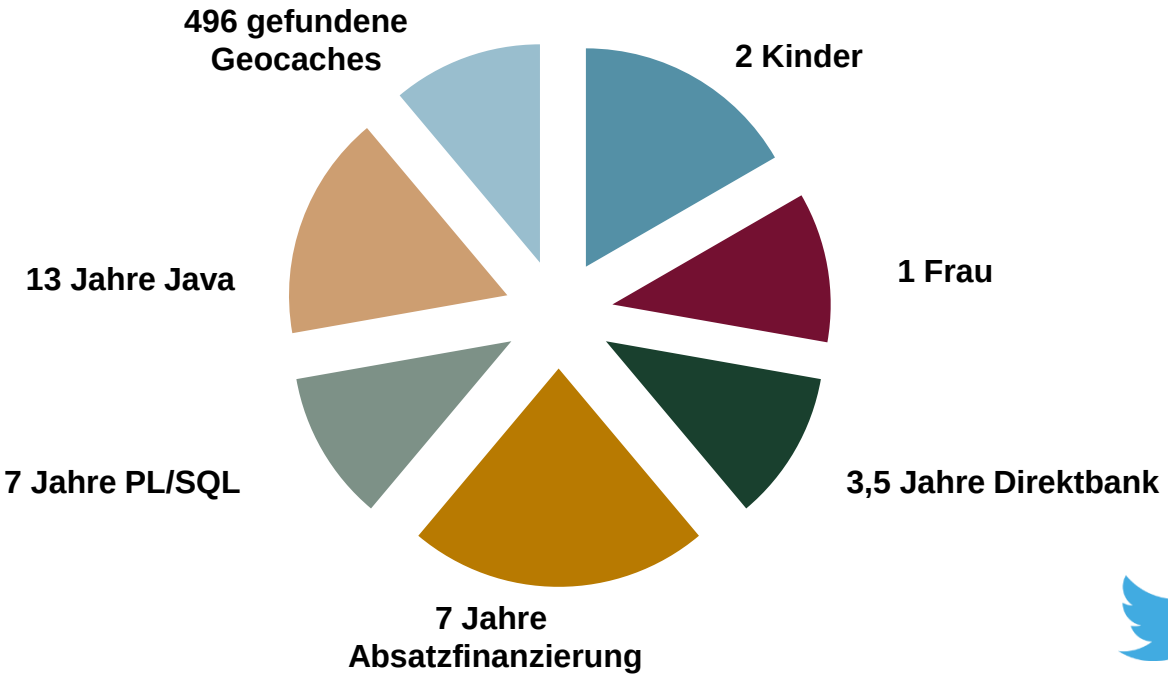
JUnit Lambda

23 Länder

mehr als 5.500 Kollegen

727 Millionen € Umsatz 2015

Wer ich bin – Principal IT Consultant im Geschäftsbereich Travel & Logistics



@ahus1de

1. **SQL - effizienter Zugriff auf die Datenbank**
2. JDBI als Abstraction für JDBC
   - Lesen von Daten
   - Schreiben von Daten
   - Transaktionen
   - Batching
   - Stored Procedures
3. Recap - JDBI als minimale Abstraction

SQL – effizienter Zugriff auf die relationale Datenbank

Effizienter Zugriff über bekannte Statements:

```
INSERT ...
UPDATE ...

SELECT ...
   SUM (...)
  OVER (PARTITION BY ... ORDER BY ...) ...  FROM ...
ORDER BY ...
```

JDBC – Standard-Interface zur Datenbank für Java VM

- Verfügbar seit Java 1.2 (1997)
- Ist für jede relationale Datenbank verfügbar
- Ist auch für nicht-relationale Datenbanken verfügbar (z. B. für Neo4j oder Hadoop)

Geringes Abstraktionslevel, hohe Fehleranfälligkeit, viel redundanter Code

Beispiele:

- Prepared Statements mit Fragezeichen (‚?‘) die an der abgezählt richtigen Stelle stehen müssen
- Transaktionssteuerung
- Fehleranfällig Datenübernahme nach SQL und zurück

JPA als Abstraktion für JDBC

- Verfügbar seit 2006, Teil von EJB 3.0 / Java EE 5
- Verschiedene Implementierungen verfügbar
- Hoher Abstraktionsgrad

Stolpersteine:

- Java-Objektmodell passt nur ungenau zur Persistenz der relationalen Datenbank
- Datenbankmodell muss ggf. auf JPA angepasst werden, damit Daten effizient geladen werden können
- 80% der täglichen Anwendungsfälle sind sehr einfach, 20% sind sehr schwer zu lösen
- Es ist schwer, es richtig zu benutzen

1. https://www.reddit.com/r/programming/comments/2cnw8x/what_orms_have_taught_me_just_learn_sql/ (2014)
2. http://www.javaperformancetuning.com/news/interview041.shtml (2004)

JDBI als Abstraktion für JDBC

- Verfügbar seit 2.0 seit 2007, 3.0 aktuell in Arbeit
- Minimale Abstraktion über JDBC
- Zugriff auf SQL
- Vermeiden von redundantem Code
- sicherer Umgang mit Parametern

Alternativen:

- Spring JDBC – Unterstützung für Transaktionen, Result-Sets, Connection Handling
- Query DSL – typsicherer Java-Code für das eigene Datenmodell
- jOOQ – typsicherer Java-Code für das eigene Datenmodell, teilweise kostenpflichtig

1. http://docs.spring.io/spring/docs/current/spring-framework-reference/html/jdbc.html
2. http://www.jooq.org/
3. http://www.querydsl.com/

1. SQL - effizienter Zugriff auf die Datenbank
2. **JDBI als Abstraction für JDBC**
   - **Lesen von Daten**
   - Schreiben von Daten
   - Transaktionen
   - Batching
   - Stored Procedures
3. Recap - JDBI als minimale Abstraction

Lesen von Daten – Abfrage definieren

Konzepte:

- Fachlich definierte Methoden als Interface (oder abstrakte Klasse)
- SQL als Annotation

```
@RegisterMapper(SightingResultMapper.class)
public interface SightingRepsitory {

    @SqlQuery("SELECT id, name FROM sighting")
    List<Sighting> findAll();

}
```

Lesen von Daten – Antwortobjekte Mappen

Konzepte:

- Optionale Mapper für Java-Typen
  (Für triviale Java Beans kann **`@MapResultAsBean`** verwendet werden)

```
public class SightingResultMapper implements ResultSetMapper<Sighting> {
    @Override
    public Sighting map(int index, ResultSet r,
        StatementContext ctx) throws SQLException {
        return Sighting.builder()
                .id(r.getInt("id"))
                .name(r.getString("name"))
                .build();
    }
}
```

* Builder für die Klasse Sighting mit Hilfe von Lombok generiert

Lesen von Daten – Parameter übergeben

Konzepte:

- Benannte Parameter in SQL und Methode

```
public interface SightingRepsitory ... {

    @SqlQuery("SELECT id, name FROM sighting WHERE name = :name")
    List<Sighting> findByName(@Bind("name") String name);

}
```

Persistenz unter Kontrolle mit JBDI für Java

1. SQL - effizienter Zugriff auf die Datenbank
2. **JDBI als Abstraction für JDBC**
   - Lesen von Daten
   - **Schreiben von Daten**
   - Transaktionen
   - Batching
   - Stored Procedures
3. Recap - JDBI als minimale Abstraction

Schreiben von Daten

Konzepte:

- Bean-Attribute sind im Zugriff

```
public interface SightingRepsitory ... {

    @SqlUpdate("INSERT INTO sighting (id, name) values (:s.id, :s.name)")
    void createNewSightingWithKnownId(@BindBean("s") Sighting sighting);

}
```

Schreiben von Daten

Konzepte:

- Generierte IDs werden zurückgeliefert

```
public interface SightingRepsitory ... {

    @SqlUpdate("INSERT INTO sighting (id, name)
                              VALUES (seq.nextval, :s.name)")
    @GetGeneratedKeys(columnName = "id",
                      value = OracleGeneratedKeyMapper.class)
    int createNewSightingAutogeneratedId(@BindBean("s") Sighting sighting);

}
```

Schreiben von Daten

Konzepte:

- Helfer für von Oracle generierte IDs via Sequences

```
/**
 * Oracle needs to be queried by index and not id (like
 * {@link org.skife.jdbi.v2.sqlobject.FigureItOutResultSetMapper} does).
 */
public static class OracleGeneratedKeyMapper implements ResultSetMapper<Long> {
    @Override
    public Long map(int index, ResultSet r, StatementContext ctx)
            throws SQLException {
        return r.getLong(1);
    }
}
```

1. SQL - effizienter Zugriff auf die Datenbank
2. **JDBI als Abstraction für JDBC**
   - Lesen von Daten
   - Schreiben von Daten
   - **Transaktionen**
   - Batching
   - Stored Procedures
3. Recap - JDBI als minimale Abstraction

Transaktionen

Konzepte:

- Annotationen in abstrakten Klassen

```
public abstract class SightingRepsitory ... {

    @Transaction
    public Set<Integer> doMultipleThings(Sighting s1, Sighting s2) {
        Set<Integer> result = new HashSet<>();
        result.add(createNewSightingAutogeneratedId(s1));
        result.add(createNewSightingAutogeneratedId(s2));
        return result;
    }

}
```

Transaktionen

Konzepte:

- Explizite Steuerung über funktionale Wrapper

```
public abstract class SightingRepsitory implements Transactional<...> {

    public Set<Integer> doMultipleThingsAnotherWay(Sighting s1, Sighting s2) {
        return inTransaction((t, status) -> {
            Set<Integer> result = new HashSet<>();
            result.add(t.createNewSightingAutogeneratedId(s1));
            result.add(t.createNewSightingAutogeneratedId(s2));
            return innerResult;
        });
    }
}
```

Transaktionen

Konzepte:

- Kombinierter Aufruf von bestehenden Repositories (Composition)

```
public abstract class Wrapper implements GetHandle {

    public Set<Integer> doMultipleThings(Sighting s1, Sighting s2) {
        return getHandle().inTransaction((conn, status) -> {
            Set<Integer> result = new HashSet<>();
            SightingRepsitory repo = conn.attach(SightingRepsitory.class);
            result.add(repo.createNewSightingAutogeneratedId(s1));
            result.add(repo.createNewSightingAutogeneratedId(s2));
            return result;
        });
    }

}
```

1. SQL - effizienter Zugriff auf die Datenbank
2. **JDBI als Abstraction für JDBC**
   - Lesen von Daten
   - Schreiben von Daten
   - Transaktionen
   - **Batching**
   - Stored Procedures
3. Recap - JDBI als minimale Abstraction

Batching

Konzept:

- Übergabe der Objekte als Array oder Iterator beim Schreiben

```
public interface SightingRepsitory {
    @SqlBatch("INSERT INTO sighting (id, name) VALUES (seq.nextval, :name)")
    @BatchChunkSize(1000)
    // we could pass an Iterator to deliver Sightings just in time
    void createNewSightings(@BindBean List<Sighting> sighting);
}
```

Batching

Konzept:

- Begrenzen der Anzahl der zurückgelieferten Zeilen
- Iterator, der die Objekte just-in-time erzeugt, wenn sie benötigt werden

```
public interface SightingRepsitory {
    @SqlQuery("SELECT id, name FROM sighting")
    @MaxRows(100)
    @MapResultAsBean
    Iterator<Sighting> selectSomeSightings();
}
```

1. SQL - effizienter Zugriff auf die Datenbank
2. **JDBI als Abstraction für JDBC**
   - Lesen von Daten
   - Schreiben von Daten
   - Transaktionen
   - Batching
   - **Stored Procedures**
3. Recap - JDBI als minimale Abstraction

Stored Procedures

Konzept:

- Out-Parameter benennen

```
public interface Calculator {
    @SqlCall("{ CALL calculate_score(:distance, :weight, :score) }")
    @OutParameter(name = "score", sqlType = Types.INTEGER)
    OutParameters calculateScore(@Bind("distance") long distance,
                                 @Bind("weight") long weight);
    }
}

/* ... */

long score = calculator.calculateScore(4, 2).getLong("score");
```

1. SQL - effizienter Zugriff auf die Datenbank
2. JDBI als Abstraction für JDBC
   - Lesen von Daten
   - Schreiben von Daten
   - Transaktionen
   - Batching
   - Stored Procedures
3. **Recap - JDBI als minimale Abstraction**

JDBI als minimale Abstraktion

Funktionsbereich	Unterstützung durch JDBI
Lesen von Daten	★★★★★
Schreiben von Daten	★★★★★
Transaktionen	★★★★★
Batching	★★★★★
Stored Procedures	★★★☆☆
Dokumentation	★★★☆☆

Fazit:

- Leichtgewichtiges Framework für diejenigen, die SQL beherrschen und nutzen wollen
- Gute Unterstützung für namensbasierte Parametrisierung von SQL Statements

@ahus1de

Links

JDBI

http://jdbi.org/

https://github.com/jdbi/jdbi

http://groups.google.com/group/jdbi

Beispielprojekt

https://github.com/ahus1/jdbi-by-example

@ahus1de

Alexander Schwartz
Principal IT Consultant

+49 171 5625767
alexander.schwartz@msg-systems.com

@ahus1de

msg systems ag (Headquarters)
Robert-Buerkle-Str. 1, 85737 Ismaning
Germany

www.msg-systems.com

.msg .consulting .solutions .partnership