jOOQ: A peace treaty between SQL and Java

SQL was never meant to be anything other than...

SQL!

Wer bin ich?

Java, SQL, PL/SQL

2001-2006 MSc Computer Science EPFL
2006-2009 Ergon Informatik AG
2009-2013 Crealogix E-Banking AG
2013 Contractor für Adobe Systems Inc.

Geek aus Zürich

Was motiviert mich?

SQL dominiert Datenbanksysteme

SQL gilt als «low level» und «staubig»

SQL könnte viel mehr

SQL soll wieder «sexy» sein

Worum geht es heute?

- Geschichte von SQL in Java
- Persistenz in Java heute
- jOOQ
- Demo
- Diskussion

SQL und Java - never ending story

JDBC

EJB 2.0 mit EntityBeans

Hibernate / TopLink

EJB 3.0 mit JPA 2.x

iBATIS / JDO / SpringData / 1'000 andere

JDBC


```
PreparedStatement stmt = connection.prepareStatement(
  "SELECT text FROM products WHERE cust_id = ? AND value < ?");
stmt.setInt(1, custID);
stmt.setBigDecimal(2, BigDecimal.ZERO);
ResultSet rs = stmt.executeQuery();

while (rs.next()) {
  System.out.println(rs.getString("TEXT"));
}
```

JDBC - die harte Wahrheit

```
01: PreparedStatement stmt = connection.prepareStatement(
02:   "SELECT p.text txt" +
03:   (isAccount ? ", NVL(a.type, ?) " : "") +
04:   "FROM products p " +
05:   (isAccount ? " INNER JOIN accounts a USING (prod_id) " : "") +
06:   " WHERE p.cust_id = ? AND p.value < ?" +
07:   (isAccount ? " AND a.type LIKE '%" + type + "%'" : "");
08: stmt.setInt(1, defaultType);
09: stmt.setInt(2, custID);
10: stmt.setBigDecimal(3, BigDecimal.ZERO);
11: ResultSet rs = stmt.executeQuery();
12:
13: while (rs.next()) {
14:   Clob clob = rs.getClob("TEXT");
15:   System.out.println(clob.getSubString(1, (int) clob.length());
16: }
17:
18: rs.close();
19: stmt.close();
```

JDBC - die harte Wahrheit

```
01: PreparedStatement stmt = connection.prepareStatement(            //
02:   "SELECT p.text txt" +                                          //
03:   (isAccount ? ", NVL(a.type, ?) " : "") +                       //
04:   "FROM products p " +                                           // Syntaxfehler wenn isAccount == false
05:   (isAccount ? " INNER JOIN accounts a USING (prod_id) " : "") + //
06:   " WHERE p.cust_id = ? AND p.value < ?" +                       //
07:   (isAccount ? " AND a.type LIKE '%" + type + "%'" : "");        // Syntaxfehler und SQL injection möglich
08: stmt.setInt(1, defaultType);                                     // Falscher bind index
09: stmt.setInt(2, custID);                                          //
10: stmt.setBigDecimal(3, BigDecimal.ZERO);                          //
11: ResultSet rs = stmt.executeQuery();                              //
12:
13: while (rs.next()) {                                              //
14:   Clob clob = rs.getClob("TEXT");                                // ojdbc6: clob.free() sollte aufgerufen
15:   System.out.println(clob.getSubString(1, (int) clob.length()); //         werden
16: }                                                                //
17:
18: rs.close();                                                      // close() nicht in finally block
19: stmt.close();                                                    //
```

JDBC - Pro und Kontra

JDBC Vorteile

- Keine Einschränkungen (Prozeduren, UDT's, Oracle-features)
- Einfach
- Schnell

JDBC Nachteile

- String-basiert
- Keine Syntax-Prüfung
- Viel Code, flache Struktur (indizierte Variablen)
  Repetitiv (Schliessen von ResultSet, Statement)
- "Vendor lock-in"

EJB 2.0 EntityBeans

```
public interface CustomerRequest extends EJBObject {
  BigInteger getId();
  String getText();
  void setText(String text);
  @Override
  void remove();
}

public interface CustomerRequestHome extends EJBHome {
  CustomerRequest create(BigInteger id);
  CustomerRequest find(BigInteger id);
}
```

EJB 2.0 – die harte Wahrheit

```
<weblogic-enterprise-bean>
  <ejb-name>com.example.CustomerRequestHome</ejb-name>
  <entity-descriptor>
    <pool>
      <max-beans-in-free-pool>100</max-beans-in-free-pool>
    </pool>
    <entity-cache>
      <max-beans-in-cache>500</max-beans-in-cache>
      <idle-timeout-seconds>10</idle-timeout-seconds>
      <concurrency-strategy>Database</concurrency-strategy>
    </entity-cache>
    <persistence>
      <delay-updates-until-end-of-tx>True</delay-updates-until-end-of-tx>
    </persistence>
    <entity-clustering>
      <home-is-clusterable>False</home-is-clusterable>
      <home-load-algorithm>round-robin</home-load-algorithm>
    </entity-clustering>
  </entity-descriptor>
  <transaction-descriptor/>
  <enable-call-by-reference>True</enable-call-by-reference>
  <jndi-name>com.example.CustomerRequestHome</jndi-name>
</weblogic-enterprise-bean>
```

EJB 2.0 - Pro und Kontra

EJB 2.0 Vorteile

- Intuitiv in der Anwendung (create(), remove(), store())
- Mächtig (Transaktionen, Caching, etc.)

EJB 2.0 Nachteile

- Nicht intuitiv in der Entwicklung (Home, Konventionen)
- Viel Konfiguration
- XDoclet
- Checked Exceptions (FinderException, CreateException)
- Repetitiv (ausser mit Codegenerator)
- Domain Model lebt im Container

Hibernate / TopLink - ORM


```
Session session = sessionFactory.openSession();
session.beginTransaction();

session.save(new Event("JUGS", new Date());
session.save(new Event("Feierabendbier", new Date());

List result = session.createQuery("from Event").list();
for (Event event : (List<Event>) result) {
  System.out.println("Event : " + event.getTitle());
}

session.getTransaction().commit();
session.close();
```

Hibernate - "navigieren"

```
List result = session.createQuery("from Event").list();
for (Event event : (List<Event>) result) {
  System.out.println("Teilnehmer für " + event);

  for (Person person : event.getParticipants()) {
    Company company = person.getCompany();

    System.out.println(person + " (" + company + ")");
  }
}
```

Hibernate - die harte Wahrheit


```
<hibernate-mapping package="org.hibernate.tutorial.hbm">
  <class name="Event" table="EVENTS">
    <id name="id" column="EVENT_ID">
      <generator class="increment"/>
    </id>
    <property name="date" type="timestamp" column="EVENT_DATE"/>
    <property name="title"/>
    <set name="participants" inverse="true">
      <key column="eventId"/>
      <one-to-many entity-name="Person"/>
    </set>
  </class>
</hibernate-mapping>
```

Hibernate - Pro und Kontra

Hibernate Vorteile

- Sehr intuitiv in der Anwendung (POJOs)
- POJO Code Generator
- Objekte können "navigiert" werden
- Hibernate implementiert JPA

Hibernate Nachteile

- Schwierig zu konfigurieren
- Caching ist sehr komplex
- HQL ist sehr limitiert. SQL kann viel mehr
- ORM tun sich schwer mit dem relationalen Datenmodell

JPA und EJB 3.0


```
EntityManager em = factory.createEntityManager();
em.getTransaction().begin();

em.persist(new Event("JUGS", new Date());
em.persist(new Event("Feierabendbier", new Date());

List result = em.createQuery("from Event").getResultList();
for (Event event : (List<Event>) result) {
  System.out.println("Event : " + event.getTitle());
}

em.getTransaction().commit();
em.close();
```

EJB 3.0 - die harte Wahrheit


```
@Entity @Table(name = "EVENTS")
public class Event {
  private Long id;
  private String title;
  private Date date;

  @Id @GeneratedValue(generator = "increment")
  @GenericGenerator(name = "increment", strategy = "increment")
  public Long getId() { /* … */ }

  @Temporal(TemporalType.TIMESTAMP)
  @Column(name = "EVENT_DATE")
  public Date getDate() { /* … */ }
```

Criteria - die harte Wahrheit

```
EntityManager em= ...
CriteriaBuilder builder = em.getCriteriaBuilder();
CriteriaQuery<Person> criteria = builder.createQuery(Person.class);
Root<Person> person = criteria.from(Person.class);
Predicate condition = builder.gt(person.get(Person_.age), 20);
criteria.where(condition);
TypedQuery<Person> query = em.createQuery(query);
List<Person> result = query.getResultList();
```

JPA - Pro und Kontra

JPA Fakten

- Hibernate HQL => JPQL
- Hibernate XML Mapping => Annotationen
- Hibernate Sessions => EntityManager (wie in C#)

JPA Vorteile

- Nach EJB 2.0 ein Standard
- Sehr "stabil" dank Hibernate / TopLink

JPA Nachteile

- CriteriaQuery

Was gibt es noch?

iBATIS / MyBatis

- XML-basiert. SQL ist externalisiert
- Eher ein "Leichtgewicht"
- Weiterentwicklung?

JDO

- Hat JDOQL (ähnlich wie HQL, JPQL)
- Unterstützt auch NoSQL stores

Spring Data, Apache DbUtils, etc

SQL kann viel mehr

```
SELECT   DENSE_RANK() OVER (ORDER BY p.votes DESC),
         LPAD(
           (p.votes * 100 / SUM(p.votes) OVER ()) || ' %',
           4, ' '
         ) AS "percent",
         p.text,
FROM     poll_options p
WHERE    p.poll_id = 12
ORDER BY p.votes DESC
```

Dasselbe mit jOOQ

```
DSL.using(connection, SQLDialect.ORACLE)
   .select(
      denseRank().over().orderBy(p.VOTES.desc()),
      p.VOTES
       .mul(100)
       .div(sum(p.VOTES).over())
       .concat(" %")
       .lpad(4, " ").as("percent"),
      p.TEXT)
   .from(POLL_OPTIONS.as("p"))
   .where(p.POLL_ID.eq(12))
   .orderBy(p.VOTES.desc())
   .fetch();
```

Dasselbe mit jOOQ in Scala (!)


```
DSL.using(connection, SQLDialect.ORACLE)
    select(
      denseRank() over() orderBy(p.VOTES desc),
      lpad(
        (p.VOTES * 100) / (sum(p.VOTES) over()) || " %",
        4, " "
      ) as "percent",
      p.TEXT
    )
    from (POLL_OPTIONS as "p")
    where (p.POLL_ID === 12)
    orderBy (p.VOTES desc)
    fetch
```

Geschichte und Community

2006 : Idee

2009 : Auf SourceForge

2010 : Öffentlich

2011 : Maven Central

ca. 200 aktive "Kontakte", 20 "Contributors"

ca. ~~9'000~~ 20'000 Downloads / Jahr

ca. 100'000 Besucher / Jahr auf www.jooq.org

Support

- CUBRID (Partner)
- DB2
- Derby
- Firebird
- H2
- HSQLDB
- Ingres
- MySQL
- Oracle
- Postgres
- SQL Server
- SQLite
- Sybase ASE
- Sybase SQL Anywhere

Demo

Diskussion