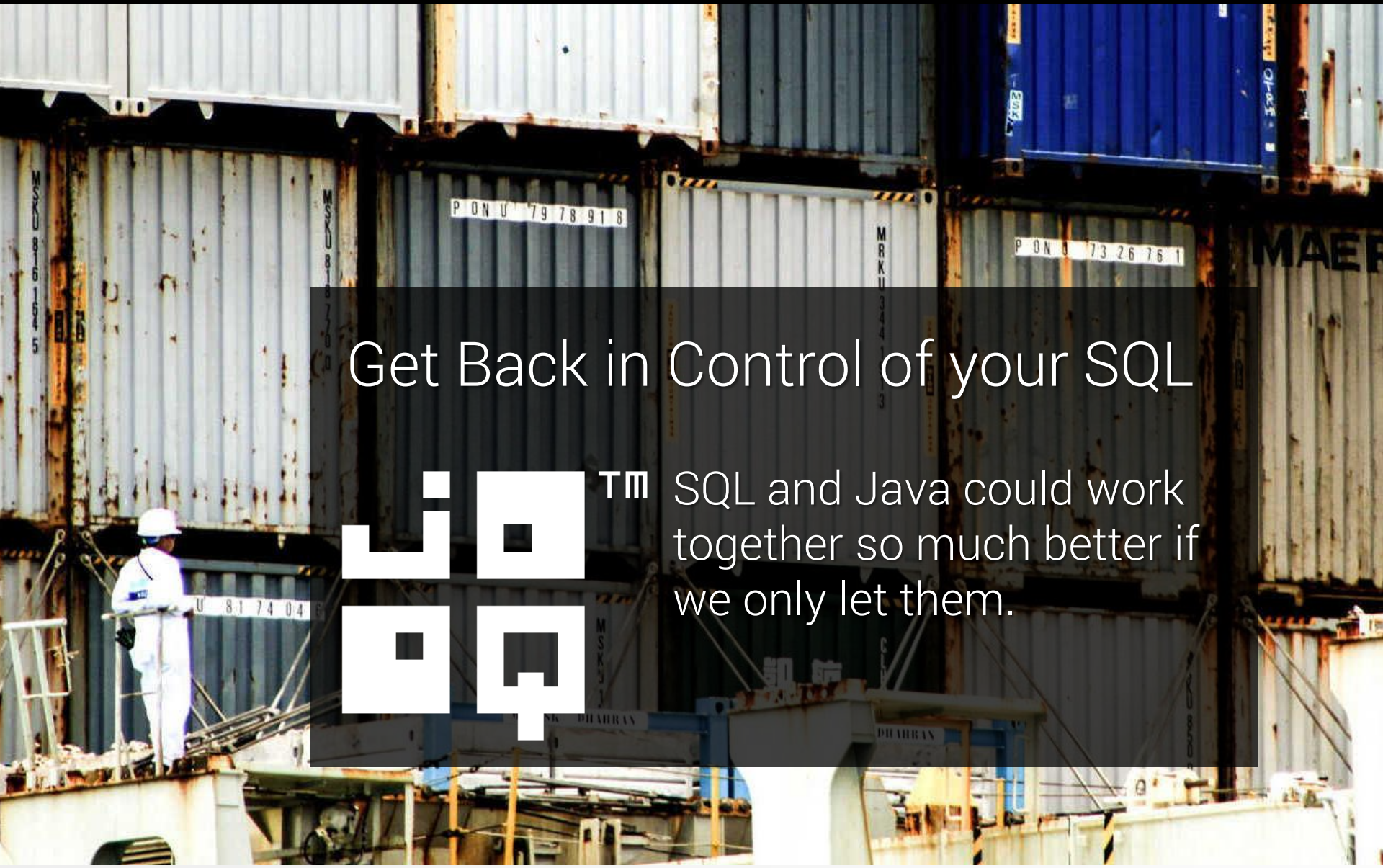


Get Back in Control of your SQL

jOOQ™

SQL and Java could work together so much better if we only let them.

About my motivation

SQL dominates database systems

SQL seems «low level» and «dusty»

SQL can do so much more

SQL should be «sexy» again

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 – the naked truth

```
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 – the naked truth

```
01: PreparedStatement stmt = connection.prepareStatement(            //
02:   "SELECT p.text txt" +                                          //
03:   (isAccount ? ", NVL(a.type, ?) " : "") +                       //
04:   "FROM products p " +                                           // Syntax error when 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 + "%'" : "");        // Syntax error and SQL injection possible
08: stmt.setInt(1, defaultType);                                     // Wrong 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() should be called
15:   System.out.println(clob.getSubString(1, (int) clob.length()); //
16: }                                                                //
17:
18: rs.close();                                                      // close() not really in finally block
19: stmt.close();                                                    //
```

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 – the naked truth

```
<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>
```

Hibernate – ORM

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

session.save(new Event("Conference", new Date());
session.save(new Event("After Party", 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 – «navigation»

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

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

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

Hibernate – the naked truth

```
<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>
```

JPA and EJB 3.0

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

em.persist(new Event("Conference", new Date());
em.persist(new Event("After Party", 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 – the naked truth

```
@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() { /* … */ }
```

EJB 3.0 – Annotatiomania™

```
@OneToMany(mappedBy = "destCustomerId")
@ManyToMany
@Fetch(FetchMode.SUBSELECT)
@JoinTable(
    name = "customer_dealer_map",
    joinColumns = {
        @JoinColumn(name = "customer_id", referencedColumnName = "id")
    },
    inverseJoinColumns = {
        @JoinColumn(name = "dealer_id", referencedColumnName = "id")
    }
)
private Collection dealers;
```

Found at http://stackoverflow.com/q/17491912/521799

JPA 3.0 Preview – Annotatiomania™

```
@OneToMany @OneToManyMore @AnyOne @AnyBody
@ManyToMany @Many
@Fetch @FetchMany @FetchWithDiscriminator(name = "no_name")
@JoinTable(joinColumns = {
    @JoinColumn(name = "customer_id", referencedColumnName = "id")
})
@PrefetchJoinWithDiscriminator
@IfJoiningAvoidHashJoins @ButUseHashJoinsWhenMoreThan(records = 1000)
@XmlDataTransformable @SpringPrefechAdapter
private Collection employees;
```

Might not be true

Criteria – the naked truth

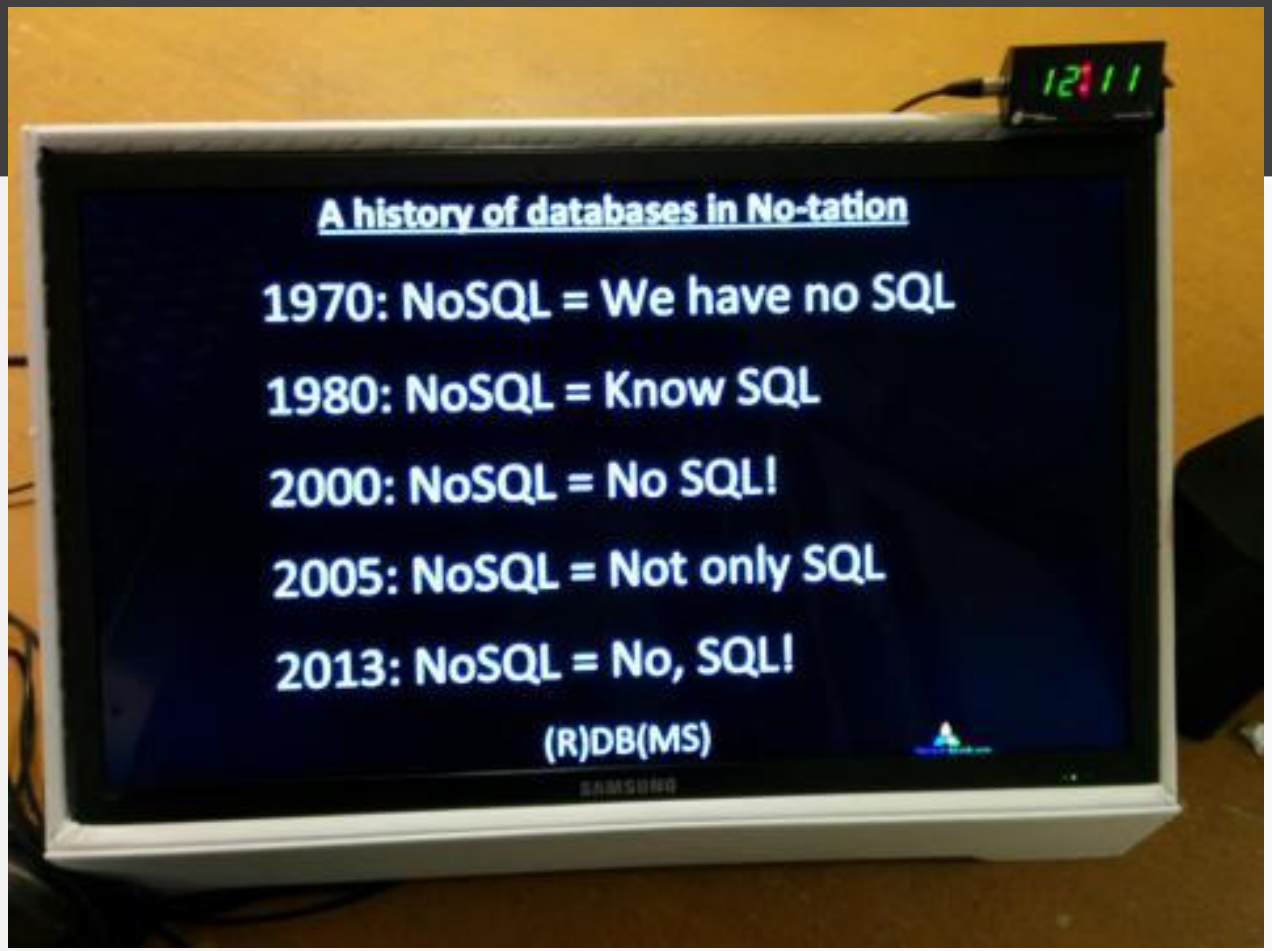
```
EntityManager em= ...
CriteriaBuilder builder = em.getCriteriaBuilder();
CriteriaQuery<Person> criteria = builder.createQuery(Person.class);
Root<Person> person = criteria.from(Person.class);
Predicate condition = builder.gt(person.get(Person_.age), 20);
criteria.where(condition);
TypedQuery<Person>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

Found at http://docs.oracle.com/javaee/6/tutorial/doc/gjrij.html

NoSQL?

...

NoSQL?



Seen at the O'Reilly Strata Conf:
History of NoSQL by Mark Madsen. Picture published by Edd Dumbill

SQL is so much more

TEXT	VOTES	RANK	PERCENT
Hibernate	1383	1	32 %
jOOQ	1029	2	23 %
EclipseLink	881	3	20 %
JDBC	533	4	12 %
Spring JDBC	451	5	10 %

Data may not be accurate…

SQL is so much more

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

The same with jOOQ

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

The same with jOOQ in Scala (!)

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

Examples