

arnaud.nauwynck@gmail.com

# Introduction to Db - Jdbc - JPA - SpringData

This document:

<http://arnaud-nauwynck.github.io/docs/Intro-DB-Jdbc-JPA-SpringData.pdf>



database java persistence

All Videos Images News Shopping More Settings Tools

About 2,150,000 results (0.66 seconds)

**A Simple Example Using JPA | Mapping Objects to Database Tables ...**  
www.informit.com › Articles › Programming › Java ▾  
Jan 6, 2011 - To demonstrate **JPA**, I'll use a simple example of a user authentication, where the credentials are stored in a **database**. ... Without these JAR files, Tomcat will not compile **Java** Server Faces (JSF) applications. ... Apache Derby is a lightweight, all-**Java database** that's good for ...

**Java persistence in database - Stack Overflow**  
stackoverflow.com/questions/735536/java-persistence-in-database ▾  
Apr 9, 2009 - As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this ...

**Developing with Java Persistence**  
[https://docs.oracle.com/cd/E40938\\_01/doc.74/e40142/dev\\_persistence.htm](https://docs.oracle.com/cd/E40938_01/doc.74/e40142/dev_persistence.htm) ▾  
The **Java Persistence** API handles how relational data is mapped to persistent entity objects, how these objects are stored in a relational **database**, and how an ...

(4) < Spring Data



< JPA API - JPQL

(3) < Hibernate/EclipseLink



< JDBC API

(2) < Driver Impl.



SQL Client Driver

(1) < DataBase

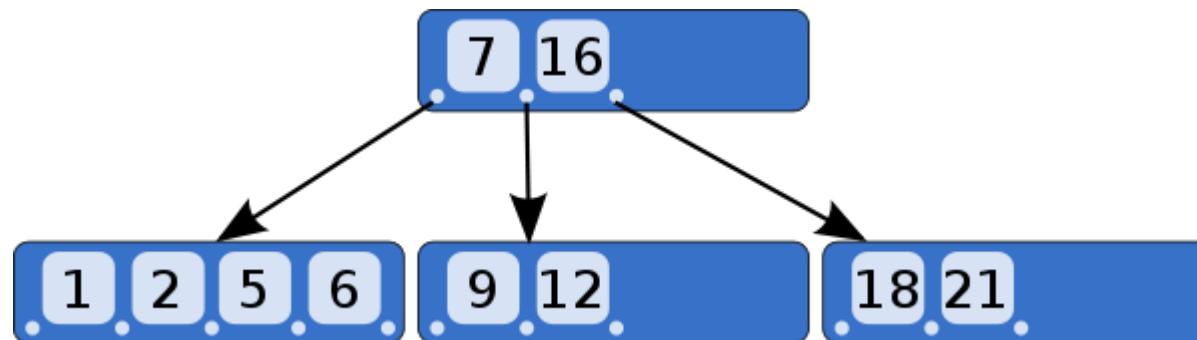
< B-Tree File



Spring JPA  
QueryDsl

Spring JDBC

# B-Tree = Balanced-Tree (not only Binary Tree)





WIKIPEDIA  
The Free Encyclopedia

# B-Tree

a **self-balancing** tree data structure  
that keeps data **sorted**  
and allows **searches**,  
**sequential** access,  
**insertions**, and **deletions**  
in **logarithmic time**.

# Sort/Compare Rows – Columns ?

(col1, col2, col3, col4, col5)

( 123, "Hello", FALSE, +1e3, 'b' )

<?

== ?

> ?

(col1, col2, col3, col4, col5)

( 123, "Text2", FALSE, +1e3, 'b' )

Choose columns ...

exemple:

Sort (col1)

Sort (col3,col5,col2)

# Lexicographical Compare Rows

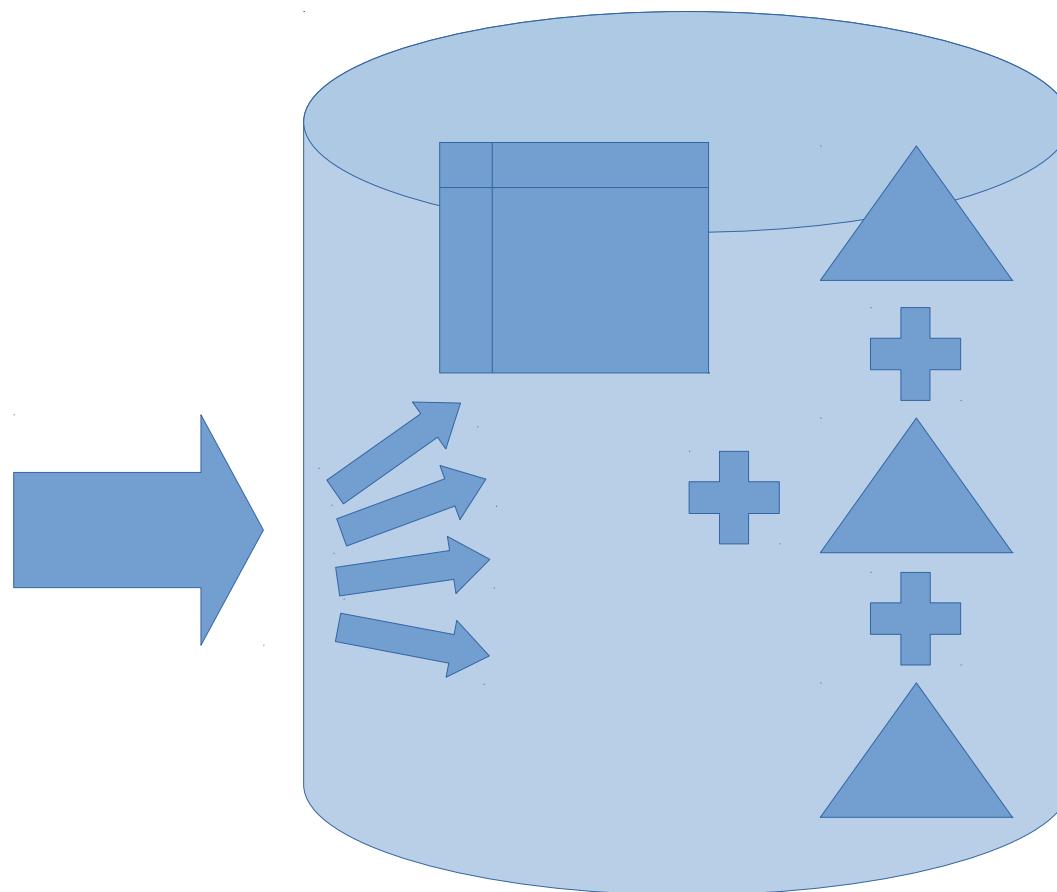
1 Row = N Columns

Compare row by Col1,Col2,ColK :

```
if (a.col1 < b.col1)      => -1
else if (a.col1 > b.col1) => +1
else {
    if (a.col2 < b.col2) ...
        ...
    ...
}
}
```

# B-Tree on (Col1,Col2...) = INDEX

Insert ROW = (ACID : Atomic) Insert Data  
+ Insert Index1 + Insert Index2 + ....

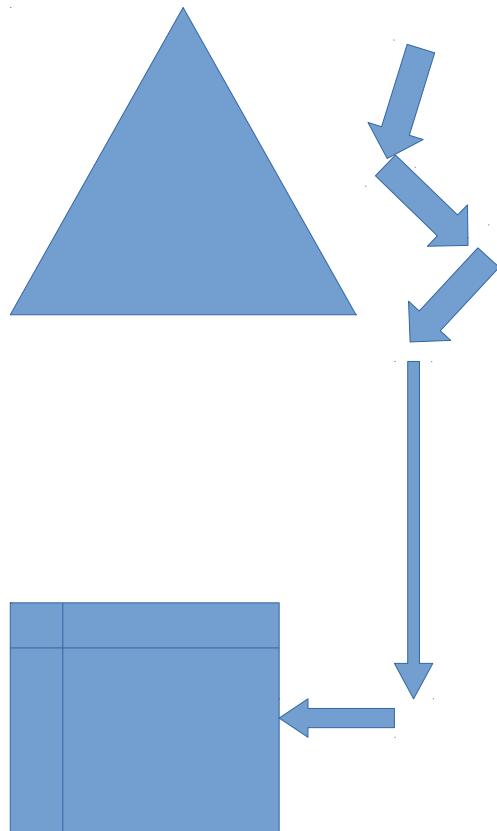


PK INDEX : col (ID)

INDEX2 : (Col2,Col3,ID)

INDEX3 : (Col3,Col2,Col6)

# Select .. where ID = ?



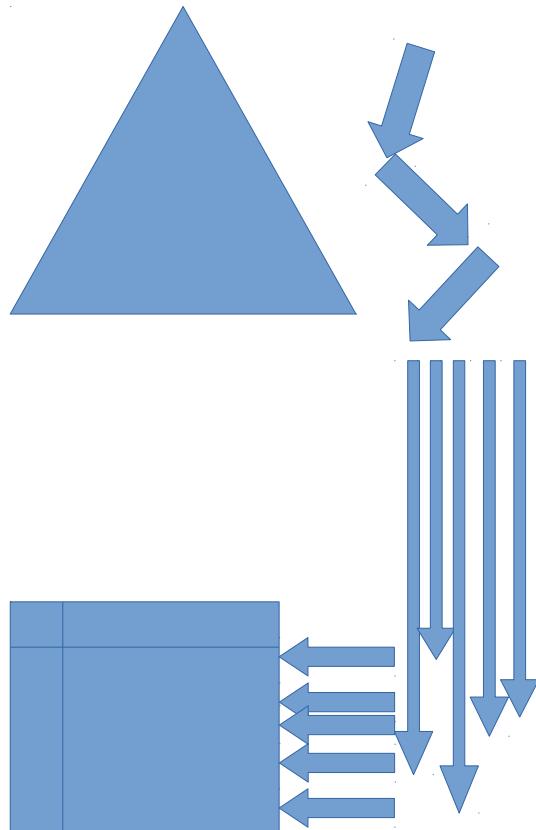
## Execution Plan

**Step 1:** Index Lookup (Unique) by ID  
 $\text{Log}(N)$  logical reads

=> get “rowId” (=address)

**Step 2:** table lookup by RowId ( $=O(1)$ )  
=> get other columns

Select .. where C2=? and C3=? ..  
... with INDEX (C2,C3,otherC...)



## Execution Plan

**Step 1:** Index Range Scan

lookup =  $\text{Log}(N)$  logical reads  
+ scan non unique

=> foreach.. get “rowId” (=address)

**Loop Step 2:** table lookup by RowId ( $=O(1)$ )  
=> get other columns

# Applicable Index(es) ?

for “.. where Col2=? And Col5=?”

PK INDEX : col ~~(ID)~~

INDEX (~~Col2, Col6, Col5~~)  
          Col1

INDEX (Col2, Col5, Col6)

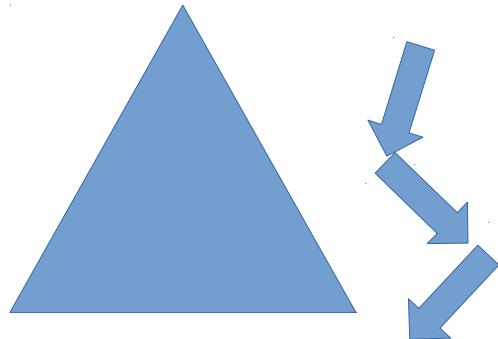
INDEX (Col5, Col2, Col7)

INDEX (~~Col1, Col2, Col5~~)

Applicable =  
Allow Lexicographical  
Top->Down Search



Select C3,C4 where C1=? ,C2=?  
With INDEX(C1,C2,..C3,C4)



## Execution Plan

**Step 1:** Index Lookup by ID  
 $\text{Log}(N)$  logical reads ...unique/scan  
=> read Col3,Col4 value from INDEX  
.. NO need table lookup by rowid

Optim = Index Coverage

# Simple SELECT Query

```
select Col1, Col2 as prettyName, function(col3,col4)  
from Table1 alias1, Table2 alias2,  
      (inner/outer..) join Table3 alias3 on ..
```

**where**

```
    Col1 = 123    -- Literal Value  
    and Col2 = ?0    -- Bind-Variable ... ?0=123  
    and Col3 = Col4  
    and alias1.Col1 = alias2.Col2
```

**Order by** Col2, Col3 **desc**

# Aggregate Query

**select C1, C2, count(\*), avg(Col3), min(Col4) ..**

**from Table ..**

**where**

**...**

**Group by C1, C2**



**Having ..**

**Order by ...**

# Advanced Select Queries

Example 1: **Nested Queries**

**select .. from (select .. where ..) where .. not in (select ..where )**

Example 2: **Analytical Queries**

**select ...**

**analyticalFunc[first,min,max,..](..) OVER (PARTITION BY ..)**

**from ... where ..**

Example 3: **GeoSpatial, FullTextSearch, ..**

# Update Queries (CRUD)

CRUD  
=Create

**INSERT [into] Table .. (col1, col2, ...)**  
**VALUES (?0, ?1, 123, ..)**

Read

Update

Delete

**UPDATE Col1=value1, col2=...**  
from ... where ..

**DELETE [from] Table ..**  
where ...

# Advanced Bulk Update

## with PL/SQL + Array

```
CREATE OR REPLACE PROCEDURE ..(a ARRAY) IS
CURSOR c IS select .. from TABLE(CAST(a ..))
BEGIN
  OPEN c;
  LOOP FETCH c BULK COLLECT INTO ...;
END
```

# Advanced Merge Upsert

Upsert = update or insert

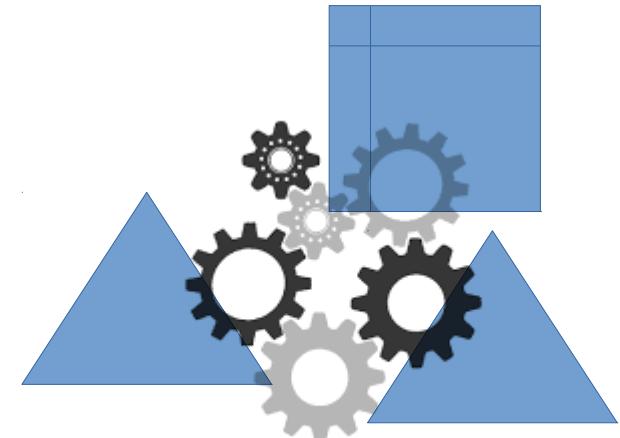
**MERGE INTO ..  
USING ..  
ON ..**

**WHEN MATCHED THEN UPDATE ..  
WHEN NOT MATCHED THEN INSERT..**

# Query Execution Engine

**“Select ..  
from ..  
where ..  
col1='val1'  
and col2=123“**

Execute Query

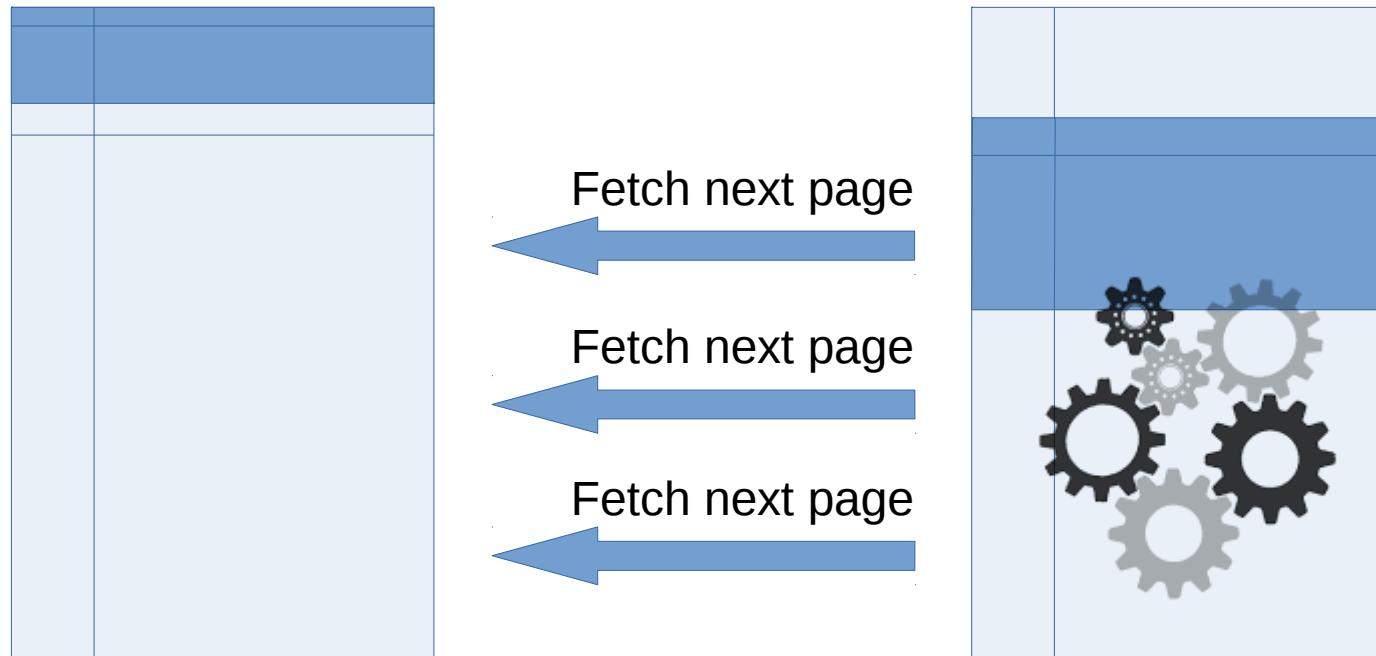


Result : Tabular Data Format = “ResultSet”

# Huge Result .. Server-Side “Cursor”

Result = Partial Data  
+ Cursor (= handle of server-side Partial Data+ Iterator)

Begin “Execute” Query



CLOSE CURSOR !!

# SoftParse – HardParse ... PrepareQuery + BindVariable

**“Select ..  
from ..  
where ..  
col1=?0  
and col2=?1“**

BindVariable:  
set(0, “val1”)  
set(1, 1234)

First Seen ?  
**HARD Parse**

Put in Cache

Compute  
**Execution Plan**



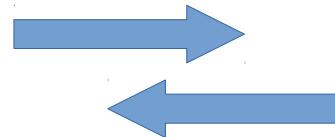
Already Seen ?  
**SOFT Parse**  
= create Cursor



# Query / Prepared Query Execution

```
Select ..  
from ..  
where ..  
col1=?0  
and col2=?1
```

Prepare (SQL)

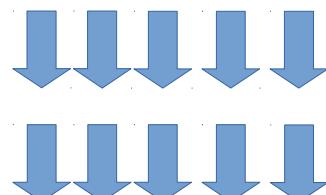


PreparedStatement



BindVariable:  
set(0, "val1")  
set(1, 1234)

ExecuteQuery



Next row  
get col1, get col2 ...  
next row  
...  
...

ResultSet (page1)

Fetch next page

... = JDBC API Explained



**JPA**  
Java Persistence API  
 HIBERNATE

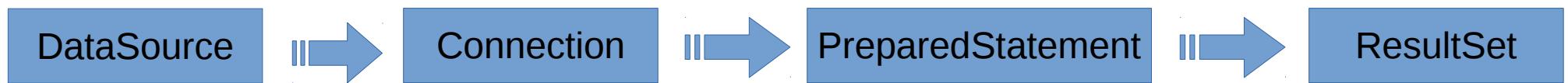
Spring JPA  
QueryDSL



Spring JDBC



```
import java.sql.*;
```



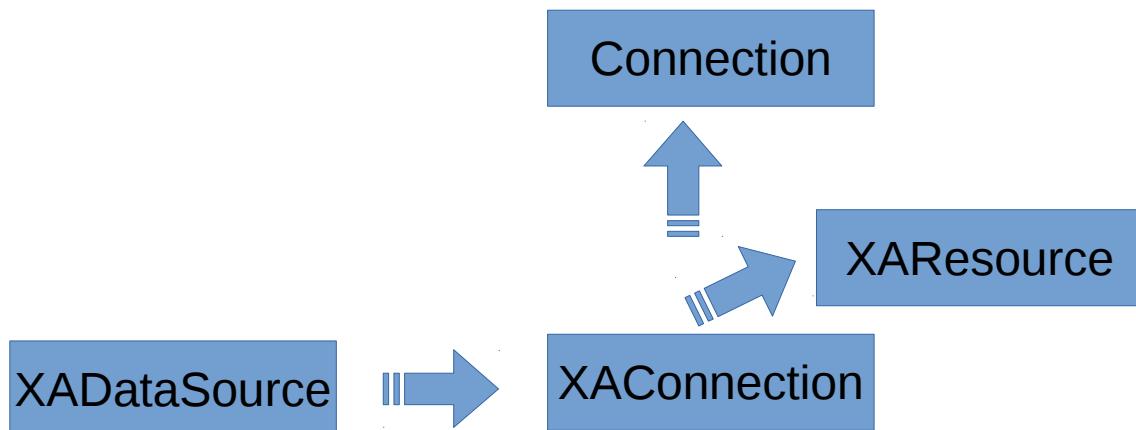
# javax.sql.DataSource

(remark: javax.\* not java.\* )

DataSource     Connection

```
public interface DataSource extends CommonDataSource, Wrapper {  
    Connection getConnection() throws SQLException;  
    Connection getConnection(String username, String password) throws SQLException;  
}  
  
interface CommonDataSource {  
    java.io.PrintWriter getLogWriter() throws SQLException;  
    void setLogWriter(java.io.PrintWriter out) throws SQLException;  
    void setLoginTimeout(int seconds) throws SQLException;  
    int getLoginTimeout() throws SQLException;  
    public Logger getParentLogger() throws SQLFeatureNotSupportedException;  
}  
  
interface Wrapper {  
    <T> T unwrap(java.lang.Class<T> iface) throws java.sql.SQLException;  
    boolean isWrapperFor(java.lang.Class<?> iface) throws java.sql.SQLException;  
}
```

# javax.sql.XADataSource

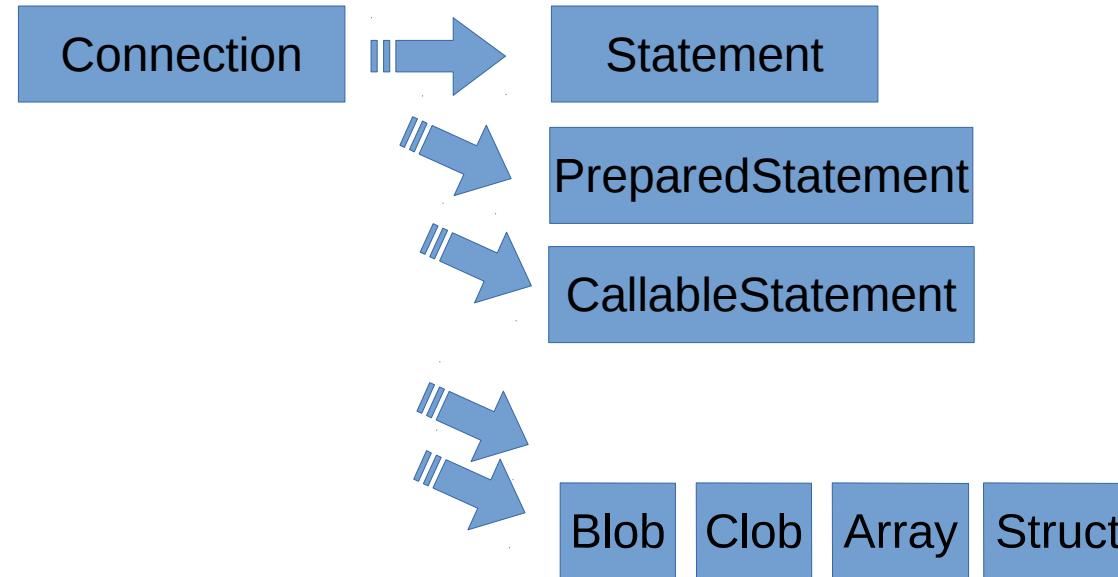


```
interface XADataSource extends CommonDataSource {
    XAConnection getXAConnection() throws SQLException;
    XAConnection getXAConnection(String user, String password) throws SQLException;
}

interface XAConnection extends PooledConnection {
    javax.transaction.xa.XAResource getXAResource() throws SQLException;
}

interface PooledConnection {
    Connection getConnection() throws SQLException;
    void close() throws SQLException;
    void addConnectionEventListener(ConnectionEventListener listener);
    void removeConnectionEventListener(ConnectionEventListener listener);
    public void addStatementEventListener(StatementEventListener listener);
    public void removeStatementEventListener(StatementEventListener listener);
}
```

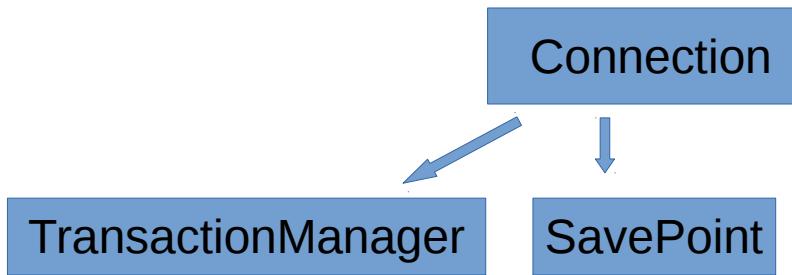
# java.sql.Connection (1/3)



```
public interface Connection {  
  
    void close()  
    boolean isClosed()  
  
    Statement createStatement()  
    PreparedStatement prepareStatement(String sql)  
    CallableStatement prepareCall(String sql)  
    // also variants with create/prepare(int resultSetType, int resultSetConcurrency)..  
  
    Clob createClob()  
    Blob createBlob()  
    NClob createNClob()  
    SQLXML createSQLXML()  
    Array createArrayOf(String typeName, Object[] elements)  
    Struct createStruct(String typeName, Object[] attributes)  
  
    throws SQLException;  
    throws SQLException;  
  
    throws SQLException;  
    throws SQLException;  
    throws SQLException;  
  
    throws SQLException;  
    throws SQLException;  
    throws SQLException;  
    throws SQLException;  
    throws SQLException;  
    throws SQLException;
```

create\* Statement      create\* Data

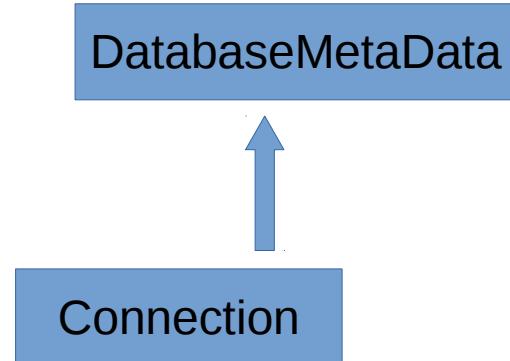
# java.sql.Connection (2/3)



transaction {

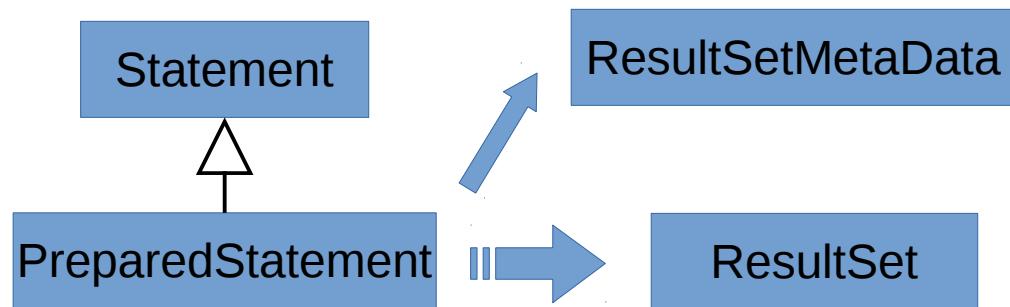
|   |                                   |
|---|-----------------------------------|
| <code>void setReadOnly(boolean readOnly)</code>   | <code>throws SQLException;</code> |
| <code>boolean isReadOnly()</code>   | <code>throws SQLException;</code> |
| <code>void setAutoCommit(boolean autoCommit)</code>   | <code>throws SQLException;</code> |
| <code>boolean getAutoCommit()</code>  | <code>throws SQLException;</code> |
| <br>  |                                   |
| <code>void commit()</code>  | <code>throws SQLException;</code> |
| <code>void rollback()</code>  | <code>throws SQLException;</code> |
| <code>// for prepare() ... see XAConnection sub-class</code>                                |                                   |
| <br>  |                                   |
| <code>Savepoint setSavepoint()</code>   | <code>throws SQLException;</code> |
| <code>Savepoint setSavepoint(String name)</code>  | <code>throws SQLException;</code> |
| <code>void rollback(Savepoint savepoint)</code>   | <code>throws SQLException;</code> |
| <code>void releaseSavepoint(Savepoint savepoint)</code>                                     | <code>throws SQLException;</code> |
| <br>  |                                   |
| <code>// TRANSACTION_NONE, TRANSACTION_READ_UNCOMMITTED, TRANSACTION_READ_COMMITTED,</code> |                                   |
| <code>// TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE</code>                       |                                   |
| <code>void setTransactionIsolation(int level)</code>  | <code>throws SQLException;</code> |
| <code>int getTransactionIsolation()</code>  | <code>throws SQLException;</code> |
| <code>// HOLD_CURSORS_OVER_COMMIT, CLOSE_CURSORS_AT_COMMIT</code>                           |                                   |
| <code>void setHoldability(int holdability)</code>   | <code>throws SQLException;</code> |
| <code>int getHoldability()</code>   | <code>throws SQLException;</code> |
| <br>  |                                   |
| <code>SQLWarning getWarnings()</code>   | <code>throws SQLException;</code> |
| <code>void clearWarnings()</code>   | <code>throws SQLException;</code> |

# java.sql.Connection (3/3)



|          |   |                                       |
|----------|---|---------------------------------------|
| metadata | DatabaseMetaData getMetaData()  | <b>throws</b> SQLException;           |
|          | <b>void</b> setCatalog(String catalog)                                    | <b>throws</b> SQLException;           |
|          | String getCatalog()   | <b>throws</b> SQLException;           |
|          | <b>void</b> setSchema(String schema)                                      | <b>throws</b> SQLException;           |
|          | String getSchema()  | <b>throws</b> SQLException;           |
|          | java.util.Map<String, Class<?>> getTypeMap()                              | <b>throws</b> SQLException;           |
|          | <b>void</b> setTypeMap(java.util.Map<String, Class<?>> map)               | <b>throws</b> SQLException;           |
|          | <b>void</b> setClientInfo(String name, String value)                      | <b>throws</b> SQLClientInfoException; |
|          | <b>void</b> setClientInfo(Properties properties)                          | <b>throws</b> SQLClientInfoException; |
|          | String getClientInfo(String name)   | <b>throws</b> SQLException;           |
|          | Properties getClientInfo()  | <b>throws</b> SQLException;           |
|          | String nativeSQL(String sql)  | <b>throws</b> SQLException;           |
|          | <b>boolean</b> isValid( <b>int</b> timeout)                               | <b>throws</b> SQLException;           |
|          | <b>void</b> abort(Executor executor)                                      | <b>throws</b> SQLException;           |
|          | <b>void</b> setNetworkTimeout(Executor executor, <b>int</b> milliseconds) | <b>throws</b> SQLException;           |
|          | <b>int</b> getNetworkTimeout()  | <b>throws</b> SQLException;           |

# java.sql.PreparedStatement (1/2)



```
public interface PreparedStatement extends Statement {  
    ResultSet executeQuery()  
    int executeUpdate()  
  
    // for batch statement  
    void addBatch()  
  
    boolean execute()  
    default long executeLargeUpdate()  
  
    ResultSetMetaData getMetaData()  
    ParameterMetaData getParameterMetaData()  
}
```

execute\*      {

metadata      {

Set parameters (cf next)

# PreparedStatement (2/2)

ParameterMetaData



PreparedStatement

```
metadata { ParameterMetaData getParameterMetaData() throws SQLException;
    clear void clearParameters() throws SQLException;
    set* void setNull(int parameterIndex, int sqlType) throws SQLException;
    void setObject(int parameterIndex, Object x, int targetSqlType) throws SQLException;
    void setObject(int parameterIndex, Object x) throws SQLException;

    void setBoolean(int parameterIndex, boolean x) throws SQLException;
    void setInt(int parameterIndex, int x) throws SQLException;
    // ... truncated set types: byte, short, long, float, double, BigDecimal
    void setString(int parameterIndex, String x) throws SQLException;
    void setURL(int parameterIndex, java.net.URL x) throws SQLException;
    void setDate(int parameterIndex, java.sql.Date x) throws SQLException;
    void setTime(int parameterIndex, java.sql.Time x) throws SQLException;
    void setTimestamp(int parameterIndex, java.sql.Timestamp x) throws SQLException;

    void setClob(int parameterIndex, Clob x) throws SQLException;
    // ... truncated: setAsciiStream(InputStream), setCharacterStream(Reader), s
    // setBytes(byte[]), setBlob(Blob), BinaryStream(InputStream), setSQLXML..

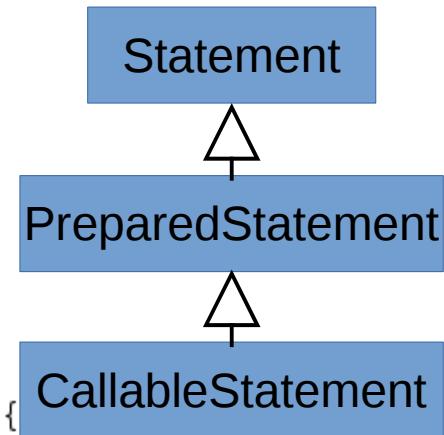
    void setRef(int parameterIndex, Ref x) throws SQLException;
    void setArray(int parameterIndex, Array x) throws SQLException;
    void setRowId(int parameterIndex, RowId x) throws SQLException;
    // ... truncated...
```

# java.sql.Statement



```
public interface Statement extends Wrapper, AutoCloseable {  
    void close() throws SQLException;  
    boolean isClosed() throws SQLException;  
    public void closeOnCompletion() throws SQLException;  
    public boolean isCloseOnCompletion()  
  
    Connection getConnection() throws SQLException;  
  
    void cancel() throws SQLException;  
  
    ResultSet executeQuery(String sql) throws SQLException;  
    int executeUpdate(String sql) throws SQLException;  
  
    void addBatch( String sql ) throws SQLException;  
    void clearBatch() throws SQLException;  
    int[] executeBatch() throws SQLException;  
  
    boolean execute(String sql) throws SQLException;  
    ResultSet getResultSet() throws SQLException;  
    int getUpdateCount() throws SQLException;  
    boolean getMoreResults() throws SQLException;  
    boolean getMoreResults(int current) throws SQLException;  
    ResultSet getGeneratedKeys() throws SQLException;  
  
    // properties with Getter/Setter:  
    // maxFieldSize, maxRows, queryTimeout, fetchDirection, poolable  
  
    // properties with Getter only:  
    // resultSetConcurrency, resultSetType, resultSetHoldability
```

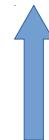
# java.sql.CallableStatement



```
public interface CallableStatement extends PreparedStatement {  
  
    void registerOutParameter(int parameterIndex, int sqlType)          tl  
    void registerOutParameter(int parameterIndex, int sqlType, int scale) tl  
    void registerOutParameter (int parameterIndex, int sqlType, String typeName) tl  
    void registerOutParameter(String parameterName, int sqlType)          tl  
    void registerOutParameter(String parameterName, int sqlType, int scale) tl  
    void registerOutParameter (String parameterName, int sqlType, String typeName) tl  
  
    void setInt(String parameterName, int x)           throws SQLException;  
    void setString(String parameterName, String x)      throws SQLException;  
    // truncated.. setBoolean,Byte,Short,Date,... Blob,Clob,Array..  
  
  
    boolean wasNull() throws SQLException;  
    void setNull (String parameterName, int sqlType, String typeName) throws SQLException  
  
    int getInt(int parameterIndex)                      throws SQLException;  
    String getString(int parameterIndex)                throws SQLException;  
    // truncated.. getBoolean,Byte,Short,Date,... Blob,Clob,Array..  
  
    int getInt(String parameterName)                  throws SQLException;  
    String getString(String parameterName)            throws SQLException;  
    // truncated.. getBoolean,Byte,Short,Date,... Blob,Clob,Array..
```

# java.sql.ResultSet (1/2)

ResultSetMetaData



ResultSet

```
public interface ResultSet extends Wrapper, AutoCloseable {  
  
    void close() throws SQLException;  
    boolean isClosed() throws SQLException;  
  
    boolean next() throws SQLException;  
  
    // access results by column index  
    int getInt(int columnIndex) throws SQLException;  
    String getString(int columnIndex) throws SQLException;  
    // .. truncate: boolean, short, long, float .. bytes,Blob,Clob..  
  
    // access results by column label  
    int getInt(String columnName) throws SQLException;  
    String getString(String columnName) throws SQLException;  
    // .. truncate: boolean, short, long, float .. bytes,Blob,Clob..  
  
    boolean wasNull() throws SQLException;  
  
    SQLWarning getWarnings() throws SQLException;  
    void clearWarnings() throws SQLException;  
  
    String getCursorName() throws SQLException;  
    ResultSetMetaData getMetaData() throws SQLException;  
    int findColumn(String columnName) throws SQLException;  
  
    // Properties: fetchDirection, fetchSize,  
    // with getter only: type, concurrency, holdability
```

# java.sql.ResultSet (2/2)

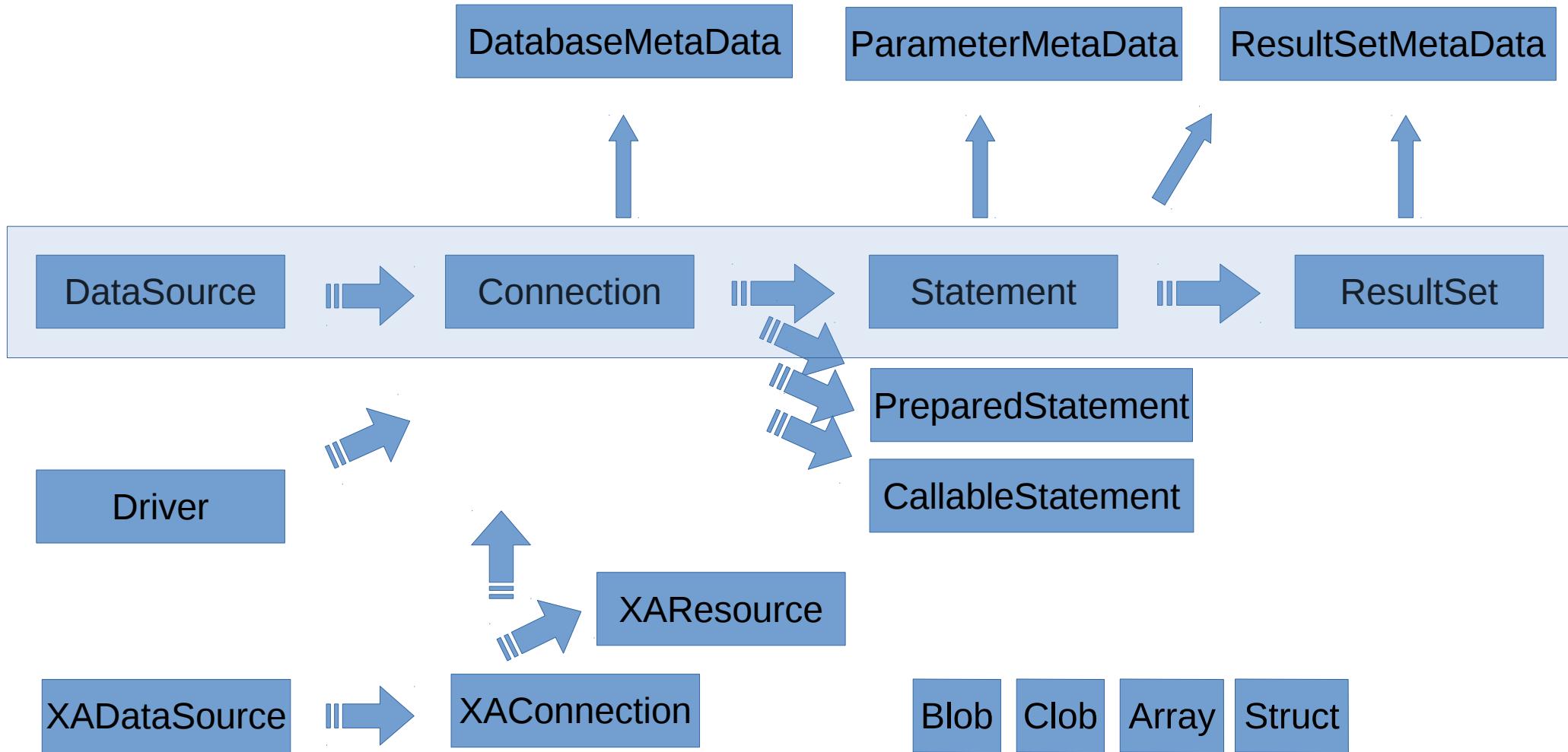
```
// Navigation
//-----
boolean isBeforeFirst()           throws SQLException;
// truncate: isAfterLast,iFirst,isLast,beforeFirst,afterLast,first,last
int getRow()                   throws SQLException;
boolean absolute( int row )    throws SQLException;
boolean relative( int rows )   throws SQLException;
boolean previous()            throws SQLException;

// Update columns by index / by name
//-----
boolean rowUpdated()            throws SQLException;
boolean rowInserted()          throws SQLException;
boolean rowDeleted()           throws SQLException;

void updateNull(int columnIndex)  throws SQLException;
void updateInt(int columnIndex, int x)  throws SQLException;
// .. truncate: boolean, short, long, float .. bytes,Blob,Clob..

void updateNull(String columnName) throws SQLException;
void updateInt(String columnName, int x) throws SQLException;
// .. truncate: boolean, short, long, float .. bytes,Blob,Clob..
```

# import java.sql.\*; (full)



# Jdbc Database App

```
<!-- choose your poison (Postgresql, MySql, Oracle, H2, ...) -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

# DataSource Code Sample

Using  
explicit Transaction  
+ commit/rollback

```
Connection conn = dataSource.getConnection();
try {
    conn.setAutoCommit(false);

    // ** do work with connection **

    conn.commit();
} catch(Exception ex) {
    conn.rollback();
} finally {
    conn.close();
}
```

Using try-close

```
try (Connection conn = dataSource.getConnection()) {
    // ** do work with connection **

}
```



Rule for ALL Resources :  
If You Open It => You Close It

# H2 DataSource HelloWorld

```
import java.io.File;
import java.sql.Connection;
import java.sql.ResultSet;

import org.h2.jdbcx.JdbcConnectionPool;
import org.junit.Test;

public class H2DataSourceTest {

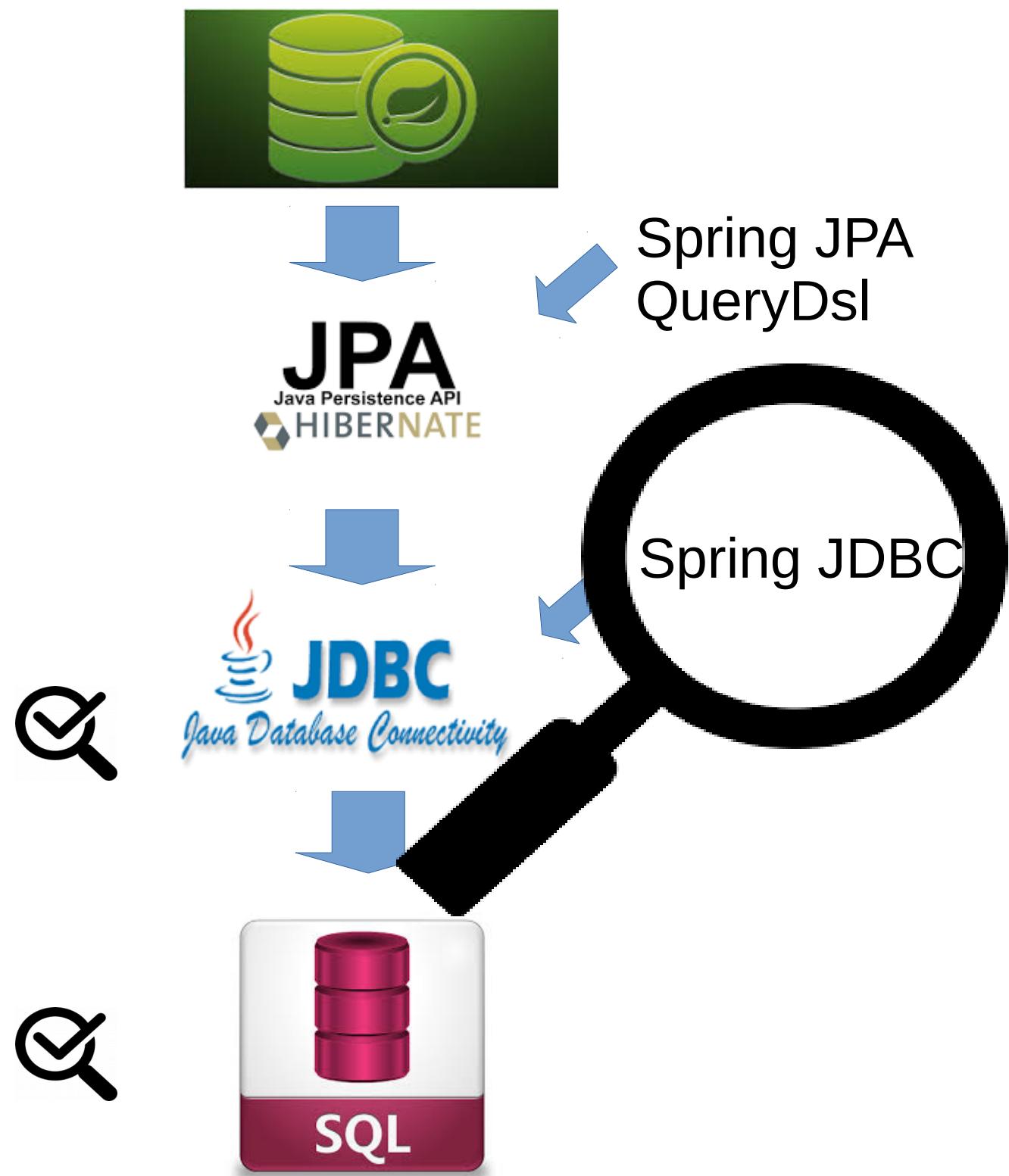
    @Test
    public void testDataSource() throws Exception {
        File dir = new File("src/test/data/dbs");
        if (!dir.exists()) {
            dir.mkdirs();
        }
        String jdbcUrl = "jdbc:h2:" + dir.getAbsolutePath() + "/db1";
        String username = "sa", password = "sa";

        JdbcConnectionPool dataSource = JdbcConnectionPool.create(jdbcUrl, username, password);

        try (Connection cx = dataSource.getConnection()) {
            ResultSet rs = cx.prepareStatement("select 1").executeQuery();
            if (rs.next()) {
                int check = rs.getInt(1);
                System.out.println("OK " + check);
            }
        }
        dataSource.dispose();
    }
}
```

# DataSource using Spring

# PreparedStatement Sample



# Spring-Jdbc Database App

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<!-- implicit
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jdbc</artifactId>
</dependency>
<dependency>
    -->

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

# Sample SpringBoot Junit DataSource

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=DataSourceTestConfiguration.class)
public class H2DataSourceSpringTest {

    @Configuration
    // @EnableAutoConfiguration() ... implies DataSourceAutoConfiguration + many others
    @Import(DataSourceAutoConfiguration.class)
    // cf breakpoint in DataSourceAutoConfiguration$NonEmbeddedConfiguration.dataSource()
    // => read file conf/application.yml, use (or default) properties:
    // spring:
    //   datasource:
    //     url:
    //     username:
    //     password:
    public static class DataSourceTestConfiguration {
    }

    @Autowired
    private DataSource dataSource;

    @Test
    public void testDataSource() throws Exception {
        try (Connection cx = dataSource.getConnection()) {
            ResultSet rs = cx.prepareStatement("select 1").executeQuery();
            if (rs.next()) {
                int check = rs.getInt(1);
                System.out.println("OK " + check);
            }
        }
    }
}
```

# @Bean for Multi DataSources

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=MultiDataSourcesTestConfiguration.class)
public class H2MultiDataSourcesSpringTest {

    private static File dir = new File("src/test/data/dbs");

    @Configuration
    // @EnableAutoConfiguration() ... implies DataSourceAutoConfiguration + many others
    // @Import(DataSourceAutoConfiguration.class)
    public static class MultiDataSourcesTestConfiguration {
        @Bean
        public DataSource dataSource1() {
            String jdbcUrl = "jdbc:h2:" + dir.getAbsolutePath() + "/db1";
            return JdbcConnectionPool.create(jdbcUrl, "sa", "sa");
        }
        @Bean
        public DataSource dataSource2() {
            String jdbcUrl = "jdbc:h2:" + dir.getAbsolutePath() + "/db2";
            return JdbcConnectionPool.create(jdbcUrl, "sa", "sa");
        }
    }

    @Autowired @Named("dataSource1")
    private DataSource dataSource1;

    @Autowired @Named("dataSource2")
    private DataSource dataSource2;

    @Test
    public void testDataSources() throws Exception {
        Assert.assertNotSame(dataSource1, dataSource2);
        checkDataSource(dataSource1);
        checkDataSource(dataSource2);
    }
    private void checkDataSource(DataSource ds) throws SQLException {
        try (Connection cx = ds.getConnection()) {
```

# (Reminder) DataSource Try-Finally Close

Using  
explicit Transaction  
+ commit/rollback

```
Connection conn = dataSource.getConnection();
try {
    conn.setAutoCommit(false);

    // ** do work with connection **

    conn.commit();
} catch(Exception ex) {
    conn.rollback();
} finally {
    conn.close();
}
```

Using try-close

```
try (Connection conn = dataSource.getConnection()) {
    // ** do work with connection **

}
```



Rule for ALL Resources :  
If You Open It => You Close It

# Same using Template Spring-Jdbc

```
@Inject
private JdbcTemplate jdbcTemplate; // = new JdbcTemplate(dataSource)

// @Transactional // NOT yet...
public Object doWithConnection() throws Exception {
    Object res = jdbcTemplate.execute(new ConnectionCallback<Object>() {
        @Override
        public Object doInConnection(Connection con) throws SQLException, DataException {
            // ** do work with connection **
            return null;
        }
    });
}

// same with jdk8 lambda (+ explicit type-checking for ambiguous overload)
Object res2 = jdbcTemplate.execute((Connection conn) -> {
    // NOT Transactional => maybe get a different connection!
    // ** do work with connection **
    return null;
});
```

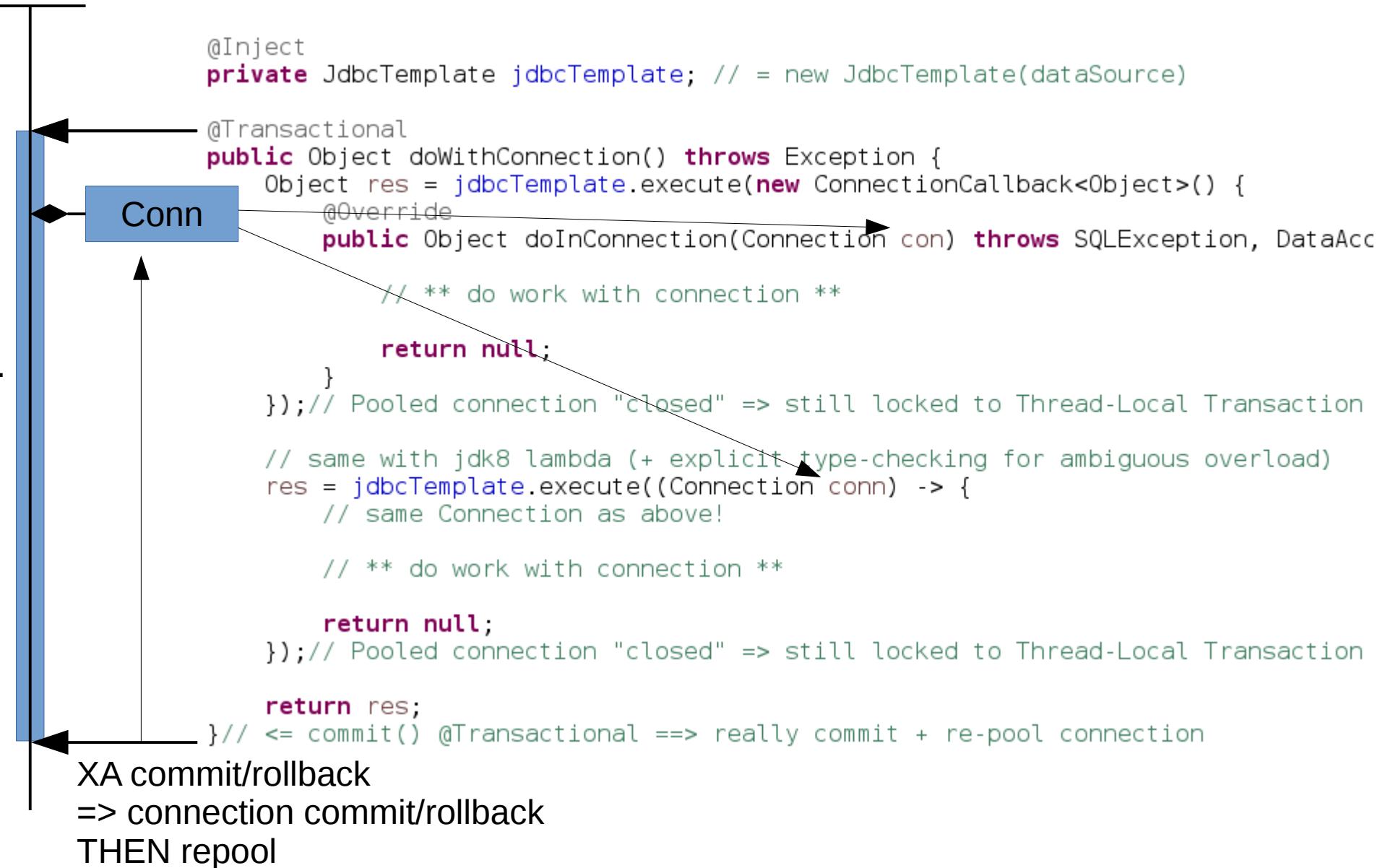
Equivalent try-finally

Same with Lambda syntax

2n execution ... different connection

# PooledConnection + Transaction = Thread-Locked : reuse

Thread-12



# (Reminder) try-finally Connection + try-finally Statement

```
Connection cx = dataSource.getConnection();
try {

    try (PreparedStatement stmt = cx.prepareStatement("select 1")) {
        try (ResultSet rs = stmt.executeQuery()) {

            if (rs.next()) {
                Assert.assertEquals(1, rs.getInt(1));
            }

        } // <= close rs
    } // <= close stmt

    cx.commit();
} catch(Exception ex) {
    cx.rollback();
} finally {
    cx.close(); // close cx after commit/rollback
}
```

# spring-jdbc

```
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;

@Component
public class JdbcBatchCommand implements CommandLineRunner {

    @Autowired
    protected JdbcTemplate jdbcTemplate;

    @Override
    public void run(String... args) throws Exception {
        jdbcTemplate.execute("select 1");

        String msg = jdbcTemplate.execute("select 'Hello' as msg", (PreparedStatement pstmt) -> {
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                return rs.getString("msg");
            }
            return null;
        });
        assert "Hello".equals(msg);

    }
}
```

# Run Jdbc App

The screenshot shows a Java application running in a debugger. The code is a `CommandLineRunner` implementation named `JdbcBatchCommand`. It uses `JdbcTemplate` to execute a query and assert the result.

```
package com.example;

import java.sql.PreparedStatement;
import java.sql.ResultSet;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;

@Component
public class JdbcBatchCommand implements CommandLineRunner {

    @Autowired
    protected JdbcTemplate jdbcTemplate;

    @Override
    public void run(String... args) throws Exception {
        jdbcTemplate.execute("select 1");

        String msg = jdbcTemplate.execute("select 'Hello' as msg", (PreparedStatement pstmt) -> {
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                return rs.getString("msg");
            }
            return null;
        });
        assertEquals("Hello", msg);
    }
}
```

The code is annotated with `@Component`, `@Autowired`, and `@Override`. The `run` method executes a query and then asserts that the result is "Hello". The assertion is highlighted in green, indicating it has passed. A tooltip for the variable `msg` shows its value as "Hello" (id=57), hash=69609650, and value= (id=63).

# Springboot built-in supports JTA @Transactional

```
import org.springframework.transaction.annotation.Transactional;

@Component
@Transactional
public class JdbcBatchCommand implements CommandLineRunner {

    @Autowired
    protected JdbcTemplate jdbcTemplate;

    @Override
    // @Transactional // repeat on each method is useless, say once on class
    public void run(String... args) throws Exception {
        jdbcTemplate.execute("select 1");

        String msg = jdbcTemplate.execute("select 'Hello' as msg", (PreparedStatement pstmt) -> {
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                return rs.getString("msg");
            }
            return null;
        });
        assert "Hello".equals(msg);
    } // after this "}" line, spring will commit (or rollback ACID Transaction)
}
```

# @Transactional JUST Works

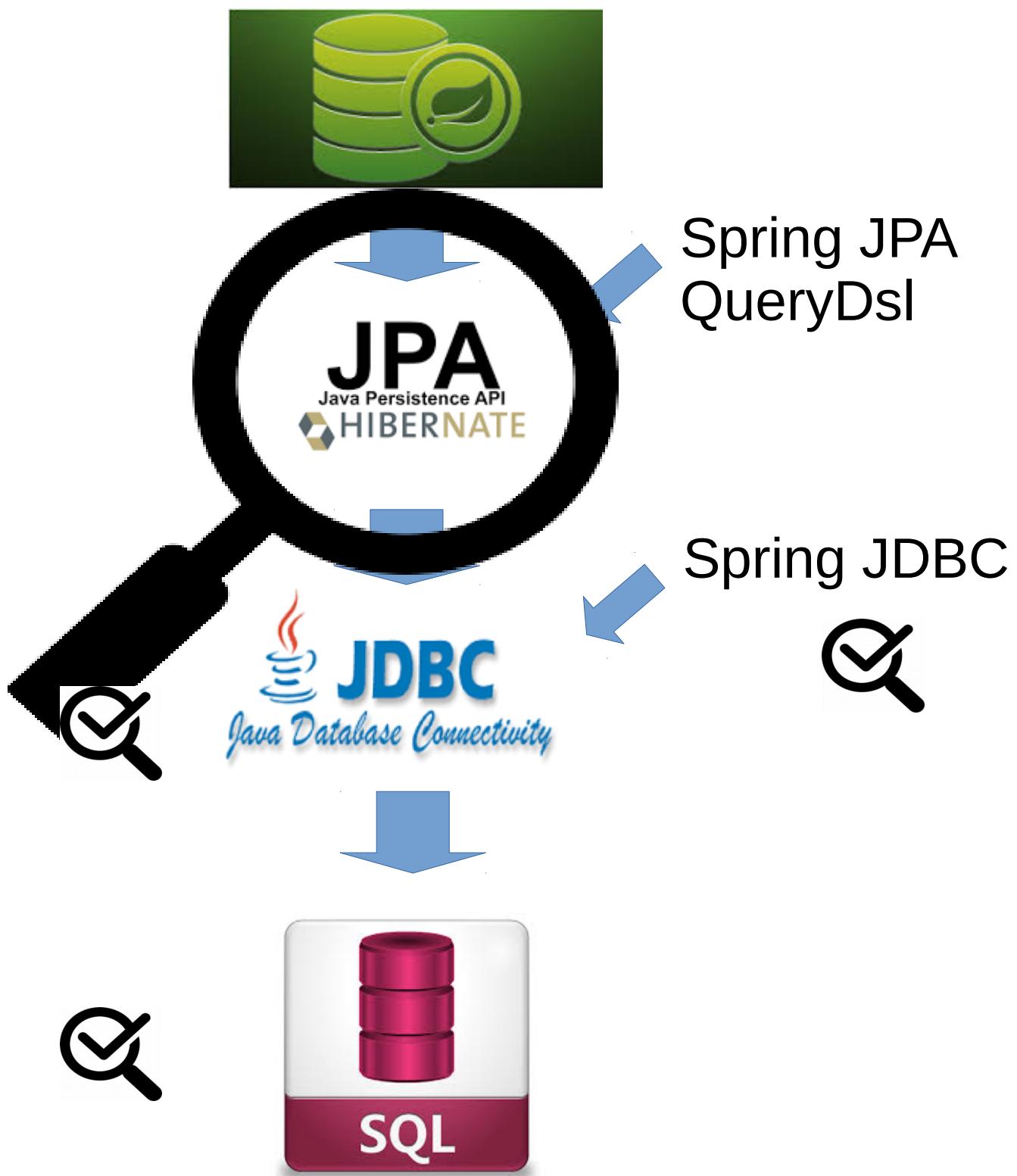
The screenshot shows a Java application running in a debugger. The stack trace indicates that the application has stopped at line 30 of the `JdbcBatchCommand` class. The source code for `JdbcBatchCommand.java` is visible below, showing a method that returns a string from a database query and includes an assertion.

**Stack Trace:**

- Thread [restartedMain] (Suspended (breakpoint at line 30 in JdbcBatchCommand))
- JdbcBatchCommand.run(String...) line: 30
- JdbcBatchCommand\$\$FastClassBySpringCGLIB\$\$749c813d.invoke(int, Object, Object[]) line: not available
- MethodProxy.invoke(Object, Object[]) line: 204
- CglibAopProxy\$CglibMethodInvocation.invokeJoinpoint() line: 720
- CglibAopProxy\$CglibMethodInvocation(ReflectiveMethodInvocation).proceed() line: 157
- TransactionInterceptor\$1.proceedWithInvocation() line: 99
- TransactionInterceptor(TransactionAspectSupport).invokeWithinTransaction(Method, Class<?>, InvocationCallback) line: 282
- TransactionInterceptor.invoke(MethodInvocation) line: 96
- CglibAopProxy\$CglibMethodInvocation(ReflectiveMethodInvocation).proceed() line: 179
- CglibAopProxy\$DynamicAdvisedInterceptor.intercept(Object, Method, Object[], MethodProxy) line: 655
- JdbcBatchCommand\$\$EnhancerBySpringCGLIB\$\$ce3bc846.run(String...) line: not available
- SpringApplication.callRunner(CommandLineRunner, ApplicationArguments) line: 800

**JdbcBatchCommand.java:**

```
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return rs.getString("msg");
        }
        return null;
    });
    assert "Hello".equals(msg);
} // after this "}" line, spring will commit (or rollback ACID Transaction)
```



# JPA (with springboot data)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <exclusions>
        <!-- default use hibernate ... explicit exclude to use another -->
        <exclusion>
            <artifactId>hibernate-entitymanager</artifactId>
            <groupId>org.hibernate</groupId>
        </exclusion>
    </exclusions>
</dependency>

<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>eclipselink</artifactId>
    <version>${eclipselink.version}</version>
</dependency>
```

# EntityManager (1/2)

```
public interface EntityManager {  
    public void close();  
    public boolean isOpen();  
  
    // CRUD : Create, Read, (No-Update,) Delete  
    public void persist(Object entity); //=INSERT + attach  
    public void remove(Object entity); //=DELETE + detach  
  
    // Query by ID  
    public <T> T find(Class<T> entityClass, Object primaryKey); // cf also variants with props, lockMode  
    public <T> T getReference(Class<T> entityClass, Object primaryKey); //=find + lazy  
    // Queries using JPA-ql  
    public Query createQuery(String qlString);  
    public <T> TypedQuery<T> createQuery(String qlString, Class<T> resultClass);  
    public Query createNamedQuery(String name);  
    public <T> TypedQuery<T> createNamedQuery(String name, Class<T> resultClass);  
    // Queries using SQL  
    public Query createNativeQuery(String sqlString);  
    public Query createNativeQuery(String sqlString, Class resultClass);  
    public Query createNativeQuery(String sqlString, String resultSetMapping);  
    public StoredProcedureQuery createNamedStoredProcedureQuery(String name);  
    public StoredProcedureQuery createStoredProcedureQuery(String procedureName);  
    public StoredProcedureQuery createStoredProcedureQuery(String procedureName, Class... resultClasses);  
    public StoredProcedureQuery createStoredProcedureQuery(String procedureName, String... resultSetMappings);  
    // Queries using dynamic criteria  
    public CriteriaBuilder getCriteriaBuilder();  
    public <T> TypedQuery<T> createQuery(CriteriaQuery<T> criteriaQuery);  
    public Query createQuery(CriteriaUpdate updateQuery);  
    public Query createQuery(CriteriaDelete deleteQuery);
```

# EntityManager (2/2)

```
public <T> T merge(T entity); // SHOULD not use.. prefer DT0<->Entity Mapper + setter on Entity
public void refresh(Object entity); // cf also variants with props, lockMode
public boolean contains(Object entity);
public void detach(Object entity);
public void clear(); // detach all + forget XA changes

public void flush();
public void setFlushMode(FlushModeType flushMode);
public FlushModeType getFlushMode();

public void lock(Object entity, LockModeType lockMode); // cf also variant with props
public LockModeType getLockMode(Object entity);

public Metamodel getMetamodel();
public EntityManagerFactory getEntityManagerFactory();
public void setProperty(String propertyName, Object value);
public Map<String, Object> getProperties();
public <T> T unwrap(Class<T> cls);

public void joinTransaction();
public boolean isJoinedToTransaction();
public EntityTransaction getTransaction();

public <T> EntityGraph<T> createEntityGraph(Class<T> rootType);
public EntityGraph<?> createEntityGraph(String graphName);
public EntityGraph<?> getEntityGraph(String graphName);
public <T> List<EntityGraph<? super T>> getEntityGraphs(Class<T> entityClass);
```

# What is an “Entity” ?

A class with @Entity

```
import javax.persistence.Entity;  
import javax.persistence.Id;  
  
@Entity  
public class Simple {
```

And an @Id

```
@Id  
private int id;  
  
public Simple() {  
}  
}
```

# Javax.persistence.\* Annotations

```
@Target({TYPE})
@Retention(RUNTIME)
public @interface Inheritance {
    // enum { SINGLE_TABLE, TABLE_PER_CLASS, JOINED }
    InheritanceType strategy() default SINGLE_TABLE;
}
```

```
@Documented
@Target(TYPE)
@Retention(RUNTIME)
public @interface Entity {
    String name() default "";
}
```

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface Id {
}

@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface Version {}
```

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface ManyToOne {
    Class targetEntity() default void.class;
    CascadeType[] cascade() default {};
    FetchType fetch() default EAGER;
    boolean optional() default true;
}
```

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface OneToMany {
    Class targetEntity() default void.class;
    CascadeType[] cascade() default {};
    FetchType fetch() default LAZY;
    String mappedBy() default "";
    boolean orphanRemoval() default false;
}
```

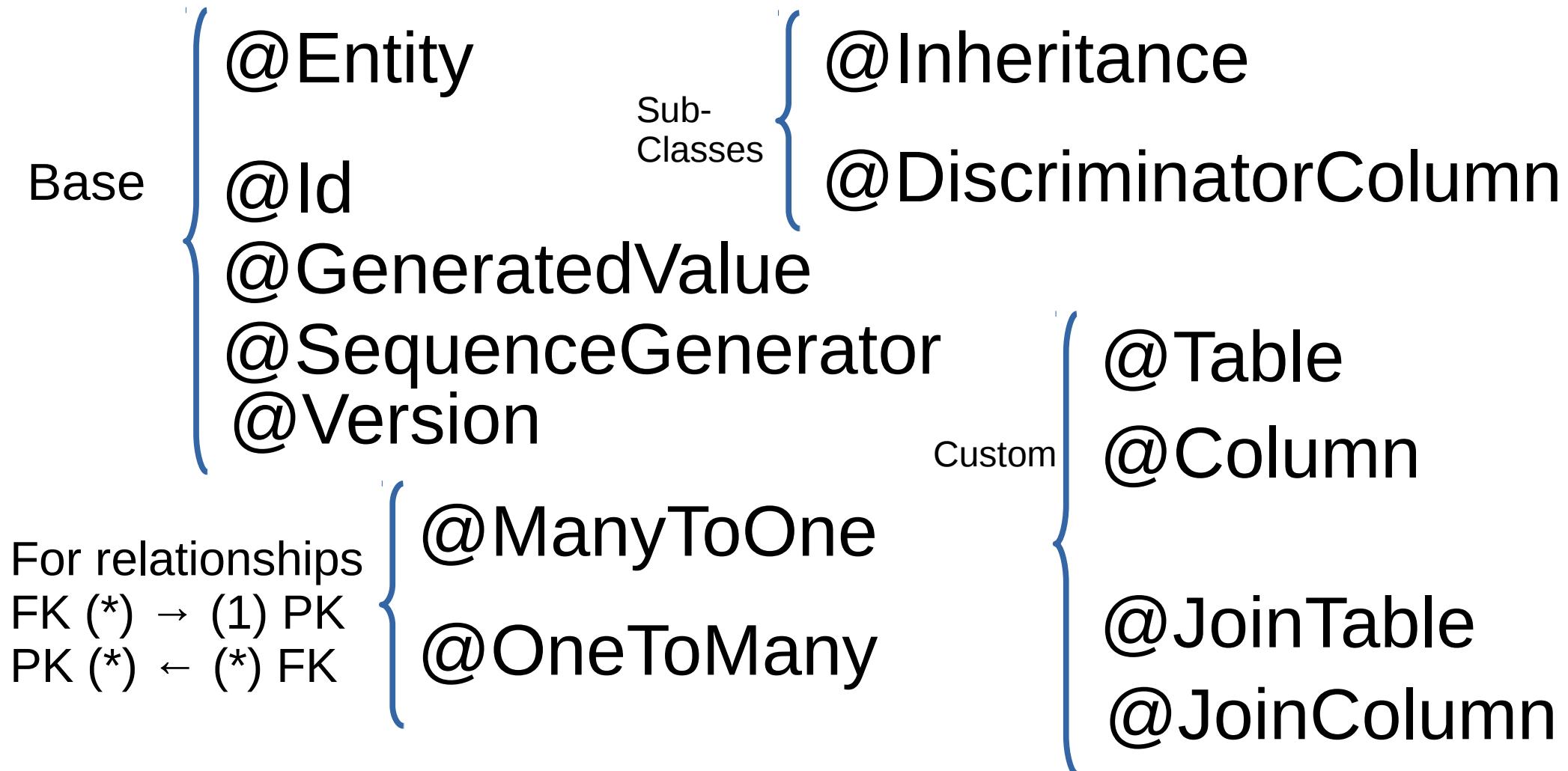
# How many java.persistence.\* Annotations ?

```
$ cd src/main/java/javax/persistence  
$ find . -name \*.java -exec grep -H '@Retention' {} \;  
  
$ find . -name \*.java -exec grep -H '@Retention' {} \; \  
| cut -d: -f1 | sed 's|\.\.\(.*\)\.java|\1|g' | wc -l
```

90

```
@PersistenceUnits @NamedEntityGraphs @SequenceGenerator @DiscriminatorColumn @PersistenceProperty  
@NamedQuery @JoinTable @GeneratedValue @SecondaryTables @DiscriminatorValue @Converter  
@StoredProcedureParameter @PreRemove @MapKeyClass @CollectionTable @Version @PrimaryKeyJoinColumn  
@IdClass @QueryHint @ElementCollection @Temporal @PersistenceContext @ManyToOne @Table @Convert @Transient  
@NamedSubgraph @MapKeyEnumerated @PersistenceContexts @MapsId @Basic @SqlResultSetMappings @AssociationOverride  
@Entity @OrderColumn @AssociationOverride @PersistenceUnit @Inheritance @PrePersist @OrderBy @Id @Cacheable  
@NamedNativeQuery @PostRemove @PreUpdate @Access @MapKey @NamedQueries @FieldResult @MappedSuperclass  
@Embeddable @Lob @ExcludeSuperclassListeners @EntityListeners @JoinColumn @SecondaryTable @OneToOne  
@Enumerated @EntityResult @PrimaryKeyJoinColumn @PostPersist @NamedAttributeNode @MapKeyColumn @UniqueConstraint  
@JoinColumns @AttributeOverrides @ColumnResult @Converts @NamedEntityGraph @Index @TableGenerator  
@StaticMetamodel @ManyToMany @AttributeOverride @PostUpdate @NamedStoredProcedureQuery @MapKeyTemporal  
@SqlResultSetMapping @PostLoad @MapKeyJoinColumn @ForeignKey @Column @Embedded @EmbeddedId @NamedNativeQuery  
@ExcludeDefaultListeners @NamedStoredProcedureQueries @ConstructorResult @OneToMany @MapKeyJoinColumn
```

# Sufficient @Annotations to know?



# Why putting @Column & @Table ?

```
@Entity
@Table(name="NORMAL_TABLE_NAME") // was implicit: "BUT VERY STRANGE ENTITY NAME"
public class ButVeryStrangeEntityName {
    @Id private int id;
}

@Entity
@Table(schema="myschema") // implicit name="NORMAL_ENTITY"
class NormalEntity {

    @Id private int id;

    @Column(name="FIELD_1") // was implicit; "FIELD1"
    private int field1;

    @Column(length=10, precision=5, scale=3) // redefine numeric precision
    private double someValue;

    // redefine database constraints
    @Column(unique=true, nullable=false, insertable=false, updatable=false)
    private String name;
}
```

# @Id with Sequence Generator

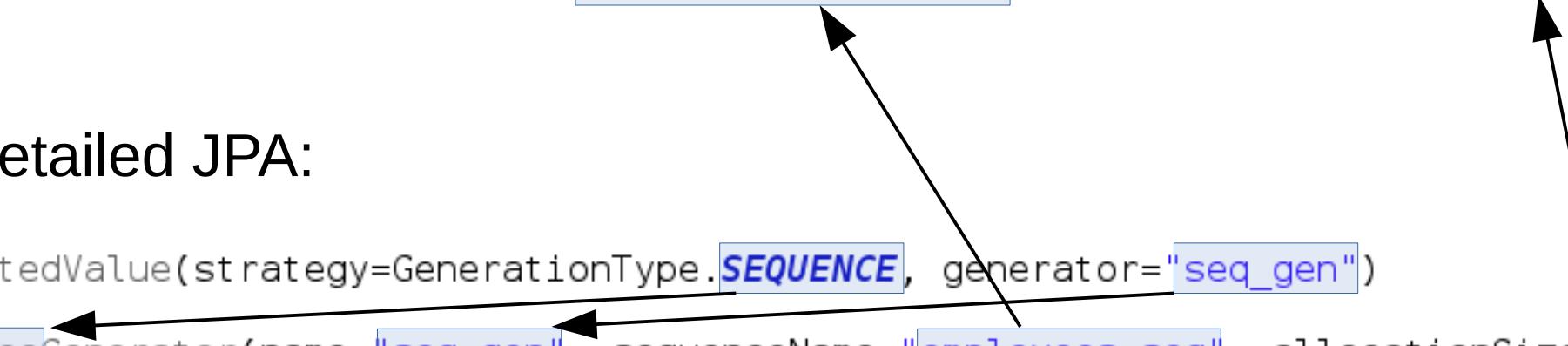
```
@Id  
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="seq_gen")  
@SequenceGenerator(name="seq_gen", sequenceName="employees_seq", allocationSize=10)  
private int id;
```

SQL:

CREATE SEQUENCE employees\_seq INCREMENT BY 10;

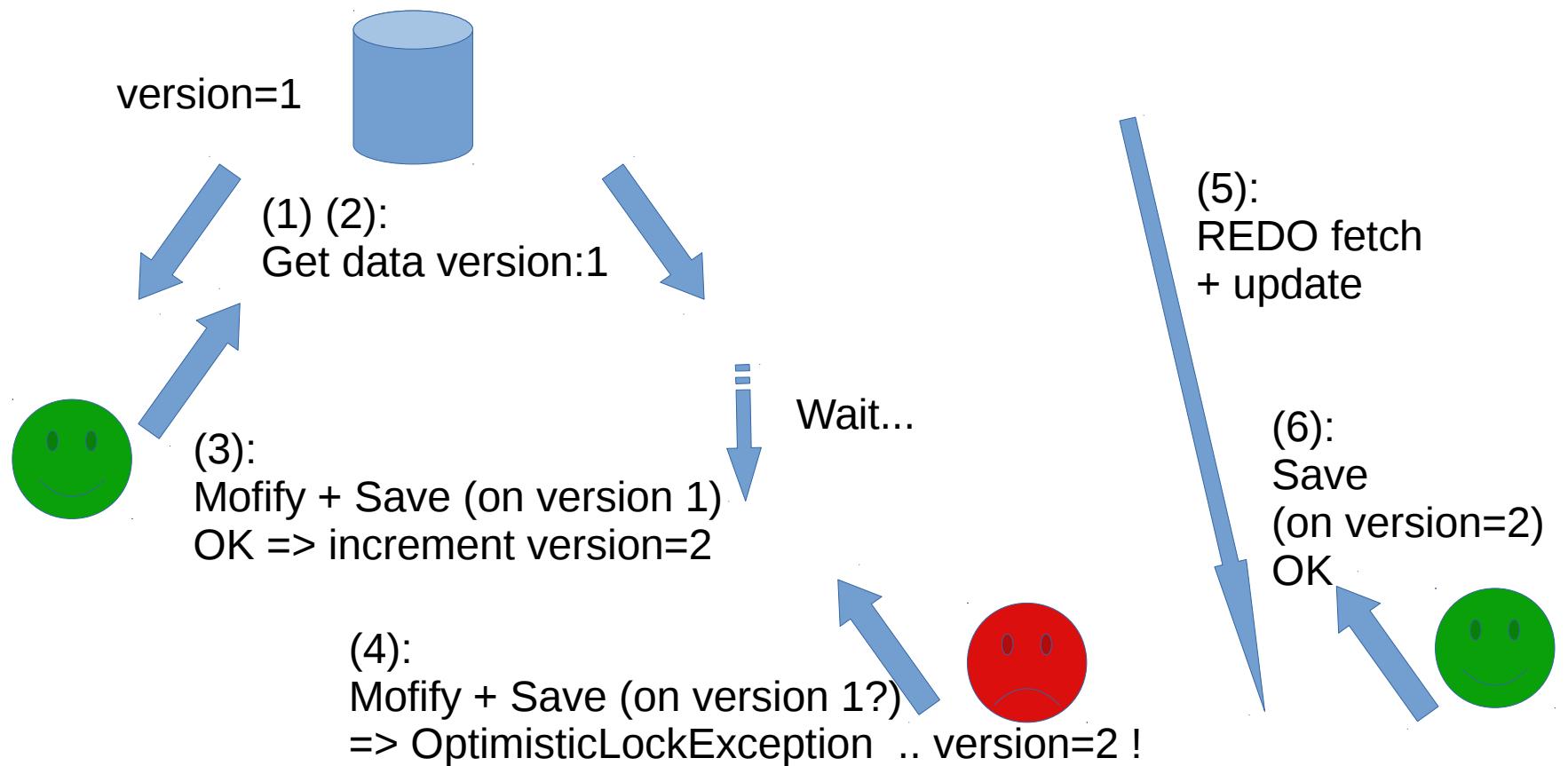
Detailed JPA:

```
@Id  
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="seq_gen")  
@SequenceGenerator(name="seq_gen", sequenceName="employees_seq", allocationSize=10)
```



# @Version ?

whenever overwriting modified value without reading  
=> **throw new OptimisticLockException();**



# Oracle Sample DataBase

[https://github.com/oracle/db-sample-schemas/human\\_resources/hr\\_cre.sql](https://github.com/oracle/db-sample-schemas/human_resources/hr_cre.sql)

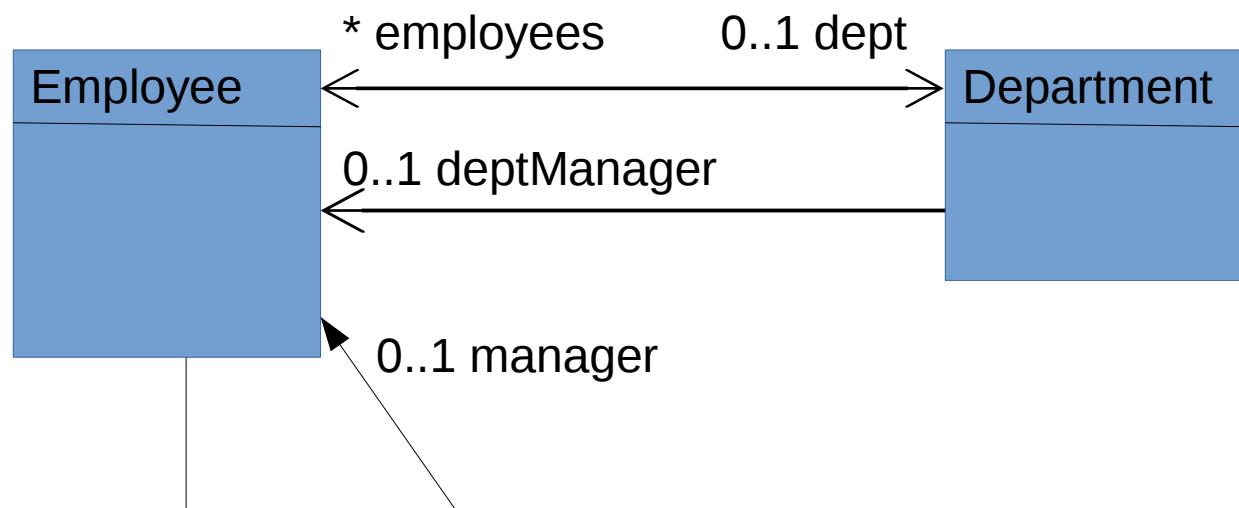
```
CREATE TABLE departments(
    department_id      NUMBER(4),
    department_name   VARCHAR2(30) CONSTRAINT dept_name_nn NOT NULL,
    manager_id        NUMBER(6),
    CONSTRAINT dept_id_pk KEY (department_id),
    CONSTRAINT dept_loc_fk FOREIGN KEY (location_id) REFERENCES locations (location_id),
    CONSTRAINT dept_mgr_fk FOREIGN KEY (manager_id) REFERENCES employees (employee_id),
);

CREATE UNIQUE INDEX dept_id_pk ON departments (department_id);
CREATE SEQUENCE departments_seq INCREMENT BY 10;

CREATE TABLE employees (
    employee_id        NUMBER(6),
    first_name         VARCHAR2(20),
    last_name          VARCHAR2(25) CONSTRAINT emp_last_name_nn NOT NULL,
    email              VARCHAR2(25) CONSTRAINT emp_email_nn NOT NULL,
    manager_id         NUMBER(6),
    department_id      NUMBER(4),
    CONSTRAINT emp_email_uk UNIQUE (email),
    CONSTRAINT emp_emp_id_pk PRIMARY KEY (employee_id),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id) REFERENCES departments,
    CONSTRAINT emp_manager_fk FOREIGN KEY (manager_id) REFERENCES employees
);

CREATE UNIQUE INDEX emp_emp_id_pk ON employees (employee_id) ;
CREATE SEQUENCE employees_seq;
```

# UML Employee - Department



# @Entity Employee-Department

```
@Data // lombok getter, setter...
@Entity
public class Employee {

    @Id
    private int id;

    @Version
    private int version;

    private String firstName, lastName, email, address;

    @ManyToOne
    private Department department;

    @ManyToOne
    private Employee manager;

}
```



Database table “EMPLOYEE”  
id (PK), version, first\_name, last\_name, ....  
department\_id (FK department.id)  
manager\_id (FK employee.id)

```
@Data // lombok getter, setter
@Entity
public class Department {

    @Id
    private int id;

    @Version
    private int version;

    private String name;

    @OneToMany(mappedBy="department")
    private List<Employee> employees;

    @ManyToOne
    private Employee deptManager;

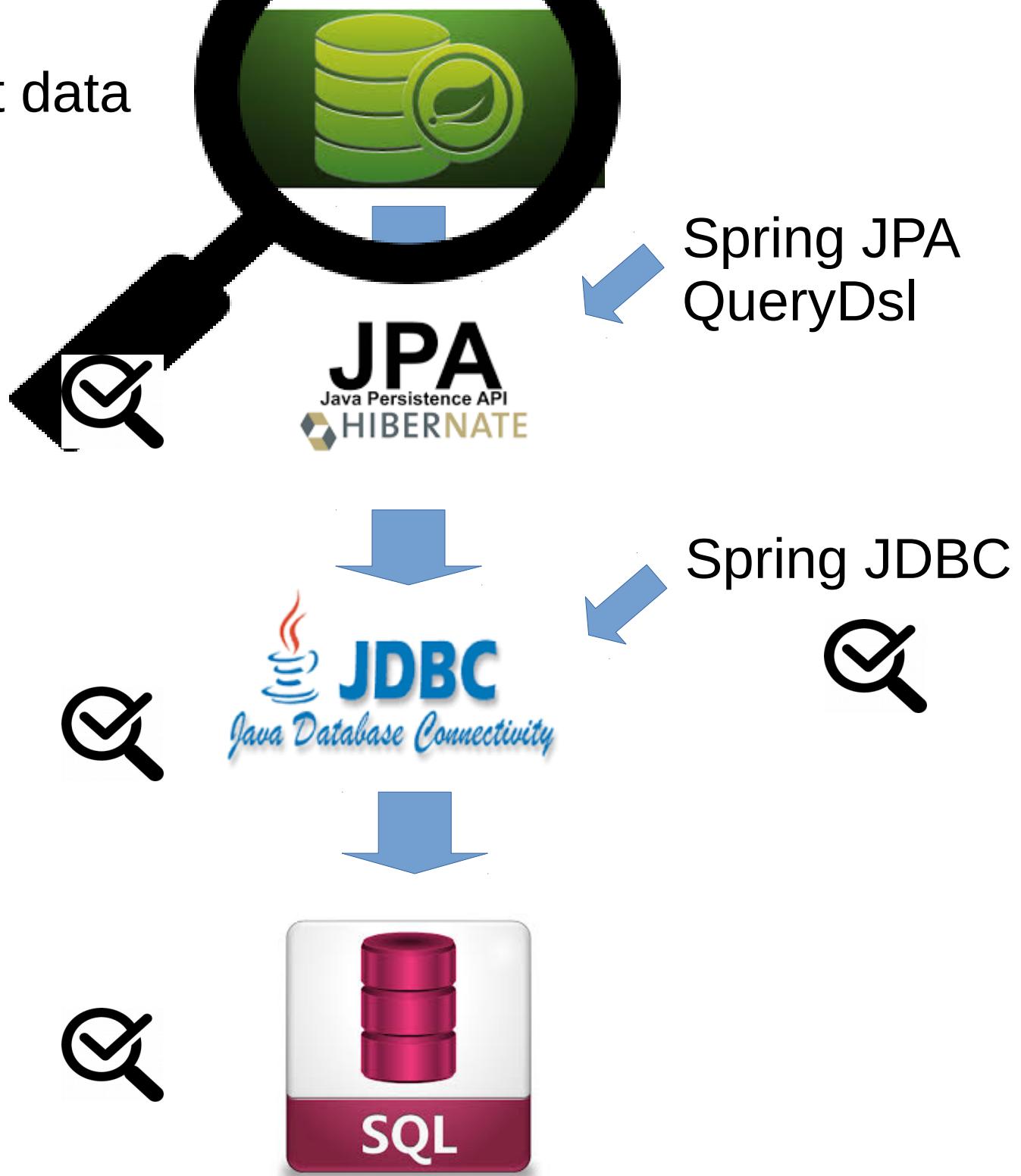
}
```



Database table “DEPARTMENT”  
id (PK), version, name,  
department\_manager\_id (FK employee.id)

# CRUD JPA Example ... corresponding SQL

springboot data



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<!-- implicit
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
</dependency>
-->
```

# Repository

```
import org.springframework.data.jpa.repository.JpaRepository;  
  
import fr.an.tests.eclipselink.domain.City;  
  
interface CityRepository extends JpaRepository<City, Long> {  
    // Small is Beautiful  
}
```

# extends JpaRepository

```
interface CityRepository extends JpaRepository<City, Long> {
    // extends JpaRepository<TEntity, TId> ... =>

    △ @Override City findOne(Long id);
    △ @Override City getOne(Long id); // throws NoSuchEntityException if not found
    △ @Override boolean exists(Long id);

    △ @Override List<City> findAll();
    △ @Override List<City> findAll(Sort sort);
    △ @Override List<City> findAll(Iterable<Long> ids);
    △ @Override Page<City> findAll(Pageable pageable);
    △ @Override long count();

    △ @Override void flush();
    △ @Override <S extends City> S save(S entity);
    △ @Override <S extends City> List<S> save(Iterable<S> entities);
    △ @Override <S extends City> S saveAndFlush(S entity);

    △ @Override void delete(Long id);
    △ @Override void delete(City entity);
    △ @Override void delete(Iterable<? extends City> entities);
    △ @Override void deleteAll();
    △ @Override void deleteInBatch(Iterable<City> entities);
    △ @Override void deleteAllInBatch();

}
```

# Repository with extra Finders

## findBy XXX And YYY

```
interface CityRepository extends JpaRepository<City, Long> {  
  
    Page<City> findByNameContainingAndCountryContainingAllIgnoringCase(String name,  
                           String country, Pageable pageable);  
  
    City findByNameAndCountryAllIgnoringCase(String name, String country);  
  
    List<City> findByNameAndCountry(String name, String country);  
}
```

# Small + Custom + Almost Complete (see also next for QueryDsl)

EmployeeRepository.java

```
package com.example.repository;

import org.springframework.data.jpa.repository.JpaRepository;

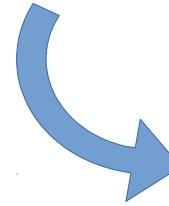
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

    Employee findOneByEmail(String email);

}
```

By naming convention .. Equivalent to:  
“Select \* from EMPLOYEE where email=?”

Extends JpaRepository  
built-in CRUD ...  
findAll, by Page, save, delete...  
(no update, use Setters)



```
com.example.repository ~ com.example.repository.EmployeeRepository.java
EmployeeRepository ~ com.example.repository
    ● findOneByEmail(String) : Employee ~ com.example.repository.EmployeeRepository
    ● findAll() : List<T> ~ org.springframework.data.jpa.repository.JpaRepository
    ● findAll(Sort) : List<T> ~ org.springframework.data.jpa.repository.JpaRepository
    ● findAll(Iterable<ID>) : List<T> ~ org.springframework.data.jpa.repository.JpaRepository
    ● save(Iterable<S>) <S extends T> : List<S> ~ org.springframework.data.jpa.repository.JpaRepository
    ● flush() : void ~ org.springframework.data.jpa.repository.JpaRepository
    ● saveAndFlush(S) <S extends T> : S ~ org.springframework.data.jpa.repository.JpaRepository
    ● deleteInBatch(Iterable<T>) : void ~ org.springframework.data.jpa.repository.JpaRepository
    ● deleteAllInBatch() : void ~ org.springframework.data.jpa.repository.JpaRepository
    ● getOne(ID) : T ~ org.springframework.data.jpa.repository.JpaRepository
    ● findAll(Example<S>) <S extends T> : List<S> ~ org.springframework.data.jpa.repository.JpaRepository
    ● findAll(Example<S>, Sort) <S extends T> : List<S> ~ org.springframework.data.jpa.repository.JpaRepository
    ● findAll(Sort) : Iterable<T> ~ org.springframework.data.repository.PagingAndSortingRepository
    ● findAll(Pageable) : Page<T> ~ org.springframework.data.repository.PagingAndSortingRepository
    ● save(S) <S extends T> : S ~ org.springframework.data.repository.CrudRepository
    ● save(Iterable<S>) <S extends T> : Iterable<S> ~ org.springframework.data.repository.CrudRepository
    ● findOne(ID) : T ~ org.springframework.data.repository.CrudRepository
    ● exists(ID) : boolean ~ org.springframework.data.repository.CrudRepository
    ● findAll() : Iterable<T> ~ org.springframework.data.repository.CrudRepository
    ● findAll(Iterable<ID>) : Iterable<T> ~ org.springframework.data.repository.CrudRepository
    ● count() : long ~ org.springframework.data.repository.CrudRepository
    ● delete(ID) : void ~ org.springframework.data.repository.CrudRepository
    ● delete(T) : void ~ org.springframework.data.repository.CrudRepository
    ● delete(Iterable<T>) : void ~ org.springframework.data.repository.CrudRepository
    ● deleteAll() : void ~ org.springframework.data.repository.CrudRepository
    ● findOne(Example<S>) <S extends T> : S ~ org.springframework.data.repository.QueryByExampleExecutor
    ● findAll(Example<S>) <S extends T> : Iterable<S> ~ org.springframework.data.repository.QueryByExampleExecutor
    ● findAll(Example<S>, Sort) <S extends T> : Iterable<S> ~ org.springframework.data.repository.QueryByExampleExecutor
```

# Sample Code using springboot-data Repository

```
@Component
@Transactional
public class JPARepositoryBatchCommand implements CommandLineRunner {

    private static final Logger LOG = LoggerFactory.getLogger(JPARepositoryBatchCommand.class);

    @Autowired
    protected EmployeeRepository employeeDAO;

    @Override
    public void run(String... args) throws Exception {
        Employee empById1 = employeeDAO.findOne(1);
        if (empById1 != null) LOG.info("Hello employee #1 : " + empById1.getFirstName() + " " +
Employee empJohn = employeeDAO.findOneByEmail("john.smith@gmail.com");
        if (empJohn == null) {
            empJohn = new Employee();
            empJohn.setEmail("john.smith@gmail.com");
            employeeDAO.save(empJohn);
        }
    }
}
```

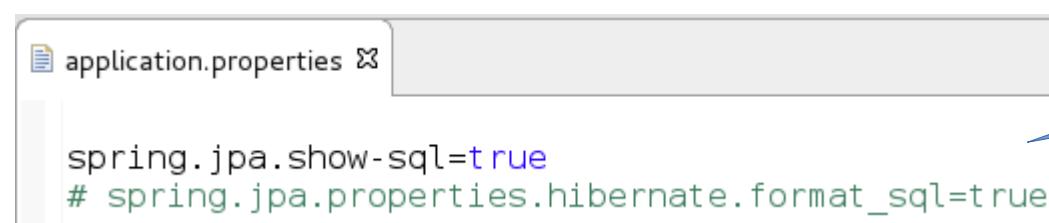
# Springboot @JPA configuration

```
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
//{@EnableJpaRepositories(basePackages="com.example.repository")}  
//{@EntityScan(basePackages="com.example.domain")}  
public class DemoJPAConfig {  
}
```

springboot dark magic  
not even 1 line .. all optional !!!

```
import org.springframework.boot.autoconfigure.domain.EntityScan;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  
  
@Configuration  
@EnableJpaRepositories(basePackages="com.example.repository")  
@EntityScan(basePackages="com.example.domain")  
public class DemoJPAConfig {  
}
```

Useless equivalent  
2 implicit lines



application.properties

```
spring.jpa.show-sql=true  
# spring.jpa.properties.hibernate.format_sql=true
```

Optional  
for debug (see next)

# Springboot hibernate... “JUST work”



```
2016-11-17 22:42:01.527 INFO 9504 --- [ restartedMain] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
Hibernate: select employee0_.id as id1_10_, employee0_.address as address2_10_, employee0_.birth_date as birth_da3_10_, employee0_.department_id as department10_
2016-11-17 22:48:26.530 INFO 9504 --- [ restartedMain] o.h.h.i.QueryTranslatorFactoryInitiator : HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select employee0_.id as id1_1_, employee0_.address as address2_1_, employee0_.birth_date as birth_da3_1_, employee0_.department_id as department1_1_, employee0_
Hibernate: insert into employee (address, birth_date, department_id, email, first_name, last_name, version, id) values (?, ?, ?, ?, ?, ?, ?, ?)
2016-11-17 22:48:28.194 INFO 9504 --- [ restartedMain] com.example.DemoApplication           : Started DemoApplication in 391.821 seconds (JVM running for 392.191)
```

CRUD ...  
select \* from EMPLOYEE where ...  
insert into EMPLOYEE (...) values (...)

# When/How/Where are my “create Table ()” ???

Tables are created/updated at startup

```
2016-11-17 22:41:59.753 INFO 9504 --- [ restartedMain] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.0.1.Final}
2016-11-17 22:41:59.851 INFO 9504 --- [ restartedMain] org.hibernate.dialect.Dialect   : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2016-11-17 22:42:00.269 INFO 9504 --- [ restartedMain] org.hibernate.tool.hbm2ddl.SchemaExport : HHH000227: Running hbm2ddl schema export
Hibernate: drop table department if exists
Hibernate: drop table employee if exists
Hibernate: drop table employee_projects if exists
Hibernate: drop table user_project if exists
Hibernate: create table department (id integer not null, name varchar(255), primary key (id))
Hibernate: create table employee (id integer not null, address varchar(255), birth_date timestamp, email varchar(255), first_name varchar(255), last_name varchar(255),
Hibernate: create table employee_projects (employee_id integer not null, projects_id integer not null)
Hibernate: create table user_project (id integer not null, employee_id integer, primary key (id))
Hibernate: alter table employee_projects add constraint UK_4jypfcavfedhsivky8co9wvqa unique (projects_id)
Hibernate: alter table employee add constraint FKbejtwvg9bxus2mffsm3swj3u9 foreign key (department_id) references department
Hibernate: alter table employee_projects add constraint FK25ggdalg559udqdvhwu3p4j3 foreign key (projects_id) references user_project
Hibernate: alter table employee_projects add constraint FK97jl81fsrbblkqfoqwg2o7yps foreign key (employee_id) references employee
Hibernate: alter table user_project add constraint FKmlfr43vjmqqlnaleuarwhhveq foreign key (employee_id) references employee
2016-11-17 22:42:00.286 INFO 9504 --- [ restartedMain] org.hibernate.tool.hbm2ddl.SchemaExport : HHH000230: Schema export complete
2016-11-17 22:42:00.318 INFO 9504 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
```

Detected H2 Database SQL language

Also create PK/FK indexes

# How??

The screenshot shows a Java debugger interface with a stack trace. The stack trace starts with a call from `SessionImpl$IdentifierLoadAccessImpl<T>.load(Serializable)` at line 2687. It then traces back through several methods in `SessionImpl`, `EntityManagerImpl`, and `NativeMethodAccessorImpl`. The stack trace continues through `DelegatingMethodAccessorImpl`, `Method.invoke`, and `SharedEntityManagerCreator$SharedEntityManagerInvocationHandler.invoke` before reaching a call to `$Proxy81.find` at line 298. Finally, it traces back to `SimpleJpaRepository<T, ID>.findOne(ID)` at line 239, which is annotated with `@Override` and `@SuppressWarnings("unchecked")`. The code editor window below shows the implementation of the `load` method in `SessionImpl.class`.

Debug

- Thread [main] (Running)
- Thread [Thread-0] (Running)
- Thread [restartedMain] (Suspended)

```
SessionImpl$IdentifierLoadAccessImpl<T>.load(Serializable) line: 2687
SessionImpl.get(Class<T>, Serializable) line: 975
EntityManagerImpl(AbstractEntityManagerImpl).find(Class<A>, Object, LockModeType, Map<String, Object>) line: 1075
EntityManagerImpl(AbstractEntityManagerImpl).find(Class<T>, Object, Map<String, Object>) line: 1039
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
Method.invoke(Object, Object...) line: 483
SharedEntityManagerCreator$SharedEntityManagerInvocationHandler.invoke(Object, Method, Object[]) line: 298
$Proxy81.find(Class, Object, Map) line: not available
SimpleJpaRepository<T, ID>.findOne(ID) line: 239
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
```

JPARepositoryBatchCommand.java SessionImpl.class

```
@Override
@SuppressWarnings("unchecked")
public final T load(Serializable id) {
    if ( this.lockOptions != null ) {
        LoadEvent event = new LoadEvent( id, entityPersister.getEntityName(), lock(
            fireLoad( event, LoadEventListener.GET ) );
        return (T) event.getResult();
    }
}
```

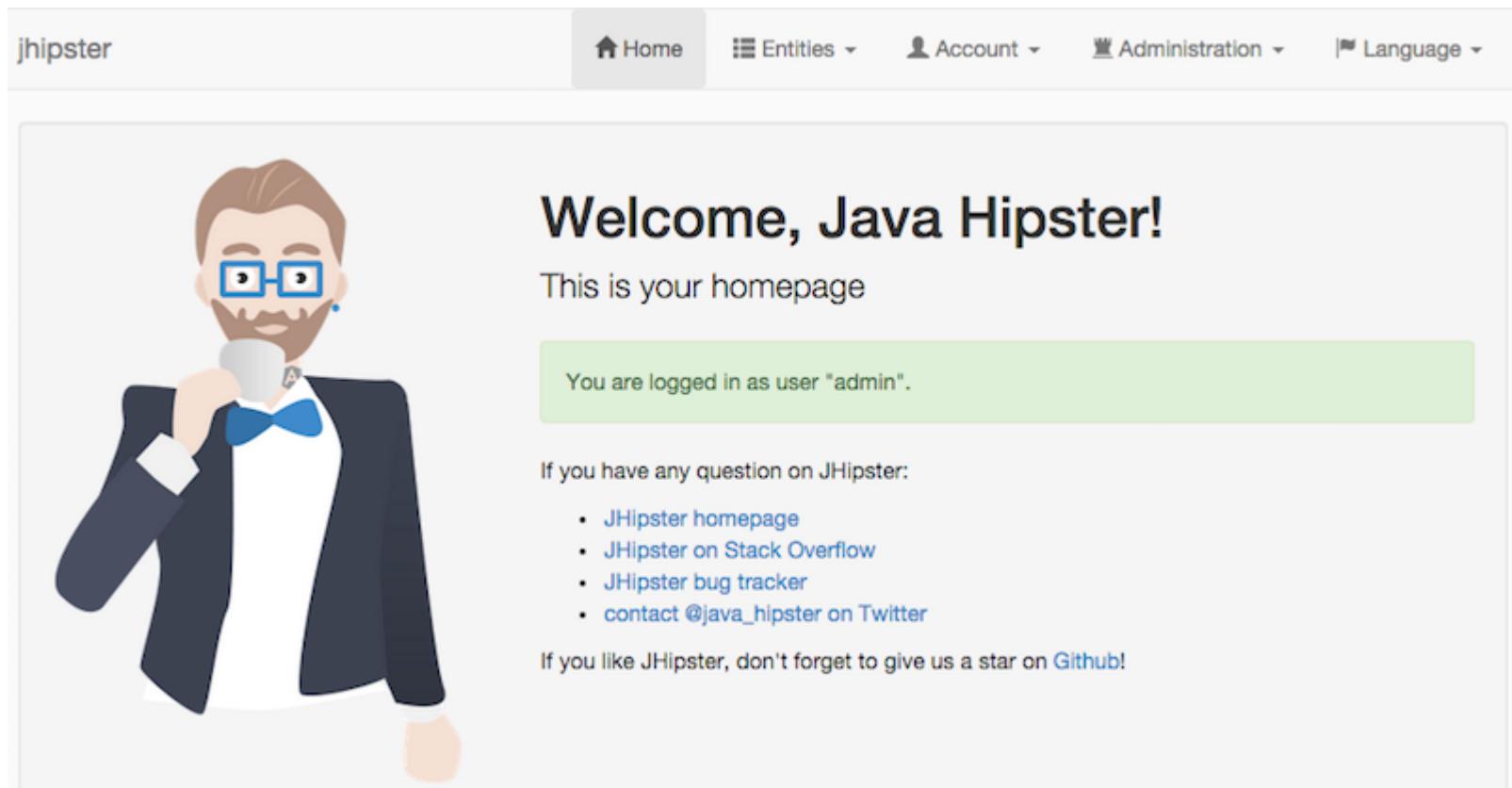
Which call JPA..  
→ Hibernate...  
→ JDBC

Call springboot-data  
JpaRepository

You code  
calls a generated  
Proxy..

# AngularJS + Springboot

In Jhipster ... you have 100% springboot on server-side  
... with Code Generator  
And also Hipe code on client-side : Html / Css + AngularJS + ..

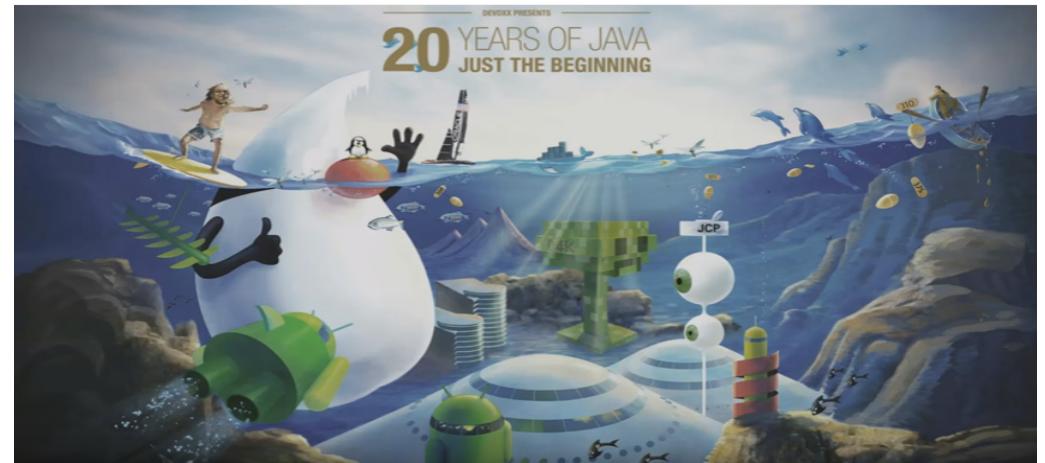


The screenshot shows the JHipster application's homepage. At the top, there is a navigation bar with the brand name "jhipster" on the left and links for "Home", "Entities", "Account", "Administration", and "Language" on the right. The main content area features a large, friendly cartoon character of a man with a beard and glasses, wearing a suit and bow tie, holding a microphone. To the right of the character, the text "Welcome, Java Hipster!" is displayed in a large, bold font. Below this, a message says "This is your homepage". A green callout box contains the text "You are logged in as user 'admin'.". Further down, there is a section titled "If you have any question on JHipster:" followed by a bulleted list of links: "JHipster homepage", "JHipster on Stack Overflow", "JHipster bug tracker", and "contact @java\_hipster on Twitter". At the bottom, a final message encourages users to "If you like JHipster, don't forget to give us a star on [Github](#)!".

# Java is Hipe

## Make Jar nor WAR – NO PHP NO NodeJS on server

20 years of Java  
Just The Beginning



Its HIPE ...because of springboot  
& open-source & java community

# Conclusion

# Conclusion

Only a very short introduction to Databases

This document:

<http://arnaud-nauwynck.github.io/docs/Intro-Db-Jdbc-JPA-SpringData.pdf>