arnaud.nauwynck@gmail.com

Extracts from
Intro to Db - Jdbc - JPA – SpringData

This document:
http://arnaud-nauwynck.github.io/docs/Intro-CodeExtract-DB-Jdbc-JPA-SpringData.pdf

SOURCE Document :

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

~ 150 pages
: lesson to read

THIS Document

~ 80 pages
: slides to present

(4) < Spring Data

Spring JPA
QueryDsl

< JPA API - JPQL
(3) < Hibernate/EclipseLink

JPA
Java Persistence API
HIBERNATE

Spring JDBC

< JDBC API
(2) < Driver Impl.

JDBC
Java Database Connectivity

SQL Client Driver
(1) < DataBase
< B-Tree File

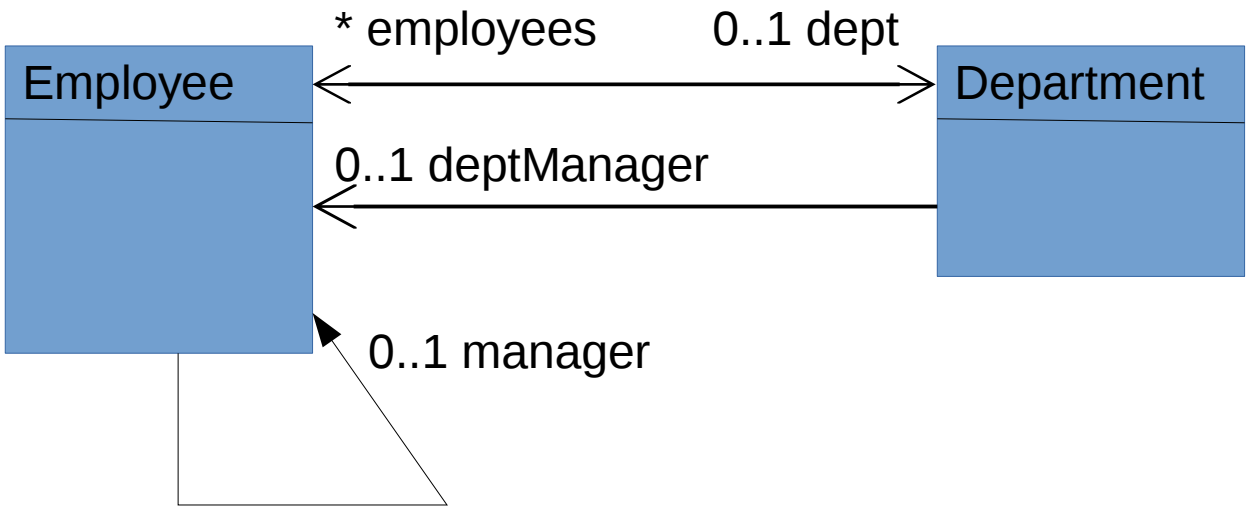
Oracle Sample DataBase

https://github.com/oracle/db-sample-schemas/

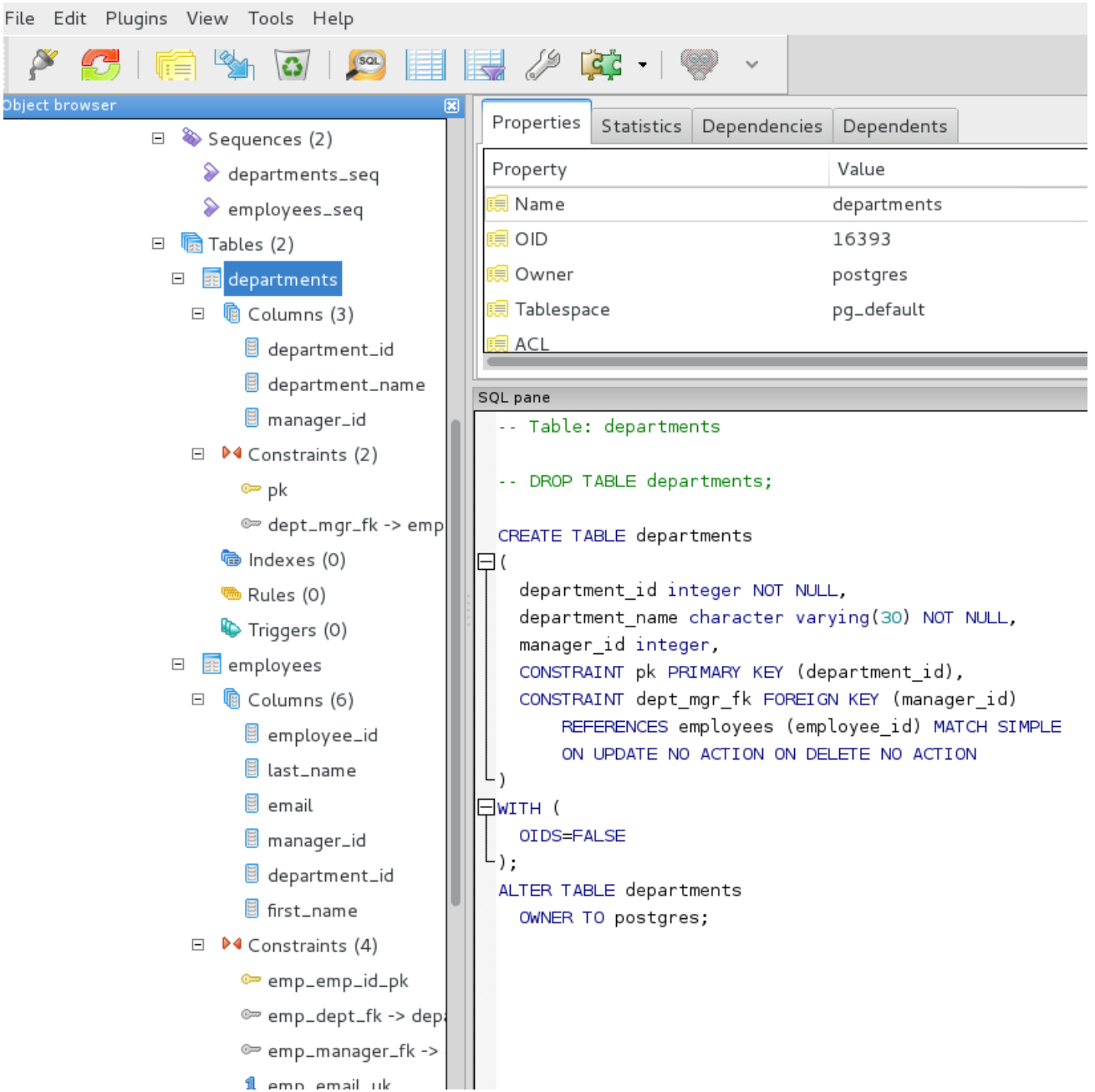
human_resources/hr_cre.sql

```
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



Example using PGAdmin3



Detailed CRUD

CRUD =

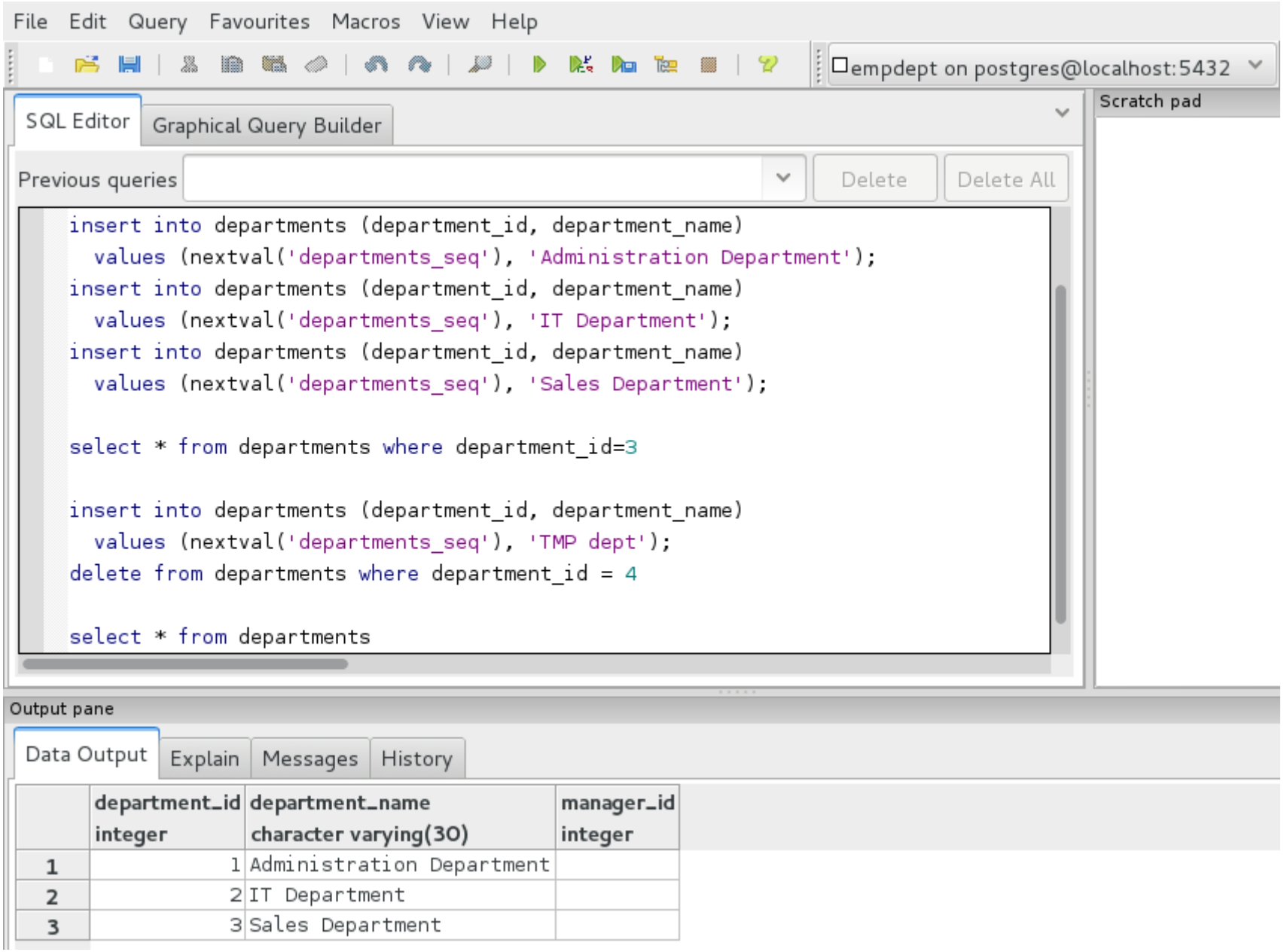
Create **INSERT into <**Table..> (col1, col2, …)
VALUES (?0, ?1, 123, ..)

Read **SELECT** * from <Table..> where <expr..>

Update **UPDATE** col1=value1, col2=…
from <Table..> where <expr..>

Delete **DELETE from** <Table..>
where <expr..>

Emp-Dept CRUD Sample



SQL Exercises

http://www.w3resource.com/sql-exercises/sql-subqueries-exercises.php

SQL [34 exercises with solution]

1. Write a query to display the name (first name and last name) for those employees who gets more salary than the employee whose ID is 163. Go to the editor

Sample table : employees

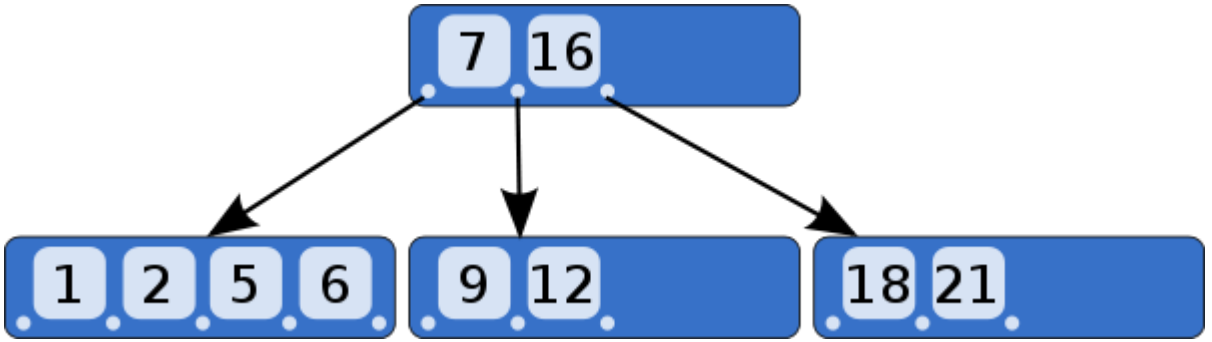
113	Luis	Popp	LPOPP	515.124.4567	2007-12-07	FI_ACCOUNT	6900.00
114	Den	Raphaely	DRAPHEAL	515.127.4561	2002-12-07	PU_MAN	11000.00
115	Alexander	Khoo	AKHOO	515.127.4562	2003-05-18	PU_CLERK	3100.00
116	Shelli	Baida	SBAIDA	515.127.4563	2005-12-24	PU_CLERK	2900.00
117	Sigal	Tobias	STOBIAS	515.127.4564	2005-07-24	PU_CLERK	2800.00
118	Guy	Himuro	GHIMURO	515.127.4565	2006-11-15	PU_CLERK	2600.00
119	Karen	Colmenares	KCOLMENA	515.127.4566	2007-08-10	PU_CLERK	2500.00
120	Matthew	Weiss	MWEISS	650.123.1234	2004-07-18	ST_MAN	8000.00
121	Adam	Fripp	AFRIPP	650.123.2234	2005-04-10	ST_MAN	8200.00

Click me to see the solution

2. Write a query to display the name (first name and last name), salary, department id, job id for those employees who works in the same designation as the employee works whose id is 169. Go to the editor

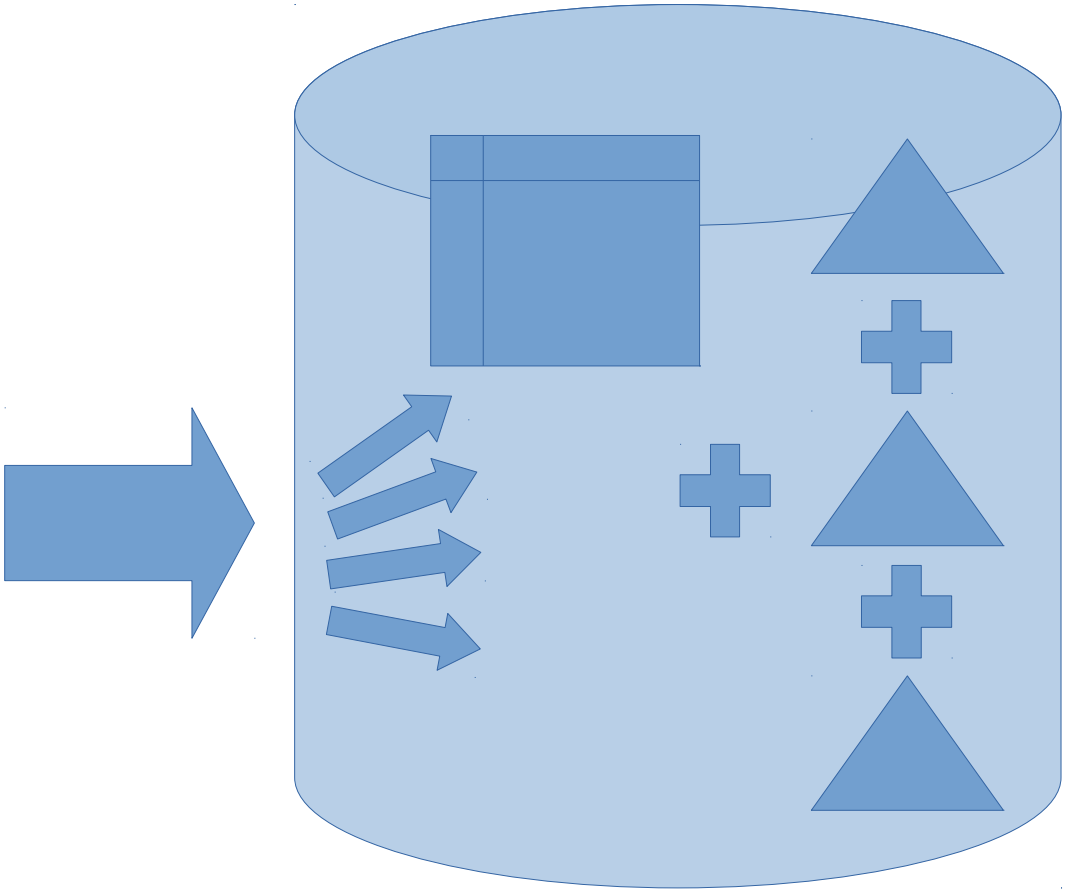
Sample table : employees

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



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

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

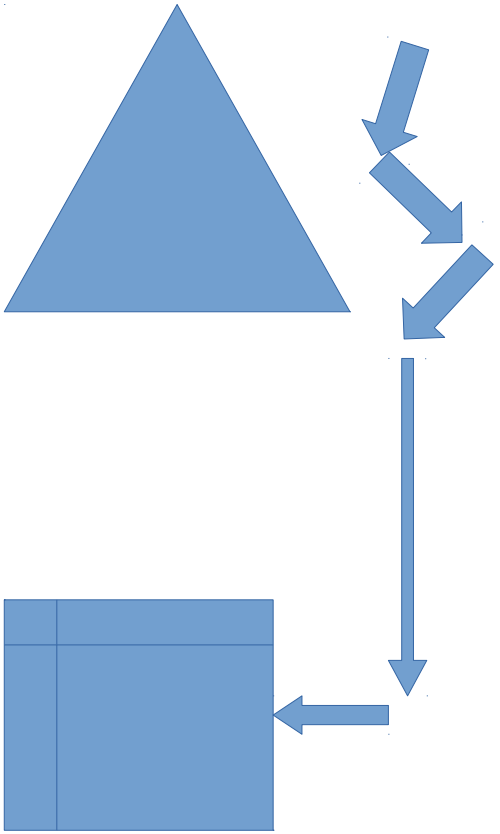


PK INDEX : col (ID)

INDEX2 : (Col2,Col3,ID)

INDEX3 : (Col3,Col2,Col6)

Select .. where ID = ?



Execution Plan

Step 1: Index Lookup (Unique) by ID
Log(N) logical reads

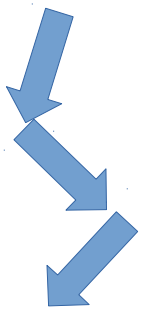
=> get “rowId” (=address)

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)~~
- INDEX (Col2,Col5,Col6)
- INDEX (Col5,Col2,Col7)
- INDEX (~~Col1,Col2,Col5)~~

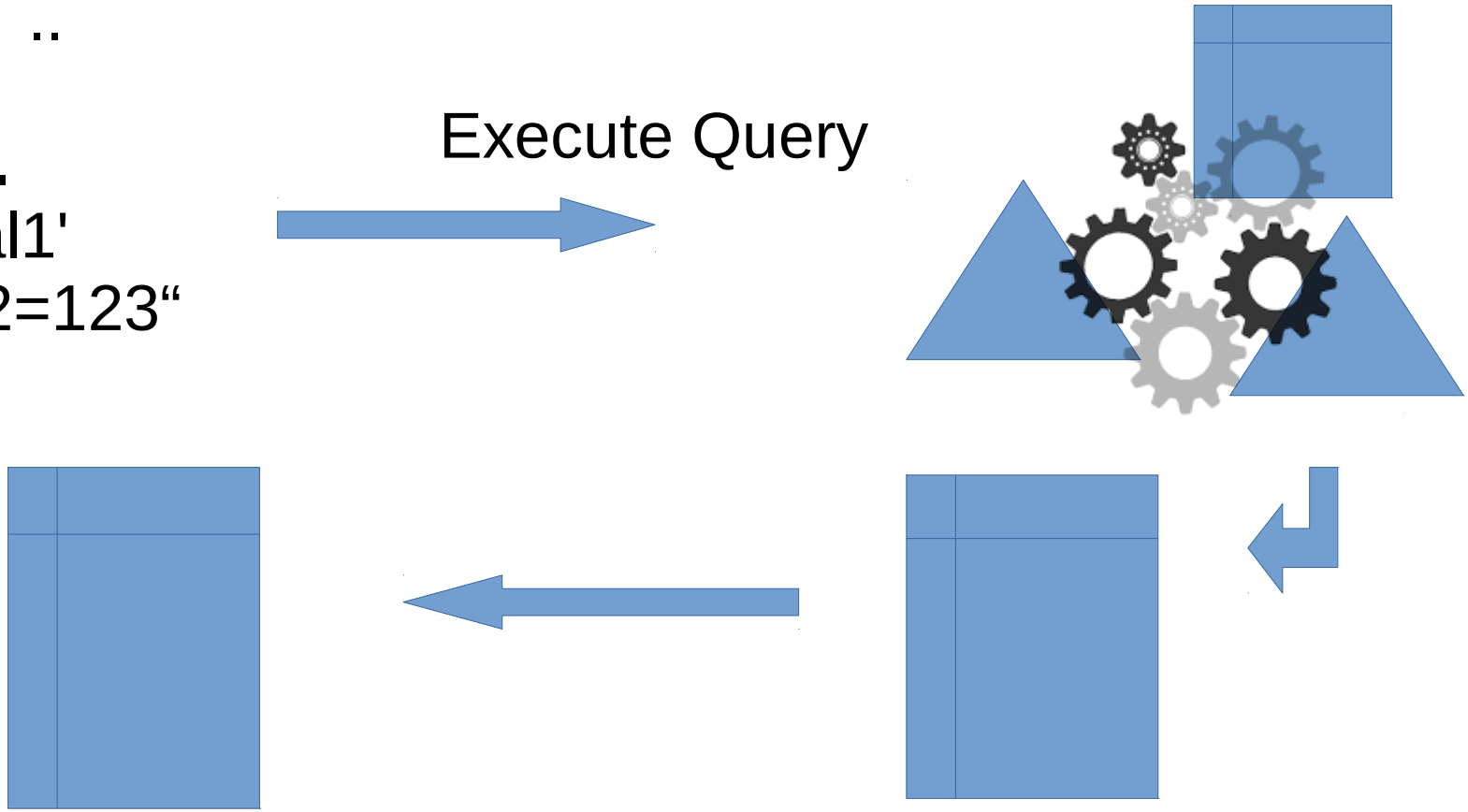
Applicable =
Allow Lexicographical
Top->Down Search



Query Execution Engine

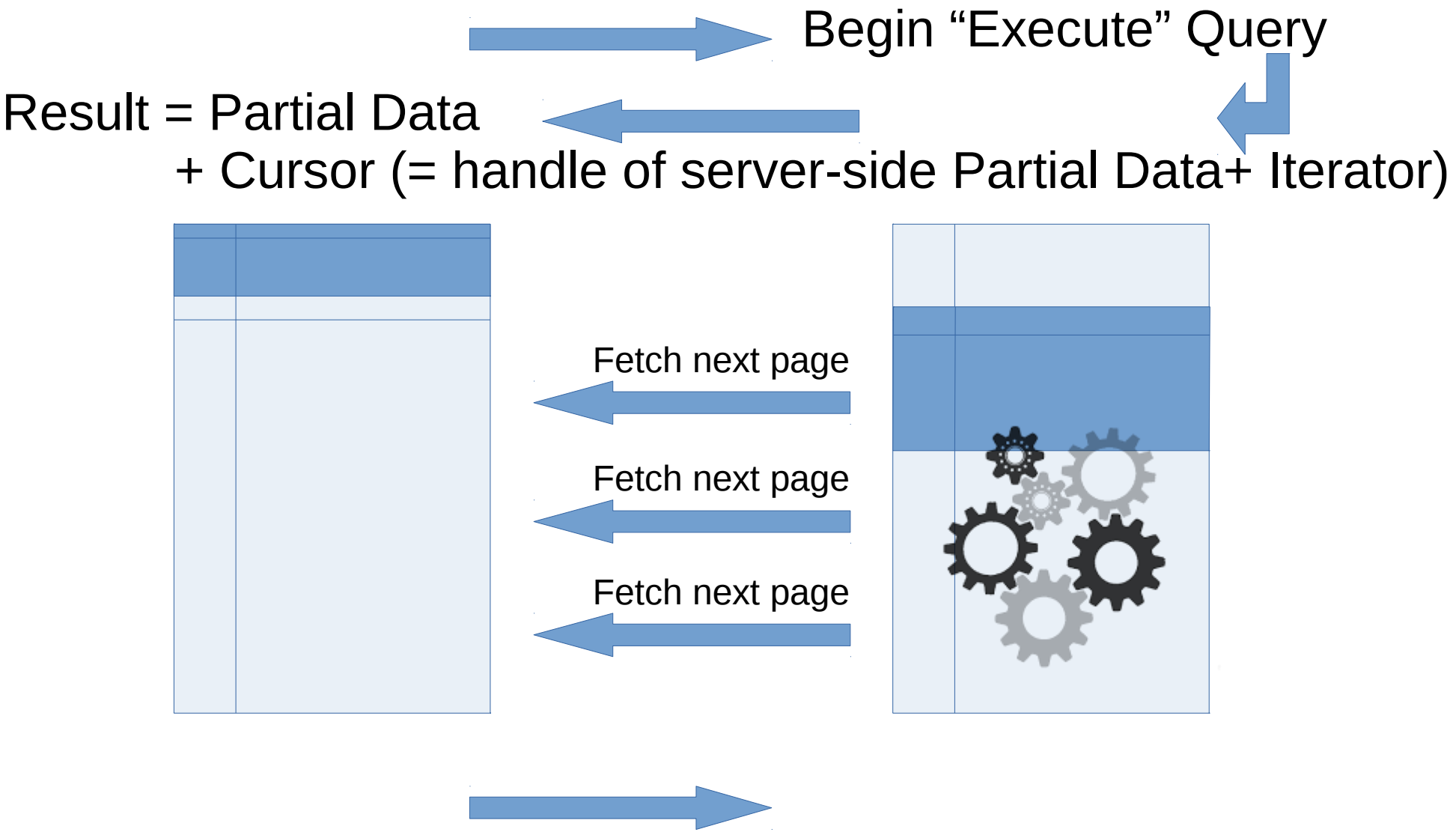
“Select ..
from ..
where ..
col1='val1'
and col2=123“

Execute Query



Result : Tabular Data Format = “ResultSet”

Huge Result .. Server-Side “Cursor”



SoftParse – HardParse … PrepareQuery + BindVariable

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

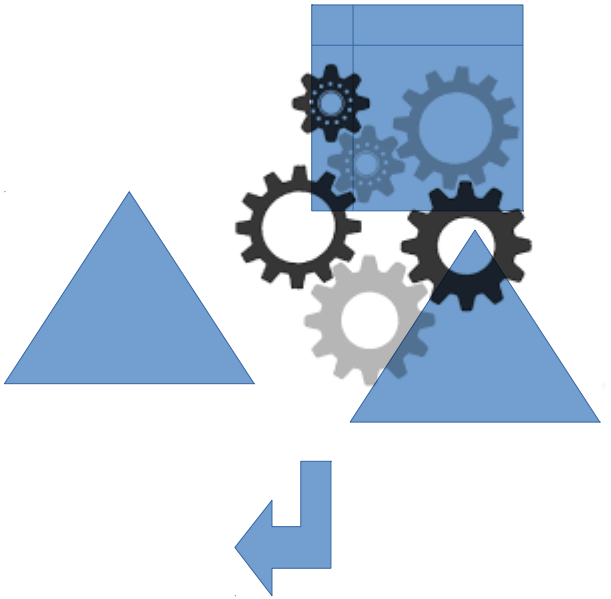
BindVariable:
set(0, “val1”)
set(1, 1234)

First Seen ?
HARD Parse

Compute
Execution Plan

Put in Cache

Already Seen ?
SOFT Parse
= create Cursor



Explain Execution Plan

empdept on postgres@lo

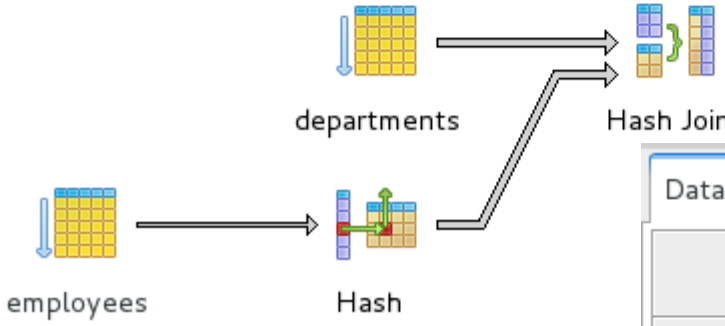
SQL Editor | Graphical Query Builder

Previous queries: select d.*, e.email from departments d, employees e wh | Delete | Delete All

```
select d.*, e.email
 from departments d, employees e
 where d.manager_id = e.employee_id;
```

Output pane

Data Output | Explain | Messages | History

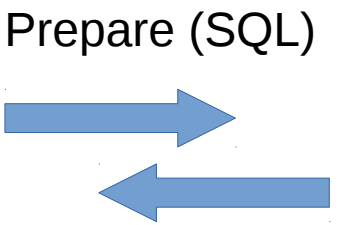


Data Output | Explain | Messages | History

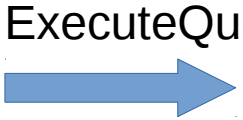
	QUERY PLAN text
1	Hash Join (cost=10.68..31.05 rows=102 width=606)
2	Hash Cond: (d.manager_id = e.employee_id)
3	-> Seq Scan on departments d (cost=0.00..16.80 rows=680 width=90)
4	-> Hash (cost=10.30..10.30 rows=30 width=520)
5	-> Seq Scan on employees e (cost=0.00..10.30 rows=30 width=520)

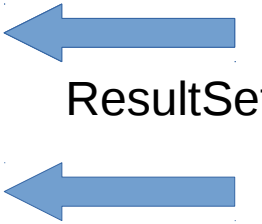
Query / Prepared Query Execution

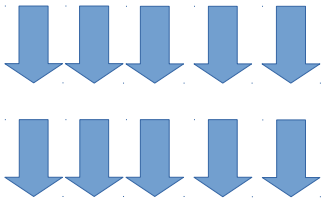
Select ..
from ..
where ..
col1=**?0**
and col2=**?1**



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







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

Spring JPA
QueryDsl

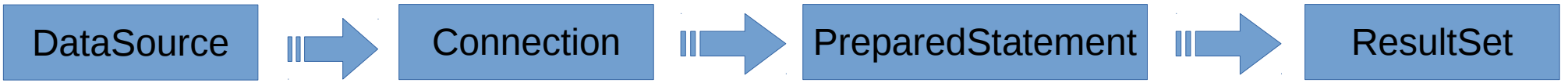
JPA
Java Persistence API
HIBERNATE

Spring JDBC

JDBC
Java Database Connectivity

SQL

import java.sql.*;



Sample Jdbc (Test with Rollback)

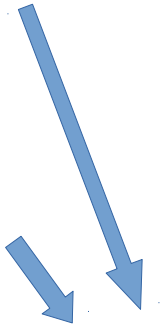
```
@Test
public void testExecuteRollback_before() throws SQLException {
    Connection cx = dataSource.getConnection();
    cx.setAutoCommit(false);
    try {
        // ** execute **
        try(PreparedStatement stmt = cx.prepareStatement("select 'Hello'")) {
            ResultSet rs = stmt.executeQuery();
            String res = rs.getString(1);
            Assert.assertEquals("Hello", res);
        }

        cx.rollback();
    } catch(Exception ex) {
        cx.rollback();
        throw new RuntimeException("Failed ..rolleback", ex);
    } finally {
        cx.close();
    }
}
```

Refactored (1/2)

```
Connection cx = dataSource.getConnection();
cx.setAutoCommit(false);
try {
    String sql = "select 'Hello'";
    try(PreparedStatement stmt =
            cx.prepareStatement(sql)) {
        ResultSet rs = stmt.executeQuery();
        String res = rs.getString(1);
        Assert.assertEquals("Hello", res);
    }
```

```
    cx.rollback();
} catch(Exception ex) {
    cx.rollback();
    throw new RuntimeException("Failed ..rolleback", ex);
} finally {
    cx.close();
}
```

```
executeRollbackVoid(cx -> {
    String sql = "select 'Hello'";
    try(PreparedStatement stmt =
            cx.prepareStatement(sql)) {
        ResultSet rs = stmt.executeQuery();
        String res = rs.getString(1);
        Assert.assertEquals("Hello", res);
    }
});
```



```
@FunctionalInterface
public static interface CxVoidCallback {
    public void executeWithConnection(Connection cx) throws SQLException;
}

private void executeRollbackVoid(CxVoidCallback callback) throws SQLException {
    Connection cx = dataSource.getConnection();
    cx.setAutoCommit(false);
```

Refactored (2/2)
“Template” + Callback

Code to run with Cx + rollback

```
@Test
public void testExecuteRollback() throws SQLException {
    executeRollbackVoid(cx -> {
        try(PreparedStatement stmt = cx.prepareStatement("select 'Hello'")) {
            ResultSet rs = stmt.executeQuery();
            String res = rs.getString(1);
            Assert.assertEquals("Hello", res);
        }
    });
}
```

Common Template Framework

.. similar to Spring Template

```
@FunctionalInterface
public static interface CxVoidCallback {
    public void executeWithConnection(Connection cx) throws SQLException;
}

private void executeRollbackVoid(CxVoidCallback callback) throws SQLException {
    Connection cx = dataSource.getConnection();
    cx.setAutoCommit(false);
    try {
        // ** execute **
        callback.executeWithConnection(cx);

        cx.rollback();
    } catch(Exception ex) {
        cx.rollback();
        throw new RuntimeException("Failed ..rolleback", ex);
    } finally {
        cx.close(); // close cx after commit/rollback
    }
}
```

Data Transfer Object for CRUD

```java
/**
 * DTO (Data Transfer Object) for Employee
 * Serializable data representation of a Row
 * agnostic of any framework
 */
@Data // lombok getter, setter
public static class EmployeeDTO implements Serializable {
    private static final long serialVersionUID = 1L;

    private int id; // Primary Key
    private int version; // field for Optimistic lock

    private String firstName;
    private String lastName;
    private String email;
    private String address;
    private Date hireDate;
    private double salary;

    // relationships fields ... lazy loaded!, replace by ForeignKey id
    // private Department department;
    private int departmentId;
    // private Employee manager;
    private int managerId;
}
```

CRUD 1 / 4 : SELECT

```java
@Test
public void testFindEmployeeById() throws SQLException {
    executeRollbackVoid(cx -> {
        final int empId = 2;
        String sql = "SELECT e.employee_id," + //
                " e.first_name, e.last_name,"+ //
                " e.email, e.address," + //
                " e.hire_date, e.salary," + //
                " e.department_id, e.manager_id " + //
                "FROM employees e " + //
                "WHERE e.employee_id = ?";
        try (PreparedStatement stmt = cx.prepareStatement(sql)) {
            stmt.setInt(1, empId);
            EmployeeDTO emp = null;
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    emp = new EmployeeDTO();
                    emp.setId(rs.getInt("employee_id"));
                    emp.setFirstName(rs.getString("first_name"));
                    emp.setLastName(rs.getString("last_name"));
                    emp.setEmail(rs.getString("email"));
                    emp.setAddress(rs.getString("address"));
                    emp.setHireDate(rs.getDate("hire_date"));
                    emp.setSalary(rs.getDouble("salary"));
                    emp.setDepartmentId(rs.getInt("department_id"));
                    emp.setManagerId(rs.getInt("manager_id"));
                }
            }
            Assert.assertNotNull(emp);
            Assert.assertEquals(empId, emp.getId());
        }
    });
}
```

Jdbc is As Verbose as Easy ...

Is it DRY ?

D don't
R repeat
Y yourself

CRUD 2 / 4 : INSERT

```
final EmployeeDTO emp = new EmployeeDTO();
// emp.setId()?? don't .. use database sequence
emp.setFirstName("John");
emp.setLastName("Smith");
emp.setEmail("jown.smith@company.com");
emp.setHireDate(new Date());
emp.setSalary(60000);
emp.setDepartmentId(3);
emp.setManagerId(4);

executeRollbackVoid(cx -> {
    String sql = "INSERT INTO employees (employee_id," + //
            " first_name, last_name,"+ //
            " email, address," + //
            " hire_date, salary," + //
            " department_id, manager_id) " + //
            "VALUES (nextval('employees_seq'),?,?,?,?,?,?,?,?)";
    try (PreparedStatement stmt = cx.prepareStatement(sql)) {
        int i = 1;
        stmt.setString(i++, emp.getFirstName());
        stmt.setString(i++, emp.getLastName());
        stmt.setString(i++, emp.getEmail());
        stmt.setString(i++, emp.getAddress());
        stmt.setDate(i++, new java.sql.Date(emp.getHireDate().getTime()));
        stmt.setDouble(i++, emp.getSalary());
        stmt.setInt(i++, emp.getDepartmentId());
        stmt.setInt(i++, emp.getManagerId());

        int updateCount = stmt.executeUpdate();
        Assert.assertEquals(1, updateCount);
    }
});//even if rollbacked... sequence is incremented each time!
```

CRUD 3 / 4 : UPDATE (All columns)

```
final int empId = 4;
final EmployeeDTO emp = new EmployeeDTO();
emp.setId(empId);
emp.setFirstName("John");
emp.setLastName("Smith");
emp.setEmail("jown.smith@company.com");
emp.setHireDate(new Date());
emp.setSalary(60000);
emp.setDepartmentId(3);
emp.setManagerId(4);

executeRollbackVoid(cx -> {
    String sql = "UPDATE employees " + //
            "SET first_name=?, last_name=?,"+ //
            " email=?, address=?," + //
            " hire_date=?, salary=?," + //
            " department_id=?, manager_id=? " + //
            "WHERE employee_id=?";
    try (PreparedStatement stmt = cx.prepareStatement(sql)) {
        int i = 1;
        stmt.setString(i++, emp.getFirstName());
        stmt.setString(i++, emp.getLastName());
        stmt.setString(i++, emp.getEmail());
        stmt.setString(i++, emp.getAddress());
        stmt.setDate(i++, new java.sql.Date(emp.getHireDate().getTime()));
        stmt.setDouble(i++, emp.getSalary());
        stmt.setInt(i++, emp.getDepartmentId());
        stmt.setInt(i++, emp.getManagerId());

        stmt.setInt(i++, emp.getId());

        int updateCount = stmt.executeUpdate();
        Assert.assertEquals(1, updateCount);
    }
});
```

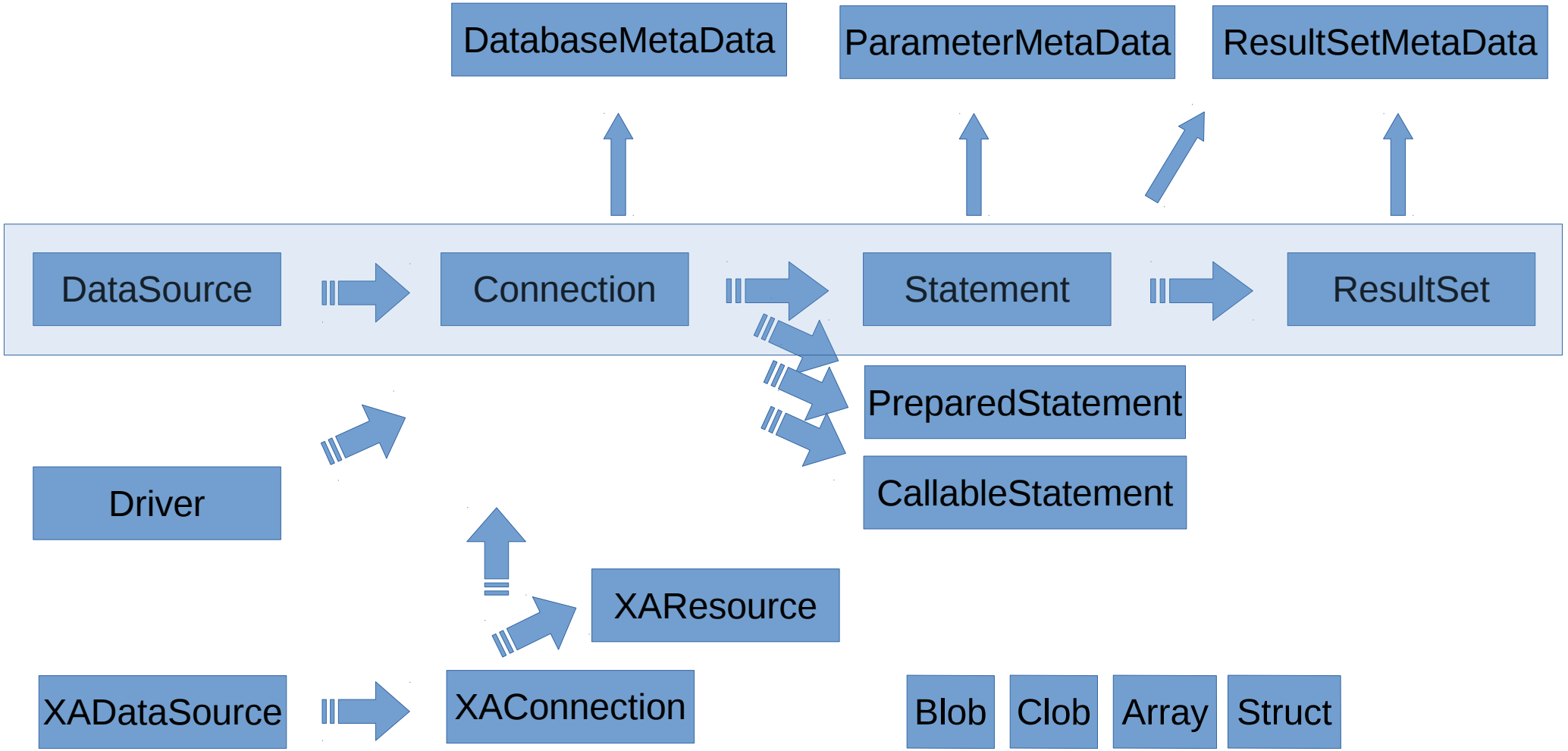
UPDATE with Versioning

```
final int empId = 4;
final int expectedCurrentVersion = 503;
final EmployeeDTO emp = createEmployeeJohnSmith(empId);

executeRollbackVoid(cx -> {
    String sql = "UPDATE employees " + //
            "SET first_name=?, last_name=?,"+ //
            " email=?, address=?," + //
            " hire_date=?, salary=?," + //
            " department_id=?, manager_id=?, " + //
            " version=? "+ //
            "WHERE employee_id=? AND version=?";
    try (PreparedStatement stmt = cx.prepareStatement(sql)) {
        int i = 1;
        stmt.setString(i++, emp.getFirstName());
        stmt.setString(i++, emp.getLastName());
        stmt.setString(i++, emp.getEmail());
        stmt.setString(i++, emp.getAddress());
        stmt.setDate(i++, new java.sql.Date(emp.getHireDate().getTime()));
        stmt.setDouble(i++, emp.getSalary());
        stmt.setInt(i++, emp.getDepartmentId());
        stmt.setInt(i++, emp.getManagerId());

        int incrementedVersion = expectedCurrentVersion + 1;
        stmt.setInt(i++, incrementedVersion);
        stmt.setInt(i++, emp.getId());
        stmt.setInt(i++, expectedCurrentVersion);

        int updateCount = stmt.executeUpdate();
        if (updateCount != 1) {
            throw new OptimisticLockException("attemp to overwriting Employee " + emp.getId() +
                    " using stale version:" + expectedCurrentVersion + " - Please refresh and retry");
        }
        Assert.assertEquals(1, updateCount);
    }
});
```

CRUD 4 / 4 : DELETE

```
@Test
public void testDelete() throws SQLException {
    final int empId = 4;

    executeRollbackVoid(cx -> {
        String sql = "DELETE FROM employees " + //
                "WHERE employee_id=?";
        try (PreparedStatement stmt = cx.prepareStatement(sql)) {
            stmt.setInt(1, empId);

            int updateCount = stmt.executeUpdate();
            if (updateCount != 1) {
                throw new EntityNotFoundException();
            }
            Assert.assertEquals(1, updateCount);
        }
    });
}
```

import java.sql.*; (full)



H2 DataSource HelloWorld

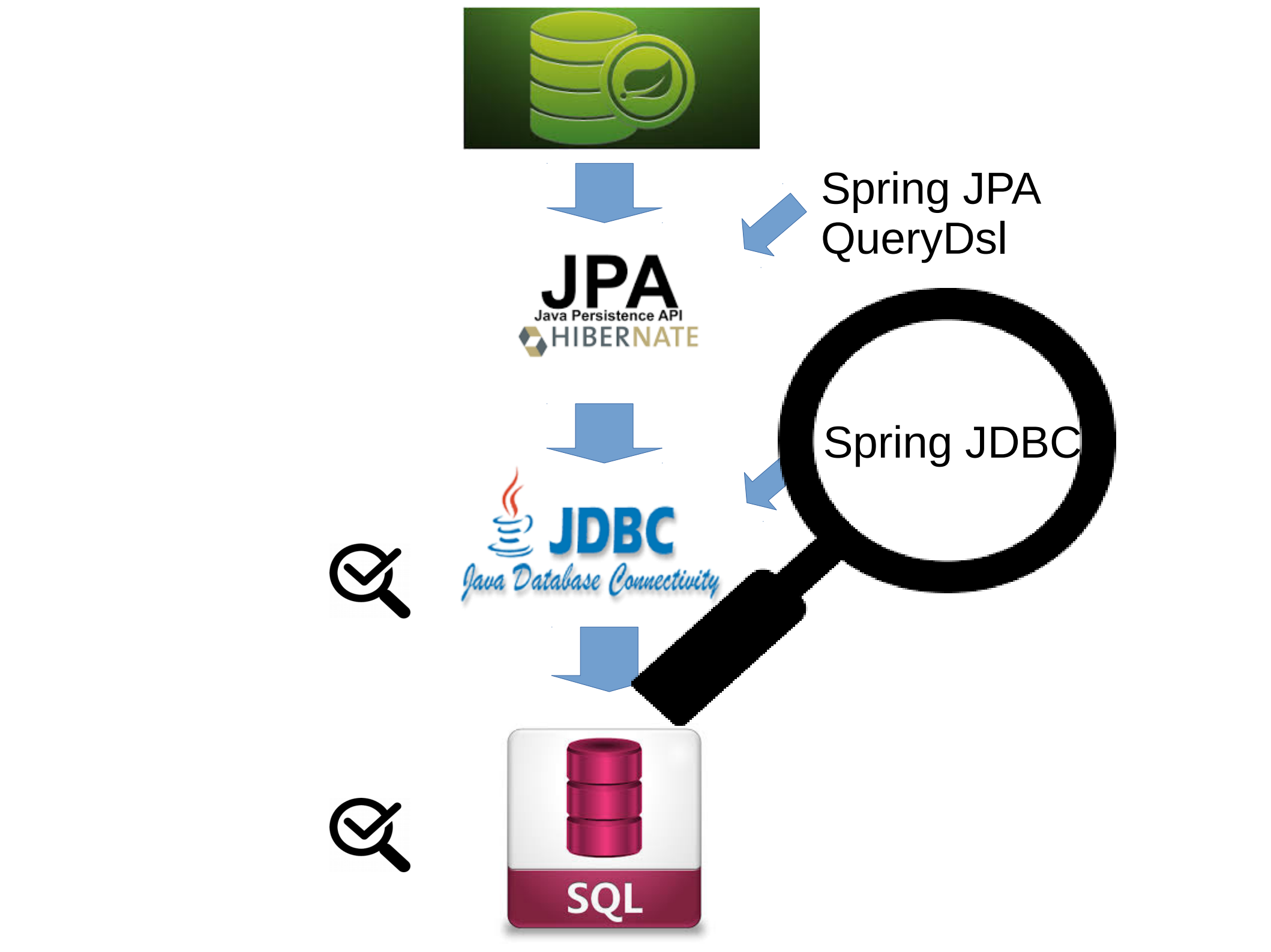
```xml
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

```java
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();
    }


}
```

Postgresql HelloWorld

```xml
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

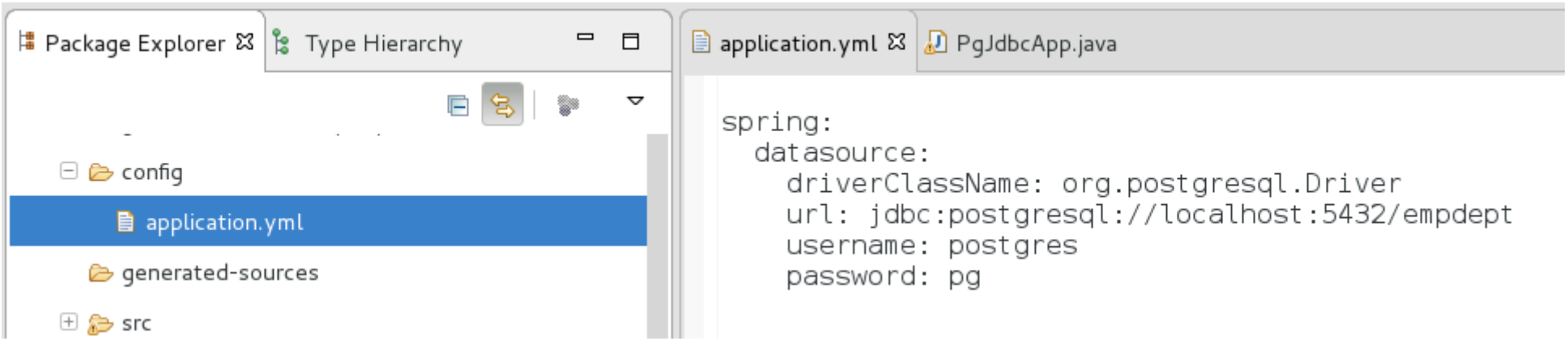
```
public class PgDatabaseTest {

    private static DataSource dataSource = createDS();
    private static DataSource createDS() {
        org.postgresql.ds.PGPoolingDataSource ds = new org.postgresql.ds.PGPoolingDataSource();
        ds.setUrl("jdbc:postgresql://localhost:5432/empdept");
        ds.setUser("postgres");
        ds.setPassword("pg");
        return ds;
    }

    @Test
    public void testDataSource() throws Exception {
        Connection cx = dataSource.getConnection();
        cx.setAutoCommit(false);
        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 JPA
QueryDsl
JPA
Java Persistence API
HIBERNATE
Spring JDBC
JDBC
Java Database Connectivity
SQL

Spring-Jdbc Database App

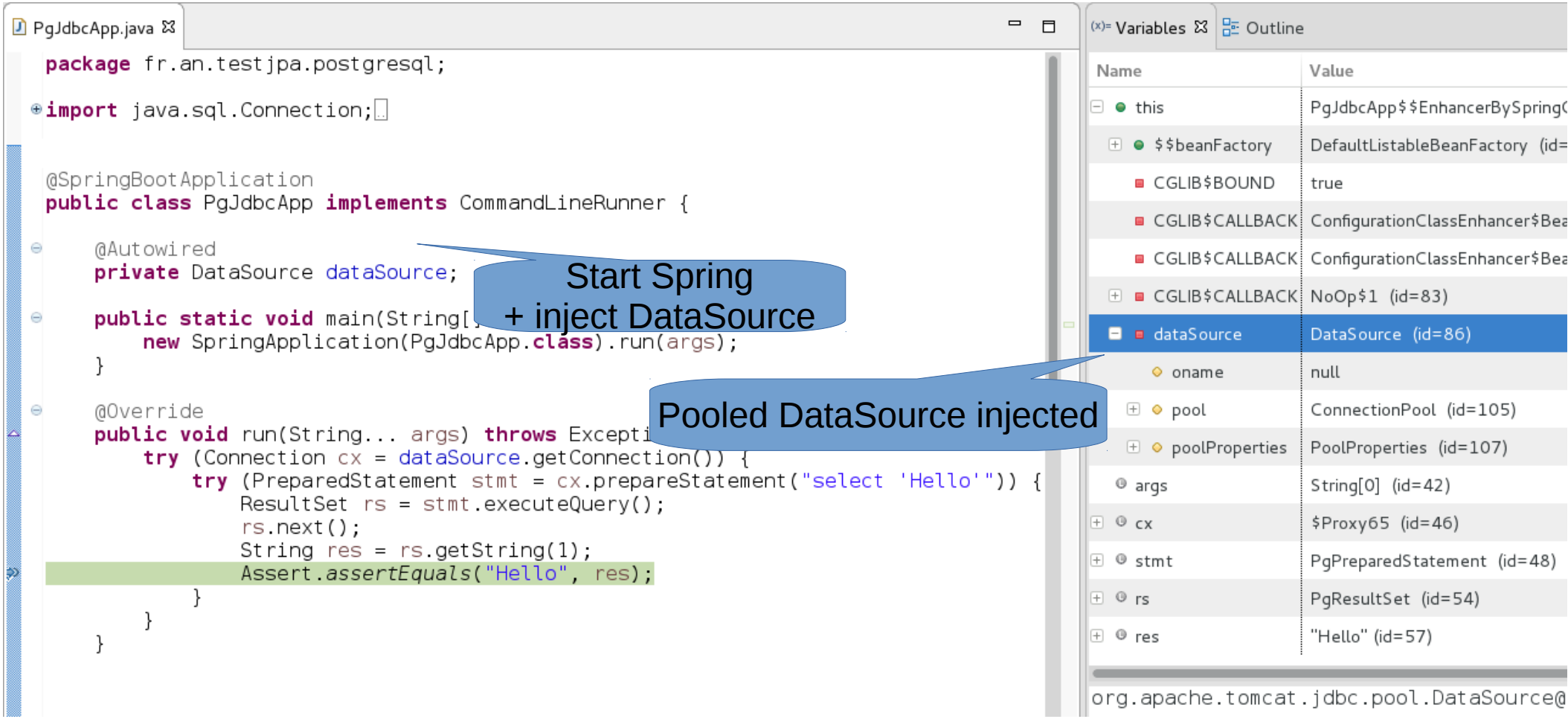
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<!-- 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>
 -->
```

Springboot config/application.yml

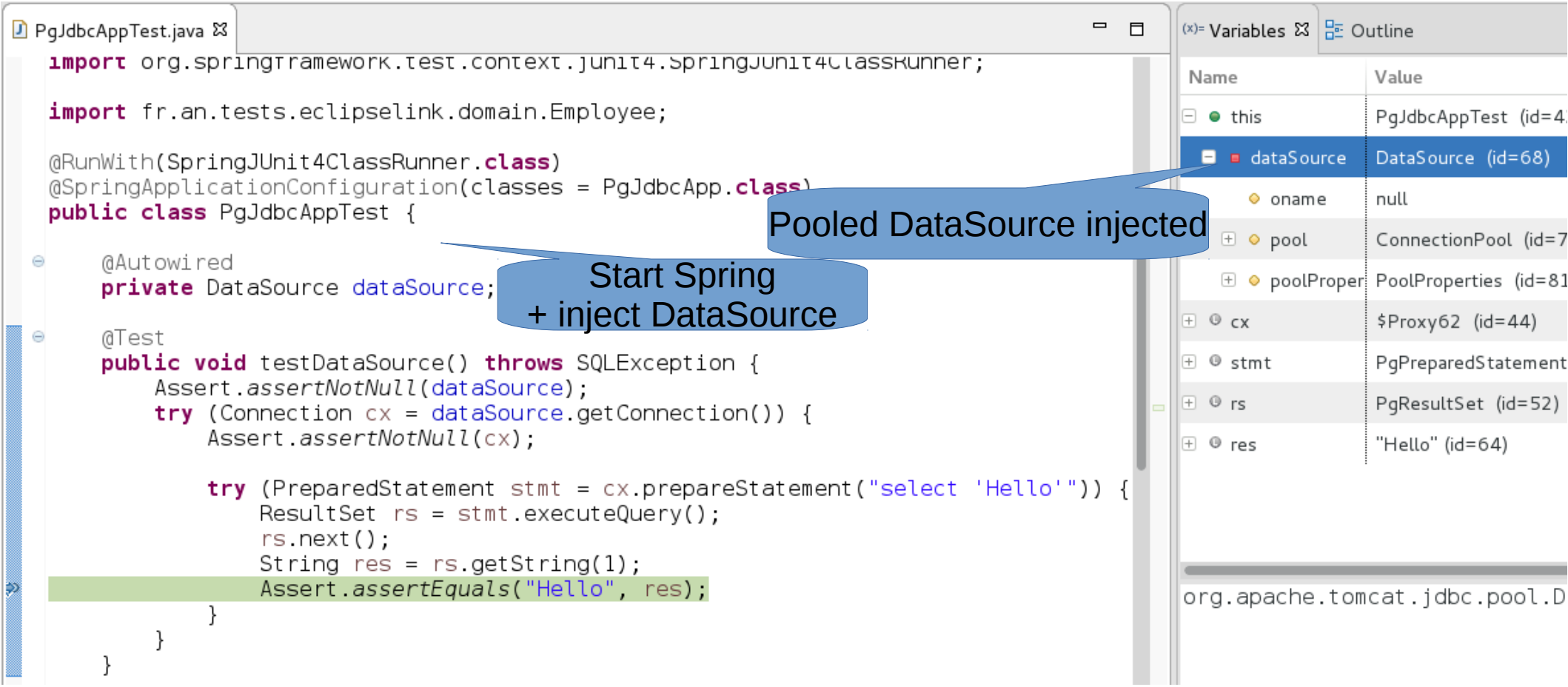


Springboot JDBC… “JUST work”

Springboot Jdbc main()



Springboot JUnit



(Reminder) 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

using Template Spring-Jdbc

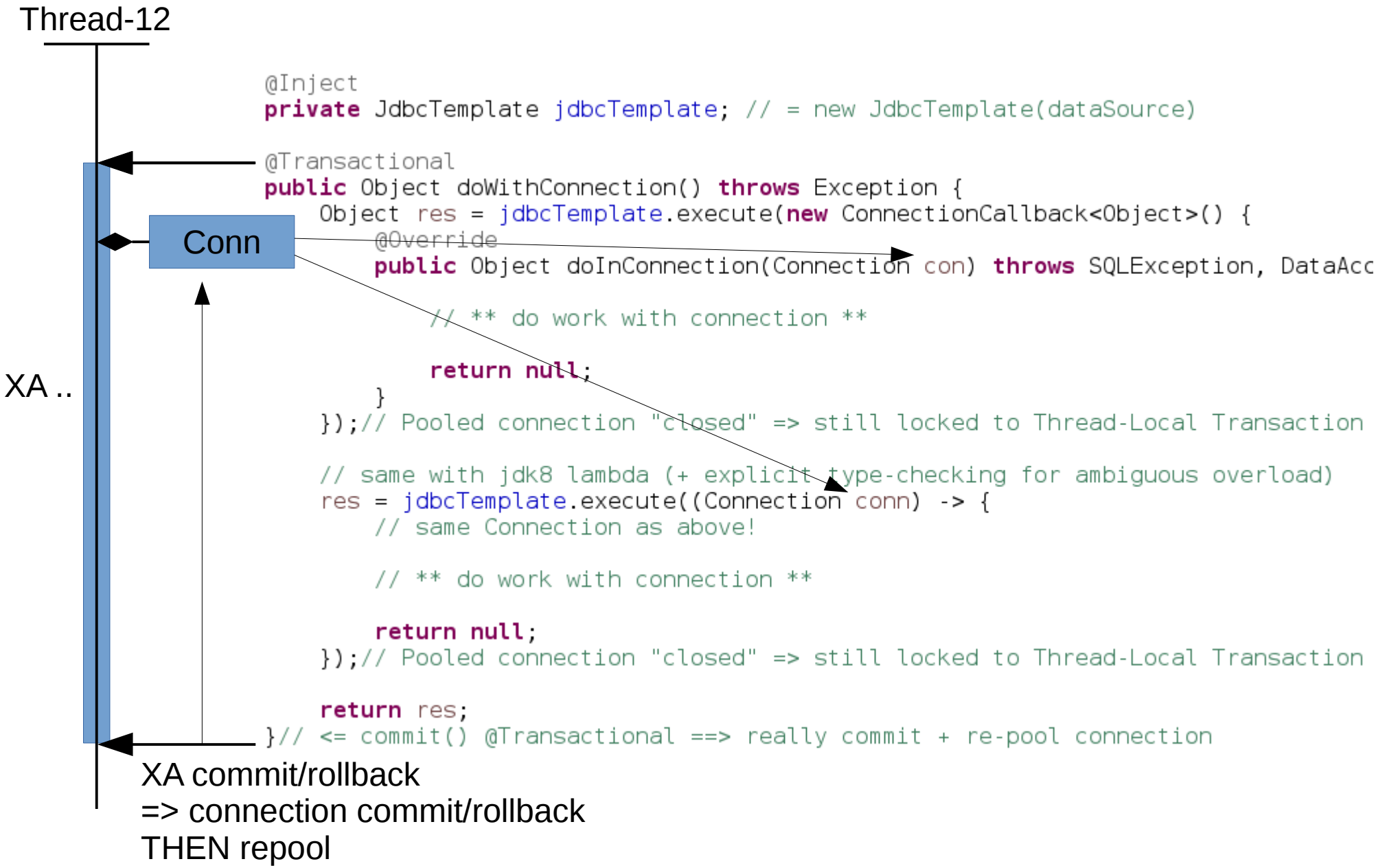
```java
@Inject
private JdbcTemplate jdbcTemplate; // = new JdbcTemplate(dataSource)

// @Transactional // NOT yet...
public Object doWithConnection() throws Exception {
    Object res = jdbcTemplate.execute(new ConnectionCallback<Object>() {
        @Override
        public Object doInConnection(Connection con) throws SQLException, DataA

            // ** 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

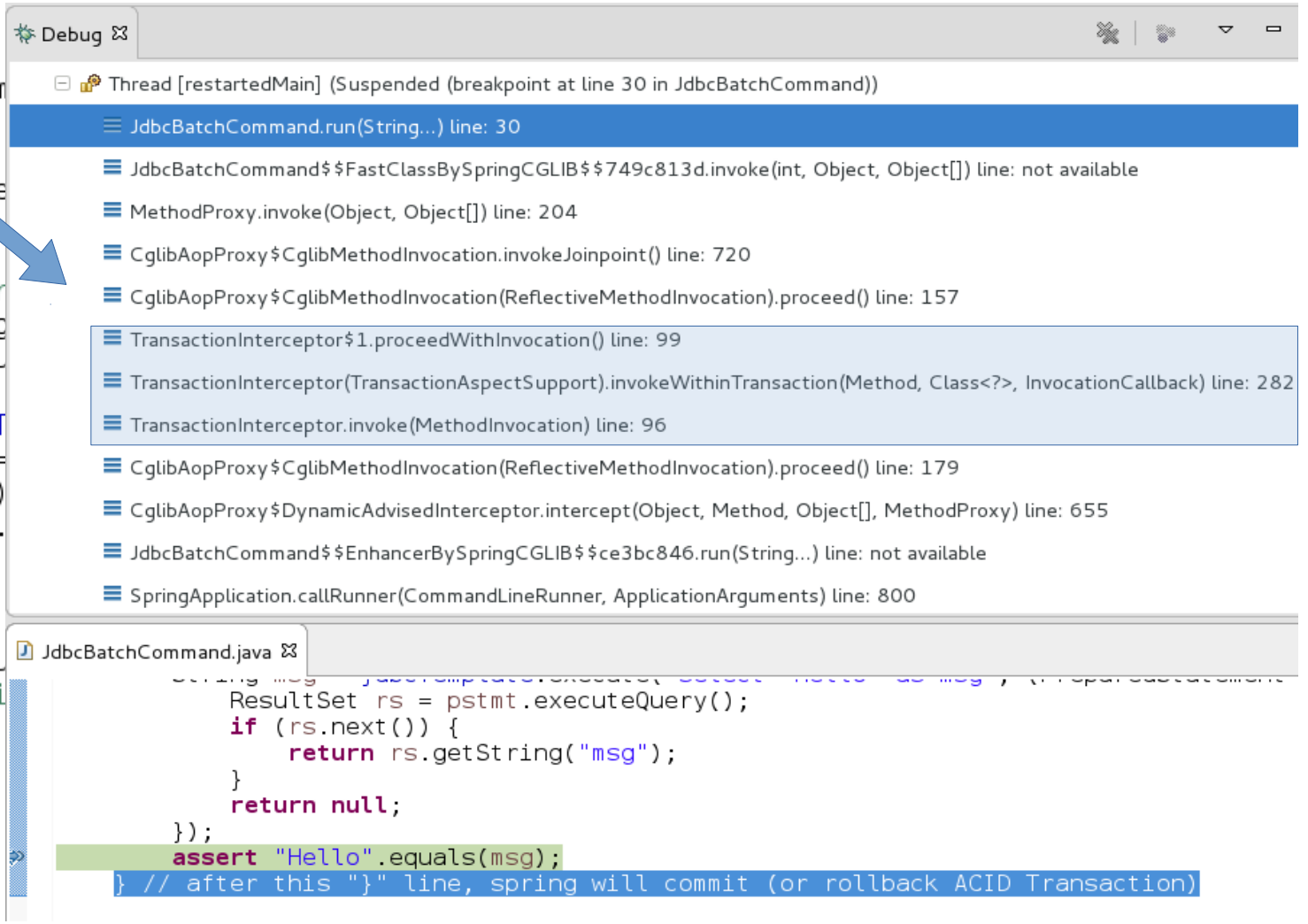
PooledConnection + Transaction = Thread-Locked : reuse

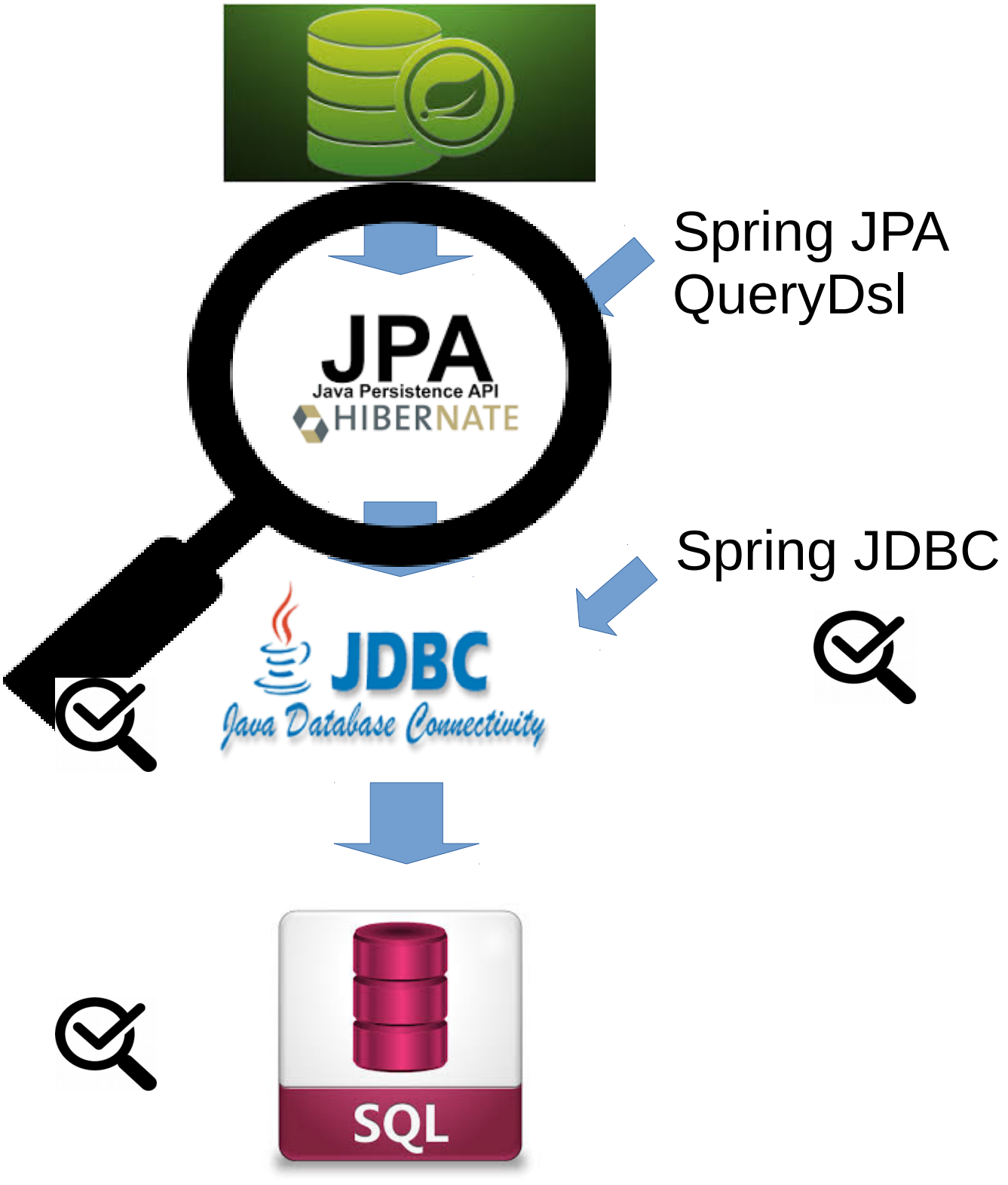



```
@Inject
private JdbcTemplate jdbcTemplate; // = new JdbcTemplate(dataSource)

@Transactional
public Object doWithConnection() throws Exception {
    Object res = jdbcTemplate.execute(new ConnectionCallback<Object>() {
        @Override
        public Object doInConnection(Connection con) throws SQLException, DataAcc

            // ** do work with connection **

            return null;
        }
    });// Pooled connection "closed" => still locked to Thread-Local Transaction

    // same with jdk8 lambda (+ explicit type-checking for ambiguous overload)
    res = jdbcTemplate.execute((Connection conn) -> {
        // same Connection as above!

        // ** do work with connection **

        return null;
    });// Pooled connection "closed" => still locked to Thread-Local Transaction

    return res;
}// <= commit() @Transactional ==> really commit + re-pool connection
```

Springboot built-in supports JTA @Transactional

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

@Component
@Transactional
public class JdbcBatchComm

    @Autowired
    protected JdbcTemp ate

    @Override
    // @Transactional // r
    public void run(String
        jdbcTemplate.execu

        String msg = jdbcT
            ResultSet rs =
            if (rs.next())
                return rs.
            }
            return null;
        });
        assert "Hello".equ
    } // after this "}" li
```




Spring JPA
QueryDsl
JPA
Java Persistence API
HIBERNATE
Spring JDBC
JDBC
Java Database Connectivity
SQL

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

```java
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);
```

What is an “Entity” ?

A class with @Entity

And an @Id

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Simple {

    @Id
    private int id;

    public Simple() {
    }

}
```

Sufficient @Annotations to know?

Base
- @Entity
- @Id
- @GeneratedValue
- @SequenceGenerator
- @Version

Sub-Classes
- @Inheritance
- @DiscriminatorColumn

For relationships
FK (*) → (1) PK
PK (1) ← (*) FK
- @ManyToOne
- @OneToMany

Custom
- @Table
- @Column
- @JoinTable
- @JoinColumn

Why putting @Column & @Table ?

```java
@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;
```

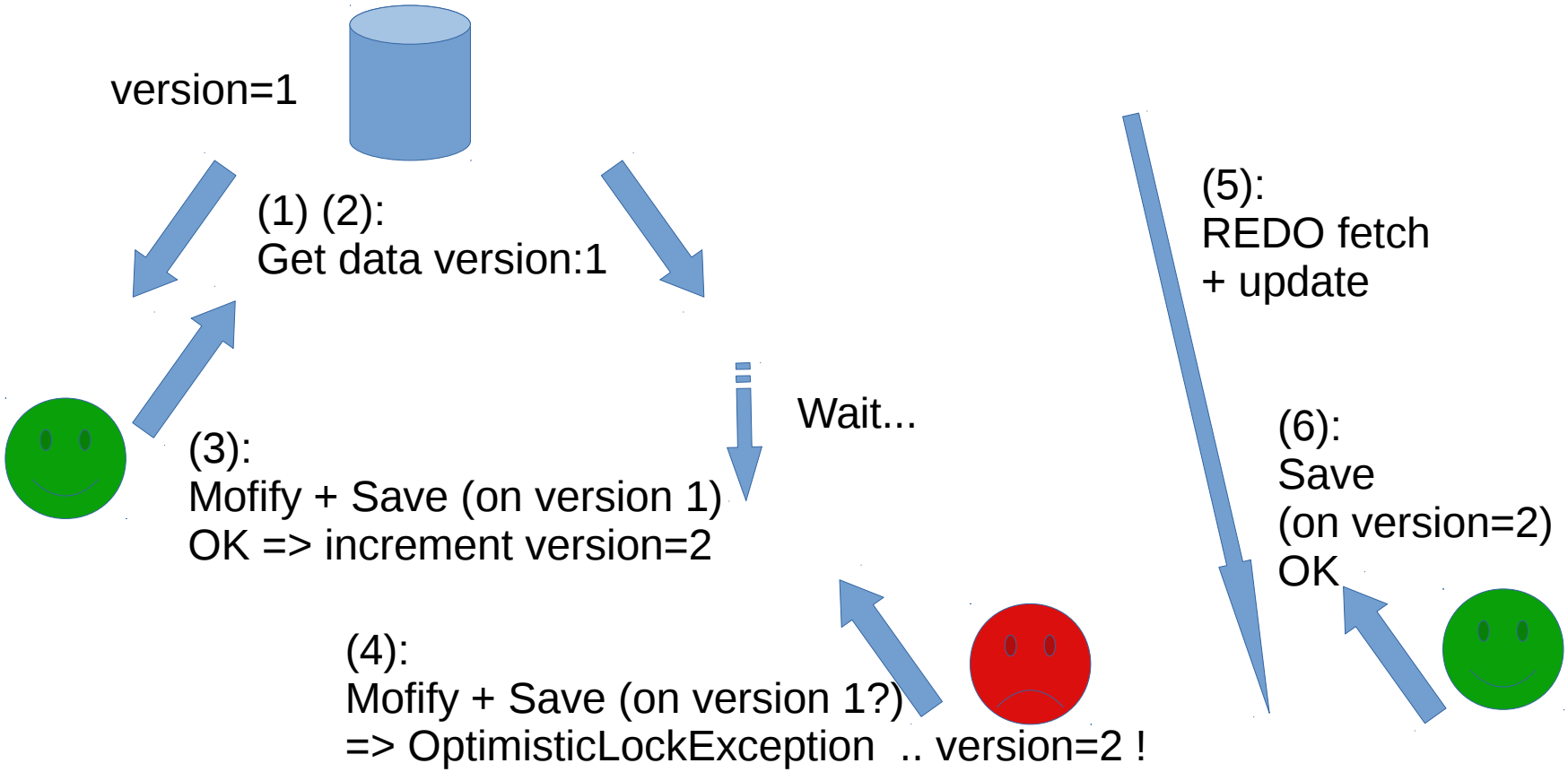
SQL:
CREATE SEQUENCE employees_seq INCREMENT BY 10;

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();`



@Entity Employee-Department

```java
@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)

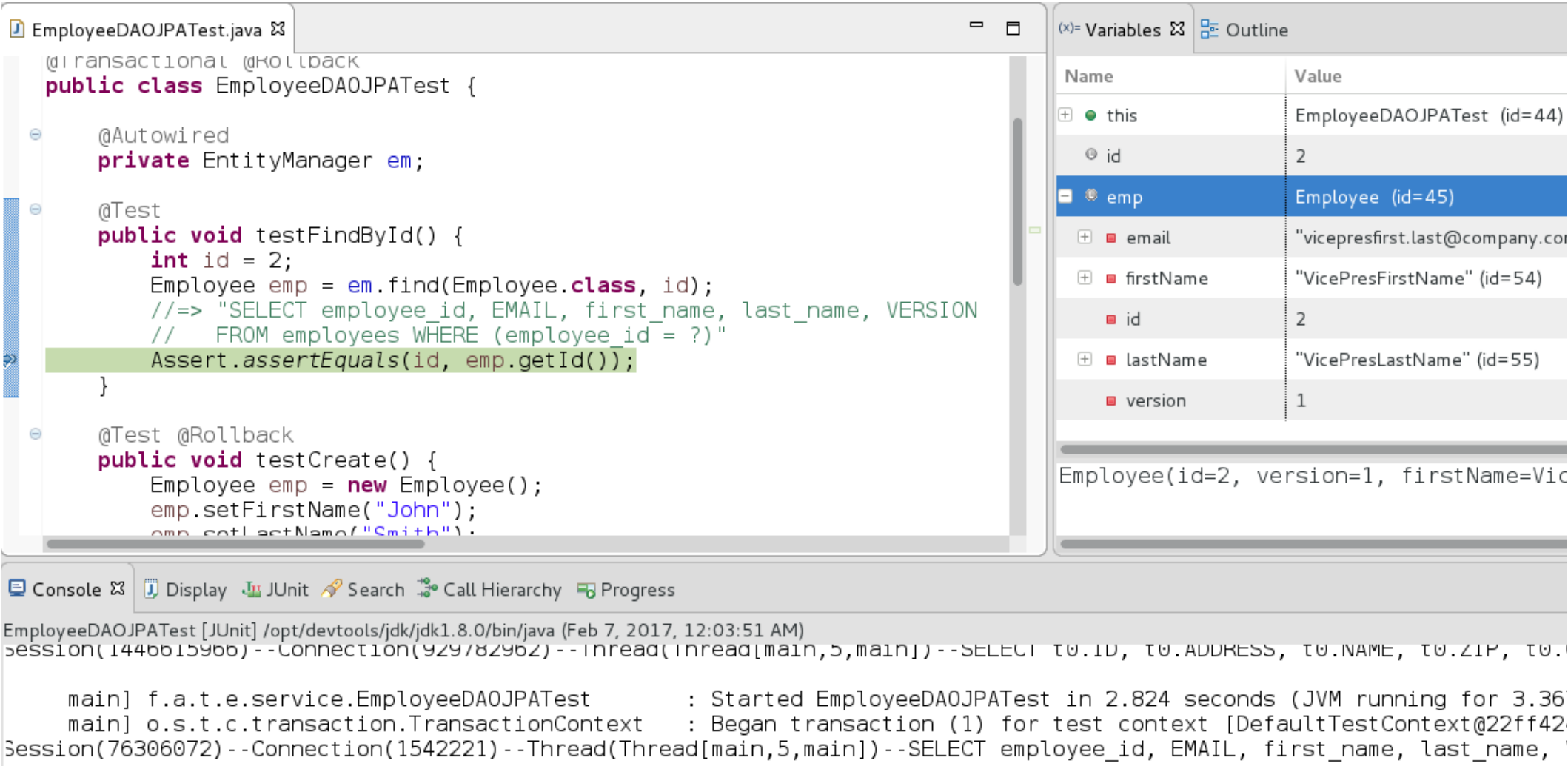
```java
@Data // lombok getter, setter
@Entity
public class Department {

    @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)

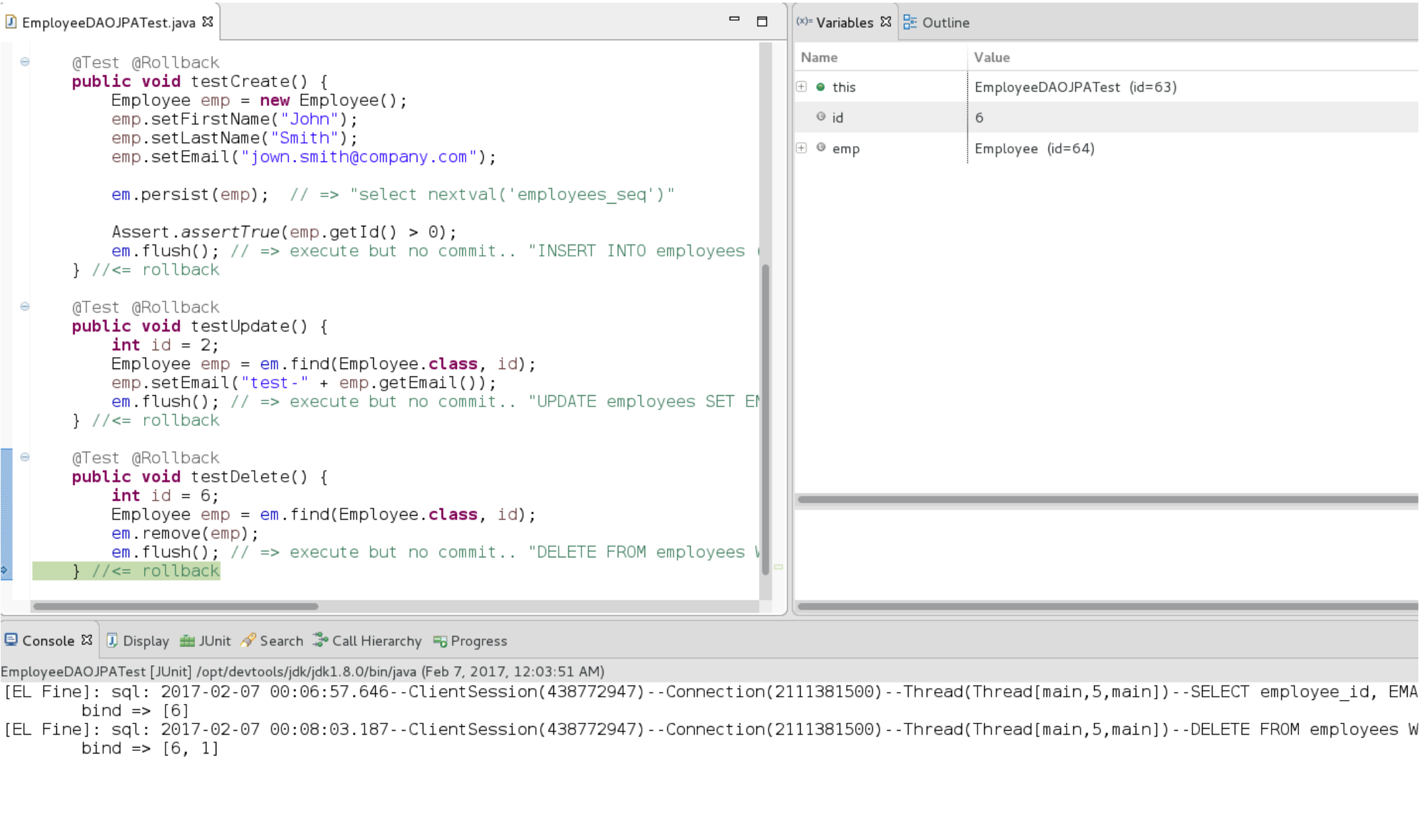
For real with @Column ...

```java
@Data // lombok getter, setter... override hashcode using id field!
@Entity
@Table(name="employees")
public class Employee {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="employees_seq")
    @SequenceGenerator(name="employees_seq", sequenceName="employees_seq", alloc
    @Column(name="employee_id")
    private int id;

    @Version
    private int version;

    @Column(name="first_name")
    private String firstName;

    @Column(name="last_name")
    private String lastName;

    private String email;

    private String address;

    @ManyToOne()
    @JoinColumn(name="department_id")
    private Department department;

    @ManyToOne
    @JoinColumn(name="manager_id")
    private Employee manager;



    @Override
    public int hashCode() {
        return id;
```

FindById with JPA



CRUD with JPA



Dynamic Query
(java.persistence.CriteriaBuilder)

```
public List<Hotel> findByQuery_JPAStrings_noBindVars(String critNameLike, String critAddressLike, String cri
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Hotel> cq = cb.createQuery(Hotel.class);
    Map<String,Object> params = new HashMap<>();
    Root<Hotel> root = cq.from(Hotel.class);
    cq.select(root);
    List<Predicate> predicates = new ArrayList<>();

    if (critNameLike != null) {
        predicates.add(cb.like(root.get("name"), critNameLike));
    }
    if (critAddressLike != null) {
        predicates.add(cb.like(root.get("address"), critAddressLike));
    }
    if (critCityNameLike != null) {
        predicates.add(cb.like(root.get("city").get("name"), critCityNameLike));
    }

    Predicate andPredicates = cb.and(predicates.toArray(new Predicate[predicates.size()]));
    cq.where(andPredicates);
    TypedQuery<Hotel> q = em.createQuery(cq);
    for(Map.Entry<String,Object> e : params.entrySet()) {
        q.setParameter(e.getKey(), e.getValue());
    }
    List<Hotel> res = q.getResultList();
    return res;
}
```

Dynamic Query using Bind-Variables

```
public List<Hotel> findByQuery_JPAStrings(String critNameLike, String critAddressLike, String critCityNameL
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Hotel> cq = cb.createQuery(Hotel.class);
    Map<String,Object> params = new HashMap<>();
    Root<Hotel> root = cq.from(Hotel.class);
    cq.select(root);
    List<Predicate> predicates = new ArrayList<>();

    if (critNameLike != null) {
        predicates.add(cb.like(root.get("name"), cb.parameter(String.class, "nameLike")));
        params.put("nameLike", critNameLike);
    }
    if (critAddressLike != null) {
        predicates.add(cb.like(root.get("address"), cb.parameter(String.class, "addressLike")));
        params.put("addressLike", critAddressLike);
    }
    if (critCityNameLike != null) {
        predicates.add(cb.like(root.get("city").get("name"), cb.parameter(String.class, "cityNameLike")));
        params.put("cityNameLike", critCityNameLike);
    }

    Predicate andPredicates = cb.and(predicates.toArray(new Predicate[predicates.size()]));
    cq.where(andPredicates);
    TypedQuery<Hotel> q = em.createQuery(cq);
    for(Map.Entry<String,Object> e : params.entrySet()) {
        q.setParameter(e.getKey(), e.getValue());
    }
    List<Hotel> res = q.getResultList();
    return res;
}
```

Dynamic Query …
String type checking?

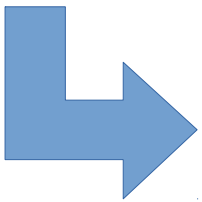
Which one is correct???
Discover it by Exception on PROD !

```
cb.get(“addr”)        // column name in Database
cb.get(“address”)  // field name in java
cb.get(“adress”)    // with a Typo
cb.get(“city_id”).get(“name”)  // after refactoring db schema
```

Generated *_ class MetaModel + Compile Type Check

```java
@Data
@Entity
public static class HotelEntity {

    @Id private int id;

    @ManyToOne private City city;
    private String name;
    private String address;
    private String zip;
}
```

```xml
<plugin>
    <groupId>com.ethlo.persistence.tools</groupId>
    <artifactId>eclipselink-maven-plugin</artifactId>
    <version>2.6.3</version>
    <executions>
        <execution>
            <id>modelgen</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>modelgen</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <includes>fr.an.tests.eclipselink.domain</includes>
        <generatedSourcesDirectory>${basedir}/generated-sources/apt</generatedSourcesDirecto
    </configuration>
</plugin>
```



Generate

```java
/** (pseudo code) Hotel MetaModel class, generated from Hotel */
public static class HotelEntity_ {

    public static final Prop<String> id = new Prop<>(String.class, "id");

    public static final Prop<City> city = new Prop<>(City.class, "city");
    public static final Prop<String> name = new Prop<>(String.class, "name");
    public static final Prop<String> address = new Prop<>(String.class, "address");
    public static final Prop<String> zip = new Prop<>(String.class, "zip");
}

public static class Prop<T> {
    public final String name;
    public final Class<T> type;
    public Prop(Class<T> type, String name) {
        this.type = type;
        this.name = name;
    }
}
```

DynamicCriteria using MetaModel

```
public List<Hotel> findByQuery(HotelSpecification spec) {
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Hotel> cq = cb.createQuery(Hotel.class);
    Map<String,Object> params = new HashMap<>();
    Root<Hotel> root = cq.from(Hotel.class);
    cq.select(root);
    List<Predicate> predicates = new ArrayList<>();

    if (spec.nameLike != null) {
        predicates.add(cb.like(root.get(Hotel_.name), cb.parameter(String.class, "nameLike")));
        params.put("nameLike", spec.nameLike);
    }
    if (spec.addressLike != null) {
        predicates.add(cb.like(root.get(Hotel_.address), cb.parameter(String.class, "addressLike")));
        params.put("addressLike", spec.addressLike);
    }
    if (spec.cityNameLike != null) {
        predicates.add(cb.like(root.get(Hotel_.city).get(City_.name), cb.parameter(String.class, "cit
        params.put("cityNameLike", spec.cityNameLike);
    }

    Predicate andPredicates = cb.and(predicates.toArray(new Predicate[predicates.size()]));
    cq.where(andPredicates);
    TypedQuery<Hotel> q = em.createQuery(cq);
    for(Map.Entry<String,Object> e : params.entrySet()) {
        q.setParameter(e.getKey(), e.getValue());
    }
    List<Hotel> res = q.getResultList();
    return res;
}
```

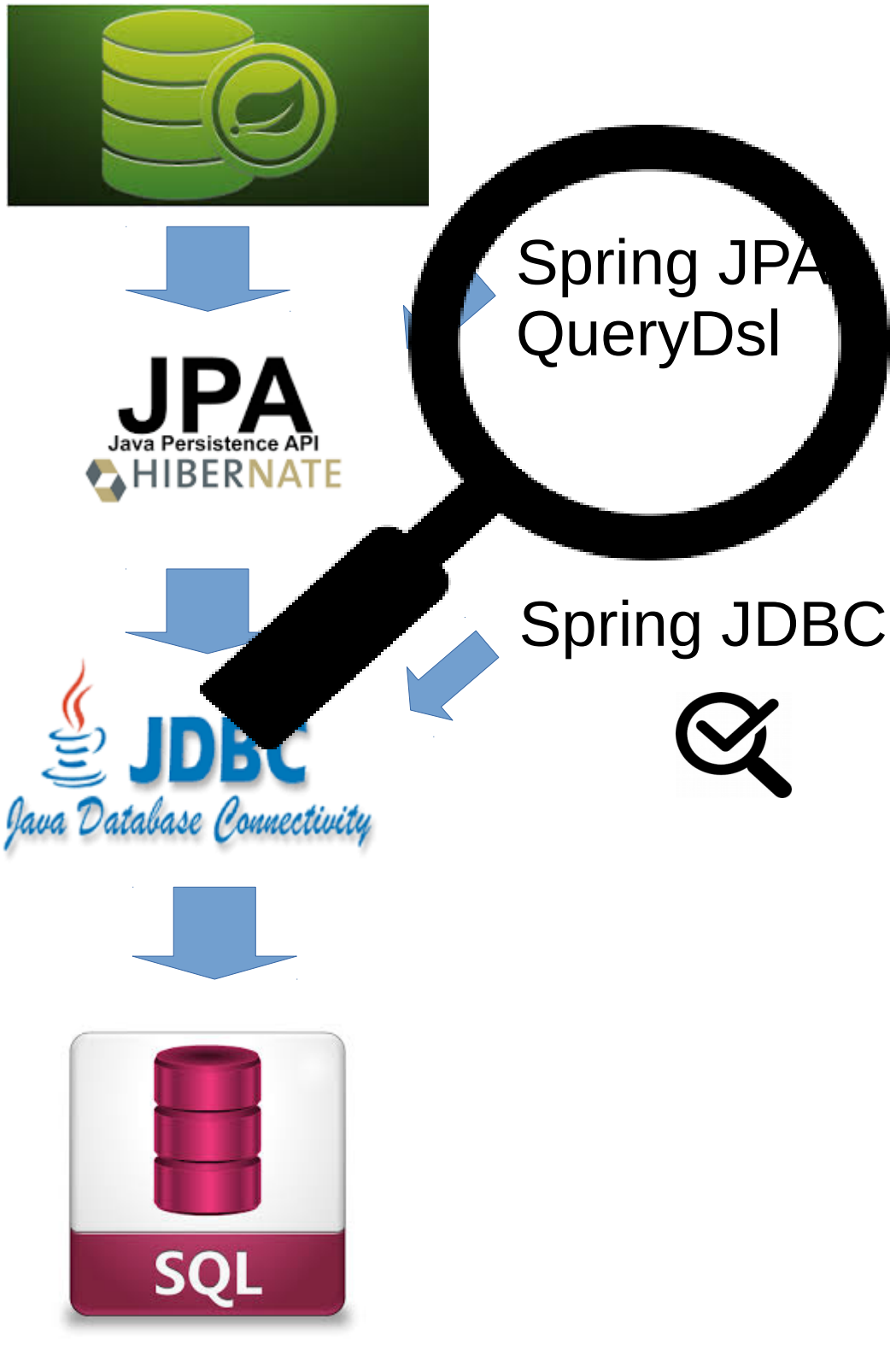
Search Parameters in Criteria class (also called “Specification”)

```
public static class HotelSpecification {
    public String nameLike;
    public String addressLike;
    public String cityNameLike;

    public HotelSpecification nameLike(String nameLike) {
        this.nameLike = nameLike;
        return this;
    }
    public HotelSpecification addressLike(String addressLike) {
        this.addressLike = addressLike;
        return this;
    }
    public HotelSpecification cityNameLike(String cityNameLike) {
        this.cityNameLike = cityNameLike;
        return this;
    }
}
```

Cascading Setters with “return this” for Fluent API

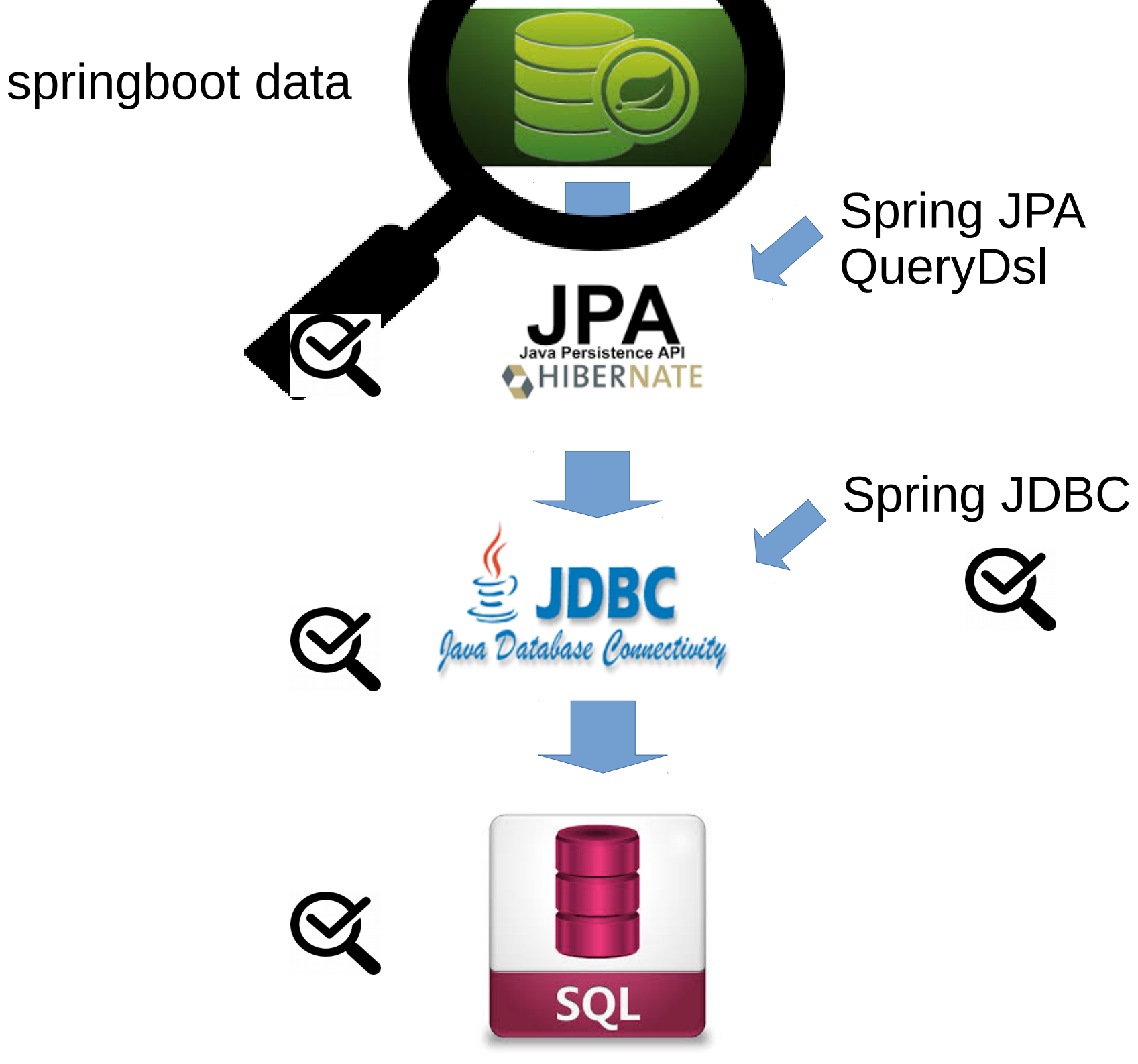
```
HotelSpecification crit1 = new HotelSpecification().
        nameLike("Hilton%");
HotelSpecification crit2 = new HotelSpecification().
        nameLike("Hilton%").cityNameLike("Paris");
HotelSpecification crit3 = new HotelSpecification().
        nameLike("Hilton%").cityNameLike("Paris").addressLike("rue Saint Lazare");

service.findByQuery(crit1);
service.findByQuery(new HotelSpecification().
        nameLike("Hilton%").cityNameLike("Paris"));
```

springboot data



QueryDsl
++More Fluent than JPA

```
public List<Hotel> findByQueryDsl(HotelSpecification spec) {
    JPAQuery<Hotel> query = new JPAQuery<>(em);
    JPAQuery<Hotel> q = query.select(QHotel.hotel).from(QHotel.hotel);

    if (spec.nameLike != null) {
        q.where(QHotel.hotel.name.like(spec.nameLike));
    }
    if (spec.addressLike != null) {
        q.where(QHotel.hotel.address.like(spec.addressLike));
    }
    if (spec.cityNameLike != null) {
        q.where(QHotel.hotel.city.name.like(spec.cityNameLike));
    }

    List<Hotel> res = q.fetch();
    return res;
}
```


springboot data
Spring JPA
QueryDsl
JPA
Java Persistence API
HIBERNATE
Spring JDBC
JDBC
Java Database Connectivity
SQL

```
<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
}
```

Sample Code
using springboot-data Repository

```java
@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 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_1_0_, employee0_.address as address2_1_0_, employee0_.birth_date as birth_da3_1_0_, employee0_.department_id as departme8_1_0_, e
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 departme8_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 FK25ggdalg559udqdvhwxu3p4j3 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 FKmlfr43vjmqglnaleuarwhhveq 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 langage

Also create PK/FK indexes

extends JpaRepository

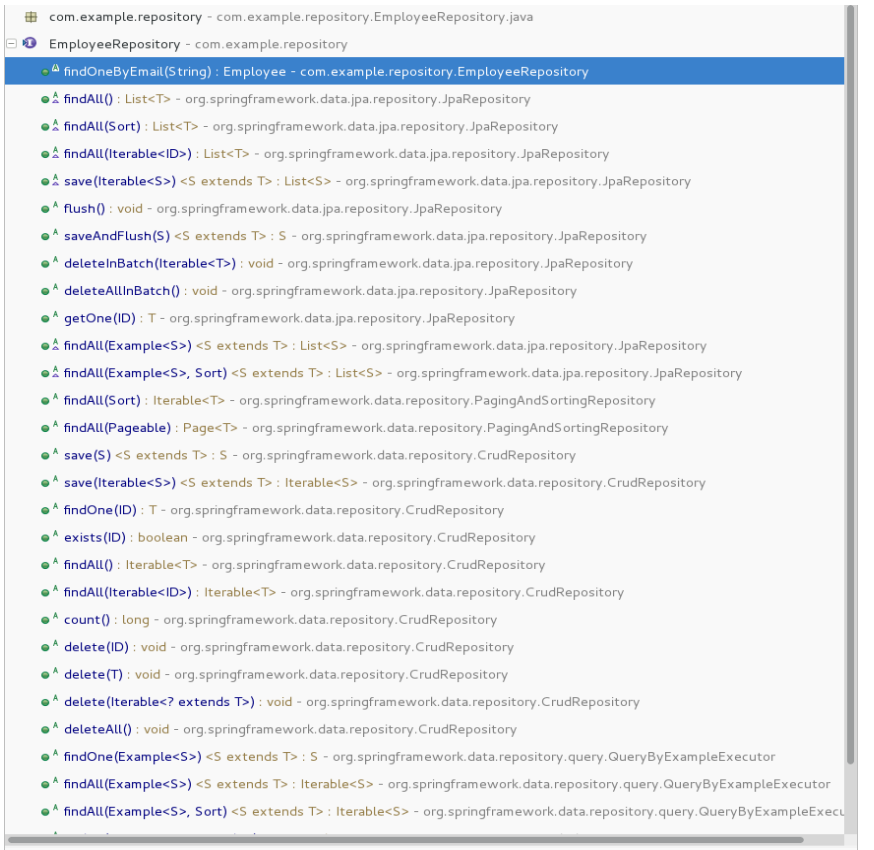
```
interface CityRepository extends JpaRepository<City, Long> {
    // extends JpaRepository<TEntity,TId> ... =>

    @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();

}
```

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)

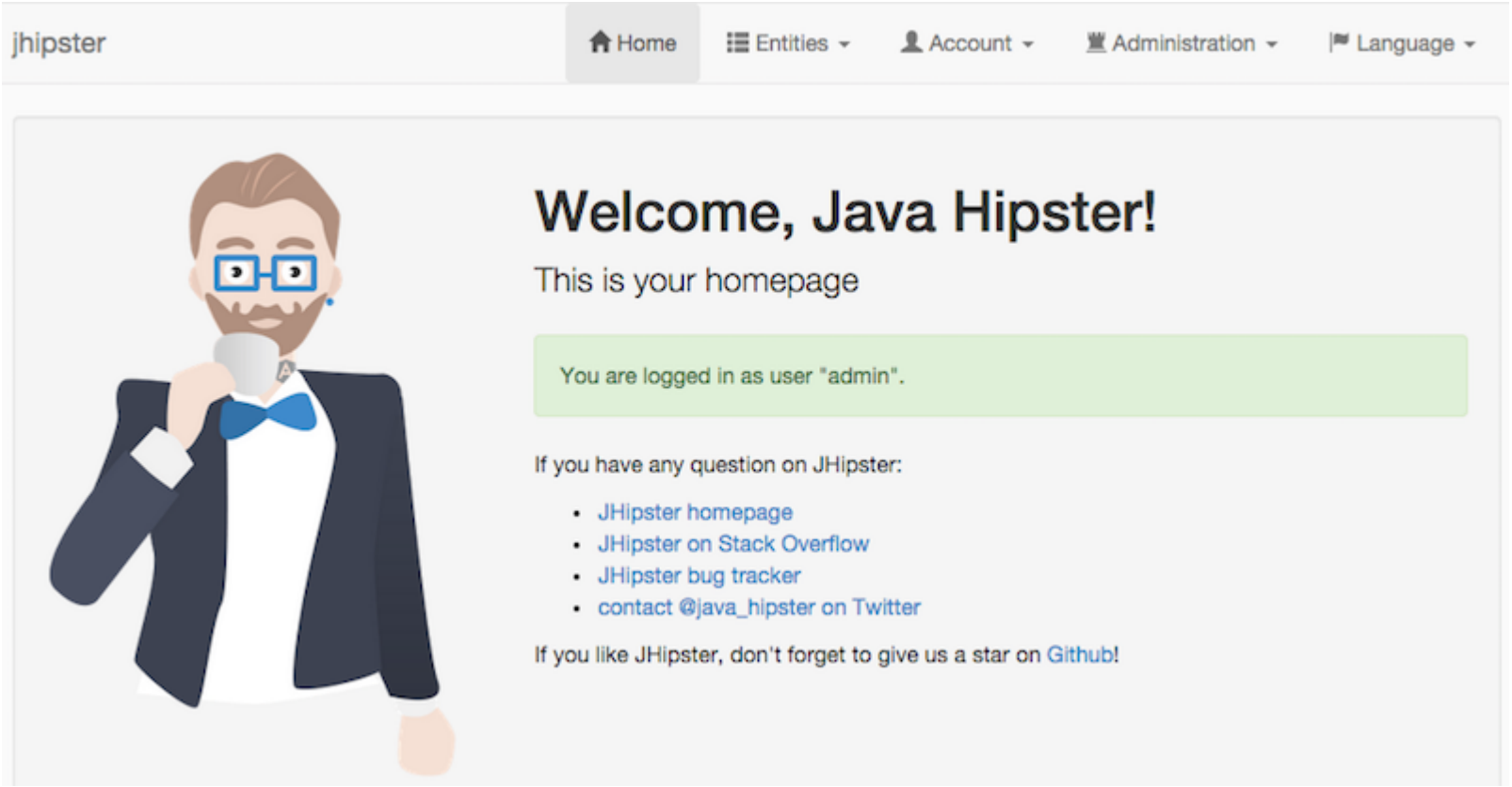
QueryDsl + Spring-Data

```
public interface HotelRepository extends JpaRepository<Hotel,Integer>,
                                         // JpaSpecificationExecutor<Hotel>,
                                         QueryDslPredicateExecutor<Hotel> {

    // using querydsl.Predicate ...  (not java.persistence.Specification)
    @Override Hotel findOne(Predicate qryDslPredicate);
    @Override Iterable<Hotel> findAll(Predicate qryDslPredicate);
    @Override Iterable<Hotel> findAll(Predicate qryDslPredicate, Sort sort);
    @Override Iterable<Hotel> findAll(Predicate qryDslPredicate, OrderSpecifier<?>... orders);
    @Override Iterable<Hotel> findAll(OrderSpecifier<?>... orders);
    @Override Page<Hotel> findAll(Predicate qryDslPredicate, Pageable pageable);
    @Override long count(Predicate qryDslPredicate);
    @Override boolean exists(Predicate qryDslPredicate);
}
```

```
QHotel.hotel h = QHotel.hotel;
Predicate pred = h.name.like(nameLike).and(h.city.like(cityLike));
List<Hotel> hotels1 = hotelRepository.findAll(pred);
List<Hotel> hotels2 = hotelRepository.findAll(h.name.like(nameLike).and(h.city.like(cityLike)));
```

AngularJS + Springboot

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



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

This document:
http://arnaud-nauwynck.github.io/docs/Intro-CodeExtract-Db-Jdbc-JPA-SpringData.pdf