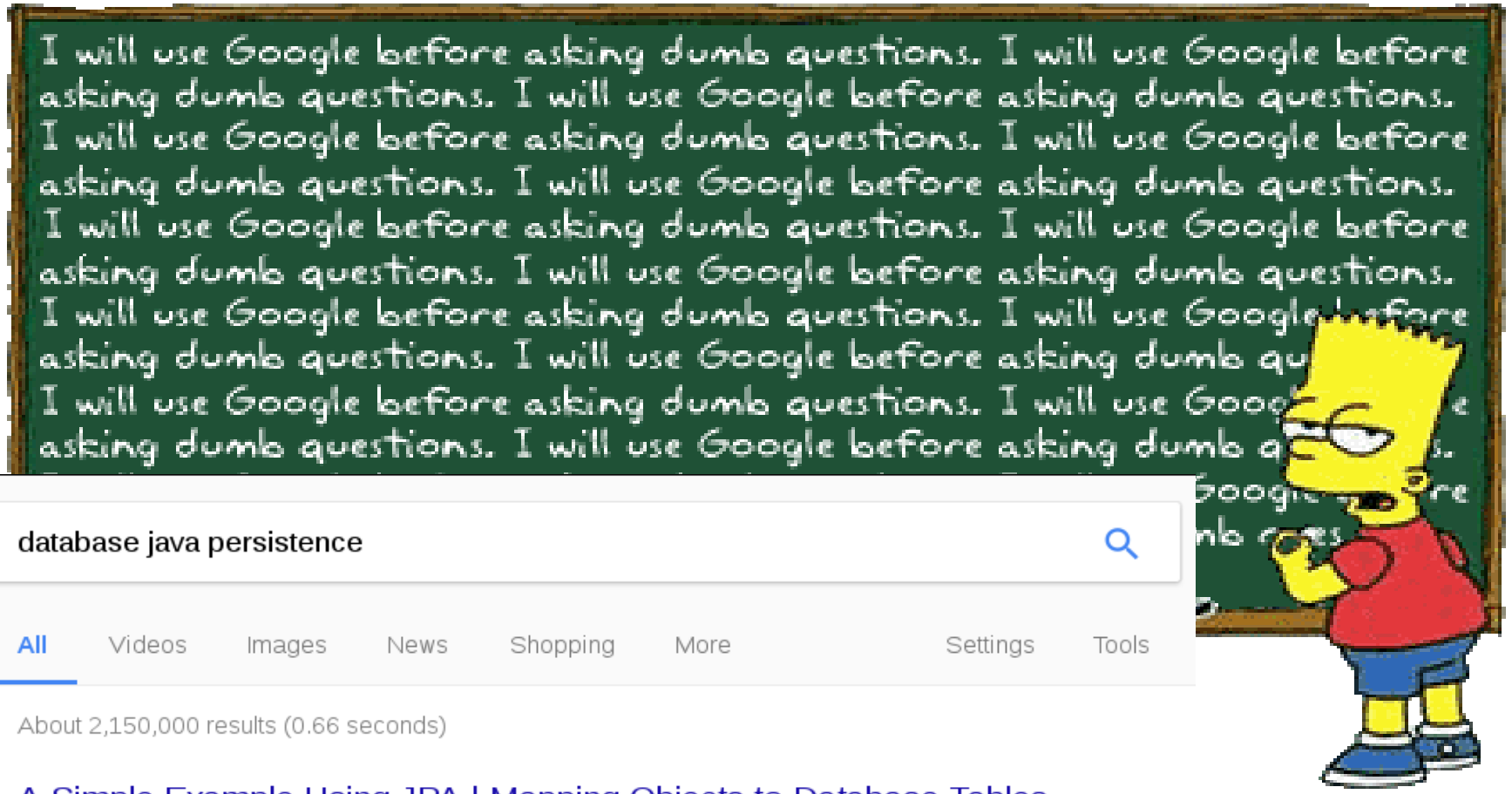
arnaud.nauwynck@gmail.com

Introduction to Db - Jdbc - JPA - SpringData

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



Google

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

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

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

Spring JPA
QueryDsl

JPA
Java Persistence API
HIBERNATE

Spring JDBC

JDBC
Java Database Connectivity

Structured Query Langage
SQL = DDL + DML

The Data Definition Language (DDL) manages table and index structure. The most basic items of DDL are the `CREATE`, `ALTER`, `RENAME`, `DROP` and `TRUNCATE` statements:

- `CREATE` creates an object (a table, for example) in the database, e.g.:

```
CREATE TABLE example(
 column1 INTEGER,
 column2 VARCHAR(50),
 column3 DATE NOT NULL,
 PRIMARY KEY (column1, column2)
);
```

- `ALTER` modifies the structure of an existing object in various ways, for example, adding a column to an existing table or a constraint, e.g.:

```
ALTER TABLE example ADD column4 NUMBER(3) NOT NULL;
```

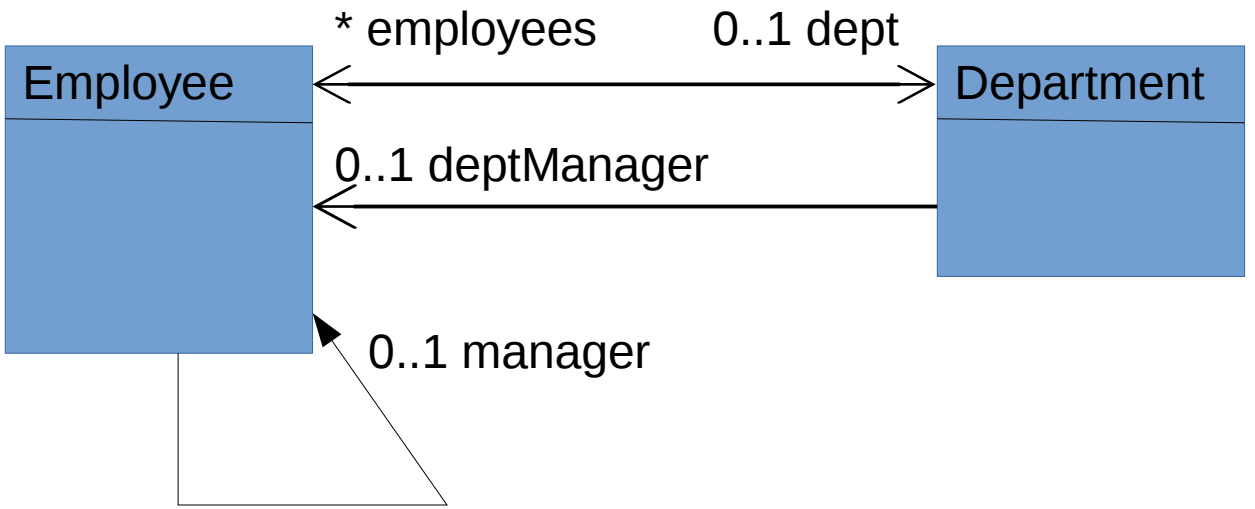
Oracle Sample DataBase

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

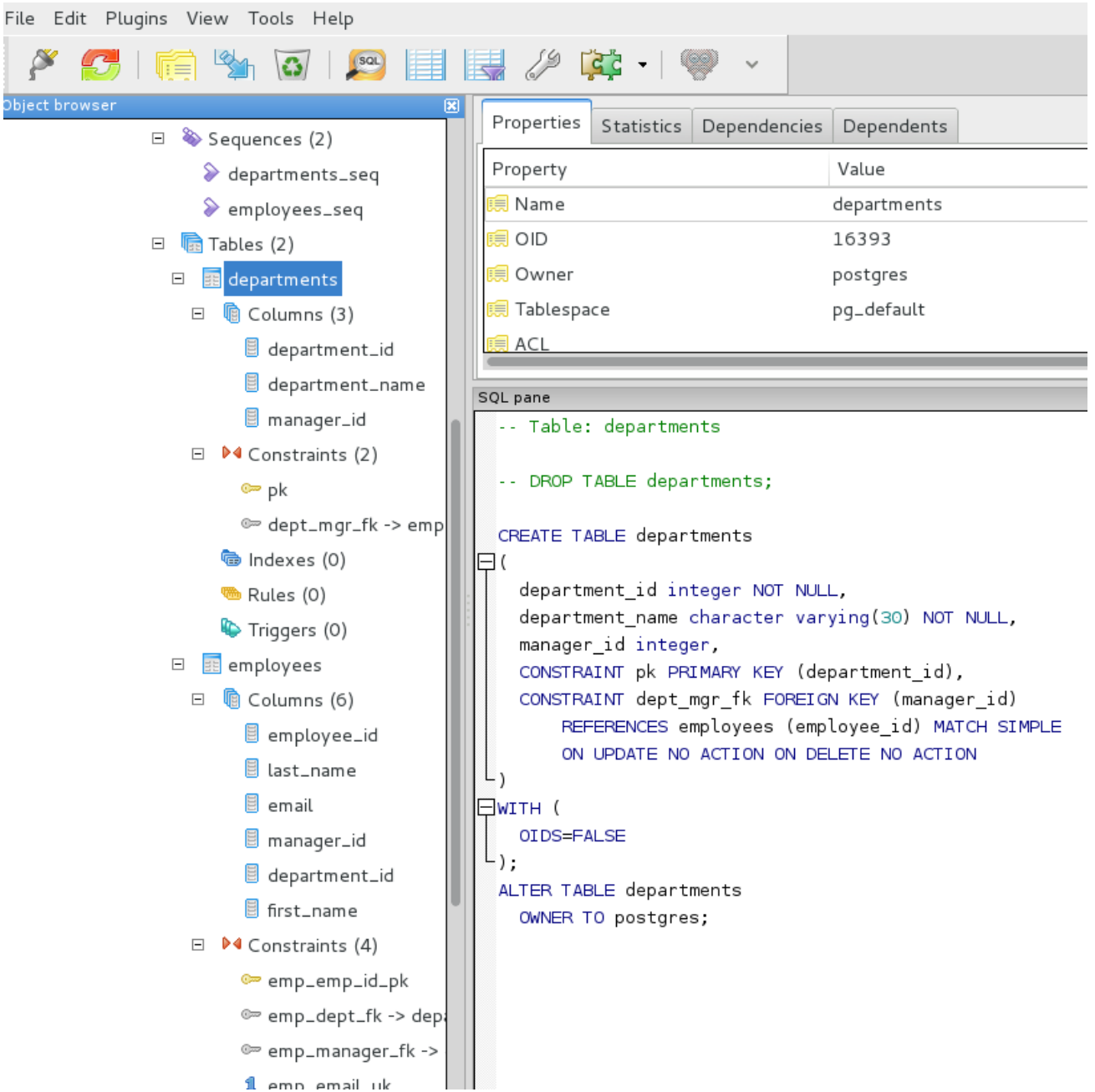
human_resources/hr_cre.sql

```
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



SQL SELECT

Simple
(with WHERE clause)

```sql
SELECT *
 FROM  Book
 WHERE price > 100.00
 ORDER BY title;
```

With “JOIN .. ON”
(and GROUP BY)

```sql
SELECT Book.title AS Title,
       count(*) AS Authors
 FROM  Book
 JOIN  Book_author
   ON  Book.isbn = Book_author.isbn
 GROUP BY Book.title;
```

With SubQueries

```sql
SELECT isbn,
       title,
       price
 FROM  Book
 WHERE price < (SELECT AVG(price) FROM Book)
 ORDER BY title;
```

SQL DML

The Data Manipulation Language (DML) is the subset of SQL used

- `INSERT` adds rows (formally tuples) to an existing table, e.g.:

```
INSERT INTO example
 (field1, field2, field3)
 VALUES
 ('test', 'N', NULL);
```

- `UPDATE` modifies a set of existing table rows, e.g.:

```
UPDATE example
 SET field1 = 'updated value'
 WHERE field2 = 'N';
```

- `DELETE` removes existing rows from a table, e.g.:

```
DELETE FROM example
 WHERE field2 = 'N';
```

Detailed CRUD

CRUD =

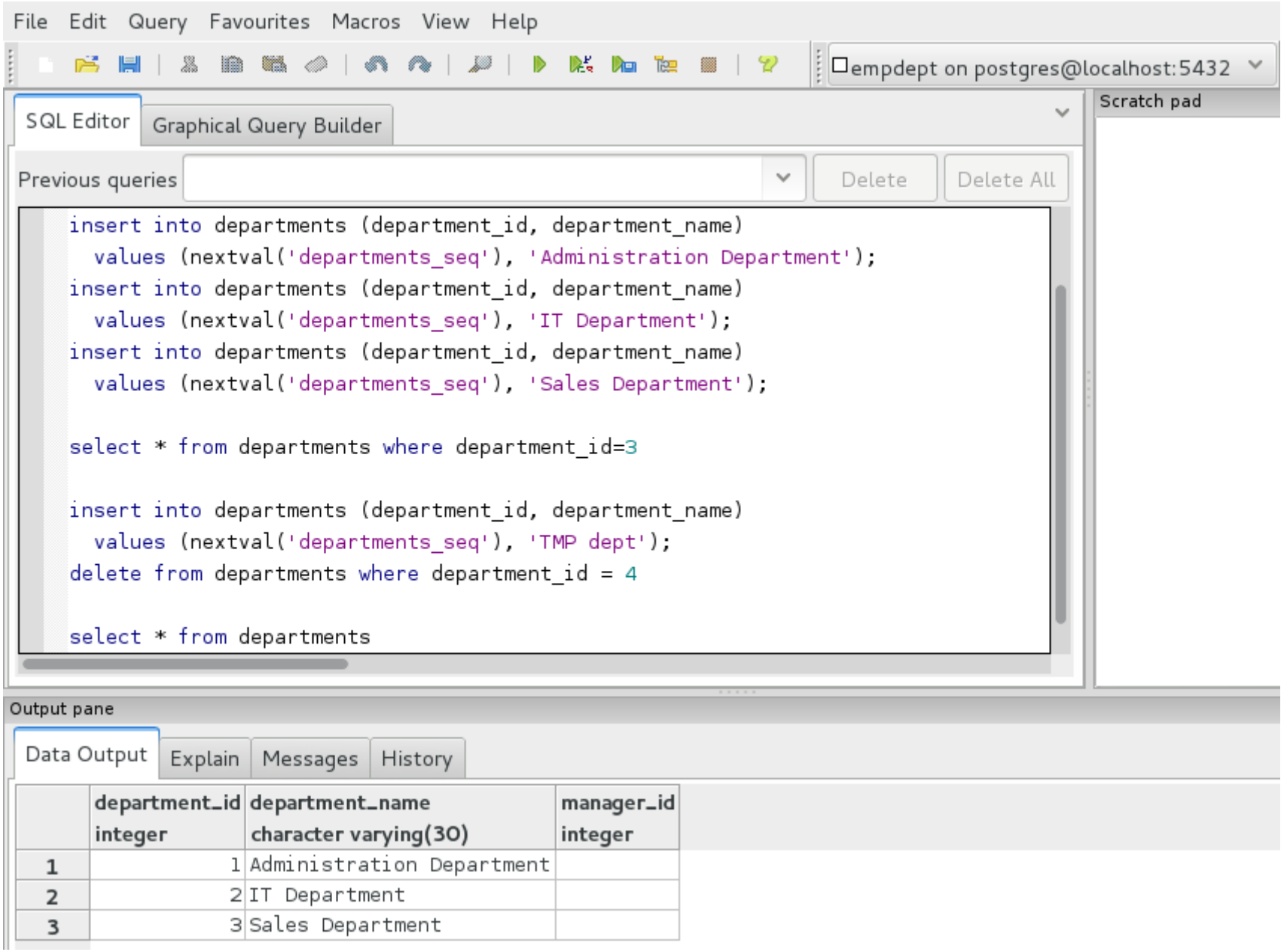
Create **INSERT into <**Table..> (col1, col2, …)
VALUES (?0, ?1, 123, ..)

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

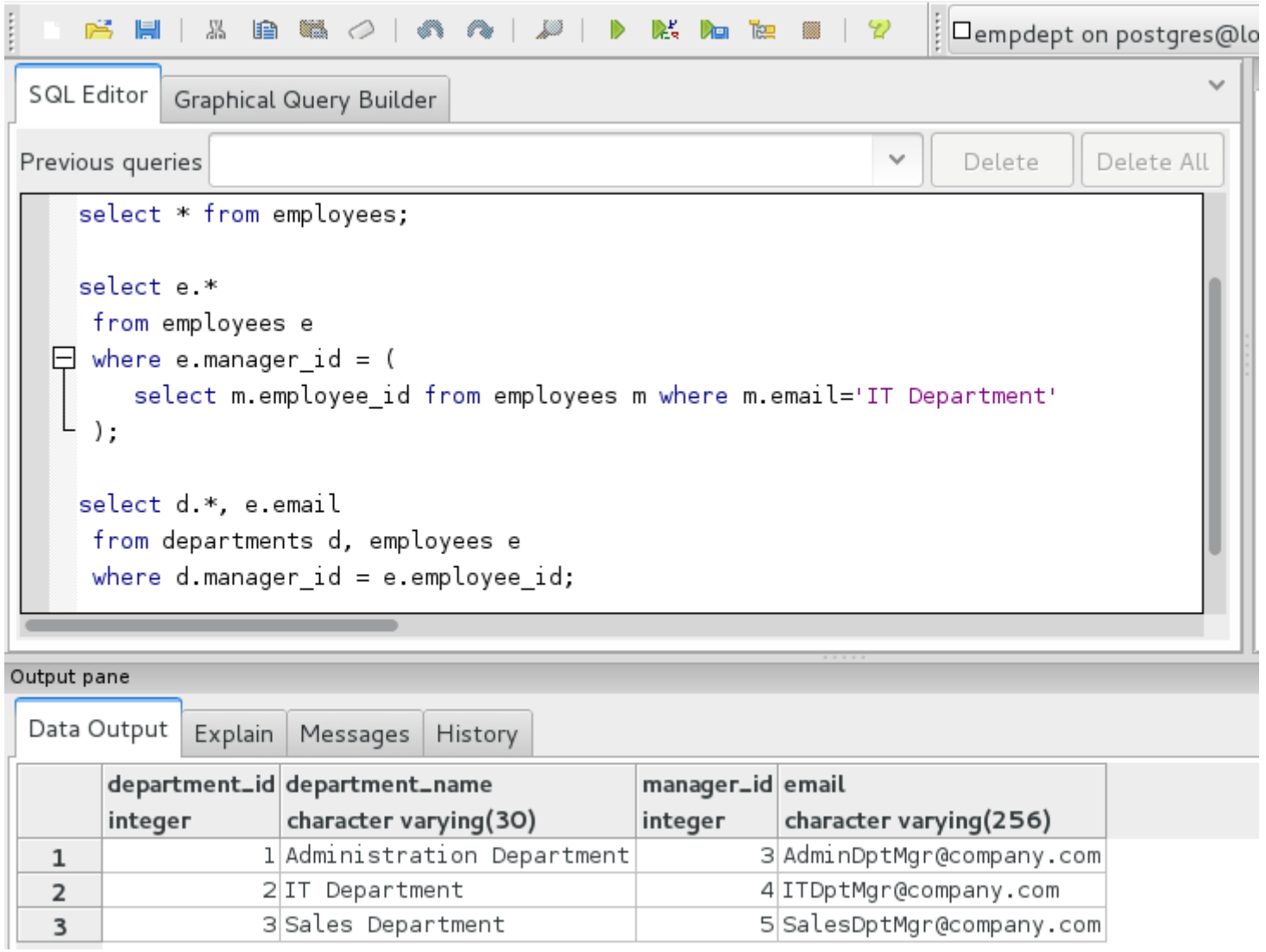
Update **UPDATE** col1=value1, col2=…
from <Table..> where <expr..>

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

Emp-Dept CRUD Sample



Emp-Dept Queries



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

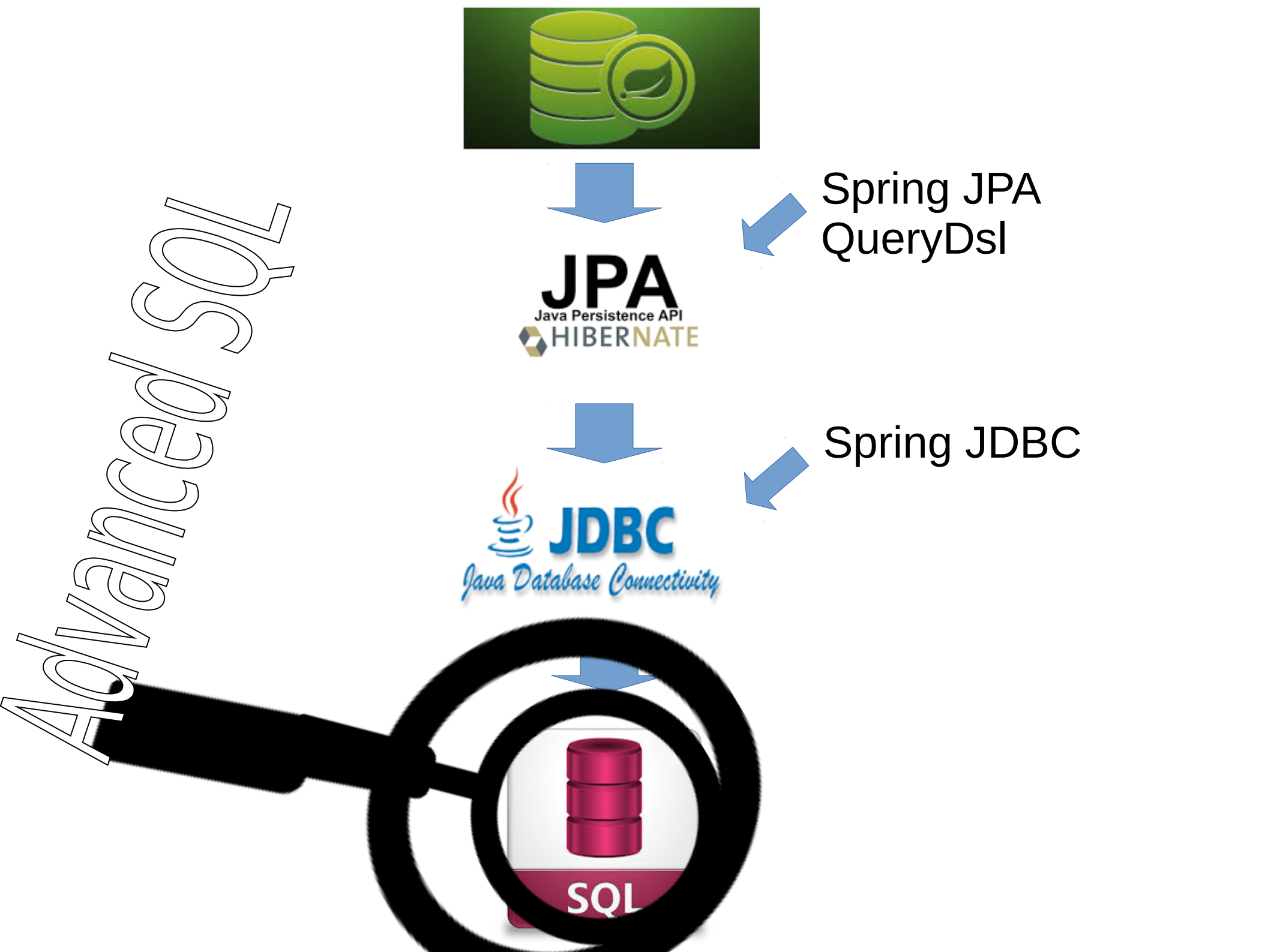
Sample table : employees

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


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

SQL Merge (=Upsert)

- `MERGE` is used to combine the data of multiple tables. It combines the `INSERT` and `UPDATE` functionality via different syntax, sometimes called "upsert".

```
MERGE INTO table_name USING table_reference ON (condition)
WHEN MATCHED THEN
UPDATE SET column1 = value1 [, column2 = value2 ...]
WHEN NOT MATCHED THEN
INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...])
```

Simple SELECT Query

[With Col1, Col2 from Table1... **]**

select /*+HINT */
    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 – explicit JOIN with alias
[limit .. firstRows .. cf also rownum **]**
Order by Col2 **asc**, Col3 **desc**

Aggregate Query

select C1, C2, **count(*)**, **avg**(Col3), min(Col4) ..
from Table ..
where
...
Group by C1, C2

Having ..

Order by ...

SQL Nested Queries

```
SELECT b.isbn, b.title, b.price, sales.items_sold, sales.company_nm
FROM Book b
  JOIN (SELECT SUM(Items_Sold) Items_Sold, Company_Nm, ISBN
        FROM Book_Sales
        GROUP BY Company_Nm, ISBN) sales
  ON sales.isbn = b.isbn
```

Example 2:
select .. **from (select** .. where ..**)** where .. **not in (select** ..where **)**

Analytical Query Functions

select …
analyticalFunc[first,last,nth,min,max,avg,..](..)
OVER (PARTITION BY ..)
from …
where ..

```
SELECT empno, deptno, sal,
       AVG(sal) OVER (PARTITION BY deptno) AS avg_dept_sal
FROM   emp;

     EMPNO     DEPTNO        SAL AVG_DEPT_SAL
---------- ---------- ---------- ------------
      7782         10       2450   2916.66667
      7839         10       5000   2916.66667
      7934         10       1300   2916.66667

      7566         20       2975         2175
      7902         20       3000         2175
      7876         20       1100         2175
      7369         20        800         2175
      7788         20       3000         2175

      7521         30       1250   1566.66667
      7844         30       1500   1566.66667
```

Hierarchical Queries

Exemple: clause “where” using Hierarchy


```
SELECT employee_id, last_name, manager_id
   FROM employees
   CONNECT BY PRIOR employee_id = manager_id;

EMPLOYEE_ID LAST_NAME                 MANAGER_ID
----------- ------------------------- ----------
        101 Kochhar                           100
        108 Greenberg                         101
        109 Faviet                            108
        110 Chen                              108
        111 Sciarra                           108
```

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
```

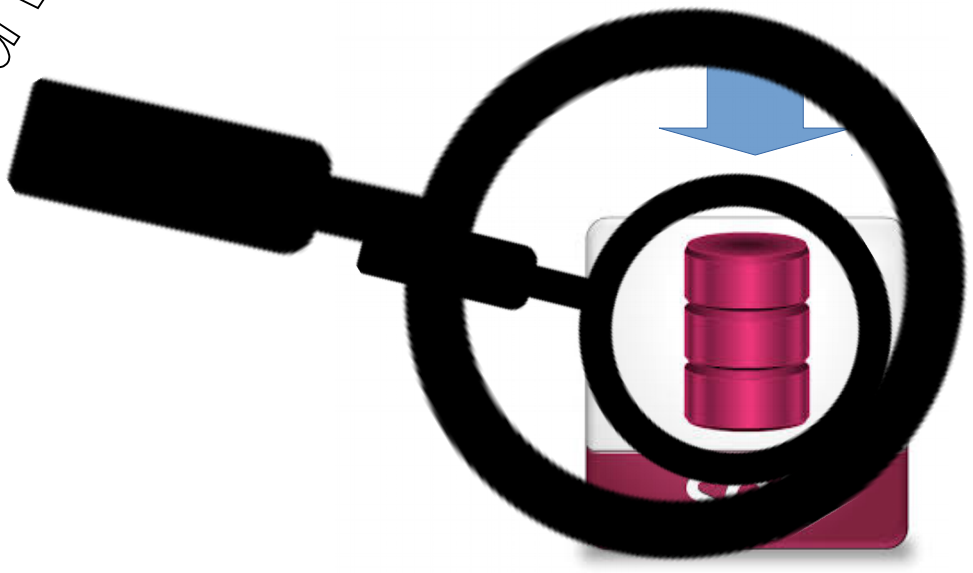
Advanced Databases

Spring JPA
QueryDsl

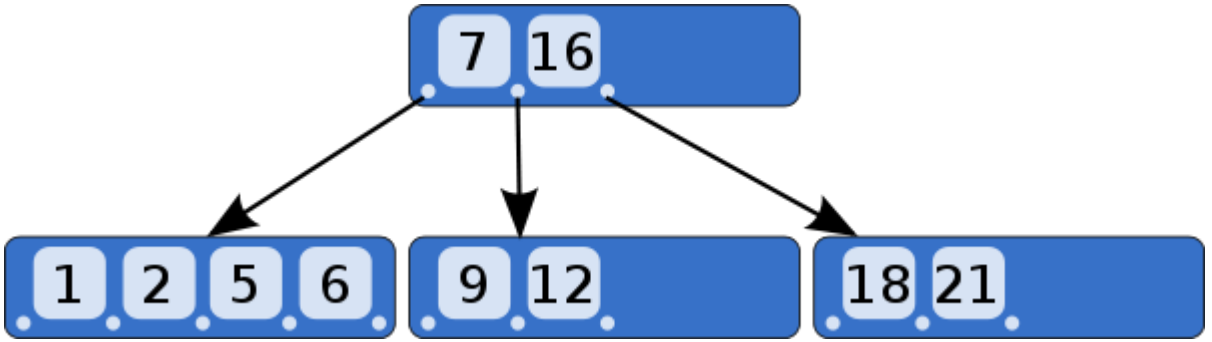
JPA
Java Persistence API
HIBERNATE

Spring JDBC

JDBC
Java Database Connectivity



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



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

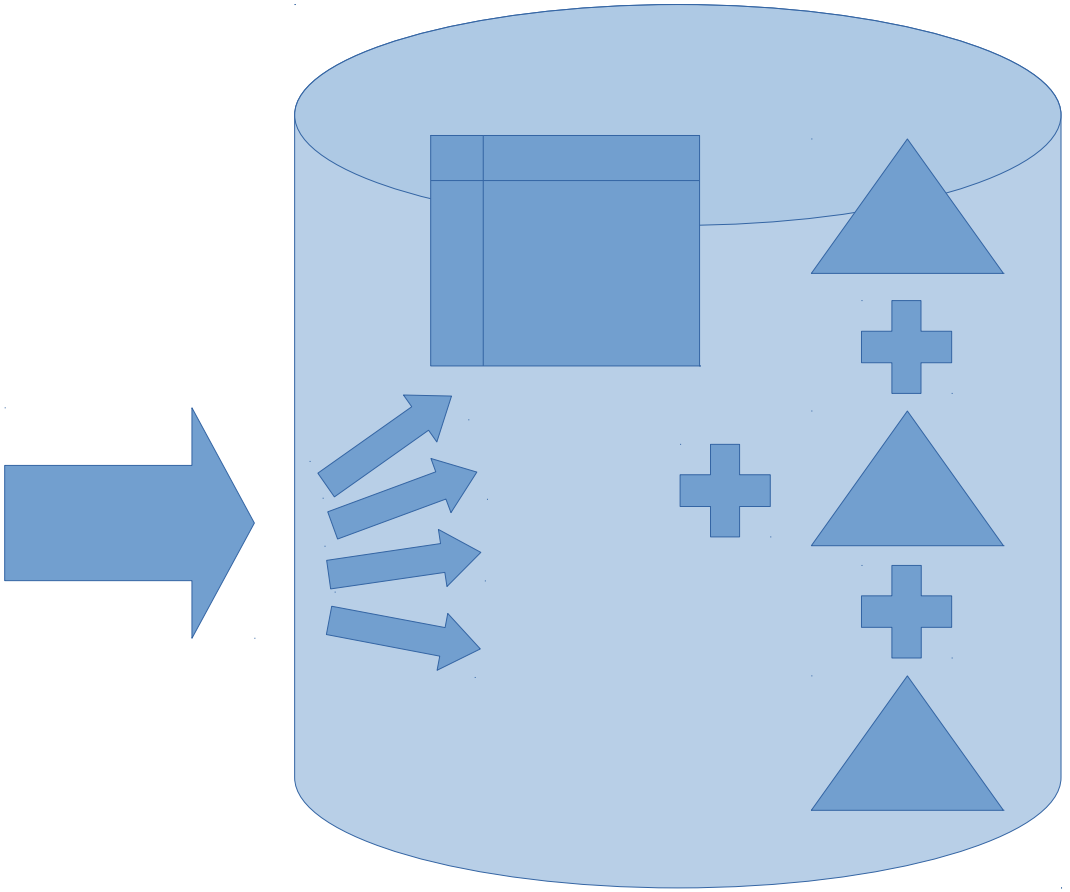
1 Row = N Columns

Compare row by Col1,Col2,ColK :

```
if (a.col1 < a.col1)          => -1
else if (a.col1 > a.col1) => +1
else {
   if (a.col2 < b.col2) … => ..
   ..   { ...                          => 0
} }
```

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

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

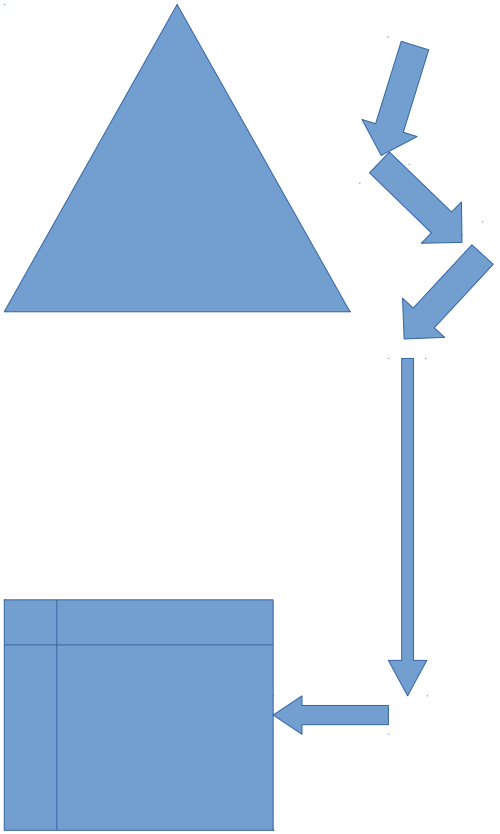


PK INDEX : col (ID)

INDEX2 : (Col2,Col3,ID)

INDEX3 : (Col3,Col2,Col6)

Select .. where ID = ?



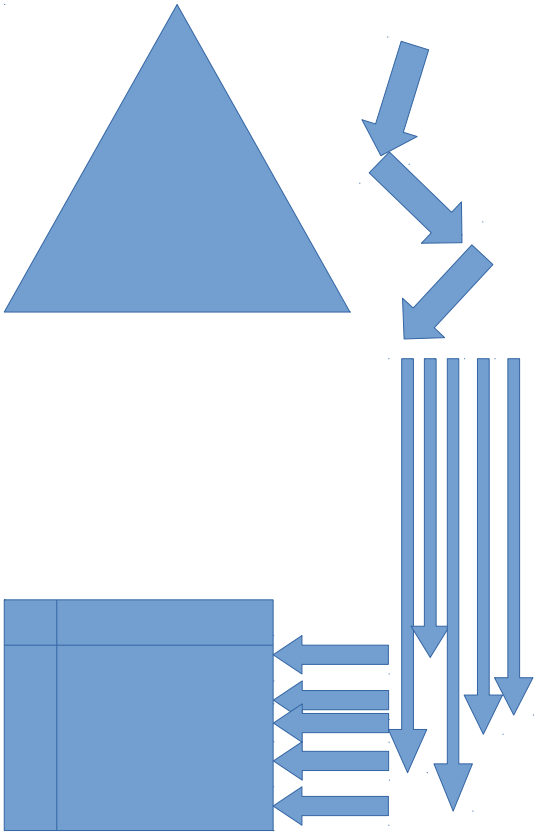
Execution Plan

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

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



Execution Plan

Step 1: Index Range Scan
lookup = Log(N) logical reads
+ scan non unique

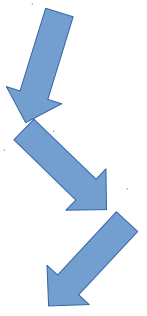
=> foreach.. get “rowId” (=address)

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

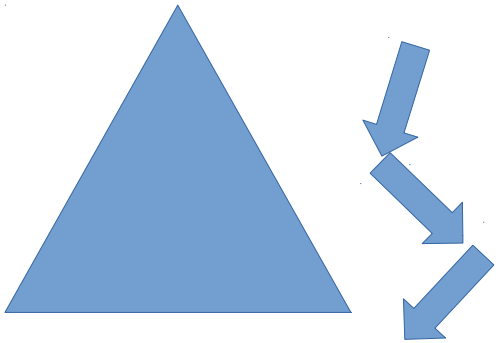
Applicable Index(es) ?
for ".. where Col2=? And Col5=?"

- PK INDEX : col ~~(ID)~~
- INDEX (Col2,~~Col6,Col5)~~
- 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
Log(N) logical reads ...unique/scan

=> read Col3,Col4 value from INDEX

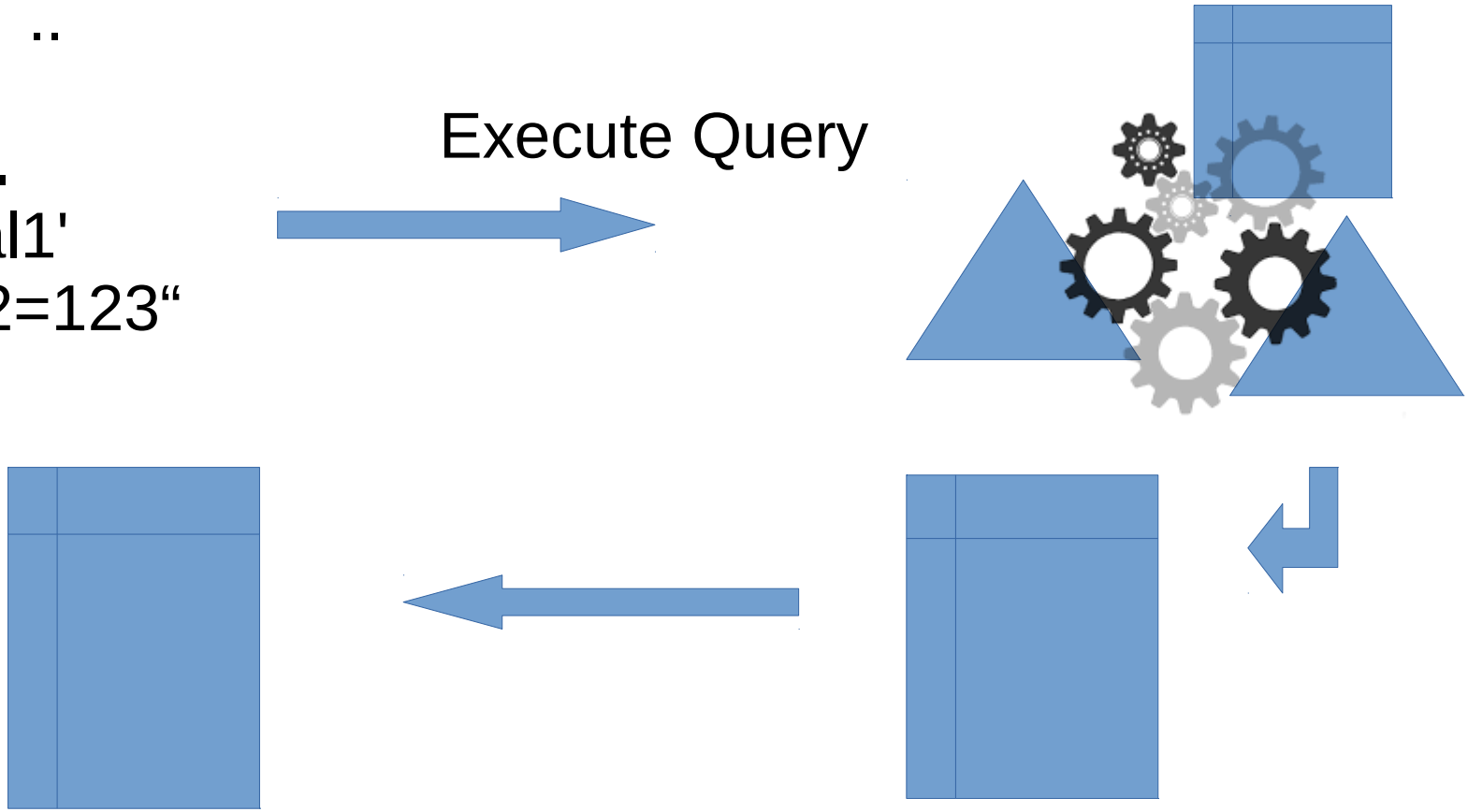
.. NO need table lookup by rowid

Optim = Index Coverage

Query Execution Engine

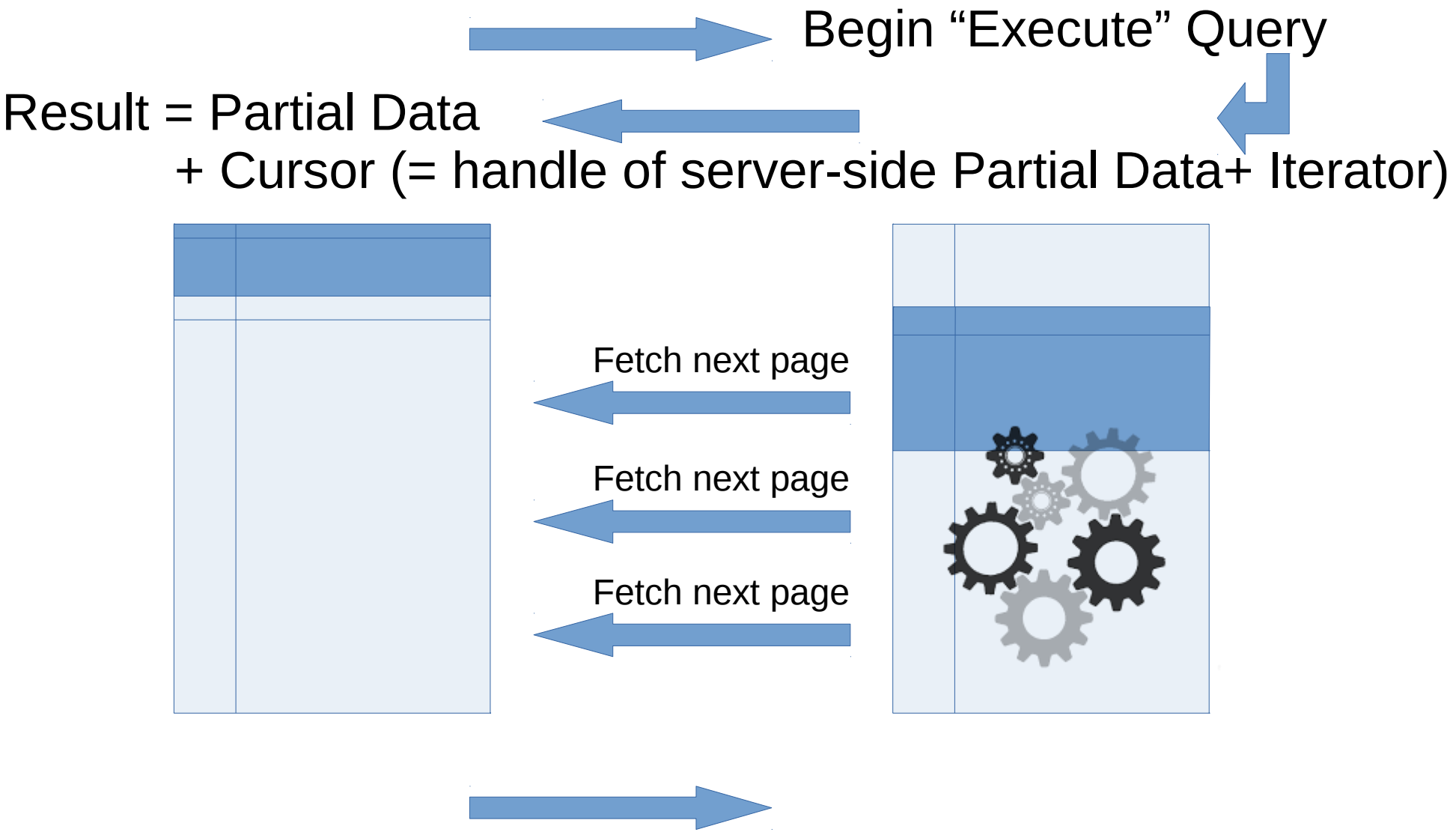
“Select ..
from ..
where ..
col1='val1'
and col2=123“

Execute Query



Result : Tabular Data Format = “ResultSet”

Huge Result .. Server-Side “Cursor”



SoftParse – HardParse … PrepareQuery + BindVariable

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

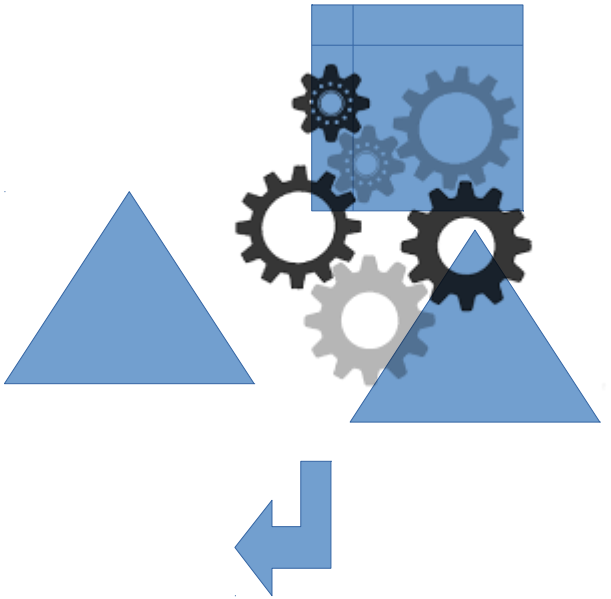
BindVariable:
set(0, “val1”)
set(1, 1234)

First Seen ?
HARD Parse

Compute
Execution Plan

Put in Cache

Already Seen ?
SOFT Parse
= create Cursor



Explain Execution Plan

empdept on postgres@lo

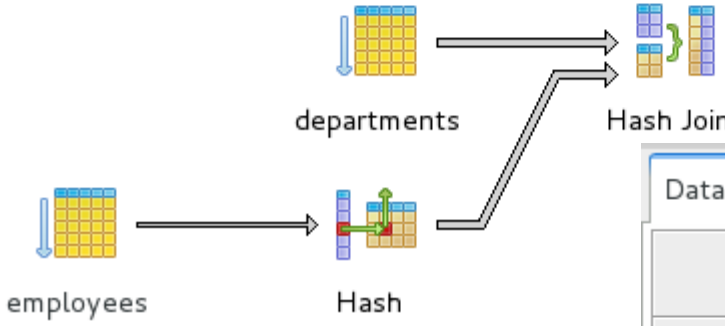
SQL Editor | Graphical Query Builder

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

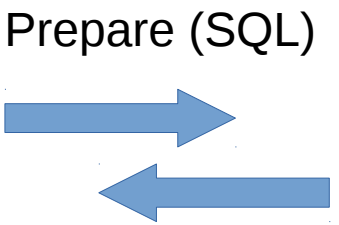


Data Output | Explain | Messages | History

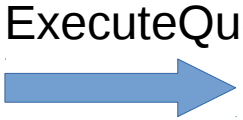
	QUERY PLAN text
1	Hash Join (cost=10.68..31.05 rows=102 width=606)
2	Hash Cond: (d.manager_id = e.employee_id)
3	-> Seq Scan on departments d (cost=0.00..16.80 rows=680 width=90)
4	-> Hash (cost=10.30..10.30 rows=30 width=520)
5	-> Seq Scan on employees e (cost=0.00..10.30 rows=30 width=520)

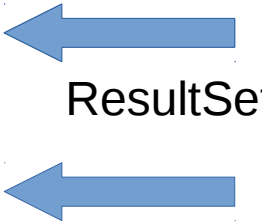
Query / Prepared Query Execution

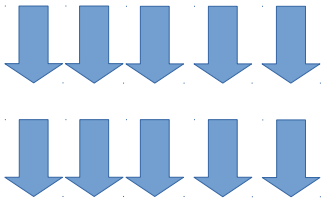
Select ..
from ..
where ..
col1=**?0**
and col2=**?1**



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







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

… = JDBC API Explained

Spring JPA
QueryDsl

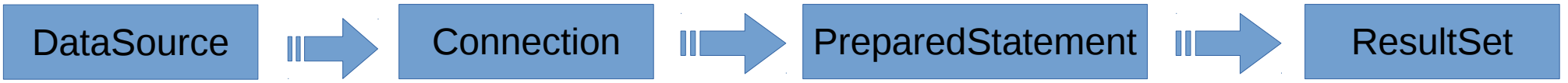
JPA
Java Persistence API
HIBERNATE

Spring JDBC

JDBC
Java Database Connectivity

SQL

import java.sql.*;



Sample Jdbc (Test with Rollback)

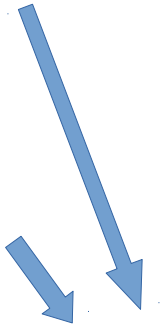
```
@Test
public void testExecuteRollback_before() throws SQLException {
    Connection cx = dataSource.getConnection();
    cx.setAutoCommit(false);
    try {
        // ** execute **
        try(PreparedStatement stmt = cx.prepareStatement("select 'Hello'")) {
            ResultSet rs = stmt.executeQuery();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

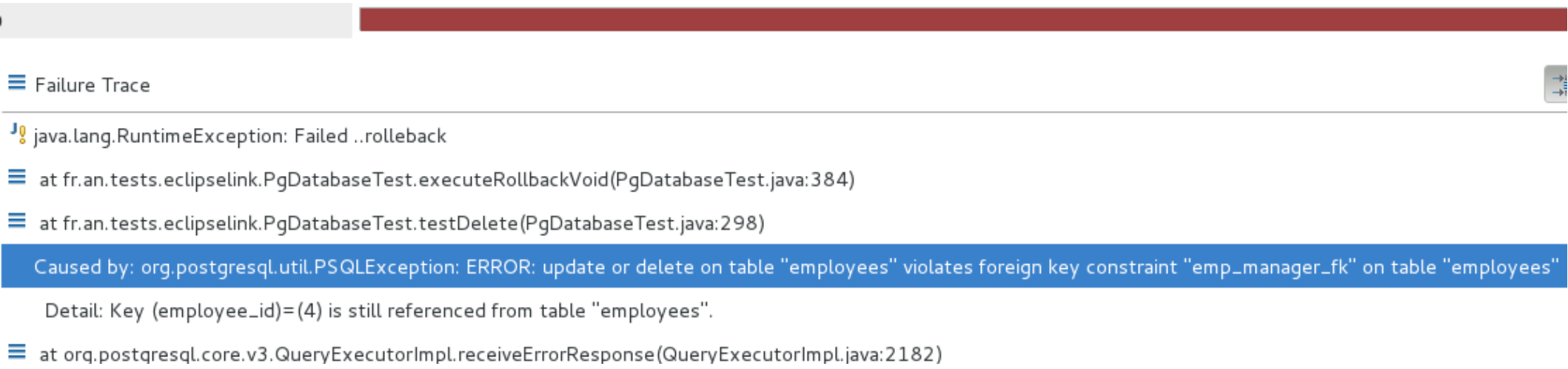
CRUD 4 / 4 : DELETE

```
@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);
        }
    });
}
```

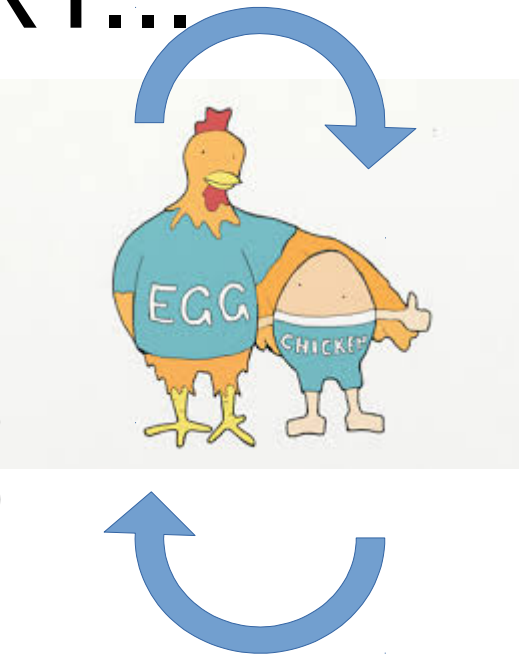
Check my Foreign Keys …

Looks the simplest code to DELETE ? …
It is the hardest to execute safely !!

Because you must clear
all incoming Foreign Keys to this PK first



And What About INSERT... Chicken & Egg

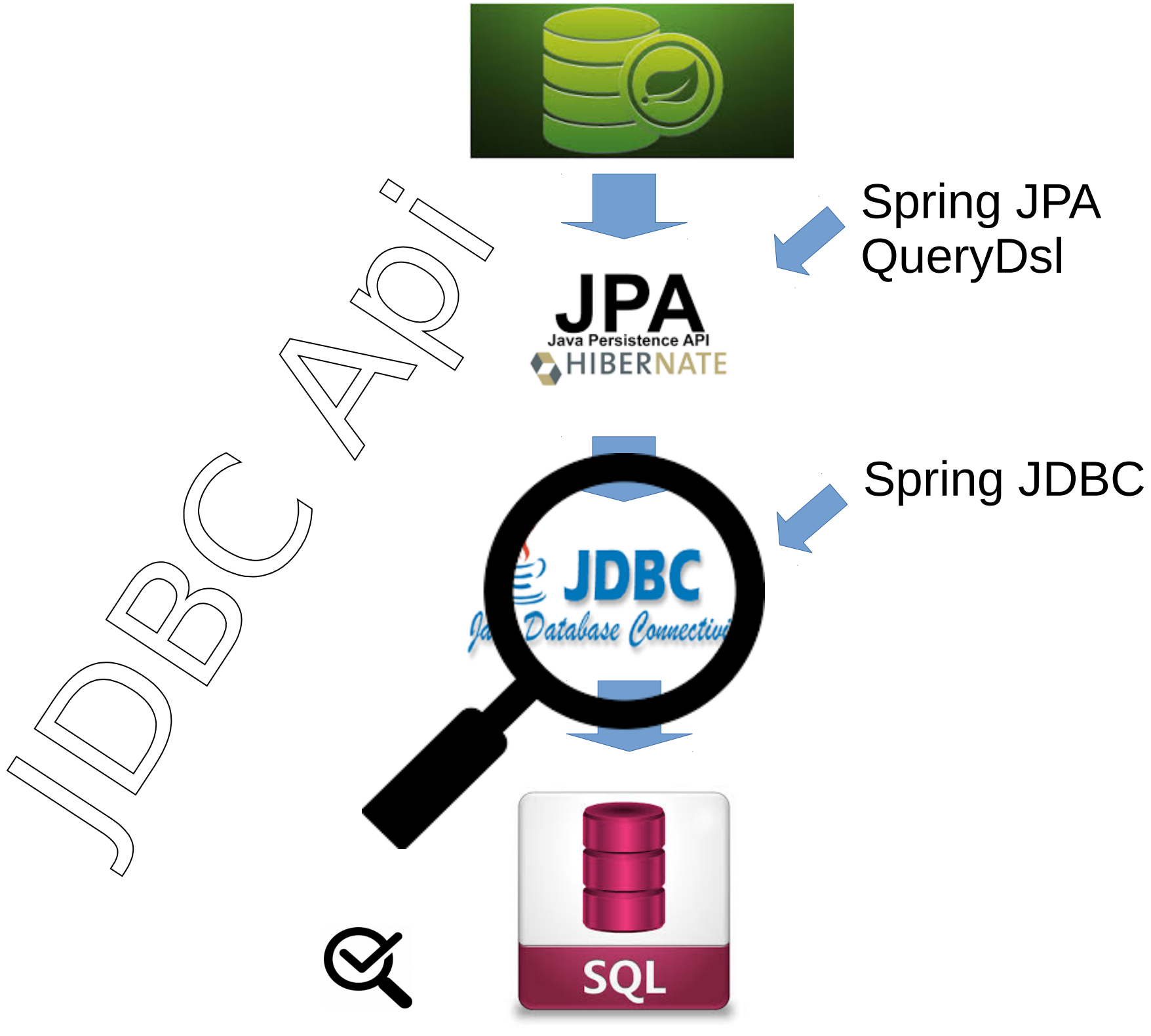


How to ?

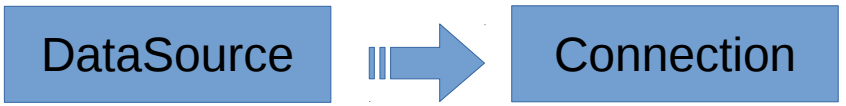
INSERT CHICKEN .. FK to EGG ID (not exist yet)
INSERT EGG .. FK to CHICKEN ID (not exist yet)

Solution 1/
INSERT nextval(..)… CHICKEN … **NULL** FK) => newChickenID
INSERT nextval(..)… EGG … CHICKEN_ID => newEggID
UPDATE CHICKEN set EGG_ID = newEggID

Solution 2/ … use Database **Deferred Constraint Check**
Select nextval(..) => newChickenID
Select nextval(..) => newEggId
INSERT … CHICKEN … newChickenID –-- no exist yet .. deferred check!
INSERT … EGG … newEggID

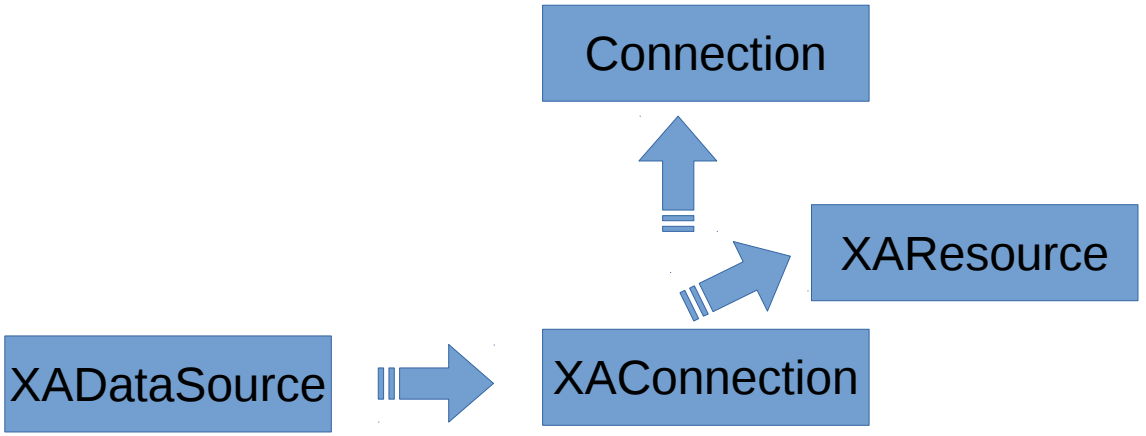

JDBC Api
Spring JPA
QueryDsl
JPA
Java Persistence API
HIBERNATE
Spring JDBC
JDBC
SQL

javax.sql.DataSource
(remark: javax.* not java.*)



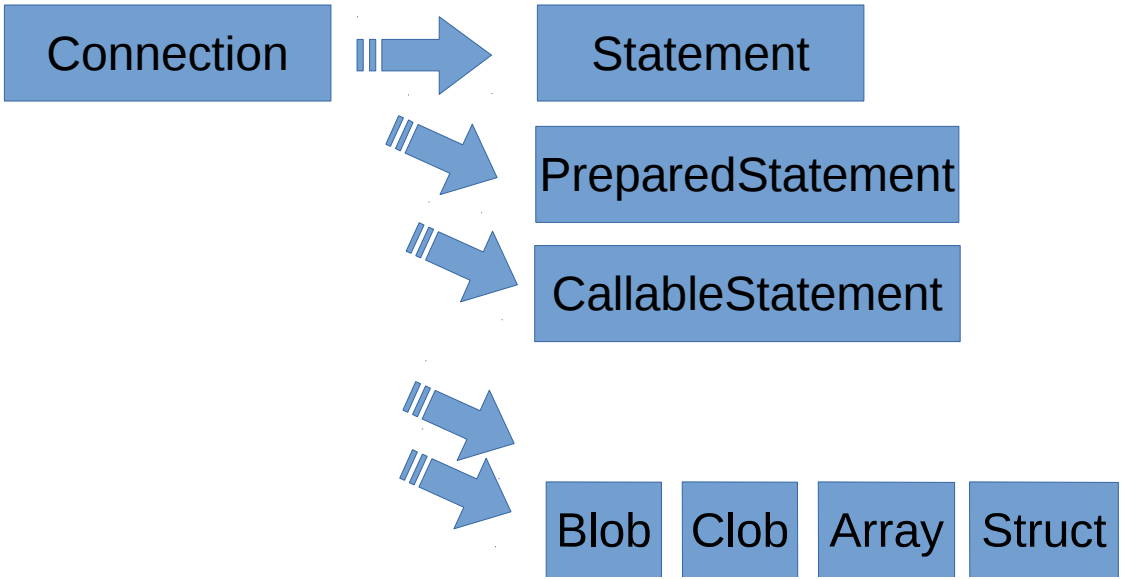

```
public interface DataSource  extends CommonDataSource, Wrapper {
    Connection getConnection() throws SQLException;
    Connection getConnection(String username, String password) throws SQLException;
}

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)

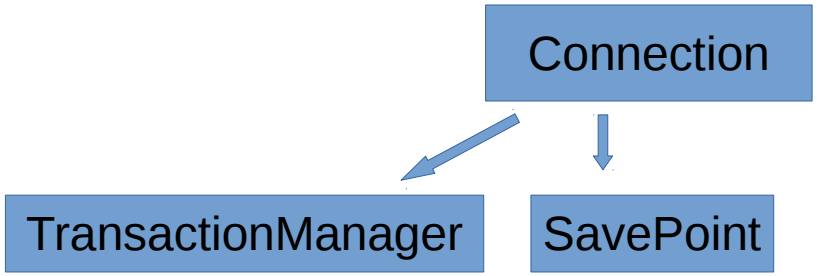



```
public interface Connection {

    void close()                                          throws SQLException;
    boolean isClosed()                                    throws SQLException;

    Statement createStatement()                           throws SQLException;
    PreparedStatement prepareStatement(String sql)        throws SQLException;
    CallableStatement prepareCall(String sql)             throws SQLException;
    // also variants with create/prepare(int resultSetType, int resultSetConcurrency)..

    Clob createClob()                                                 throws SQLException;
    Blob createBlob()                                                 throws SQLException;
    NClob createNClob()                                               throws SQLException;
    SQLXML createSQLXML()                                             throws SQLException;
    Array createArrayOf(String typeName, Object[] elements)          throws SQLException;
    Struct createStruct(String typeName, Object[] attributes)         throws SQLException;
```

create* Statement

create* Data

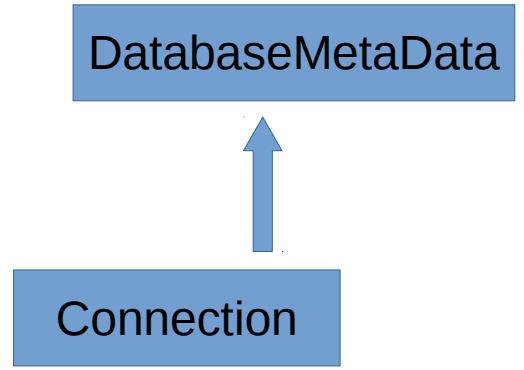
java.sql.Connection (2/3)



transaction

```
void setReadOnly(boolean readOnly)              throws SQLException;
boolean isReadOnly()                            throws SQLException;
void setAutoCommit(boolean autoCommit)          throws SQLException;
boolean getAutoCommit()                         throws SQLException;

void commit()                                   throws SQLException;
void rollback()                                 throws SQLException;
// for prepare() ... see XAConnection sub-class

Savepoint setSavepoint()                        throws SQLException;
Savepoint setSavepoint(String name)             throws SQLException;
void rollback(Savepoint savepoint)              throws SQLException;
void releaseSavepoint(Savepoint savepoint)      throws SQLException;

// TRANSACTION_NONE, TRANSACTION_READ_UNCOMMITTED, TRANSACTION_READ_COMMITTED,
// TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE
void setTransactionIsolation(int level)         throws SQLException;
int getTransactionIsolation()                   throws SQLException;
// HOLD_CURSORS_OVER_COMMIT, CLOSE_CURSORS_AT_COMMIT
void setHoldability(int holdability)            throws SQLException;
int getHoldability()                            throws SQLException;

SQLWarning getWarnings()                        throws SQLException;
void clearWarnings()                            throws SQLException;
```

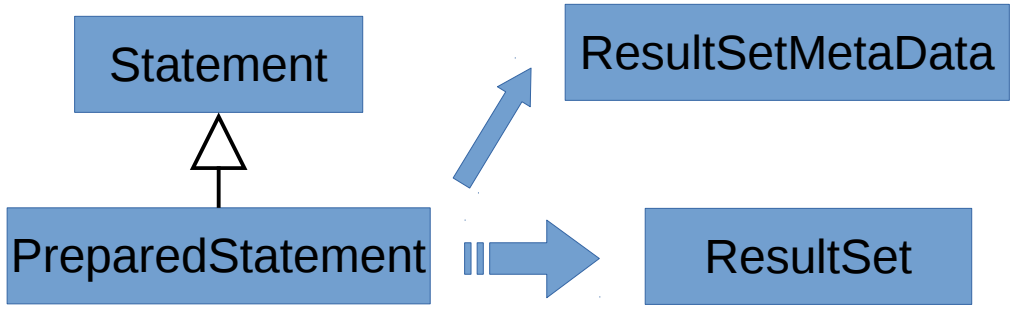
java.sql.Connection (3/3)



metadata

```
DatabaseMetaData getMetaData()                          throws SQLException;
void setCatalog(String catalog)                         throws SQLException;
String getCatalog()                                     throws SQLException;
void setSchema(String schema)                           throws SQLException;
String getSchema()                                      throws SQLException;

java.util.Map<String, Class<?>> getTypeMap()    throws SQLException;
void setTypeMap(java.util.Map<String, Class<?>> map)  throws SQLException;

void setClientInfo(String name, String value)  throws SQLClientInfoException;
void setClientInfo(Properties properties)      throws SQLClientInfoException;
String getClientInfo(String name)              throws SQLException;
Properties getClientInfo()                     throws SQLException;

String nativeSQL(String sql)                            throws SQLException;

boolean isValid(int timeout)                            throws SQLException;
void abort(Executor executor)                           throws SQLException;
void setNetworkTimeout(Executor executor, int milliseconds) throws SQLException;
int getNetworkTimeout()                                 throws SQLException;
```

java.sql.PreparedStatement (1/2)



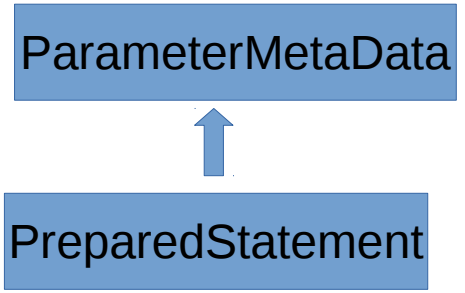
execute*

metadata

```
public interface PreparedStatement extends Statement {

    ResultSet executeQuery()                    throws SQLException;
    int executeUpdate()                         throws SQLException;

    // for batch statement
    void addBatch()                             throws SQLException;

    boolean execute()                           throws SQLException;
    default long executeLargeUpdate()           throws SQLException {

    ResultSetMetaData getMetaData()             throws SQLException;
    ParameterMetaData getParameterMetaData()    throws SQLException;
```

Set parameters (cf next)

PreparedStatement (2/2)



metadata

clear

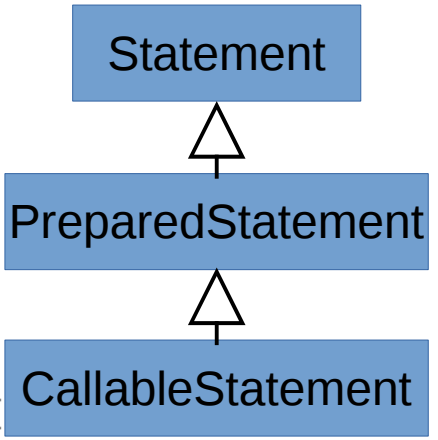
set*

```
ParameterMetaData getParameterMetaData()                    throws SQLException;

void clearParameters()                                      throws SQLException;

void setNull(int parameterIndex, int sqlType)               throws SQLException;
void setObject(int parameterIndex, Object x, int targetSqlType)
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)

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

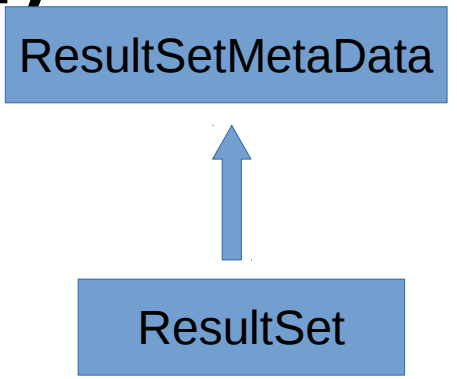

```
public interface Statement extends Wrapper, AutoCloseable {
    void close()                                    throws SQLException;
    boolean isClosed()                              throws SQLException;
    public void closeOnCompletion()                 throws SQLException;
    public boolean isCloseOnCompletion()            throws SQLException;

    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)                    th
    void registerOutParameter(int parameterIndex, int sqlType, int scale)         th
    void registerOutParameter (int parameterIndex, int sqlType, String typeName)  th
    void registerOutParameter(String parameterName, int sqlType)                  th
    void registerOutParameter(String parameterName, int sqlType, int scale)       th
    void registerOutParameter (String parameterName, int sqlType, String typeName) th

    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 SQLExce

    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)

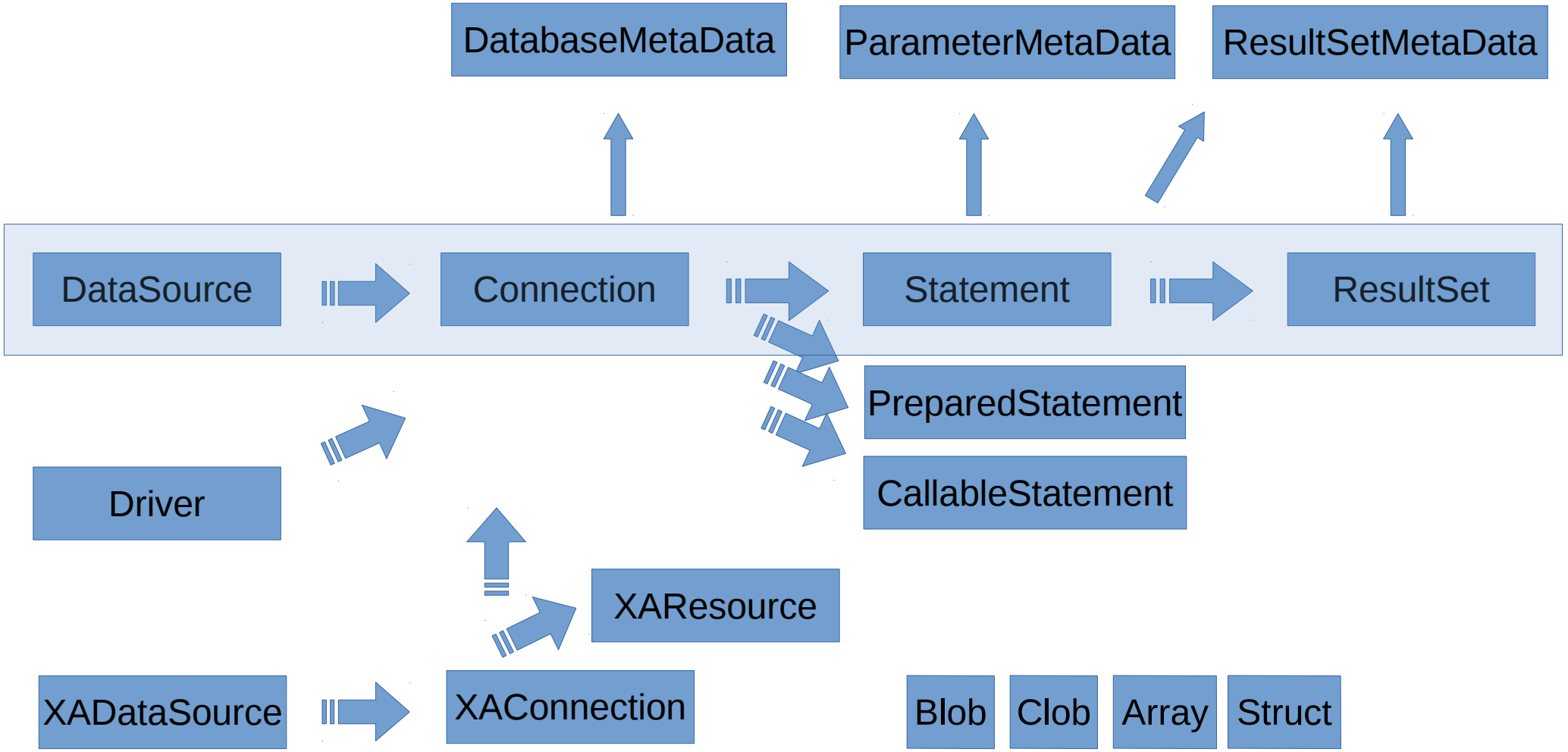



```
public interface ResultSet extends Wrapper, AutoCloseable {

    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 columnLabel)            throws SQLException;
    String getString(String columnLabel)      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 columnLabel)        throws SQLException;

    // Properties: fetchDirection, fetchSize,
    // with getter only: type, concurrency, holdabitity
```

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

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

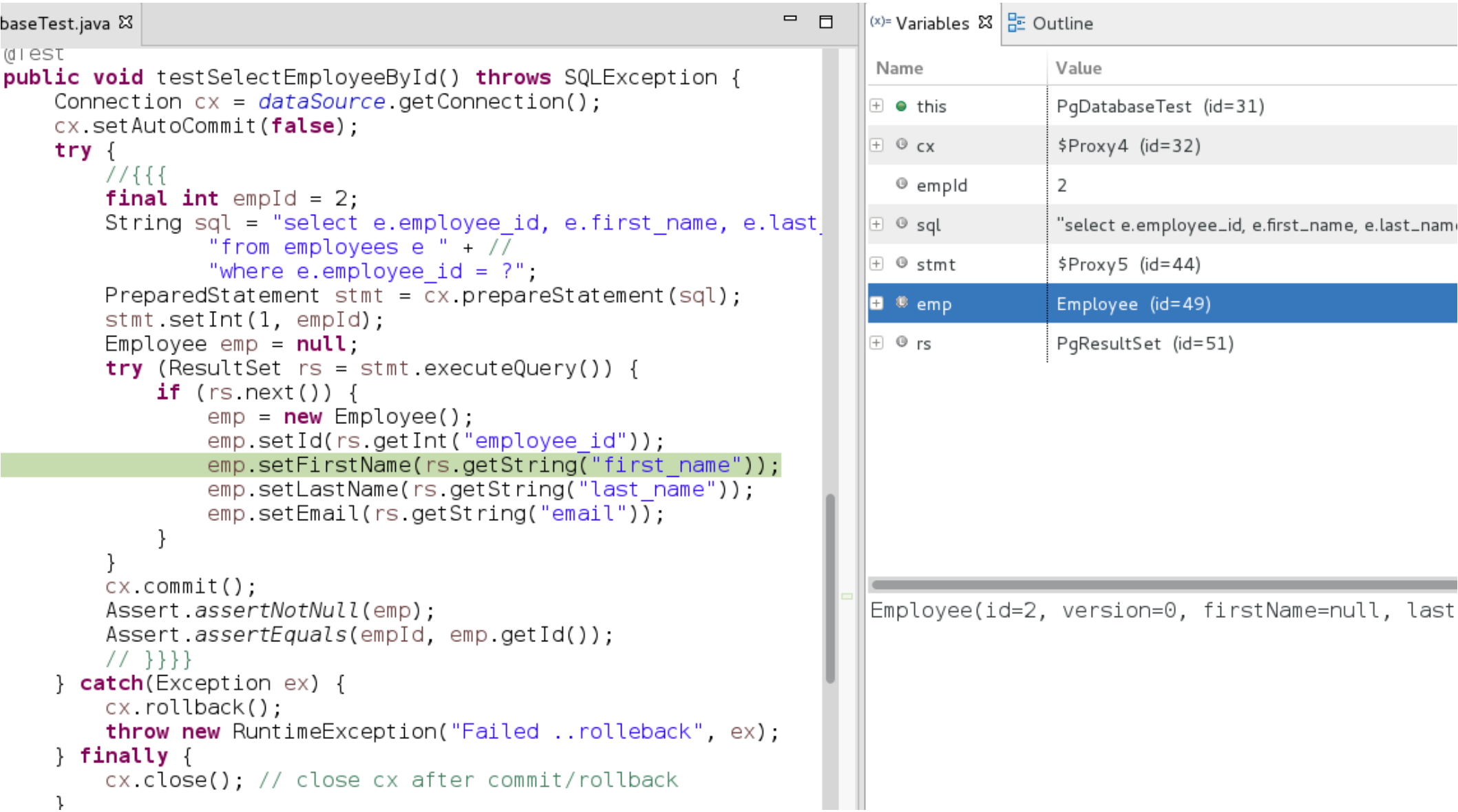
```java
import java.io.File;
import java.sql.Connection;
import java.sql.ResultSet;

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();
    }


}
```

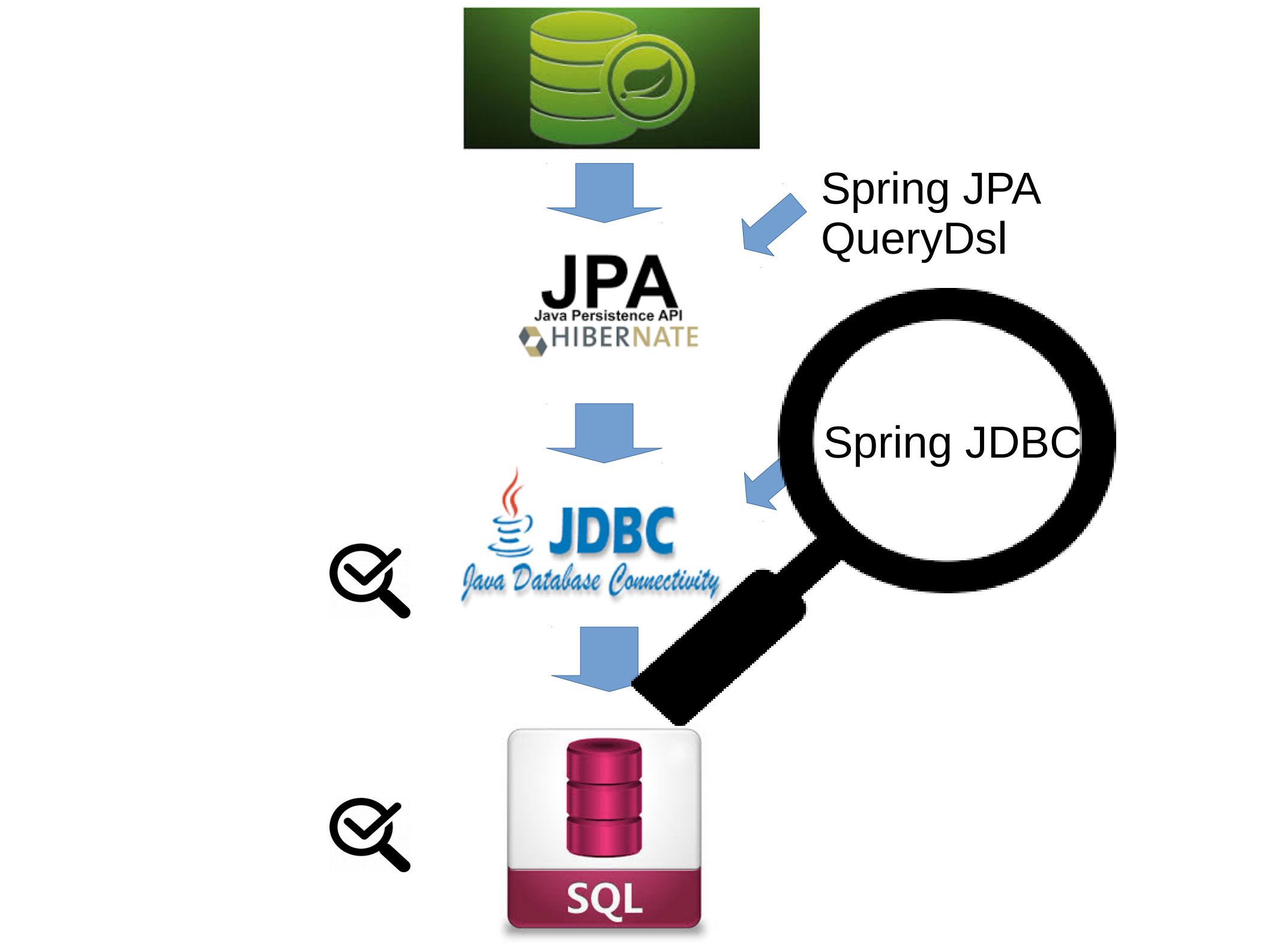
Postgresql Hello

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

```java
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
        }
    }

}
```

PreparedStatement Sample

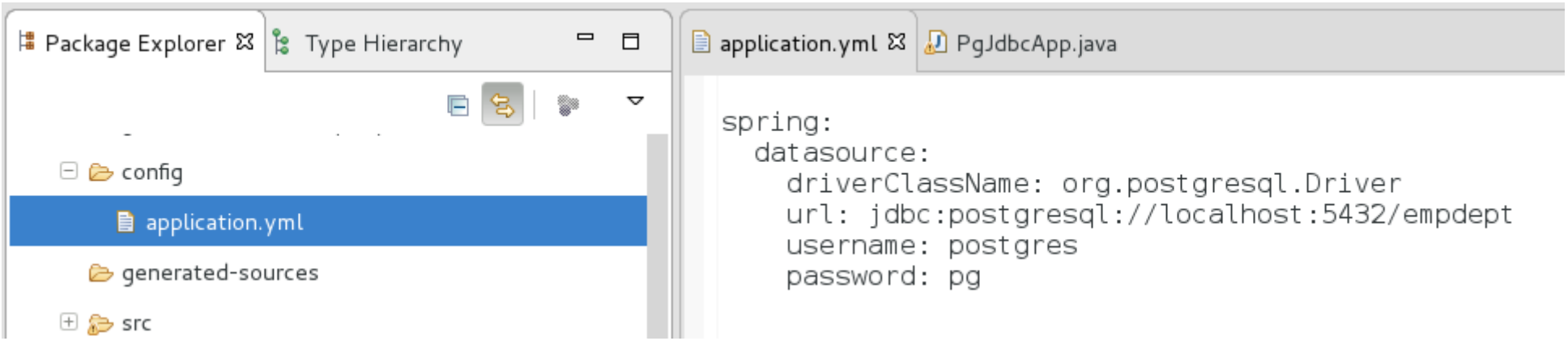



Spring JPA
QueryDsl
JPA
Java Persistence API
HIBERNATE
Spring JDBC
JDBC
Java Database Connectivity
SQL

Spring-Jdbc Database App

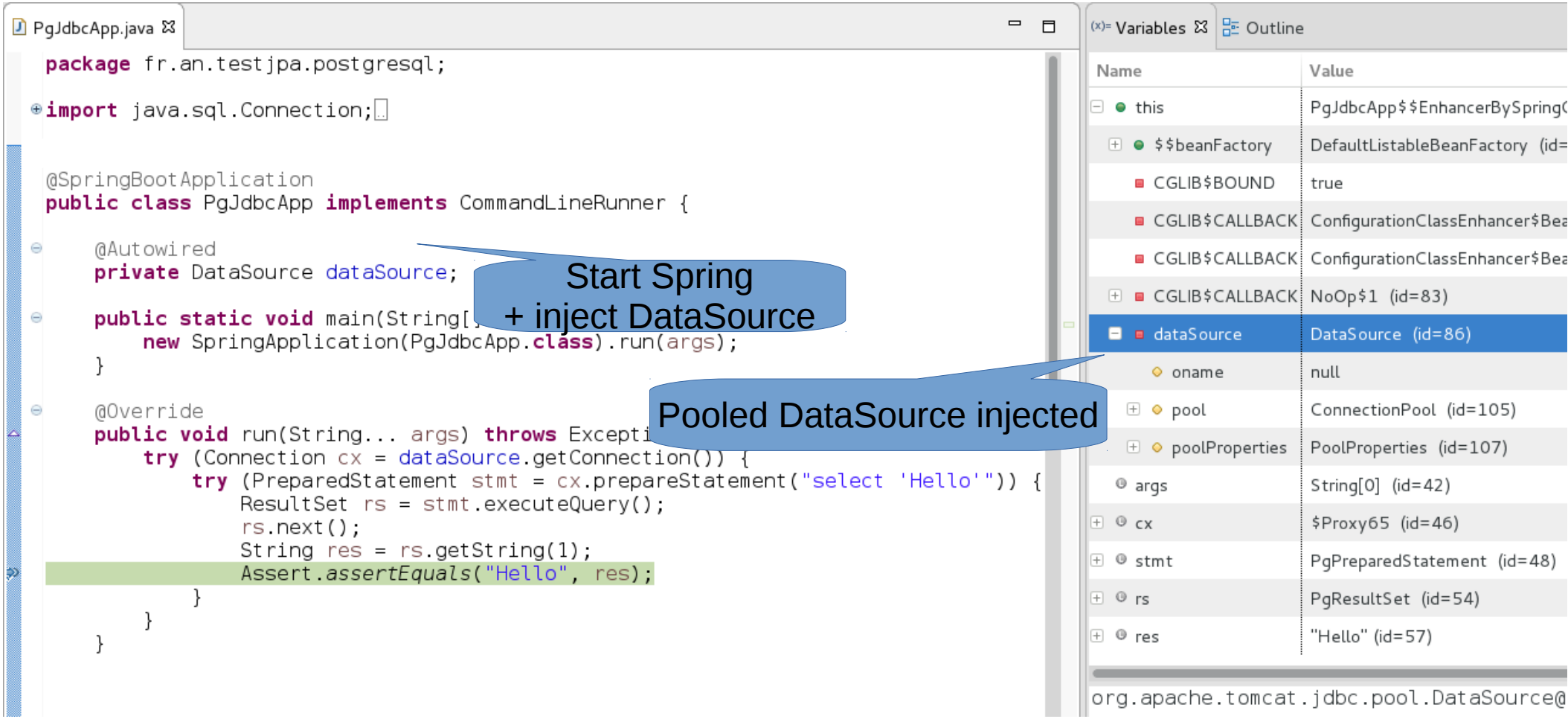
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<!-- 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

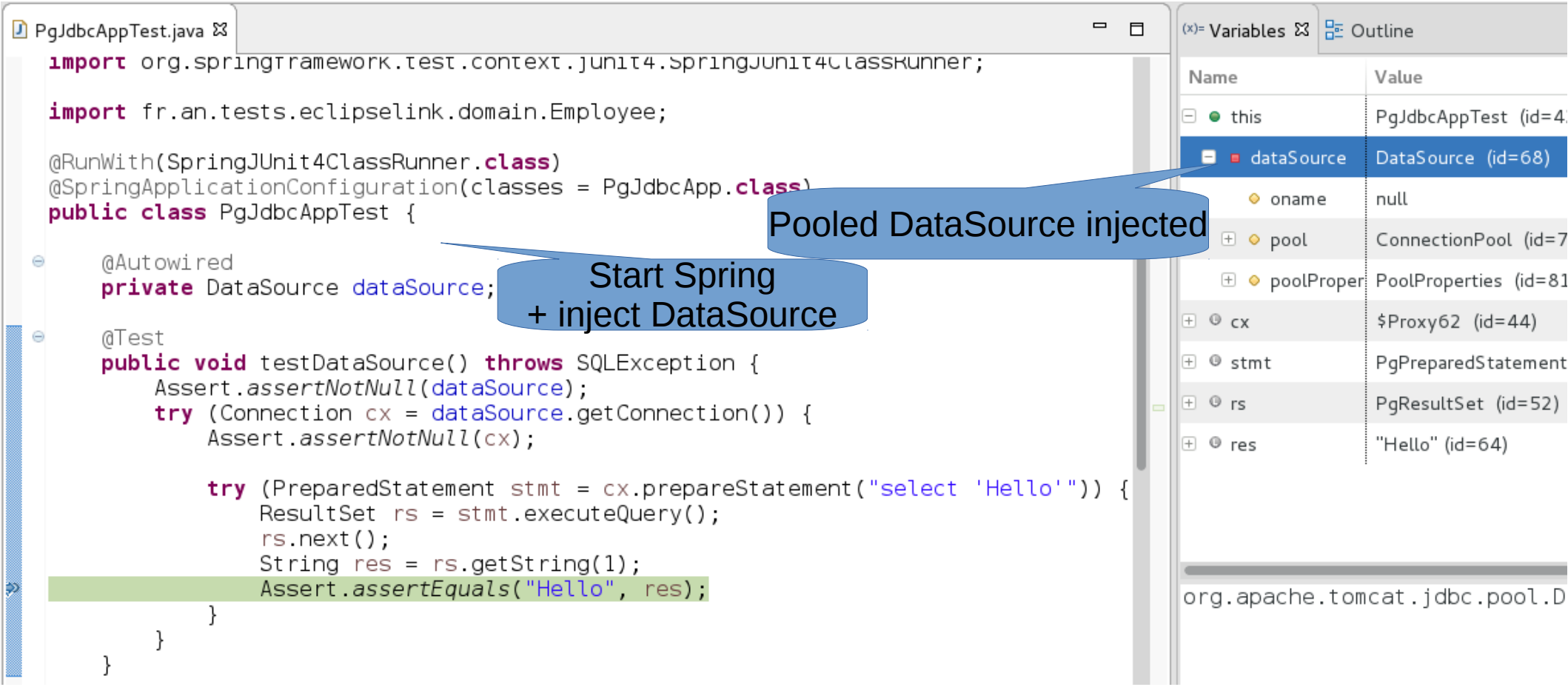


Springboot JDBC… “JUST work”

Springboot Jdbc main()



Springboot JUnit



SpringBoot … explicit DataSourceAutoConfiguration

```java
@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);
            }
        }
    }
```

Explicit @Bean (example for Multi DataSources)

```java
@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 **

}
```

2n execution … different connection

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

Same using Template Spring-Jdbc

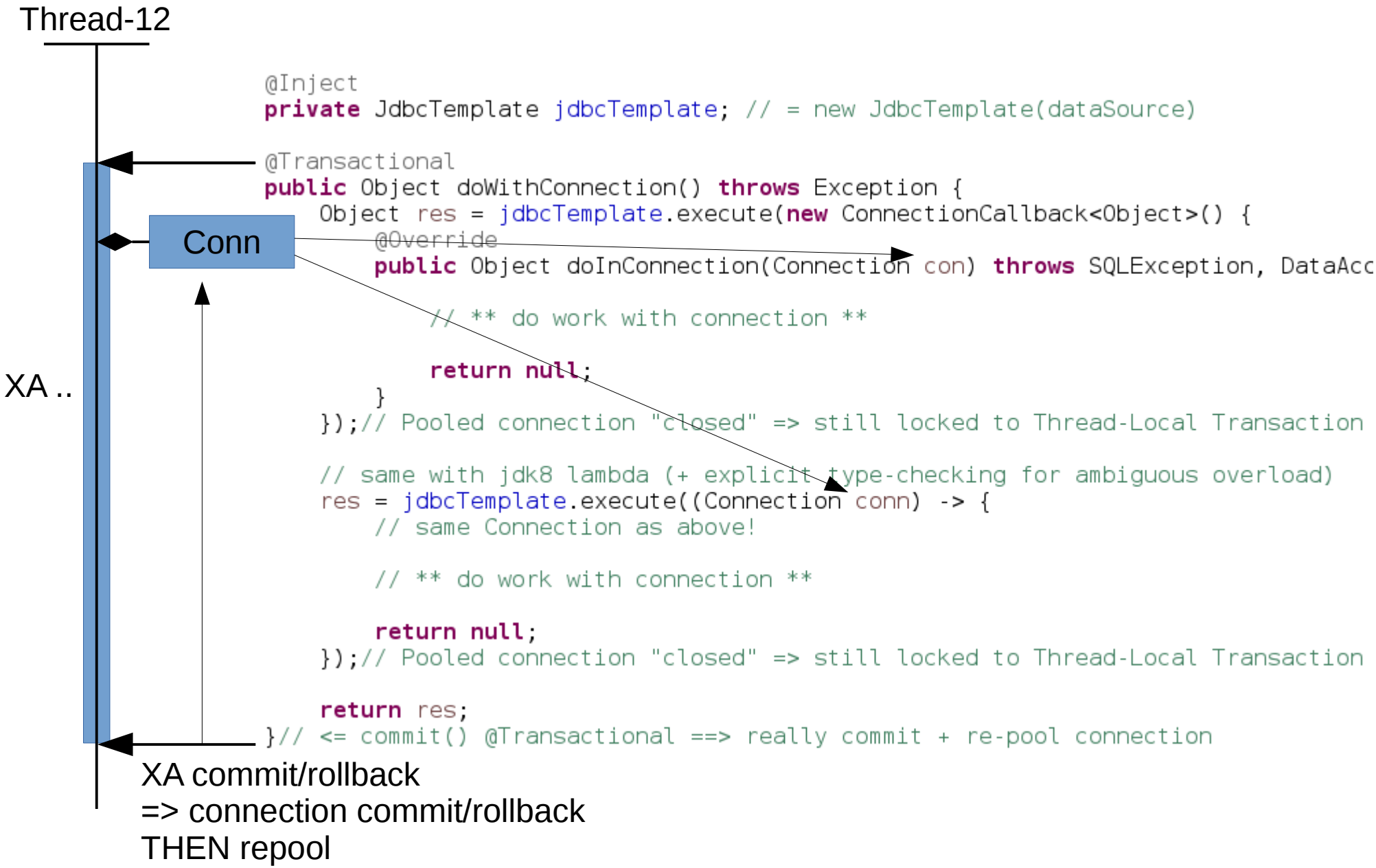
```
@Inject
private JdbcTemplate jdbcTemplate; // = new JdbcTemplate(dataSource)

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

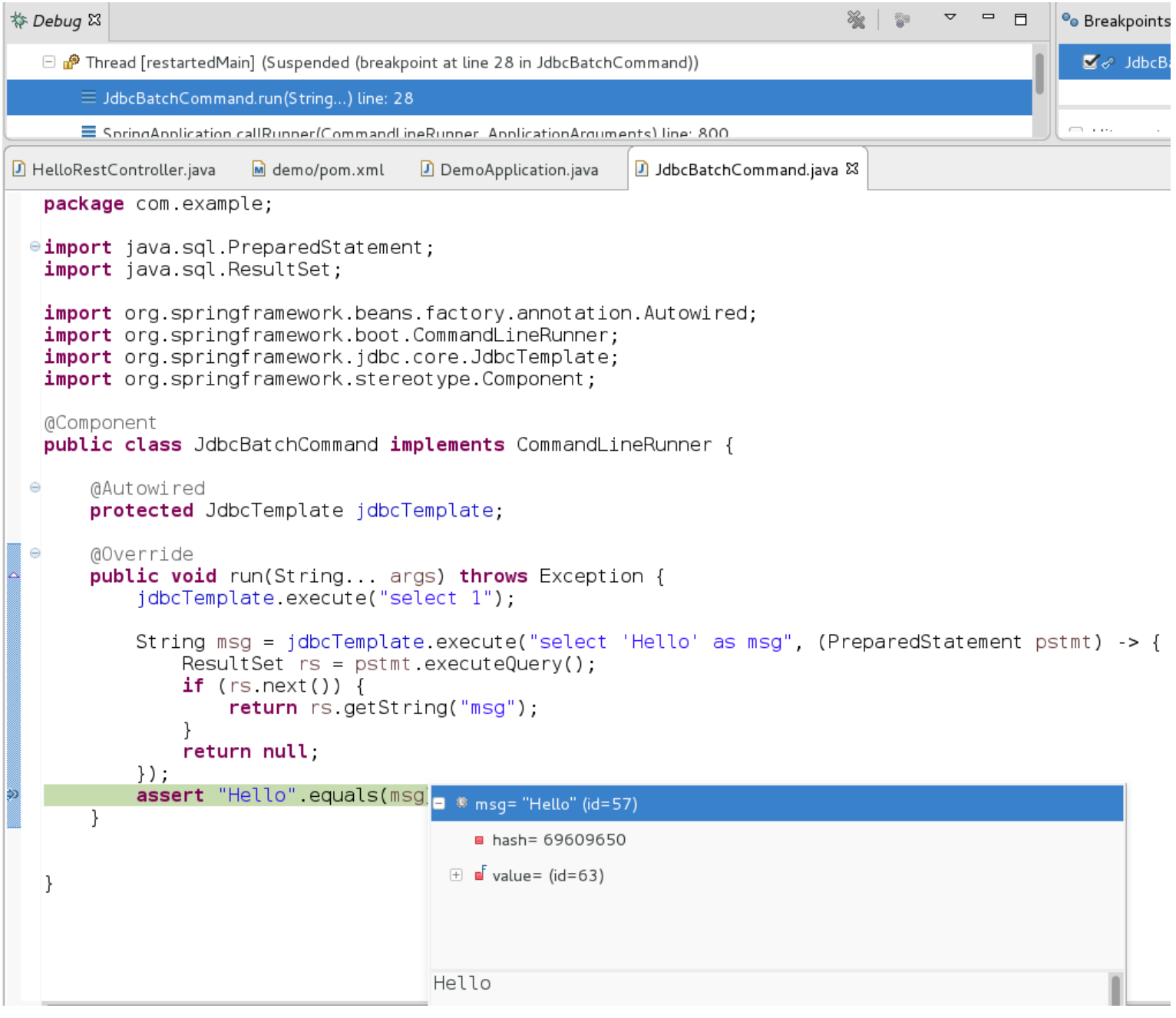
(Reminder) try-finally Connection + try-finally Statement

```
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
}
```

JdbcTemplate(dataSource)

```java
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);

    }
```

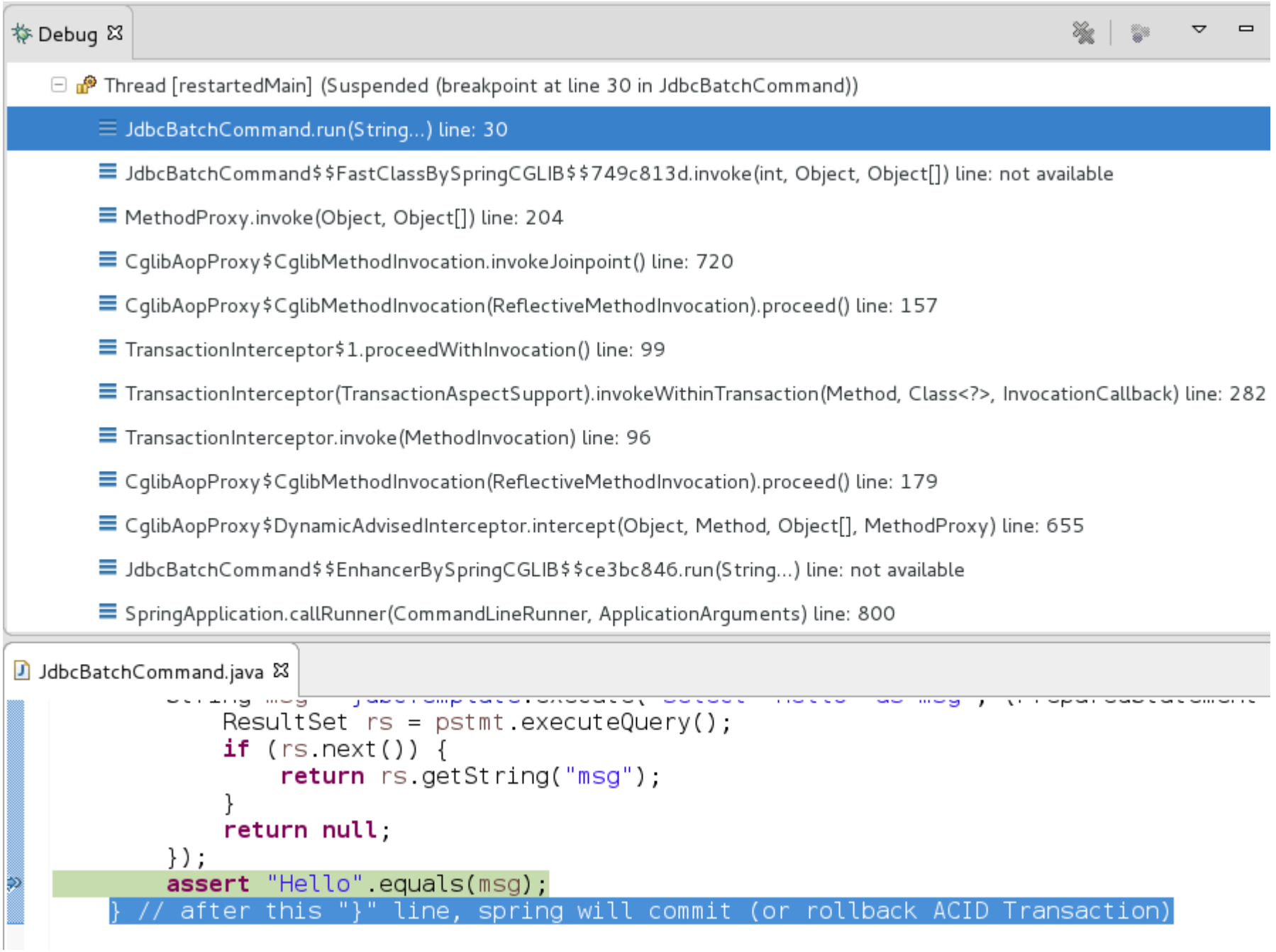
Run Jdbc App

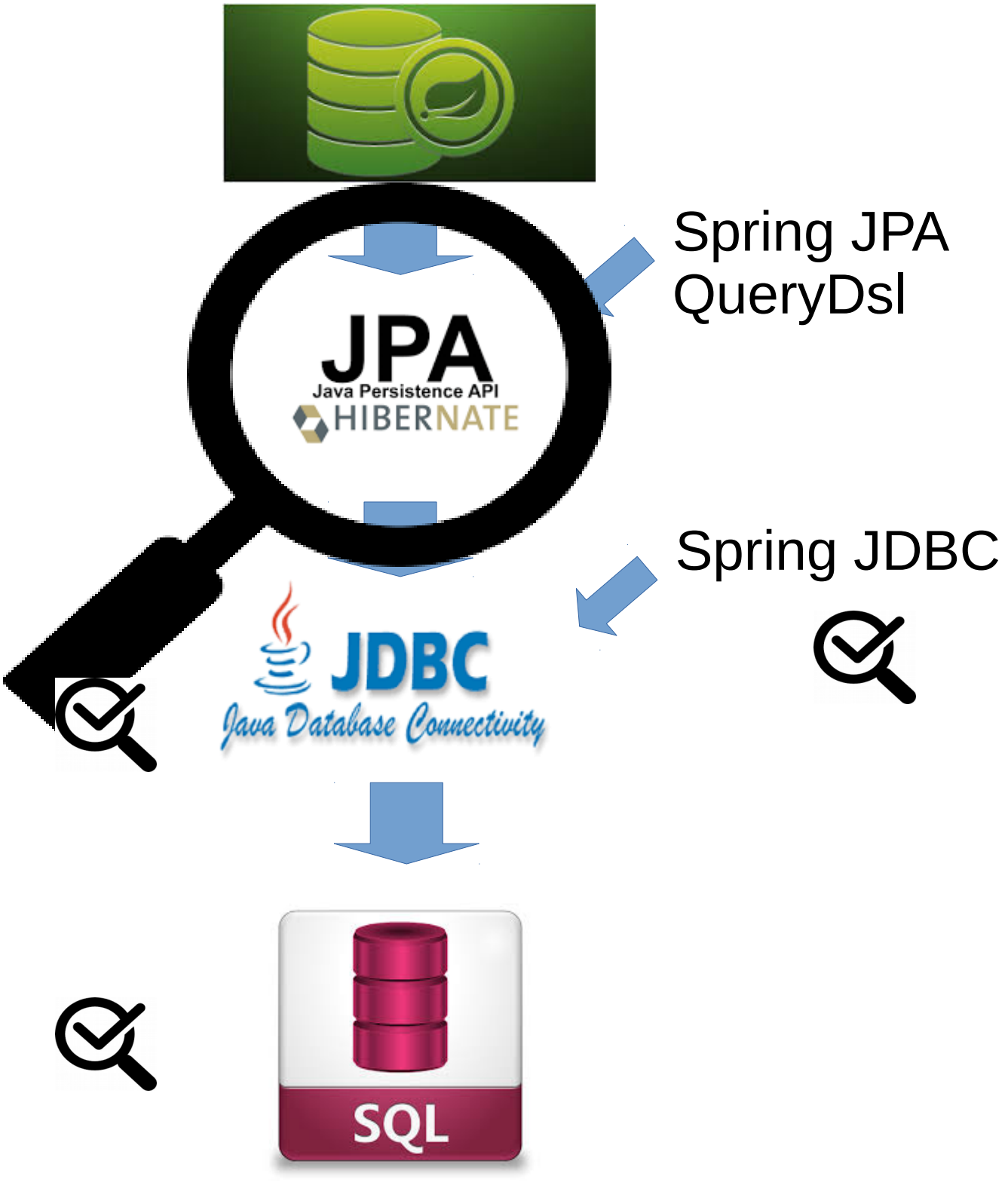

Debug
Thread [restartedMain] (Suspended (breakpoint at line 28 in JdbcBatchCommand))
JdbcBatchCommand.run(String...) line: 28
Breakpoints
HelloRestController.java
demo/pom.xml
DemoApplication.java
JdbcBatchCommand.java
package com.example;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
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;
});
assert "Hello".equals(msg
}
}
msg= "Hello" (id=57)
hash= 69609650
value= (id=63)
Hello

Springboot built-in supports JTA @Transactional

```java
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




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 (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 DTO<->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

And an @Id

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

@Entity
public class Simple {

    @Id
    private int id;

    public Simple() {
    }

}
```

Javax.persistence.* Annotations

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

```java
@Documented
@Target(TYPE)
@Retention(RUNTIME)
public @interface Entity {

    String name() default "";

}
```

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

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

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

```java
@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 @PrimaryKeyJoinColumns
@IdClass @QueryHint @ElementCollection @Temporal @PersistenceContext @ManyToOne @Table @Convert @Transient
@NamedSubgraph @MapKeyEnumerated @PersistenceContexts @MapsId @Basic @SqlResultSetMappings @AssociationOverri
@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 @UniqueConstra
@JoinColumns @AttributeOverrides @ColumnResult @Converts @NamedEntityGraph @Index @TableGenerator
@StaticMetamodel @ManyToMany @AttributeOverride @PostUpdate @NamedStoredProcedureQuery @MapKeyTemporal
@SqlResultSetMapping @PostLoad @MapKeyJoinColumn @ForeignKey @Column @Embedded @EmbeddedId @NamedNativeQuerie
@ExcludeDefaultListeners @NamedStoredProcedureQueries @ConstructorResult @OneToMany @MapKeyJoinColumns
```

Sufficient @Annotations to know?

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

Sub-Classes
- @Inheritance
- @DiscriminatorColumn

For relationships
FK (*) → (1) PK
PK (*) ← (*) 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;
```

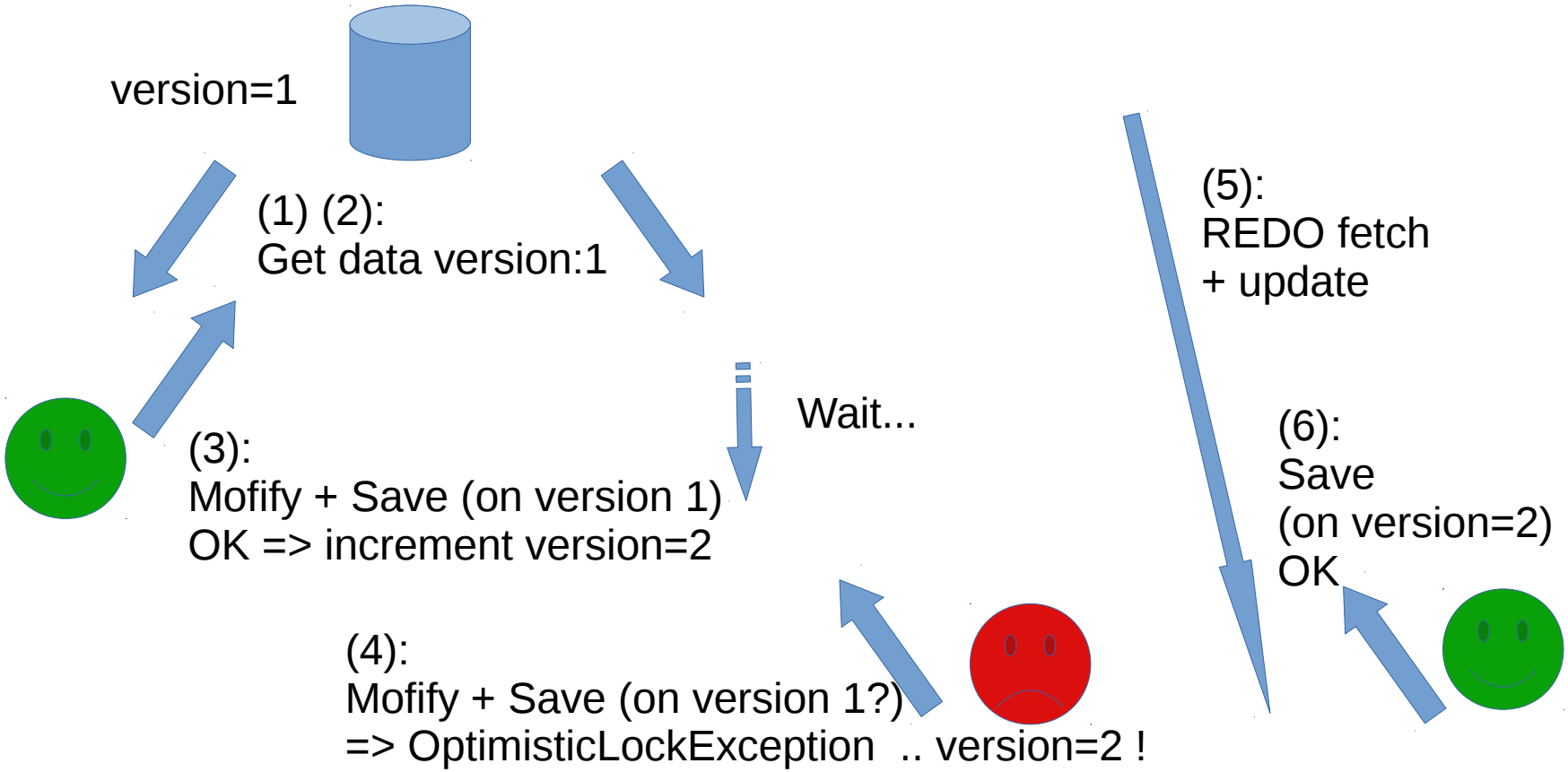
SQL:
CREATE SEQUENCE employees_seq INCREMENT BY 10;

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)

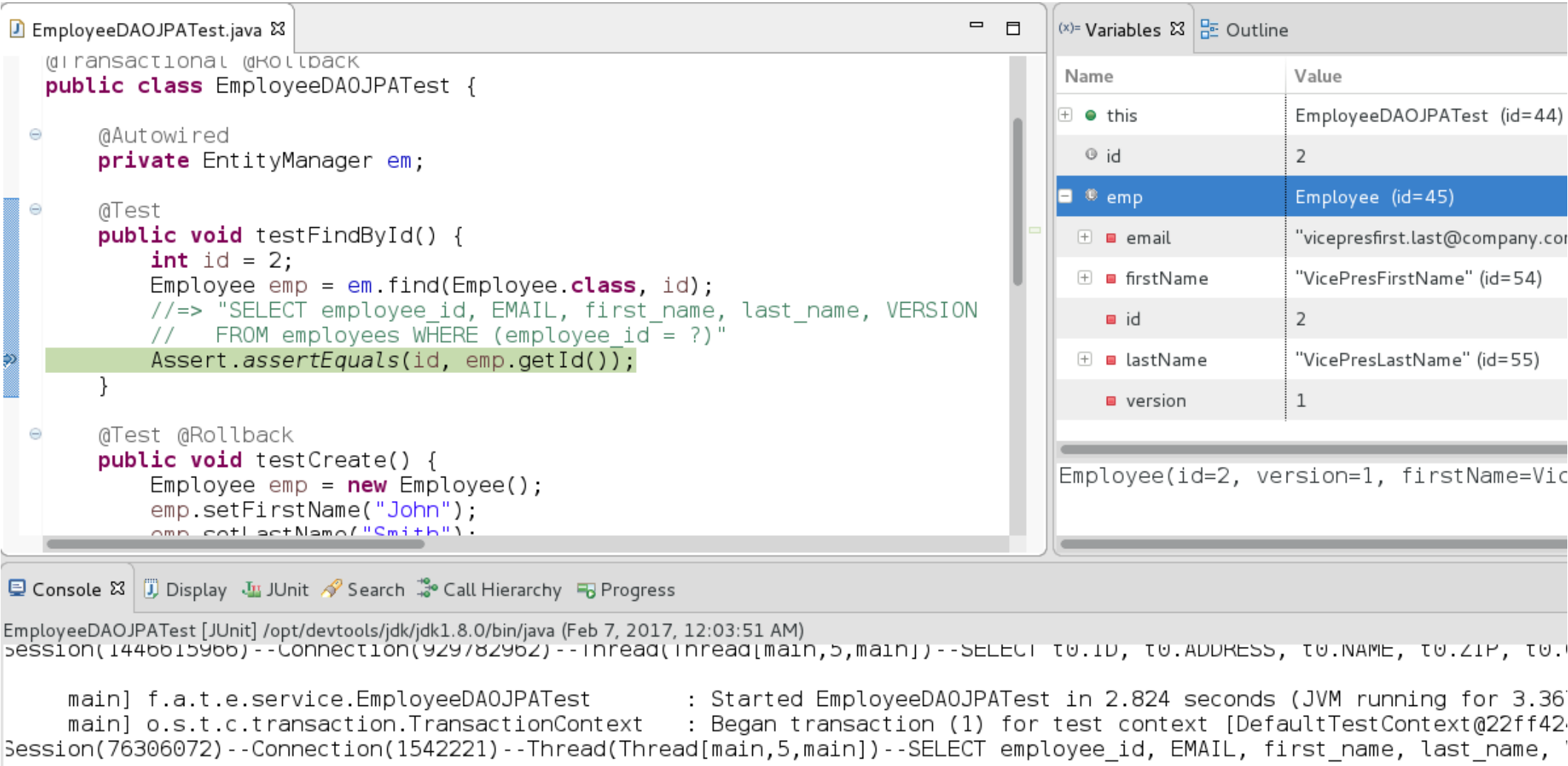
```java
@Data // lombok getter, setter
@Entity
public class Department {

    @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)

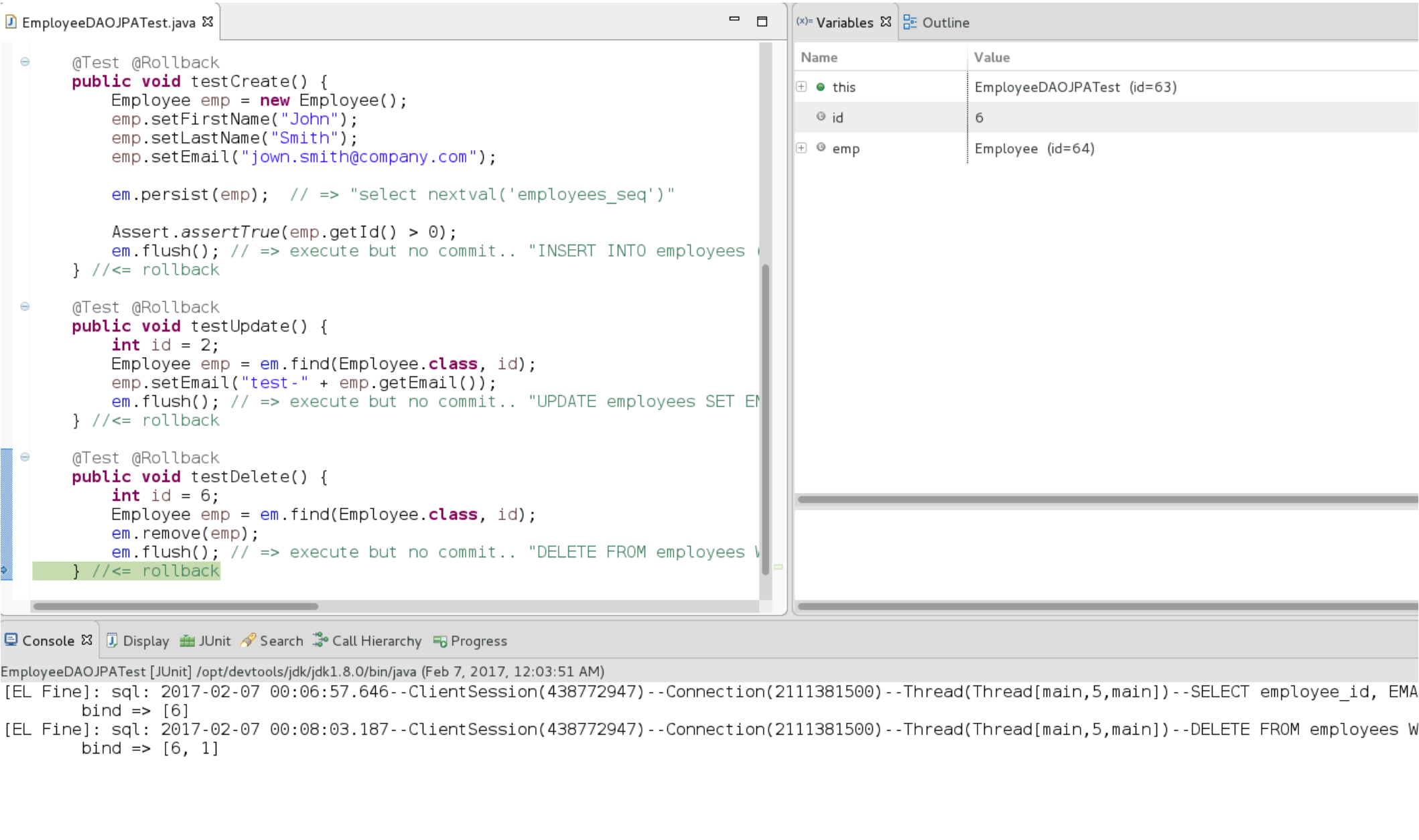
For real with @Column ...

```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;
}
```

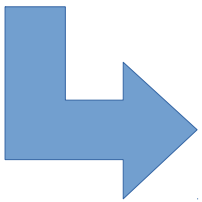
Dynamic Query …
String type checking?

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;
}
```



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;
    }
}
```

Real MetaModel ..
(Real? ... see javac processor)

```java
import javax.annotation.Generated;
import javax.persistence.metamodel.SetAttribute;
import javax.persistence.metamodel.SingularAttribute;
import javax.persistence.metamodel.StaticMetamodel;

@Generated(value = "org.hibernate.jpamodelgen.JPAMetaModelEntityProcessor")
@StaticMetamodel(Hotel.class)
public abstract class Hotel_ {

    public static volatile SingularAttribute<Hotel, String> zip;
    public static volatile SingularAttribute<Hotel, String> address;
    public static volatile SetAttribute<Hotel, Review> reviews;
    public static volatile SingularAttribute<Hotel, City> city;
    public static volatile SingularAttribute<Hotel, String> name;
    public static volatile SingularAttribute<Hotel, Long> id;

}
```

Pom.xml MetaModel plugin (example for eclipselink)

```
<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>
```

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;
}
```

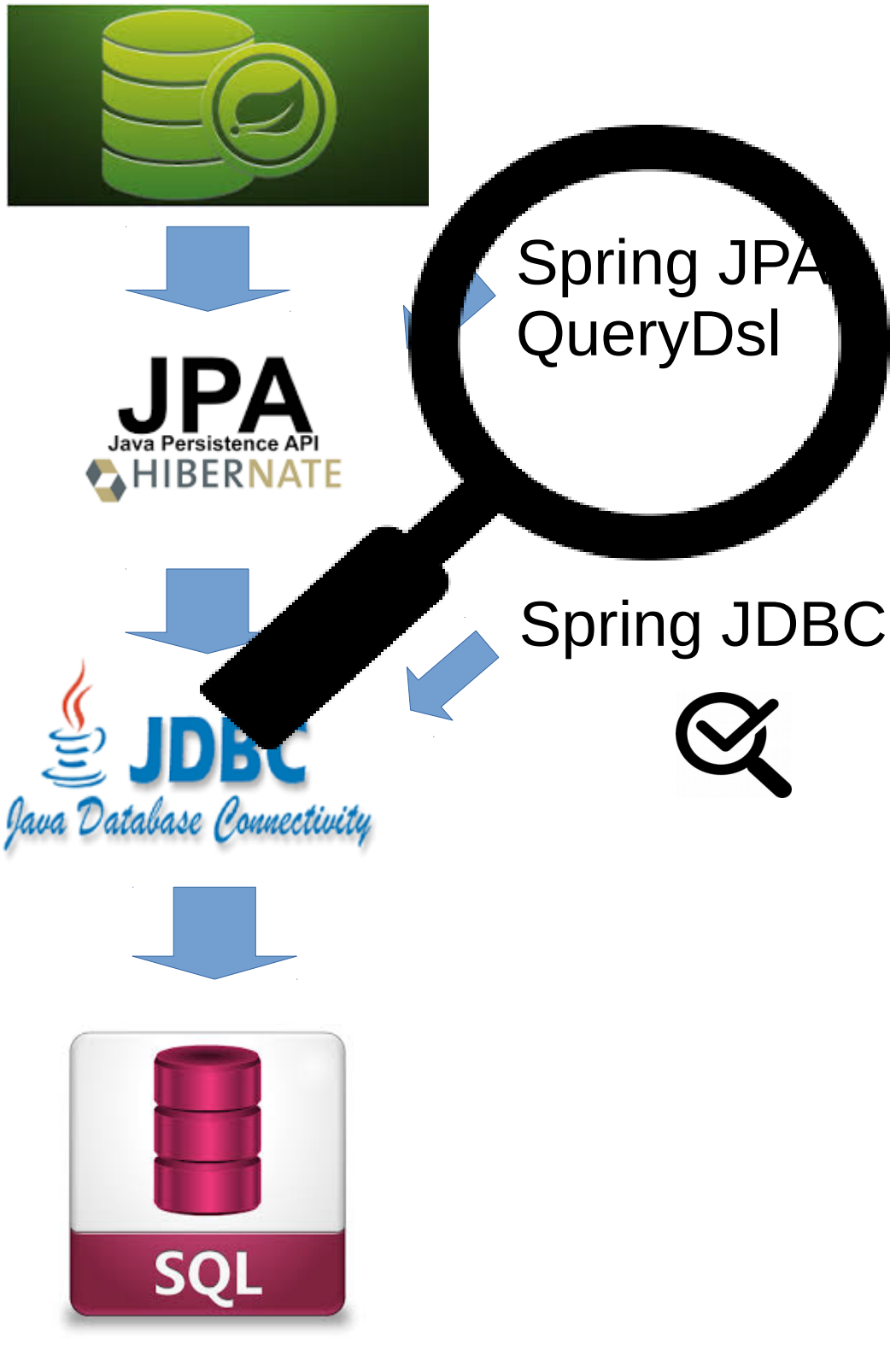
Search Parameters in Criteria class (also called “Specification”)

```
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



```
<dependency>
    <groupId>com.querydsl</groupId>
    <artifactId>querydsl-jpa</artifactId>
    <version>${querydsl.version}</version>
</dependency>
<!-- implicit
<dependency>
    <groupId>com.querydsl</groupId>
    <artifactId>querydsl-core</artifactId>
    <version>${querydsl.version}</version>
</dependency>
-->

<!-- can also perform query in-memory on List<T> -->
<dependency>
    <groupId>com.querydsl</groupId>
    <artifactId>querydsl-collections</artifactId>
    <version>${querydsl.version}</version>
</dependency>
```

Pom.xml QueryDsl plugin

```
<plugin>
    <groupId>com.mysema.maven</groupId>
    <artifactId>apt-maven-plugin</artifactId>
    <version>1.1.3</version>
    <executions>
        <execution>
            <id>querydsl</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>process</goal>
            </goals>
            <configuration>
                <outputDirectory>${basedir}/generated-sources/querydsl</outputDirectory>
                <processor>com.querydsl.apt.jpa.JPAAnnotationProcessor</processor>
            </configuration>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>com.querydsl</groupId>
            <artifactId>querydsl-apt</artifactId>
            <version>${querydsl.version}</version>
        </dependency>
    </dependencies>
</plugin>
```

QueryDsl Generated Q* class similar but richer than *_ class

```
/**
 * QHotel is a Querydsl query type for Hotel
 */
@Generated("com.querydsl.codegen.EntitySerializer")
public class QHotel extends EntityPathBase<Hotel> {

    private static final long serialVersionUID = -1929614749L;

    private static final PathInits INITS = PathInits.DIRECT2;

    public static final QHotel hotel = new QHotel("hotel");

    public final StringPath address = createString("address");

    public final QCity city;

    public final NumberPath<Long> id = createNumber("id", Long.class);

    public final StringPath name = createString("name");

    public final SetPath<Review, QReview> reviews = this.<Review, QReview>cr

    public final StringPath zip = createString("zip");

    public QHotel(String variable) {
        this(Hotel.class, forVariable(variable), INITS);
    }

    public QHotel(Path<? extends Hotel> path) {
        this(path.getType(), path.getMetadata(), PathInits.getFor(path.getMe
```

QueryDsl
++More Fluent than JPA

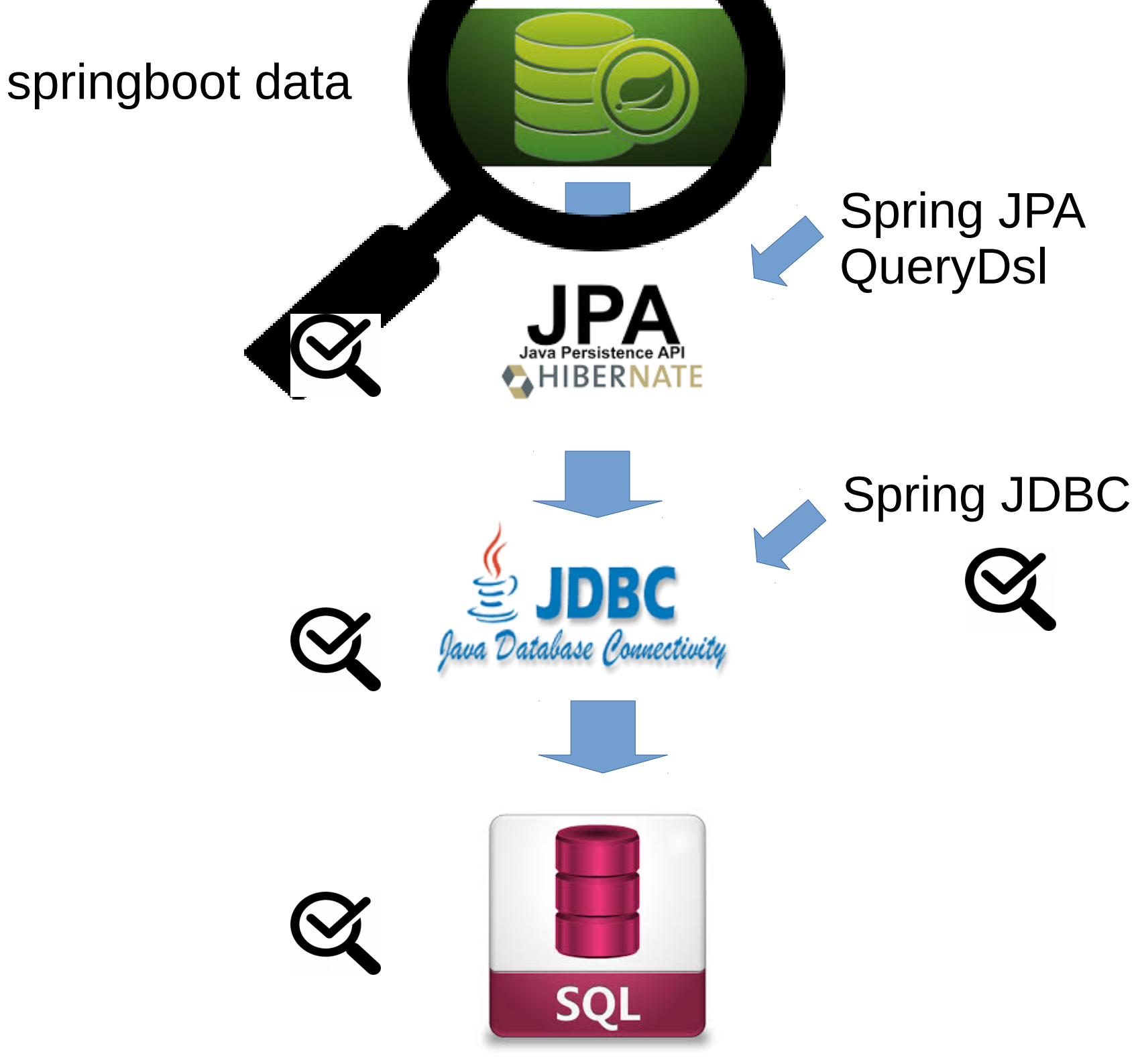
```
public List<Hotel> findByQueryDsl(HotelSpecification spec) {
    JPAQuery<Hotel> query = new JPAQuery<>(em);
    JPAQuery<Hotel> q = query.select(QHotel.hotel).from(QHotel.hotel);

    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;
}
```

QueryDsl + Bind-Variables ! (not a clean API for parameters!!)

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

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

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

public static Param<String> param(QueryMetadata qm, String value) {
    Param<String> param = new Param<>(String.class);
    qm.setParam(param, value);
    return param;
}
```

QueryDsl Predicate (no bind-var!)

```
public com.querydsl.core.types.Predicate specificationToQueryDslPredicate(HotelSpecification spec) {
    BooleanBuilder res = new BooleanBuilder();

    if (spec.nameLike != null) {
        res.and(QHotel.hotel.name.like(spec.nameLike)); // NO bind-var!
    }
    if (spec.addressLike != null) {
        res.and(QHotel.hotel.address.like(spec.addressLike)); // NO bind-var!
    }
    if (spec.cityNameLike != null) {
        res.and(QHotel.hotel.city.name.like(spec.cityNameLike)); // NO bind-var!
    }
    return res.getValue();
}

public List<Hotel> findByQueryDslPredicate(com.querydsl.core.types.Predicate predicate) {
    JPAQuery<Hotel> query = new JPAQuery<>(em);
    JPAQuery<Hotel> q = query.select(QHotel.hotel).from(QHotel.hotel);
    q.where(predicate);
    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
}
```

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

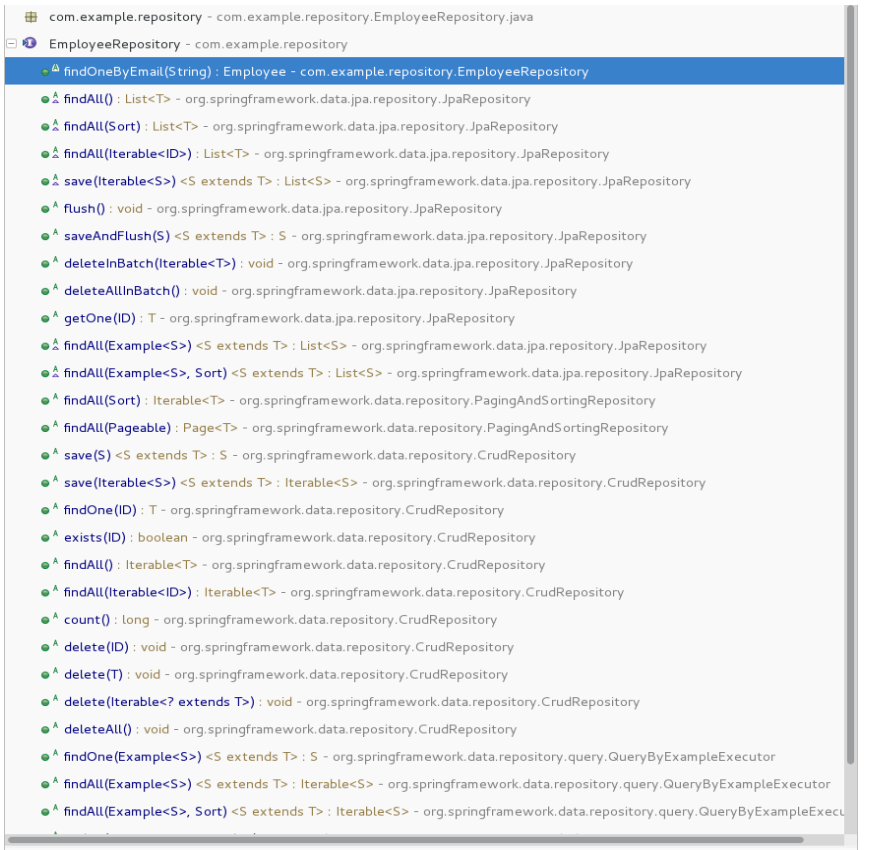
```
interface CityRepository extends JpaRepository<City, Long> {

    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)

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 @JPA configuration

```java
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 !!!

```java
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_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 ()” ???

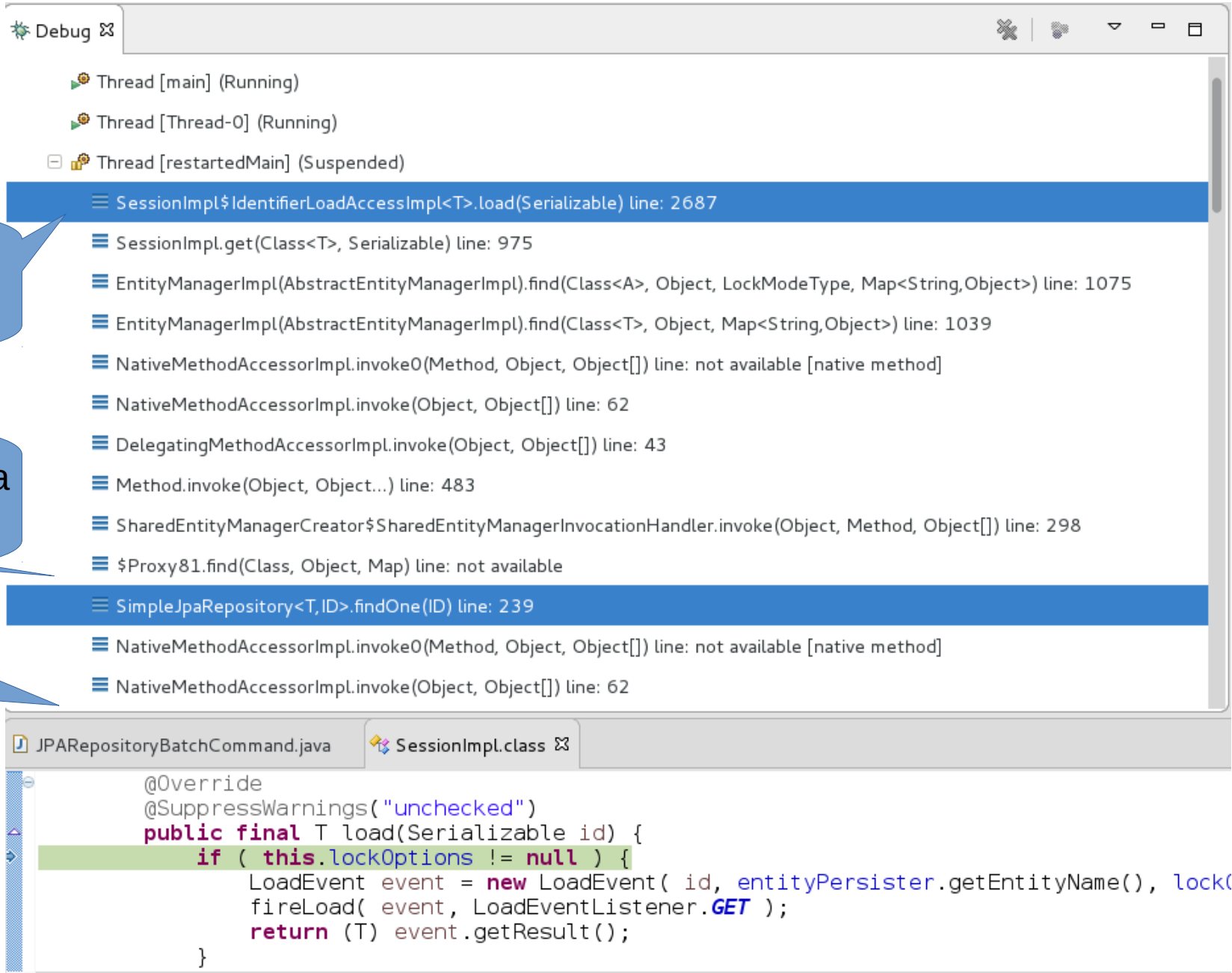
Tables are created/updated at startup

```
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

How??



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

Call springboot-data JpaRepository

You code calls a generated Proxy..

JPA Dyn Criteria + Spring-Data = Specification (!= querydsl Predicate !)

```java
package org.springframework.data.jpa.domain;

import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;

/**
 * Specification in the sense of Domain Driven Design.
 *
 * @author Oliver Gierke
 * @author Thomas Darimont
 */
public interface Specification<T> {

    /**
     * Creates a WHERE clause for a query of the referenced entity in form of a {@link Predicate} for
     * {@link Root} and {@link CriteriaQuery}.
     *
     * @param root
     * @param query
     * @return a {@link Predicate}, must not be {@literal null}.
     */
    Predicate toPredicate(Root<T> root, CriteriaQuery<?> query, CriteriaBuilder cb);
}
```

Spring-Data + Specification..

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

    @Override Hotel findOne(Specification<Hotel> spec);
    @Override List<Hotel> findAll(Specification<Hotel> spec);
    @Override Page<Hotel> findAll(Specification<Hotel> spec, Pageable pageable);
    @Override List<Hotel> findAll(Specification<Hotel> spec, Sort sort);
    @Override long count(Specification<Hotel> spec);
}
```

Sample spring-data Specification

```
public static class HotelJPASpecification implements Specification<Hotel> {

    public String nameLike;
    public String addressLike;
    public String cityNameLike;
    // setters..

    @Override
    public Predicate toPredicate(Root<Hotel> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
        List<Predicate> predicates = new ArrayList<>();

        if (nameLike != null) {
            predicates.add(cb.like(root.get("name"), nameLike)); // NO bind-var !!
        }
        if (addressLike != null) {
            predicates.add(cb.like(root.get("address"), addressLike)); // NO bind-var !!
        }
        if (cityNameLike != null) {
            predicates.add(cb.like(root.get("city").get("name"), cityNameLike));  // NO bind-var !!
        }
        Predicate andPredicates = cb.and(predicates.toArray(new Predicate[predicates.size()]));
        return andPredicates;
    }

}
```

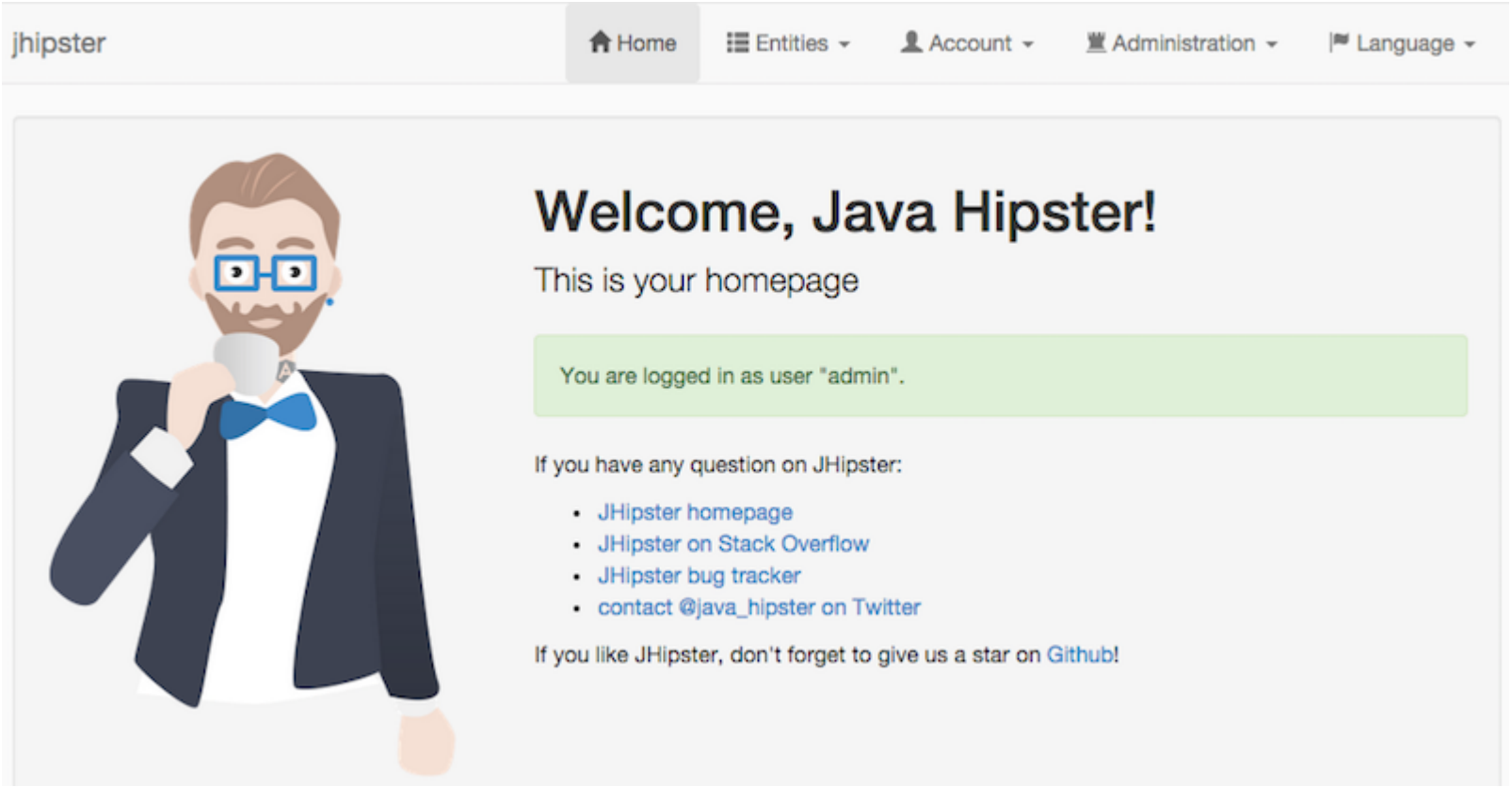
Better.. QueryDsl + Spring-Data

```
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


ENTITY
ORM
SPRINGFRAMEWORK
JPA
HIBERNATE
MANYTOONE
CRITERIABUILDER
ENTITYMANAGER
DATABASE
JAVA
JDBC
SELECT
PREPAREDSTATEMENT
QUERYDSL
SPRINGBOOT
ONETOMANY
CRUD
DATABASEENTITY
SPRING-DATA
@ID
@ENTITY
DELETE
INSERT
UPDATE
DYNAMICQUERY
@SEQUENCEGENERATOR
ORACLE

Conclusion

Only a (not so short) introduction to
Databases < JDBC < JPA < Spring-Data

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