Database migrations

the missing piece in Java EE

BOF3528

Rikard Thulin

This is a short version of this talk

Full length video available at:
http://vimeo.com/105880153

squeed
Rikard Thulin

We will discuss how

schema migrations

improves the software engineering life cycle and the

possibilities to improve Java for the future

Roadmap

- Introduction to the topic and why it is so important (7 min)
- Schema Migrations demo (8 min)
- The current situation, the tools of the trade (5 min)
- Future possibilities at hand to solve this problem (5 min)
- Discussion (15 min)

About me

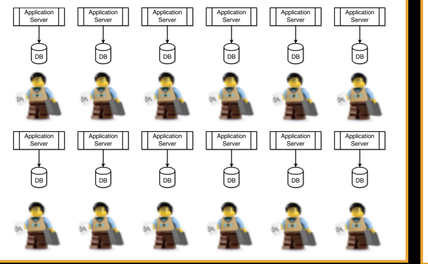
squeed

Part 1

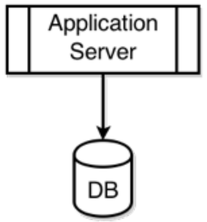
Introduction to the topic

and why it is so important

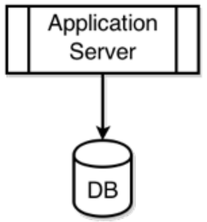
Dev team



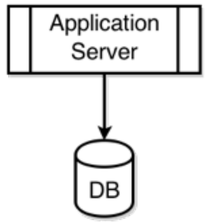
CI



QA



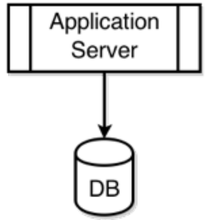
Staging cluster



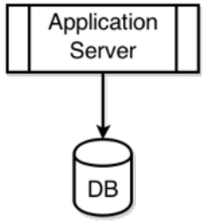
How do we keep

everything CONSISTENT?

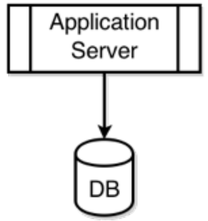
Staging cluster



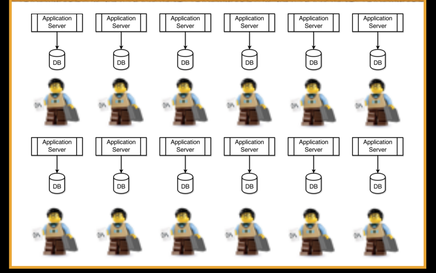
Prod cluster



Prod cluster



Dev team



push

VCS

{code}

pull

Build server

publish

How do we keep
{ code }
CONSISTENT?

promote

Prod

Staging

QA

DEV/CI

release

Repository

artifacts

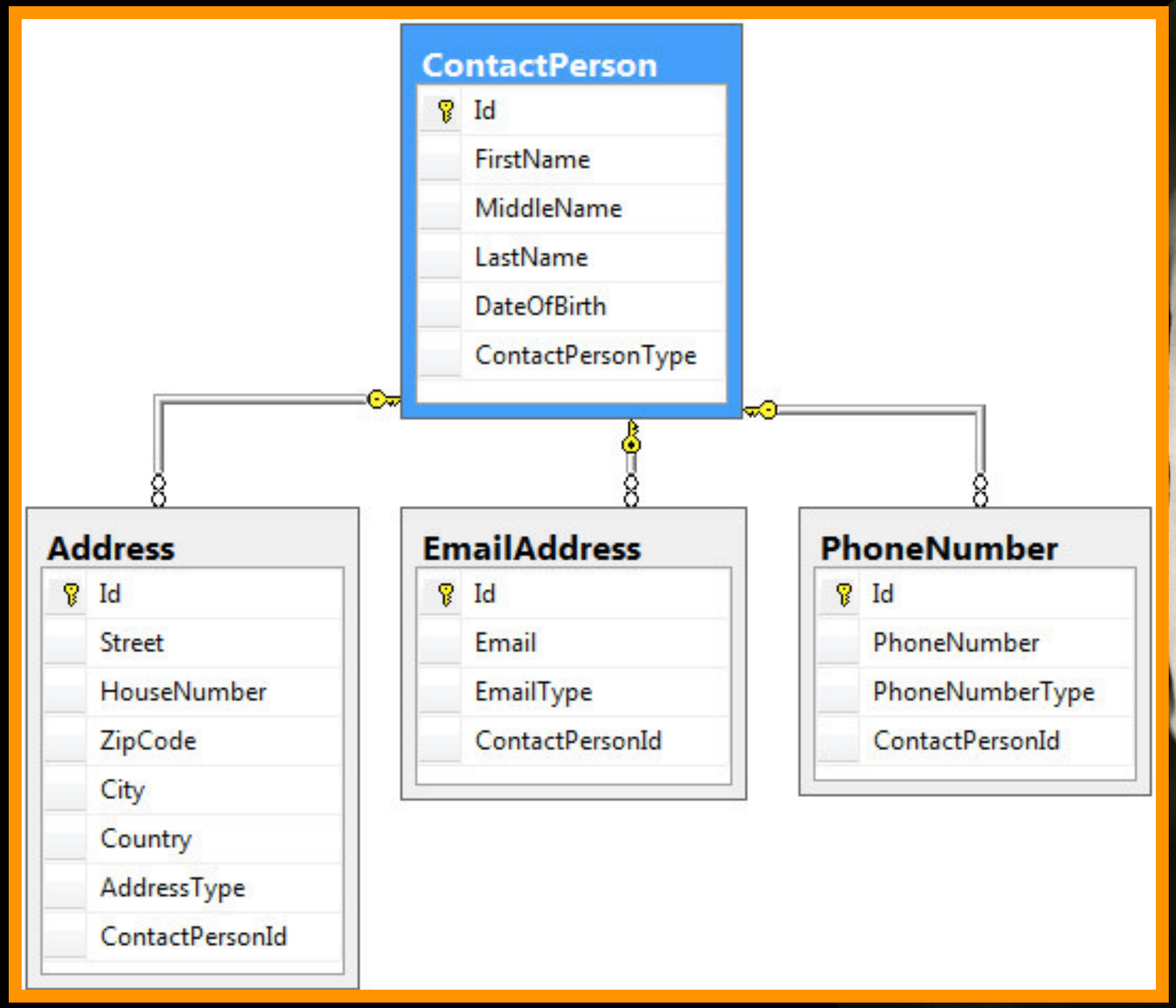
What is the

problem?

Two entities

Person.java

```
import ...

@Entity
public class Person {

    @Id
    int id;

    @Column
    private String name;

}
```


ContactPerson
Id
FirstName
MiddleName
LastName
DateOfBirth
ContactPersonType
Address
Id
Street
HouseNumber
ZipCode
City
Country
AddressType
ContactPersonId
EmailAddress
Id
Email
EmailType
ContactPersonId
PhoneNumber
Id
PhoneNumber
PhoneNumberType
ContactPersonId


EVIL

Really?

Is the database EVIL?

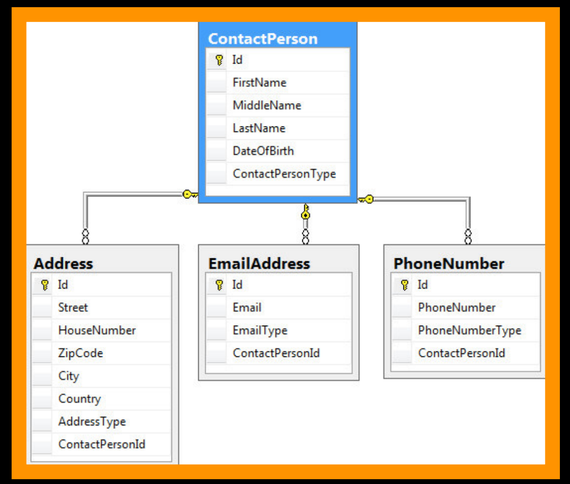
Source

Person.java

```
import ...

@Entity
public class Person {

    @Id
    int id;

    @Column
    private String name;

}
```

artifact

Database



manual, script, …

HOW

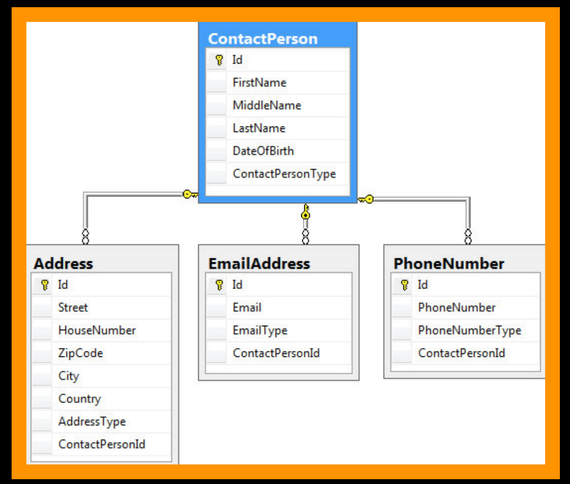
Source

Person.java

```
import ...

@Entity
public class Person {

    @Id
    int id;

    @Column
    private String name;

}
```

artifact

Database



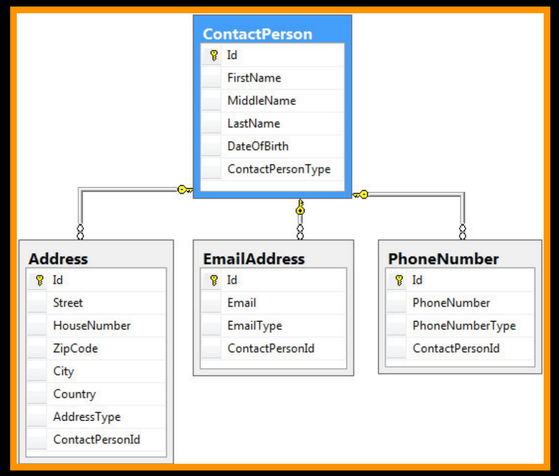
manual, script, …

Source

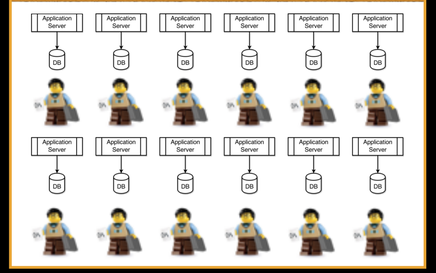
artifact

Person.java

```
import ...

@Entity
public class Person {

    @Id
    int id;

    @Column
    private String name;

}
```



Dev team



push

VCS

{code + migrations}

pull

Build server

publish

How do we keep
{ code + database }
CONSISTENT?

promote

Prod

Staging

QA

DEV/CI

release

Repository

artifacts

{code + migrations}

We get

Shiny Happy People

Schema Migration

DEMO DEMO DEMO DEMO

Schema Migration

DEMO DEMO DEMO DEMO

Migrations are applied automatically
during deployment

Schema Migration

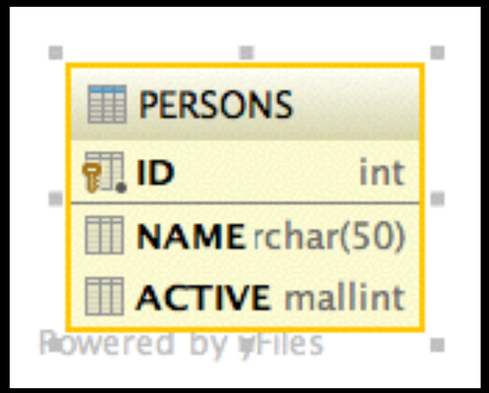
DEMO DEMO DEMO DEMO

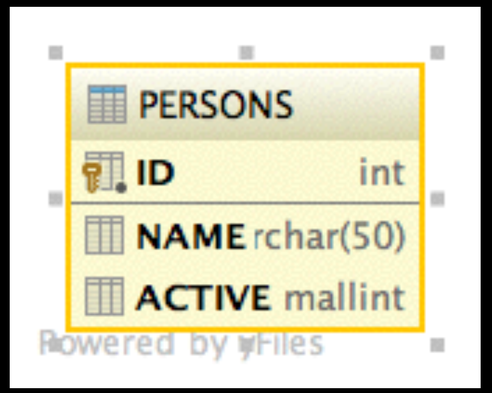
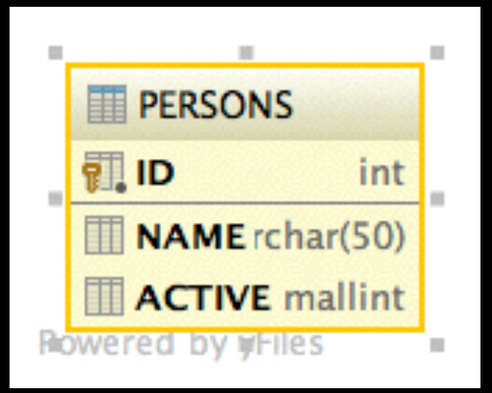
Spring based application

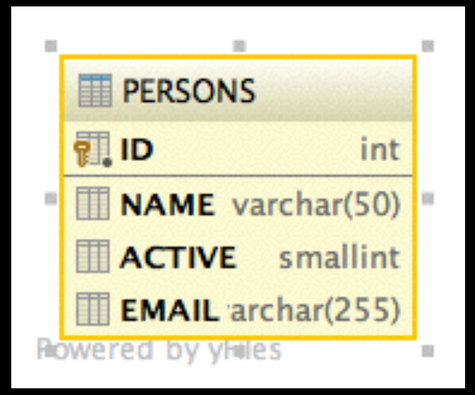
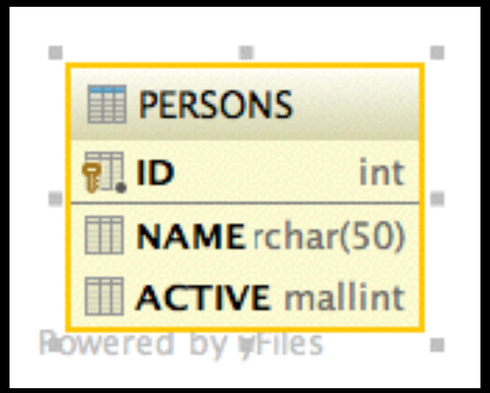
DEMONSTRATION OF

Shiny Happy People

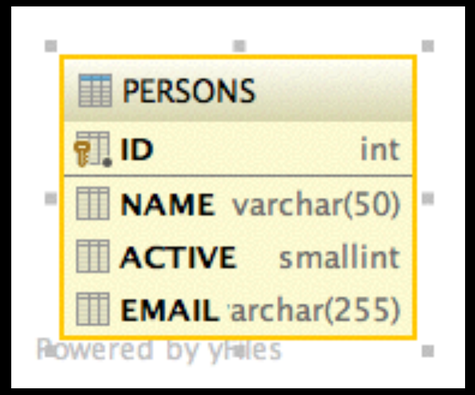
LOCAL	DEV	PROD
		
Version 0	Version 0	Version 0

LOCAL	DEV	PROD
		
Version 1	Version 0	Version 0

LOCAL	DEV	PROD
		
Version 1	Version 1	Version 0

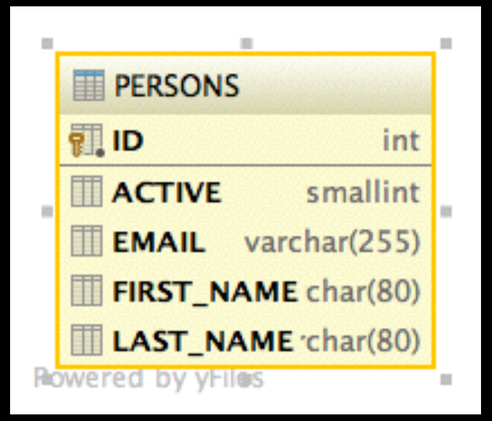
LOCAL	DEV	PROD
		
Version 2	Version 1	Version 0

LOCAL



Version 2

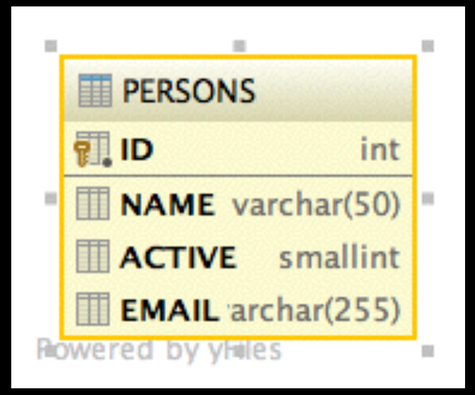
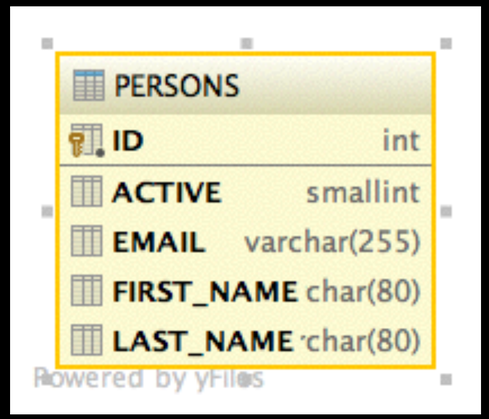
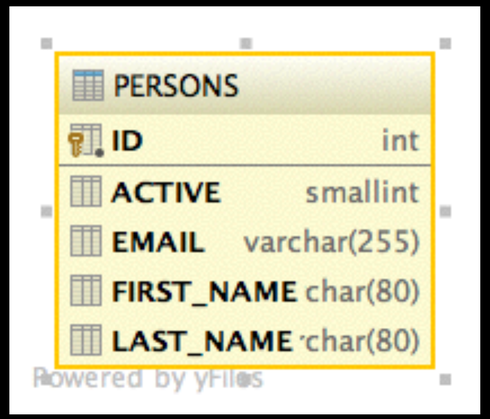
DEV



Version 3

PROD

Version 0

LOCAL	DEV	PROD
		
Version 2	Version 3	Version 3

What can we do?

create/drop TABLE
create/drop INDEX
add/remove CONSTRAINT
java REFACTORINGS
STORED PROCEDURES
create/drop VIEWS
insert DATA

and more!

What

is the trick?

When

is the trick?

Another view

Automatization of:
a sequences of instructions to be executed
given the revision of the source code

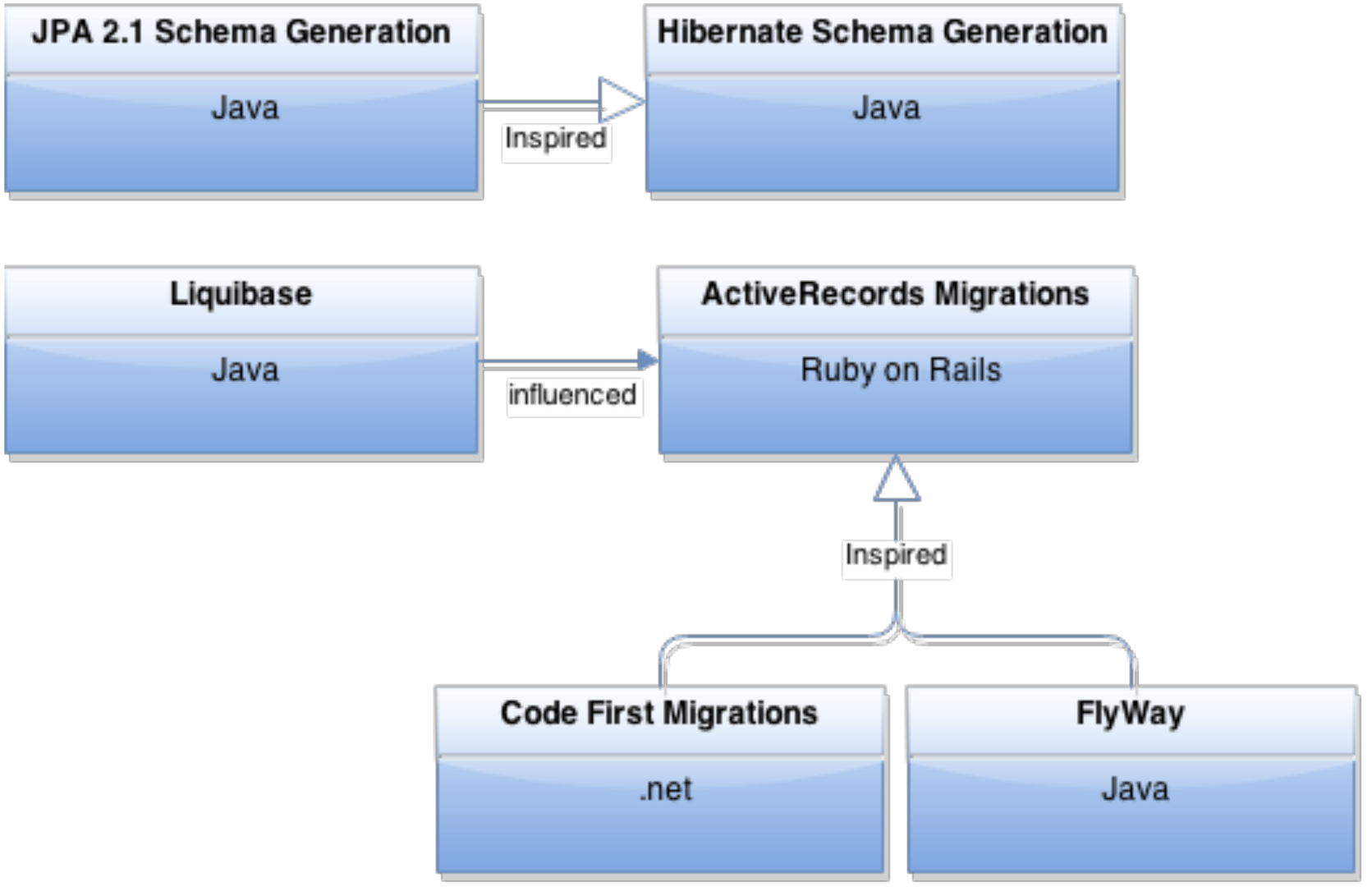
Database Change **Management**
Real Life Tricks

- Possible to use on an existing database
- Quick and simple to implement
- Database copies mix and match
- Wash copies of production database
- DBA approval / review process
- One click release possible

Part 2

The current situation

tools of the trade


JPA 2.1 Schema Generation
Java
Inspired
Hibernate Schema Generation
Java
Liquibase
Java
influenced
ActiveRecords Migrations
Ruby on Rails
Inspired
Code First Migrations
.net
FlyWay
Java

Tools of the Trade

~~ActiveRecord-Ruby~~ (Ruby)

~~Entity Framework Migrations~~ (.net)

Tools of the Trade

Hibernate & **JPA**

Schema Generation

Tools of the Trade

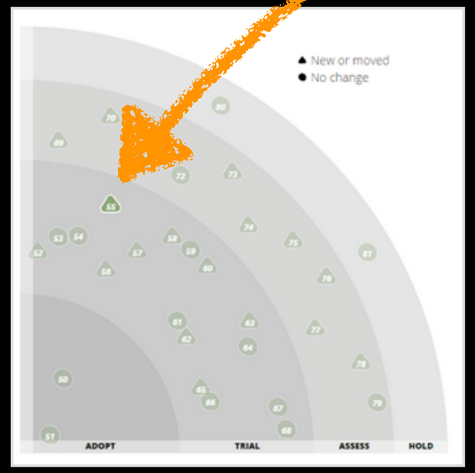
FlyWay & Liquibase

Convention over configuration
Gaining popularity

Stronger execution order
Supports different DB / environment
Stereotype of being “heavyweight"
Automatic rollback

Tools of the Trade

FlyWay



Convention over configuration
Get up and running in 5 minutes
Gaining popularity

Tools of the Trade

Liquibase

Stronger execution order
Supports different DB for different environment
Stereotype of being “heavyweight"
Automatic rollback

Tools of the Trade

I [Nathan Voxland] would summarize the differences as:

FlyWay is "lower level" with you specifying exactly the SQL you want ran whereas Liquibase is "higher level" with you specifying what you want changed and Liquibase computing the SQL

FlyWay manages changes by filename whereas Liquibase manages changes by order in a file.

Nathan Voxland, Liquibase developer

Tools of the Trade

summary

Schema migration is

- **Not** solved by Java EE today
- **Solved** by 3PP
- **Already** in competitors standards

Part 3

Future possibilities

to solve this problem

The future

my proposal

Add schema migration to

Java EE

Reasons to add Migrations to Java EE

- We are behind the competition
- Schema migrations is not well know to the broad developer community
- Increased Quality and Productivity

- Adding complexity to the platform

Advanced tooling

the future?

the future?

```
class User {
}
```

the future?

```
@Migration(id="1", author="duke")
class CreateUserTable {

  @CreateTable(User.class)
  @AddColumn(User.class)
  long id;

    @AddColumn(entity = User.class, size = "255")
    String name;
}
```

Warning: the following slides are possibilities for the future — not what we have

the future?

```
@Migration(id="1",
author="duke")
class CreateUserTable {
   @AddColumn(User.class)
   long id;
   …
}
```

Migration2Code Generator

```
@Entity
class User {
   @Id
   long id;
      …
}
```

Warning: the following slides are possibilities for the future — not what we have

the future?

```
@Entity
class User {

  @Id
  long id;

  @Column(size="255")
  String name;
}
```

the future?

```
@Migration(id="2", author="duke")
class AddColumnsToUser {

    @DropColumn(User.class)
    String name;

    @AddColumn(entity = User.class, size="80")
    String firstName;

    @AddColumn(entity = User.class, size="80")
    String lastName;

    @AddColumn(entity = User.class, size="255")
    String email;
}
```

Warning: the following slides are possibilities for the future — not what we have

the future?

```
@Entity
class User {

   @Id
   long id;

   @Column(size="80")
   String firstName;

   @Column(size="80")
   String lastName;

   @Column(size="255")
   String email;
}
```

Wrap up

YOU! Should use Schema Migration
WE? → JCP → JSR!
What? Lets figure it out!
When? Now. Lets start!

rikard.thulin@squeed.com

Discussion
and
Q & A

rikard.thulin@squeed.com