JAX, 26.04.2018

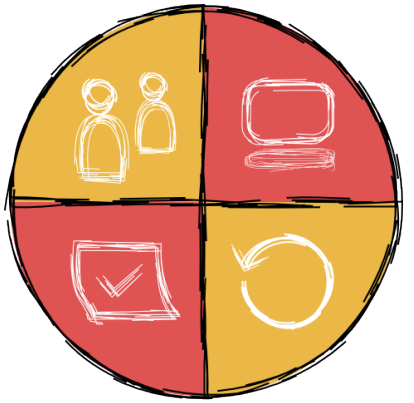
# Continuous Database Integration mit Flyway

Sandra Parsick
mail@sandra-parsick.de
@SandraParsick

# Zu meiner Person

- Sandra Parsick
- Freiberuflicher Softwareentwickler und Consultant im Java-Umfeld
- Schwerpunkte:
  - Java Enterprise Anwendungen
  - Agile Methoden
  - Software Craftmanship
  - Automatisierung von Entwicklungsprozessen
- Trainings
- Workshops
- Softwerkskammer Ruhrgebiet

- Twitter: @SandraParsick
- Blog: http://blog.sandra-parsick.de
- E-Mail: mail@sandra-parsick.de



# Agenda

- Continuous Database Integration (CDBI)
- Flyway
- Fallstricke

# Continuous Database Integration

- Definition
- Motivation
- Aufbau

# Definition

*„Continuous Database Integration (CDBI) is the process of rebuilding your database and test data any time a change is applied to a project's version control repository“*

(aus Continuous Integration by Paul M. Duvall, Steve Matyas und Andrew Glover)

# Motivation

- Alle Entwickler teilen sich eine Testdatenbank.
- Keiner weiß, welche Datenbankskripte auf welchen Datenbankinstanzen ausgeführt worden.
- Testdatenbank unterscheidet sich von der Produktionsdatenbank.
- Datenbankmigrationsskripte verteilen sich auf Emails, Release Notes, Ticketsysteme, etc.

# Aufbau

- Behandle den Datenbank-Code wie einen ganz normalen Source-Code
  - Alle Datenbank Artefakte (DDL, DML, Konfigurationen, Testdaten, Stored Procedures, Functions etc) gehören ins VCS.
  - Jede Änderung an den DB Artefakten wird getestet.
- Jeder Entwickler hat seine eigene Datenbank / Testdatenbanken ähneln den Produktionsdatenbanken.
  - Automatisiertes Aufsetzen der Datenbank.
- Änderungen an der Datenbank sind nachvollziehbar.
  - Historie der Änderungen

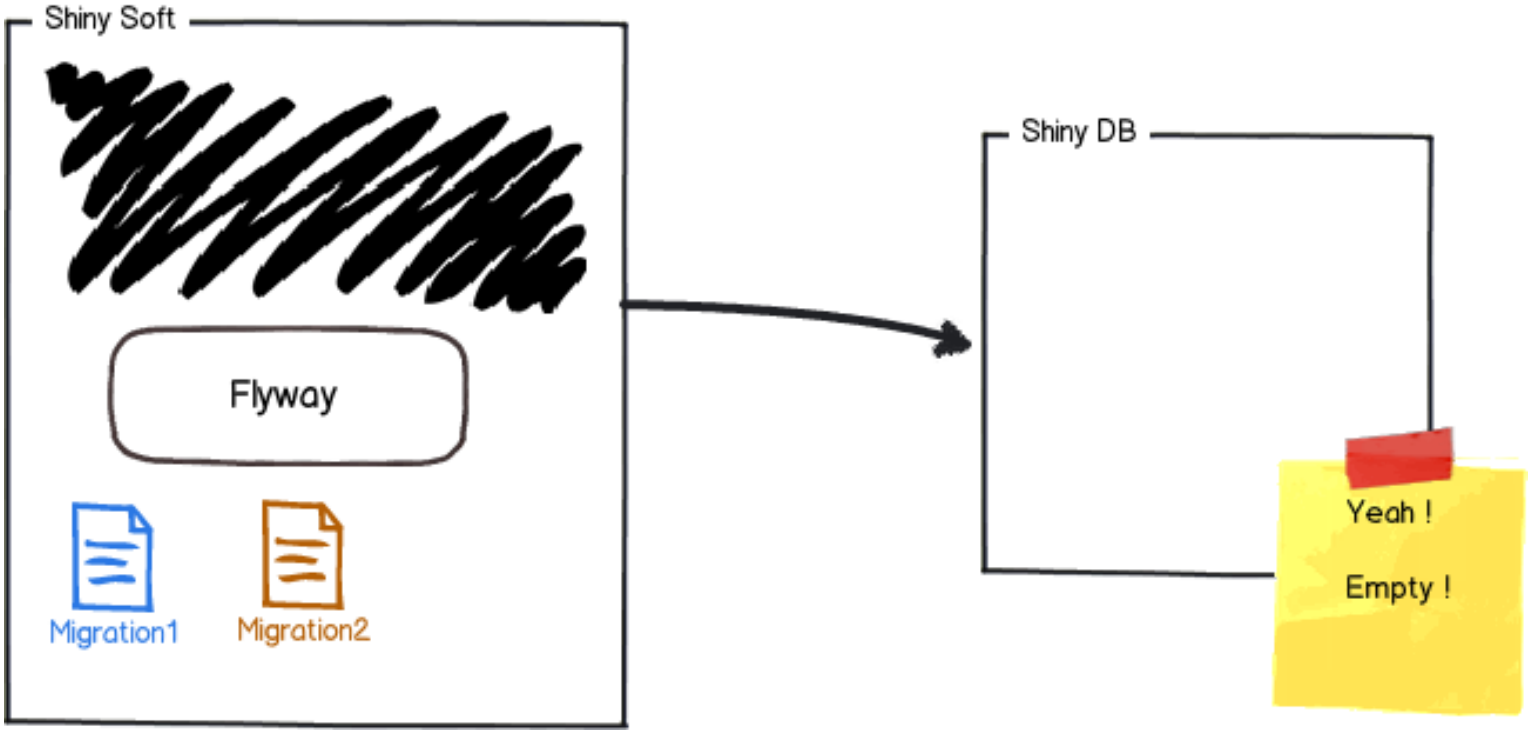
# Flyway

- Was ist Flyway?
- Wie funktioniert Flyway?
- Wie werden Migrationsskripte für Flyway geschrieben?
- Was kann Flyway nicht?
- Wie kann Flyway benutzt werden?
- Wie unterscheidet sich Flyway zu Liquibase?

# Was ist Flyway?


- Migration Framework für Relationale Datenbanken basierend auf Java
- Erstellt eine Datenbank „from scratch“
- Verwaltet den Stand der Datenbank
- Vier Migrationsmodi:
    - SQL-, Java-basierte Migration
    - Versionierte, wiederholbare Migration
- Aktuelle Version: 5.0.7
- Homepage: http://flywaydb.org/
- Twitter: @flywaydb

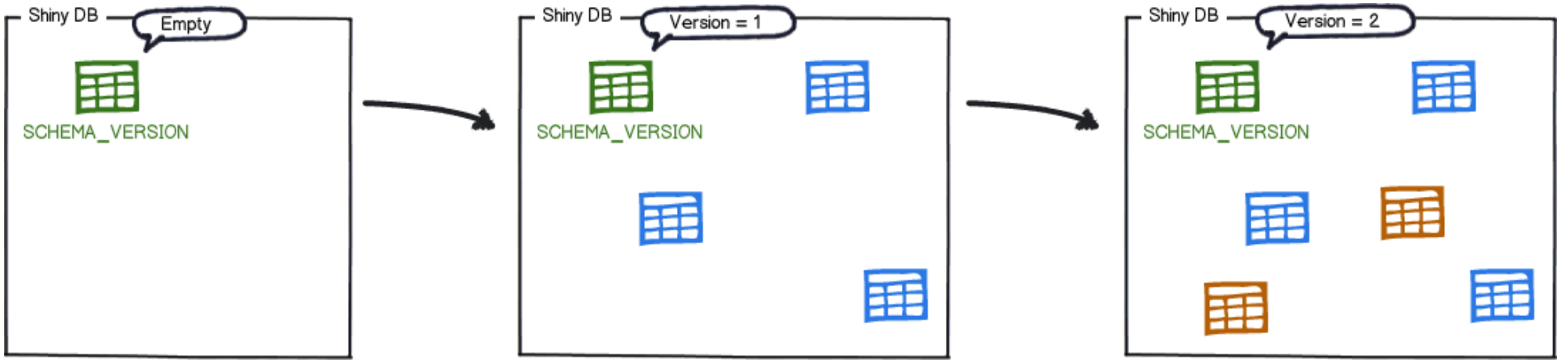
# Wie funktioniert Flyway?




Reference: flywaydb.org

# Wie funktioniert Flyway?

migrate



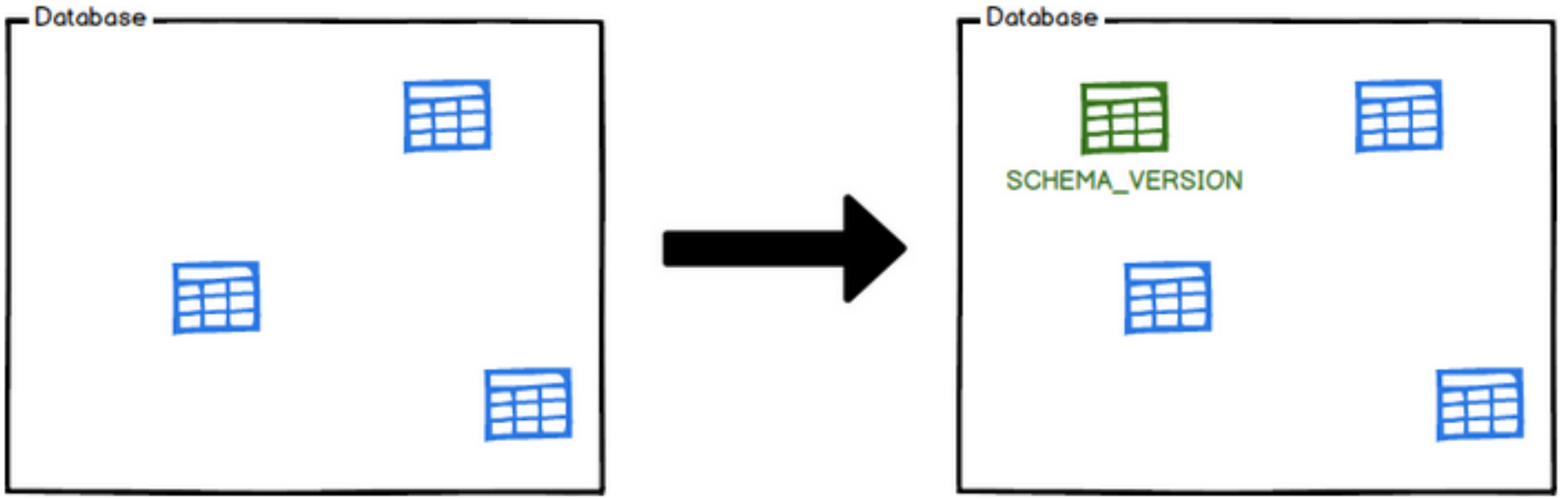

Reference: flywaydb.org

schema_version

| installed_rank | version | description | type | script | checksum | installed_by | installed_on | execution_time | success |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Initial Setup | SQL | V1__Initial_Setup.sql | 1996767037 | axel | 2016-02-04 22:23:00.0 | 546 | true |
| 2 | 2 | First Changes | SQL | V2__First_Changes.sql | 1279644856 | axel | 2016-02-06 09:18:00.0 | 127 | true |

# Wie funktioniert Flyway?

baseline




Reference: flywaydb.org

# Migrationsskripte

- Vier Möglichkeiten

| | Versioniert | Wiederholbar |
|---|---|---|
| SQL-basiert | ✓ | ✓ |
| Java-basiert | ✓ | ✓ |

# Versionierte Migration

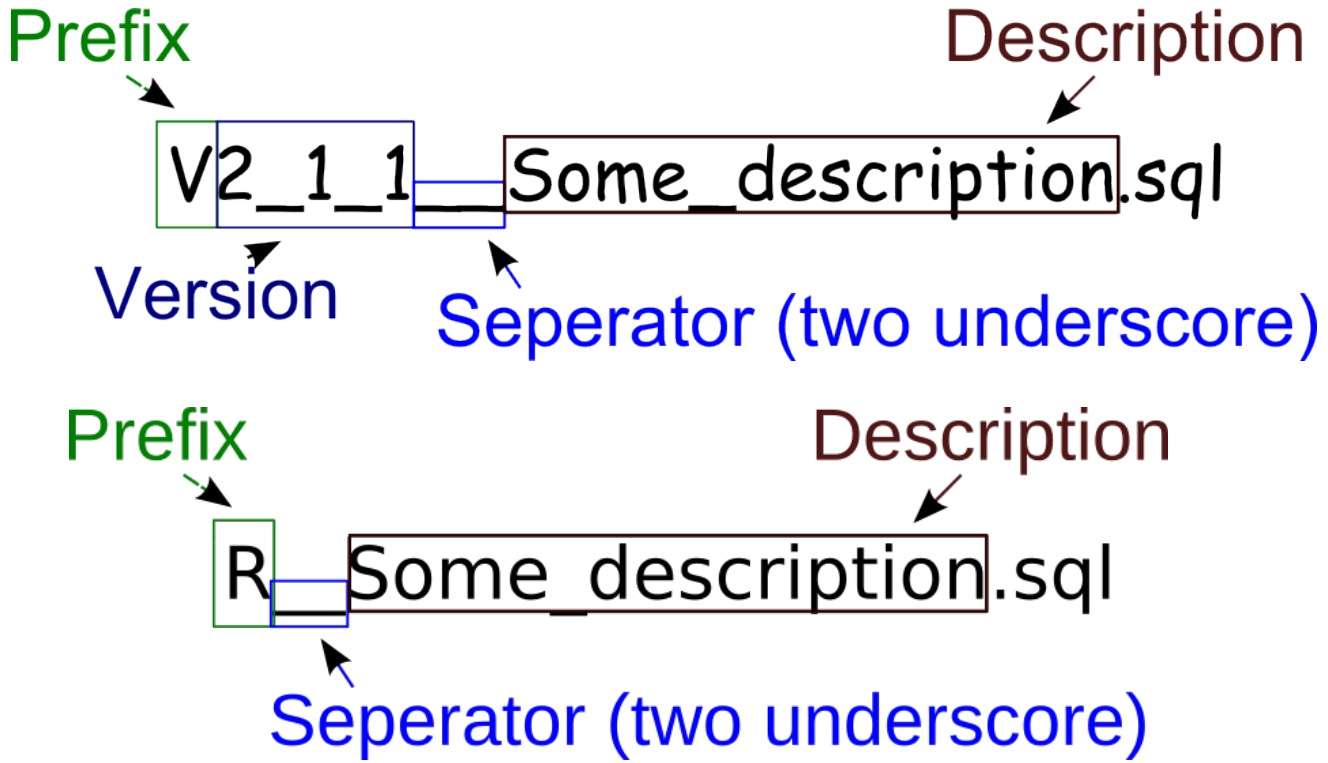
- **Eigenschaften**
  - Skripte haben eine eindeutige Version
  - Werden genau einmal ausgeführt
- **Typische Anwendungsfälle**
  - DDL Änderungen (CREATE/ALTER/DROP für TABLES,INDEXES,FOREIGN KEYS,...)
  - Einfache Datenänderungen

# Wiederholbare Migration

- **Eigenschaften**
  - Skripte haben keine Versionsnummer
  - Werden immer dann ausgeführt, wenn sich ihre Checksumme ändert
  - Werden immer dann ausgeführt, nachdem alle versionierte Skripte ausgeführt wurden
- **Typische Anwendungsfälle**
  - (Wieder-) Erstellung von  views / procedures / functions / packages / ...
  - Massenreimport von Stammdaten

# SQL Migration

- **Typische Anwendungsfälle**
  - DDL Änderungen (CREATE/ALTER/DROP für TABLES,VIEWS,TRIGGERS,SEQUENCES,...)
  - Einfache Datenänderungen
- **Benamung der Skripte**

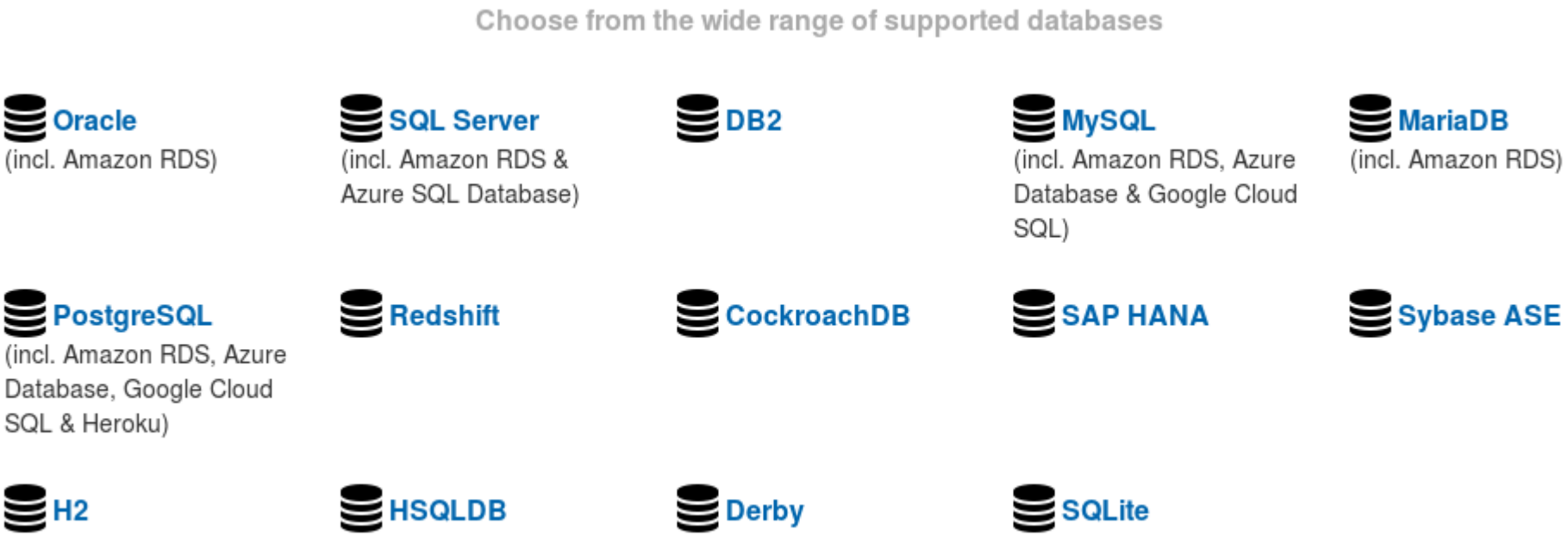


# SQL Migration

- **Syntax**
  - Statement kann über mehrere Zeile gehen
  - Platzhaltersupport
  - Kommentare:  Single (–) oder Multi-Line (/* */)
  - Datenbank-spezifische SQL Syntax
- **Beispiel**

```
1   /* Create a table for person */
2
3   Create table person (
4       first_name varchar(128),
5       last_name varchar(128)
6   );
```

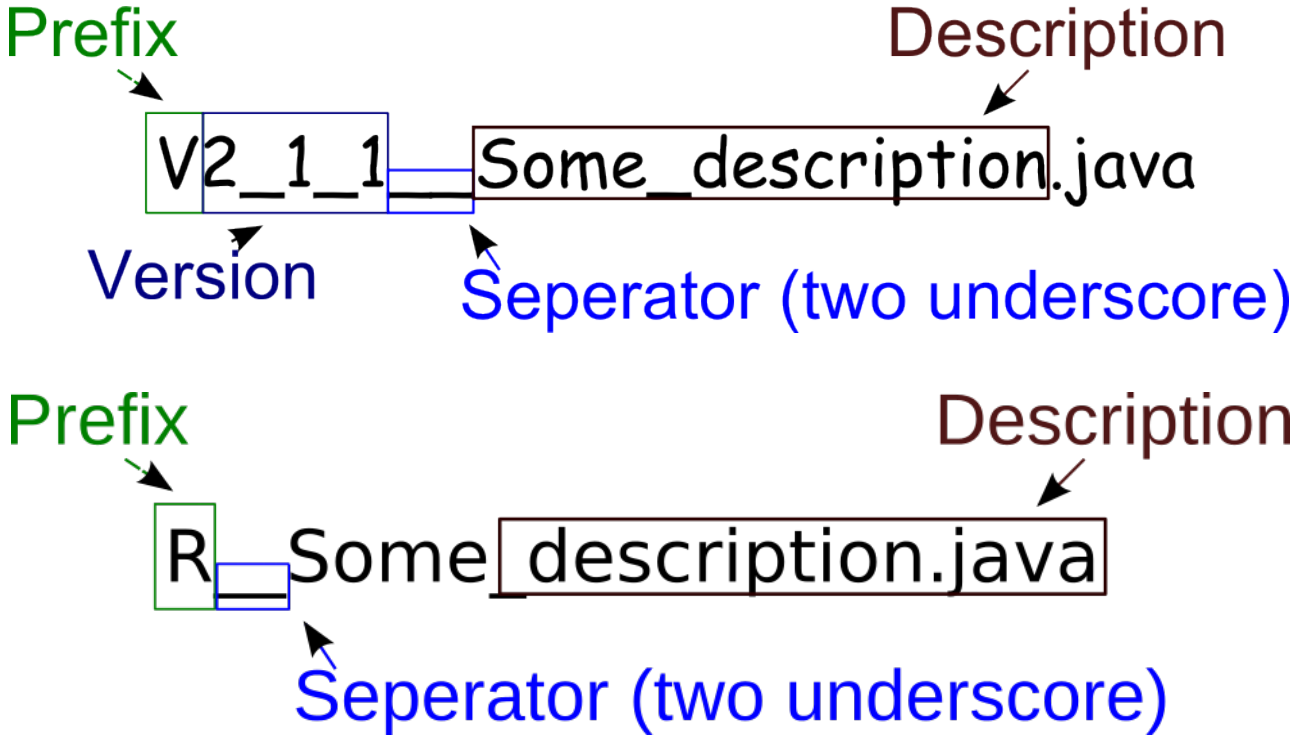
# Unterstützte Datenbanken




Reference: flywaydb.org

# Java Migration

- **Typische Anwendungsfälle**
  - BLOB & CLOB Änderungen
  - Fortgeschrittene Änderungen an Massendaten (Neuberechnungen, fortgeschrittene Formatänderungen, …)
- **Benamung der Java Klassen**



# Java Migration

## Beispiel

```java
package db.migration;

import java.sql.Connection;
import java.sql.Statement;
import org.flywaydb.core.api.migration.jdbc.JdbcMigration;


public class V1_1_0__Insert_Data implements JdbcMigration {

    @Override
    public void migrate(Connection connection) throws Exception {
        try (Statement statement = connection.createStatement()) {
            statement.execute("Insert into person (first_name, last_name) Values ('Alice', 'Bob')");
        }

    }

}

```

# Java Migration

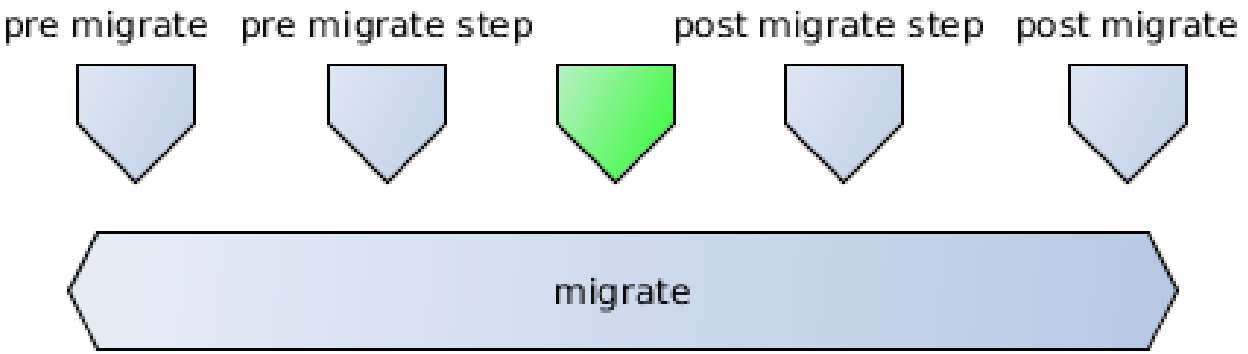
## Beispiel Spring Support

```java
package db.migration;

import org.flywaydb.core.api.migration.spring.SpringJdbcMigration;
import org.springframework.jdbc.core.JdbcTemplate;


public class V1_2_0__Create_Table_With_Spring_Support implements SpringJdbcMigration {

    @Override
    public void migrate(JdbcTemplate jdbcTemplate) throws Exception {
       jdbcTemplate.execute("Create table address (street Varchar(128), place Varchar(128))");
    }

}

```

# Migration für Fortgeschrittene - Callbacks

- **Typische Anwendungsfälle**
  - Stored Procedure Kompilierung
  - Materialized View Update

- **Flyway Lifecycle (Beispiel migrate)**



# SQL Callbacks

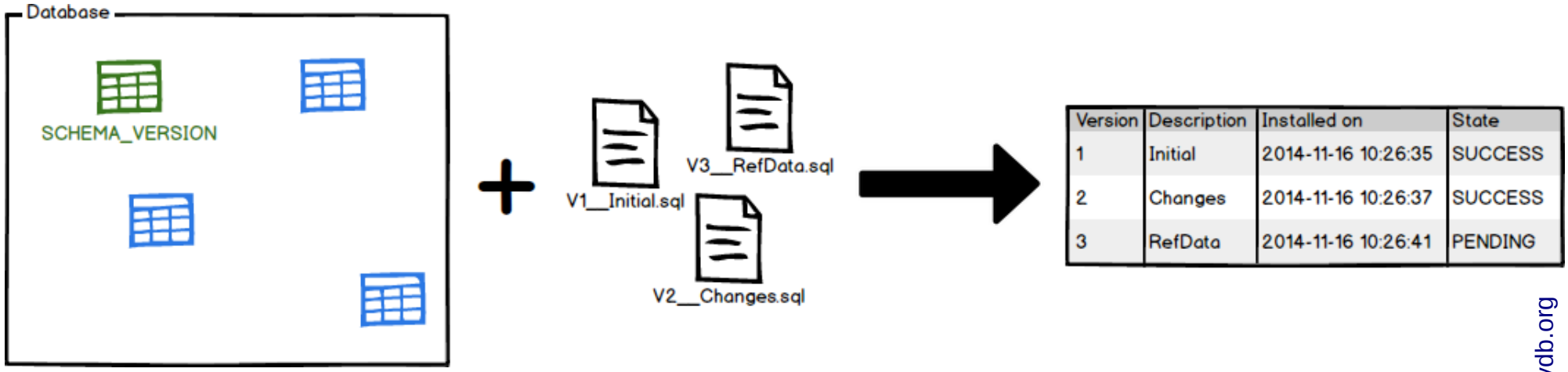
- **Beispiel migrate-Lifecycle:**
  - SQL Callback Skripte werden anhand deren Namen erkannt:
    - BeforeMigrate.sql
    - BeforeEachMigrate.sql
    - AfterEachMigrate.sql
    - AfterMigrate.sql

# Java Callbacks

```
public interface FlywayCallback {
    /**
     * Runs before the clean task executes.
     *
     * @param connection A valid connection to the database.
     */
    void beforeClean(Connection connection);

    /**
     * Runs after the clean task executes.
     *
     * @param connection A valid connection to the database.
     */
    void afterClean(Connection connection);

    /**
     * Runs before the migrate task executes.
     *
     * @param connection A valid connection to the database.
     */
    void beforeMigrate(Connection connection);
```

# Weitere Flyway Befehle

info

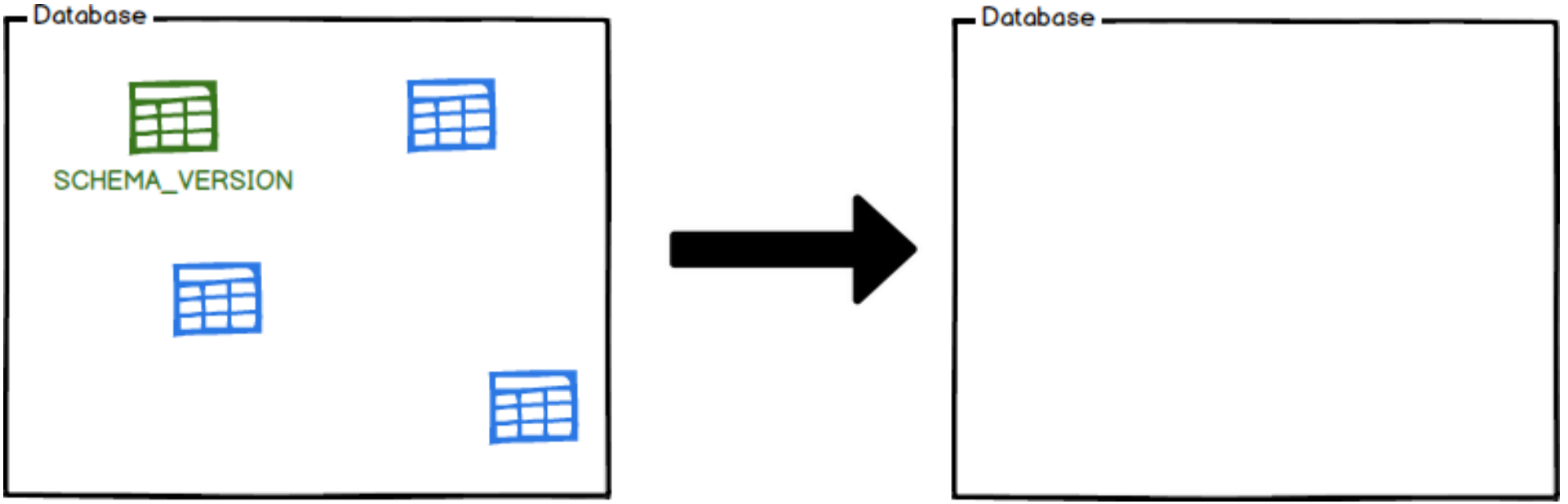



| Version | Description | Installed on | State |
|---|---|---|---|
| 1 | Initial | 2014-11-16 10:26:35 | SUCCESS |
| 2 | Changes | 2014-11-16 10:26:37 | SUCCESS |
| 3 | RefData | 2014-11-16 10:26:41 | PENDING |

Reference: flywaydb.org

# Weitere Flyway Befehle

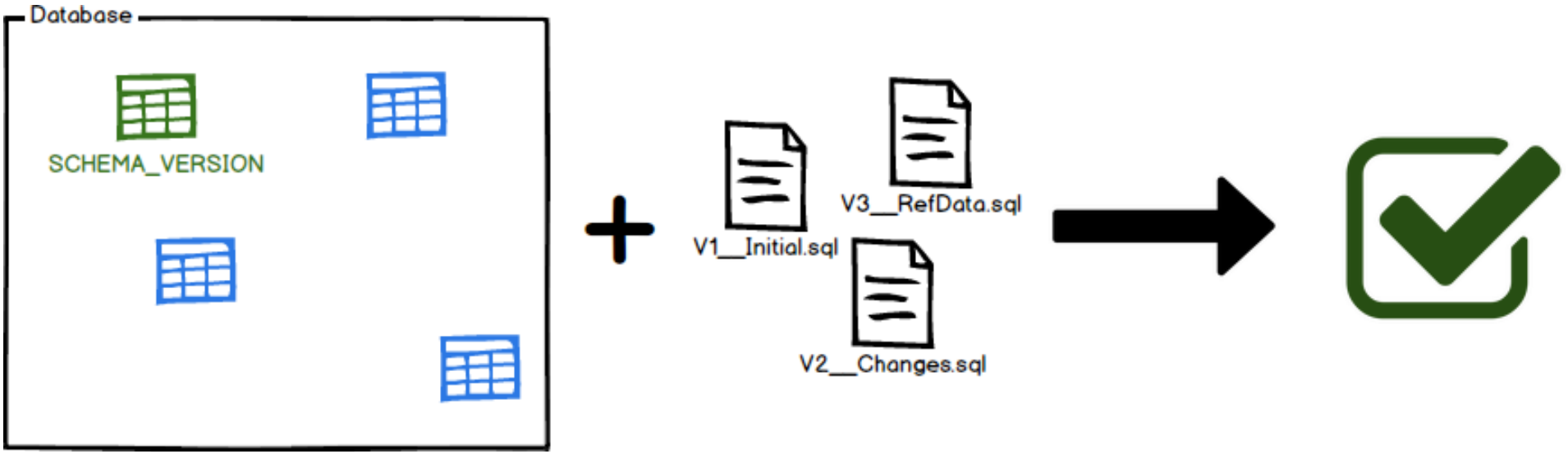
clean




Reference: flywaydb.org

# Weitere Flyway Befehle

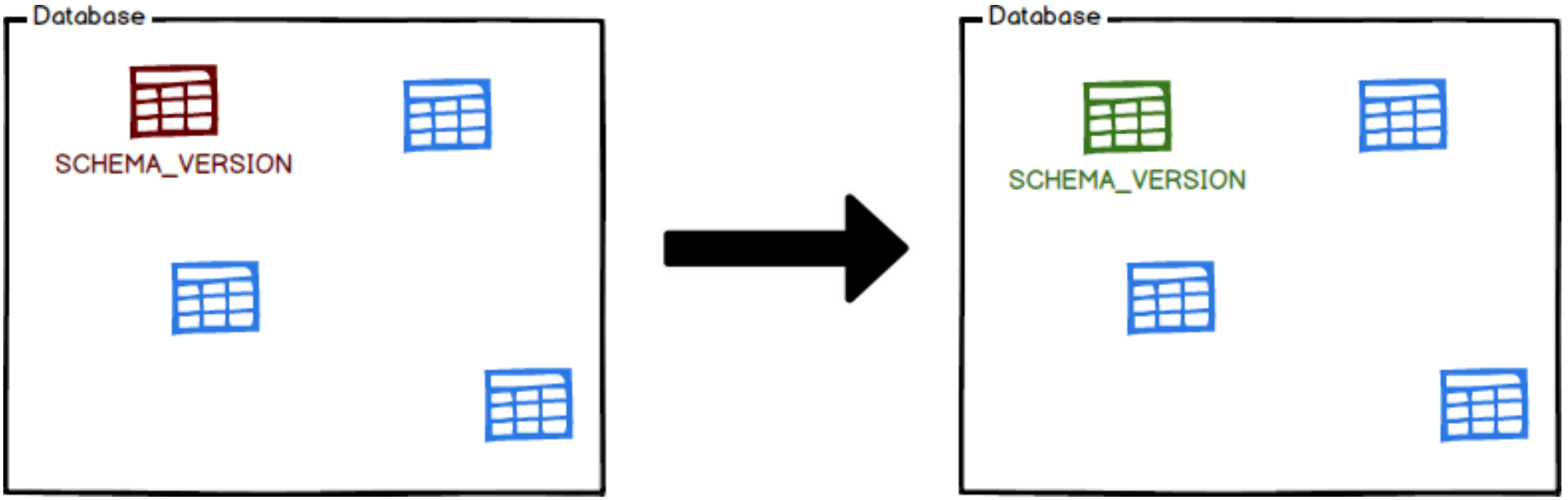
validate




Reference: flywaydb.org

# Weitere Flyway Befehle

repair




Reference: flywaydb.org

# Was kann Flyway nicht?

- Rollback Skripte aufrufen (Community Edition)
- „Write once, run on many database vendors“

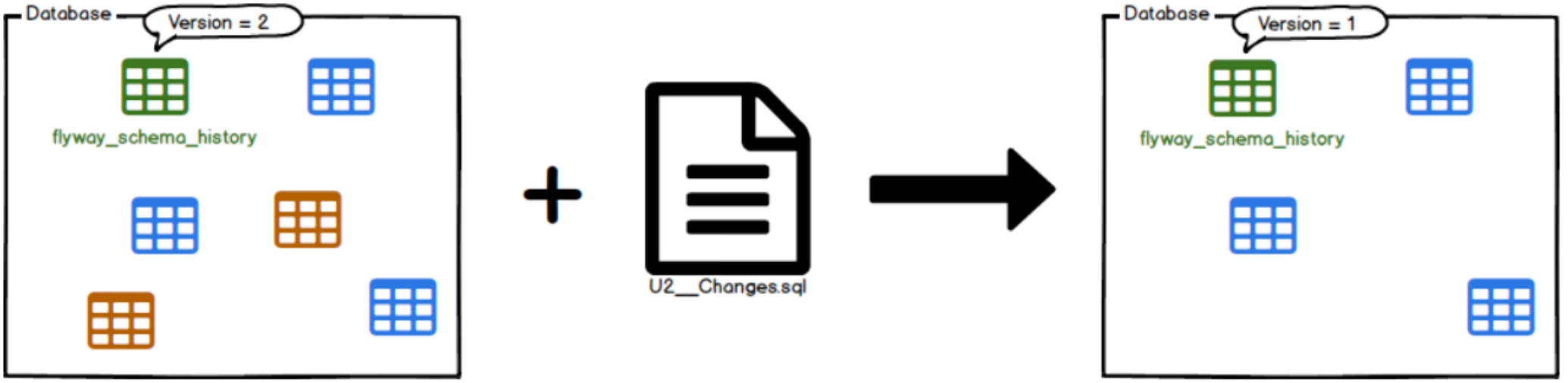
# Neu seit Version 5

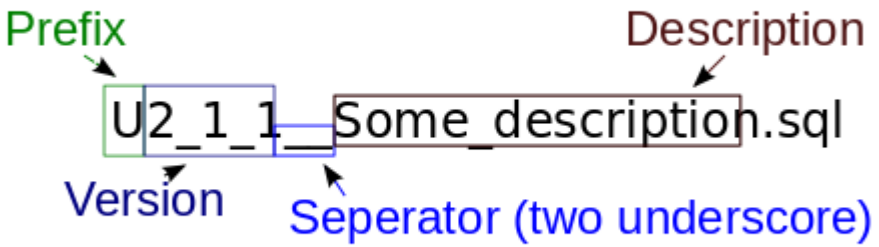
- Unterscheidung zwischen
  - Community Edition
  - Pro Edition
  - Enterprise Edition

| | Community Edition | Pro Edition | Enterprise Edition |
|---|---|---|---|
| SQL-based migrations | ✔ | ✔ | ✔ |
| Java-based migrations | ✔ | ✔ | ✔ |
| Repeatable migrations | ✔ | ✔ | ✔ |
| Placeholder replacement | ✔ | ✔ | ✔ |
| Callbacks | ✔ | ✔ | ✔ |
| Custom migration resolvers/executors | ✔ | ✔ | ✔ |
| Safe for multiple nodes in parallel | ✔ | ✔ | ✔ |
| Native SQL dialect support (PL/SQL, SQLPL, T-SQL, ...) | ✔ | ✔ | ✔ |
| Latest database versions compatibility | ✔ | ✔ | ✔ |
| Java 8 / 9 compatibility | ✔ | ✔ | ✔ |
| Oracle SQL*Plus compatibility | | ✔ | ✔ |
| Custom error handlers | | ✔ | ✔ |
| Dry runs | | ✔ | ✔ |
| Undo | | ✔ | ✔ |
| Display query results | | ✔ | ✔ |
| Older database versions compatibility | | | ✔ |
| Java 6 / 7 compatibility | | | ✔ |
| License | Apache v2 | Commercial | Commercial |

# Pro/Enterprise Feature: Undo

undo




Reference: flywaydb.org

# Pro/Enterprise Feature: Dry Run

- Set up read-only connection
- Generierung einer einzelnen SQL-Datei, die alle Befehle enthält, die regulär ausgeführt werden würden
- Aktivierung durch Property `flyway.dryRunOutput=/my/sql/dryrun-outputfile.sql`

# Pro/Enterprise Feature: Custom Error Handler

Default Fehlermeldung in Flyway

```
Migration V1__Create_person_table.sql failed
-----------------------------------------
SQL State  : 42001
Error Code : 42001
Message    : Syntax error in SQL statement "CREATE TABLE1[*] PERSON "; expected "OR, FORCE, VIEW, ...
Location   : V1__Create_person_table.sql (/flyway-tutorial/V1__Create_person_table.sql)
Line       : 1
Statement  : create table1 PERSON
```

Reference: flywaydb.org

# Pro/Enterprise Feature: Custom Error Handler

```
DB: Warning: execution completed with warning (SQL State: 99999 - Error Code: 17110)
```

```java
package org.mycompany.mypkg;

import org.flywaydb.core.api.FlywayException;
import org.flywaydb.core.api.errorhandler.Context;
import org.flywaydb.core.api.errorhandler.ErrorHandler;
import org.flywaydb.core.api.errorhandler.Warning;

public class OracleProcedureFailFastErrorHandler implements ErrorHandler {
    @Override
    public boolean handle(Context context) {
        for (Warning warning : context.getWarnings()) {
            if ("99999".equals(warning.getState()) && warning.getCode() == 17110) {
                throw new FlywayException("Compilation failed");
            }
        }
        return false;
    }
}
```

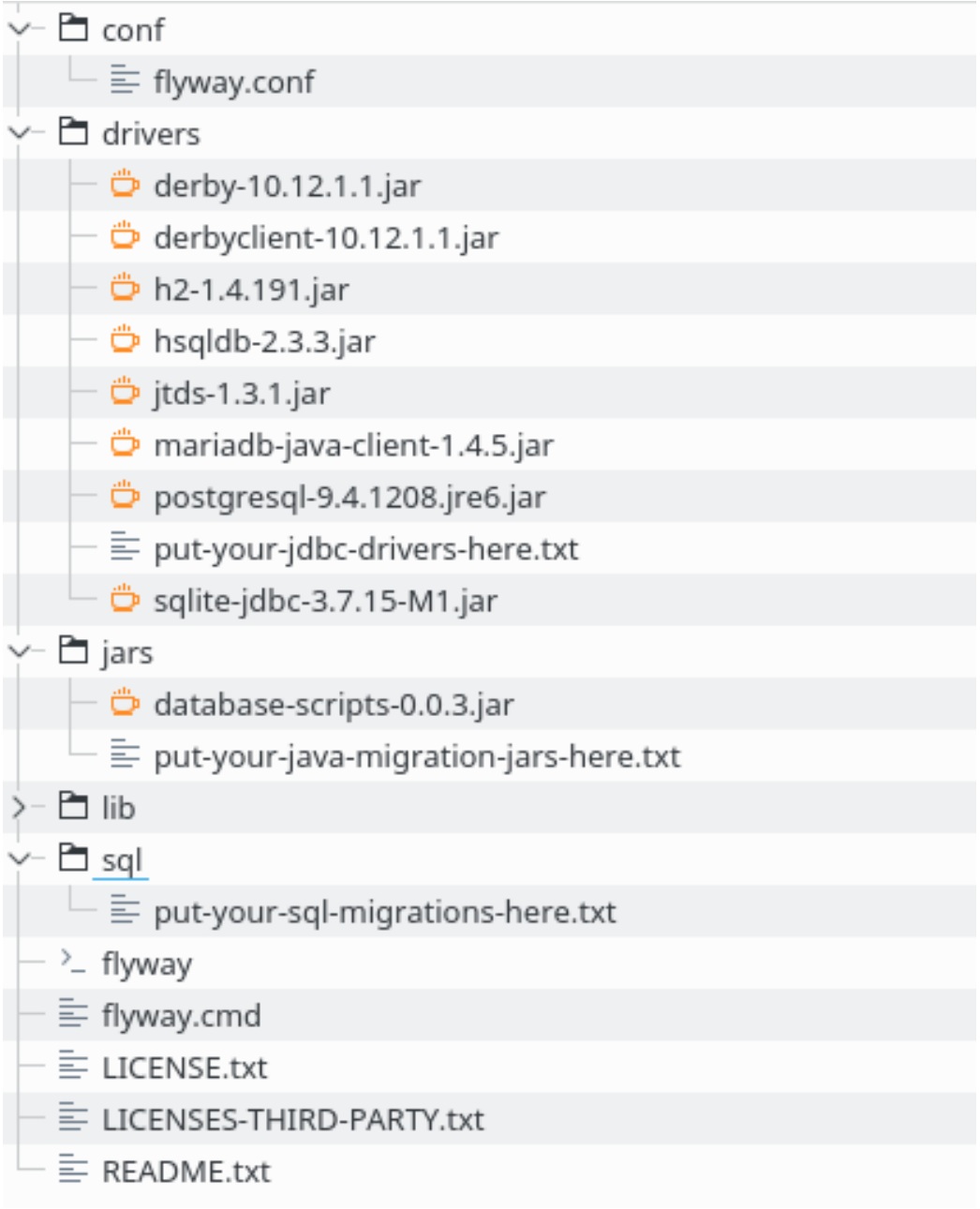
```
flyway.errorHandlers=org.mycompany.mypkg.OracleProcedureFailFastErrorHandler
```

Reference: flywaydb.org

# Wie kann Flyway benutzt werden?

- **Flyway Clients:**
  - Java API
  - Maven Plugin
  - Command-line Tool
  - Gradle Plugin
  - SBT Plugin
  - Ant task
- **Third Party Plugins:**
  - Spring Boot
  - Grails
  - Dropwizard
  - Play
  - Und weitere

# Command-line Tool

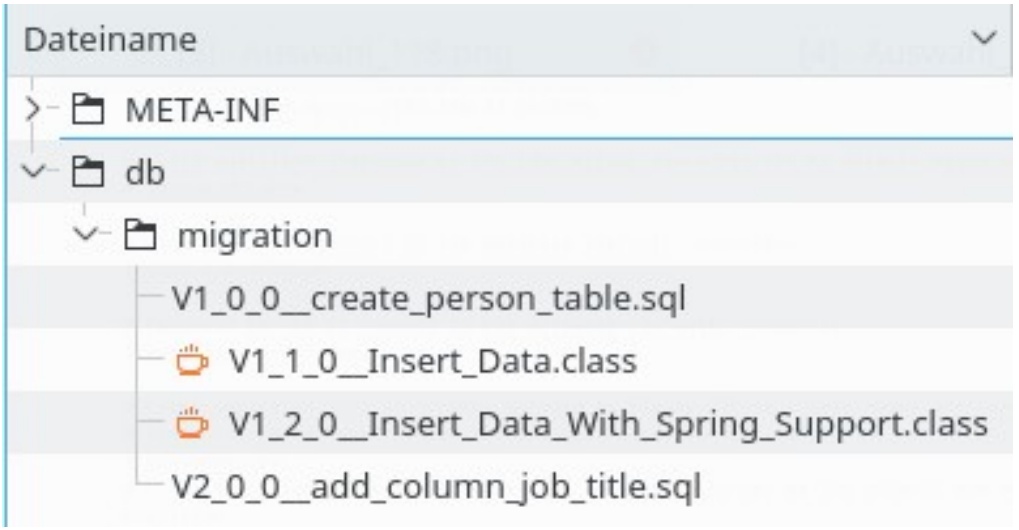


# Command-line Tool

flyway.conf

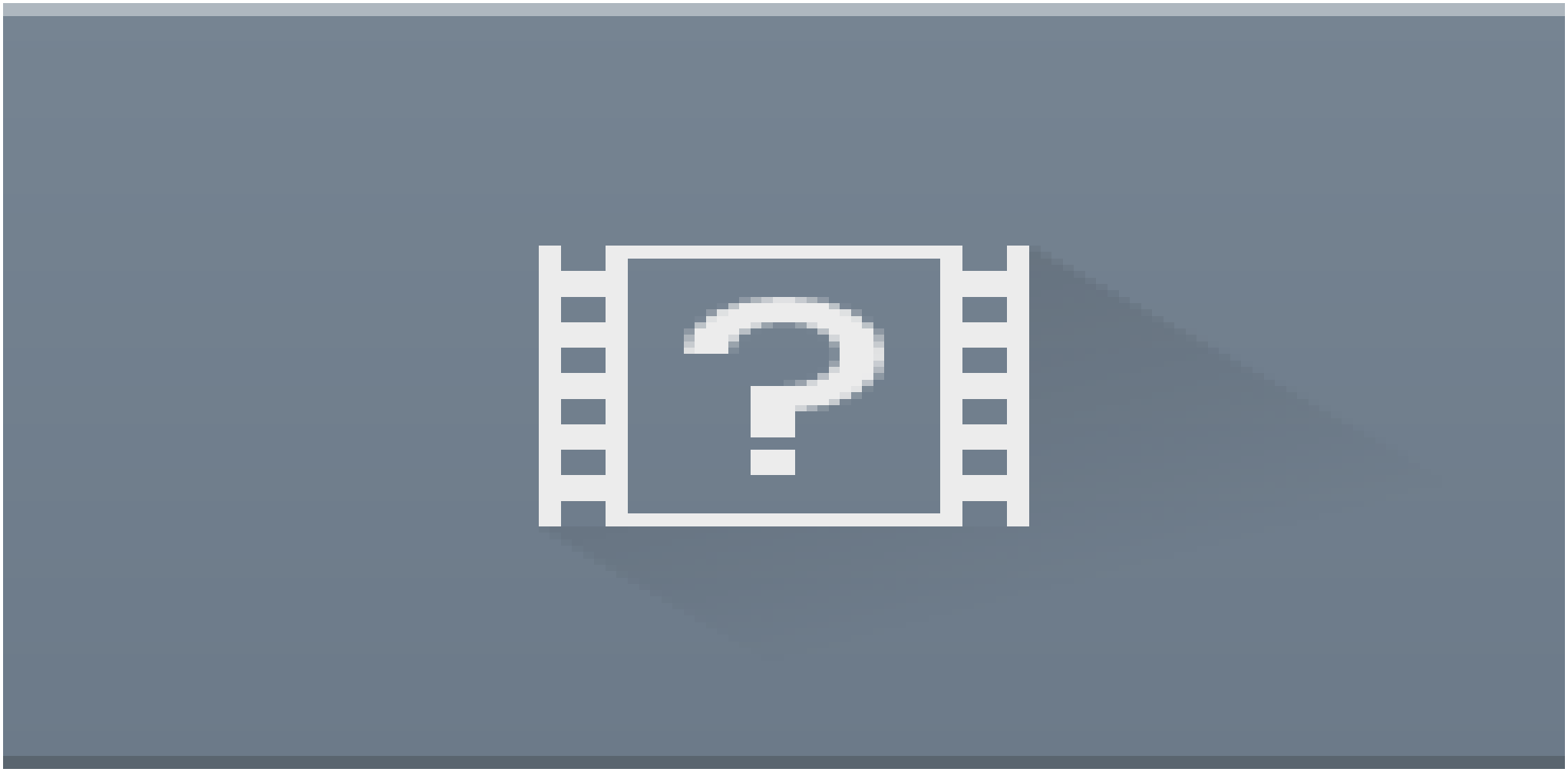
```
flyway.url=jdbc:mysql://192.168.33.10:3306

# Fully qualified classname of the jdbc driver (autodetected by default based on flyway.url)
# flyway.driver=

# User to use to connect to the database (default: <<null>>)
flyway.user=flyway

# Password to use to connect to the database (default: <<null>>)
flyway.password=flyway

# Comma-separated list of schemas managed by Flyway. These schema names are case-sensitive.
# (default: The default schema for the datasource connection)
# Consequences:
# - The first schema in the list will be automatically set as the default one during the
migration.
# - The first schema in the list will also be the one containing the metadata table.
# - The schemas will be cleaned in the order of this list.
flyway.schemas=flyway_demo

# Name of Flyway's metadata table (default: schema_version)
# By default (single-schema mode) the metadata table is placed in the default schema for the
connection provided by the datasource.
# When the flyway.schemas property is set (multi-schema mode), the metadata table is placed in
the first schema of the list.
# flyway.table=

# Comma-separated list of locations to scan recursively for migrations. (default:
filesystem:<<INSTALL-DIR>>/sql)
# The location type is determined by its prefix.
# Unprefixed locations or locations starting with classpath: point to a package on the classpath
and may contain both sql and java-based migrations.
# Locations starting with filesystem: point to a directory on the filesystem and may only contain
sql migrations.
flyway.locations=db/migration
```

# Command-line Tool




database-scripts-0.0.3.jar

# Command-line Tool



# Command-line Tool

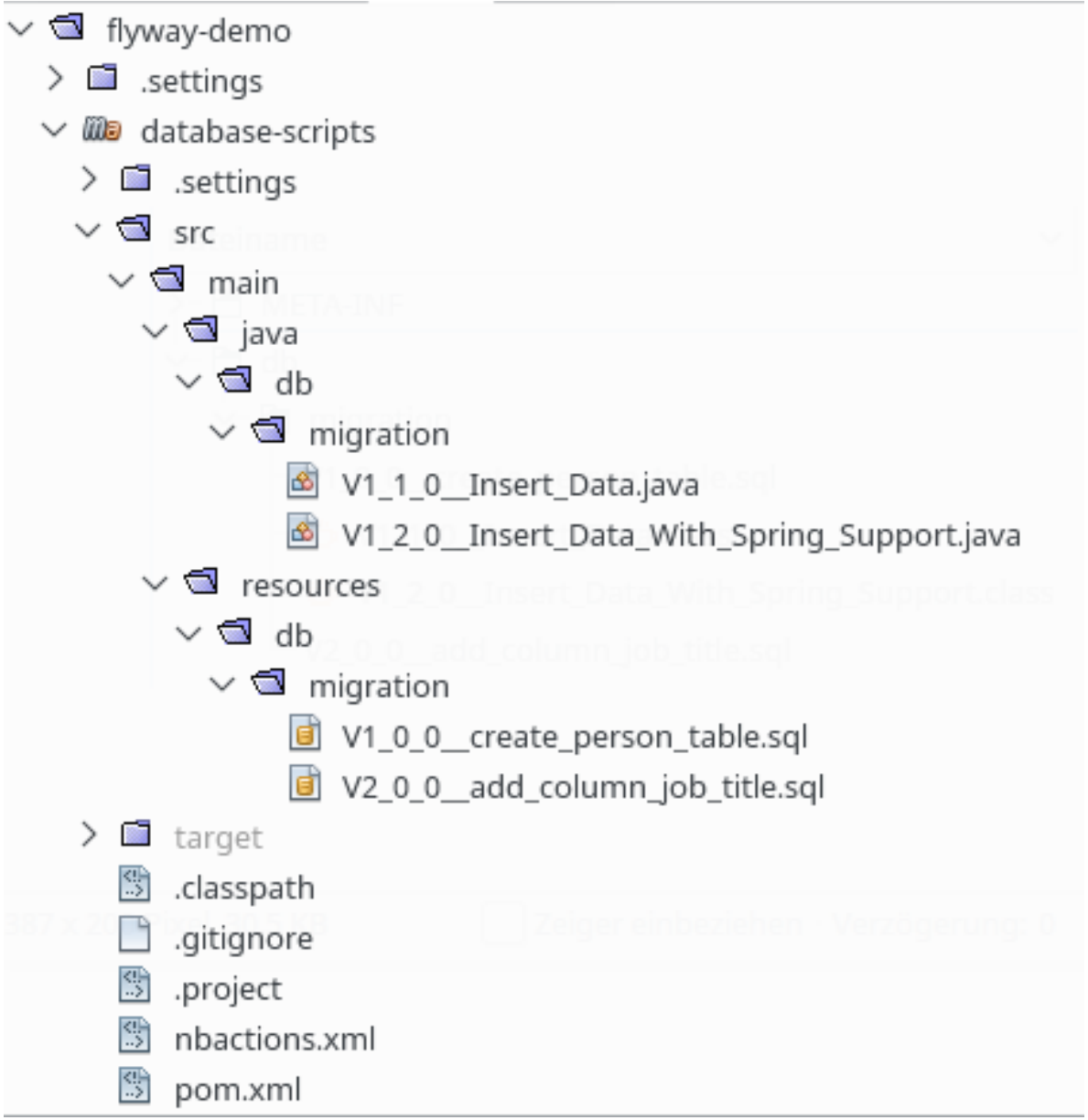
## Pipeline Deployment Pipeline Demo

Recent Changes

### Stage View

| | DB Migration | Webdeployment |
|---|---|---|
| Average stage times: (Average full run time: ~7s) | 6ms | 7s |
| #20 Dec 02 10:54 — No Changes | 4ms | 8s |
| #19 Dec 02 10:37 — No Changes | 14ms | 8s failed |
| #18 Oct 17 17:35 — No Changes | 3ms | 7s |
| #17 Oct 17 17:34 — No Changes | 3ms | 7s |

# Maven Plugin

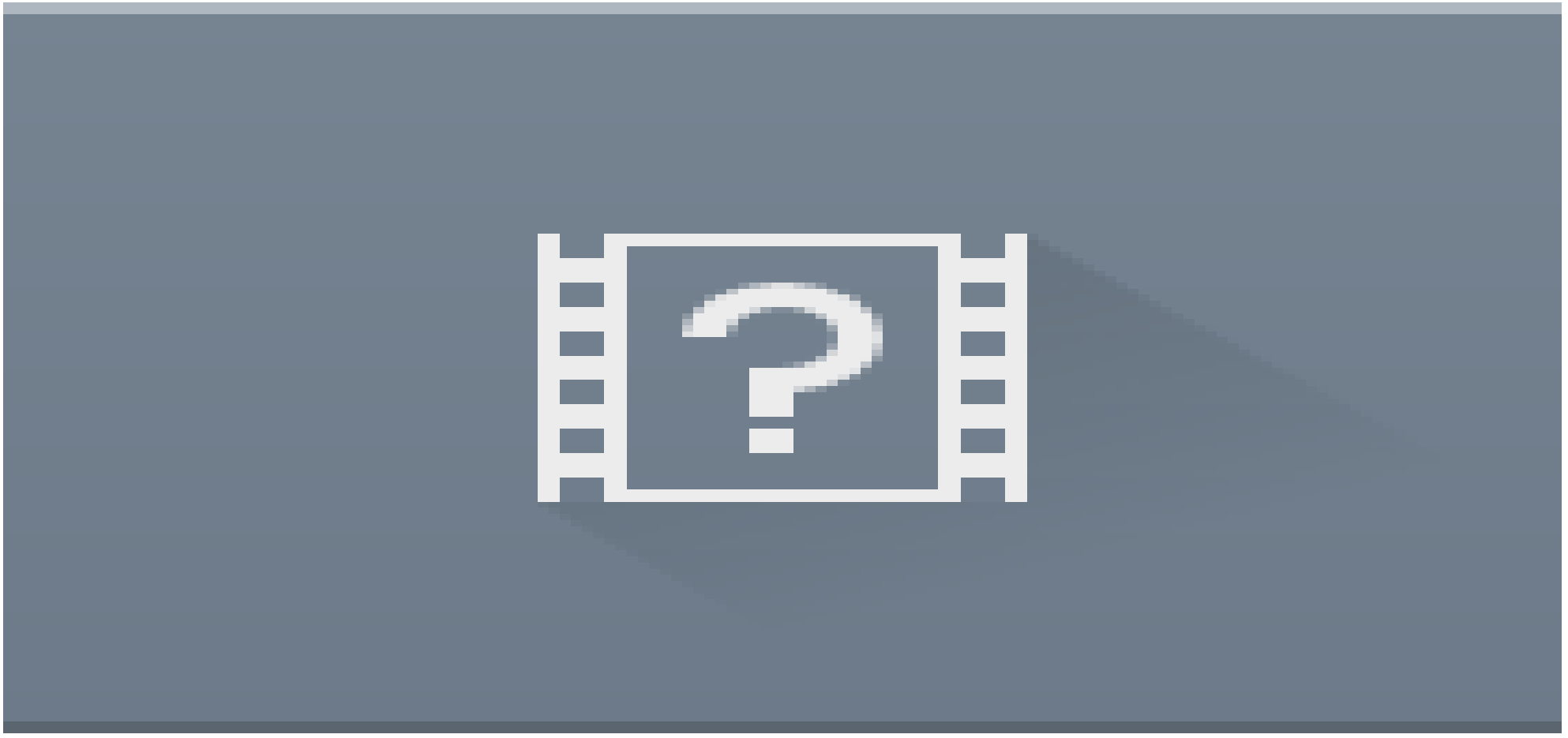


# Maven Plugin

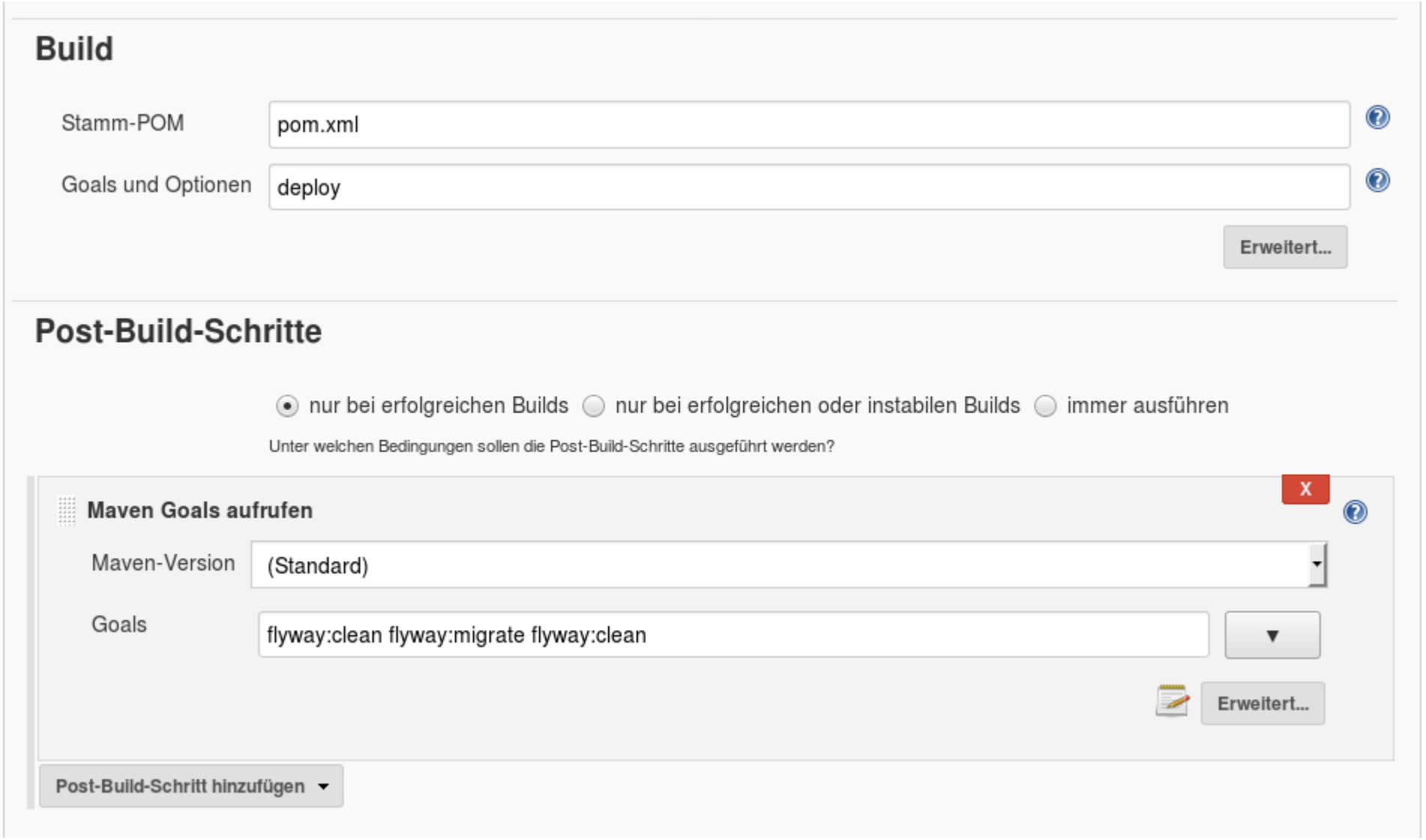
pom.xml

```
<build>
    <plugins>
        <plugin>
            <groupId>org.flywaydb</groupId>
            <artifactId>flyway-maven-plugin</artifactId>
            <version>${flyway.version}</version>
            <configuration>
                <schemas>
                    <schema>flyway_demo</schema>
                </schemas>
                <user>flyway</user>
                <password>flyway</password>
                <url>jdbc:mysql://192.168.33.10:3306</url>
            </configuration>
        </plugin>

    </plugins>
</build>
```

# Maven Plugin



# Maven Plugin



# Migrationstests mit JUnit und Testcontainers

# Migrationstests mit JUnit und Testcontainers

```
public class DbMigrationITest {

    @Rule
    public MySQLContainer mysqlDb = new MySQLContainer();

    @Test
    public void testDbMigrationFromTheScratch(){
        Flyway flyway = new Flyway();
        flyway.setDataSource(mysqlDb.getJdbcUrl(), mysqlDb.getUsername(), mysqlDb.getPassword())

        flyway.migrate();
    }
```

# Migrationstests mit JUnit und Testcontainers

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running db.migration.DbMigrationITest
INFO  - ertyClientProviderStrategy - Found docker client settings from environment
INFO  - ckerClientProviderStrategy - Found Docker environment with Environment variables, system properties and defaults. Resolved:
    dockerHost=unix:///var/run/docker.sock
    apiVersion='{UNKNOWN_VERSION}'
    registryUrl='https://index.docker.io/v1/'
    registryUsername='sparsick'
    registryPassword='null'
    registryEmail='null'
    dockerConfig='DefaultDockerClientConfig[dockerHost=unix:///var/run/docker.sock,registryUsername=sparsick,registryPassword=<null>,registryEma

INFO  - DockerClientFactory        - Docker host IP address is localhost
INFO  - DockerClientFactory        - Connected to docker:
  Server Version: 17.05.0-ce
  API Version: 1.29
  Operating System: Linux Mint 18.2
  Total Memory: 19511 MB
       ℹ Checking the system...
       ✔ Docker version is newer than 1.6.0
       ✔ Docker environment has more than 2GB free
       ✔ File should be mountable
       ✔ Exposed port is accessible
INFO  - 🐳 [mysql:latest]          - Creating container for image: mysql:latest
INFO  - 🐳 [mysql:latest]          - Starting container with ID: 2668be66c2631e49b5bcb4e180665d223525ec896ea78034326076d5f9063d53
INFO  - 🐳 [mysql:latest]          - Container mysql:latest is starting: 2668be66c2631e49b5bcb4e180665d223525ec896ea78034326076d5f9063d53
INFO  - 🐳 [mysql:latest]          - Waiting for database connection to become available at jdbc:mysql://localhost:32769/test using query 'SELECT
INFO  - 🐳 [mysql:latest]          - Obtained a connection to container (jdbc:mysql://localhost:32769/test)
INFO  - 🐳 [mysql:latest]          - Container mysql:latest started
INFO  - VersionPrinter             - Flyway 4.0.3 by Boxfuse
INFO  - DbSupportFactory           - Database: jdbc:mysql://localhost:32769/test (MySQL 5.7)
INFO  - DbValidate                 - Successfully validated 2 migrations (execution time 00:00.011s)
INFO  - MetaDataTableImpl          - Creating Metadata table: `test`.`schema_version`
INFO  - DbMigrate                  - Current version of schema `test`: << Empty Schema >>
INFO  - DbMigrate                  - Migrating schema `test` to version 1.0.0 - create person table
INFO  - DbMigrate                  - Migrating schema `test` to version 2.0.0 - add column job title
INFO  - DbMigrate                  - Successfully applied 2 migrations to schema `test` (execution time 00:00.133s).
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 13.9 sec
```

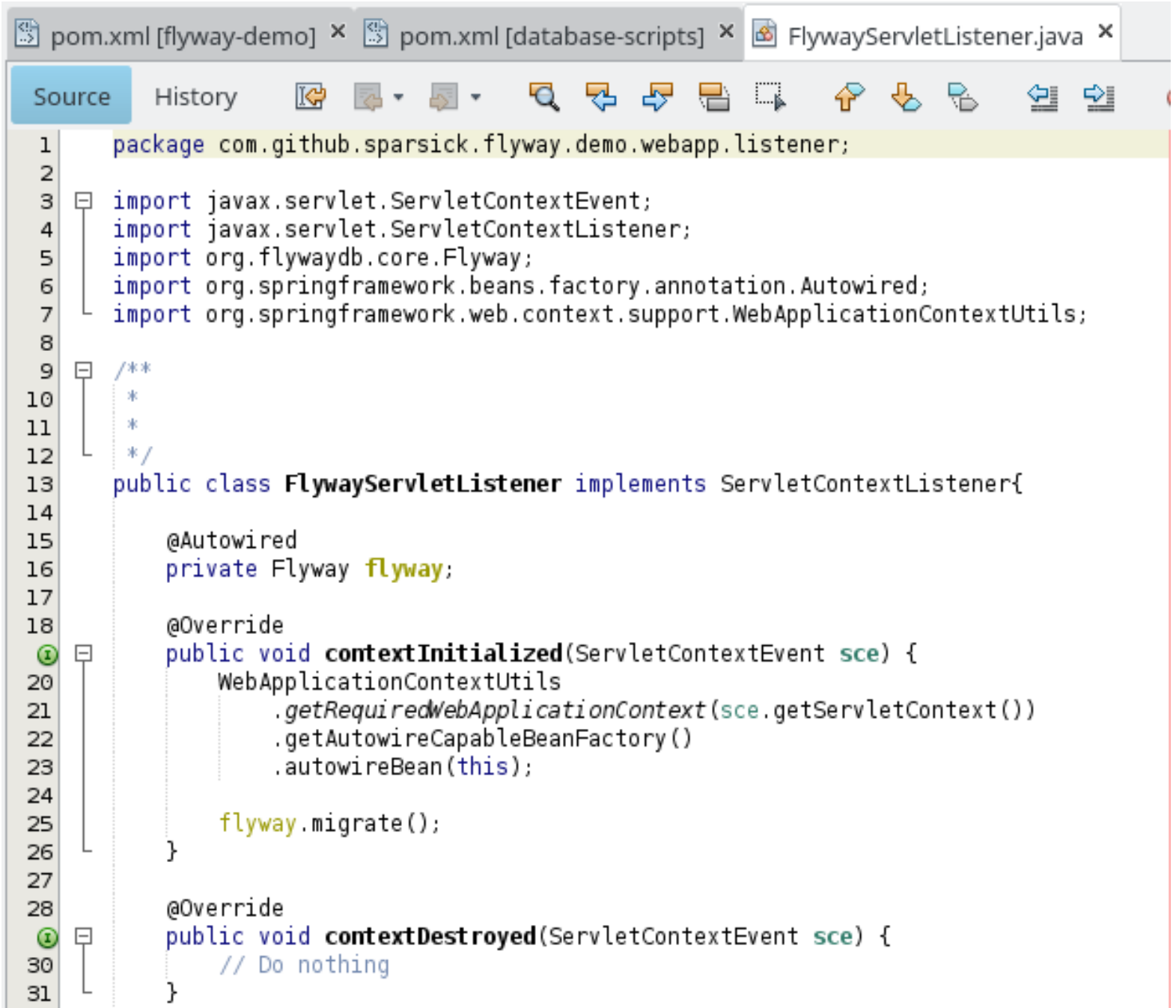
# Integrationstest für die Persistenzschicht

```java
public class PersonRepositoryITest {

    @Rule
    public MySQLContainer mysqlDb = new MySQLContainer();

    @Test
    public void saveAndLoadAPerson() {
        Flyway flyway = new Flyway();
        flyway.setDataSource(mysqlDb.getJdbcUrl(), mysqlDb.getUsername(), mysqlDb.getPassword());
        flyway.migrate();

        PersonRepository personRepositoryUnderTest = new PersonRepository(flyway.getDataSource());
        Person person = new Person("Alice", "Bob");
        personRepositoryUnderTest.save(person);

        List<Person> persons = personRepositoryUnderTest.findAllPersons();

        assertThat(persons.size(), Is.is(1));
        assertThat(persons.get(0), Is.is(person));
    }
```

# Testcontainers

- Temporary database containers - spezielle MySQL, PostgreSQL, Oracle XE und Virtuoso container
- Webdriver containers - Dockerized Chrome oder Firefox browser für Selenium/Webdriver Operationen mit automatischer Videoaufnahme
- Generic containers – irgendein Docker Container
- Docker compose – Wiederverwendung von Docker Compose YAML Datei
- Dockerfile containers – Container direkt von einem Dockerfile

# Java API


pom.xml [flyway-demo]
pom.xml [database-scripts]
FlywayServletListener.java
Source
History
1 package com.github.sparsick.flyway.demo.webapp.listener;
2
3 import javax.servlet.ServletContextEvent;
4 import javax.servlet.ServletContextListener;
5 import org.flywaydb.core.Flyway;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.web.context.support.WebApplicationContextUtils;
8
9 /**
10 *
11 *
12 */
13 public class FlywayServletListener implements ServletContextListener{
14
15 @Autowired
16 private Flyway flyway;
17
18 @Override
20 public void contextInitialized(ServletContextEvent sce) {
21 WebApplicationContextUtils
22 .getRequiredWebApplicationContext(sce.getServletContext())
23 .getAutowireCapableBeanFactory()
24 .autowireBean(this);
25 flyway.migrate();
26 }
27
28 @Override
30 public void contextDestroyed(ServletContextEvent sce) {
31 // Do nothing
}

# Java API

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframewor
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
    <context:annotation-config/>
    <context:component-scan base-package="com.github.sparsick.flyway.demo.webapp"/>

    <bean id="wicketApplication" class="com.github.sparsick.flyway.demo.webapp.WicketApplication"/>

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
          destroy-method="close">
        <property name="url" value="jdbc:mysql://192.168.33.10:3306/flyway_demo" />
        <property name="username" value="flyway" />
        <property name="password" value="flyway" />
    </bean>

    <bean id="flyway" class="org.flywaydb.core.Flyway">
        <property name="dataSource">
            <bean class="org.apache.commons.dbcp.BasicDataSource" parent="dataSource">
                <property name="url" value="jdbc:mysql://192.168.33.10:3306"/>
            </bean>
        </property>
        <property name="schemas">
            <list>
                <value>flyway_demo</value>
            </list>
        </property>
    </bean>
</beans>
```

Spring Context

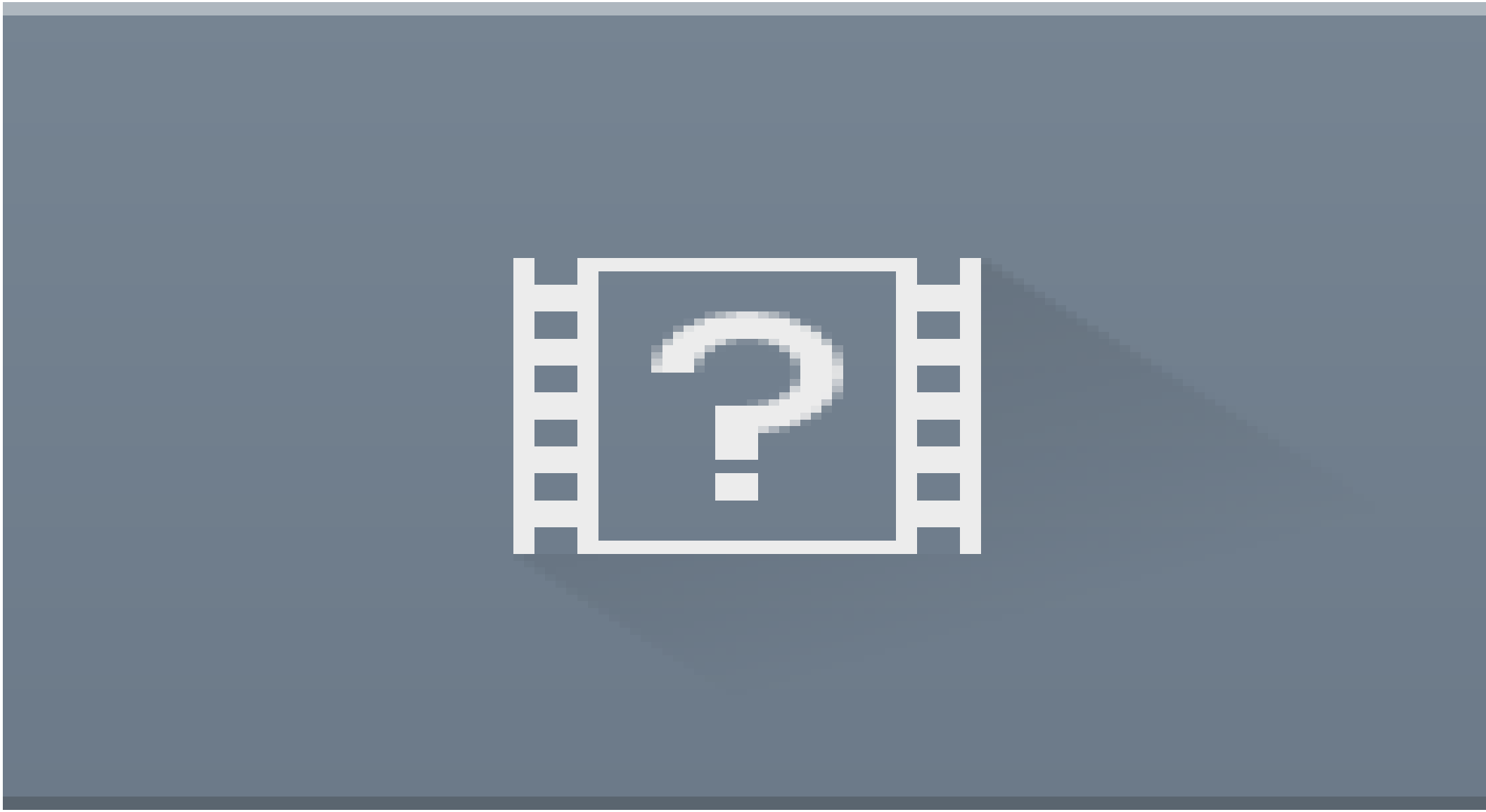
# Java API

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:META-INF/spring/*.xml</param-value>
</context-param>
```

web.xml

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    <listener-class>com.github.sparsick.flyway.demo.webapp.listener.FlywayServletListener</listener-clas
</listener>
```

# Java API



# Aufbau CDBI

- Behandle den Datenbank-Code wie einen ganz normalen Source-Code
  - ➔ Alle Datenbank Artefakte (DDL, DML, Konfigurationen, Testdaten, Stored Procedures, Functions etc) gehören ins VCS. 
  - ➔ Jede Änderung an den DB Artefakten wird getestet. 
- Jeder Entwickler hat seine eigene Datenbank / Testdatenbanken ähneln den Produktionsdatenbanken.
  - ➔ Automatisiertes Aufsetzen der Datenbank. 
- Änderungen an der Datenbank sind nachvollziehbar.
  - ➔ Historie der Änderungen 

# Vergleich mit Liquibase

| Migration types |  | Liquibase |
|---|---|---|
| Plain Old Sql migrations | ✔ | ⊘[1] |
| Java migrations | ✔ | ⊘[1] |
| Xml migrations | ⊘ | ✔ |
| Repeatable migrations | ✔ | ✔ |
| DDL abstraction DSL | ⊘ | ✔ |

1. Sql files and Java classes can be used indirectly through references in xml migrations

Reference: flywaydb.org

| Execution |  | Liquibase |
|---|---|---|
| Command-line | ✔ | ✔ |
| API (Java) | ✔ | ✔ |
| API (Android) | ✔ | ⊘ |
| Maven | ✔ | ✔ |
| Gradle | ✔ | ✔[2] |
| Ant | ✔ | ✔ |
| SBT | ✔ | ✔[2] |

2. Not out of the box. Available through a 3rd party. May be outdated.

Reference: flywaydb.org

| Other |  | Liquibase |
|---|---|---|
| Auto creation of schema | ✔ | ⊘ |
| Auto creation of schema history table | ✔ | ✔ |
| Cluster-safe | ✔ | ✔ |
| Checksum validation | ✔ | ✔ |
| Placeholder replacement | ✔ | ✔ |
| Multiple schema support | ✔ | ⊘ |
| Clean existing schema | ✔ | ⊘ |
| Output to SQL file | ⊘ | ✔ |
| Available on Maven Central | ✔ | ✔ |
| License | Apache v2 | Apache v2 |

Reference: flywaydb.org

# Fallstricke

# Keine Instanz-spezifischen Daten

**Beispiel**

```
1
2   GRANT SELECT, INSERT ON usermgm.* TO
3   `technical-user`@'192.168.33.10' IDENTIFIED BY 'pA$$w0rt';
4
```

# Keine Instanz-spezifischen Daten

**Möglicher Lösungsansatz:**

```
1
2    GRANT SELECT, INSERT ON usermgm.* TO
3    `technical-user`@'*' IDENTIFIED BY 'pA$$w0rt';
4
```

- Zugriffskontrolle über eine Firewalls (iptables)

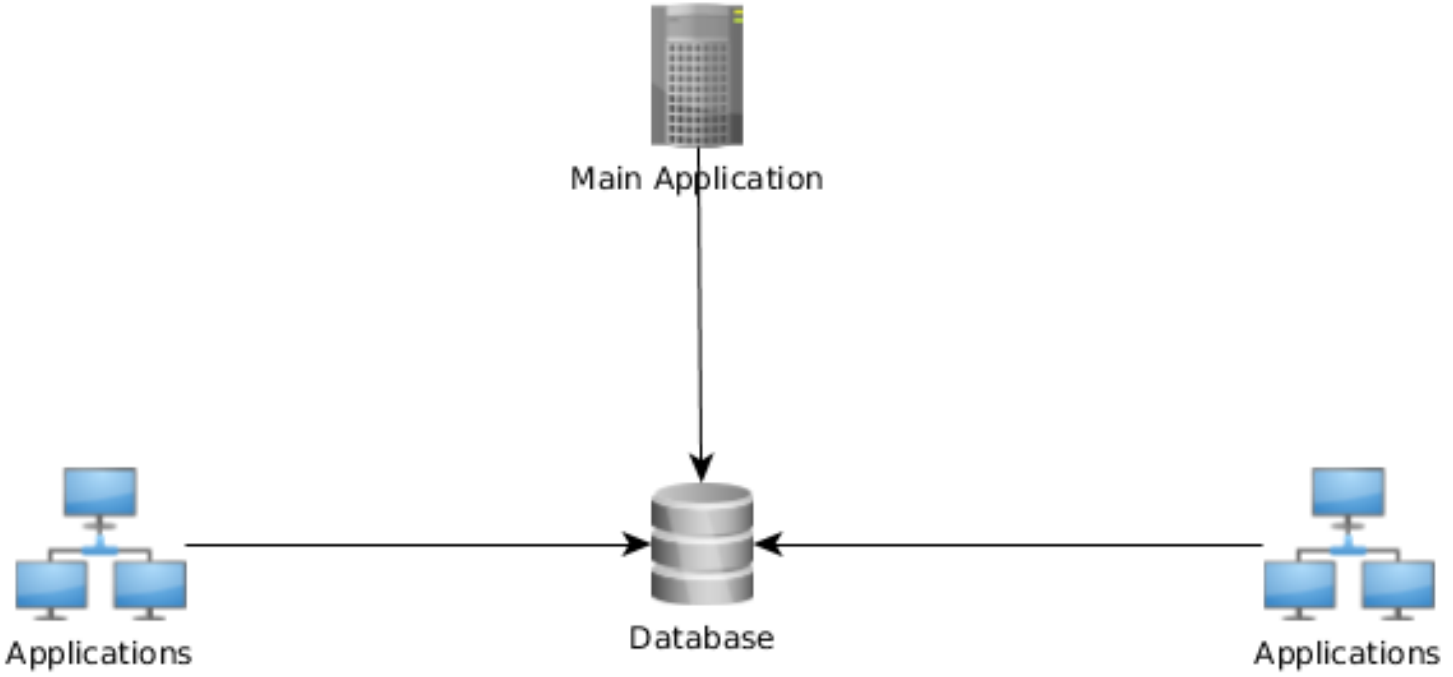
# Keine Instanz-spezifischen Daten

**Möglicher Lösungsansatz:**

```
1    GRANT SELECT, INSERT ON usermgnt.* TO
2    'technical-user' @ '${address}' By '${password}';
3
4
```

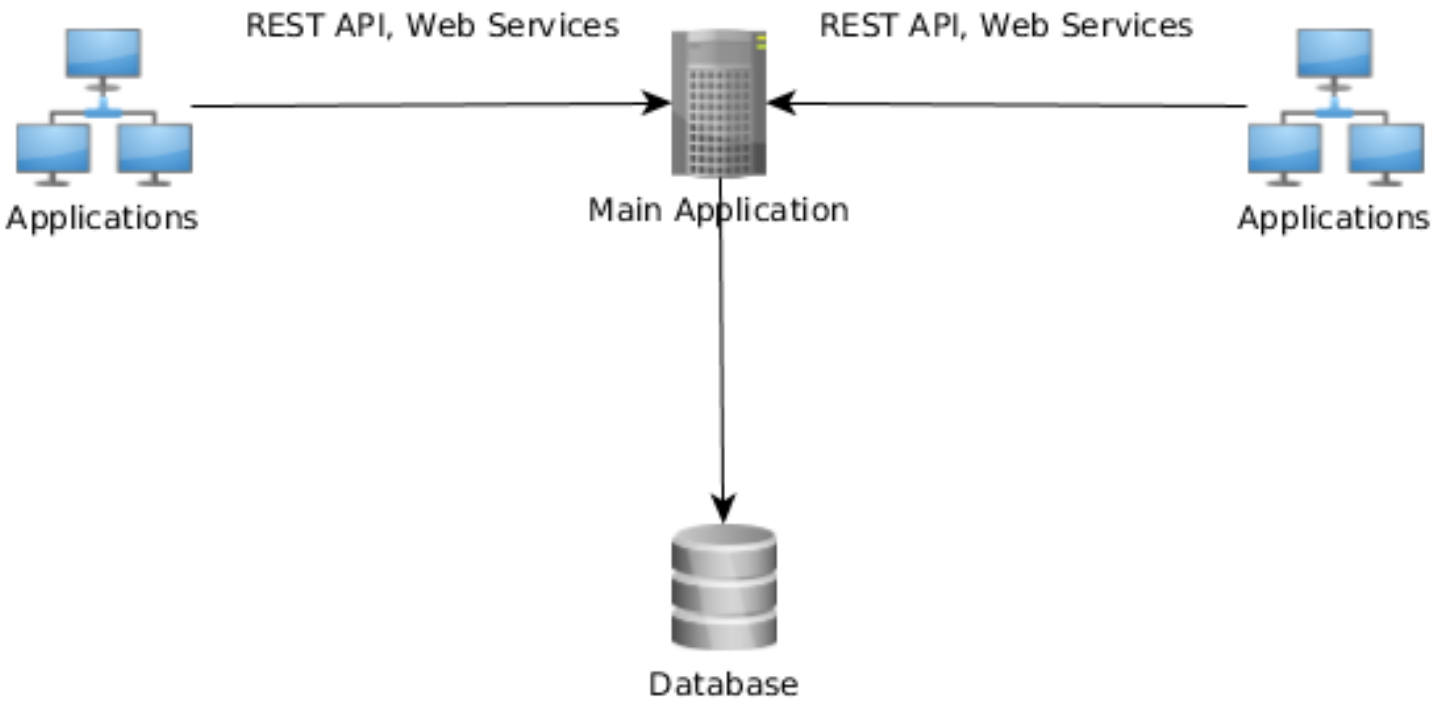
# Datenbank wird von mehreren Applikationen benutzt

**Ausgangslage :**



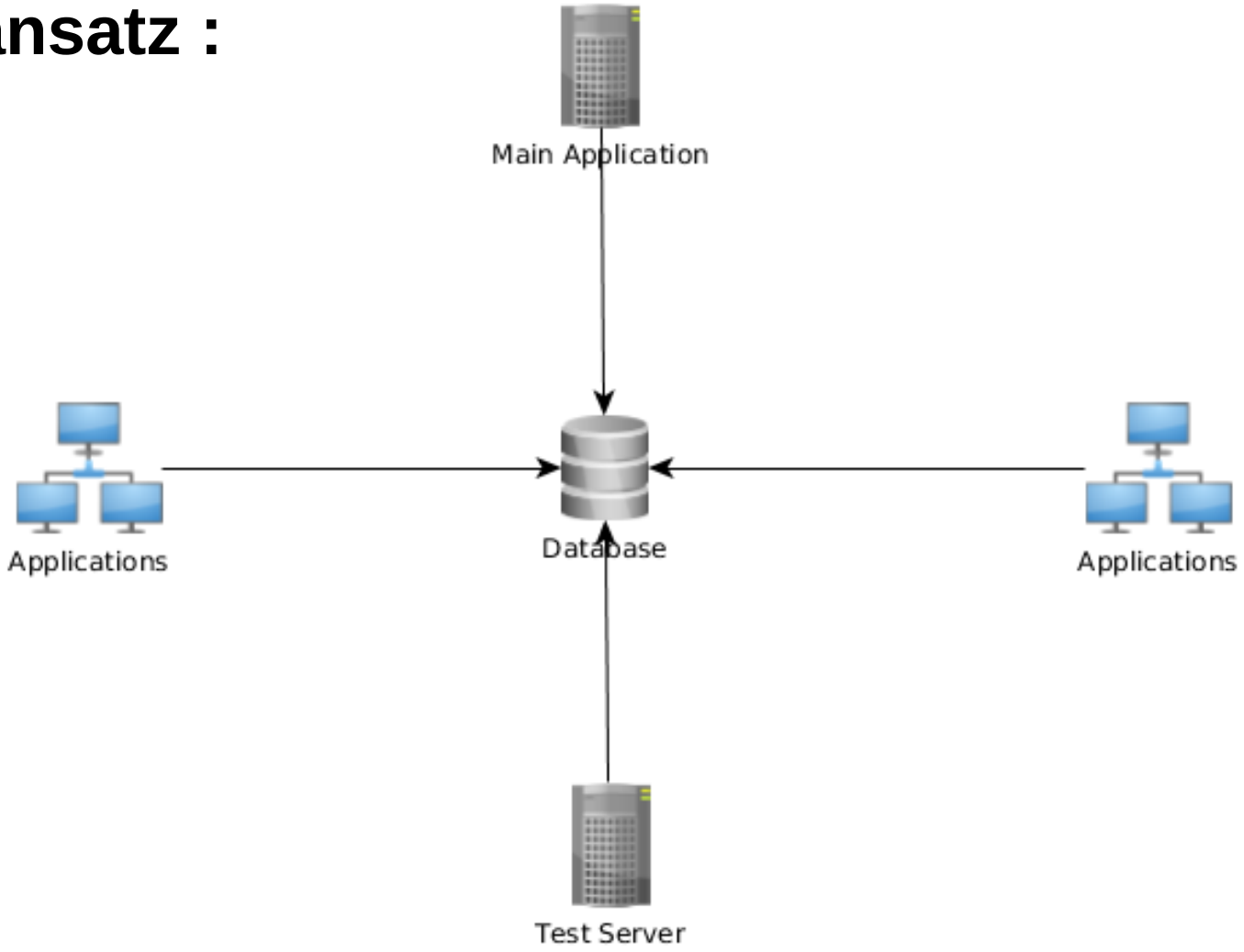
# Datenbank wird von mehreren Applikationen benutzt

**Lösungsansatz :**

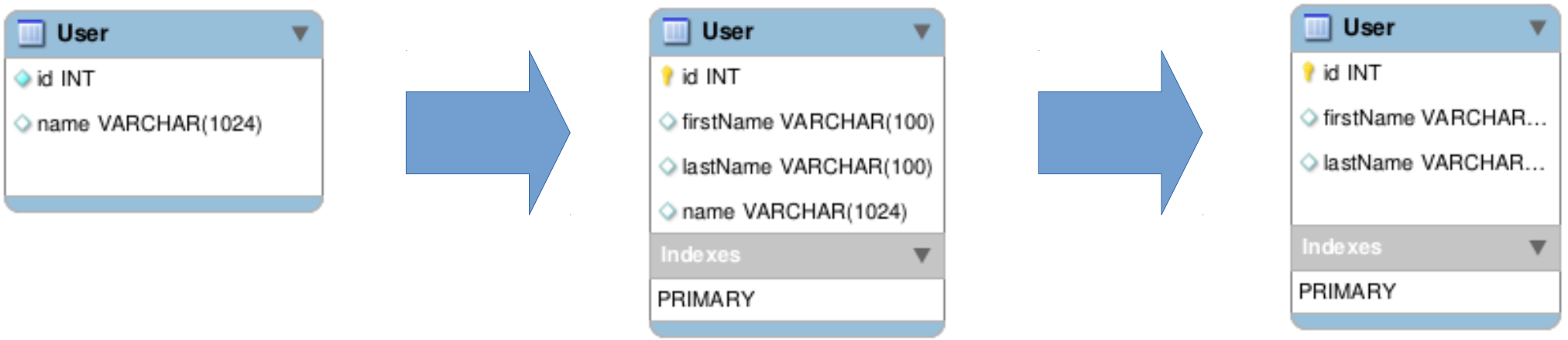


# Datenbank wird von mehreren Applikationen benutzt

**Lösungsansatz :**



# Datenbank wird von mehreren Applikationen benutzt



# Weitere Fallstricke (Auszug)

- Datenänderung dauern zu lange
- Datenlöschung
- Faktor Mensch
- ...

# Weitere Informationen

- Continuous Integration von Paul M. Duvall, Steve Matyas und Andrew Glover
- Refactoring Databases: Evolutionary Database Design von Scott J. Ambler und Pramodkumar J. Sadalage
- Flyway Documentation
  http://flywaydb.org/documentation/migration/
  http://flywaydb.org/getstarted/
- Source code:
  https://github.com/sparsick/flyway-talk

# Fragen?

https://github.com/sparsick/flyway-talk
mail@sandra-parsick.de
@SandraParsick