Zürcher Hochschule
für Angewandte Wissenschaften

zh aw School of Engineering

Bachelor of Science (BSc) in Informatik

Modul Advanced Software Engineering 1 (ASE2)

LE 02 – Spring Framework

Spring Boot
Datenbank Migration und Refaktorisierung

Institut für Angewandte Informationstechnologie (InIT)

Walter Eich (eicw) / Matthias Bachmann (bacn)

https://www.zhaw.ch/de/engineering/institute-zentren/init/

Lernziele LE 02 – Spring Framework

- Die Studierenden
  - können mittels Flyway und Liquibase eine Datenbank Refaktorisierung durchführen

Initialisierung der Datenbank
Migration der Datenbank

- Mittels Database Bootstrap in Java
  - Hands-on: Definition eines Beans
  - Hands-on: Bootstrap mittels Java Code
- Mittels SQL Dateien
- Datenbank Migration mit FlyWay
  - Hands-on: Flyway einbauen
- Datenbank Migration mit Liquibase
  - Hands-on: Liquibase

Initialisierung der Datenbank

- Datenbank-Seed oder Bootstrap
  - Automatisiert über den **JPA Provider und ORM** (nur für Entwicklung geeignet) Keine Versionsmigration möglich
  - Code-getrieben (Java-Code) Database Bootstrap in Java (geeignet für Daten)
    - Beispiel: 12-spring-boot-jpa-flyway und 13-spring-boot-jpa-liquibase
  - Mittels SQL Scripts für Schema und Daten
    - **data.sql** file or (data-**h2**.sql, data-**mysql**.sql) in src/main/resources
    - **schema.sql** or (schema-h2.sql) in src/main/resources
    - spring.jpa.hibernate.ddl-auto=none
  - Mittels SQL-Skripts (https://flywaydb.org/)
    - Unterstützung von **Versionen – Refaktorisierung der DB**
    - Beispiel: **12-spring-boot-jpa-flyway**
  - Mittels Change Sets (http://www.liquibase.org/)
    - Unterstützung von **Versionen– Refaktorisierung der DB**
    - Formate: xml, yaml, json, sql
    - Beispiel: **13-spring-boot-jpa-liquibase**

Grundlage für Datenbank Refaktorisierung

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

- Verwenden Sie als Grundlage das Projekt Hello-Rest

  https://github.zhaw.ch/bacn/ase2-spring-boot-hellorest

- Lösungsprojekte
  - Database Bootstrap
    https://github.zhaw.ch/bacn/ase2-spring-boot-hellorest/blob/master/readme/database-bootstrap.md
  - FlyWay:
    https://github.zhaw.ch/bacn/ase2-spring-boot-hellorest/blob/master/readme/flyway.md
  - Liquibase:
    https://github.zhaw.ch/bacn/ase2-spring-boot-hellorest/blob/master/readme/liquibase.md

Hands-on: Definition eines Beans

- Die @Configuration Annotation bewirkt, dass beim Booten von Spring die Bean aufgerufen wird
- Die Klasse **`DatabaseBootstrap`** inplementiert das Interface **`InitializingBean`** und die Methode **`afterPropertiesSet`**
  - Sicherstellen dass, die Methode erst nach der Initialisierung ausgeführt wird

```
@Configuration
public class DevConfiguration {

    @Bean
    public DatabaseBootstrap databaseBootstrap() {
        return new DatabaseBootstrap();
    }

}
```

Hands-on: Bootstrap mittels Java Code

```
public class DatabaseBootstrap implements InitializingBean {

    @Autowired
    CustomerRepository repository;

    private static Logger log = LoggerFactory.getLogger(DatabaseBootstrap.class);

    @Override
    public void afterPropertiesSet() throws Exception {
        if (repository.findByFirstnameAndLastname("Felix", "Muster") == null) {
            Customer customer = new Customer();
            customer.setFirstname("Felix");
            customer.setLastname("Muster");
            repository.save(customer);
            log.info(customer.getFirstname() + " " + customer.getLastname() +
                    " created");
        }
        log.info("Bootstrap finished");

    }
```

data-h2.sql

- Einfügen von Daten in die Datenbank mit Hilfe von SQL

```
insert into `user` (`id`,`email`,`full_name`,`password`) values (1,'admin@admin.ch','admin','admin');
insert into `user` (`id`,`email`,`full_name`,`password`) values (2,'user@user.ch','user','user');
insert into `role` (`id`,`role`) values (1,'ROLE_ADMIN');
insert into `role` (`id`,`role`) values (2,'ROLE_USER');
insert into `user_role` (`user_id`,`role_id`) values (1,1);
insert into `user_role` (`user_id`,`role_id`) values (2,2);
```

```
insert into `author` (`id`,`first_name`,`last_name`,`email`) values
(1,'Felix','Muster','Felix.Muster@example.com');
insert into `author` (`id`,`first_name`,`last_name`,`email`) values
(2,'Matthias','Bachmann','M.Bachmann@united-portal.com');
insert into `author` (`id`,`first_name`,`last_name`,`email`) values
(3,'Wendy','Guthrie','nunc@semper.co.uk');
insert into `author` (`id`,`first_name`,`last_name`,`email`) values
(4,'Yeo','Collins','adipiscing@convallis.net');
insert into `author` (`id`,`first_name`,`last_name`,`email`) values
(5,'Tara','Aguilar','felis.purus.ac@gravidaPraesenteu.co.uk');
insert into `author` (`id`,`first_name`,`last_name`,`email`) values
(6,'Yolanda','Strickland','fringilla.Donec@sitametorci.org');
```

Flyway - Motivation

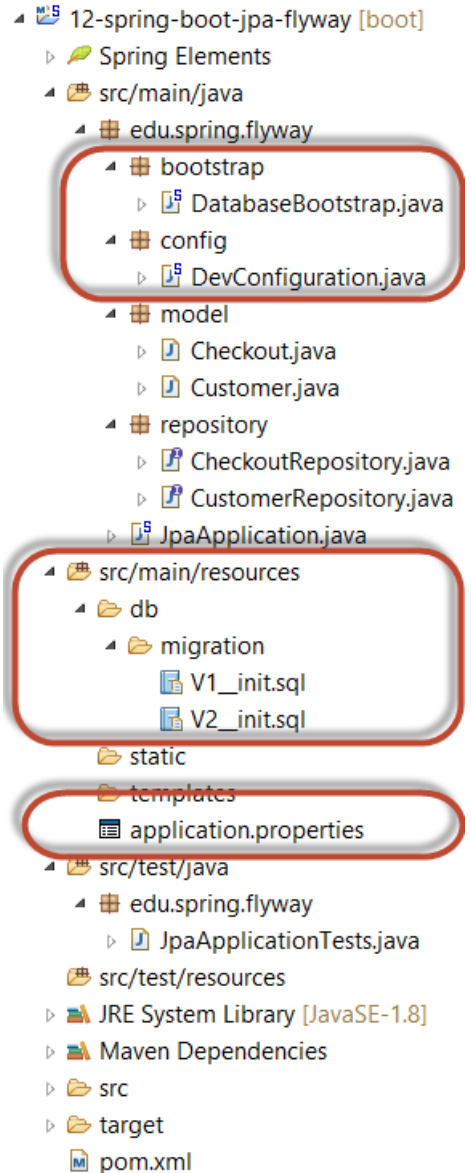
- Alle Entwickler teilen sich eine Testdatenbank
  - Keiner weiss, welche Datenbankskripte auf welchen Datenbankinstanzen ausgeführt wurden
  - Die Testdatenbank unterscheidet sich von der Produktionsdatenbank
- Datenbankmigrationsskripte werden nicht als Teil des Quellcode angesehen
- Produktionsdatenbanken müssen migriert werden: V1.0 -> V2.0
  - Wie macht man das? … Mit Migrationsskripten
  - Besser als integraler Bestandteil eines Projektes
  - Automatische Migration beim Starten der Anwendung
    z.B. Atlassian Jira oder Confluence bei Versionsupdate -> Automigration

Beispiel: HelloRest mit Flyway
Setup: Projektstruktur und pom-Datei

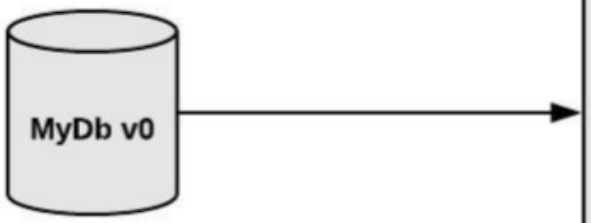
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

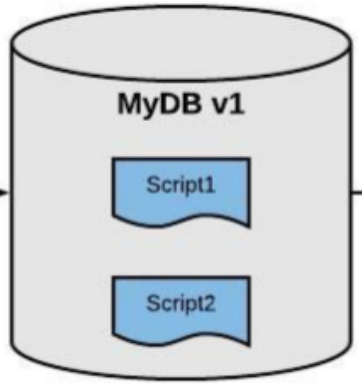
- Für das Setup wird eine Dependency zu flyway-core benötigt

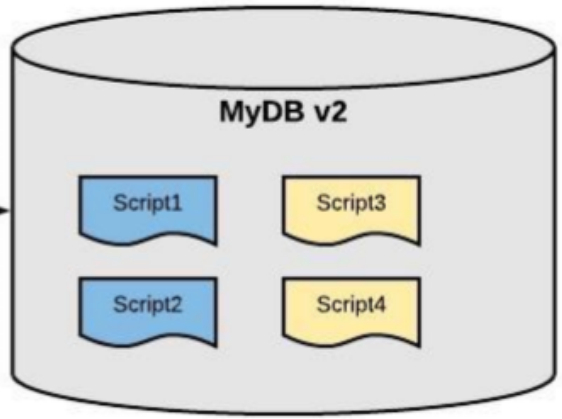
```
<dependency>
  <groupId>org.flywaydb</groupId>
  <artifactId>flyway-core</artifactId>
</dependency>
```



Flyway Motivation







DB_VERSION_CONTROL_TABLE

Script name	Author	Checksum	DBversion

DB_VERSION_CONTROL_TABLE

Script name	Author	Checksum	DBversion
Script1	John	xyz	1
Script 2	Katy	yzx	1

DB_VERSION_CONTROL_TABLE

Script name	Author	Checksum	DBversion
Script1	John	xyz	1
Script2	Katy	yzx	1
Script3	Jack	abc	2
Script4	Anne	hgk	2

Was ist Flyway?

- Migration Framework für Relationales Datenbanken basierend auf Java
- Erstellt eine Datenbank «from Scratch»
- Verwaltet den Stand der Datenbank
- Vier Migrationsmodi
  - SQL Migration
  - Java Migration
  - Versionierte Migration
  - Wiederholbare Migration

	Versioniert	Wiederholbar
SQL-basiert	✔	✔
Java-basiert	✔	✔

- Homepage: http://flywaydb.org
- Benutzung: Maven/Gradle Plugin, Console Command Line Tool, Java API
- Kommandozeilen Befehle: info, clean, validate, repair, migrate, baseline

Was ist Flyway?
Unterstützte Datenbanken

Oracle
10g and later, all editions
(incl. Amazon RDS)

SQL Server
2008 and later
(incl. Amazon RDS)

SQL Azure
latest

MySQL
5.1 and later
(incl. Amazon RDS & Google Cloud SQL)

MariaDB
10.0 and later
(incl. Amazon RDS)

Phoenix
4.2.2 and later

PostgreSQL
9.0 and later
(incl. Heroku & Amazon RDS)

Vertica
6.5 and later

AWS Redshift
latest

DB2
9.7 and later

DB2 z/OS
9.1 and later

Derby
10.8.2.2 and later

H2
1.2.137 and later

Hsql
1.8 and later

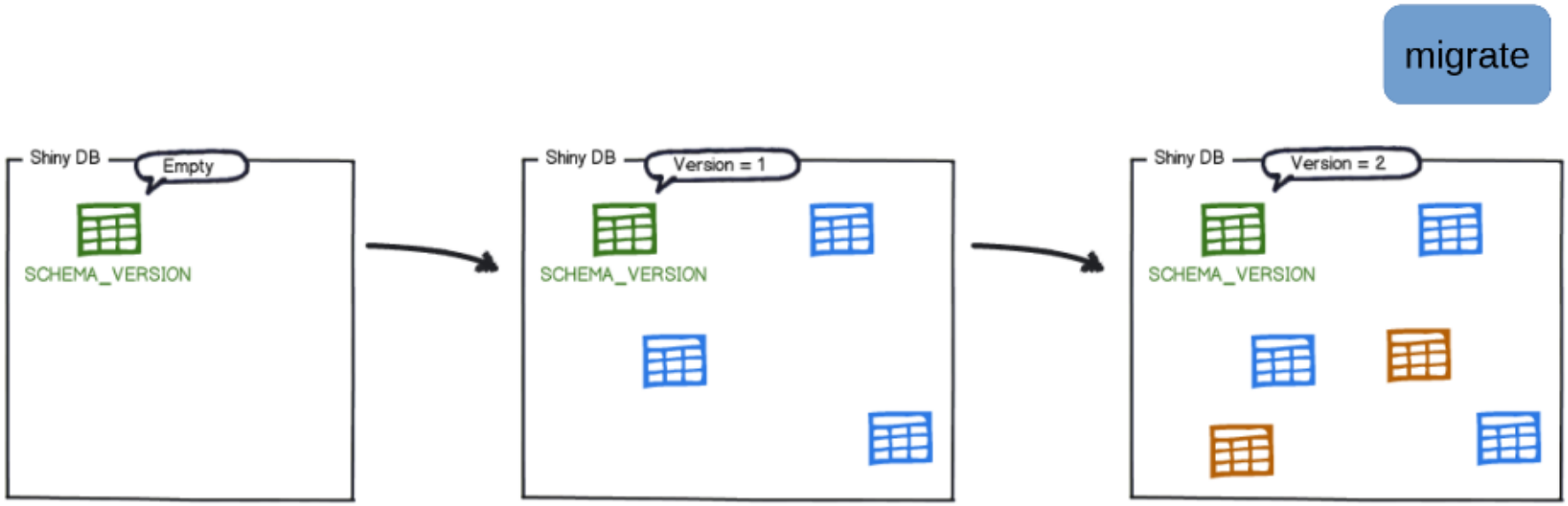
SQLite
3.7.2 and later

SAP HANA
latest

solidDB
6.5 and later

Sybase ASE
12.5 and later

Wie funktioniert Flyway?



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

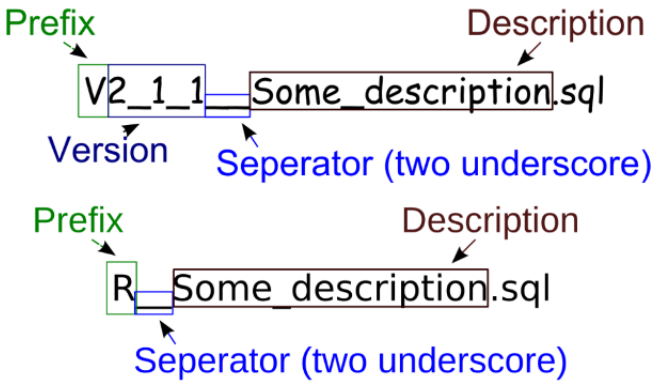
Wie funktioniert Flyway?

- Versionierte Migration
  - Skripte haben eine eindeutige Version
  - Werden genau einmal ausgeführt
  - Anwendungsfall: DDL Skripte
- Wiederholbare Migration
  - Skripte haben keine Versionsummer und werden ausgeführt wenn sich ihre Checksumme ändert und nachdem alle anderen Skript ausgeführt wurden
  - Anwendungsfälle (Wieder-)Erstellung von Views/Procedures bzw. Import

	Versioniert	Wiederholbar
SQL-basiert	✔	✔
Java-basiert	✔	✔

Wie funktioniert Flyway?
SQL Migration

- SQL-Migration
  - Einfache Datenänderung bzw. DDL Änderungen
  - Benennung (siehe Grafik)
  - Location: **src/main/resources/db/migration**
- Syntax
  - Statement kann über mehrere Zeilen gehen
  - Platzhaltersupport
  - Kommentare: Single (-) oder Multiline (/* */)
- Beispiel

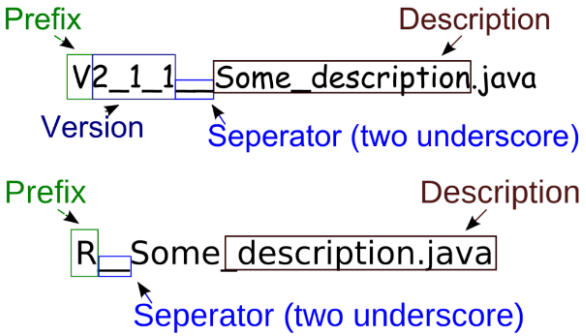



```sql
/* Create a table for person */

Create table person (
    first_name varchar(128),
    last_name varchar(128)
);
```

Wie funktioniert Flyway? Java-Migration

- Typische Anwendungsfälle
  - BLOB & CLOB Änderungen
  - Fortgeschrittene Änderungen an Massendaten (Neuberechnen, Formate, etc)
  - Location: src/mainjava/db/migration
- Benennung der Klassen
- Beispiel:




```
public class V1_2__Another_user implements JdbcMigration {
    public void migrate(Connection connection) throws Exception {
        PreparedStatement statement =
            connection.prepareStatement("INSERT INTO test_user (name) VALUES ('Obelix')");

        try {
            statement.execute();
        } finally {
            statement.close();
        }
    }
}
```

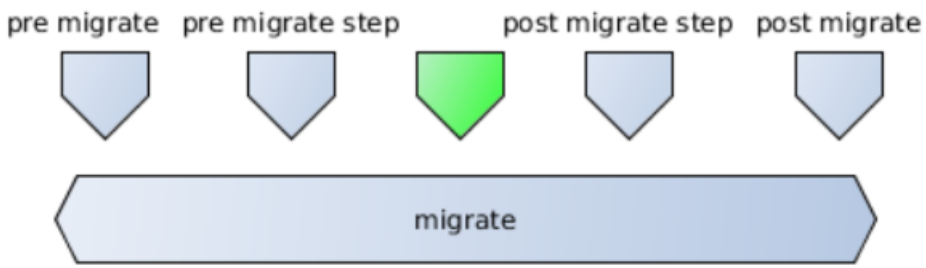
Wie funktioniert Flyway? Flyway und Spring

- Spring erkennt, dass die Library vorhanden ist
- Der Default-Pfad für die Migrations-Dateien ist
  - src/main/resources/db/migration bzw src/main/java/db/migration
- Im Default-Pfad befinden sich die einzelnen sql-Migrationsskripte
  - Beispiel: V1_init.sql und V2_init.sql
  - Falls die DB in einer Datei persistiert wird, muss bei Änderung der Skripte die DB gelöscht werden.
- Beispiel Java-Migration

```
public class V1_2__Another_user implements SpringJdbcMigration {
    public void migrate(JdbcTemplate jdbcTemplate) throws Exception {
        jdbcTemplate.execute("INSERT INTO test_user (name) VALUES ('Obelix')");
    }
}
```

Wie funktioniert Flyway?
Flyway Lifecycle

- SQL Callback Skripte werden anhand ihrer Namen erkannt
  - BeforeMigrate.sql
  - AfterMigrate.sql
- Java Callback
  - org.flywaydb.core.api.callback.FlywayCallback



beforeMigrate	Before Migrate runs
beforeEachMigrate	Before every single migration during Migrate
afterEachMigrate	After every single migration during Migrate
afterMigrate	After Migrate runs
beforeClean	Before Clean runs
afterClean	After Clean runs
beforeInfo	Before Info runs
afterInfo	After Info runs
beforeValidate	Before Validate runs
afterValidate	After Validate runs
beforeBaseline	Before Baseline runs
afterBaseline	After Baseline runs
beforeRepair	Before Repair runs
afterRepair	After Repair runs

Flyway Migrationsskripte

- V1__init.sql

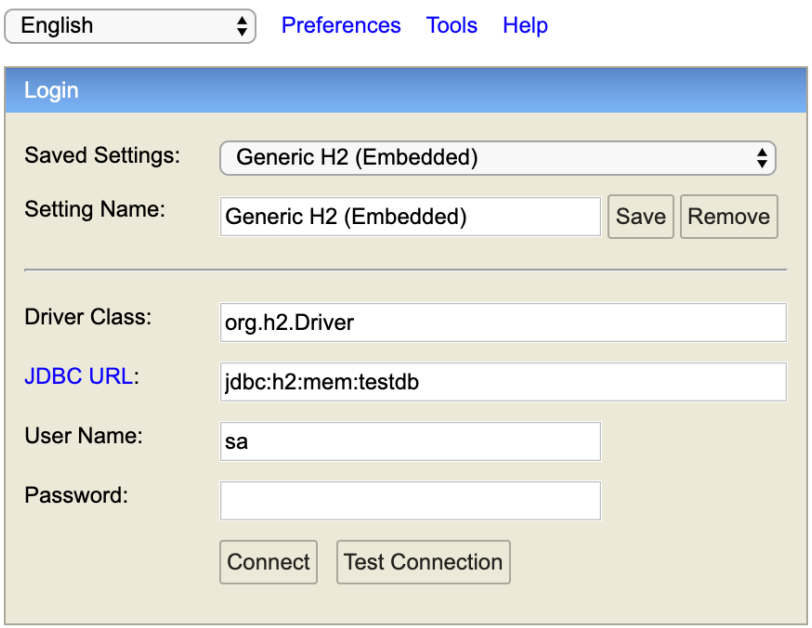
```
CREATE TABLE CUSTOMER (
  id BIGINT GENERATED BY DEFAULT AS IDENTITY,
  firstname varchar(255) not null,
  lastname varchar(255) not null
);
CREATE TABLE CHECKOUT (
  id BIGINT GENERATED BY DEFAULT AS IDENTITY,
  customer_id BIGINT
);
```

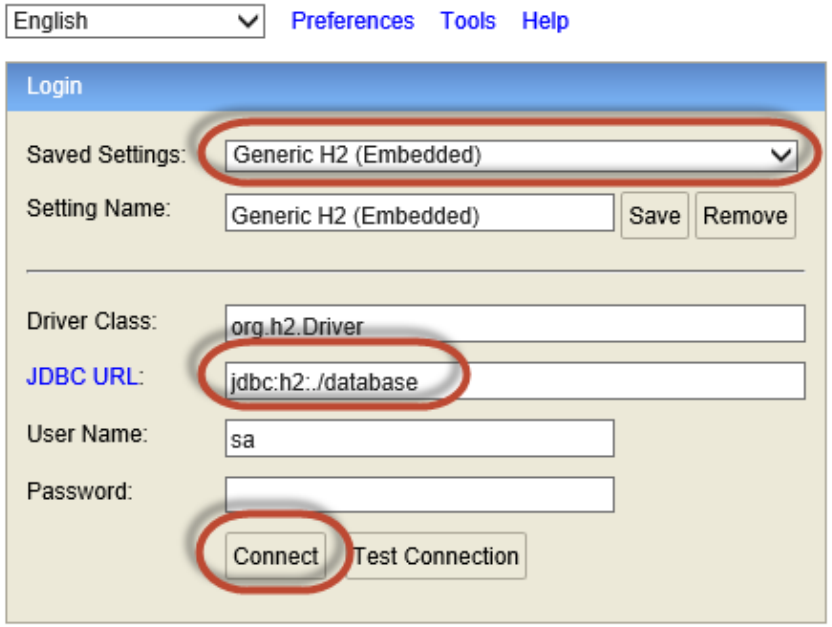
- V2__init.sql

```
insert into CUSTOMER (firstname, lastname) values ('Matthias', 'Bachmann');
```

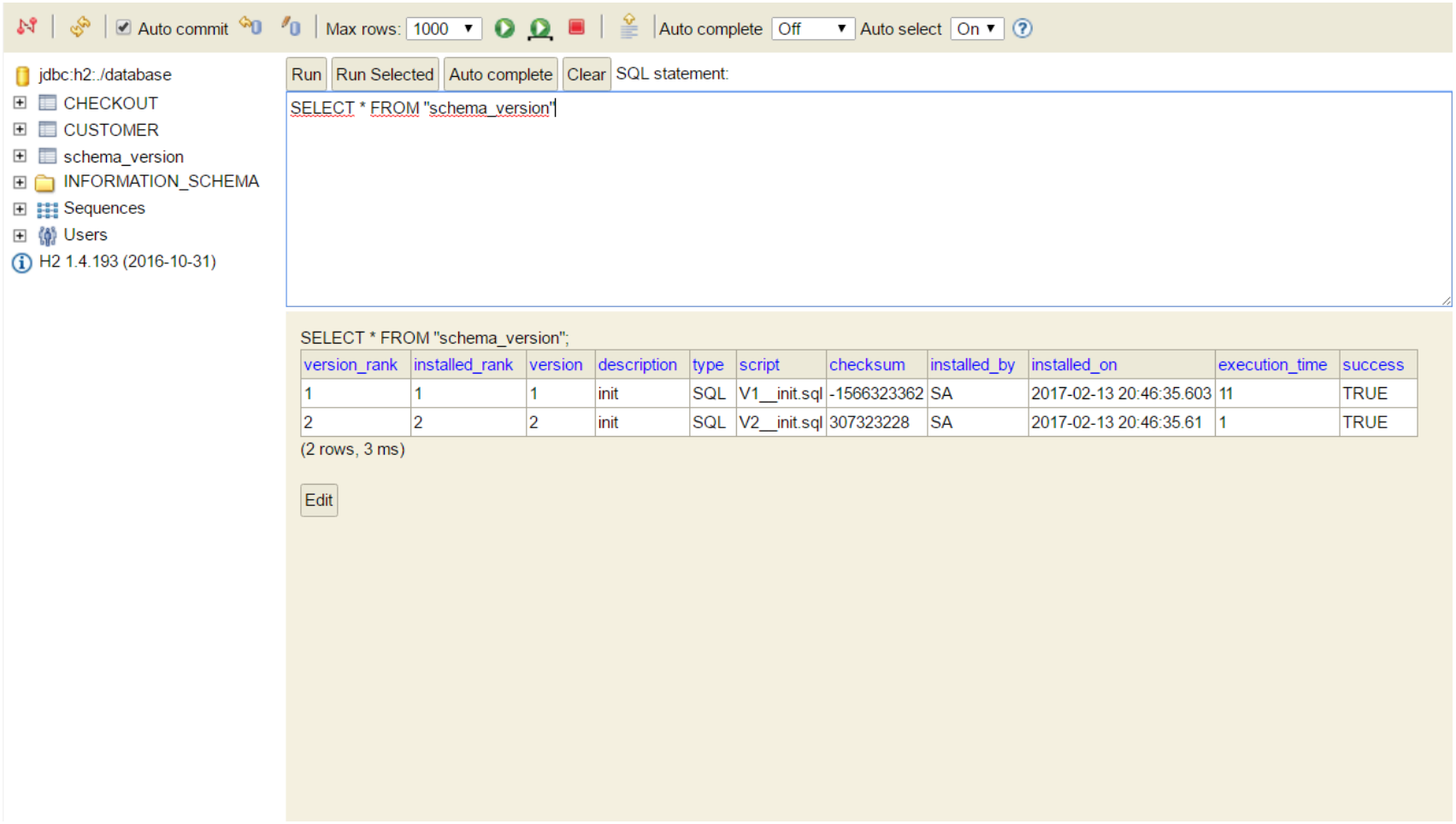
Datenbank Konsole (1) Flyway

- http://localhost:8080/h2-console
- `jdbc:h2:mem:testdb` oder `jdbc:h2:./database` (mit Datei)
  - (`./database` Dateiname wie in `application.properties`)





Datenbank Konsole (2) - Flyway



version_rank	installed_rank	version	description	type	script	checksum	installed_by	installed_on	execution_time	success
1	1	1	init	SQL	V1__init.sql	-1566323362	SA	2017-02-13 20:46:35.603	11	TRUE
2	2	2	init	SQL	V2__init.sql	307323228	SA	2017-02-13 20:46:35.61	1	TRUE

Hands-on: Flyway einbauen

- Dateien db/migration
  https://github.zhaw.ch/bacn/ase-spring-boot-01/tree/master/12-spring-boot-jpa-flyway/src/main/resources/db/migration
- Anpassen von application.properties
  https://github.zhaw.ch/bacn/ase-spring-boot-01/blob/master/12-spring-boot-jpa-flyway/src/main/resources/application.properties
- Anpassen der pom Datei

```
<dependency>
  <groupId>org.flywaydb</groupId>
  <artifactId>flyway-core</artifactId>
</dependency>
```

Was ist Liquibase?

- Java-basiertes Werkzeug zur automatisierten Migration von relationalen Datenbank-Schemata
- Integration in den Entwicklungs-, Build- und Auslieferungsprozess via…
  - Maven
  - Ant
  - Grails
  - Servlet Listener
  - Startup der Anwendung
  - Java API
  - Kommandozeile
- Unterstützung gängiger Datenbanken: Oracle, MySQL, MSSQL, PostgreSQL, DB2, Apache Derby, H2, etc.

(http://www.liquibase.org/)

Setup

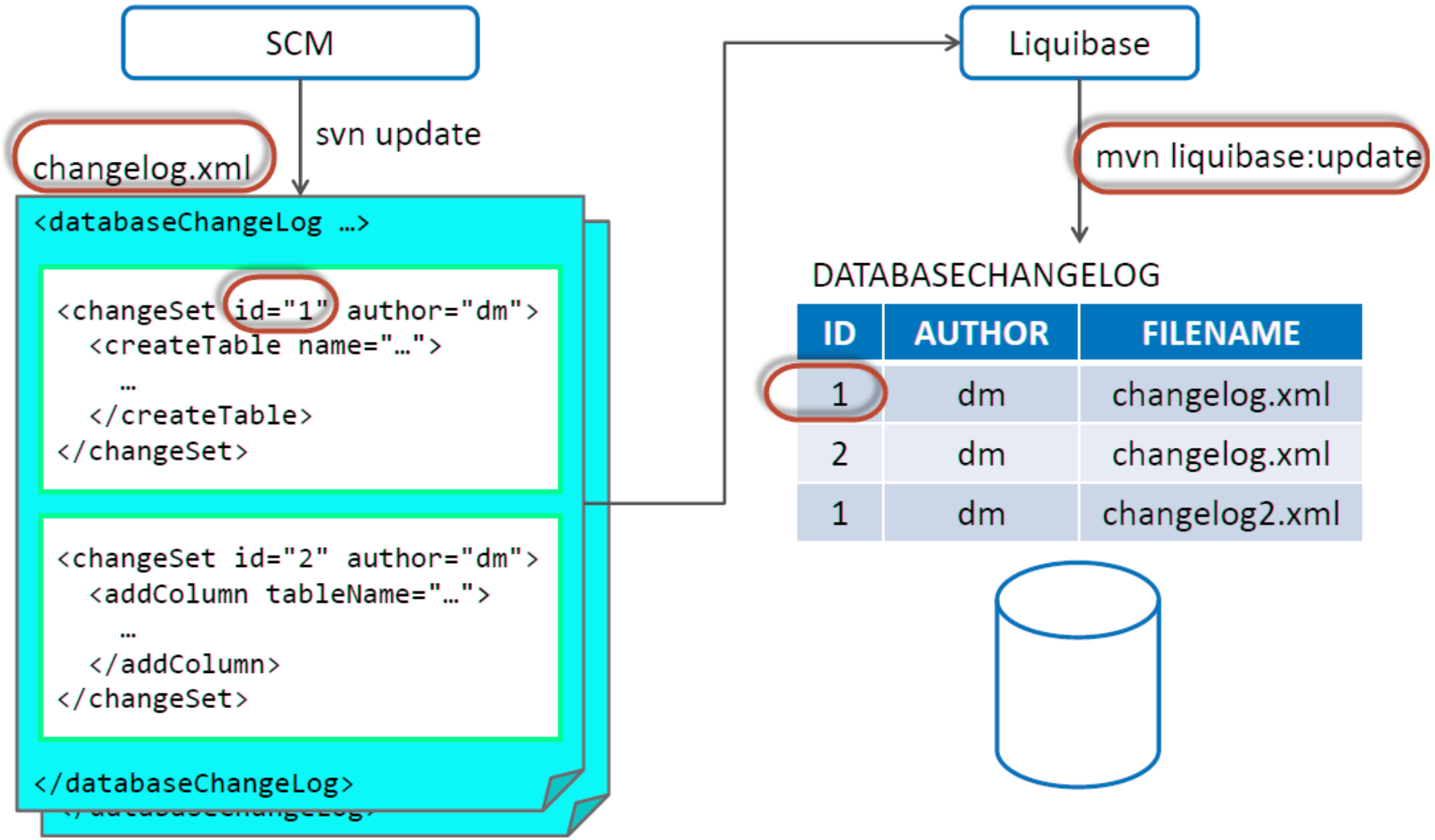
- Für Liquibase wird die `liquibase-core` Dependency benötigt

```
<dependency>
   <groupId>org.liquibase</groupId>
   <artifactId>liquibase-core</artifactId>
   <version>4.3.1</version>
</dependency>
```

- Für die Ausführung von generateChangeLog oder Diff wird ein Plugin benötigt

```
<plugin>
   <groupId>org.liquibase</groupId>
   <artifactId>liquibase-maven-plugin</artifactId>
   <version>4.2.2</version>
   <configuration>
      <propertyFileWillOverride>true</propertyFileWillOverride>
      <propertyFile>src/main/resources/db/liquibase.properties</propertyFile>
   </configuration>
</plugin>
```

Wie funktioniert Liquibase?



Liquibase Optionen

- Liquibase als maven plugin
- Liquibase als Command Line Tool
- Liquibase integriert in DB-Mgt-Tools (z.B. DBeaver)

- Liquibase
  - **generateChangeLog**: erstellt den ChangeLog einer DB
  - **diff**: Vergleicht die DB
    - Version 1 mit 2 oder
    - Version 1 mit JPA Model
  - **update**: Aktualisiert die Datenbank mit ChangeLogs

Wie funktioniert Liquibase? Refactorings mit DSL oder Custom SQL

- Refactorings
  - Anweisungen zur Migration des Datenbankschemas
  - "Liquibase-DSL" = DBMS-unabhängige `XML`-Tags
    - Structural Refactorings, z.B. "`Create Table`"
    - Data Quality Refactorings, z.B. "`Add Unique Constraint`"
    - Referential Integrity Refactorings, z.B. "`Add Foreign Key`"
    - Non-Refactoring Transformations, z.B. "`Insert Data`"
    - Architectural Refactorings, z.B. "`Create Index`"
- Custom SQL
  - Erlaubt DBMS-spezifische DDL bzw. DML
  - Statements in XML-Dateien oder
  - Import von SQL-Skripten

Wie funktioniert Liquibase? Refactorings mit DSL oder Custom SQL

- **Structural Refactorings**
  - Add Column, Rename Column, Modify Column, Drop Column, Alter Sequence, Create Table, Rename Table, Drop Table, Create View, Rename View, Drop View, Merge Columns, Create Stored Procedure
- **Data Quality Refactorings**
  - Add Lookup Table, Add Not-Null Constraint, Remove Not-Null Constraint, Add Unique Constraint, Drop Unique Constraint, Create Sequence, Drop Sequence, Add Auto-Increment, Add Default Value, Drop Default Value
- **Referential Integrity Refactorings**
  - Add Foreign Key Constraint, Drop Foreign Key Constraint, Drop All Foreign Key Constraints, Add Primary Key Constraint, Drop Primary Key Constraint
- **Non-Refactoring Transformations**
  - Insert / Load Data, Load Update Data, Update Data, Delete Data, Tag Database, Stop
- **Architectural Refactorings** Create Index, Drop Index
- **Custom Refactorings**
  - Modifying Generated SQL, Custom SQL, Custom SQL File, Custom Refactoring Class, Execute Shell Command

Wie funktioniert Liquibase?

- Beispiel ChangeLog

```
<databaseChangeLog xmlns=http://www.liquibase.org/xml/ns/dbchangelog
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
  http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-2.0.xsd">
  <changeSet id="1" author="mba">
    <createTable tableName="PERSON">
      <column name="ID" type="BIGINT" autoIncrement="true">
        <constraints primaryKey="true" nullable="false" />
      </column>
      <column name="FIRSTNAME" type="VARCHAR(255)" />
      <column name="LASTNAME" type="VARCHAR(255)" />
    </createTable>
  </changeSet>
   …
  </databaseChangeLog>
```

Wie funktioniert Liquibase? Integrität von Änderungen

- Was passiert, wenn ein ChangeSet im **Nachhinein** geändert ("korrigiert") wird?
  - Ein Entwickler hat im Normalfall nach einem Check-In des Quellcodes keine Information darüber, in welchen Umgebungen das ChangeSet bereits ausgeführt wurde
  - Nachträgliche Änderungen können zu Inkonsistenzen führen
- Schutz ausgeführter ChangeSets mit einer MD5-Summe
  - MD5-Summe wird in der Tabelle DATABASECHANGELOG hinterlegt
  - Prüfung bei jeder Ausführung und Abbruch der Migration bei erkannten Änderungen
- Korrekturen sind im Normalfall nur über kompensierende ChangeSets möglich

Wie funktioniert Liquibase?
Weitere Anweisungen zu ChangeSets

- ChangeSets können neben Refactorings weitere Anweisungen beinhalten
  - Ausführung
    - Vorbedingungen: `dbms="mysql", context="test"`
    - Wiederholte Ausführung: `runOnChange, runAlways`
    - Fehlerverhalten: `failOnError` (default true)
  - Rollback-Strategien
    - Liquibase-Refactoring (z.B. createTable) , SQL, Referenz auf ein ChangeSet

```
<changeSet id="1" author= "mba"
  runOnChange="true" runAlways="false"
  failOnError="false" dbms="oracle"   context="test">
  <-- Refactorings -->
</changeSet>
```

```
mvn liquibase:rollback -Dliquibase.rollbackTag=1.0
mvn liquibase:rollback -Dliquibase.rollbackCount=1
mvn liquibase:rollback -Dliquibase.rollbackDate=05.07.2012
```

Wie funktioniert Liquibase? Weitere Operationen

- status
  - Ausgabe aller auszuführenden ChangeSets

    ```
    mvn liquibase:status
    ```
- dbDoc
  - Erzeugung einer Javadoc-ähnlichen Dokumentation aller bereits ausgeführten und noch auszuführenden ChangeSets

    ```
    mvn liquibase:dbDoc
    ```
- generateChangeLog
  - Erzeugung einer initialen DatabaseChangeLog-Datei
  - Bootstrapping für Liquibase bzw. Erstellen einer neuen Baseline

    ```
    mvn liquibase:generateChangeLog
    ```
- diff
  - Vergleich der aktuellen Umgebung mit einer Referenzumgebung (db oder JPA)
  - Ausgabe der Differenzen als ChangeSets oder Reports

Wie funktioniert Liquibase? Empfehlung

- Empfehlung: Aufteilen der DatabaseChangeLogs **anhand der jeweils aktuellen Anwendungsversion:**
  - liquibase/master.xml
  - liquibase/1.0.0.xml
  - liquibase/1.1.0.xml
  - liquibase/1.2.0.xml
- Inhalt von master.xml

```
<databaseChangeLog …>
  <include file="liquibase/1.0.0.xml" />
  <include file="liquibase/1.1.0.xml" />
  <include file="liquibase/1.2.0.xml" />
</databaseChangeLog>
```

Spring und Liquibase

- Default ist
  - src/main/resources db/changelog/db.changelog.yaml
- Default kann geändert werden in: application.properties
  `liquibase.change-log=classpath:/db/changelog-option2/master.yaml`
- master.yaml kann Dateien importieren von einem Order include

```yaml
databaseChangeLog:
  - changeSet:
      id: 1
      author: matthiasbachmann
      changes:
        - createTable:
            tableName: customer
            columns:
              - column:
                  name: id
                  type: bigint
```

Beispiel: Bei Änderungen müssen die H2-DB-Dateien gelöscht werden

```yaml
databaseChangeLog:


    - includeAll:
        path: include
        relativeToChangelogFile: true
```

ChangeLog im YAML-Format

```yaml
databaseChangeLog:
  - changeSet:
      id: 1
      author: matthiasbachmann
      changes:
        - createTable:
            tableName: customer
            columns:
              - column:
                  name: id
                  type: bigint
                  autoIncrement: true
                  constraints:
                    primaryKey: true
                    nullable: false
              - column:
                  name: firstname
                  type: varchar(255)
                  constraints:
                    nullable: false
              - column:
                  name: lastname
                  type: varchar(255)
                  constraints:
                    nullable: false
```

Wie funktioniert Liquibase? Weitere Operationen

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

- tag
  - Setzen eines Tags in der aktuellen Umgebung
  - Ermöglicht späteres Rollback

```
mvn liquibase:tag –Dliquibase.tag=1.0
```

- rollback
  - Zurückrollen von Änderungen in der aktuellen Umgebung
  - Unterstützte Modi: Tag, Count und Date

- dropAll
  - Entfernt alle Datenbank-Objekte aus dem aktuellen Schema
  - Ausnahme: Functions, Procedures, Packages

```
mvn liquibase:dropAll
```

GenerateChangeLog

- Voraussetzungen für Reverse Engineering der Datenbank
  - Anpassung der url in application.properties

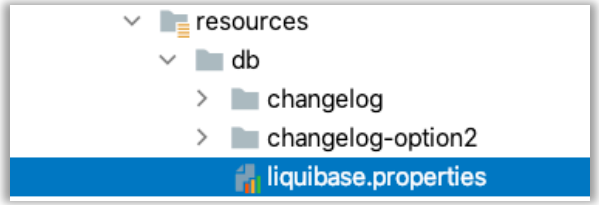
```
spring.datasource.url=jdbc:h2:file:./data/testdb
```

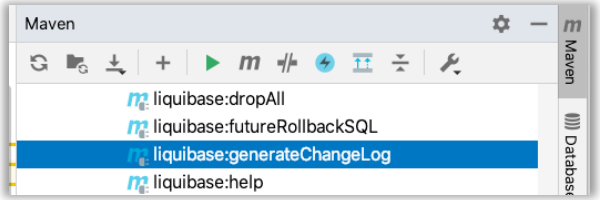
  - liquibase.properties Datei in resources/db

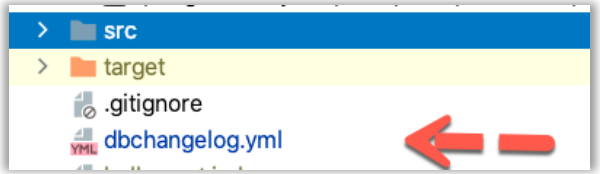
```
# The output changeLogFile can be xml, yml or json
outputChangeLogFile: dbchangelog.yml
driver: org.h2.Driver
url: jdbc:h2:file:./data/testdb
username: sa
password:
promptOnNonLocalDatabase: false

# diff used for generateChangeLog and diff
diffTypes: tables, views, columns, indexes,
foreignkeys, primarykeys, uniqueconstraints, data

# ChangeLog configuration
databaseChangeLogTableName: DATABASECHANGELOG
databaseChangeLogLockTableName: DATABASECHANGELOGLOCK
```





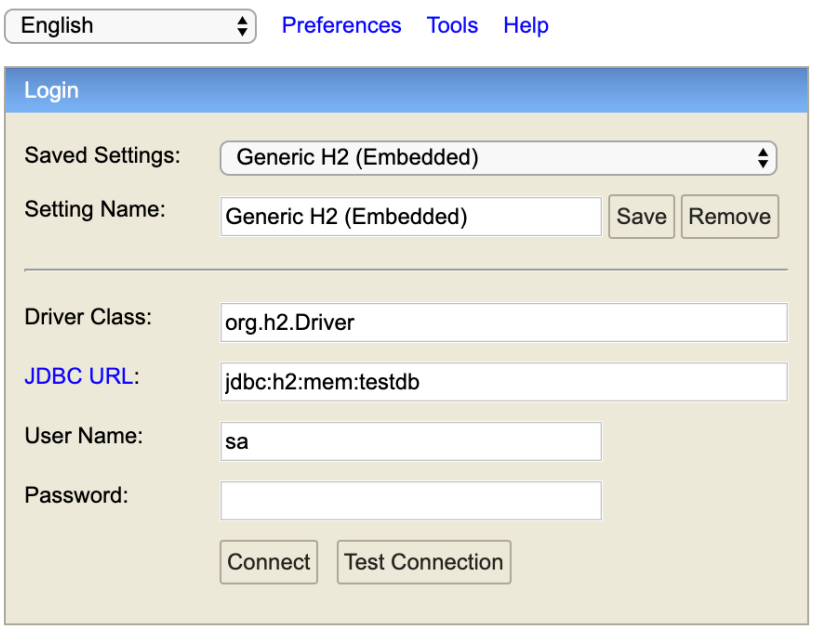


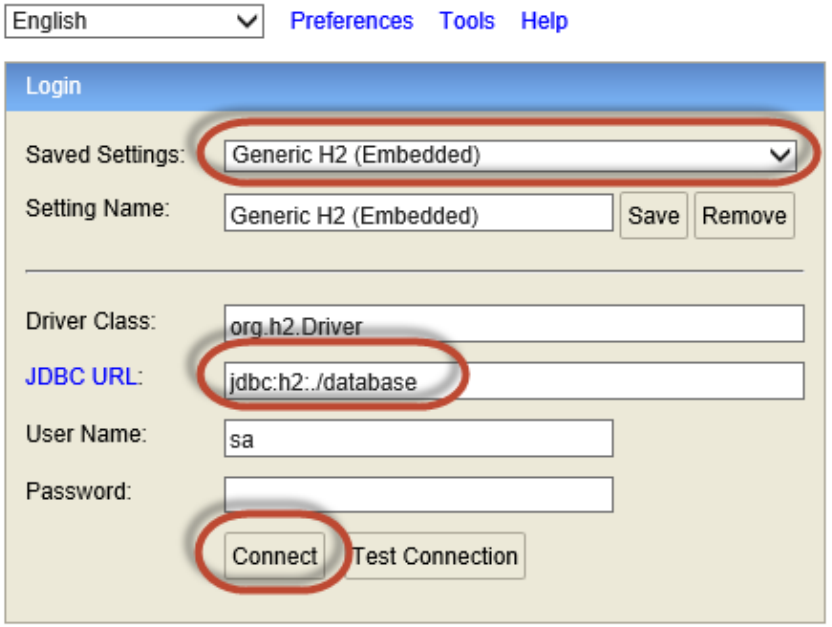
Hands-on: Liquibase

- Checkout Branch `Database Bootstrap...`
  Anleitung:
  https://github.zhaw.ch/bacn/ase2-spring-boot-hellorest/blob/liquibase/readme/liquibase.md
- Anpassung application.properties Datei
  https://github.zhaw.ch/bacn/ase2-spring-boot-hellorest/blob/liquibase/src/main/resources/application.properties
- Change Log
  https://github.zhaw.ch/bacn/ase2-spring-boot-hellorest/tree/liquibase/src/main/resources/db/changelog-option2
- Anpassung pom-Datei
  Siehe Setup Folie

Datenbank Konsole (1) Liquibase

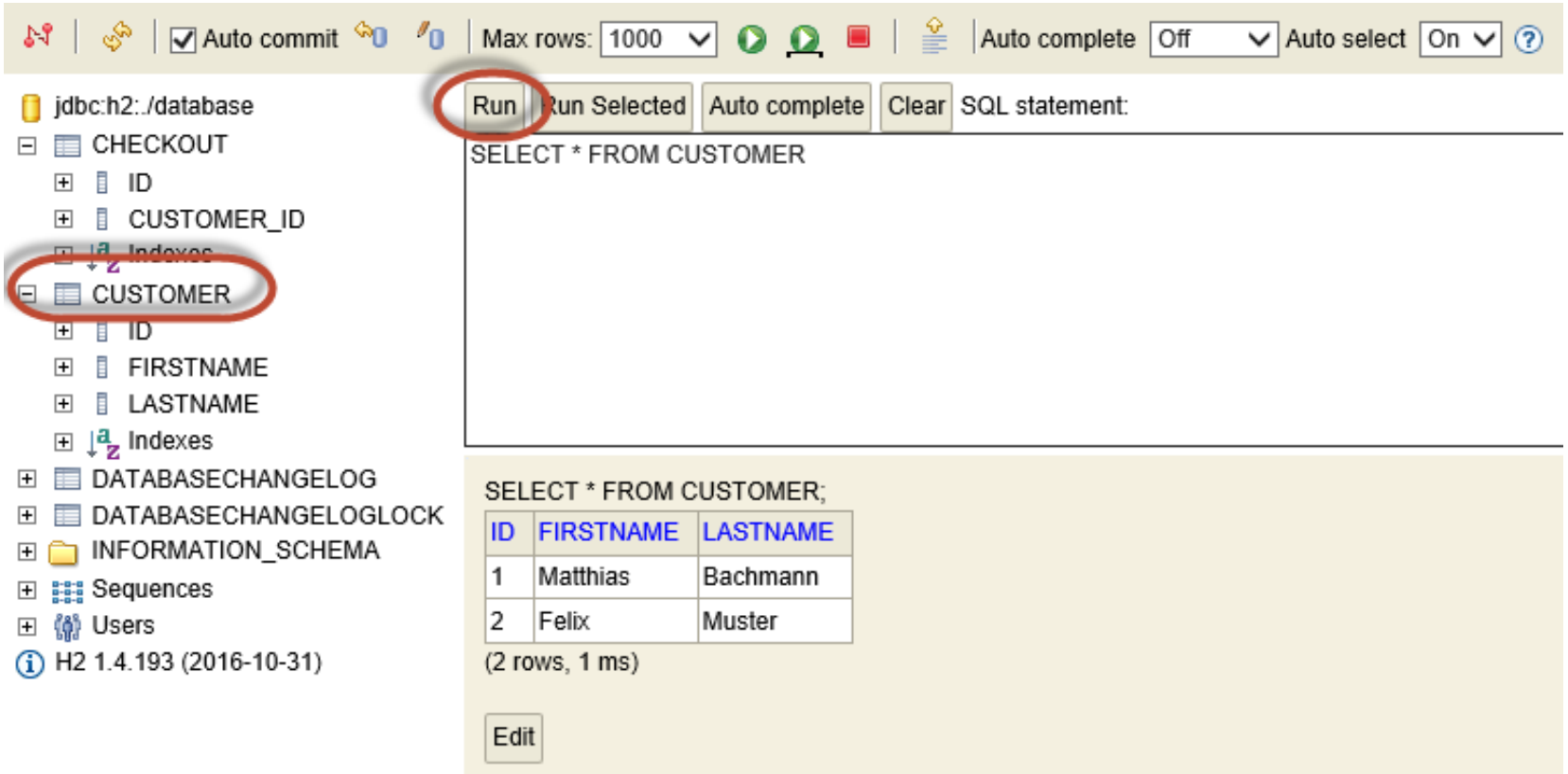
- http://localhost:8080/h2-console
- `jdbc:h2:mem:testdb` oder `jdbc:h2:./database` (mit Datei) bzw. `jdbc:h2:file:./data/testdb` (Beispiel generateChangeLog)
  - `:./data/testdb` Dateiname wie in `application.properties`)





Datenbank Konsole (2) - Liquibase

- Beinhalted die beiden Tabellen DATABASECHANGELOG und DATABASECHANGELOGLOCK



Zusammenfassung

- Datenbank Initialisierung mittels Bootstrap in Java
- Mittels SQL Dateien schema-h2-sql, data-h2.sql
- Datenbank Migration mit FlyWay
- Datenbank Migration mit Liquibase