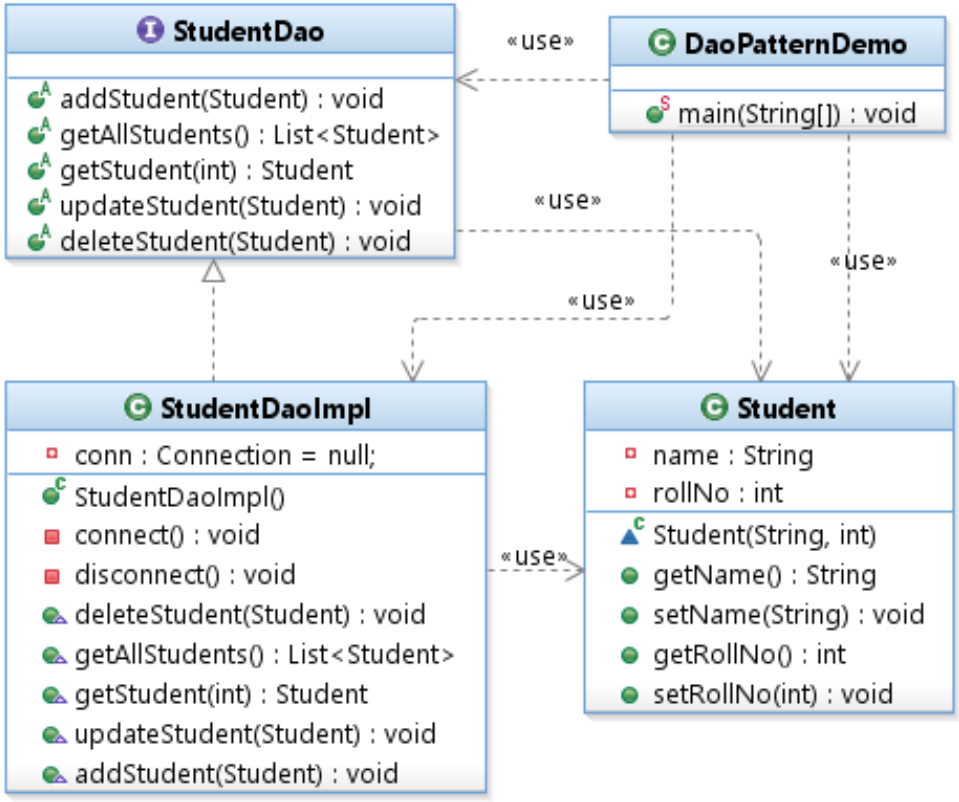
Information systems modeling

Tomasz Kubik

Data Access Objects Pattern

- used to separate low level data accessing API or operations from high level business services
- isolates the application/business layer from the persistence layer (usually a relational database, but it could be any other persistence mechanism) using an abstract API.



https://www.baeldung.com/java-dao-pattern

https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm
https://www.journaldev.com/16813/dao-design-pattern

DAO and JDBC

1. Load the driver

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. Get connection

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

3. Create statement

```
Statement stmt = conn.prepareStatement(SQL);
```

4. Execute query

```
String sql = "SELECT id, name FROM Employees";
ResultSet rs = stmt.executeQuery(sql);
```

5. Process result set

```
int id = rs.getInt(1);
```

Working with JdbcTemplate in Spring

https://www.javatpoint.com/spring-JdbcTemplate-tutorial

EmployeeDao.java

```
public class EmployeeDao {
private JdbcTemplate jdbcTemplate;

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}

public int saveEmployee(Employee e){
    String query="insert into employee values(
    '"+e.getId()+"','"+e.getName()+"','"+e.getSalary()+"')";
    return jdbcTemplate.update(query);
}
…
```

applicationContext.xml

```
<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
<property name="username" value="system" />
<property name="password" value="oracle" />
</bean>

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="ds"></property>
</bean>

<bean id="edao" class="com.javatpoint.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
```

Working with JdbcTemplate in Spring Boot

https://www.javacodegeeks.com/2016/03/springboot-working-jdbctemplate.html

```
@Configuration
@ComponentScan
@EnableTransactionManagement
@PropertySource(value = { "classpath:application.properties" })
public class AppConfig
{
    @Autowired
    private Environment env;

    @Value("${init-db:false}")
    private String initDatabase;

    @Bean
    public static PropertySourcesPlaceholderConfigurer placeHolderConfigurer()
    {
        return new PropertySourcesPlaceholderConfigurer();
    }

    @Bean
    public JdbcTemplate jdbcTemplate(DataSource dataSource)
    {
        return new JdbcTemplate(dataSource);
    }

    @Bean
    public PlatformTransactionManager transactionManager(DataSource dataSource)
    {
        return new DataSourceTransactionManager(dataSource);
    }

    @Bean
    public DataSource dataSource()
    {
        BasicDataSource dataSource = new BasicDataSource();
        dataSource.setDriverClassName(env.getProperty("jdbc.driverClassName"));
        dataSource.setUrl(env.getProperty("jdbc.url"));
        dataSource.setUsername(env.getProperty("jdbc.username"));
        dataSource.setPassword(env.getProperty("jdbc.password"));
        return dataSource;
    }
…
```

Working with JdbcTemplate in Spring Boot

https://www.javacodegeeks.com/2016/03/springboot-working-jdbctemplate.html

```
public class User
{
    private Integer id;
    private String name;
    private String email;
// setters & getters
}

@Repository
public class UserRepository
{
    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Transactional(readOnly=true)
    public List<User> findAll() {
        return jdbcTemplate.query("select * from users", new UserRowMapper());
    }
}

class UserRowMapper implements RowMapper<User>
{
    @Override
    public User mapRow(ResultSet rs, int rowNum) throws SQLException
    {
        User user = new User();
        user.setId(rs.getInt("id"));
        user.setName(rs.getString("name"));
        user.setEmail(rs.getString("email"));
        return user;
    }
}
```

Working with JdbcTemplate in Spring Boot

https://www.javacodegeeks.com/2016/03/springboot-working-jdbctemplate.html

pom.xml

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

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

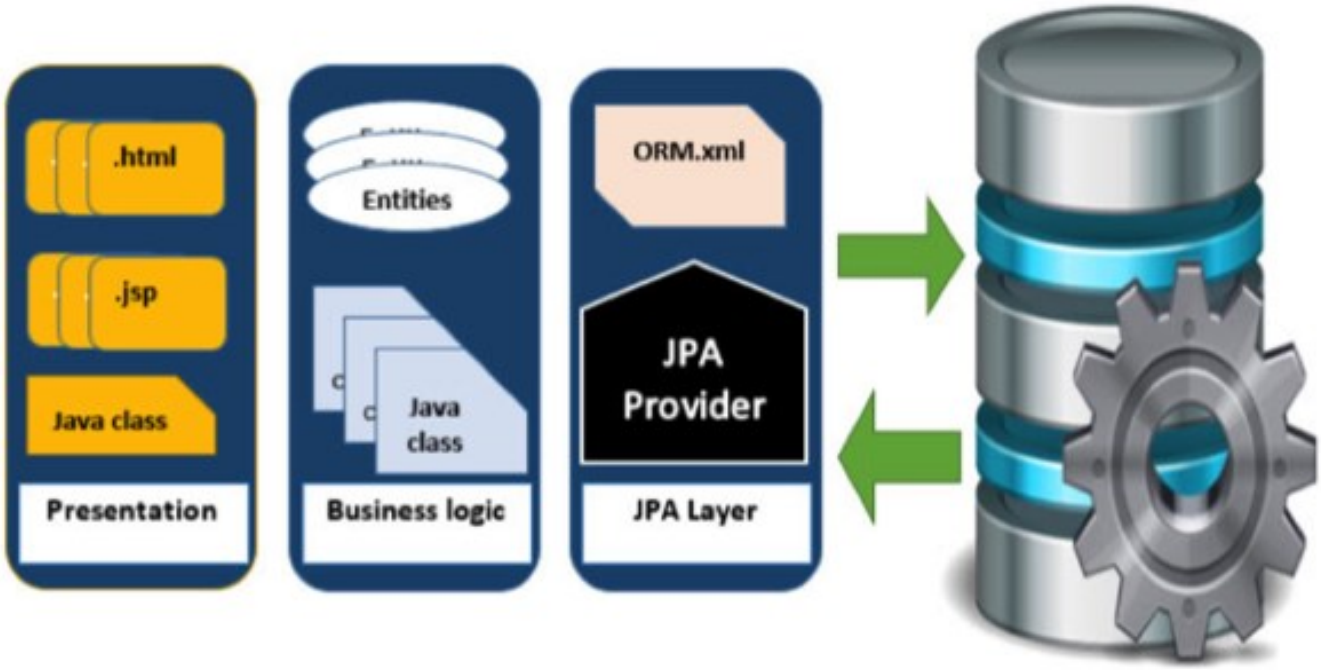
application.property

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/test
spring.datasource.username=root
spring.datasource.password=admin
```

Java Persistence API (JPA)

- Java specification for accessing, persisting, and managing data between Java objects / classes and a relational database.
- was defined as part of the EJB 3.0 (Enterprise JavaBeans) specification as a replacement for the EJB 2 CMP Entity Beans specification
- is now considered the standard industry approach for Object to Relational Mapping (ORM) in the Java Industry
- allows POJO (Plain Old Java Objects) to be easily persisted without requiring the classes to implement any interfaces or methods
- allows ORM mappings (how class maps to a relational database table) to be defined by the use of annotations or in XML
- defines a runtime `EntityManager` API for processing queries and transaction on the objects against the database
- defines an object-level query language, JPQL, to allow querying of the objects from the database
- most of the persistence vendors have released implementations of JPA
  - Hibernate (acquired by JBoss, acquired by Red Hat),
  - TopLink (acquired by Oracle),
  - Kodo JDO (acquired by BEA, acquired by Oracle)

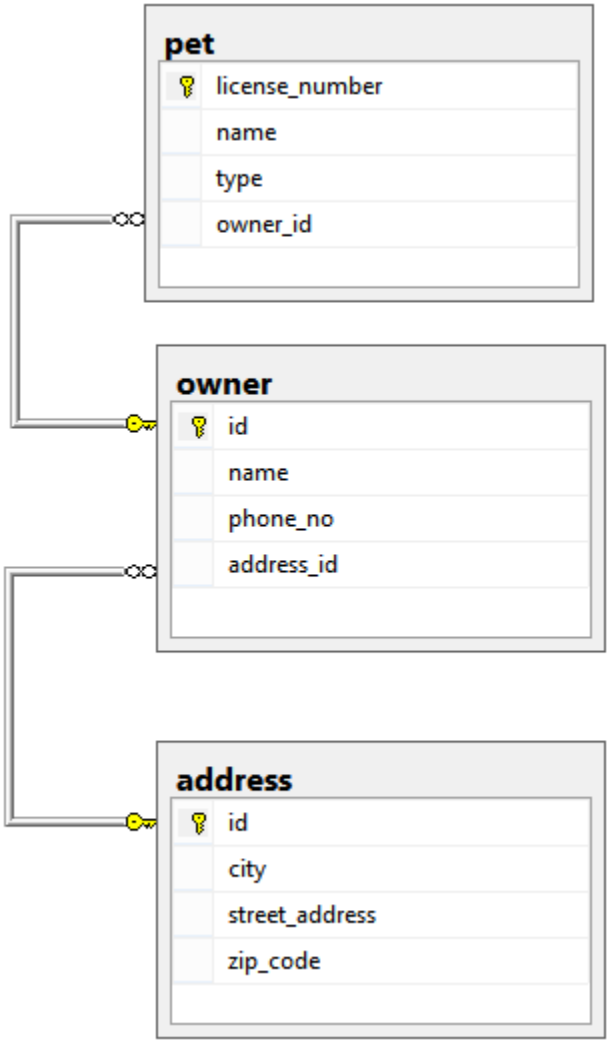
JPA placement



https://www.tutorialspoint.com/jpa/jpa_introduction.htm

Database schema (simple-jpa)

```
create table address (
  id integer not null,
  city varchar(255),
  street_address varchar(255),
  zip_code varchar(255),
  primary key (id)
)
create table owner (
  id integer not null,
  name varchar(255),
  phone_no varchar(255),
  address_id integer,
  primary key (id)
)
create table pet (
  license_number integer not null,
  name varchar(255),
  type integer,
  owner_id integer,
  primary key (license_number)
)
alter table owner add constraint FK_Owner_Address
  foreign key (address_id) references address

alter table pet add constraint FK_Pet_Owner
  foreign key (owner_id) references owner
```



Pet entity (simple-jpa)

```
@Entity
@Table(name = "pet")
public class Pet {
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	private Integer licenseNumber;


	private String name;
	private PetType type;

//	@ManyToOne(fetch = FetchType.LAZY)
	@ManyToOne
	@JoinColumn(name = "owner_id")
	private Owner owner;
```

Address entity (simple-jpa)

```
@Entity
@Table(name = "address")
public class Address {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;
    private String streetAddress;
    private String city;
    private String zipCode;
```

Owner entity

```
@Entity
@Table(name = "owner")
public class Owner {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;
    private String name;
    @Column(name = "phone_no")
    private String phoneNumber;
    // The @JoinColumn annotation combined with a @OneToOne mapping
    // indicates that a given column in the owner entity refers
    // to a primary key in the reference entity
    @JoinColumn(name = "address_id")
    @OneToOne
    Address address;
    // When using a @OneToMany mapping we can use the mappedBy parameter
    // to indicate that the given column is owned by another entity.
    @OneToMany(mappedBy = "owner")
    List<Pet> pets;
```

Working with persisted object

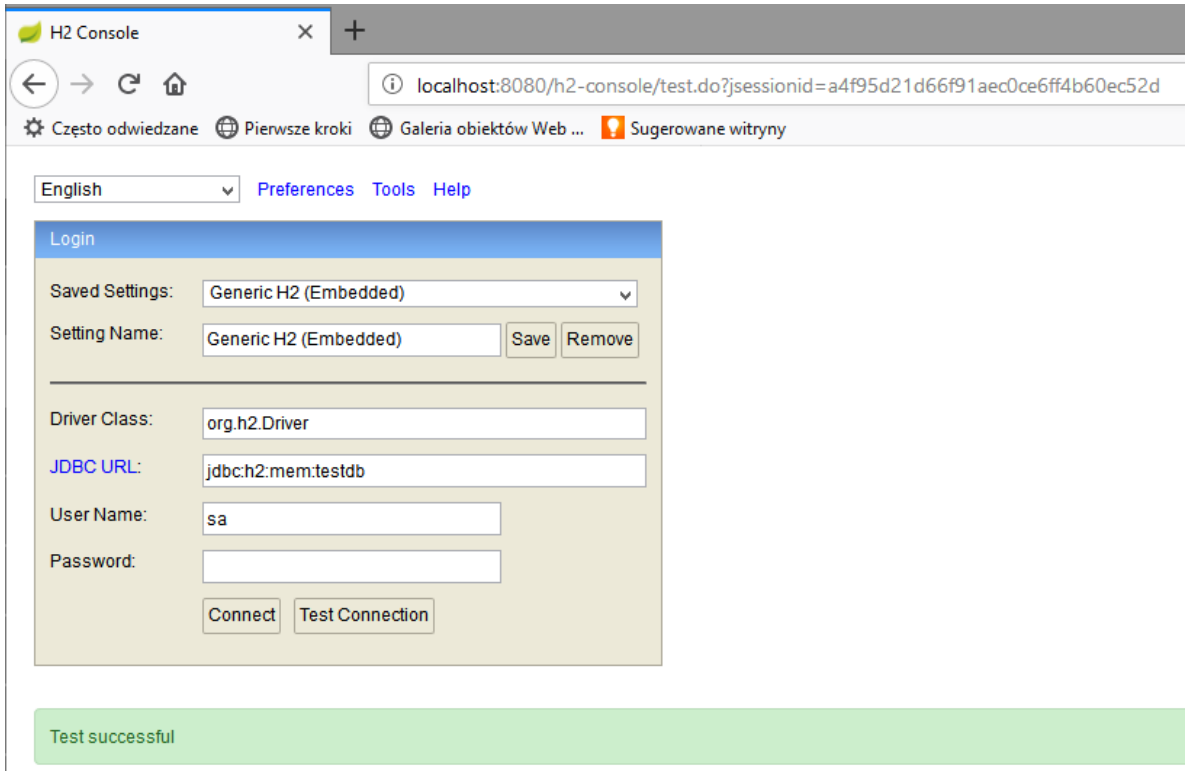
```
Address a = new Address()
    .withStreetAddress("Wyspianskiego 11")
    .withZipCode("50-400")
    .withCity("Wroclaw");
a = addressRepo.save(a);
Owner o1 = new Owner(0, "John", "555000001", a);
Owner o2 = new Owner(0, "Adam", "555000002", a);

o1 = ownerRepo.save(o1);
o2 = ownerRepo.save(o2);

Pet p1 = new Pet("Tom", PetType.CAT, o1);
p1 = petRepo.save(p1);
Pet p2 = new Pet("Jerry", PetType.MOUSE, o1);
p2 = petRepo.save(p2);

for (Pet pet : petRepo.findAll()) {
        log.info(pet.toString());
}
log.info("");
```

Configuration

- H2 database
  - very fast, open source, JDBC API
  - embedded and server modes; in-memory databases
  - browser based Console application
  - small footprint: around 2 MB jar file size
  - good for testing
- `application.properties`

```
spring.h2.console.enabled=true
spring.jpa.show-sql = true
```

H2-console



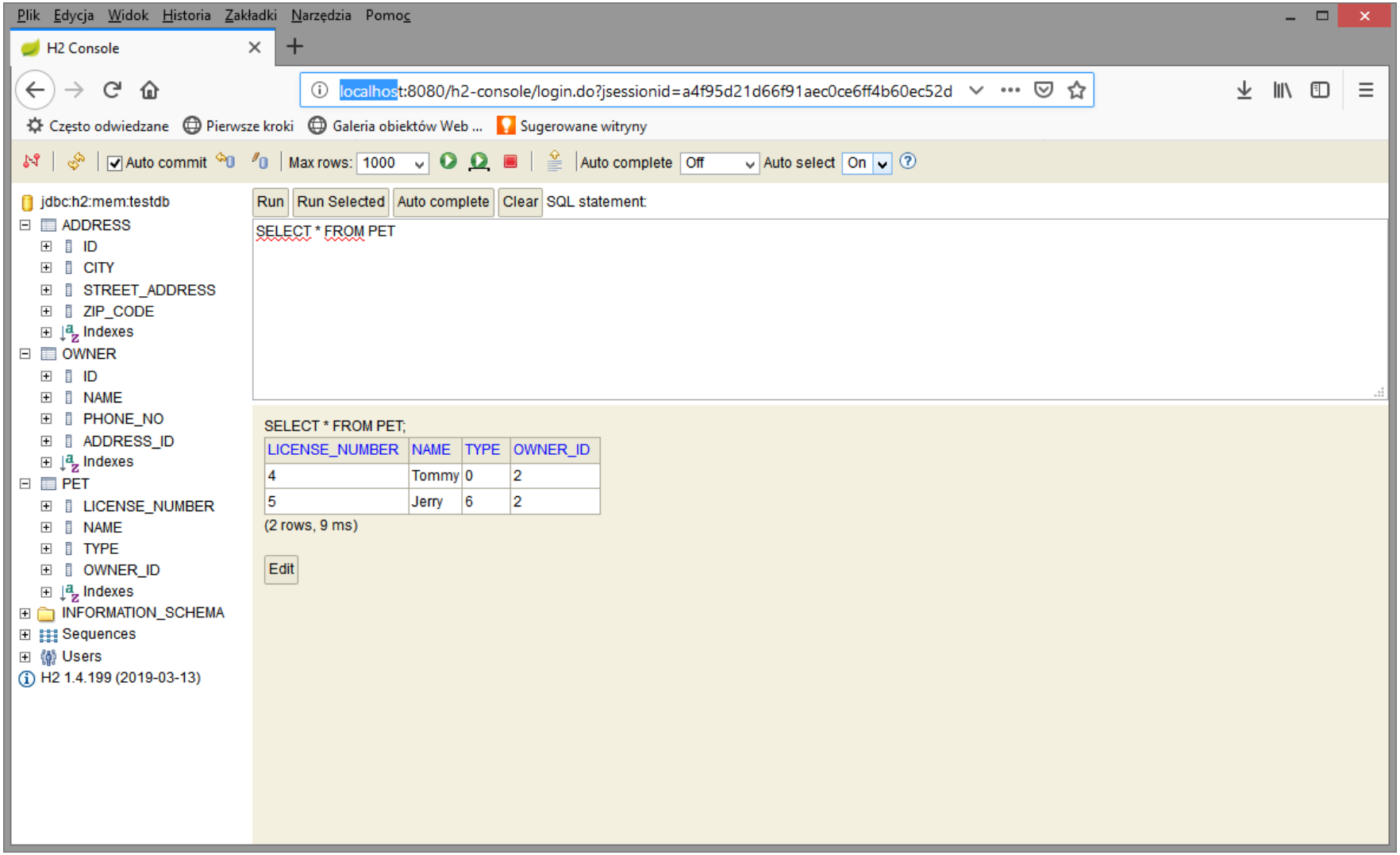
Url to h2-console:

```
http://localhost:8080/h2-console
```

Credentials:

```
JDBC URL: jdbc:h2:mem:testdb
User Name: sa
Pasword: <leave this empty>
```

H2-console


Plik Edycja Widok Historia Zakładki Narzędzia Pomoc
H2 Console
localhost:8080/h2-console/login.do?jsessionid=a4f95d21d66f91aec0ce6ff4b60ec52d
Często odwiedzane
Pierwsze kroki
Galeria obiektów Web ...
Sugerowane witryny
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:mem:testdb
ADDRESS
ID
CITY
STREET_ADDRESS
ZIP_CODE
Indexes
OWNER
ID
NAME
PHONE_NO
ADDRESS_ID
Indexes
PET
LICENSE_NUMBER
NAME
TYPE
OWNER_ID
Indexes
INFORMATION_SCHEMA
Sequences
Users
H2 1.4.199 (2019-03-13)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PET
SELECT * FROM PET;
LICENSE_NUMBER NAME TYPE OWNER_ID
4 Tommy 0 2
5 Jerry 6 2
(2 rows, 9 ms)
Edit

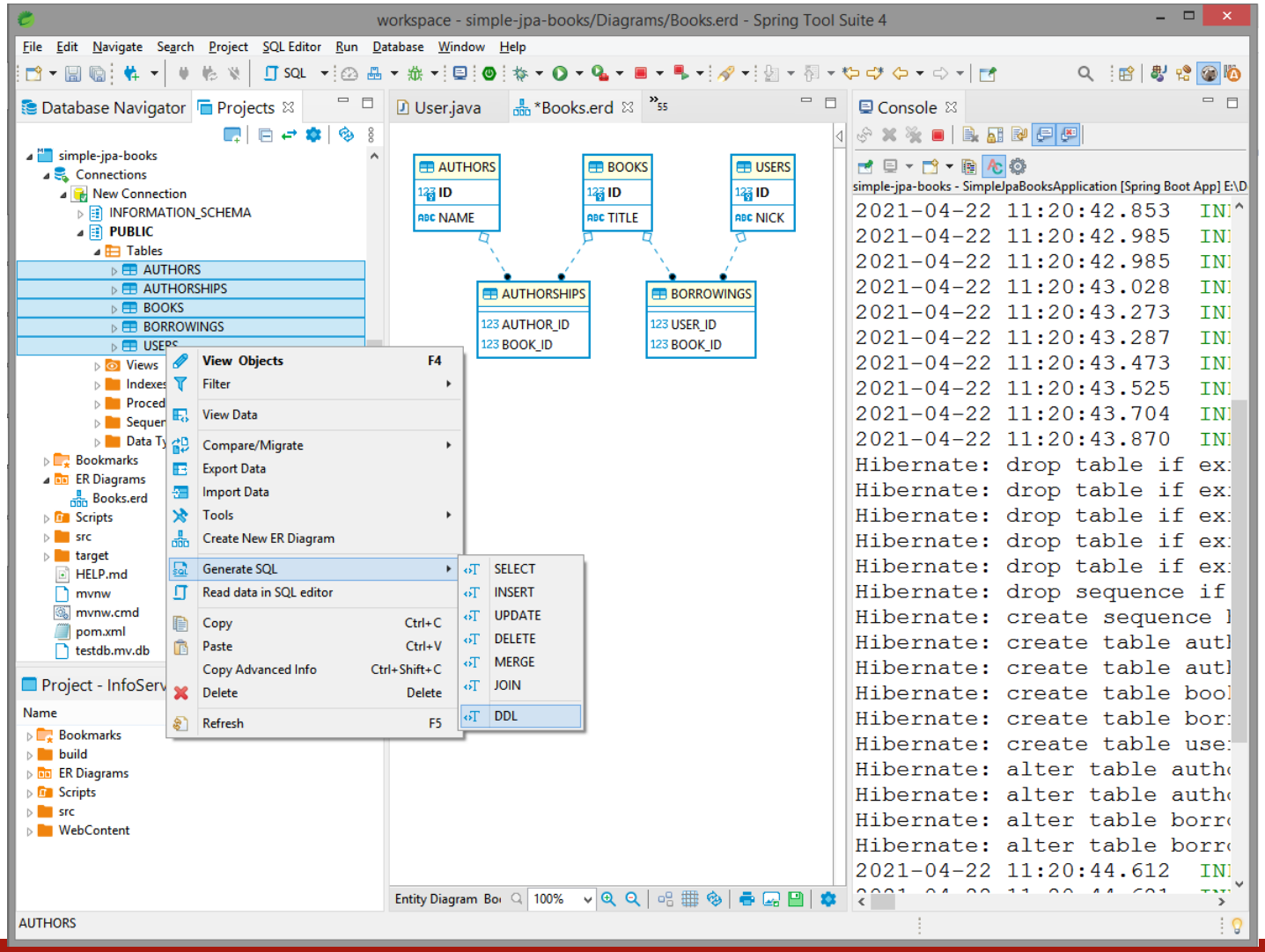
Links

##
JavaScript + AngularJS
##
https://www.w3schools.com/js/js_syntax.asp
https://angularjs.org/
https://docs.angularjs.org/tutorial/step_06
https://start.spring.io/

##
JPA
##
https://attacomsian.com/blog/spring-data-jpa-repositories
https://www.baeldung.com/spring-data-repositories
https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html
https://attacomsian.com/blog/spring-data-jpa-repositories

simple-jpa-books database schema

- DDL commands retrieved from DBeaver after connecting to the database



Database schema

- DDL commands retrieved from DBeaver after connecting to the database

```
CREATE TABLE AUTHORS (
  ID INTEGER NOT NULL,
  NAME VARCHAR(255),
  CONSTRAINT CONSTRAINT_4 PRIMARY KEY (ID)
);
CREATE UNIQUE INDEX PRIMARY_KEY_4 ON AUTHORS (ID);

CREATE TABLE BOOKS (
  ID INTEGER NOT NULL,
  TITLE VARCHAR(255),
  CONSTRAINT CONSTRAINT_3 PRIMARY KEY (ID)
);
CREATE UNIQUE INDEX PRIMARY_KEY_3 ON BOOKS (ID);

CREATE TABLE USERS (
  ID INTEGER NOT NULL,
  NICK VARCHAR(255),
  CONSTRAINT CONSTRAINT_4D PRIMARY KEY (ID)
);
CREATE UNIQUE INDEX PRIMARY_KEY_4D ON USERS (ID);

CREATE TABLE AUTHORSHIPS (
  AUTHOR_ID INTEGER NOT NULL,
  BOOK_ID INTEGER NOT NULL,
  CONSTRAINT FK1GKJW4T9AQA7VM09BM7O3KVY FOREIGN KEY (AUTHOR_ID) REFERENCES AUTHORS(ID) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT FKMI52O5YJL1F9FOTFMKCNS6GKH FOREIGN KEY (BOOK_ID) REFERENCES BOOKS(ID) ON DELETE RESTRICT ON UPDATE RESTRICT
);
CREATE INDEX FK1GKJW4T9AQA7VM09BM7O3KVY_INDEX_A ON AUTHORSHIPS (AUTHOR_ID);
CREATE INDEX FKMI52O5YJL1F9FOTFMKCNS6GKH_INDEX_A ON AUTHORSHIPS (BOOK_ID);

CREATE TABLE BORROWINGS (
  USER_ID INTEGER NOT NULL,
  BOOK_ID INTEGER NOT NULL,
  CONSTRAINT FK6Q1QAI7ON9RCRYJI44LYGLQT1 FOREIGN KEY (BOOK_ID) REFERENCES BOOKS(ID) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT FKAEXIAOWFDKA601NS4QV7PU0RE FOREIGN KEY (USER_ID) REFERENCES USERS(ID) ON DELETE RESTRICT ON UPDATE RESTRICT
);
CREATE INDEX FK6Q1QAI7ON9RCRYJI44LYGLQT1_INDEX_7 ON BORROWINGS (BOOK_ID);
CREATE INDEX FKAEXIAOWFDKA601NS4QV7PU0RE_INDEX_7 ON BORROWINGS (USER_ID);
```

Hints

1. Modelling many to many relations
https://www.baeldung.com/jpa-many-to-many

2. Lifecycle
https://thorben-janssen.com/persist-save-merge-saveorupdate-whats-difference-one-use/

3. Why Set Is Better Than List in @ManyToMany
https://dzone.com/articles/why-set-is-better-than-list-in-manytomany

4. Spring Boot with H2 Database
https://www.baeldung.com/spring-boot-h2-database

5. Accessing H2 Database runing within SpringBootApplication

How to enable H2 Database Server Mode in Spring Boot
https://stackoverflow.com/questions/55830010/how-to-enable-h2-database-server-mode-in-spring-boot

How to access in memory h2 database of one spring boot application from another spring boot application
https://stackoverflow.com/questions/43256295/how-to-access-in-memory-h2-database-of-one-spring-boot-application-from-another/43276769#43276769

6. Just to inform
Not needed in applicaiton.properties because of dependencies existing in pom.xml
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

7. Browsing H2 testdb database

application.properties
spring.datasource.url=jdbc:h2:mem:testdb
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto=create
spring.datasource.username=sa
spring.datasource.password=password
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.h2.console.settings.trace=true
spring.h2.console.settings.web-allow-others=true

pom.xml (needed to connect by h2-console)

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

in source code of SpringBootApplication (needed to connect by dbeaver)

```
@Bean(initMethod = "start", destroyMethod = "stop")
public Server h2Server() throws SQLException {
    return Server.createTcpServer("-tcp", "-tcpAllowOthers", "-tcpPort", "9092");
}
```

Settings for h2-console
running on http://localhost:8080/h2-console/
Driver class: org.h2.Driver
JDBC URL: jdbc:h2:mem:testdb

Settings for dbeaver (thanks to the injected bean, consult remark above)
URI template: jdbc:h2:tcp://localhost:9092/mem:testdb
Default database: mem.testdb

changes if database is going to be stored in a file
in application.properties:
spring.datasource.url=jdbc:h2:./testdb
in dbeaver settings
JDBC URL: jdbc:h2:tcp://localhost:9092/./testdb