

POLITECHNIKA WROCLAWSKA
WYDZIAŁ ELEKTRONIKI

**Projekt z rozproszonych i obiektowych
systemów baz danych**

**Projekt aplikacji wykorzystującej
mechanizmy replikacji MySQL**

AUTOR:

Dymon Jakub

Nowacki Paweł

Sulżycki Paweł

PROWADZĄCY ZAJĘCIA:

Dr inż. Robert Wójcik, W4/K-9

OCENA PRACY:

Wrocław 2018

Spis treści

Spis rysunków	3
1. Wstęp	4
1.1. Cele projektu	4
1.2. Założenia projektowe	4
1.3. Zakres projektu	4
2. Replikacja w systemie baz danych MySQL	5
2.1. Mechanizm replikacji	5
2.2. Replikacja Multi-master	5
3. Modele baz danych	6
3.1. Model konceptualny	6
3.2. Model logiczny	6
3.3. Model fizyczny	7
4. Implementacja baz danych w środowisku MySQL	8
4.1. Opis implementacji mechanizmu replikacyjnego multi-master	8
4.1.1 Przygotowanie środowiska	8
4.1.2 Konfiguracja maszyn wirtualnych	8
4.1.3 Konfiguracja wewnątrz MySQL	9
4.2. Schemat bazy danych	10
5. Projekt i implementacja aplikacji klienckiej	12
5.1. Funkcje aplikacji - diagram przypadków użycia	12
5.2. Realizacja wybranych funkcjonalności	12
5.2.2. Usuwanie projektów	15
5.2.4. Dodawanie projektu	19
6. Wdrożenie i testowanie aplikacji	21
6.1. Instalowanie i konfiguracja aplikacji	21
6.2. Testy akceptacyjne wybranych operacji	21
6.2.1. Operacja dodawania	22
6.2.2. Operacja usuwania	23
6.2.3. Operacja Edycji	24
6.2.4. Operacja odłączenie urządzenia	25
7. Podsumowanie	27
Literatura	28

Spis rysunków

Rysunek 1 Schemat mechanizmu replikacji.	5
Rysunek 2 Model konceptualny bazy danych.	6
Rysunek 3 Model logiczny bazy danych.	6
Rysunek 4 Model fizyczny bazy danych.	7
Rysunek 5 Tabela prezentująca bazy danych.	10
Rysunek 6 Diagram przypadków użycia.	12
Rysunek 7 Widok aplikacji listy projektów.	15
Rysunek 8 Widok edycji w aplikacji.	19
Rysunek 9 Widok następnego formularza w aplikacji.	20
Rysunek 10 Startowy widok aplikacji.	21
Rysunek 11 Stan początkowy baz danych.	22
Rysunek 12 Widok aplikacji po dodaniu nowego projektu.	22
Rysunek 13 Stan bazy po dodaniu nowego projektu.	23
Rysunek 14 Widok aplikacji po usunięciu projektu.	23
Rysunek 15 Stan bazy po usunięciu projektu.	24
Rysunek 16 Widok aplikacji po operacji edycji.	24
Rysunek 17 Widok stanu bazy po wykonaniu operacji edycji.	25
Rysunek 18 Replikacja bazy na dwóch urządzeniach.	25
Rysunek 19 Brak replikacji na urządzeniu.	26

1. Wstęp

1.1. Cele projektu

Celem projektu jest stworzenie aplikacji internetowej na potrzeby działu kadr przedsiębiorstwa. Przedsiębiorstwo posiada trzy lokalizacje, a w każdej z nich znajduje się dział kadr, który potrzebuje dostępu do danych o wszystkich pracownikach firmy. Wszystkie lokalizacje są równorzędne i potrzebują zarówno odczytywać, jak i edytować dane pracowników.

1.2. Założenia projektowe

System pozwala na dodawanie i usuwanie pracowników, przeglądanie i edycję danych osobowych pracowników i zależności hierarchicznych pomiędzy nimi. Ponadto system replikacji bazy danych pozwala na obsługę systemu z różnych oddziałów firmy rozszaniach po całym świecie. System ma formę aplikacji internetowej z systemem logowania i jest dostępny dla odpowiednio przeszkolonych pracowników działu kadr. Konta użytkowników są zakładane oddgórnie przez administratora.

Aplikacja wykorzystuje relacyjną bazę danych MySQL oraz logikę biznesową zaimplementowaną z użyciem języka Java i biblioteki Spring Framework oraz pochodnych. Każda instancja aplikacji posiada własną bazę danych o takich samej zawartości i jest dobierana na podstawie lokalizacji. Stosowana jest replikacja migawkowa typu multimaster na poziomie bazy danych i służy propagacji danych pomiędzy różnymi instancjami tej samej aplikacji.

1.3. Zakres projektu

Pierwszy rozdział zawiera wiedzę teoretyczną, na której oparty został projekt. W drugim rozdziale zamieszczono schematy baz danych. Trzeci rozdział poświęcony został opisowi konfiguracji bazy danych MySQL. Kolejny rozdział zawiera opis implementacji aplikacji. Ostatni rozdział to opis przeprowadzonych testów.

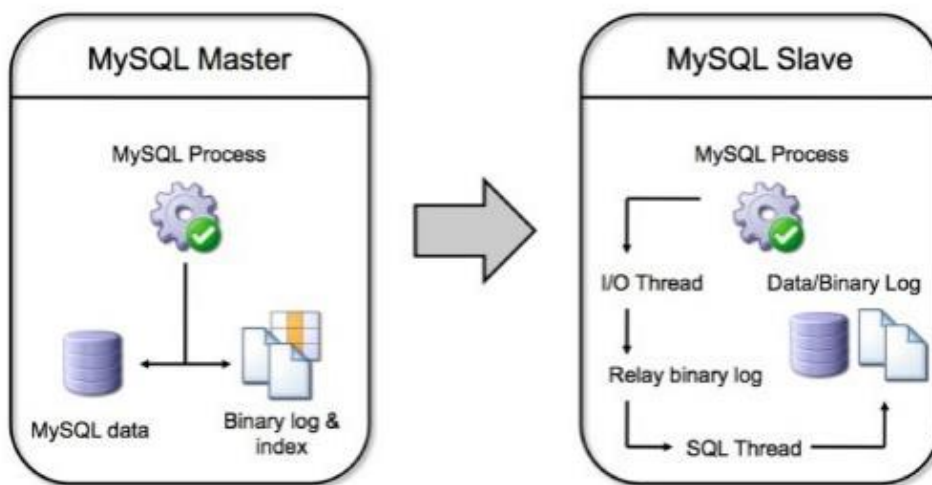
2. Replikacja w systemie baz danych MySQL

2.1. Mechanizm replikacji

Replikacja danych MySQL opiera się o bardzo prostą zasadę. Serwer główny (master), prowadzi swego rodzaju dziennik, w którym zapisuje każdą czynność jaką wykonał. Wykorzystuje do tego bin-logi. Są to pliki binarne zawierające instrukcje jakie wykonał master. Serwer zapasowy (slave) odczytuje te dane i kolejno wykonuje zapytania uzupełniając bazę kolejnymi rekordami. Efektem tej pracy są dwie identyczne bazy danych. Po skonfigurowaniu mechanizmu replikacji na serwerze master pojawia się dodatkowy wątek, który odpowiada za wysyłanie bin-logów do serwerów slave. Z kolei serwer zapasowy tworzy dwa wątki:

- I/O Thread – odpowiada za odbieranie dziennika od serwera głównego i zapisuje go w plikach tymczasowych relay-log.
- SQL Thread – zajmuje się parsowaniem tych plików i wykonywaniem zapytań do bazy.

Całą sytuację przedstawia schemat poniżej (rysunek 1).



Rysunek 1 Schemat mechanizmu replikacji.

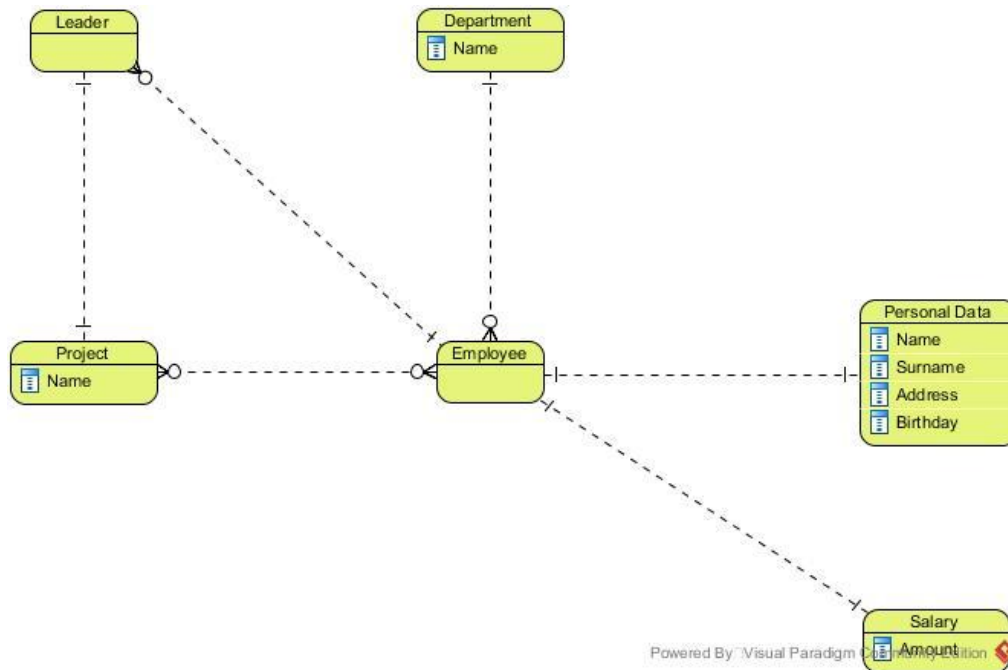
2.2. Replikacja Multi-master

To metoda replikacji baz danych, która umożliwia przechowywanie danych przez grupę komputerów i aktualizację przez dowolnego członka grupy. Wszyscy członkowie reagują na zapytania danych klientów. System replikacji typu multi-master jest odpowiedzialny za propagowanie modyfikacji danych dokonanych przez każdego członka do reszty grupy i rozwiązywanie wszelkich konfliktów, które mogą powstać między równoległymi zmianami dokonywanymi przez różnych członków.

Replikację multi-master można skonstruować z replikacją master-slave, w której pojedynczy element grupy jest określany jako "master" dla danej części danych i jest jedynym węzłem, który może modyfikować ten element danych. Inni członkowie, którzy chcą zmodyfikować element danych, muszą najpierw skontaktować się z węzłem głównym

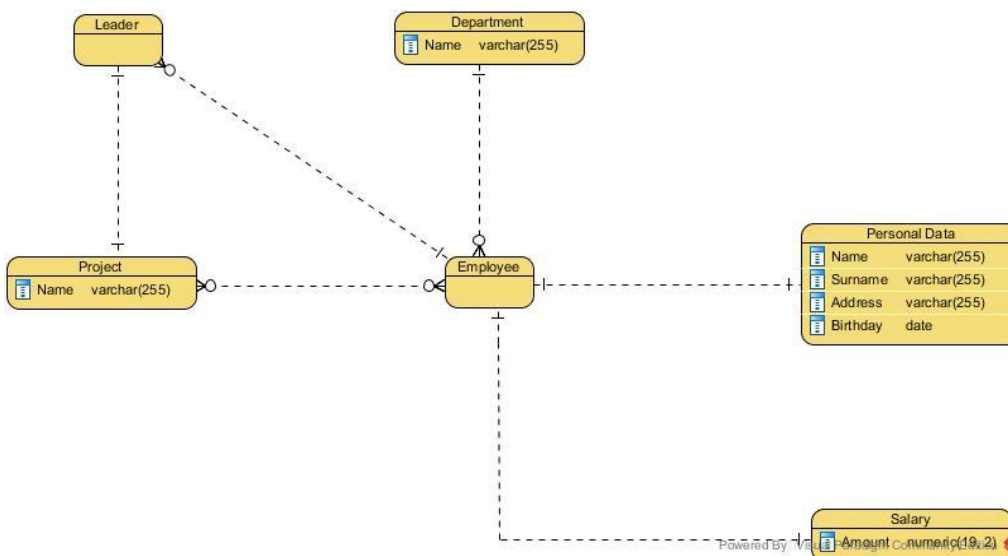
3. Modele baz danych

3.1. Model konceptualny



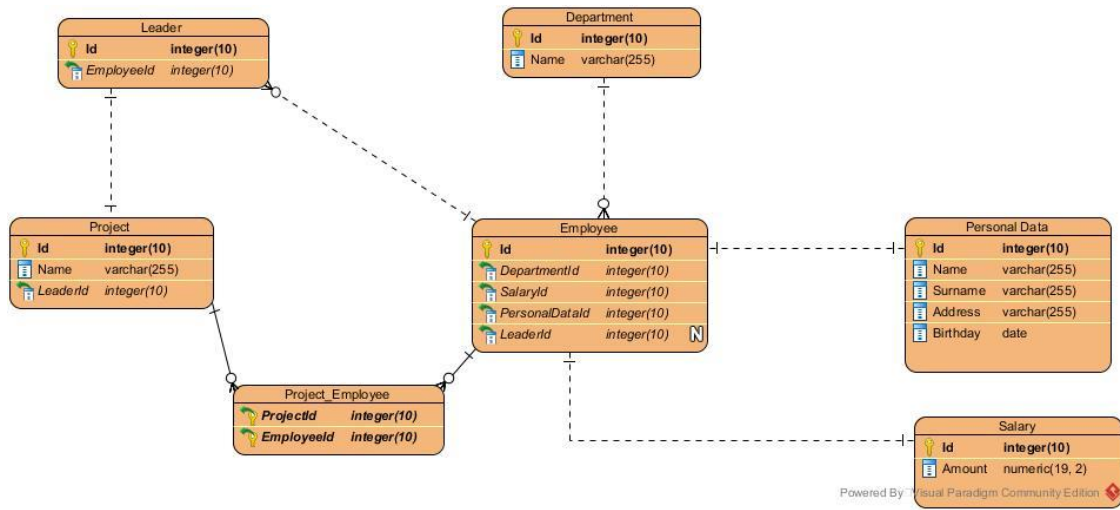
Rysunek 2 Model konceptualny bazy danych.

3.2. Model logiczny



Rysunek 3 Model logiczny bazy danych.

3.3. Model fizyczny



Rysunek 4 Model fizyczny bazy danych.

4. Implementacja baz danych w środowisku MySQL

4.1. Opis implementacji mechanizmu replikacyjnego multi-master

Implementację mechanizmu replikacyjnego rozpoczęto od przygotowania odpowiedniego środowiska, na którym będzie działać baza danych.

4.1.1 Przygotowanie środowiska

1. Instalacja VMware Workstation 14 Player na urządzeniu - hoście z systemem Windows
2. Instalacja maszyny wirtualnej z systemem Ubuntu 16.04 LTS
3. Instalacja najnowszej wersji MySQL.
4. Powielenie maszyny wirtualnej, tak aby otrzymać łącznie 3 maszyny.
5. Konfiguracja wirtualnej sieci lokalnej w celu umożliwienia komunikacji między maszynami.

4.1.2 Konfiguracja maszyn wirtualnych

Kolejnym krokiem było konfiguracja maszyn wirtualnych w celu umożliwienia replikacji.

1. Dla każdej z maszyn wygenerowano unikalne ID grupy przy użyciu komendy
`uuidgen`
oraz sprawdzono adres IP maszyn przy użyciu
`ifconfig`
Zapisane dane potrzebne będą do dalszej konfiguracji.
2. Następnie utworzono pliki konfiguracyjne przy użyciu komendy
`sudo gedit /etc/mysql/my.cnf`
3. Wypełnienie pliku konfiguracyjnego bazy danych według poniższego szablonu:

```
!includedir /etc/mysql/conf.d/  
!includedir /etc/mysql/mysql.conf.d/  
  
[mysqld]  
  
# General replication settings  
gtid_mode = ON  
enforce_gtid_consistency = ON  
master_info_repository = TABLE  
relay_log_info_repository = TABLE
```



```

binlog_checksum = NONE
log_slave_updates = ON
log_bin = binlog
binlog_format = ROW
transaction_write_set_extraction = XXHASH64
loose-group_replication_bootstrap_group = OFF
loose-group_replication_start_on_boot = OFF
loose-group_replication_ssl_mode = REQUIRED
loose-group_replication_recovery_use_ssl = 1

# Shared replication group configuration
loose-group_replication_group_name = [wygenerowany id grupy]
loose-group_replication_ip_whitelist = [ip serwerów oddzielone ","]
loose-group_replication_group_seeds = [ip serwerów z portem 33061 oddzielone ","]

# Single or Multi-primary mode? Uncomment these two lines
# for multi-primary mode, where any host can accept writes
loose-group_replication_single_primary_mode = OFF
loose-group_replication_enforce_update_everywhere_checks = ON

# Host specific replication configuration
server_id = [id serwera]
bind-address = 127.0.0.1
report_host = [ip serwera]
loose-group_replication_local_address = [ip serwera]:33061

```

4. Po zapisaniu pliku konfiguracyjnego należało zrestartować bazę danych przy użyciu komendy

```
sudo systemctl restart mysql
```

5. Kolejnym krokiem było dodanie wyjątków do zapory systemowej:

```
sudo ufw allow 33061
```

4.1.3 Konfiguracja wewnątrz MySQL

1. Po zalogowaniu się do bazy danych kolejnymi krokami były:
2. Utworzenie użytkownika bazy danych odpowiedzialnego za replikację
3. Instalacja pluginu umożliwiającego obsługę replikacji.
4. Po pomyślnej instalacji mechanizm replikacji był już praktycznie gotowy. Pozostało tylko utworzyć grupę baz danych:

```

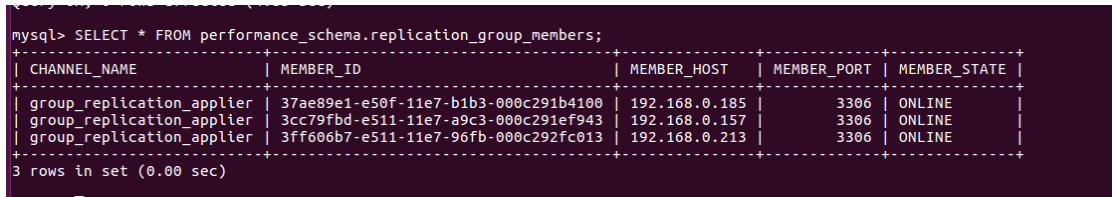
SET GLOBAL group_replication_bootstrap_group=ON;
START GROUP_REPLICATION;
SET GLOBAL group_replication_bootstrap_group=OFF;

```

5. Ostatnim krokiem było wystartowanie replikacji na pozostałych maszynach:

START GROUP_REPLICATION;

Poniżej widoczna jest tabela prezentująca bazy danych połączone w grupę replikacji (rysunek 5):



```
mysql> SELECT * FROM performance_schema.replication_group_members;
```

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	37ae89e1-e50f-11e7-b1b3-000c291b4100	192.168.0.185	3306	ONLINE
group_replication_applier	3cc79fbd-e511-11e7-a9c3-000c291ef943	192.168.0.157	3306	ONLINE
group_replication_applier	3ff606b7-e511-11e7-96fb-000c292fc013	192.168.0.213	3306	ONLINE

3 rows in set (0.00 sec)

Rysunek 5 Tabela prezentująca bazy danych.

Prezentacja działania mechanizmu replikacji dostępna jest w formie wideo pod adresem: <https://youtu.be/CZzqGOR5zJk>.

4.2. Schemat bazy danych

Schemat bazy danych jest tworzony przez mechanizm HBM2DDL biblioteki Hibernate na podstawie klas encji aplikacji klienckiej. Aby było to możliwe należało skonfigurować aplikację z użyciem pliku application.properties.

```
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/hrs
spring.datasource.username=root
spring.datasource.password=admin
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto = create
spring.jpa.hibernate.naming.physical-
strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
spring.jpa.properties.hibernate.hbm2ddl.import_files=import.sql
```

Po skonfigurowaniu aplikacji należało stworzyć klasy encji, a następnie uruchomić aplikację.

Poniżej przedstawiono klasę encji reprezentującą tabelę *project*.

```
package hrs.database.project;
```

```
import hrs.database.employee.Employee;
import lombok.Getter;
import lombok.Setter;
```

```
import javax.persistence.*;
import javax.validation.constraints.NotNull;
import java.util.Collection;
```

```

@Entity
@Getter
@Setter
public class Project {

    @Id
    @GeneratedValue
    private Long id;

    @NotNull
    private String name;

    @NotNull
    @ManyToOne
    @JoinColumn
    private Leader leader;

    @NotNull
    @ManyToMany
    @JoinTable(name = "project_employee", joinColumns = {
        @JoinColumn(name = "project_id")
    }, inverseJoinColumns = {
        @JoinColumn(name = "employee_id")
    })
    private Collection<Employee> employees;
}

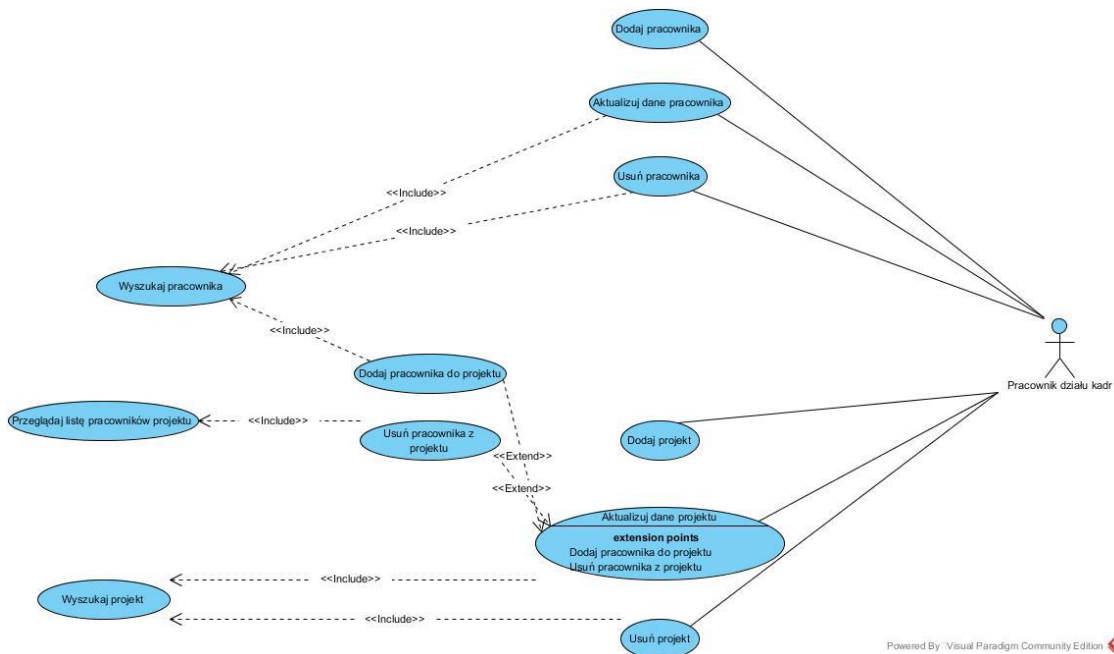
```

Kolumny tabeli reprezentowane są przez pola klasy, natomiast ich typ definiuje typ zmiennej, a właściwości - adnotacje. Powyższa klasa zawiera też definicję tabeli łączącej *project_employee* zrealizowaną za pomocą adnotacji *@JoinTable*.

5. Projekt i implementacja aplikacji klienckiej

5.1. Funkcje aplikacji - diagram przypadków użycia

Aplikacja realizuje przypadki użycia przedstawione na diagramie. Zostały zaimplementowane jedynie wybrane funkcjonalności, które umożliwiają zaprezentowanie działania mechanizmu replikacji (rysunek 6).



Rysunek 6 Diagram przypadków użycia.

5.2. Realizacja wybranych funkcjonalności

W ramach projektu zrealizowano część aplikacji odpowiedzialną za zarządzanie projektami, a więc przewidującą przypadki użycia: "dodaj projekt", "aktualizuj dane projektu" i "usuń projekt" i ich nieodłączną część "wyszukaj projekt".

5.2.1. Wyszukiwanie projektu

Jak pokazano na diagramie przypadków użycia nieodłączną częścią niektórych procesów jest wyszukiwanie projektu, które zostało zrealizowane z użyciem biblioteki Spring Data JPA.

```
package hrs.database.project;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
```

```

import java.util.List;

public interface ProjectRepo extends JpaRepository<Project, Long> {

    @Query("select new hrs.database.project.ProjectListDO(p.id, p.name,
p.leader.employee.personalData.name, p.leader.employee.personalData.surname, count(e)) from Project p
left join p.employees as e group by p.id, p.name")
    List<ProjectListDO> findProjects();

}

```

Wykonanie metody *findProjects* jest równoznaczne z wykonaniem zapytania JPQL podanego w adnotacji. Wartością zwracaną jest lista obiektów typu *ProjectListDO*.

```

package hrs.database.project;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@NoArgsConstructor
public class ProjectListDO {

    private Long id;

    private String name;

    private String leader;

    private Long size;

    public ProjectListDO(Long id, String name, String leaderName, String leaderSurname, Long size) {
        this.id = id;
        this.name = name;
        this.leader = leaderName + ' ' + leaderSurname;
        this.size = size;
    }

}

```

Następnie lista jest udostępniana użytkownikowi przez punkt dostępu typu REST zdefiniowany w metodzie *findProjects* klasy *ProjectListResource*.

```

package hrs.rest.employee;

import hrs.database.project.ProjectRepo;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

```

```

@Controller
@RequestMapping("/project")
public class ProjectListResource {

    @Autowired
    private ProjectRepo projectRepo;

    @RequestMapping
    public String findProjects(Model model) {
        model.addAttribute("projects", projectRepo.findProjects());
        return "project-list";
    }
}

```

Wartością zwracaną metody jest nazwa szablonu strony internetowej, która ma zostać pokazana w przeglądarce w odpowiedzi na zapytanie metodą GET na adres *<host>/project*. Szablon zostanie wypełniony danymi przekazanymi jako atrybut modelu o nazwie *projects*. Za połączenie modelu i widoku odpowiada biblioteka Thymeleaf. W szablonie zastosowano także bibliotekę Bootstrap, dostarczającą gotowych komponentów.

```

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
>
<head>
    <meta charset="utf-8"/>
    <title>HRS</title>
    <link th:href="@{~/bootstrap/css/bootstrap.min.css}" rel="stylesheet">
    <link th:href="@{~/css/main.css}" rel="stylesheet">
</head>
<body>
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <a href="/project/create" class="btn btn-success" role="button">Dodaj projekt</a>
        </div>
    </div>
    <div class="row">
        <div class="col-md-12">
            <table class="table-striped">
                <thead>
                    <th>Nazwa</th>

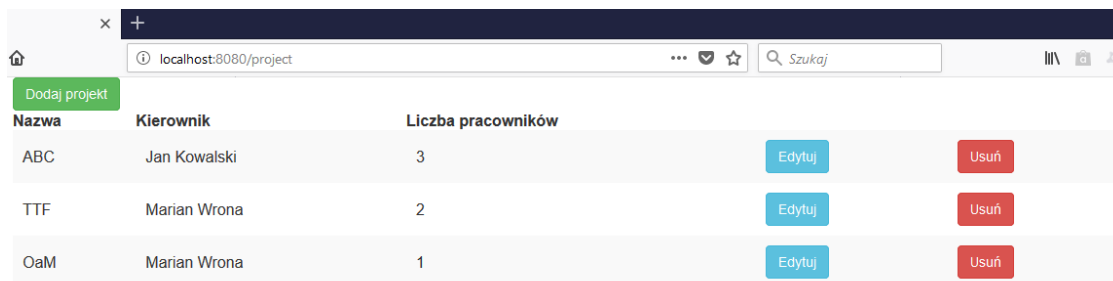
```

```

<th>Kierownik</th>
<th>Liczba pracowników</th>
<th></th>
<th></th>
</thead>
<tbody>
<tr th:each="project : ${projects}">
  <td th:text="${project.name}"></td>
  <td th:text="${project.leader}"></td>
  <td th:text="${project.size}"></td>
  <td>
    <a th:href="/project/edit?id=" + ${project.id}" class="btn btn-info"
role="button">Edytuj</a>
  </td>
  <td>
    <a th:href="/project/remove?id=" + ${project.id}" class="btn btn-danger"
role="button">Usuń</a>
  </td>
</tr>
</tbody>
</table>
</div>
</div>
<script th:src="@{~/jquery/jquery-3.2.1.min.js}"></script>
<script th:src="@{~/bootstrap/js/bootstrap.min.js}"></script>
</div>
</body>
</html>

```

Po otwarciu adresu <http://localhost:8080/project> można przeglądać listę projektów (rysunek 7).



Nazwa	Kierownik	Liczba pracowników		
ABC	Jan Kowalski	3	Edytuj	Usuń
TTF	Marian Wrona	2	Edytuj	Usuń
OaM	Marian Wrona	1	Edytuj	Usuń

Rysunek 7 Widok aplikacji listy projektów.

5.2.2. Usuwanie projektów

Jak można zauważyć w kodzie szablonu naciśnięcie przycisku *Usuń* wywoła wysłanie zapytania typu GET na adres `<host>/project?remove=<id>`, gdzie *id* zostanie zastąpione

odpowiednią wartością. Aby możliwe było odebranie zapytania, należało zdefiniować w klasie *ProjectListResource* kolejny punkt dostępu, reprezentowany metodą *removeProject*.

```
@RequestMapping("/remove")
public String removeProject(@RequestParam("id") Long id){
    projectRepo.delete(id);
    return "redirect:/project";
}
```

Metoda przyjmuje id jako parametr i usuwa z bazy odpowiedni rekord z pomocą metody *delete* zdefiniowanej w klasie *ProjectRepo* przez interfejs *JpaRepository* pochodzący z biblioteki Spring Data JPA. Nie trzeba więc tworzyć zapytania samodzielnie. Po wykonaniu usuwania aplikacja przekierowuje na adres *<host>/project*, gdzie użytkownik może zobaczyć zaktualizowaną listę projektów.

5.2.3. Edytowanie danych projektu

Aby możliwe było edytowanie danych projektu należało stworzyć nowy szablon definiujący odpowiedni formularz.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
>
<head>
    <meta charset="utf-8"/>
    <title>HRS</title>
    <link th:href="@{~/bootstrap/css/bootstrap.min.css}" rel="stylesheet">
    <link th:href="@{~/css/main.css}" rel="stylesheet">
</head>
<body>
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <h2 th:text="${activity}"></h2>
        </div>
    </div>
    <div class="row">
        <div class="col-md-12">
            <form action="/project/save" method="POST" id="edit-form" th:object="${project}">
                <input type="hidden" name="id" th:value="*{id}">
                <div class="form-group">
                    <label class="control-label col-sm-2" for="name">Nazwa*:</label>
                    <div class="col-sm-10">
                        <input type="text" class="form-control" id="name" name="name" th:value="*{name}"
                            required>
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>
```



```

        </div>
        <div class="form-group">
            <label class="control-label col-sm-2" for="leaderCandidateId">Kierownik*:</label>
            <div class="col-sm-10">
                <select name="leaderCandidateId" id="leaderCandidateId"
th:field="*{leaderCandidateId}">
                    <option th:each="employee : ${employees}"
                        th:value="{employee.id}"
                        th:text="{employee.name}"></option>
                </select>
            </div>
        </div>
        <button type="submit" class="btn btn-success">Zapisz</button>
        <a href="/project" class="btn btn-warning" role="button">Anuluj</a>
    </div>
</form>
</div>
</div>
<script th:src="@{~/jquery/jquery-3.2.1.min.js}"></script>
<script th:src="@{~/bootstrap/js/bootstrap.min.js}"></script>
<script th:src="@{~/js/person-edit.js}"></script>
</div>
</body>
</html>

```

Szablon zawiera formularz składający się z pola tekstowego zawierającego nazwę projektu oraz listy rozwijanej pozwalającej na wybór pracownika mającego zostać liderem. Do obsługi stworzonego szablonu w klasie *ProjectEditResource* zdefiniowano dwa nowe punkty dostępu:

- do odczytu danych edytowanego projektu - GET *<host>/project/edit?id=<id>*
- do zapisu zmian - POST *<host>/project/save*

```
package hrs.rest.employee;
```

```

import hrs.database.employee.Employee;
import hrs.database.employee.LeaderCandidateDO;
import hrs.database.employee.EmployeeRepo;
import hrs.database.project.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

```

```

@Controller
@RequestMapping("/project")
public class ProjectEditController {

    @Autowired
    private ProjectRepo projectRepo;

    @Autowired
    private EmployeeRepo employeeRepo;

    @Autowired
    private LeaderRepo leaderRepo;

    @RequestMapping("/edit")
    public String openProject(@RequestParam("id") Long id, Model model){
        ProjectEditDO project = new ProjectEditDO(projectRepo.findOne(id));
        model.addAttribute("project", project);
        List<LeaderCandidateDO> leaderCandidates = employeeRepo.findLeaderCandidates();
        model.addAttribute("employees", leaderCandidates);
        model.addAttribute("activity", "Edytuj projekt");
        return "project-edit";
    }

    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String saveProject(@ModelAttribute("project") ProjectEditDO projectData) {
        Project project = projectRepo.findOne(projectData.getId());
        project.setName(projectData.getName());
        if (project.getLeader().getEmployee().getId() != projectData.getLeaderCandidateId()) {
            Leader formerLeader = project.getLeader();
            project.setLeader(createLeader(projectData));
            projectRepo.save(project);
            leaderRepo.delete(formerLeader);
        } else {
            projectRepo.save(project);
        }
        return "redirect:/project";
    }

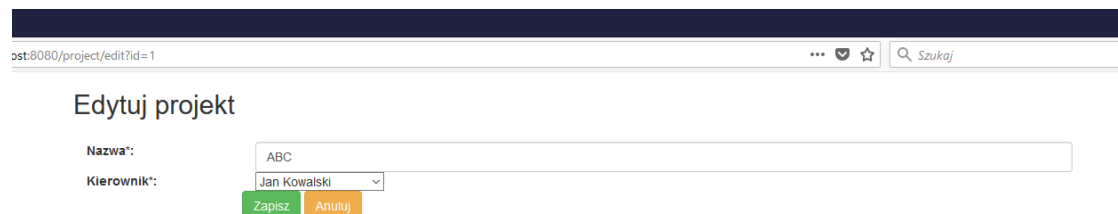
    private Leader createLeader(@ModelAttribute("project") ProjectEditDO projectData) {
        Employee chosenLeader = employeeRepo.findOne(projectData.getLeaderCandidateId());
        return leaderRepo.save(new Leader(chosenLeader));
    }
}

```

W przypadku odczytu danych oprócz danych samego projektu pobierana jest lista pracowników - potencjalnych liderów - służąca wypełnieniu listy rozwijanej. Zapis polega natomiast na uzupełnieniu odpowiednich atrybutów encji - a także, jeśli to konieczne, stworzeniu

rekordu nowego lidera i usunięciu starego. Ponownie operacje te są realizowane za pomocą gotowych mechanizmów biblioteki Spring Data JPA.

Po otwarciu adresu <http://localhost:8080/project> i naciśnięciu "Edytuj" przy pierwszym projekcie, można edytować jego dane (rysunek 8).



Rysunek 8 Widok edycji w aplikacji.

5.2.4. Dodawanie projektu

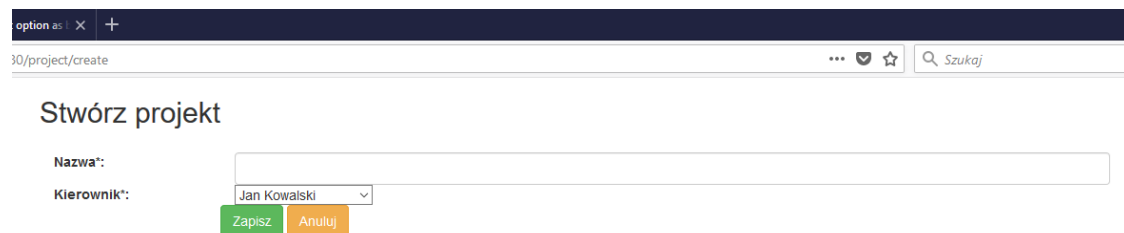
Na bazie tego samego szablonu, a jedynie dodając nowy punkt dostępu - otwierający pusty formularz - i modyfikując istniejący - służący do zapisu - w klasie *ProjectEditResource* można umożliwić użytkownikowi tworzenie nowych projektów.

```
@RequestMapping("/create")
public String createProject(Model model) {
    ProjectEditDO project = new ProjectEditDO();
    model.addAttribute("project", project);
    List<LeaderCandidateDO> leaderCandidates = employeeRepo.findLeaderCandidates();
    model.addAttribute("employees", leaderCandidates);
    model.addAttribute("activity", "Stwórz projekt");
    return "project-edit";
}

@RequestMapping(value = "/save", method = RequestMethod.POST)
public String saveProject(@ModelAttribute("project") ProjectEditDO projectData) {
    Project project = Optional.ofNullable(projectData.getId()).map(projectRepo::findOne).orElse(new Project());
    project.setName(projectData.getName());
    if (project.getLeader() == null) {
        project.setLeader(createLeader(projectData));
        project.setEmployees(new ArrayList<>());
        projectRepo.save(project);
    } else if (project.getLeader().getEmployee().getId() != projectData.getLeaderCandidateId()) {
        Leader formerLeader = project.getLeader();
        project.setLeader(createLeader(projectData));
        projectRepo.save(project);
        leaderRepo.delete(formerLeader);
    }
}
```

```
} else {  
  projectRepo.save(project);  
}  
return "redirect:/project";  
}
```

Po naciśnięciu przycisku "Dodaj projekt" można zobaczyć następujący formularz.



option as | × +

30/project/create ... ♥ ☆ Szukaj

Stwórz projekt

Nazwa:

Kierownik: Jan Kowalski ▾

Zapisz Anuluj

Rysunek 9 Widok następnego formularza w aplikacji.

6. Wdrożenie i testowanie aplikacji

6.1. Instalowanie i konfiguracja aplikacji

Instrukcja dla systemu operacyjnego Linux Ubuntu 17.10.

1. Należy przygotować bazę danych zgodnie z opisem z rozdziału "Opis implementacji mechanizmu relokacyjnego multi-master".

2. W konsoli bazy danych należy uruchomić polecenie

```
create database hrs;
```

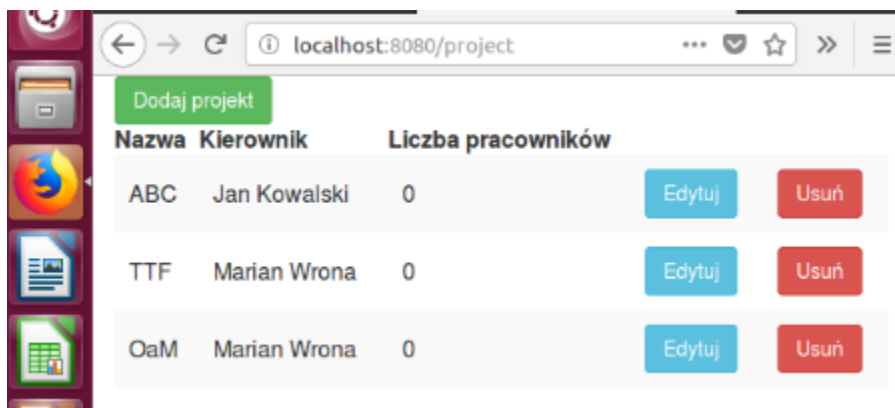
3. W folderze głównym aplikacji uruchomić skrypt run.sh poleceniem

```
sh run.sh
```

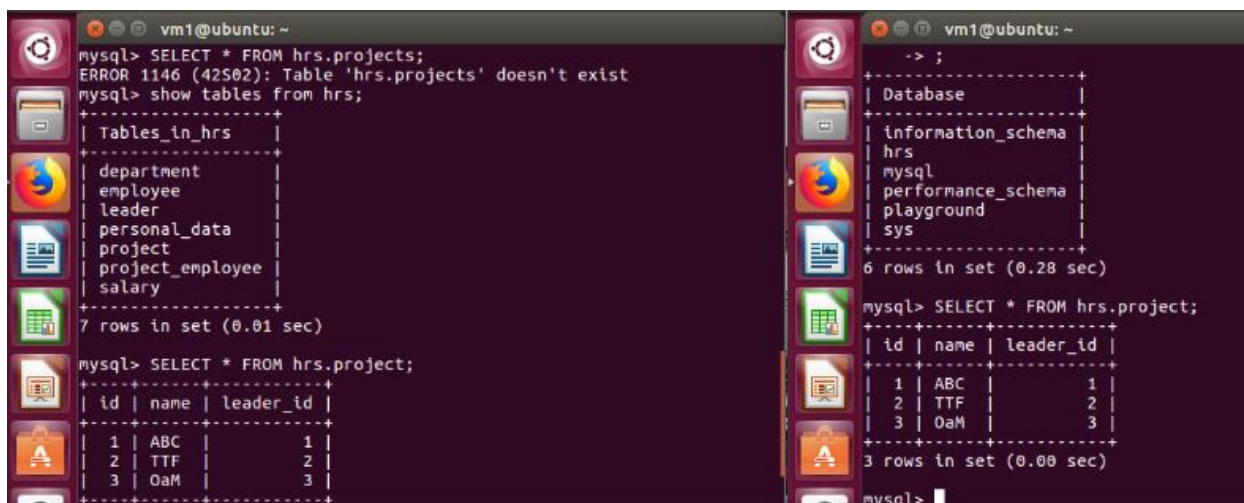
4. Aplikacja będzie odtąd dostępna w przeglądarce internetowej pod adresem <http://localhost:8080/project>

6.2. Testy akceptacyjne wybranych operacji

Testowanie odbyło się na trzech maszynkach wirtualnych skonfigurowanych ze sobą w architekturze mulit-master. Poniżej na rysunku jest przedstawiony widok naszej aplikacji (zawartość bazy pierwszej maszyny) jak i stan na pozostałych dwóch bazach (rysunek 10) (rysunek 11). Każda operacja była wykonywana na różnej maszynie wirtualnej tak, żeby pokazać działanie replikacji multi-master.



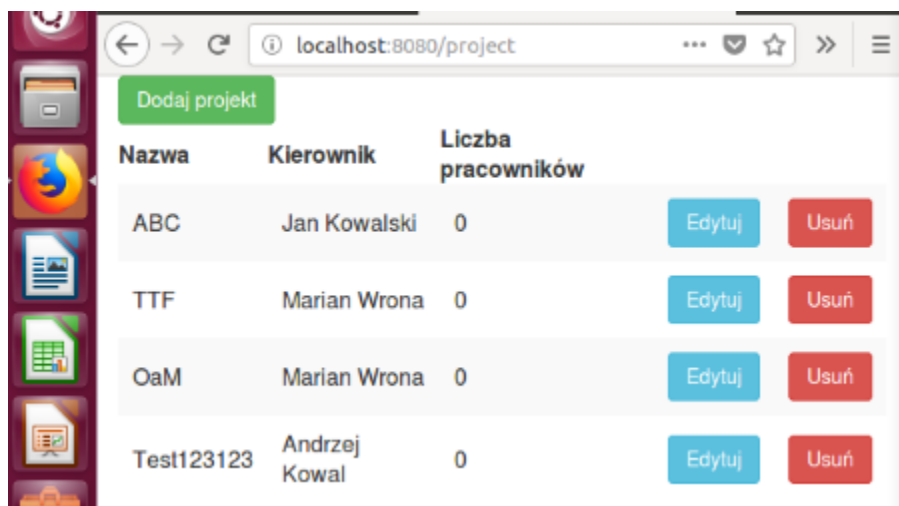
Rysunek 10 Startowy widok aplikacji.



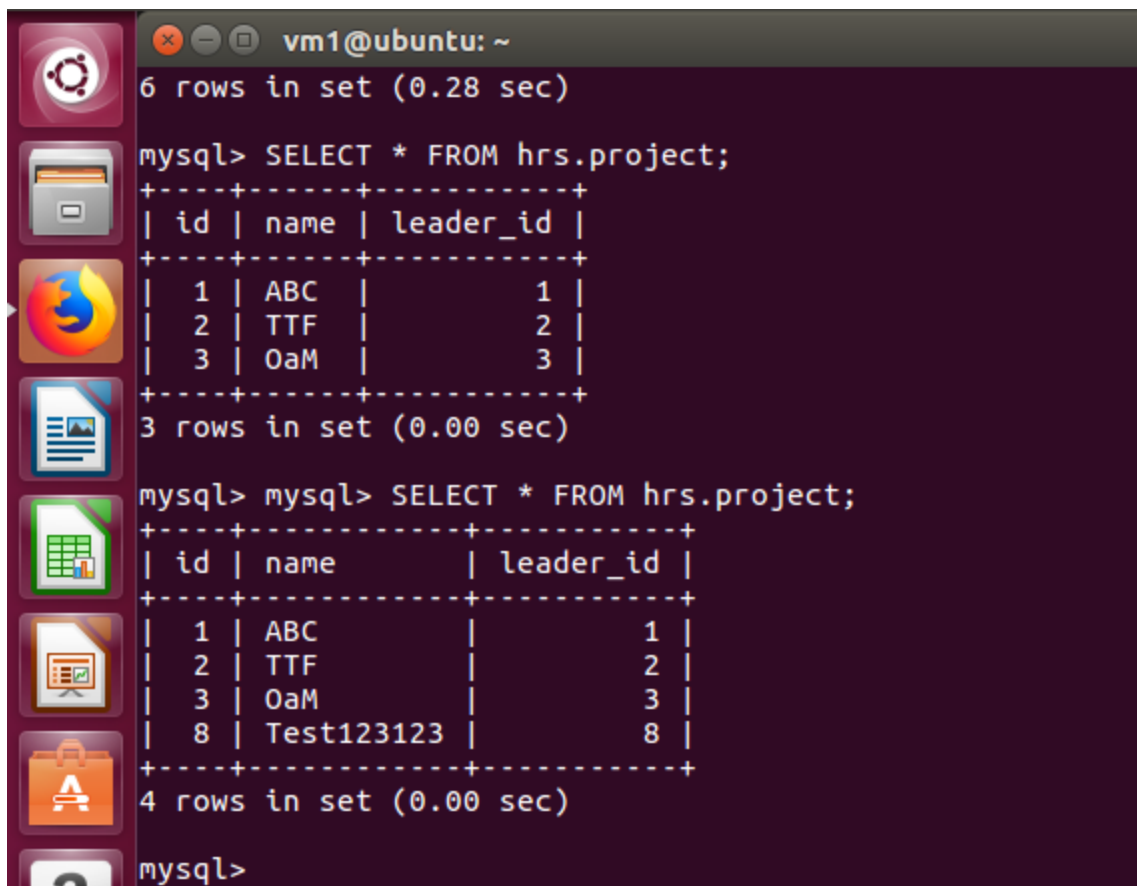
Rysunek 11 Stan początkowy baz danych.

6.2.1. Operacja dodawania

Po dodaniu nowego projektu możemy zobaczyć, że od razu w czasie rzeczywistym otrzymaliśmy zmianę zarówno na widoku naszej aplikacji na jednej bazie jak i pozostałych bazach (rysunek 12) (rysunek 13).



Rysunek 12 Widok aplikacji po dodaniu nowego projektu.



```
vm1@ubuntu: ~
6 rows in set (0.28 sec)

mysql> SELECT * FROM hrs.project;
+-----+-----+-----+
| id | name | leader_id |
+-----+-----+-----+
| 1 | ABC | 1 |
| 2 | TTF | 2 |
| 3 | OaM | 3 |
+-----+-----+-----+
3 rows in set (0.00 sec)

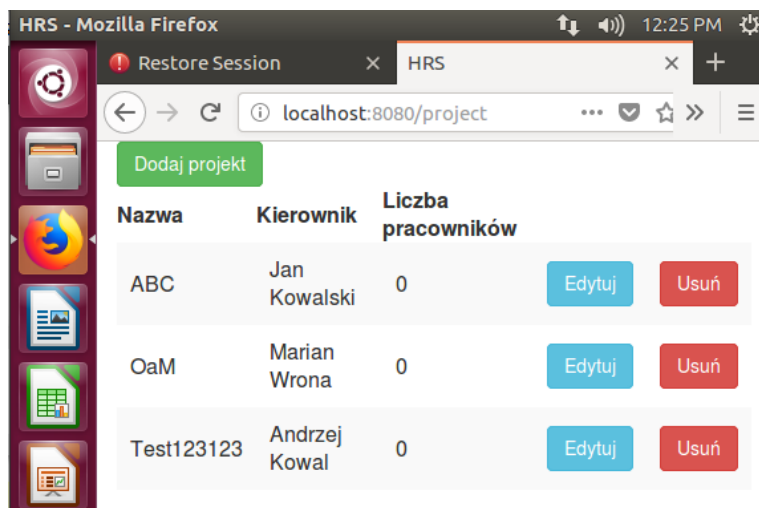
mysql> mysql> SELECT * FROM hrs.project;
+-----+-----+-----+
| id | name | leader_id |
+-----+-----+-----+
| 1 | ABC | 1 |
| 2 | TTF | 2 |
| 3 | OaM | 3 |
| 8 | Test123123 | 8 |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

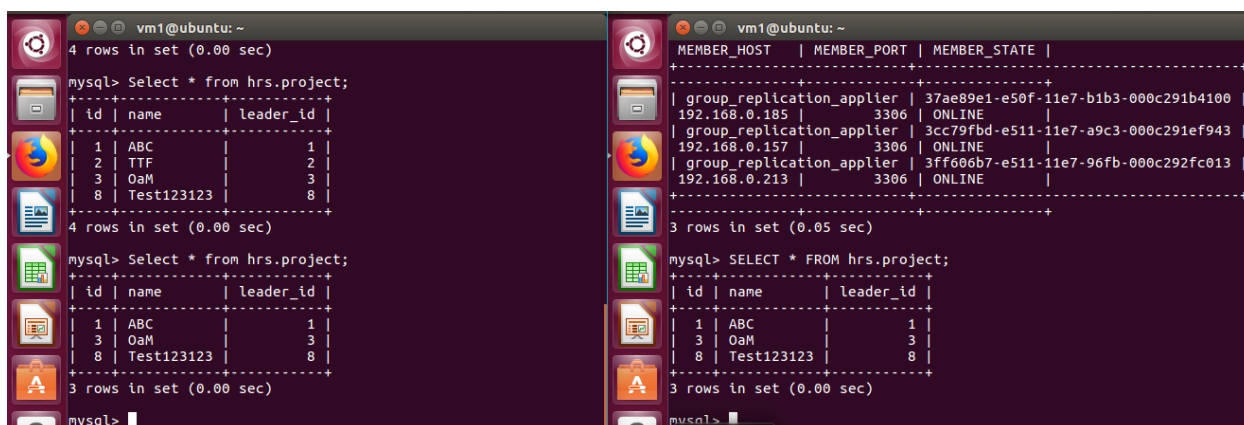
Rysunek 13 Stan bazy po dodaniu nowego projektu.

6.2.2 Operacja usuwania

Po wykonaniu operacji usuwania możemy zobaczyć zmianę na widoku aplikacji oraz zmianę w stanie baz danych (rysunek 14) (rysunek 15).



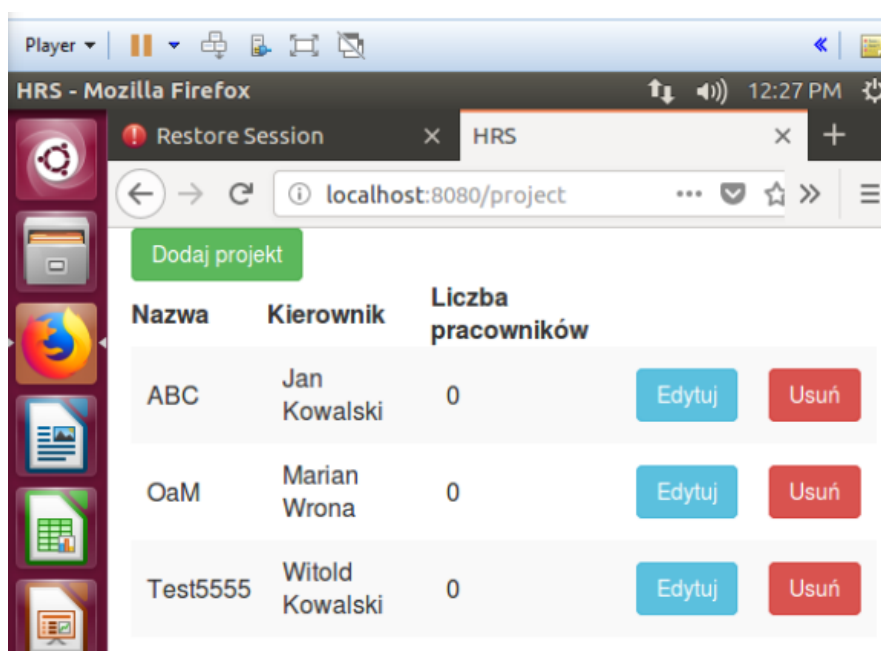
Rysunek 14 Widok aplikacji po usunięciu projektu.



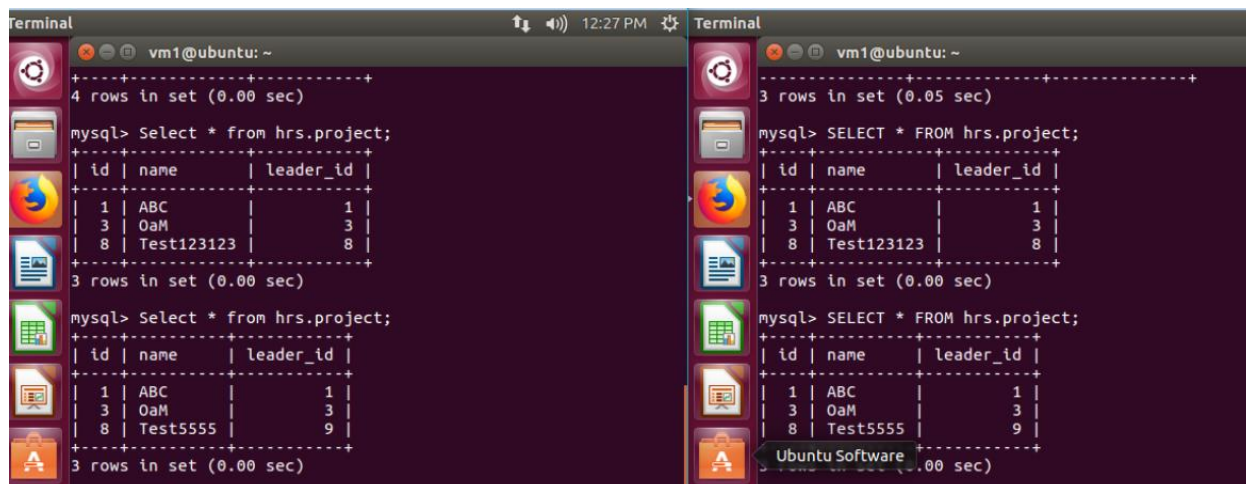
Rysunek 15 Stan bazy po usunięciu projektu.

6.2.3. Operacja Edycji

Na poniższych rysunkach możemy zobaczyć widok aplikacji oraz stan bazy po wykonaniu operacji edycji (rysunek 16) (rysunek 17).



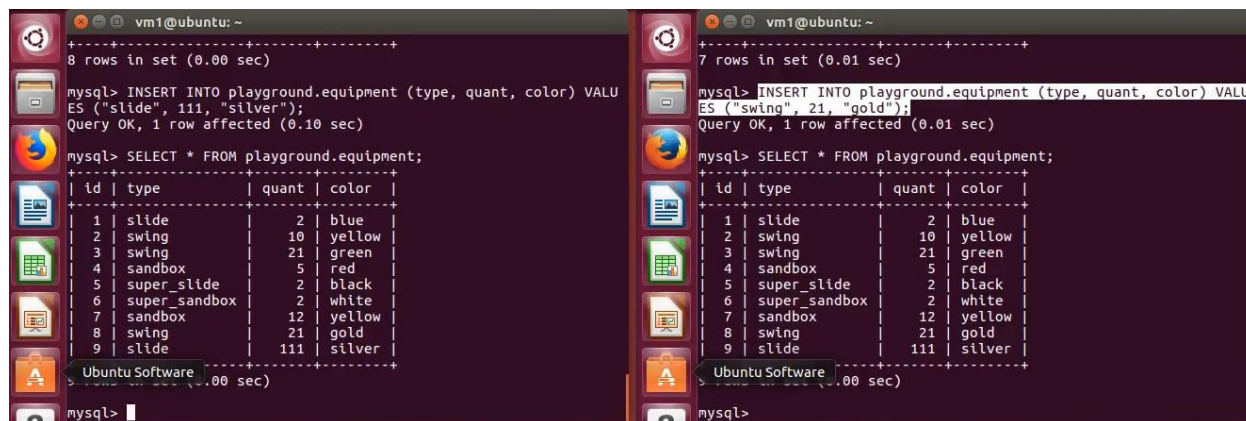
Rysunek 16 Widok aplikacji po operacji edycji.



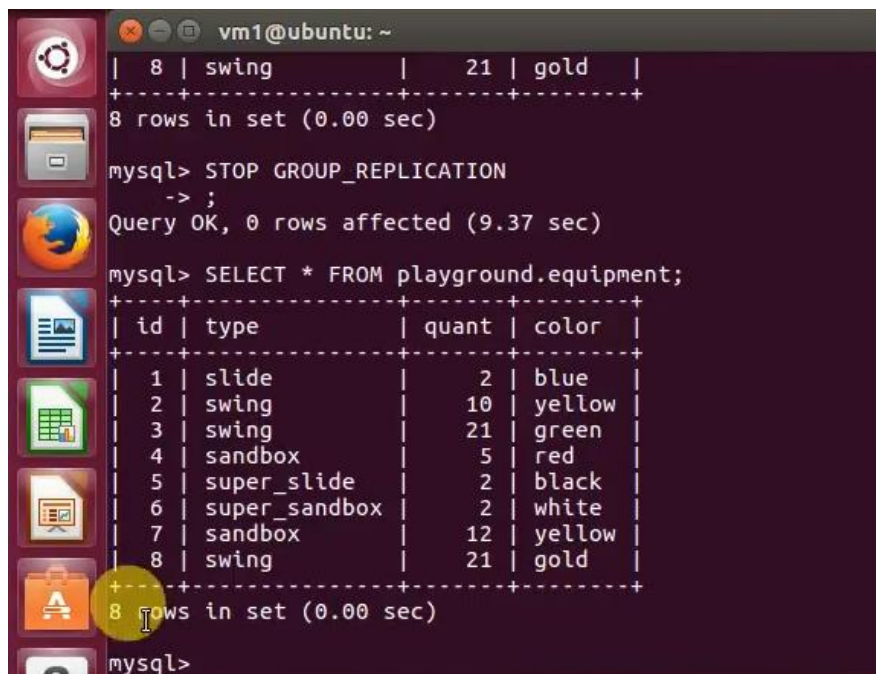
Rysunek 17 Widok stanu bazy po wykonaniu operacji edycji.

6.2.4. Operacja odłączenie urządzenia

Na rysunku (rysunek 18) możemy zauważyć operacje dodawania rekordu do bazy oraz wyświetlenie zawartości tych baz. Jak możemy zauważyć dodany rekord pojawił się na podłączonych ze sobą bazach. Natomiast na rysunku (rysunek 19) możemy zobaczyć, że replikacja na bazie została zatrzymana i wprowadzony nowy rekord się nie pojawił.



Rysunek 18 Replikacja bazy na dwóch urządzeniach.



```
vm1@ubuntu: ~  
+----+-----+-----+-----+  
| 8 | swing |      21 | gold |  
+----+-----+-----+-----+  
8 rows in set (0.00 sec)  
  
mysql> STOP GROUP_REPLICATION  
-> ;  
Query OK, 0 rows affected (9.37 sec)  
  
mysql> SELECT * FROM playground.equipment;  
+----+-----+-----+-----+  
| id | type      | quant | color |  
+----+-----+-----+-----+  
| 1 | slide     |      2 | blue  |  
| 2 | swing     |     10 | yellow|  
| 3 | swing     |     21 | green |  
| 4 | sandbox   |      5 | red   |  
| 5 | super_slide |      2 | black |  
| 6 | super_sandbox |      2 | white |  
| 7 | sandbox   |     12 | yellow|  
| 8 | swing     |     21 | gold  |  
+----+-----+-----+-----+  
8 rows in set (0.00 sec)  
  
mysql>
```

Rysunek 19 Brak replikacji na urządzeniu.

Poniżej podaje do nagranych przez nas źródeł, gdzie możemy zobaczyć cały proces replikacji danych w architekturze multi-master.

- źródło: https://www.youtube.com/watch?v=RNjxY1Gp_18&feature=youtu.be
- źródło: <https://www.youtube.com/watch?v=knksTXE6U4A&feature=youtu.be>
- źródło: <https://www.youtube.com/watch?v=dE7Ef1QGg0o&feature=youtu.be>
- źródło: <https://www.youtube.com/watch?v=CZzqGOR5zJk&feature=youtu.be>

7. Podsumowanie

W ramach zajęć projektowych udało się skonfigurować i użyć w działającej aplikacji internetowej mechanizm replikacji synchronicznej typu multimaster.

Bazę danych i mechanizm replikacji oparto o system MySQL i dostępne w nim narzędzia konsolowe i pliki konfiguracyjne. Ponadto stworzono aplikację kliencką stworzoną na platformie Java z użyciem bibliotek Spring Framework. W ramach aplikacji stworzono także obiektowy schemat bazy danych, który został przetłumaczony na stosowany w systemie MySQL system relacyjny z użyciem mechanizmów ORM biblioteki Hibernate.

Przy projektowaniu i implementacji systemu posługiwano się oprogramowaniem Visual Paradigm (do tworzenia modeli) i IntelliJ IDEA (do pisania kodu języka Java).

Celem przetestowania mechanizmu replikacji aplikację kliencką wraz z odpowiednio skonfigurowaną bazą danych umieszczono na trzech maszynach wirtualnych z systemem Ubuntu stworzonych z użyciem oprogramowania VMware i zrealizowano na każdej z nich operacje CRUD. Replikacja przebiegła pomyślnie zarówno przy wszystkich maszynach aktywnych, jak i po wyłączeniu replikacji na wybranych maszynach i uruchomieniu jej ponownie.

Wobec powyższego można uznać, że projekt zakończył się sukcesem.

Literatura

- [1] Wrycza S. i in., *Język UML 2.0 w modelowaniu systemów informatycznych*, Helion, Gliwice 2005, s. 33-50
- [2] <https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-replication-multi-master.html>
- [3] <https://docs.spring.io/spring/docs/5.0.1.RELEASE/spring-framework-reference/core.html#spring-core>
- [4] <http://docs.spring.io/spring-data/jpa/docs/1.10.4.RELEASE/reference/html/>
- [5] https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3564/85378_conceptual,1.html