Politechnika Świętokrzyska Wydział Elektrotechniki, Automatyki i Informatyki Zakład Zastosowań Informatyki	
Technologie Obiektowe – Projekt	
Temat: Porównanie Hibernate i MyBatis	**Adrian Jakubczyk** 1ID21A

1. **Hibernate i MyBatis.**

MyBatis	Hibernate
Opis	
MyBatis jest frameworkiem o otwartym kodzie źródłowym, cechuje go prostota użytkowania i wydajność. Dostarcza funkcję automatycznego wiązania, która mapuje zapytania SQL z obiektami wybranego języka programowania. Używany jest język SQL który jest łatwy do zrozumienia i użytkowania przez deweloperów. MyBatis wspiera niezależne interfejsy, zapisane procedury, dynamiczny SQL, itp. MyBatis nie jest modelem relacyjno obiektowym (ORM), co pozwala uniknąć wszelkich problemów związanych z mapowaniem.	Hibernate jest narzędziem o otwartym kodzie źródłowym, które służy do mapowania obiektowo-relacyjnego (ORM). Odwzorowuje obiekty z domeny aplikacji na relacje bazy danych i vice versa. Jest frameworkiem Javy, który implementuje specyfikacje Java Persistence API do łatwej interakcji aplikacji Javy z bazą danych. Korzysta z własnego języka zapytań - HQL (Hibernate Query Language), w związku z czym, cechuje się skalowalnością i łatwością migracji. Zapewnia przydatne funkcje, doskonałe mapowanie, niezależność danych, przenośność co zwiększa szybkość i łatwość procesu tworzenia oprogramowania.
Pierwsze wrażenia	
MyBatis jest łatwy do pojęcia i w większości składa się z pisania zapytań SQL.	Hibernate jest dużym i złożonym frameworkiem co może z początku sprawiać kłopoty.
Zależność od baz danych	
MyBatis używa języka SQL, który może być zależny od używanej bazy danych.	Hibernate używa HQL, który jest nie jest zależny od bazy danych.
Używanie zapisanych procedur	
Użycie zapisanych procedur jest proste w przypadku MyBatis ze względu na implementację użycia języka SQL.	Użycie zapisanych procedur może być problematyczne.
Zmiana typu bazy danych	
Ze względu na korzystanie z SQL przez MyBatis nie ma możliwości łatwej zmiany bazy danych.	W przypadku Hibernate'a zmiana bazy danych jest łatwa, ponieważ korzysta z HQL, który nie jest od niej zależny.

Mapowanie	
W bardziej skomplikowanych przypadkach, użytkownik musi napisać zapytanie i obsłużyć mapowanie zbioru wynikowego.	Hibernate posiada wbudowany mechanizm mapowania, więc użytkownik nie musi się o to martwić.
Raporty i statystyki	
MyBatis nie posiada własnego systemu raportów, konieczne jest użycie log4j.	Hibernate posiada własny system raportów i statystyk.
Obsługa Data Access Object (DAO)	
Tworzenie interfejsu dostępu do danych (DAO) jest trudniejsze w przypadku MyBatis.	Tworzenie interfejsu dostępu do danych (DAO) w porównaniu do MyBatis jest łatwiejsze.
Pamięć podręczna drugiego stopnia	
Mechanizm ten nie jest włączony domyślnie i wymaga dodatkowej konfiguracji	Hibernate posiada dobrze działającą pamięć podręczną drugiego stopnia.

Najważniejsze punkty:

- Hibernate skupia się na obiektach i mapowaniu ich do bazy danych przy niewielkim wysiłku ze strony dewelopera, który chciałby się skupić na warstwie biznesowej aplikacji.
- MyBatis jest skoncentrowany na bazie danych.
- MyBatis jest łatwy w użytkowaniu dla nowych twórców oprogramowania, ponieważ jest niewielkim narzędziem i korzysta głównie z SQL, gdzie Hibernate jest bardziej skomplikowanym i większym narzędziem.
- MyBatis jest zwykle używany w przypadkach gdzie model danych nie jest idealnie odwzorowany na model obiektu i wymagana jest całkowita kontrola nad zapytaniami SQL. Hibernate jest używany, gdy deweloper ma całkowitą kontrolę nad bazą danych i mapowanie danych i obiektów jest odpowiednio zsynchronizowane.
- Hibernate mapuje klasy Javy do tabel bazy danych a MyBatis mapuje wyrażenia SQL do metod Javy.
- W przypadku pobierania wyników skomplikowanych zapytań, MyBatis działa znakomicie. Hibernate musi najpierw załadować cały graf obiektów, proces ten może być skomplikowany i długi.

Powyższe porównanie jasno określa różnice między danymi rozwiązaniami. Zarówno MyBatis jak i Hibernate są narzędziami open-source używanymi na rynku. Wybór zależy od konkretnej sytuacji i preferencji użytkownika. MyBatis jest skoncentrowany na danych i jest używany w przypadku, gdy chcemy stworzyć i utrzymywać własną bazę danych SQL. Hibernate jest używany w przypadku gdy użytkownik chce się skupić jedynie na warstwie biznesowej.

2. Wykorzystane technologie.

- Java 8
- Spring Boot
- Jmeter
- MariaDB
- Lombok

3. Mechanizmy mapowania.

3.1. MyBatis:

- Wykorzystanie interfejsu i odpowiednich adnotacji

```
@Mapper
public interface AccountMapper {
    @Insert("INSERT INTO Accounts(email,nickname,login,password hash,ban expired at) VALUES
(#{email},#{nickname},#{login},#{passwordHash},#{ban expired at}) ")
    @Options(useGeneratedKeys = true, keyProperty = "id")
    void insertAccount(@Param("account") Account account);
    @Update("UPDATE Accounts SET email = #{email}, nickname =
#{nickname},login=#{login},password hash = #{passwordHash},ban expired at=#{banExpiredAt}
WHERE id = ${id}")
    void updateAccount(@Param("account") Account account);
    @Select("SELECT id,email,nickname,login,password_hash as passwordHash,ban_expired_at as
banExpiredAt,created_at as createdAt, modified_at as modifiedAt, deleted_at as deletedAt
WHERE id = ${id}")
    Account getAccountById(@Param("id") Long id);
    @Select("SELECT id,email,nickname,login,password hash as passwordHash,ban expired at as
banExpiredAt,created at as createdAt, modified at as modifiedAt, deleted at as deletedAt from
accounts")
    List<Account> getAllAccounts();
    @Delete("DELETE FROM Accounts WHERE id = ${id}")
    void deleteAccount(@Param("id") Long id);
}
```

- Wykorzystanie pliku XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-
3-mapper.dtd">
<mapper namespace="Account">
    <select id="getAllAccounts" parameterType="java.lang.Long"
resultMap="pl.psk.to.mmo.model.Account">
        SELECT
               id,
               email,
               nickname,
               login,
               password hash as passwordHash,
               ban_expired_at as banExpiredAt,
               created_at as createdAt,
               modified_at as modifiedAt,
               deleted at as deletedAt
    </select>
    <select id="getAccountById" parameterType="java.lang.Long"
resultType="pl.psk.to.mmo.model.Account">
        SELECT
            id,
            email,
            nickname,
            login,
            password_hash as passwordHash,
            ban_expired_at as banExpiredAt,
            created at as createdAt,
            modified at as modifiedAt,
            deleted_at as deletedAt
            WHERE id = ${id}
    </select>
    <insert
            id="insertAccount"
            parameterType="pl.psk.to.mmo.model.Account"
            flushCache="true"
            timeout="20">
        INSERT INTO Accounts(email,nickname,login,password hash,ban expired at)
        VALUES
(#{account.email},#{account.nickname},#{account.login},#{account.passwordHash},#{account.ban e
xpired at})

    </insert>
    <update id="updateAccount" parameterType="pl.psk.to.mmo.model.Account">
        UPDATE Accounts
        SET email = #{account.email},
```

```
            nickname = #{account.nickname},
            login=#{account.login},
            password_hash = #{account.passwordHash},
            ban_expired_at=#{account.banExpiredAt}
        WHERE id = ${account.id}
    </update>
    <delete id="deleteAccount" parameterType="java.lang.Long">
        DELETE FROM Accounts WHERE id = ${id}
    </delete>
</mapper>
```

3.2. Hibernate:

- Wykorzystanie adnotacji i usługi

Model:

```
@Entity
@Table(name = "EMPLOYEE")
public class Employee {
    @Id
    @GeneratedValue
    @Column(name = "id")
    private int id;

    @Column(name = "first name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "salary")
    private int salary;

    public Employee() {}
    public Employee(String fname, String lname, int salary) {
        this.firstName = fname;
        this.lastName = lname;
        this.salary = salary;
    }
}
```

Sesja i transakcje:

```
public class ManageEmployee {
    private static SessionFactory factory;
    public static void main(String[] args) {

        try {
            factory = new AnnotationConfiguration().
                    configure().
                    //addPackage("com.xyz") //add package if used.
                            addAnnotatedClass(Employee.class).
                            buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Failed to create sessionFactory object." + ex);
            throw new ExceptionInInitializerError(ex);
        }

        ManageEmployee ME = new ManageEmployee();

        /* Add few employee records in database */
        Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
        Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
        Integer empID3 = ME.addEmployee("John", "Paul", 10000);

        /* List down all the employees */
        ME.listEmployees();

        /* Update employee's records */
        ME.updateEmployee(empID1, 5000);

        /* Delete an employee from the database */
        ME.deleteEmployee(empID2);

        /* List down new list of the employees */
        ME.listEmployees();
```

```
    }

    /* Method to CREATE an employee in the database */
    public Integer addEmployee(String fname, String lname, int salary){
        Session session = factory.openSession();
        Transaction tx = null;
        Integer employeeID = null;

        try {
            tx = session.beginTransaction();
            Employee employee = new Employee();
            employee.setFirstName(fname);
            employee.setLastName(lname);
            employee.setSalary(salary);
            employeeID = (Integer) session.save(employee);
            tx.commit();
        } catch (HibernateException e) {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
        return employeeID;
    }

    /* Method to  READ all the employees */
    public void listEmployees( ){
        Session session = factory.openSession();
        Transaction tx = null;

        try {
            tx = session.beginTransaction();
            List employees = session.createQuery("FROM Employee").list();
            for (Iterator iterator = employees.iterator(); iterator.hasNext();){
                Employee employee = (Employee) iterator.next();
                System.out.print("First Name: " + employee.getFirstName());
                System.out.print("  Last Name: " + employee.getLastName());
                System.out.println("  Salary: " + employee.getSalary());
            }
            tx.commit();
        } catch (HibernateException e) {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
    }

    /* Method to UPDATE salary for an employee */
    public void updateEmployee(Integer EmployeeID, int salary ){
        Session session = factory.openSession();
        Transaction tx = null;

        try {
            tx = session.beginTransaction();
            Employee employee = (Employee)session.get(Employee.class, EmployeeID);
            employee.setSalary( salary );
            session.update(employee);
            tx.commit();
        } catch (HibernateException e) {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
    }

    /* Method to DELETE an employee from the records */
    public void deleteEmployee(Integer EmployeeID){
        Session session = factory.openSession();
        Transaction tx = null;

        try {
            tx = session.beginTransaction();
            Employee employee = (Employee)session.get(Employee.class, EmployeeID);
            session.delete(employee);
            tx.commit();
```

```
        } catch (HibernateException e) {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
    }
}
```

https://www.tutorialspoint.com/hibernate/hibernate_annotations.htm

- Wykorzystanie pliku XML

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name = "Employee" table = "EMPLOYEE">

        <meta attribute = "class-description">
            This class contains the employee detail.
        </meta>

        <id name = "id" type = "int" column = "id">
            <generator class="native"/>
        </id>

        <property name = "firstName" column = "first_name" type = "string"/>
        <property name = "lastName" column = "last_name" type = "string"/>
        <property name = "salary" column = "salary" type = "int"/>

    </class>
</hibernate-mapping>
```

https://www.tutorialspoint.com/hibernate/hibernate_examples.htm

4. Projekt.

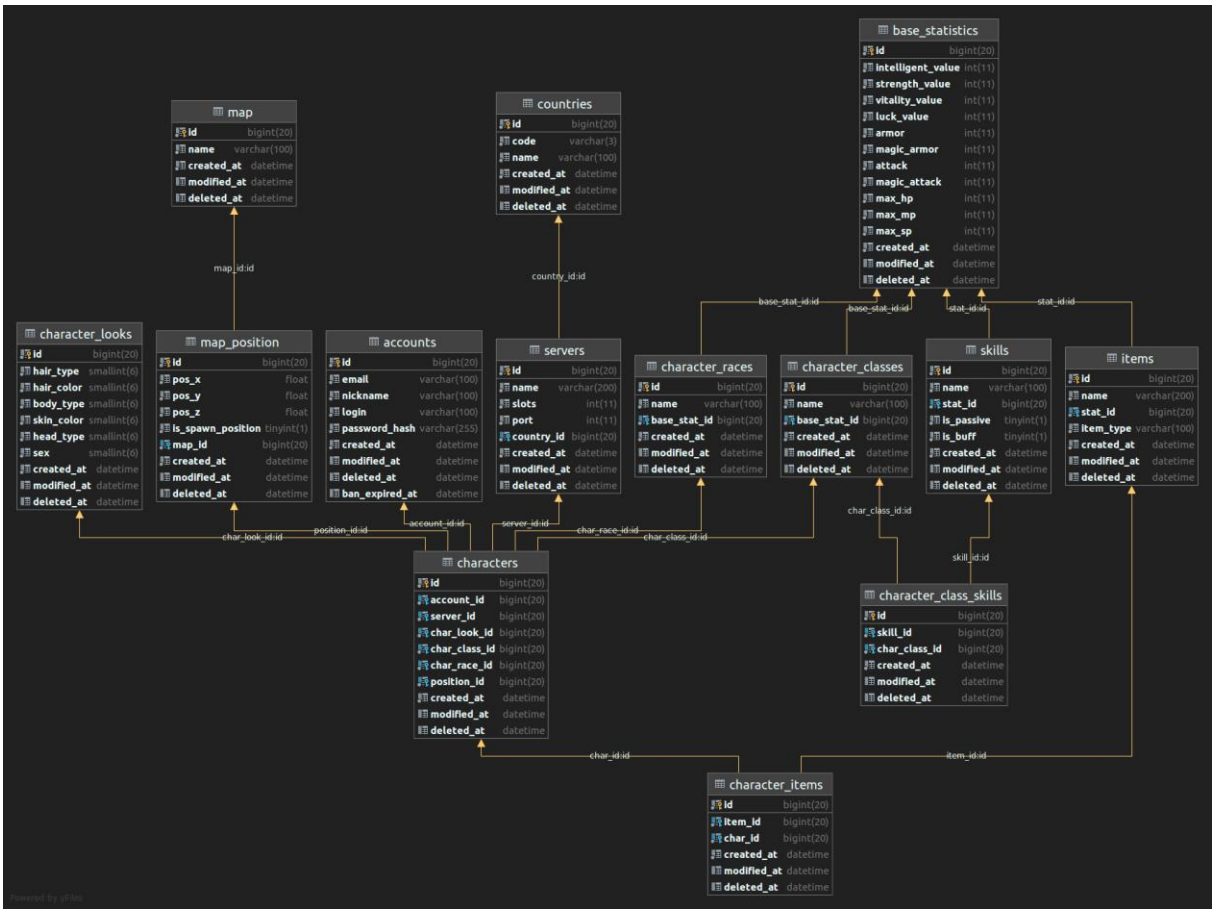
4.1. Schemat bazy danych.

```sql
CREATE DATABASE mmo;

USE mmo;

CREATE TABLE countries
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    code VARCHAR(3) NOT NULL,
    name VARCHAR(100) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL
);

CREATE TABLE servers
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(200) NOT NULL,
    slots INT NOT NULL DEFAULT 1,
    port INT NOT NULL DEFAULT 2000,
    country_id BIGINT NOT NULL,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,
    CONSTRAINT country_id_server_fk FOREIGN KEY(country_id) REFERENCES countries(id)
);

CREATE TABLE accounts
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(100) NOT NULL,
    nickname VARCHAR(100) NOT NULL,
    login VARCHAR(100) NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,
    ban_expired_at DATETIME NULL
);

CREATE TABLE base_statistics
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    intelligent_value INT NOT NULL DEFAULT 0,
    strength_value INT NOT NULL DEFAULT 0,
    vitality_value INT NOT NULL DEFAULT 0,
    luck_value INT NOT NULL DEFAULT 0,
    armor INT NOT NULL DEFAULT 0,
    magic_armor INT NOT NULL DEFAULT 0,
    attack INT NOT NULL DEFAULT 0,
    magic_attack INT NOT NULL DEFAULT 0,
    max_hp INT NOT NULL DEFAULT 100 COMMENT 'Health points',
    max_mp INT NOT NULL DEFAULT 200 COMMENT 'Mana points',
    max_sp INT NOT NULL DEFAULT 500 COMMENT 'Stamina points',
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
```

```
    deleted_at DATETIME NULL
);


CREATE TABLE character_looks
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    hair_type SMALLINT NOT NULL DEFAULT 0,
    hair_color SMALLINT NOT NULL DEFAULT 0,
    body_type SMALLINT NOT NULL DEFAULT 0,
    skin_color SMALLINT NOT NULL DEFAULT 0,
    head_type SMALLINT NOT NULL DEFAULT 0,
    sex SMALLINT NOT NULL DEFAULT 0 CHECK ( sex = 1 OR sex = 0 ) ,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL
);

CREATE TABLE character_races
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    base_stat_id BIGINT NOT NULL,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,
    CONSTRAINT base_stat_char_r_fk FOREIGN KEY(base_stat_id) REFERENCES base_statistics(id)
);

CREATE TABLE character_classes
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    base_stat_id BIGINT NOT NULL,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,
    CONSTRAINT base_stat_char_c_fk FOREIGN KEY(base_stat_id) REFERENCES base_statistics(id)
);

CREATE TABLE map
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL
);

CREATE TABLE map_position
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    pos_x FLOAT NOT NULL DEFAULT 0.0,
    pos_y FLOAT NOT NULL DEFAULT 0.0,
    pos_z FLOAT NOT NULL DEFAULT 0.0,
    is_spawn_position BOOLEAN NOT NULL DEFAULT 0,
    map_id BIGINT NOT NULL,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,
```

```
    CONSTRAINT map_id_map_fk FOREIGN KEY(map_id) REFERENCES map(id)
);

CREATE TABLE characters
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    account_id BIGINT NOT NULL,
    server_id BIGINT NOT NULL,
    char_look_id BIGINT NOT NULL,
    char_class_id BIGINT NOT NULL,
    char_race_id BIGINT NOT NULL,
    position_id BIGINT NOT NULL,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,

    CONSTRAINT account_id_char_fk FOREIGN KEY(account_id) REFERENCES accounts(id),
    CONSTRAINT server_id_char_fk FOREIGN KEY(server_id) REFERENCES servers(id),
    CONSTRAINT char_look_id_fk FOREIGN KEY(char_look_id) REFERENCES character_looks(id),
    CONSTRAINT char_class_id_fk FOREIGN KEY(char_class_id) REFERENCES character_classes(id),
    CONSTRAINT char_race_id_fk FOREIGN KEY(char_race_id) REFERENCES character_races(id),
    CONSTRAINT position_id_pos_fk FOREIGN KEY(position_id) REFERENCES map_position(id)
);

CREATE TABLE skills
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL ,
    stat_id BIGINT NOT NULL,
    is_passive BOOLEAN NOT NULL DEFAULT 0,
    is_buff BOOLEAN NOT NULL DEFAULT 0,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,
    CONSTRAINT stat_id_skill_fk FOREIGN KEY(stat_id) REFERENCES base_statistics(id)
);

CREATE TABLE character_class_skills
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    skill_id BIGINT NOT NULL,
    char_class_id BIGINT NOT NULL ,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,

    CONSTRAINT skill_id_chr_skill_fk FOREIGN KEY(skill_id) REFERENCES skills(id),
    CONSTRAINT char_class_id_chr_skill_fk FOREIGN KEY(char_class_id) REFERENCES character_classes(id)
);



CREATE TABLE items
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(200) NOT NULL,
    stat_id BIGINT NOT NULL,
    item_type VARCHAR(100) NOT NULL COMMENT 'ENUM FOR ITEM TYPE',
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,
```

```
    CONSTRAINT stat_id_item_fk FOREIGN KEY(stat_id) REFERENCES base_statistics(id)
);

CREATE TABLE character_items
(
    id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    item_id BIGINT NOT NULL,
    char_id BIGINT NOT NULL ,
    created_at DATETIME NOT NULL DEFAULT NOW(),
    modified_at DATETIME NULL,
    deleted_at DATETIME NULL,

    CONSTRAINT item_id_char_fk FOREIGN KEY(item_id) REFERENCES items(id),
    CONSTRAINT char_id_char_item_fk FOREIGN KEY(char_id) REFERENCES characters(id)
);
```



4.2. Aplikacja MyBatis.

- Model Account:

```
@EqualsAndHashCode(callSuper = true)
@Data
public class Account extends Auditable {
    private Long id;
    private String email;
    private String nickname;
    private String login;
    private String passwordHash;
    private Date banExpiredAt;
}
```

- Plik XML z mapowaniem:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-
3-mapper.dtd">
<mapper namespace="pl.psk.to.mmo.mmo_mybatis.mapper.AccountMapper">
    <sql id="allFields">
        SELECT
            id,
            email,
            nickname,
            login,
            password_hash as passwordHash,
            ban expired at as banExpiredAt,
            created at as createdAt,
            modified at as modifiedAt,
            deleted_at as deletedAt
    </sql>
    <select id="getAllAccounts" parameterType="pl.psk.to.mmo.mmo_mybatis.model.base.Criteria"
resultType="pl.psk.to.mmo.mmo mybatis.model.Account">
        <include refid="allFields"/>
        FROM accounts
        <if test="criteria.sortColumn != null">
            ORDER BY ${criteria.sortColumn} ${criteria.sortType}
        </if>
        <if test="criteria.limit != null">
            LIMIT ${criteria.offset},${criteria.limit}
        </if>
    </select>
    <select id="getAccountById" parameterType="java.lang.Long"
resultType="pl.psk.to.mmo.mmo mybatis.model.Account">
        <include refid="allFields"/>
        FROM accountd
        WHERE id = ${id}
    </select>
    <insert
            id="insertAccount"
            parameterType="pl.psk.to.mmo.mmo_mybatis.model.Account"
            flushCache="true"
            timeout="20">
        INSERT INTO accounts(email,nickname,login,password hash,ban expired at)
        VALUES
(#{account.email},#{account.nickname},#{account.login},#{account.passwordHash},#{account.banEx
piredAt})

    </insert>
    <update id="updateAccount" parameterType="pl.psk.to.mmo.mmo mybatis.model.Account">
        UPDATE accounts
        SET email = #{account.email},
        nickname = #{account.nickname},
        login=#{account.login},
        password_hash = #{account.passwordHash},
        ban expired at=#{account.banExpiredAt}

        <if test="account.id != null">
            WHERE    id = ${account.id}
        </if>


    </update>
    <delete id="deleteAccount" parameterType="java.lang.Long">
```

```
        DELETE FROM accounts WHERE id = ${id}
    </delete>
</mapper>
```

- Maper AccountMapper:

```
@Mapper
public interface AccountMapper {
    @Options(useGeneratedKeys = true, keyProperty = "id")
    Boolean insertAccount(@Param("account") Account account);

    Boolean updateAccount(@Param("account") Account account);

    Account getAccountById(@Param("id") Long id);

    List<Account> getAllAccounts(@Param("criteria")Criteria criteria);

    Boolean deleteAccount(@Param("id") Long id);
}
```

- Kontroler AccountController:

```
@RestController
@AllArgsConstructor
public class AccountController {

    private final AccountMapper accountMapper;

    @PostMapping("/accounts/all")
    public ResponseEntity<List<Account>> getAllAccounts(@RequestBody Criteria criteria) {
        return ResponseEntity.ok(accountMapper.getAllAccounts(criteria));
    }
    @PostMapping("/accounts/add")
    public ResponseEntity<Boolean> insertAccount(@RequestBody Account account) {
        return ResponseEntity.ok(accountMapper.insertAccount(account));
    }
    @PostMapping("/accounts/update")
    public ResponseEntity<Boolean> updateAccount(@RequestBody Account account) {
        return ResponseEntity.ok(accountMapper.updateAccount(account));
    }
    @PostMapping("/accounts/delete")
    public ResponseEntity<Boolean> deleteAccount(@RequestBody Long id) {
        return ResponseEntity.ok(accountMapper.deleteAccount(id));
    }
}
```

4.3. Aplikacja Hibernate.

- Model Account:

```
@EqualsAndHashCode(callSuper = true)
@Data
@Entity
@Table(name = "accounts")
public class Account extends Auditable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String email;
    private String nickname;
    private String login;
    private String passwordHash;
    private Date banExpiredAt;
}
```

- Klasa do obsługi paginacji

```
@Data
public class AccountPage {
    private int pageNumber = 0;
    private int pageSize = 10;
    private Sort.Direction sortDirection = Sort.Direction.ASC;
    private String sortBy = "id";
}
```

- Repozytorium AccountRepository:

```
@Repository
public interface AccountRepository extends PagingAndSortingRepository<Account,Long> {
}
```

- Usługa AccountService:

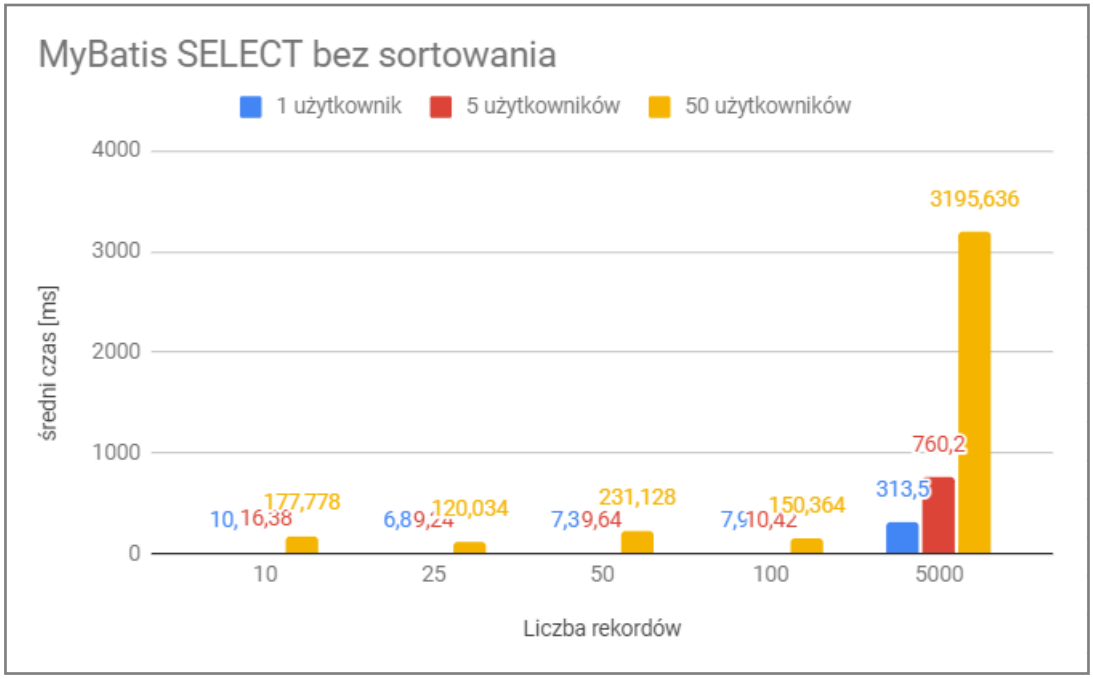
```
@Service
public class AccountService {
    private final AccountRepository accountRepository;

    public AccountService(AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

    public Page<Account> getAccounts(AccountPage accountPage){
        Sort sort = Sort.by(accountPage.getSortDirection(),accountPage.getSortBy());
        Pageable pageable =
PageRequest.of(accountPage.getPageNumber(),accountPage.getPageSize(),sort);

        return accountRepository.findAll(pageable);
    }

    public Account addAccount(Account account){
        return accountRepository.save(account);
    }

    public Account updateAccount(Account account){
        Optional<Account> optionalAccount = accountRepository.findById(account.getId());
        Account result=null;
        if(optionalAccount.isPresent()){
            Account foundAccount = optionalAccount.get();
            foundAccount.setEmail(account.getEmail());
            foundAccount.setNickname(account.getNickname());
            foundAccount.setLogin(account.getLogin());
            foundAccount.setPasswordHash(account.getPasswordHash());
            foundAccount.setBanExpiredAt(account.getBanExpiredAt());
            result = accountRepository.save(foundAccount);
        }
        return result;
    }
    public Boolean deleteAccount(Long id){
        accountRepository.deleteById(id);
        return Boolean.TRUE;
    }
}
```

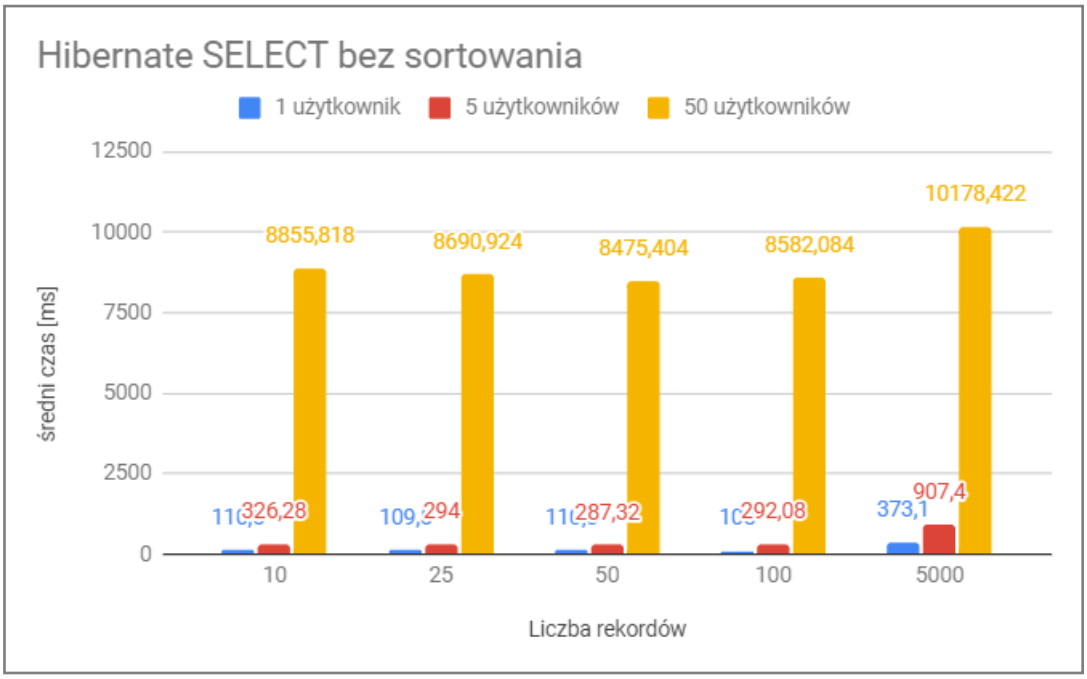
- Kontroler AccountController:

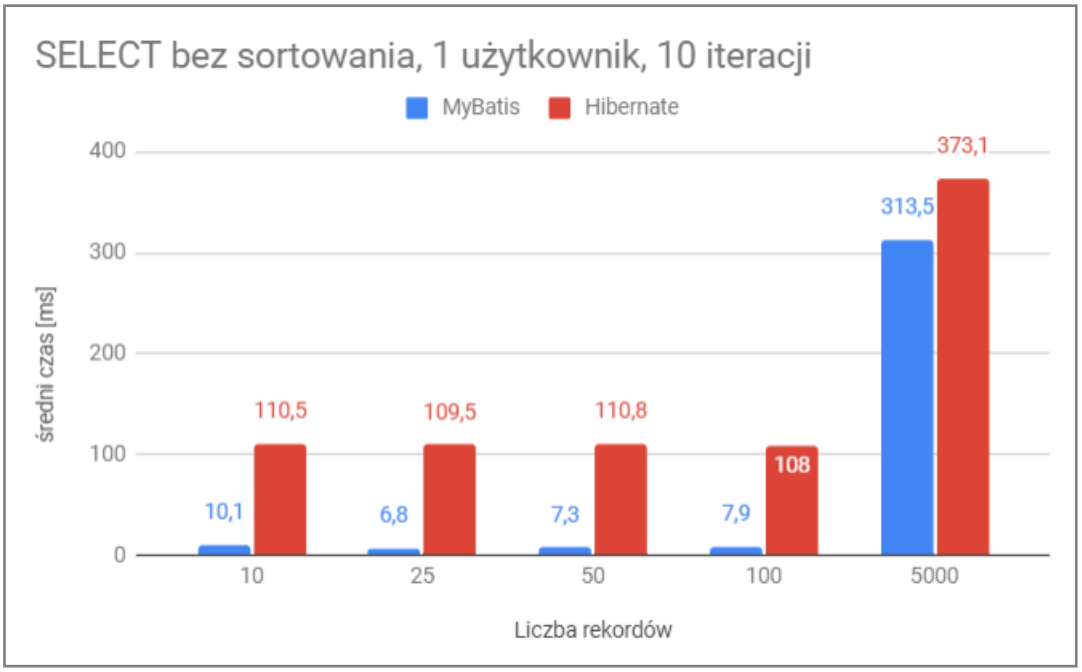
```
@RestController
@RequestMapping("/accounts")
public class AccountController {
    private final AccountService accountService;

    public AccountController(AccountService accountService) {
        this.accountService = accountService;
    }

    @GetMapping
    public ResponseEntity<Page<Account>> getAccounts(AccountPage accountPage){
        return new ResponseEntity<>(accountService.getAccounts(accountPage), HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<Account> addAccount(@RequestBody Account account){
        return new ResponseEntity<>(accountService.addAccount(account),HttpStatus.OK);
    }

    @PostMapping("/update")
    public ResponseEntity<Account> updateAccount(@RequestBody Account account){
        return new ResponseEntity<>(accountService.updateAccount(account),HttpStatus.OK);
    }

    @PostMapping("/delete")
    public ResponseEntity<Boolean> updateAccount(@RequestBody Long id){
        return new ResponseEntity<>(accountService.deleteAccount(id),HttpStatus.OK);
    }
}
```

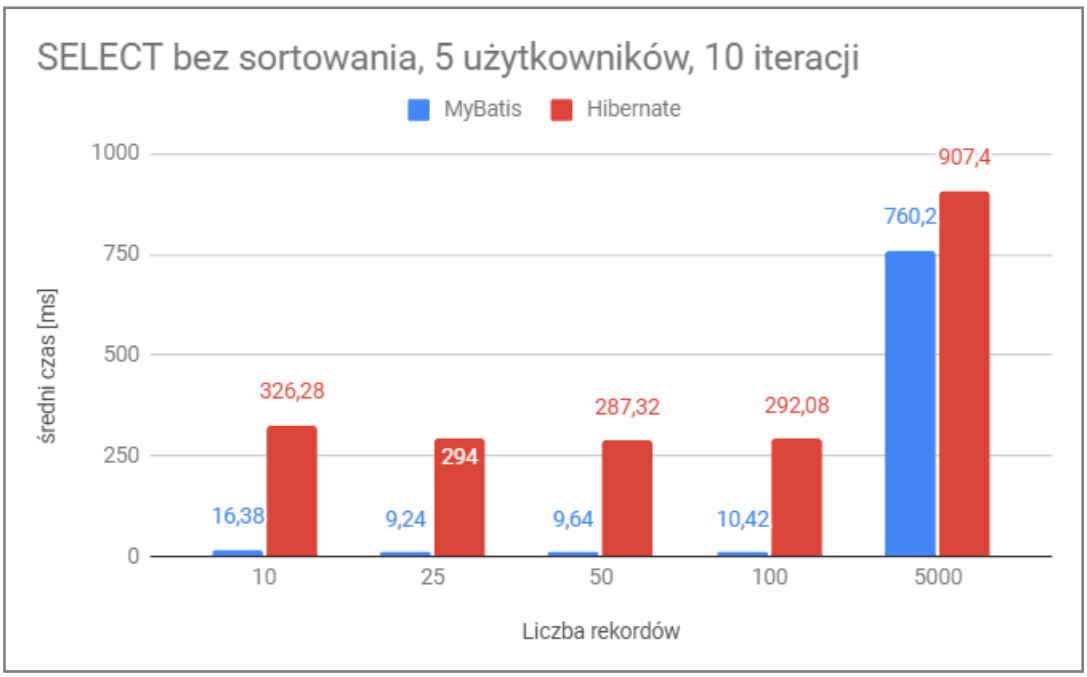
5. Testy wydajnościowe.

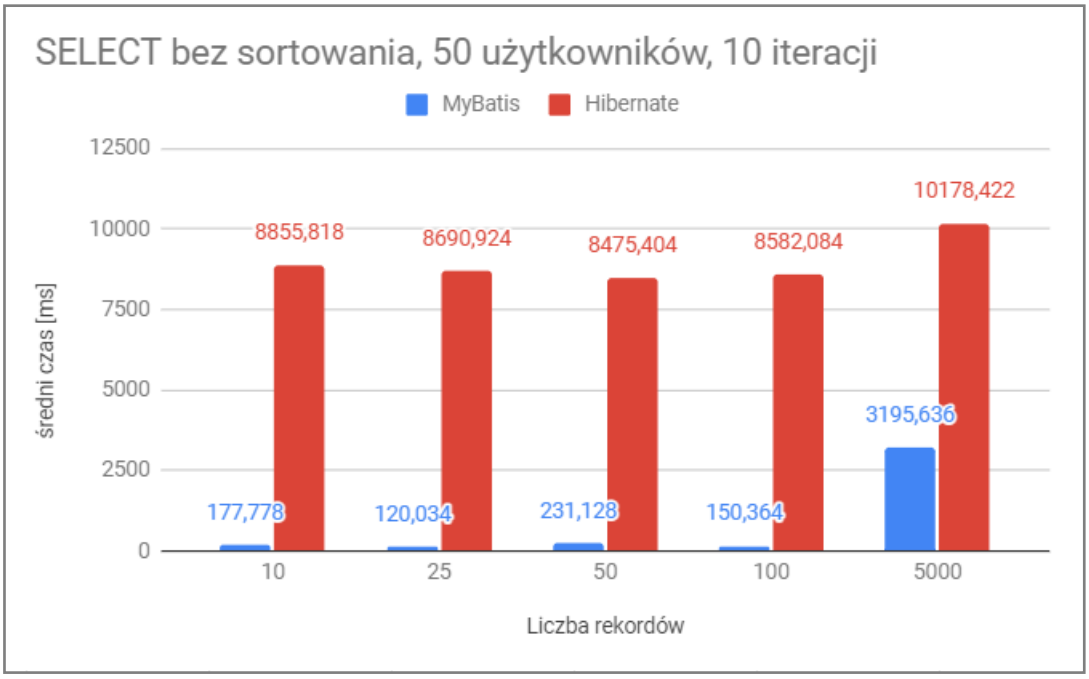
5.1. SELECT.

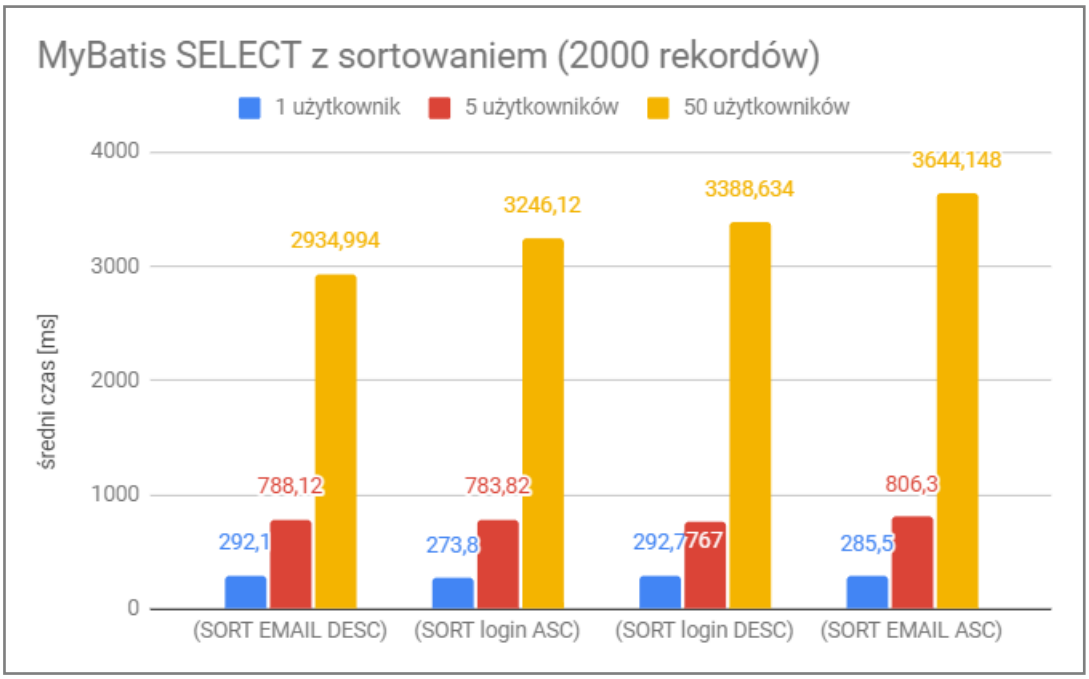


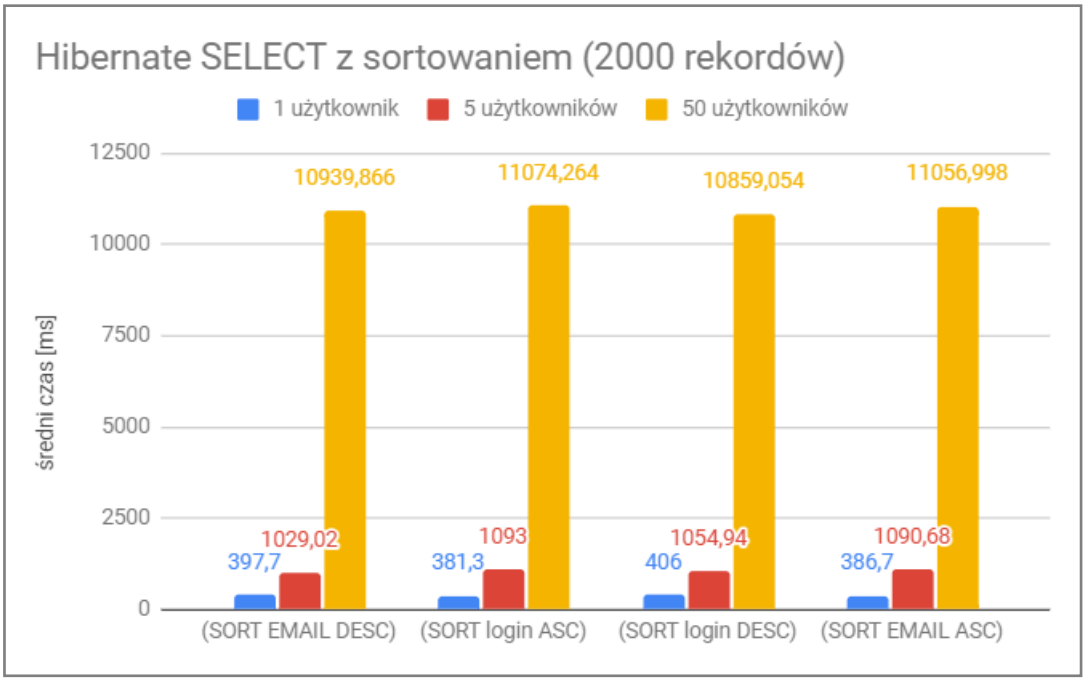


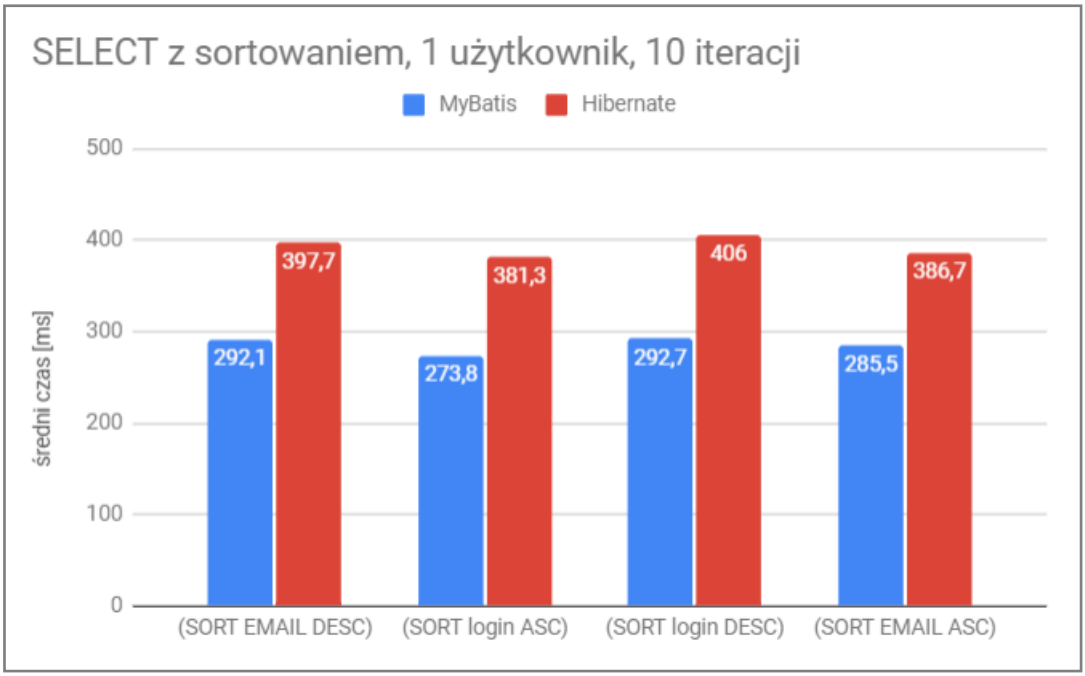

SELECT bez sortowania, 1 użytkownik, 10 iteracji
MyBatis
Hibernate
400
300
200
100
0
średni czas [ms]
110,5
109,5
110,8
108
373,1
313,5
10,1
6,8
7,3
7,9
10
25
50
100
5000
Liczba rekordów

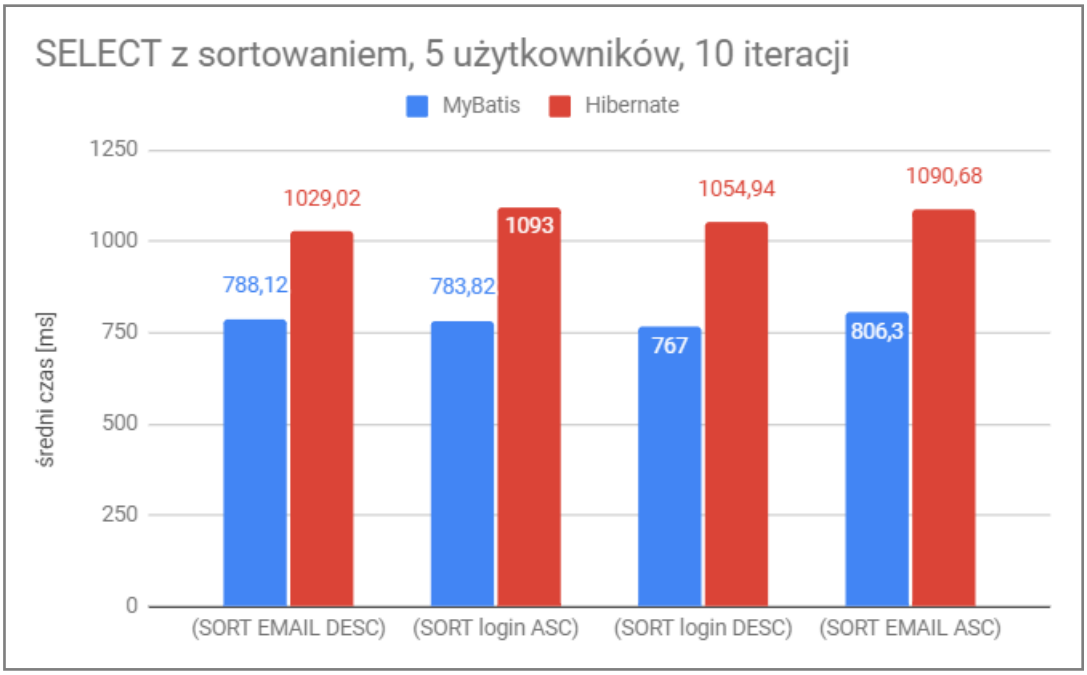

SELECT bez sortowania, 5 użytkowników, 10 iteracji
MyBatis
Hibernate
1000
750
500
250
0
średni czas [ms]
326,28
294
287,32
292,08
907,4
760,2
16,38
9,24
9,64
10,42
10
25
50
100
5000
Liczba rekordów

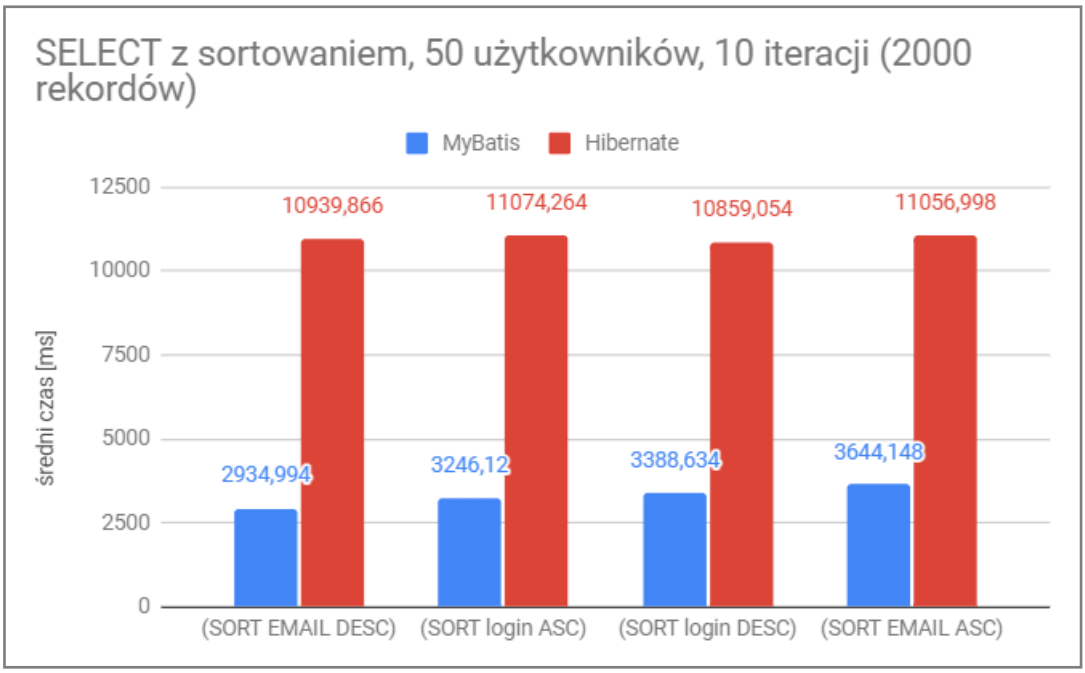

SELECT bez sortowania, 50 użytkowników, 10 iteracji
MyBatis
Hibernate
12500
10000
7500
5000
2500
0
średni czas [ms]
8855,818
8690,924
8475,404
8582,084
10178,422
3195,636
177,778
120,034
231,128
150,364
10
25
50
100
5000
Liczba rekordów


MyBatis SELECT z sortowaniem (2000 rekordów)
1 użytkownik
5 użytkowników
50 użytkowników
4000
3000
2000
1000
0
średni czas [ms]
2934,994
3246,12
3388,634
3644,148
788,12
783,82
806,3
292,1
273,8
292,7767
285,5
(SORT EMAIL DESC)
(SORT login ASC)
(SORT login DESC)
(SORT EMAIL ASC)

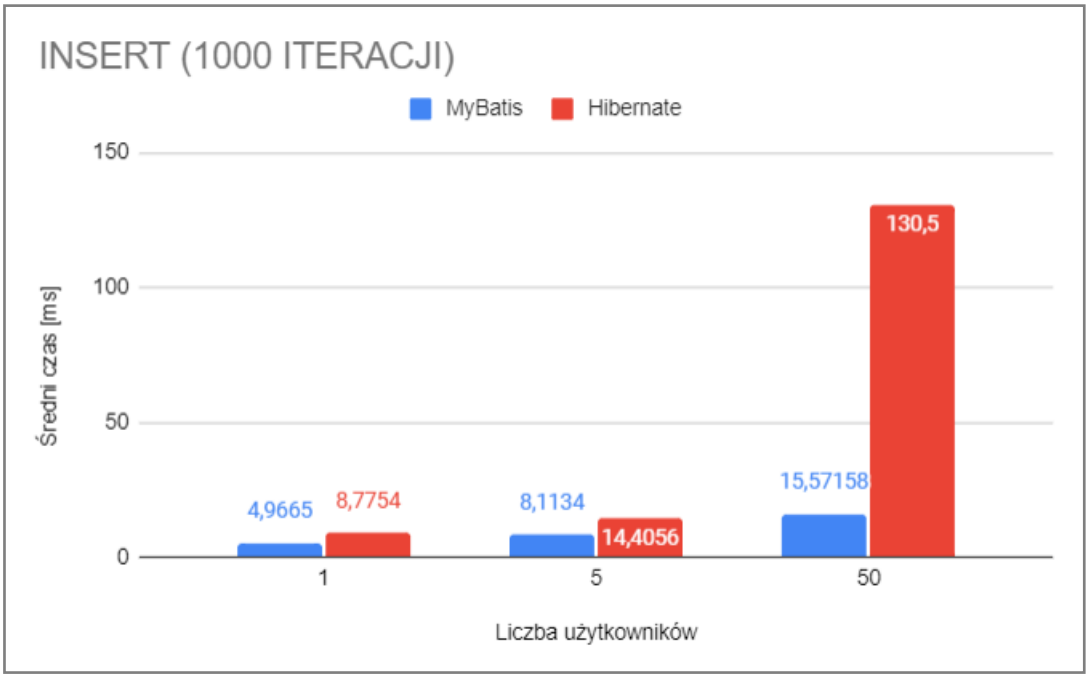

Hibernate SELECT z sortowaniem (2000 rekordów)
1 użytkownik
5 użytkowników
50 użytkowników
średni czas [ms]
12500
10000
7500
5000
2500
0
10939,866
11074,264
10859,054
11056,998
1029,02
1093
1054,94
1090,68
397,7
381,3
406
386,7
(SORT EMAIL DESC)
(SORT login ASC)
(SORT login DESC)
(SORT EMAIL ASC)


SELECT z sortowaniem, 1 użytkownik, 10 iteracji
MyBatis
Hibernate
średni czas [ms]
500
400
300
200
100
0
397,7
381,3
406
386,7
292,1
273,8
292,7
285,5
(SORT EMAIL DESC)
(SORT login ASC)
(SORT login DESC)
(SORT EMAIL ASC)

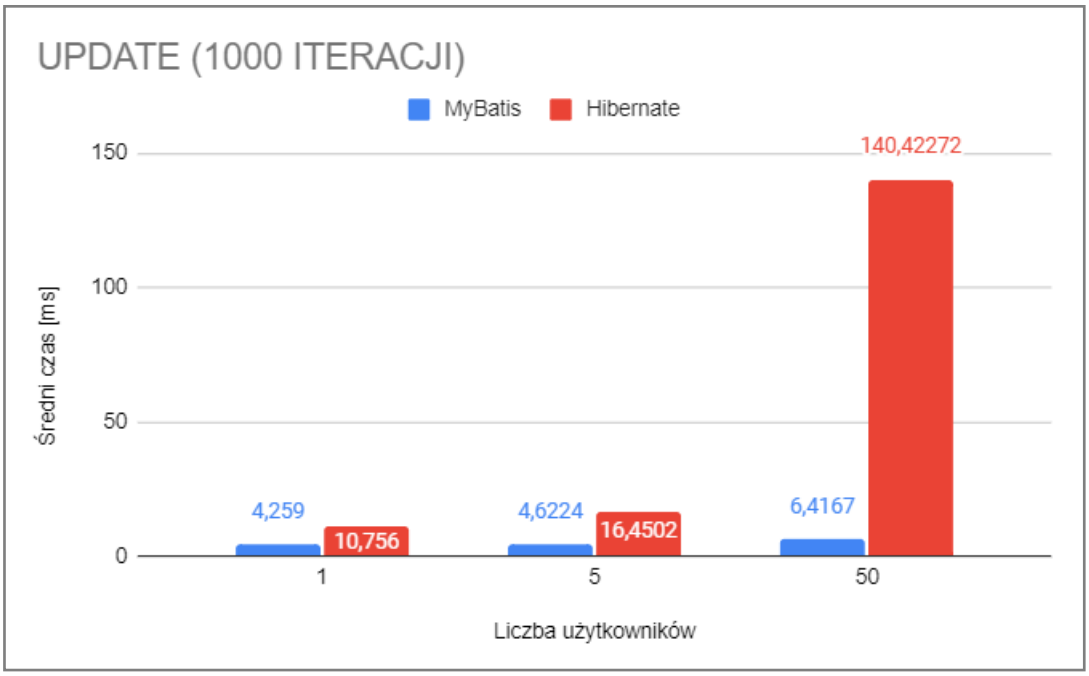

SELECT z sortowaniem, 5 użytkowników, 10 iteracji
MyBatis
Hibernate
średni czas [ms]
1250
1000
750
500
250
0
788,12
1029,02
783,82
1093
767
1054,94
806,3
1090,68
(SORT EMAIL DESC)
(SORT login ASC)
(SORT login DESC)
(SORT EMAIL ASC)


SELECT z sortowaniem, 50 użytkowników, 10 iteracji (2000 rekordów)
MyBatis
Hibernate
średni czas [ms]
12500
10000
7500
5000
2500
0
2934,994
10939,866
3246,12
11074,264
3388,634
10859,054
3644,148
11056,998
(SORT EMAIL DESC)
(SORT login ASC)
(SORT login DESC)
(SORT EMAIL ASC)

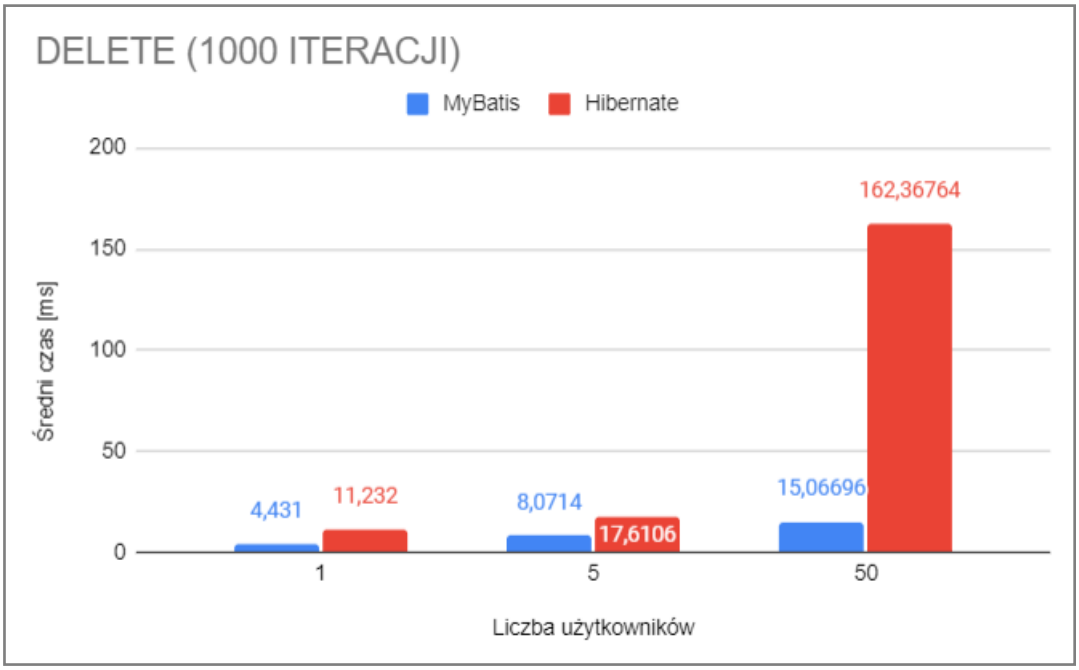
5.2. INSERT.



5.3. UPDATE.



5.4. DELETE.


DELETE (1000 ITERACJI)
MyBatis
Hibernate
200
150
100
50
0
Średni czas [ms]
4,431
11,232
8,0714
17,6106
15,06696
162,36764
1
5
50
Liczba użytkowników

6. Wnioski.

Powyższe porównanie jasno określa różnice między danymi rozwiązaniami. Wybór zależy od konkretnej sytuacji i preferencji użytkownika. MyBatis jest skoncentrowany na danych i jest używany w przypadku, gdy chcemy stworzyć i utrzymywać własną bazę danych SQL. Hibernate jest używany w przypadku gdy użytkownik chce się skupić jedynie na warstwie biznesowej.

Różnica szybkości rozwiązań jest coraz bardziej zauważalna wraz z wzrostem liczby klientów. Czas realizowania zapytań przez MyBatis, przy dziesięciokrotnym zwiększeniu liczby klientów, wzrastał średnio o 150%, podczas gdy czas odpowiedzi Hibernate wzrastał o 900%.