|  |  |
| --- | --- |
| **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\_expired\_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

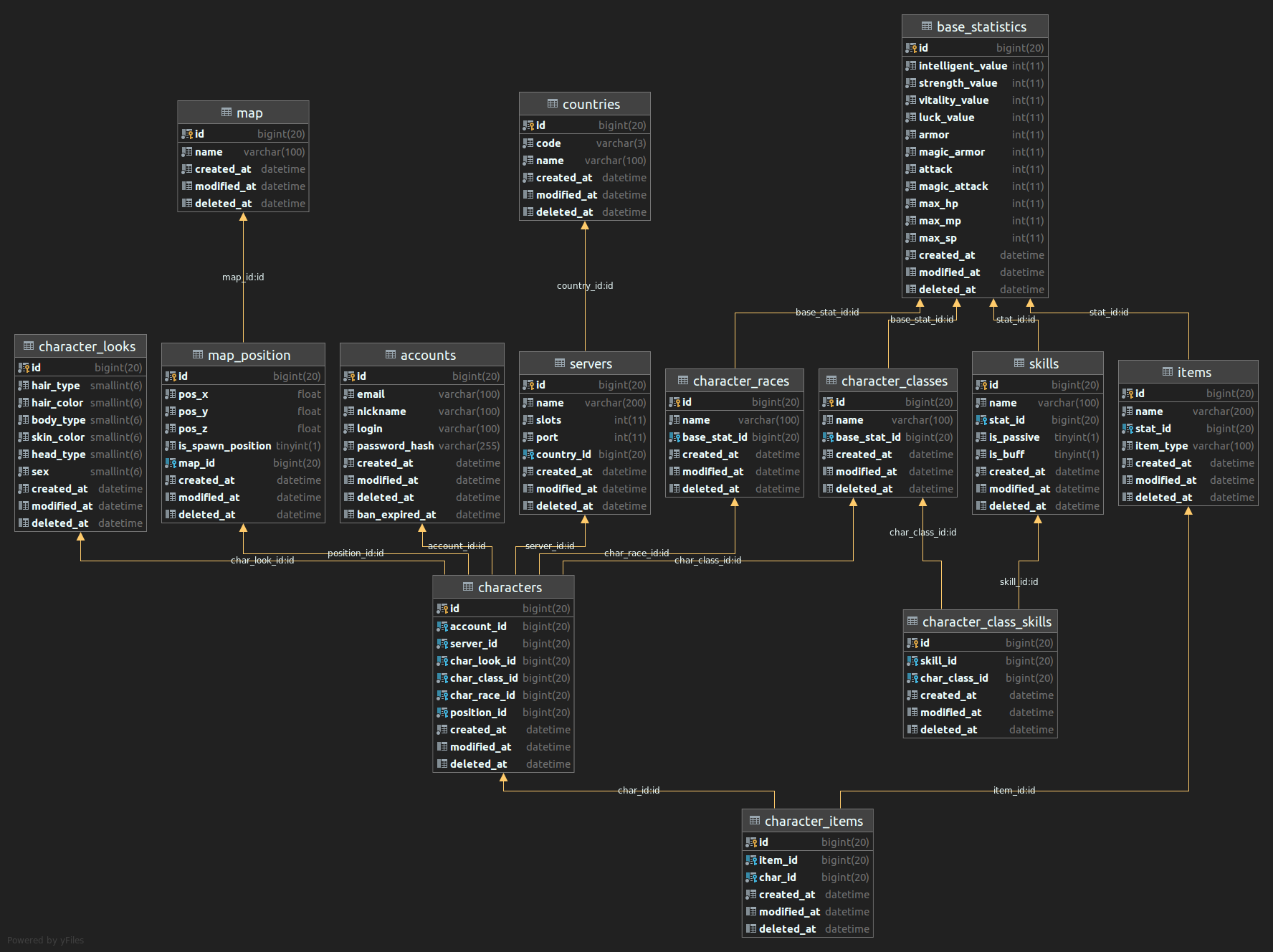
|  |
| --- |
| <?xml version = "1.0" encoding = "utf-8"?> <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name = "Employee" table = "EMPLOYEE">   <meta attribute = "class-description">  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.

|  |
| --- |
| 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.banExpiredAt})   </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:

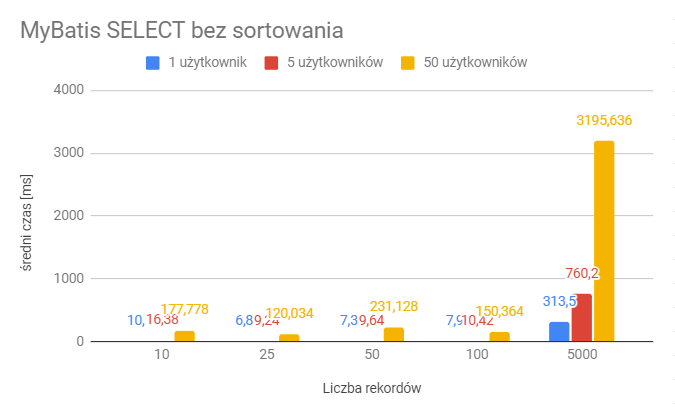
|  |
| --- |
| @Service public class AccountService {  private final AccountRepository accountRepository;   public AccountService(AccountRepository accountRepository) {  this.accountRepository = accountRepository;  }   public Page<Account> getAccounts(AccountPage accountPage){  Sort sort = Sort.*by*(accountPage.getSortDirection(),accountPage.getSortBy());  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*;  } } |

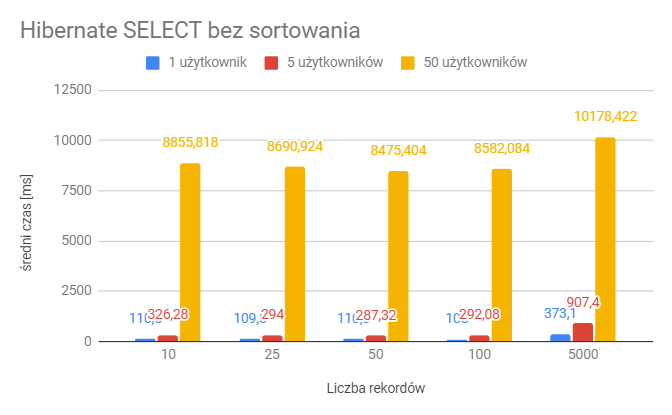
* Kontroler AccountController:

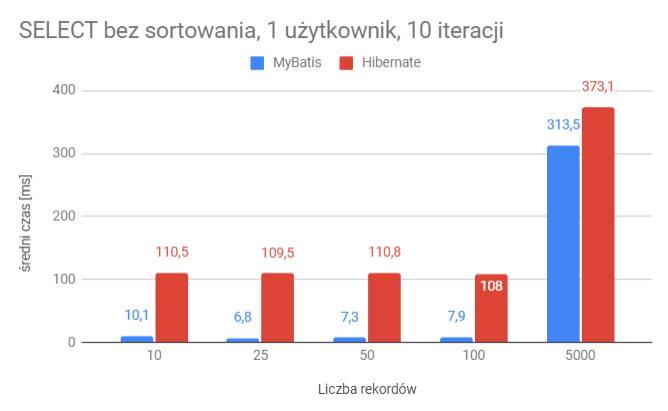
|  |
| --- |
| @RestController @RequestMapping("/accounts") public class AccountController {  private final AccountService accountService;   public AccountController(AccountService accountService) {  this.accountService = accountService;  }   @GetMapping  public ResponseEntity<Page<Account>> getAccounts(AccountPage 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*);  } } |

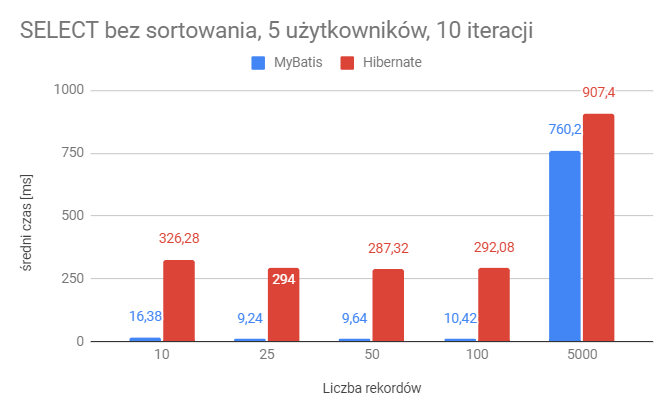
# 5. Testy wydajnościowe.

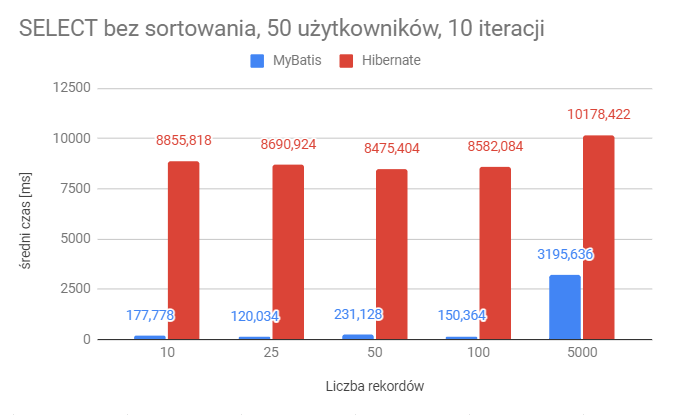
## 5.1. SELECT.

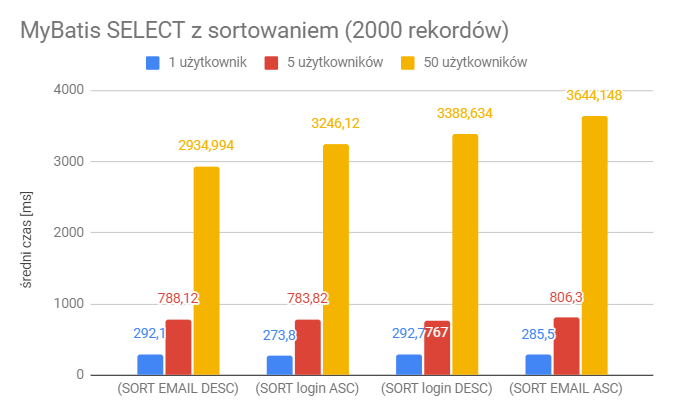


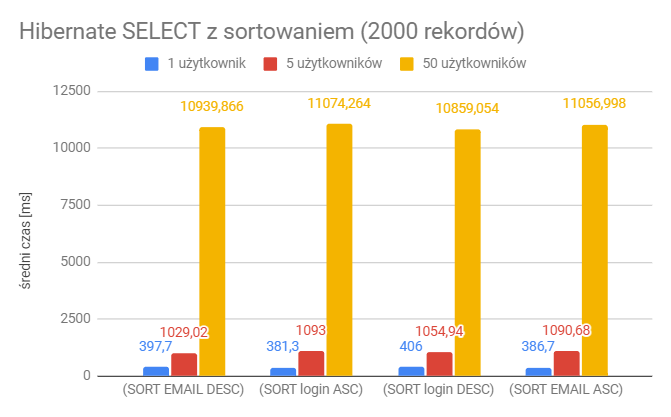


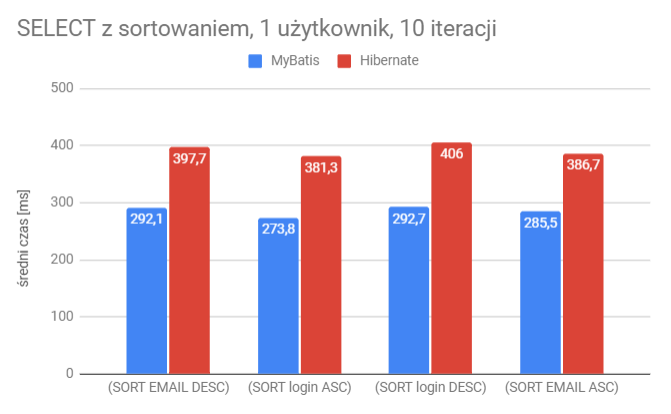


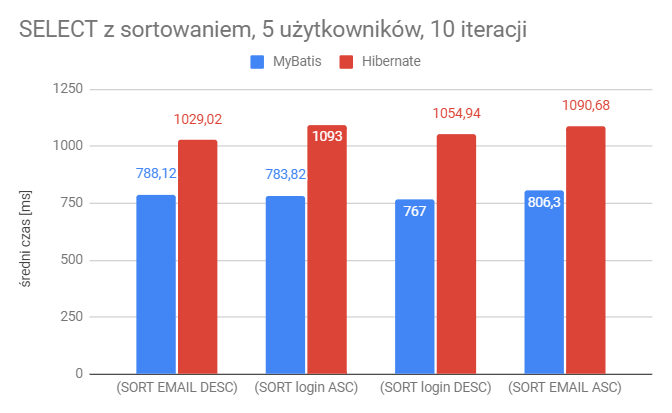


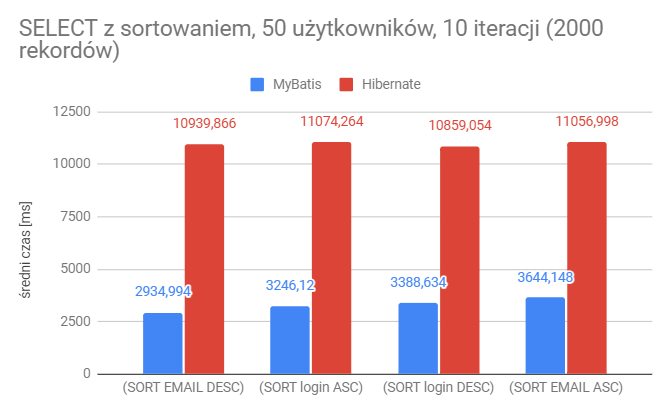




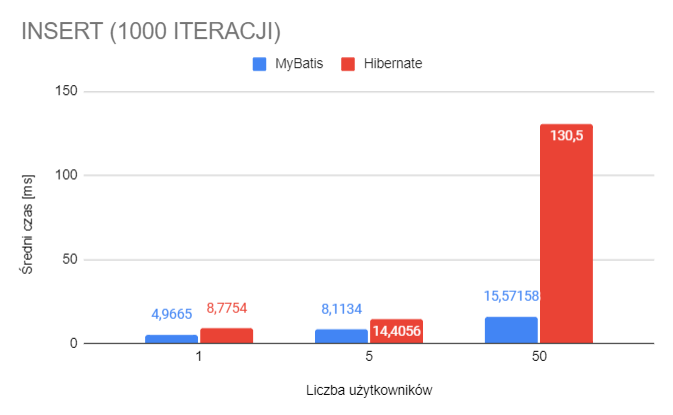




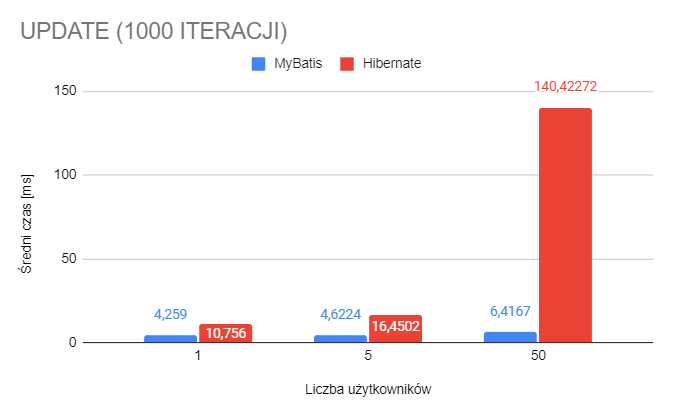




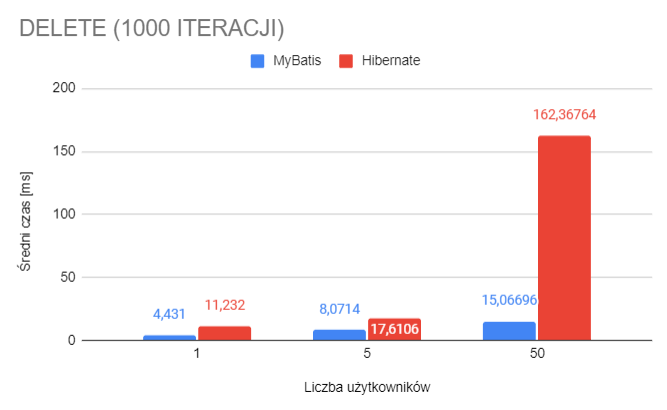
## 5.2. INSERT.



## 5.3. UPDATE.



## 5.4. DELETE.



# 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%.