Porównanie 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 does not have its own log statistics so one has to log with log4j. MyBatis nie posiada własnego systemu raportów, konieczne jest użycie log4j.	Hibernate has its own log statistics. 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.

https://www.educba.com/mybatis-vs-hibernate/

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.

Przykład w MyBatis:

Tworzymy interfejs mappera:

```
@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")
    List<Account> getAllAccounts();
    @Delete("DELETE FROM Accounts WHERE id = ${id}")
    void deleteAccount(@Param("id") Long id);
}
```

Adnotacja @Mapper daje nam tu informacje, że jest to mapper z MyBatis. Adnotacja @Param daje informacje w przypadku rapperów xml jak będzie nazywał się nasz parametr. Definicje metod możemy wykonać poprzez odpowiednie adnotacje i wpisanie tam kodu SQL bądź utworzenie mappera który wygląda analogicznie.

```
<?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.passw
ordHash},#{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>
```