Сравнительный анализ моделей работы с данными в Java-приложениях

В.А. Дашук, Д.Е. Намиот

***Аннотация* — В этой статье рассматриваются современные подходы работы с базами данных в Java проектах. Рассмотрены преимущества и недостатки при работе с помощью низкоуровневой технологии JDBC. Представлен обзор спецификации JPA, являющейся стандартом отображения POJO объектов в реляционные базы данных. В работе дан обзор наиболее популярных JPA-реализаций: Hibernate, EclipseLink и OpenJPA. Также отмечены возможности ORM системы MyBatis, которая не реализует JPA, но является их альтернативой. В ходе работы предложен тестовый набор, сравнивающий производительность JPA-реализаций. Дано сравнение MyBatis и одной из JPA-реализаций на базе Hibernate. Представлены результаты и выводы проведенных тестов, дающие понимание, какие решения лучше подходят для разных типов ситуаций.**

***Ключевые слова* — JDBC, MyBatis, JPA, Hibernate, EclipseLink, OpenJPA.**

I. Введение

В основание этой статьи положена выпускная квалификационная работа, выполненная одним из авторов на факультете ВМК МГУ имени М.В. Ломоносова.

При проектировании любого корпоративного приложения большое значение приобретает вопрос организации его доступа к данным. Java имеет широкий спектр библиотек, фреймворков, IDE, инструментов и поставщиков серверов, решающих задачи обмена данными, управления транзакциями и многое другое, что сделало данный язык популярным в корпоративной среде.

Организовать взаимодействие с базой данных в программе на Java можно различными способами. Наиболее простой и прямой — использовать API JDBC (Java Database Connectivity), в данном случае код будет содержать встроенные запросы. Но в случае изменений схемы базы данных разработчикам придется адаптировать исходный код приложений, чтобы получить доступ к измененным элементам. Часто данный процесс адаптации осуществляется вручную.

Статья получена 1 июня 2020.
В.А. Дашук – МГУ имени М.В. Ломоносова (email 4studywork@gmail.com).
Д.Е. Намиот - МГУ имени М.В. Ломоносова (email dnamiot@gmail.com).

Более сложным, но зато более гибким решением будет воспользоваться одной из библиотек ORM (Object-Relational Mapping), например, Hibernate, EclipseLink, OpenJPA для преобразования концепций программы (классов, методов и атрибутов и т.д.) в концепции базы данных (таблицы, столбцы, строки и т.д.). ORM обеспечивают высокий уровень абстракции реляционной базы данных, что позволяет использовать язык программирования, вместо использования операторов SQL и хранимых процедур. Как следствие, взаимодействие между прикладной программой и базой данных может стать более динамичным, но и более сложным для понимания.

Помимо основной функциональности каждый фреймворк имеет различия, начиная от конфигурации и заканчивая методами оптимизации. Понимание того, как технологии работы с базами данных стремятся заменить или дополнить существующие в программных проектах, может помочь руководителям проектов в выборе наиболее подходящей технологии.

Целью настоящей работы является проведение сравнительного анализа моделей работы с данными в Java-приложениях.

В задачи настоящей работы входит: изучение современных технологий работы с базами данных в Java-приложениях; определение преимуществ и недостатков при работе с помощью низкоуровневых технологий JDBC, и высокоуровневых, на примере, ORM-технологий; разработка тестов для проведения сравнительного анализа производительности JPA-реализаций; анализ результатов тестирования и представление рекомендаций по использованию современных ORM-технологий.

Результатом работы является разработанное приложение с тестовым набором для сравнения различных подходов работы с данными и проведение измерений.

Оставшаяся часть статьи структурирована следующим образом. В разделе II рассмотрены преимущества и недостатки использования низкоуровневой технологии работы с базой данных JDBC. В разделе III дан обзор ORM MyBatis, интерес к которому являются в том, что он не реализует JPA в отличие от других ORM технологий, рассмотренных в данной статье. Раздел IV дает представление о спецификации JPA, ее основных концепций. В разделе V дан обзор JPA-реализаций на примере Hibernate, EclipseLink и OpenJPA. VI раздел посвящен сравнительному анализу производительности ORM фреймворков. В заключительном VII разделе сравниваются два ORM фреймворка MyBatis и Hibernate.

II. JDBC - Java Database Connectivity

JDBC — это API-интерфейс, состоящий из набора классов и интерфейсов, предоставляющий прямой и простой способ организации взаимодействия с базой данных в программе Java. Для такого доступа необходимо наличие jdbc драйвера для конкретной СУБД, реализующей этот API. Каждый производитель пишет jdbc - драйвер под свою базу данных, и не зависимо от реализации, каждый драйвер за счет стандартизации содержит одни и те же методы. Что позволяет в случае смены базы данных поменять jdbc - драйвер и, внеся небольшие корректировки в код, получить работоспособное приложение, не меняя его бизнес-логику.

JDBC — это низкоуровневая технология, работа с которой представляет собой отправку запросов SQL непосредственно из исходного кода; кроме того, она предоставляет разработчикам Java доступ к данным таблицы базы данных из приложений Java. В рамках JDBC данные представлены в той же структуре, что и в реляционных СУБД, - как записи, состоящие из полей и размещенные в таблицах.

Пример сохранения одной сущности в базу данных с помощью JDBC:

```
public void savePerson(Person p) throws
SQLException {
  String query = "INSERT INTO PERSON VALUES
  (DEFAULT, ?, ?)";
  PreparedStatement prepStat =
    connection.prepareStatement(query);
  prepStat.setString(1, p.getName());
  prepStat.setString(2, p.getSurname());
  prepStat.executeUpdate();
}
```

API JDBC предоставляет стандартный набор классов и интерфейсов Java с соответствующими методами для выполнения операторов SQL: class.forName, для загрузки драйвера JDBC; DriverManager.getConnection, для открытия соединения с базами данных, PrepareStatement для выполнения запросов.

Долгое время JDBC являлась наиболее широко используемой технологией хранения данных на платформах Java. Тем не менее, доступ к базе данных с помощью JDBC API создает много накладных расходов; управление ресурсами базы данных и обработка исключений возлагается на разработчика; для каждого запроса необходимо создавать новое подключение, что приводит к многократному повторению кода. Кроме того, объектно-реляционное сопоставление данных становится нетривиальной задачей, так как разработчик должен создать классы для сопоставления таблиц в базе данных с соответствующими возможными ошибками или упущениями и ручной проверкой.

Но, несмотря на недостатки использования JDBC, некоторые Enterprise-приложения по-прежнему используют данную технологию, так как она дает возможность работать с данными на низком уровне, позволяя, обрабатывать сложные запросы, которые невозможно выразить с помощью фреймворков более высокого уровня.

III. MyBatis

MyBatis (ранее iBatis) — это постоянная среда Java с открытым исходным кодом, предоставляющая простой в использовании API для взаимодействия с базой данных[36]. Данный фреймворк освобождает разработчика от необходимости писать шаблонный и ненужный код JDBC.

MyBatis является удобным маппером с поддержкой пользовательских SQL-запросов, хранимых процедур и расширенных отображений. Он не умеет создавать схему и в целом ничего о схеме не знает. MyBatis превращает вызов метода в запрос к базе, и как результат вызова метода, возвращает результат запроса, что позволяет избежать кодирования JDBC и ручной настройки параметров для получения результатов. Он может использовать простой XML или аннотации для конфигурации и отображения примитивов, интерфейсов Map и Java POJO в записи базы данных [38].

MyBatis удобен для случаев, когда в приложении имеется достаточно много различных сложных выборок из нескольких таблиц; в случае сложных запросов к базе на агрегацию данных; при работе с хранимыми процедурами; и в случае, если возникает необходимость написать свой ORM. MyBatis позволяет отображать SQL-запросы на сущности. Поэтому производительность запроса полностью определяется разработчиком. Это позволяет полностью контролировать, какие запросы и как именно будут отправляться в БД.

Для базовой конфигурации необходимо указать источник данных с именем драйвера и url к БД, логином и паролем. Также в конфигурационном файле указываются type alias (объекты) и mapper (xml-файлы, которые хранят SQL-запросы).

Представим пример отображения с помощью MyBatis:

```
public interface PersonDao{
     void insert(Person person);
     List<Person> selectAll ();
     void update(String firstName);
}
```

Дан интерфейс PersonDao с методами вставки, выборки всех сущностей и обновлением:

```
<mapper namespace="ru.mybatis.dao.PersonDao" >

   <insert id="insert">
    insert into Person(firstname, middlename,
    lastname)
    values(#{firstName}, #{middleName},#{lastName}
   </insert>

   <select id=" selectAll" resultType="Person">
    select * from Person
   </select>

   <update id="update">
     update person set firstName=#{firstName};
   </update>
</mapper>
```

Для отображения используется XML, в котором указан список запросов на чистом SQL.

Аналогичный пример с использованием аннотаций:

```
public interface PersonDao{
 @Insert(insert into Person(firstname, middlename,
 lastname)
 values(#{firstName},#{middleName},#{lastName})
 void insert(Person person);

 @Select(select * from Person)
 List<Person> selectAll ();

 @Update(update person set firstName=#{firstName})
 void update(String firstName);
```

```
}
```

Пример использования хранимых процедур

```
<select id="callongProcedure",
parameterType="Criteria" statemenType="CALLABLE">
 {call long_procedure(#{id, jdbcType=INTEGER,
mode=IN},
#{firstParameter, jdbcType=VARCHAR, mode=IN},
#{secondParameter, jdbcType=, mode=IN}
)}
</select>
```

Как видно из примера выше MyBatis не упраздняет наличие SQL-кода в проекте, однако он делает его написание и сопровождение гораздо легче.

Несмотря на все преимущества и интересные идеи, это не полноценное persistence-решение, поскольку в нем отсутствуют функции, требуемые от универсальной инфраструктуры, такие как переносимость между различными системами баз данных.

IV. JAVA PERSISTENCE API (JPA)

Сегодня Java-разработчики взаимодействуют с JDBC через платформы доступа к данным, реализующие Object-Relational Mapping (ORM), такие как: Hibernate, OpenJPA, EclipseLink и т.д. Они освобождают разработчиков от написания Java-кода для управления соединениями к базе данных, запросами, кэшем и объектно-реляционным отображением Java объектов, в результате чего разработчики могут сосредоточится на написании бизнес-логики при разработке программ.

Java Persistence API (JPA) — это спецификация API, разработанная для отображения Plain Old Java Object (POJO) в реляционные базы данных [46]. Реализации JPA выполняют моделирование метаданных и создание схем с помощью аннотаций или с помощью XML-файлов. Аннотации являются более предпочтительным стандартом, чем XML, из-за удобства использования.

JPA предоставляет API для CRUD операций с использованием Java Persistence Query Language (JPQL), который является объектно-ориентированным языком запросов. Также JPA обеспечивает поддержку управления транзакциями с использованием Java Transaction API (JTA) [24].

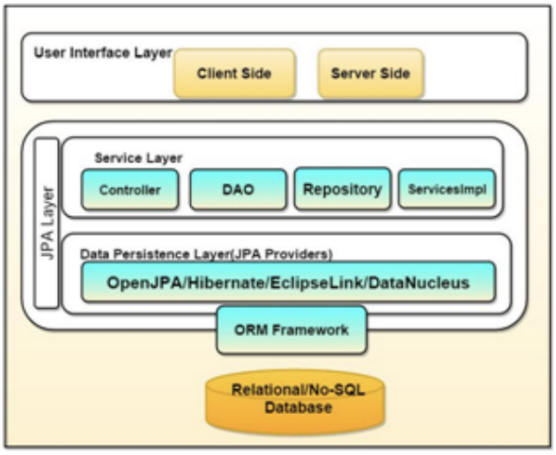

Рис. 1. Архитектура JPA [46].

На Рис. 1 показана трехуровневая архитектура JPA, где верхним уровнем является графический интерфейс пользователя (GUI), который связывается с клиентом/сервером для выполнения операций на внешнем интерфейсе. Средний уровень — это уровень JPA, который подразделяется на уровень службы с использованием контроллеров, DAO, репозиториев и реализации службы, а также persistence уровень, который определяет процедуру отображения между реляционной базой данных и объектами Java. Нижний уровень определяет тип СУБД для сохранения данных в табличном формате.

JPA содержит набор концепций, которые определяют, какие объекты в приложении должны сохраняться и как они должны сохраняться, тем самым позволяя преодолеть разрыв между объектами Java (то есть классами и интерфейсами Java) и реляционными базами данных (то есть таблицами и столбцами).

Рассмотрим основные концепции JPA:

Основные интерфейсы размещены в пакете javax. Persistence и представлены на Рис. 2.

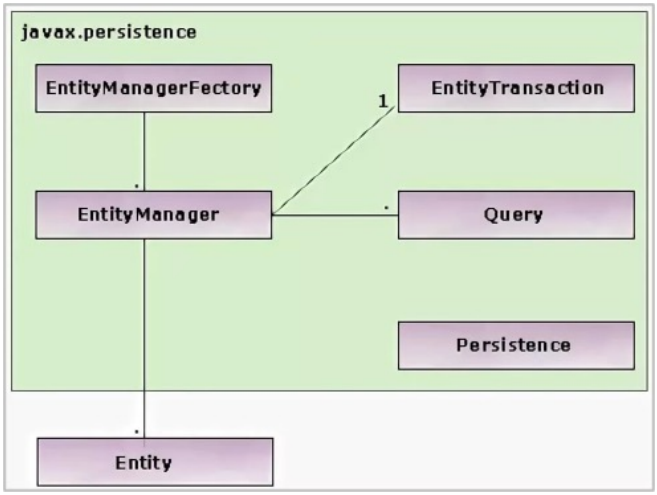

Рис. 2. Основные интерфейсы JPA [13].

Интерфейс Persistence содержит статический метод createEntityManagerFactory(), то есть он создает EntityManagerFactory.

```
entityManagerFactory =
Persistence.createEntityManagerFactory(unitName);
```

EntityManagerFactory на вход принимает строку, которая является именем persistence unit, определенного в META-INF/persistence.xml

Persistence создает EntityManagerFactory на основе конфигурации, здесь также определяется какой провайдер будет использоваться.

Самое главное назначение EntityManagerFactory создавать EntityManager. EntityManager определяет сеанс взаимодействия с базой данной. То есть при выполнении сеанса взаимодействия с базой данных создается Entitymanager, выполняется манипуляция и затем Entitymanager закрывается. Объекты Entitymanager легковесны и занимают мало памяти.

Каждый экземпляр EntityManager связан с экземпляром EntityTransaction, который позволяет управлять транзакциями с помощью методов begin(), commit(), rollback().

```
EntityManager em =
entityManagerFactory.createEntityManager();
em.getTransaction().begin();
//Some actions
em.getTransaction().commit();
```

EntityManager может содержать несколько объектов Query и получать результат запроса.

EntityManager содержит несколько Entity. Entity-классы представляют основу JPA. Entity (Сущность) — POJO-класс, связанный с базой данных с помощью аннотации (@Entity) или через "XML" - файл описания. Такой класс должен удовлетворять следующим требованиям:

- он должен иметь пустой конструктор;
- не может быть final-классом и не может содержать final-полей или свойств;
- не может быть интерфейсом, enum или вложенным классом;
- должен содержать хотя бы одно поле (свойство) аннотированное @Id – идентификатор, в SQL подобный идентификатор называется первичным ключом.

Entity-класс может:

- наследоваться от другого Entity класса или иного другого класса;
- являться наследником других классов;
- иметь непустые конструкторы;
- иметь различные дополнительные методы и свойства.

Для каждого сохраняемого объекта JPA создает новую таблицу в выбранной базе данных.

Другой концепцией JPA является постоянство поля. Когда объект в Java определяется как объект, все поля в нем автоматически сохраняются как разные столбцы в таблице объектов. Поле, которое не нужно сохранять в базе данных помечается аннотацией @Transient.

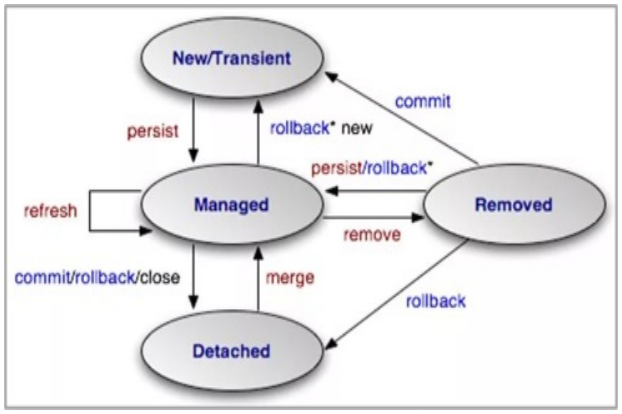

Рис. 3. Жизненный цикл Entity [46].

На Рис. 3 показан жизненный цикл Entity. Рассмотрим подробнее состояния Entity:

1. New: когда будущая сущность (Entity) создается оператором new она пока еще является просто классом Java и JPA-реализация ничего о нем не знает;
2. Managed: после сохранения в базу данных с помощью метода persist(), сущность переходит в состояние Managed и это значит что она уже есть в persitence context (набор entity, управляемый entity manage), в этот момент сущность получает id (первичный ключ, который есть в базе данных) и с ней можно работать, то есть сущность связана с persitence context и имеет связанный с ней идентификатор. Она может как существовать в базе данных, так и отсутствовать в ней. Сущность управляется Entity Manager, в этом состоянии можно изменять значение ее полей, которые будут автоматически сохраняться в БД. Перечитывать данные из базы в случае, когда данные изменились после того, как она была загружена с помощью метода refresh();

Далее два пути Removed или Detached:

3. Removed: здесь можно удалить сущность с помощью метода remove(), после чего она не управляется *entity manager* (интерфейс для доступа к БД), имеет идентификатор и запланирована к удалению из базы данных;
4. Detached: здесь можно закрыть или очистить сущность с помощью методов close()/clear(), она переходит в состояние Detached. Это значит, что есть сущность, есть id, но она больше не управляется entity manager и не связана с persistence context. С помощью метода merge() сущность возвращается в состояние Managed.

Следующая концепция определяет отношения между таблицами базы данных. JPA указывает четыре аннотации отношений, с помощью которых можно управлять отношениями между различными таблицами базы данных в приложении.:

@OneToOne(один-к-одному), @ManyToOne (многие-к-одному), @OneToMany(один-ко-многим), @ManyToMany (многие-к-многим).

Наконец класс EntityManager содержит общие операции создания, чтения, обновления и удаления (CRUD) в базу данных.

JPA предлагает множество преимуществ по сравнению с традиционными JDBC, такие как [11]:

- Более эффективное кэширование: в Java Persistence API реализован доступ к данным через кэш, и разработчику не нужно беспокоится об обновлении кэша или других действий в отличии от JDBC [14];
- Поддерживаемые языки запросов: реализации JPA предоставляют разработчику больше гибкости в построении доступа к данным с помощью собственного языка запросов JPQL, в тоже время поддерживается классический язык структурированных запросов (SQL);
- Низкие затраты на программирование: JPA освобождает разработчика от ручной обработки данных в операциях CRUD, следовательно, сокращается время разработки и затраты на обслуживание;
- Простота сохранения данных;
- Независимость от драйвера JDBC: в JPA такая информация как имя базы данных, сервер, пользователь и пароль и т.д., находится в файле конфигурации ORM (например, hibernateconfig.xml);
- Простота подключения: JPA отлично масштабируется в любой среде, независимо от того, используется ли она во внутренней сети, обслуживающей сотни пользователей, или сотни тысяч пользователей;
- Меньшие издержки памяти для поддержки API во время пиковых нагрузок.

Важно отметит, что JPA является только спецификацией и для работы ему нужна реализация. В настоящее время Hibernate, OpenJPA и EclipseLink считаются наиболее успешными реализациями JPA. Hibernate самая популярная из открытых реализаций последней версии спецификации (JPA 2.1).

V. Обзор JPA-реализаций

A Hibernate

Hibernate — наиболее успешная реализация Java Persistence API, решающая задачу объектно-реляционного отображения (ORM), с огромным сообществом, поддерживающим проект.

Hibernate также автоматически генерирует запросы для сохранения, манипулирования и извлечения данных и значительно сокращает время разработки, затрачиваемое на ручное написание SQL и JDBC кода.

В Hibernate изменения, которые лежат в persitence context могут не сразу попадать в базу данных из-за flush policy. Flush – выполнение накопленных в persitence context запросов в БД. В JPA существует два варианта для настройки момента выполнения flush: commit (во время выполнения commit) или auto (время выполнения Hibernate выбирает сам).

Связи между сущностями могут быть настроены с помощью стандартных аннотаций (или xml) JPA: One-to-one, One-to-many(many-to-one), Many-to-many.

Для загрузки сущностей из базы данных можно использовать: HQL, Native SQL, Criteria API. HQL -язык запросов Hibernate (Hibernate Query Language), который очень похож на родной SQL. В сравнении с SQL, HQL полностью объектно-ориентирован и использует понятия наследования, полиморфизма и связывания.

Управление транзакциями предоставляется разработчикам приложений через интерфейс Transaction.

Для снижения нагрузки на БД и оптимизации в Hibernate реализовано 3 уровня кэшей [20]

- Кэш первого уровня (First-level cache) – не может быть отключен, кэширует сущности в persistence context;
- Кэш второго уровня (Second-level cache) – должен быть явно включен и настроен, кэширует сущности среди нескольких entity manager;
- Кэш запросов (Query cache) – кэширует результаты запросов по запросу и его параметрам;

При загрузке сущности из базы данных или сохранении она сразу попадает в persistence context, повторный вызов будет уже обслужен из кэша и обращение к базе не произойдет. То есть в роли кэша в сессии выступает persistence context.

Основная функция первичного кэша состоит в том, чтобы синхронизировать данные сессии и данные в таблице базы данных [54]. Интересно поведение кэша первого уровня при использовании ленивой загрузки. При загрузке объекта методом load() или объекта с лениво загружаемыми полями, лениво загружаемые данные в кэш не попадут.

При обращении к данным будет выполнен запрос в базу, и данные будут загружены и в объект, и в кэш. А вот следующая попытка лениво загрузить объект приведёт к тому, что объект сразу вернут из кэша и уже полностью загруженным.

Если кэш первого уровня существует на уровне сессии и persistence context, то кэш второго уровня находится на уровне SessionFactory.

В Hibernate жизненный цикл объекта SessionFactory соответствует жизненному циклу приложения, поэтому кэш второго уровня привязан к объекту-фабрике сессий. Так как кэш второго уровня реализуется третьей стороной, такой как, например, EhCache, JCache, следовательно, его можно подключить или удалить, по умолчанию не подключен [54].

Чтение из кэша второго уровня происходит только в том случае, если нужный объект не был найден в кэше первого уровня.

Если он все еще не найден в кэше второго уровня, он будет найден в таблице базы данных. Когда приложение добавляет, удаляет или изменяет данные в таблице базы данных, Hibernate одновременно обновляет данные в кэше, чтобы обеспечить согласованность кэша данных и реальных данных в таблице базы данных.

Hibernate не допускает большинство операций без использования транзакций. Перед стартом транзакции необходимо выполнить метод beginTransaction() объекта сессии Session, возвращающего ссылку, которую можно использовать для подтверждения или отката транзакции.

Любое вызываемое в Hibernate исключение автоматически вызывает rollback().

B EclipseLink

EclipseLink — эталонная реализация для JPA с открытым исходным кодом. EclipseLink, созданный Eclipse Foundation основан на базе другого продукта TopLink компании Oracle [57].

EclipseLink предоставляет сервисы, позволяющие разрабатывать гибкие и эффективные приложения с широким спектром функций [58].

EclipseLink, являясь реализацией JPA, позволяет осуществлять отображение объектов Java на реляционную базу данных, используя Java аннотации или XML.

Приложениям, использующим данный сервис доступны следующие возможности:

- объектно-ориентированный язык JPQL;
- управление транзакциями;
- настройка кэширования;
- поддержка специфических возможностей базы данных;

Севрис MOXy (Mapping Objects на XML): этот сервис позволяет разработчикам Java эффективно связывать классы Java с XML и JSON. MOXy реализует JAXB, позволяет сохранять данные в формате XML и обрабатывать сложные XML-структуры.

Сервис Service Data Object (SDO) позволяет генерировать статические объекты данных из XSD.

Сервис Database Web Services (DBWS) генерирует необходимые файлы конфигурации на основе предоставленных артефактов базы данных для выполнения операций с базами данных посредством Web сервисов.

Сервис Enterprise Information Systems (EIS) используется для нереляционных баз данных, которые не предоставляют доступ с помощью JDBC и SQL [58].

Одно из главных преимуществ EclipseLink заключается в возможности вызывать собственные функции SQL непосредственно в своих запросах JPQL. В Hibernate это напрямую невозможно.

EclipseLink предоставляет несколько разных типов кэша, которые имеют разные требования к памяти. Размер кэша (в количестве кэшированных объектов) также можно настроить.

FULL кэширует все объекты и не удаляет их, даже если памяти мало. Размер кэша удваивается при достижении максимального размера.

WEAK - кэширует только те объекты, которые не были собраны сборщиком мусора. Объекты доступны для сборки мусора, когда приложение больше не ссылается на них на стороне сервера.

SOFT - похож на WEAK кэш, за исключением того, что кэш использует “мягкие” ссылки вместо “слабых”. По умолчанию EclipseLink использует SOFT_CACHE.

SOFT_WEAK и HARD_WEAK - похожи на WEAK кэш, за исключением того, что использует подкэш, чтобы гарантировать, что сборка мусор будет происходить только в том случае, если в JVM недостаточно памяти.

SOFT_WEAK и HARD_WEAK обеспечивают более эффективное использование памяти. Они освобождают объекты по мере их сбора мусора, за исключением фиксированного числа недавно использованных объектов [58].

Еще одна особенность EclipseLink — это возможность использования хранимых процедур и именованных запросов [49]. EclipseLink JPA поддерживает как реляционные [55], так и нереляционные базы данных.

EclipseLink позволяет интегрировать гетерогенную базу данных, в том числе, благодаря функциям, дающим возможность пользователю выбирать, как значения базы данных должны быть преобразованы в модель домена, а затем преобразованы обратно для записи в базу данных [55].

EclipseLink поддерживает самые современные и оптимизированные службы ORM, а также полный набор интерфейсов и методов из спецификации JPA. Уровень персистентности данных API EclipseLink включает в себя три основных функции: отображение метаданных (постоянные данные), общий кэш (метод кэширования координат) и оптимизация запросов с использованием JPQL.

C OpenJPA

OpenJPA — это еще одна реализация JPA, предоставляемая Apache Software, также соответствующая стандартам JPA. Цель OpenJPA - обеспечить высокую производительность и быстрый доступ к данным с помощью ORM [21].

OpenJPA сохраняет информацию через утвержденный стандартный язык запросов JPQL. Он также поддерживает классический язык структурированных запросов (SQL), который можно использовать для выполнения быстрых запросов с реляционной базой данных.

Open JPA — это полностью совместимая среда, включающая поддержку Java. Функции сохранения данных в OpenJPA, такие как JPQL, автоматическая схема базы данных, пакетный оператор и двухуровневое кэширование, повышают производительность API [50].

Кроме того, OpenJPA поддерживает автоматизацию схемы базы данных при запуске приложения, устанавливая свойство конфигурации, такое как jpa.jdbc.SynchronizeMappings [7]. и в том числе файл jar, например Openjpa-all 2.2.0 [7]. Другим важным свойством в файле persistence является openjpa.jdbc.Schema со значением = "openJPA", без которого сущности преобразуются в метаданные, но не сохраняются в базе данных.

OpenJPA обеспечивает поддержку обработки транзакций и множество механизмов блокировки для повышения общей производительности API.

Кеширование в OpenJPA имеет два уровня: кэш данных (DataCache), который извлекает сущности, загруженные из баз данных, и кэш запросов (QueryCache), в котором хранится первичный ключ столбец, возвращенный в транзакции [6].

VI. Сравнительный анализ производительности JPA-реализаций

Высокая производительность является фундаментальным требованием любого высоко нагрузочного проекта. Данный раздел посвящен сравнительному анализу производительности наиболее популярных JPA-реализаций в различных сценариях использования.

a. Тестирование JPQL-запросов.

Первый выполненный тест с использованием JPQL-запросов.

Тест проведен с использованием класса SimplePerson – это класс, состоящий из девяти текстовых полей, трех полей, хранящих значение в виде даты и времени, одно числовое поле. Первичный ключ генерируется автоматически. Данные генерируются автоматически.

Тестируются операции select, update, delete. Тесты проведены для 1000 пользователей, 100 000 и 500 000 пользователей с использованием СУБД PostgreSQL. В таблице ниже представлены результаты, в виде среднего времени при выполнении пяти итераций.

SELECT

С помощью запроса SELECT выполняется выборка всех данных из таблицы.

Таблица 1 Выборка данных

	1 000	100 000	500 000
EclipseLink	9 ms	462 ms	2 807 ms
Hibernate	11 ms	378 ms	1 907 ms
OpenJPA	51 ms	454 ms	2 106 ms

Из таблицы 1 видно, что при небольшом количестве записей наилучший результат при выборке данных из таблицы показывает EclipseLink, следом идет Hibernate, OpenJpa показывает результаты в пять раз хуже. При большем количестве записей, лучший результат показывает реализация Hibernate, затем OpenJpa. EclipseLink показывает наихудший результат при наибольшем количестве записей.

UPDATE

В таблице представлено среднее время при выполнении запроса изменяющего имя пользователя во всех записях таблицы SimplePerson.

Таблица 2 Операция обновления

	1 000	100 000	500 000

EclipseLink	23 ms	1 813 ms	15 941 ms
Hibernate	22 ms	1 790 ms	13 791 ms
OpenJPA	55 ms	2 011 ms	17 336 ms

Обновление данных в таблице быстрее всего производится с помощь реализации Hibernate, EclipseLink показывает сопоставимые результаты. В то время как OpenJPA показывает наихудшие результаты.

DELETE

Ниже представлено среднее время выполнения запроса на удаление всех записей таблицы.

Таблица 3 Удаление данных

	1 000	100 000	500 000
EclipseLink	16 ms	494 ms	671 ms
Hibernate	14 ms	456 ms	605 ms
OpenJPA	28 ms	524 ms	718 ms

Удаление записей из таблицы показывает аналогичные результаты предыдущего теста. Наилучшие показатели у Hibernate, далее EclipseLink и с заметным отрывом OpenJPA.

b. Тестирование кэша.

Для тестирования кэша взят класс SimplePerson, описанный в первом тесте. Создается 17 000 записей, далее происходит выборка всех данных из таблицы три раза подряд, затем выборка имени пользователя три раза подряд, и выборка фамилии пользователя также три раза подряд все операции выполняются в рамках одной сессии.

Таблица 4 Кэширование данных

	H	E	O
Все записи (1-й раз)	*1 298 ms*	*1 060 ms*	*3 064 ms*
Все записи (2-й раз)	880 ms	622 ms	740 ms
Все записи (3-й раз)	858 ms	580 ms	152 ms
First Name (1-й раз)	*613 ms*	*1 258 ms*	*1 450 ms*
First Name (2-й раз)	90 ms	70 ms	119 ms
First Name (3-й раз)	87 ms	55 ms	116 ms
Last Name (1-й раз)	*540 ms*	*988 ms*	*1 340 ms*
Last Name (2-й раз)	41 ms	48 ms	99 ms
Last Name (3-й раз)	38 ms	60 ms	85 ms

По результатам таблицы видно, что во втором и в третьем вызовах все запросы показывают существенные улучшения по сравнению с первым вызовом, благодаря встроенным по умолчанию кэшам первого уровня.

c. Тест CRUD операций с использованием транзакций.

Следующий тест анализирует производительность JPA-реализации на примере операций создания и сохранение пользователя в базу данных, обновление и удаление пользователя с помощью транзакций, с использованием стратегии «один Entity Manager на одну транзакцию приложения».

Тестируются операции create, update (изменение поля Имя), delete. Тесты проведены для 1000 позиций пользователей, 10 000 и 100 000 пользователей с использованием СУБД PostgreSQL.

CREATE

Создание и сохранение пользователя с помощью транзакций.

Таблица 5 Транзакции

	1 000	10 000	100 000
EclipseLink	10 087 ms	111 529 ms	996 982 ms
Hibernate	11 217 ms	100 583 ms	994 394 ms
OpenJPA	11 388 ms	114 713 ms	1 004 612 ms

UPDATE

Обновление поля Имя для каждого пользователя с помощью транзакций.

Таблица 6 Обновление в транзакции

	1 000	10 000	100 000
EclipseLink	10 375 ms	47 583 ms	999 902 ms
Hibernate	10 758 ms	49 746 ms	1 017 725 ms
OpenJPA	10 209 ms	48 058 ms	997 425 ms

DELETE

Удаление пользователя с помощью транзакций.

Таблица 7 Удаление в транзакции

	1 000	10 000	100 000
EclipseLink	10 286 ms	84 935 ms	742 914 ms
Hibernate	11 045 ms	103 880 ms	735 837 ms
OpenJPA	10 031 ms	91 422 ms	745 513 ms

В целом все реализации показывают примерно одинаковые результаты с незначительным разбросом. EclipseLink лучше справился с четырьмя тестами, Hibernate с тремя и OpenJPA с двумя из девяти.

d. Тест наследование в JPA.

Следующий тест анализируют производительность EclipsLink, Hibernate и OpenJPA при реализации стратегии наследования InheritanceType.JOINED.

Для проведения теста используется иерархия из трех классов, каждый наследуется от предыдущего.

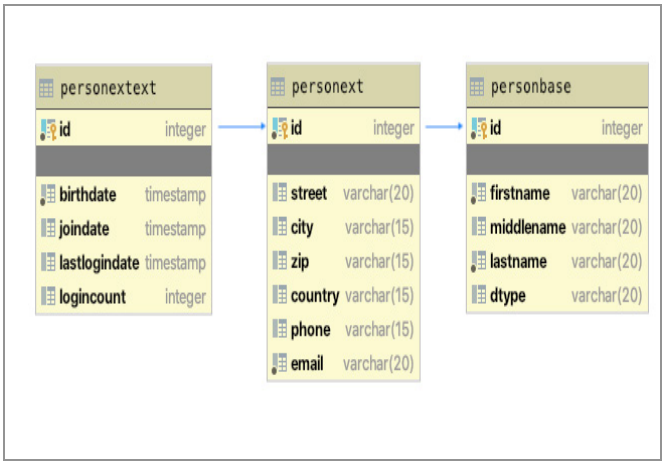

Рис. 4. Схема базы данных PostgreSQL со стратегией наследование Joined.

Протестированы операции create, и delete. Тесты проведены для 1000 позиций пользователей, 10 000 и 100 000 пользователей с использованием СУБД PostgreSQL.

В таблице ниже представлены результаты, в виде среднего времени при выполнении пяти итераций.

CREATE

Сохранение пользователя с помощью транзакций.

Таблица 8 Создание записей в транзакции

	1 000	10 000	100 000
EclipseLink	14 584 ms	126 748 ms	1 272 268 ms
Hibernate	13 928 ms	127 517 ms	1 230 551 ms
OpenJPA	27 464 ms	123 813 ms	1 228 921 ms

При сохранении пользователей в базу данных наилучшую производительность показывает при количестве записей 1 000 штук Hibernate, при большем количестве записей EclipseLink.

DELETE

Удаление пользователя с помощью транзакций.

Таблица 9. Удаление в транзакции

	1 000	10 000	100 000
EclipseLink	15 499 ms	141 315 ms	1 452 083 ms
Hibernate	16 487 ms	143 389 ms	1 459 271 ms
OpenJPA	28 654 ms	143 389 ms	1 507 774 ms

В целом в тесте наследования наилучшие результаты показала релизация EclipseLink. OpenJPA показала очень низкие результаты на небольших данных.

e. Тестирование связи "Один-ко-многим"

Для проведения тестирования отображения "Один-ко-многим" или "one-to-many" создан класс Person и класс Address, а также две таблицы для хранения данных соответственно. При создании каждого пользователя создается два адреса.

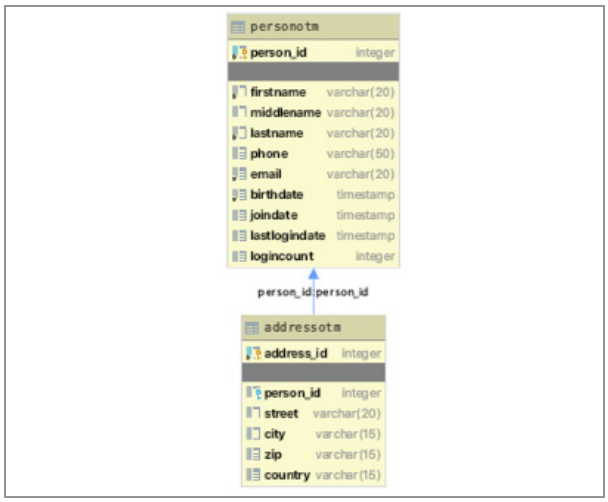

Рис.5. Схема базы данных PostgreSQL со связью "один-ко-многим".

Протестированы операции create, и delete. Тесты проведены для 1000, 10 000 и 100 000 пользователей соответственно с использованием СУБД PostgreSQL. В таблице ниже представлены результаты теста.

CREATE

В каждой транзакции создается один пользователь с двумя адресами.

Таблица 10

	1 000	10 000	100 000
EclipseLink	15 354 ms	141 908 ms	1 463 920 ms
Hibernate	10 573 ms	100 330 ms	1 426 622 ms
OpenJPA	27 541 ms	142 955 ms	1 471 893 ms
MyBatis	10 654 ms	90 653 ms	1 413 479 ms

DELETE

В каждой транзакции удаляется пользователь и все адреса, привязанные к его id.

Таблица 11

	1 000	10 000	100 000
EclipseLink	20 274 ms	222 318 ms	4 447 358 ms
Hibernate	19 460 ms	221 744 ms	4 418 799 ms
OpenJPA	23 910 ms	226 713 ms	4 393 602 ms
MyBatis	11 378 ms	134 357 ms	3 295 006 ms

Наилучшие результаты показывает MyBatis, что не удивительно, так как отображение производится не на таблицы, как в JPA-реализации, а на SQL запросы. Среди реализаций лучше всех с тестом справился Hibernate.

f. Тестирование JPQL-запросов с разными базами данных.

В последнем тесте JPQL-запросы протестированы на нескольких базах данных: PostgreSQL, MariaDB, MySQL. Как и в-первом тесте, все запросы тестируются на классе SimplePerson, состоящим из текстовых и числовых полей.

Тестируются операции select, update, delete. Тесты проведены для 10 000 пользователей, результаты представлены в виде среднего значения пяти итераций.

CREATE

В таблице ниже представлено среднее время при сохранении нового пользователя.

Таблица 12

	10 000		
	PostgreSQL	MariaDB	MySQL
EclipseLink	44 ms	32 ms	34 ms
Hibernate	68 ms	20 ms	45 ms
OpenJPA	92 ms	67 ms	100 ms

UPDATE

В таблице ниже представлено среднее время при выполнении запроса изменяющего имя пользователя.

Таблица 13

	10 000		
	PostgreSQL	MariaDB	MySQL
EclipseLink	188 ms	30 ms	70 ms
Hibernate	195 ms	27 ms	74 ms
OpenJPA	304 ms	41 ms	97 ms

DELETE

В таблице ниже представлено среднее время при удалении пользователя.

Таблица 6.16

	10 000		
	PostgreSQL	MariaDB	MySQL
EclipseLink	157 ms	10 ms	13 ms
Hibernate	165 ms	9 ms	14 ms
OpenJPA	185 ms	24 ms	36 ms

Результаты теста выше показали, что, в то время как одни реализации эффективны при работе с одной базой данных, в это же время другие более эффективны с другими базами данных.

При работе с базой данных PostgreSQL и MySQL лучше использовать реализации EclipseLink или Hibernate, при этом EclipseLink показал чуть лучшие результаты.

g. Memory Usage, CPU u Total Classes Loaded.

Тест ниже оценивает влияние и нагрузку при выполнении запроса на ЦП (центральный процессор).

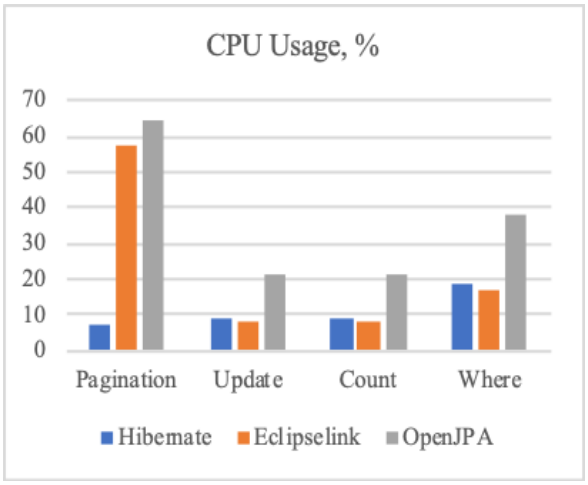

Рис. 6. CPU Usage.

На Рис. 6.3 показано сколько процентов ресурсов ЦП использовалось при выполнении запросов на пагинацию, подсчет числа записей, оператора where и замены имени пользователя в таблице SimplePerson с помощью JPQL-запросов. Все операции производились на таблице, состоящей из 500 000 записей.

Чем меньше значение CPU Usage, тем меньше нагрузка на процессор при выполнении операции.

Из Рис. 6.3 видно, что наибольшую нагрузку на ЦП дает OpenJPA, особенно при разбиении на страницы. Hibernate показал наименьшую нагрузку на ЦП при выполнении запросов.

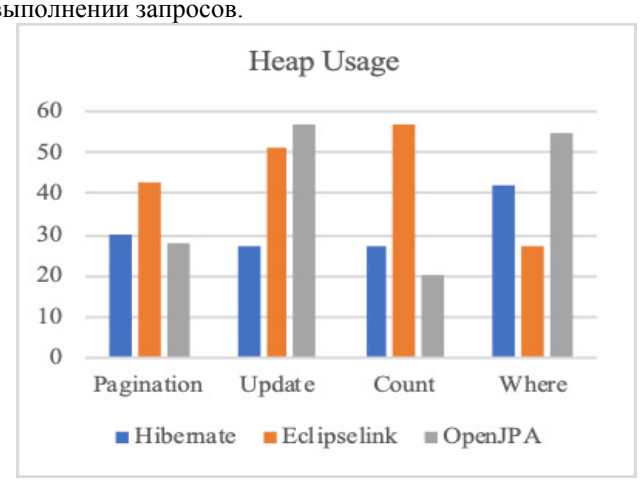

Рис. 7. Heap Usage.

Память в JVM состоит из трех основных сегментов: куча(heap), Permanent Generation и стек (stack).

Рис. 6.4 Heap Memory Usage показывает зависимость количества памяти, выделяемой из heap (кучи), в мегабайтах от времени.

В реализациях JPA измерение начального размера heap(кучи) и максимальных единиц, получаемых в конце запроса, влияет на производительность приложения. На рис. 6.4 показано среднее значение размера heap(кучи), используемое при выполнении в различных запросах.

На рис. Total classes loaded показано количество загруженных классов при выполнении запросов.

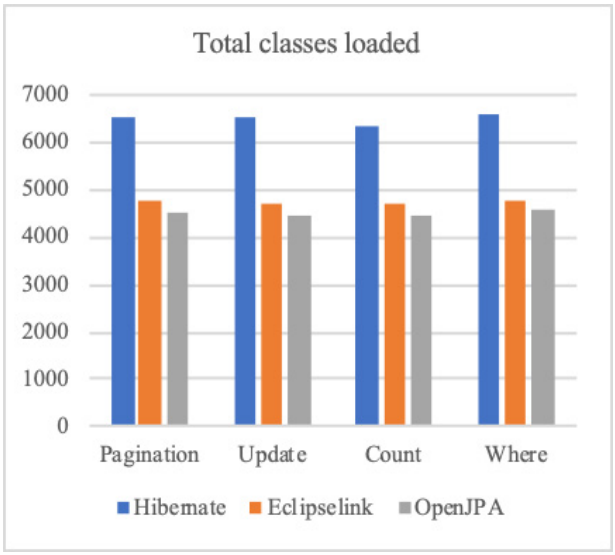

Рис. 8. Total classes loaded.

Анализ производительности с точки зрения количества загруженных классов дает разработчику знания о внутренней архитектуре JPA. Общее количество загружаемых классов представлено процентом байтов, а также количеством байтов, выделенных каждому классу.

Он описывает визуальное представление всех загруженных классов с количеством общих классов в одном приложении по шкале от 0 до 7000 классов. Хотя Hibernate использует максимальное количество классов для выполнения операций, производительность Hibernate по сравнению с другими реализациями JPA является оптимальной.

VII. MyBatis vs Hibernate

Для полного обзора ORM технологий сравним одну из JPA-реализаций, на примере Hibernate и Mybatis. Как было показано в предыдущем разделе, MyBatis отображается на запросы, а не на сущности, следовательно, не нужно привязывать сущности к маппингу как в Hibernate. MyBatis не реализует JPA. В MyBatis используются SQL-запросы, а значит легко найти и воспроизвести в них ошибку.

В Mybatis легко использовать хранимые процедуры, так как в них используются именованные параметры. Но в случае с использованием XML сложно отладить программу, также есть проблемы с использованием синтаксиса (например, знак меньше в запросах может воспринимать как кавычку).

Hibernate отображается на сущности, а не запросы. В Hibernate нет запросов, следовательно не обязательно знать SQL, а можно использовать HQL или JPQL. Hibernate сам генерирует запросы, и не всегда понятно как это происходит внутри.

Hibernate очень неудобен при использовании хранимых процедур.

В целом стоит отметить, что MyBatis гораздо проще Hibernate, но предоставляет меньше возможностей. У MyBatis низкий порог для входа, чтобы использовать его в проекте достаточно добавить dependency, установить конфигурацию и далее выбрать использование аннотаций или xml.

Hibernate стоит использовать на проектах с частым использованием CRUD операций. Он очень удобен в новых проектах с часто меняющейся схемой БД. Также стоит отдать предпочтение Hibernate там, где возникает необходимость в использование кэширования.

MyBatis стоит использовать, если в проекте есть какой-то поиск, если часто используются выборки, отчеты. Также MyBatis более удобен в случае использования ненормализованной базы данных.

VIII. Заключение

В работе исследованы современные технологии работы с базами данных в Java-приложениях. Определены преимущества и недостатки при работе как с помощью низкоуровневых технологий JDBC, так и высокоуровневых, на примере, ORM-фреймворков MyBatis, EclipseLink, Hibernate и OpenJPA.

У каждого из подходов есть свои преимущества и недостатки. В случае использования JDBC высокий уровень развития современных технологий затрудняет разработчику выяснить, какие SQL-запросы будут выполняться в заданном месте исходного кода программы или какие методы исходного кода фактически обращаются к данной таблице или столбцу базы данных. И наоборот, высокий уровень абстракции, обеспечиваемый ORM, затрудняет определение влияния на программный код изменений в схеме базы данных. В целом в большинстве проектов сегодня Java-разработчики взаимодействуют с JDBC через ORM-технологии. Так как они освобождают разработчиков от написания Java-кода для управления соединениями к базе данных, запросами, кэшем и объектно-реляционным отображением Java объектов.

Для единого взаимодействия с базами данных был разработан стандарт JPA, в нем определен общий набор интерфейсов, аннотаций и прочих служб, которые реализуют поставщики JPA: EclipseLink, Hibernate и OpenJPA.

В рамках данной работы были разработаны и проведены различные сценарии для оценки производительности JPA-реализаций. По результатам проведенных тестов можно сделать следующие выводы:

Во-первых, сложно выделить одну реализацию, которая показала бы наилучшие результаты во всех теста. При этом EclipseLink и Hibernate показывают довольно близкие результаты, в отличие от OpenJPA, который показал худшие результаты по большинству операций.

Также стоит заметить, что EclipseLink и Hibernate более оптимизированы для небольшого количества запросов, выполняемых одновременно и при резком увеличение количества запросов, время на их выполнение резко возрастает.

Если сравнивать MyBatis, Hibernate и EclipseLink, то наилучшие результаты в плане производительности показывает MyBatis. Основное отличие заключается в том, как происходит отображение таблицы БД на сущности. Hibernate и EclipseLink отображает таблицы БД на сущности, давая доступ к данным, для получения данных генерируются SQL запросы. MyBatis в свою очередь отображается не на таблицы, а на SQL запросы,

за формирование запросов отвечает разработчик и от него зависит, как быстро будет работать приложение. В проектах, которые часто используют сложные запросы к базе данных, при работе с хранимыми процедурами, и поиском стоит использовать MyBatis. В проектах, где часто используются CRUD операции, возникает необходимость кэширования и часто меняется схема БД стоит использовать Hibernate или EclipseLink.

На одном проекте можно смешивать ORM, при строгом разграничении сфер. Большие отчеты стоит реализовывать с MyBatis, сущности с Hibernate или EclipseLink, что даст высокую скорость разработки.

Тест с разными базами данных показал, что провайдеры показывают разную производительность при работе с разными базами. Например, при работе с базой данных PostgreSQL и MySQL лучше использовать реализации EclipseLink или Hibernate, при этом EclipseLink показал чуть лучшие результаты. При работе с базой данных MariaDB лучше использовать Hibernate. То есть при выборе ORM-реализации также стоит учитывать то, с какой базой данных она будет взаимодействовать.

Хотя Hibernate использует максимальное количество классов для выполнения операций, производительность Hibernate по сравнению с другими реализациями JPA является оптимальной. Hibernate и EclipseLink также показали низкую нагрузку на ЦП.

Интересным направлением исследования и дальнейшего развития системы является разработка набора функций для упрощения миграции между реализациями JPA.

Библиография

[1] Neha Dhingra Review on JPA Based ORM Data Persistence Framework, International Journal of Computer Theory and Engineering, Vol. 9, October 2017

[2] Benchmark Results Comparison JPA Performance Benchmark (JPAB), "JPA Performance Benchmark", Copyright 2012 ObjectDB Software Ltd. Available: http://www.jpab.org/

[3] J. Keogh, C.J. Date, H. Darwen. (2002). J2EE: The Complete Reference: A Guide to the SQL Standard,New York, NY: Tata McGraw-Hill Education.

[4] Kumari, Sonia; Yadav, Manvendra; Kumari, Seema Rani. Database Connection Technology // International Journal of Advanced Research in Computer Science; Udaipur Том 8, Изд. 5, (May 2017).

[5] B.Vasavi, Y.V.Sreevani, G.Sindhu Priya(2011) “Hibernate technology for an efficient business application extension”, Journal of Global Research in Computer Science, 2(6), 118-125

[6] Lascano, J. "JPA implementations versus pure JDBC." URL: http://www. espe. edu. ec/portal/files/sitiocongreso/ congreso/c_computacion/PaperJPAversusJDBC_edisonl ascano. pdf. 2008.

[7] Apache OpenJPA User Guide 2.3 . [Online]. Available: http://openjpa.apache.org/builds/latest/docs/docbook/manual/main. html. 2011.

[8] M. Goeminne and T. Mens, “Towards a survival analysis of database framework usage in Java projects,” in Int’l Conf. Software Maintenance and Evolution, 2015.

[9] Mane, Dashrath, Ojha, Namrata, and Chitnis, Ketaki. The spring framework: An open source java platform for developing robust java applications. International Journal of Innovative Technology and Exploring Engineering.

[10] Neha Dhingra, Dr. Emad Abdelmoghith, Dr. Hussein T. Mouftah. PERFORMANCE EVALUATION OF JPA BASED ORM TECHNIQUES // Proceedings of 2nd International Conference on Computer Science Networks and Information Technology, Heldon 27th - 28th Aug 2016, in Montreal, Canada, ISBN: 9788193137369

[11] Alexandre Decan, Mathieu Goeminne,and Tom Mens On the Interaction of Relational Database Access Technologies in Open Source Java Projects // Postproceeding of the SATTOSE 2015 Research Seminar on Advanced Tools and Techniques for Software Evolution. To be published in CEUR.WS workshop proceedings (2017)

[12] Loup Meurice, Csaba Nagy, Anthony Cleve Detecting and Preventing Program Inconsistencies Under Database Schema Evolution // 2016 IEEE International Conference on Software Quality, Reliability and Security

[13] JPA implementations versus pure JDBC Conference: Congreso de Ciencia y Tecnología de la ESPE-2008, At Quito, Ecuador, 2008

[14] Liang, D. “Rapid Java Application Development Using Sun ONE TM Studio 4”, New Jersey: Prentice Hall, 2003,1st edition, pg. 514

[15] Penchikala, S., O Reilly on Java.com: The independent source for Enterprise Java (2006), “JDBC 4.0 Enhancements in Java SE 6”, Retrieved April 14, 2008, from www.onjava.com website: http://www.onjava.com/pub/a/onjava/2006/08/02/jjdb c-4-enhancements-in-java-se-6.html

[16] E. E. Ogheneovo, P. O. Asagba, N. O. Ogini. An Object Relational Mapping Technique for Java Framework, International Journal of Engineering Science Invention, 2013, Pages: 01-9, Vol: 6

[17] Tse-Hsun Chen, Weiyi Shang, Parminder Flora CacheOptimizer: helping developers configure caching frameworks for hibernate-based database-centric web applications // Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering Pages 666-677 Seattle, WA, USA — November 13 - 18, 2016

[18] Hibernate (2008). “Java Persistence with Hibernate”, retrieved April 12, 2008, from www.hibernate.org website: http://www.hibernate.org/397.html

[19] Penchikala, S., O Reilly on Java.com: The independent source for Enterprise Java (2006), “JDBC 4.0 Enhancements in Java SE 6”, Retrieved April 14, 2008

[20] HIBERNATE - Relational Persistence for Idiomatic Java , http://www.Hibernate.org/hib_docs/v3/reference/en/html/preface.html

[21] Apache OpenJPA User Guide 2.3. http://openjpa.apache.org/builds/latest/docs/docbook/manual/main. html. 2011.

[22] M. Goeminne and T. Mens. Towards a survival analysis of database framework usage in Java projects. In Int’l Conf. Software Maintenance and Evolution (ICSME), 2015.

[23] Haseeb Yousaf. “Performance evaluation of Java Object-Relational Map- ping tools”. PhD thesis. 2012.

[24] EclipseLink Documentation Center; http://wiki.eclipse.org/EclipseLink/Documentation_Center; accessed Oct. 2011

[25] Doug Clarke. https://dzone.com/articles/ introducing-eclipselink. Jun. 30, 2008

[26] Spring Data, https://spring.io/projects/spring-data

[27] Thorben Janssen, Hibernate Tips: More than 70 solutions to common Hibernate problems, Thoughts on Java, 2018. — 256 p.

[28] W. Jing, R. Fan, The research of hibernate cache technique and applicationof ehcache component, in: 2011 IEEE 3rd International Conference on Communication Software and Networks, (ICCSN), IEEE, 2011, pp.160–162

[29] A. Silberschatz, H. Korth, and S. Sudarshan, Database System Concepts. McGraw-Hill, 2010.

[30] R. Stephens, Beginning Database Design Solutions. John Wiley & Sons, 2010.

[31] C. Bauer and G. King, Java persistence with Hibernate. Man- ning Pubs Co Series, Manning, 2007.

[32] M. Keith and M. Schincariol, Pro JPA 2: mastering the Java Per- sistence API. Apress Series, Apress, 2009.

[33] “JSR 317: Java Persistence API, Version 2.0.” http://jcp. org/en/jsr/detail?id=317, Dec. 2009

[34] Analysis of ORM based JPA Implementations. Neha Dhingra, Ottawa, Canada, 2017 Published 2017, Computer Science

[35] Performance Improvement of OpenJPA by Query Dependency Analysis, Miki Enoki, Yosuke Ozawa, Tamiya Onodera // Conference on Database Systems for Advanced Applications 2010: Database Systems for Advanced Applications pp 370-379

[36] MYBATIS Tutorial. –http://www.tutorialspoint. com/mybatis/.

[37] MyBatis Introduction. –http://www.mybatis.org/mybatis-3/.

[38] Comparative analysis of data persistence technologies for large-scale models Authors: Konstantinos Barmpis, Dimitrios S. Kolovos // Proceedings of the 2012 Extreme Modeling WorkshopOctober 2012 Pages 33-38

[39] Optimizing JPA Performance: An EclipseLink, Hibernate, and OpenJPA Comparison by Daniel Rubio, Jul. 20, 10//Performance Zone
[40] Memory-Efficient index for cache invali- dation mechanism with OpenJPA, In International Conference on Web Information Systems Engineering, 2014, Pages: 696-703.
[41] E. E. Ogheneovo, P. O. Asagba, N. O. Ogini. An Object Relational Mapping Technique for Java Framework, International Journal of Engineering Science Invention, 2013.
[42] M. Keith, M. Schincariol. Introduction to InPro JPA 2, Apress, 2013, Pages: 1-110
[43] S. Rodrguez. JPA implementations comparison: Hibernate, Toplink Essentials, OpenJPA and Eclipselink, Available At: newblock http://blog.eisele.net/2009/01/jpa-implementations-comparison.html 2011
[44] M. Debnath. Manage Concurrent Access to JPA Entity with Locking, Available At: http://www.developer.com/java/manage-concurrent-access-to-jpa-entity-with- locking.html, 2013
[45] H. Yousaf. Performance Evaluation of Java Object Relational Mapping, University of Georgia Athens, Master Thesis, 2012
[46] M. Keith and M. Schincariol, Pro JPA 2: mastering the Java Persistence API. Apress Series, Apress, 2009.
[47] S. Devijver. The problem with JPA/Hibernate (or the future of ORM), Available At: https://dzone.com/articles/problem-jpahibernate-or-future-orm, 2008
[48] Hibernate developer guide: http://docs.jboss.org/ hibernate/core/4.0/devguide/en-US/html/.
[49] EclipseLink project wiki: http://wiki.eclipse.org/ Category: EclipseLink/Documentation/JPA.
[50] Apache OpenJPA User Guide: http://openjpa.apache.org/ builds/2.1.1/apache-openjpa/docs/manual.html.
[51] Design and Application of the Hibernate Persistence Layer Data Report System using JasperReports, B. S. Sapre, R. V. Thakare, S. V. Kakade. International Journal of Engineering and Innovative Technology (IJEIT) , 2012
[52] M. Keith, M. Schincariol. Introduction to InPro JPA 2, Apress, 2013
[53] C. Bauer, G. King. Java Persistance with Hibernate, 2006, Pages: 123-302
[54] Hibernate ORM 4.0 API documentation: http://docs. jboss.org/hibernate/orm/4.0/javadocs/
[55] Eclipselink Transaction management. Introduction to EclipseLink Transactions. Available At: https://wiki.eclipse.org/, 2014
[56] OpenJPA. What is Enhancement Anyway? In OpenJPA , Available At: http://openjpa.apache.org/entity-enhancement.html, 2017
[57] EclipseLink 2.3 API documentation: http://www.eclipse.org/eclipselink/api/2.3/index.html.
[58] EclipseLink project wiki: http://wiki.eclipse.org/ Category: EclipseLink/Documentation/JPA.

Comparative analysis of data access technologies in Java applications

Vitaliya Dashuk, Dmitry Namiot

Abstract — **This article discusses modern approaches to working with databases in Java projects. The advantages and disadvantages of working with low-level JDBC technology are considered. This article provides an overview of the JPA specification, a standard for mapping POJO objects to relational databases. The paper gives an overview of the most popular JPA implementations: Hibernate, EclipseLink, and OpenJPA. The capabilities of the MyBatis ORM system, which does not implement JPA, but presents an alternative, are also noted. In the course of the work, a test suite has been proposed that compares the performance of JPA implementations. A comparison of MyBatis and one of the Hibernate-based JPA implementations are given. The results and conclusions of the tests are presented, giving an understanding of which solutions are best suited for different types of situations.**

Keywords — **JDBC, MyBatis, JPA, Hibernate, EclipseLink, OpenJPA.**

REFERENCES

[1] Neha Dhingra Review on JPA Based ORM Data Persistence Framework, International Journal of Computer Theory and Engineering, Vol. 9, October 2017

[2] Benchmark Results Comparison JPA Performance Benchmark (JPAB), "JPA Performance Benchmark", Copyright 2012 ObjectDB Software Ltd. Available: http://www.jpab.org/

[3] J. Keogh, C.J. Date, H. Darwen. (2002). J2EE: The Complete Reference: A Guide to the SQL Standard,New York, NY: Tata McGraw-Hill Education.

[4] Kumari, Sonia; Yadav, Manvendra; Kumari, Seema Rani. Database Connection Technology // International Journal of Advanced Research in Computer Science; Udaipur Том 8, Изд. 5, (May 2017).

[5] B.Vasavi, Y.V.Sreevani, G.Sindhu Priya(2011) "Hibernate technology for an efficient business application extension", Journal of Global Research in Computer Science, 2(6), 118-125

[6] Lascano, J. "JPA implementations versus pure JDBC." URL: http://www. espe. edu. ec/portal/files/sitiocongreso/ congreso/c_computacion/PaperJPAversusJDBC_edisonl ascano. pdf. 2008.

[7] Apache OpenJPA User Guide 2.3 . [Online]. Available: http://openjpa.apache.org/builds/latest/docs/docbook/manual/main. html. 2011.

[8] M. Goeminne and T. Mens, "Towards a survival analysis of database framework usage in Java projects," in Int'l Conf. Software Maintenance and Evolution, 2015.

[9] Mane, Dashrath, Ojha, Namrata, and Chitnis, Ketaki. The spring framework: An open source java platform for developing robust java applications. International Journal of Innovative Technology and Exploring Engineering.

[10] Neha Dhingra, Dr. Emad Abdelmoghith, Dr. Hussein T. Mouftah. PERFORMANCE EVALUATION OF JPA BASED ORM TECHNIQUES // Proceedings of 2nd International Conference on Computer Science Networks and Information Technology, Heldon 27th - 28th Aug 2016, in Montreal, Canada, ISBN: 9788193137369

[11] Alexandre Decan, Mathieu Goeminne,and Tom Mens On the Interaction of Relational Database Access Technologies in Open Source Java Projects // Postproceeding of the SATTOSE 2015 Research Seminar on Advanced Tools and Techniques for Software Evolution. To be published in CEUR.WS workshop proceedings (2017)

[12] Loup Meurice, Csaba Nagy, Anthony Cleve Detecting and Preventing Program Inconsistencies Under Database Schema Evolution // 2016 IEEE International Conference on Software Quality, Reliability and Security

[13] JPA implementations versus pure JDBC Conference: Congreso de Ciencia y Tecnología de la ESPE-2008, At Quito, Ecuador, 2008

[14] Liang, D. "Rapid Java Application Development Using Sun ONE TM Studio 4", New Jersey: Prentice Hall, 2003,1st edition, pg. 514

[15] Penchikala, S., O Reilly on Java.com: The independent source for Enterprise Java (2006), "JDBC 4.0 Enhancements in Java SE 6", Retrieved April 14, 2008, from www.onjava.com website: http://www.onjava.com/pub/a/onjava/2006/08/02/jjdb c-4-enhancements-in-java-se-6.html

[16] E. E. Ogheneovo, P. O. Asagba, N. O. Ogini. An Object Relational Mapping Technique for Java Framework, International Journal of Engineering Science Invention, 2013, Pages: 01-9, Vol: 6

[17] Tse-Hsun Chen, Weiyi Shang, Parminder Flora CacheOptimizer: helping developers configure caching frameworks for hibernate-based database-centric web applications // Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering Pages 666-677 Seattle, WA, USA — November 13 - 18, 2016

[18] Hibernate (2008). "Java Persistence with Hibernate", retrieved April 12, 2008, from www.hibernate.org website: http://www.hibernate.org/397.html

[19] Penchikala, S., O Reilly on Java.com: The independent source for Enterprise Java (2006), "JDBC 4.0 Enhancements in Java SE 6", Retrieved April 14, 2008

[20] HIBERNATE - Relational Persistence for Idiomatic Java , http://www.Hibernate.org/hib_docs/v3/reference/en/html/preface.html

[21] Apache OpenJPA User Guide 2.3. http://openjpa.apache.org/builds/latest/docs/docbook/manual/main. html. 2011.

[22] M. Goeminne and T. Mens. Towards a survival analysis of database framework usage in Java projects. In Int'l Conf. Software Maintenance and Evolution (ICSME), 2015.

[23] Haseeb Yousaf. "Performance evaluation of Java Object-Relational Map- ping tools". PhD thesis. 2012.

[24] EclipseLink Documentation Center; http://wiki.eclipse.org/EclipseLink/Documentation_Center; accessed Oct. 2011

[25] Doug Clarke. https://dzone.com/articles/ introducing-eclipselink. Jun. 30, 2008

[26] Spring Data, https://spring.io/projects/spring-data

[27] Thorben Janssen, Hibernate Tips: More than 70 solutions to common Hibernate problems, Thoughts on Java, 2018. — 256 p.

[28] W. Jing, R. Fan, The research of hibernate cache technique and applicationof ehcache component, in: 2011 IEEE 3rd International Conference on Communication Software and Networks, (ICCSN), IEEE, 2011, pp.160–162

[29] A. Silberschatz, H. Korth, and S. Sudarshan, Database System Concepts. McGraw-Hill, 2010.

[30] R. Stephens, Beginning Database Design Solutions. John Wiley & Sons, 2010.

[31] C. Bauer and G. King, Java persistence with Hibernate. Man- ning Pubs Co Series, Manning, 2007.

[32] M. Keith and M. Schincariol, Pro JPA 2: mastering the Java Persistence API. Apress Series, Apress, 2009.
[33] "JSR 317: Java Persistence API, Version 2.0." http://jcp.org/en/jsr/detail?id=317, Dec. 2009
[34] Analysis of ORM based JPA Implementations. Neha Dhingra, Ottawa, Canada, 2017 Published 2017, Computer Science
[35] Performance Improvement of OpenJPA by Query Dependency Analysis, Miki Enoki, Yosuke Ozawa, Tamiya Onodera // Conference on Database Systems for Advanced Applications 2010: Database Systems for Advanced Applications pp 370-379
[36] MYBATIS Tutorial. –http://www.tutorialspoint. com/mybatis/.
[37] MyBatis Introduction. –http://www.mybatis.org/mybatis-3/.
[38] Comparative analysis of data persistence technologies for large-scale models Authors: Konstantinos Barmpis, Dimitrios S. Kolovos // Proceedings of the 2012 Extreme Modeling WorkshopOctober 2012 Pages 33-38
[39] Optimizing JPA Performance: An EclipseLink, Hibernate, and OpenJPA Comparison by Daniel Rubio, Jul. 20, 10//Performance Zone
[40] Memory-Efficient index for cache invali- dation mechanism with OpenJPA, In International Conference on Web Information Systems Engineering, 2014, Pages: 696-703.
[41] E. E. Ogheneovo, P. O. Asagba, N. O. Ogini. An Object Relational Mapping Technique for Java Framework, International Journal of Engineering Science Invention, 2013.
[42] M. Keith, M. Schincariol. Introduction to InPro JPA 2, Apress, 2013, Pages: 1-110
[43] S. Rodrguez. JPA implementations comparison: Hibernate, Toplink Essentials, OpenJPA and Eclipselink, Available At: newblock http://blog.eisele.net/2009/01/jpa-implementations-comparison.html 2011
[44] M. Debnath. Manage Concurrent Access to JPA Entity with Locking, Available At: http://www.developer.com/java/manage-concurrent-access-to-jpa-entity-with- locking.html, 2013
[45] H. Yousaf. Performance Evaluation of Java Object Relational Mapping, University of Georgia Athens, Master Thesis, 2012
[46] M. Keith and M. Schincariol, Pro JPA 2: mastering the Java Persistence API. Apress Series, Apress, 2009.
[47] S. Devijver. The problem with JPA/Hibernate (or the future of ORM), Available At: https://dzone.com/articles/problem-jpahibernate-or-future-orm, 2008
[48] Hibernate developer guide: http://docs.jboss.org/hibernate/core/4.0/devguide/en-US/html/.
[49] EclipseLink project wiki: http://wiki.eclipse.org/ Category: EclipseLink/Documentation/JPA.
[50] Apache OpenJPA User Guide: http://openjpa.apache.org/builds/2.1.1/apache-openjpa/docs/manual.html.
[51] Design and Application of the Hibernate Persistence Layer Data Report System using JasperReports, B. S. Sapre, R. V. Thakare, S. V. Kakade. International Journal of Engineering and Innovative Technology (IJEIT) , 2012
[52] M. Keith, M. Schincariol. Introduction to InPro JPA 2, Apress, 2013
[53] C. Bauer, G. King. Java Persistance with Hibernate, 2006, Pages: 123-302
[54] Hibernate ORM 4.0 API documentation: http://docs.jboss.org/hibernate/orm/4.0/javadocs/
[55] Eclipselink Transaction management. Introduction to EclipseLink Transactions. Available At: https://wiki.eclipse.org/, 2014
[56] OpenJPA. What is Enhancement Anyway? In OpenJPA , Available At: http://openjpa.apache.org/entity-enhancement.html, 2017
[57] EclipseLink 2.3 API documentation: http://www.eclipse.org/eclipselink/api/2.3/index.html.
[58] EclipseLink project wiki: http://wiki.eclipse.org/ Category: EclipseLink/Documentation/JPA.