# ЛАБОРАТОРНАЯ РАБОТА №7 Применение технологии ORM на базе Hibernate

## 1. Что такое транзакция?

Под транзакцией понимается ряд действий (не обязательно в БД), которые воспринимаются системой, как единый пакет, т.е. или все действия проходят успешно, или все откатываются на исходные позиции.

Существует два типа транзакций — локальные и распределенные. Локальная транзакция работает с одним источником, например одна БД; распределенная использует несколько — например jms source и БД.

Если механизм локальных транзакций прост, то для распределенных требуется определенная инфраструктура. Распределенный процесс требует некого элемента, который будет синхронизировать работу всех источников. Этот элемент — менеджер распределенных транзакций. Подобные менеджеры выпускаются большим количеством компаний, и все они требую определенного изучения документации.

Кроме того, можно использовать любой jEE сервер, т.к. они содержат уже настроенные компоненты.

## 2. В чем отличие JPA от Hibernate?

Hibernate — библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения, самая популярная реализация спецификации JPA.

Java Persistence API — спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных. JPA реализует концепцию ORM. Существует несколько реализаций этого интерфейса, одна из самых популярных использует для этого Hibernate

Сама по себе спецификация JPA не является инструментом или фреймворком; скорее, она определяет набор концепций, которые могут и должны быть реализованы любым другим инструментом.

Можно рассматривать JPA как интерфейс, а Hibernate – как реализацию

## 3. Что такое Entity?

**В широком смысле Entit** – это сущность, которая является отображением в базе данных. Связь между Entity – это зависимость одной сущности от другой. Очень часто используются они в построении больших БД.

Если быть точнее, Entity (Сущность) — POJO-класс, связанный с БД с помощью аннотации (@Entity) или через XML. К такому классу предъявляются следующие требования:

* Должен иметь пустой конструктор (public или protected)
* Не может быть вложенным, интерфейсом или enum
* Не может быть final и не может содержать final-полей/свойств
* Должен содержать хотя бы одно @Id-поле

При этом entity может:

* Содержать непустые конструкторы
* Наследоваться и быть наследованным
* Содержать другие методы и реализовывать интерфейсы

Entities могут быть связаны друг с другом (один-к-одному, один-ко-многим, многие-к-одному и многие-ко-многим)[[4]](https://ru.wikipedia.org/wiki/Java_Persistence_API#cite_note-4).

## 4. Конфигурационный файл Hibernate?

Hibernate требует заранее знать – где найти информацию о сопоставлении, которая определяет, как ваши классы Java связаны с таблицами базы данных. Hibernate также требует набора параметров конфигурации, связанных с базой данных и других связанных параметров. Вся такая информация обычно предоставляется в виде стандартного файла свойств Java с именем **hibernate.properties** или в виде файла XML с именем **hibernate.cfg.xml**.

Я рассмотрю файл **hibernate.cfg.xml** в формате XML, чтобы указать необходимые свойства Hibernate в моих примерах. Большинство свойств принимают значения по умолчанию, и нет необходимости указывать их в файле свойств, если это действительно не требуется. Этот файл хранится в корневом каталоге пути к классу вашего приложения.

Ниже приведен список важных свойств, вам необходимо будет настроить для баз данных в автономной ситуации:

* **hibernate.dialect**

Это свойство заставляет Hibernate генерировать соответствующий SQL для выбранной базы данных.

* **hibernate.connection.driver\_class**

Класс драйвера JDBC.

* **hibernate.connection.url**

URL JDBC для экземпляра базы данных.

* **hibernate.connection.username**

Имя пользователя базы данных.

* **hibernate.connection.password**

Пароль базы данных.

* **hibernate.connection.pool\_size**

Ограничивает количество соединений, ожидающих в пуле соединений базы данных Hibernate.

* **hibernate.connection.autocommit**

Позволяет использовать режим автоматической фиксации для соединения JDBC.

Если вы используете базу данных вместе с сервером приложений и JNDI, вам придется настроить следующие свойства:

* **hibernate.connection.datasource**

Имя JNDI, определенное в контексте сервера приложений, которое вы используете для приложения.

* **hibernate.jndi.class**

Класс InitialContext для JNDI.

* **hibernate.jndi. <JNDIpropertyname>**

Передаетлюбое свойство JNDI, которое вам нравится, в InitialContext JNDI.

* **hibernate.jndi.url**

Предоставляет URL для JNDI.

* **hibernate.connection.username**

Имя пользователя базы данных.

* **hibernate.connection.password**

Пароль базы данных.

## 5. Аннотация @Autowired?

Аннотация @Autowired отмечает конструктор, поле или метод как требующий автозаполнения инъекцией зависимости Spring.

Чтобы аннотация @Autowire присвоила переменной значения соответствующего bean-а, необходимо чтобы этот bean либо был объявлен в xml конфигурации приложения, либо существовал класс с соответствующей инъекцией управления.

Т.о., Фреймворк Spring сам найдет нужный bean и подставит его значение в свойство, которое отмечено аннотацией @Autowired.

Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания bean'а и перед тем, как любой из методов класса будет вызван.

@Autowired

private FieldService propertyService;

«Традиционное» использование аннотации ‒ с сеттером.

@Autowired

public void setSetterService(SetterService setterService) {

this.setterService = setterService;

}

Только один конструктор может выполнять эту аннотацию. Этот конструктор может быть любого типа (private, protected), а не только public.

Аннотация @Autowire может быть использована в методе с любым именем и с любым количеством принимаемых параметров. В этом случае Spring попытается присвоить каждому аргументу значение соответствующих bean-а. Метод также не обязан быть public.

С помощью аннотации @Qualifier можно отметить конкретного «кандидата» для автозаполнения, если их несколько (например, есть два bean-а).

## 6. Что такое JPQL (Java Persistence query Language)?

– это платформенно-независимый объектно-ориентированный язык запросов, являющийся частью спецификации Java Persistence API (JPA).

JPQL используется для написания запросов к сущностям, хранящимся в реляционной базе данных. JPQL во многом похож на SQL, но в отличие от последнего, оперирует запросами, составленными по отношению к сущностям JPA, в отличие от прямых запросов к таблицам базы данных.

В дополнение к получению объектов (SELECT-запросы), JPQL поддерживает запросы, основанные на операторах UPDATE и DELETE.

## 7. Интерфейс JpaRepository?

**–** это интерфейс фреймворка Spring Data предоставляющий набор стандартных методов JPA для работы с БД.

На основе этого интерфейса Spring Data предоставит реализации с методами, которые используются в Entity Manager.

Правила:

1 – Имя репозитория должно начинаться с имени сущности **Name**Reposytory.

**2** – Второй Generic должен быть оберточным типом того типа, которым есть ID нашей сущности.

**3** – Первый Generic должен быть объектом нашей сущности, для которой мы создали Repository, это указывает на то, что Spring Data должен предоставить реализацию методов для работы с этой сущностью.

**4** – Мы должны унаследовать свой интерфейс от JpaRepository, иначе Spring Data не предоставит реализацию для нашего репозитория.

## 8. Аннотация @Id, @GeneratedValue?

@Id

С помощью аннотации @Id мы указываем первичный ключ (Primary Key) данного класса.

@GeneratedValue

Эта аннотация используется вместе с аннотацией @Id и определяет такие паметры, как strategy и generator.

По умолчанию аннотация @Id автоматически определяет наиболее подходящую стратегию генерации первичного ключа для использования, но можно переопределить это, применив аннотацию @GeneratedValue, которая принимает два параметра strategy и generator.

## 9. Как в проекте Maven подключить JPA?

Нажимаем на название проекта правой кнопкой и выбираем Add Framework support. Отмечаем галочкой Spring, JavaEE Persistence и Hibernate. Если зависимости подгрузились, то будет предложено выбрать библиотеку maven. После этого в проект добавятся необходимые xml — META-INF\persistence.xml и sping-config.xml.

## 10. Методы JpaRepository?

* [**T**](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html) **getReferenceById(**[**ID**](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html) **id)**

- Возвращает ссылку на объект с заданным идентификатором. В зависимости от того, как реализован провайдер персистентности JPA, очень вероятно, что он всегда будет возвращать экземпляр и выдавать EntityNotFoundException при первом доступе. Некоторые из них будут немедленно отклонять недействительные идентификаторы.

* **<S extends** [**T**](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)**>** [**List**](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html)**<S> findAll(**[**Example**](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Example.html)**<S> example)**

- Возвращает все объекты, соответствующие заданному [Example](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Example.html). Если совпадений не найдено, возвращается пустое [Iterable](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterable.html) значение.

* **<S extends** [**T**](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)**>** [**List**](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html)**<S> findAll(**[**Example**](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Example.html)**<S> example,** [**Sort**](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Sort.html) **sort)**

- Возвращает все объекты, соответствующие заданному [Example](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Example.html), применяя указанный Sort. Если совпадений не найдено, возвращается пустое [Iterable](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterable.html) значение.

* **void deleteAllInBatch(**[**Iterable**](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterable.html)**<**[**T**](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)**> entities)**

- Удаляет указанные объекты в пакете, что означает, что будет создан один запрос. Этот тип операции оставляет кеш первого уровня JPA и базу данных несинхронизированными.

* **void deleteAllByIdInBatch(**[**Iterable**](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterable.html)**<**[**ID**](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)**> ids)**

- Удаляет объекты, идентифицированные заданными идентификаторами, с помощью одного запроса. Этот тип операции оставляет кеш первого уровня JPA и базу данных несинхронизированными.

* **void deleteAllInBatch()**

- Удаляет все объекты в пакетном вызове.

* **void flush()**

- Сбрасывает все ожидающие изменения в базе данных.

* **<S extends** [**T**](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)**> S saveAndFlush(S entity)**

- Сохраняет объект и мгновенно удаляет изменения.

* **<S extends** [**T**](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)**>** [**List**](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html)**<S> saveAllAndFlush(**[**Iterable**](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterable.html)**<S> entities)**

- Сохраняет все объекты и мгновенно сбрасывает изменения.