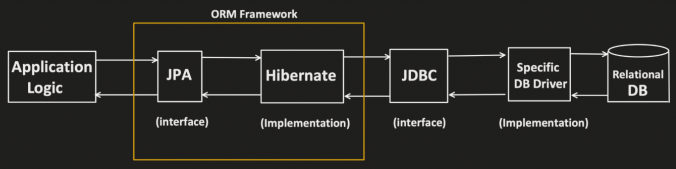
JPA ARCHITECTURE (PART2)



ORM (Object-Relational Mapping)

- Act as a bridge between Java Object and Database tables.
- Unlike JDBC, where we have to work with SQL, with this, we can interact with database using Java Objects.

Lets first see, 1 happy flow first, before deep diving into JPA

pom.xml

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

application.properties

```
#database connection properties
spring.datasource.url=jdbc:h2:mem:userDB
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
```



```
@RestController
@RequestMapping(value = "/api/")
public class UserController {

    @Autowired
    UserDetailsService userDetailsService;

    @GetMapping(path = "/test-jpa")
    public List<UserDetails> getUser() {
        UserDetails userDetails = new UserDetails( name: "xyx",
                email: "xyz@conceptandcoding.com");
        userDetailsService.saveUser(userDetails);
        return userDetailsService.getAllUsers();
    }
}
```

```
@Service
public class UserDetailsService {

    @Autowired
    private UserDetailsRepository userDetailsRepository;

    public void saveUser(UserDetails user) {
        userDetailsRepository.save(user);
    }

    public List<UserDetails> getAllUsers() {
        return userDetailsRepository.findAll();
    }
}
```

```
@Repository
public interface UserDetailsRepository extends
        JpaRepository<UserDetails, Long> {
}
```

```
@Entity
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    // Constructors
    public UserDetails() {
    }

    public UserDetails(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getters and setters

}
```

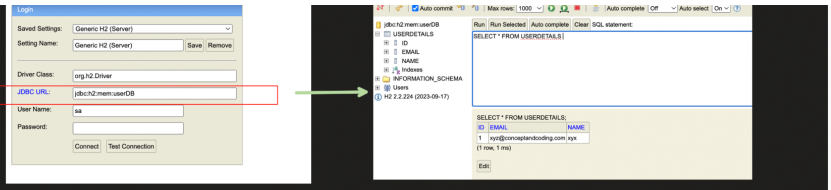
Output:

```
[
    {
        "id": 1,
        "name": "xyx",
        "email": "xyz@conceptandcoding.com",
        "age": 28
    }
]
```

To enable the console, we can add, below two properties in "application.properties"

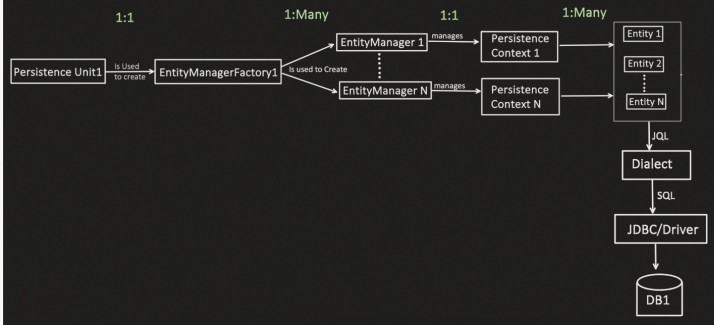
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

http://localhost:8080/h2-console



Looks very simple, but what's happening inside?

JPA Architecture/Components involved:



pom.xml

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

1. Persistence Unit

- Logical grouping of Entity classes which share same configurations.
- Configuration details like:
  - Database connection properties
  - JPA Provider (hibernate etc.) etc.

Persistence.xml

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.1">

    <persistence-unit name="persistentUnitNameHere" transaction-type="RESOURCE_LOCAL">

        <!-- Logical grouping of Entity Classes which all shares same configurations -->
        <class>com.conceptandcoding.entity.SampleEntity1</class>
        <class>com.conceptandcoding.entity.SampleEntity2</class>

        <!-- which JPA provider we want to use, hibernate or OpenJPA or EclipseLink etc... -->
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>


        <!-- DB Connection Properties -->
        <properties>

            <!-- specific provider dialect -->
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect" />

            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:userDB" />
            <property name="javax.persistence.jdbc.user" value="sa" />
            <property name="javax.persistence.jdbc.password" value="" />
            <!-- more properties here -->
        </properties>

    </persistence-unit>

</persistence>
```

application.properties

```
#database connection properties
spring.datasource.url=jdbc:h2:mem:userDB
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# Define packages to scan
# (its optional, spring boot looks for all @Entity, but still if we want specific package lookup
spring.jpa.packages-to-scan=com.conceptandcoding.entity

#define provider
#(its optional, spring boot automatically picks based on provider)
spring.jpa.properties.javax.persistence.provider=org.hibernate.jpa.HibernatePersistenceProvider
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

#transaction type, Default is RESOURCE_LOCAL only
spring.jpa.properties.javax.persistence.transactionType=RESOURCE_LOCAL
```

2. EntityManagerFactory

- Using Persistence Unit configuration, EntityManagerFactory object get created during application startup.
- If any property is not provided, default one is picked and set.
- 1 EntityManagerFactory for 1 Persistence Unit.
- This class act as a Factory to create an object of EntityManager.

LocalContainerEntityManagerFactoryBean.java

```
@Override
protected EntityManagerFactory createNativeEntityManagerFactory() throws PersistenceException {
    Assert.state( expression: this.persistenceUnitInfo != null,  message: "PersistenceUnitInfo not initialized");

    PersistenceProvider provider = getPersistenceProvider();
    if (provider == null) {
        String providerClassName = this.persistenceUnitInfo.getPersistenceProviderClassName();
        if (providerClassName == null) {
            throw new IllegalArgumentException(
                    "No PersistenceProvider specified in EntityManagerFactory configuration, " +
                            "and chosen PersistenceUnitInfo does not specify a provider class name either");
        }
        Class<?> providerClass = ClassUtils.resolveClassName(providerClassName, getBeanClassLoader());
        provider = (PersistenceProvider) BeanUtils.instantiateClass(providerClass);
    }

    if (logger.isDebugEnabled()) {
        logger.debug("Building JPA container EntityManagerFactory for persistence unit '" +
                this.persistenceUnitInfo.getPersistenceUnitName() + "'");
    }
    EntityManagerFactory emf =
            provider.createContainerEntityManagerFactory(this.persistenceUnitInfo, getJpaPropertyMap());
    postProcessEntityManagerFactory(emf, this.persistenceUnitInfo);

    return emf;
}
```

What if we want to manually create and object of EntityManagerFactory?

```
@Configuration
public class AppConfig {

    @Bean
    public DataSource dataSource() {
        HikariDataSource dataSource = new HikariDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setJdbcUrl("jdbc:h2:mem:userDB");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

    @Bean
    public JpaVendorAdapter jpaVendorAdapter() {
        HibernateJpaVendorAdapter adapter = new HibernateJpaVendorAdapter();
        adapter.setGenerateDdl(true);
        adapter.setDatabasePlatform("org.hibernate.dialect.H2Dialect");
        return adapter;
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory(
            DataSource dataSource,
            JpaVendorAdapter jpaVendorAdapter) {
        LocalContainerEntityManagerFactoryBean emf1 = new LocalContainerEntityManagerFactoryBean();
        emf1.setDataSource(dataSource);
        emf1.setJpaVendorAdapter(jpaVendorAdapter);
        emf1.setPackagesToScan("com.conceptandcoding.learningspringboot.jpa");
        emf1.setPersistenceUnitName("uniqueFactoryName"); //  unique name for our EntityManagerFactory
        return emf1;
    }
}
```

3. Transaction Manager association with EntityManagerFactory

During persistence unit, we have specified the value for "transaction-type" value either:
- RESOURCE_LOCAL (default)
- JTA (Java Transaction API)

Transaction Manager could be of 2 type:

- Manager, which managing transaction For 1 DB.

- Manager, which managing transaction which can span across multiple DB. That's also possible using JTA.

During application startup, after EntityManagerFactory object created, based on RESOURCE_LOCAL or JTA, transaction manager object get created.

UseCase1: Transaction Manager for managing txn for 1 DB

application.properties

```
#database connection properties
spring.datasource.url=jdbc:h2:mem:user@@
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# Define packages to scan
# (its optional, spring boot looks for all @Entity, but still if we want specific package lookup only)
spring.jpa.packages-to-scan=com.conceptandcoding.entity

#define provider
#(its optional, spring boot automatically picks based on provider)
spring.jpa.properties.javax.persistence.provider=org.hibernate.jpa.HibernatePersistenceProvider
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

#transaction type, Default is RESOURCE_LOCAL only
spring.jpa.properties.javax.persistence.transactionType=RESOURCE_LOCAL
```

JpaTransactionManager.java
(implementation of PlatformTransactionManager)

```
@Override
public void setBeanFactory(BeanFactory beanFactory) throws BeansException {
    if (getEntityManagerFactory() == null) {
        if (!(beanFactory instanceof ListableBeanFactory lbf)) {
            throw new IllegalStateException("Cannot retrieve EntityManagerFactory by persistence unit name " +
                    "in a non-listable BeanFactory: " + beanFactory);
        }
        setEntityManagerFactory(EntityManagerFactoryUtils.findEntityManagerFactory(lbf, getPersistenceUnitName()));
    }
}
```

```
beanType = {Class@10098} "class org.springframework.orm.jpa.JpaTransactionManager" ... Navigate
    cachedConstructor = null
    name = "org.springframework.orm.jpa.JpaTransactionManager"
    module = {Module@13639} "unnamed module @71623278"
    classLoader = {ClassLoaders$AppClassLoader@10105}
    classData = null
    packageName = "org.springframework.orm.jpa"
    componentType = null
```

We can create it manually too:

```java
@Configuration
public class AppConfig {

    @Bean
    public DataSource dataSource() {
        HikariDataSource dataSource = new HikariDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setJdbcUrl("jdbc:h2:mem:userDB");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

    @Bean
    public JpaVendorAdapter jpaVendorAdapter() {
        HibernateJpaVendorAdapter adapter = new HibernateJpaVendorAdapter();
        adapter.setGenerateDdl(true);
        adapter.setDatabasePlatform("org.hibernate.dialect.H2Dialect");
        return adapter;
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory(
            DataSource dataSource,
            JpaVendorAdapter jpaVendorAdapter) {
        LocalContainerEntityManagerFactoryBean emf1 = new LocalContainerEntityManagerFactoryBean();
        emf1.setDataSource(dataSource);
        emf1.setJpaVendorAdapter(jpaVendorAdapter);
        emf1.setPackagesToScan("com.conceptandcoding.learningspringboot.jpa");
        emf1.setPersistenceUnitName("uniqueFactoryName"); //  unique name for our EntityManagerFactory
        return emf1;
    }

    @Bean
    public JpaTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
        return new JpaTransactionManager(entityManagerFactory);
    }
}
```

UseCase2: Transaction Manager for creating txn which can span across multiple DB

I will create a separate video, in which I will explain, how we can handle txn which can span across multiple database.
As it required some time for explaining JTA related nuisance and keyworks like Atomikos, XADataSource and how it uses 2PC to orchestrate txn etc.. which is out of the context for todays topic.

But at this point, we can understand that, TRANSACTION MANAGER is created based on what "transaction-type" we provided in the persistence unit (application.properties) file.

---------- Above all steps happened during application startup, below steps happen when API gets invoked ----------

4. EntityManager and Persistence Context

EntityManager :
- Its an interface in JPA that provides methods to perform CRUD operations on entities.
  - **persist()** (for saving)
  - **merge()** (for updating)
  - **find()** (for fetching)
  - **remove()** (for deleting)
  - **createQuery()** (for executing JPQL queries)

- EntityManager interface methods are implemented by JPA Vendors like Hibernate etc.
- EntityManagerFactory helps to create an Object of EntityManager.

PersistenceContext:

- Consider its a first level cache.
- For each EntityManager, PersistenceContext object is created, which hold list of Entities its working on.
- Also manage the life cycle of entity.

Entity Manager Insert/Update/Delete operations are Transaction bounded. Means, it first check if 'Transaction' is open, if not, it will throw exception.
But not all (READ operations are not Transaction bounded)

```java
@Service
public class UserDetailsService {

    @Autowired
    private UserDetailsRepository userDetailsRepository;

    public void saveUser(UserDetails user) {
        userDetailsRepository.save(user);
    }

    public List<UserDetails> getAllUsers() {
        return userDetailsRepository.findAll();
    }
}
```

```java
@Repository
public interface UserDetailsRepository extends
        JpaRepository<UserDetails, Long> {
}
```

Internally JpaRepository all Insert, Update, Delete methods are annotated with @Transactional, so even if we do not write, spring framework takes care of it.

```java
@Transactional
public <S extends T> S save(S entity) {
    Assert.notNull(entity, message: "Entity must not be null");
    if (this.entityInformation.isNew(entity)) {
        this.entityManager.persist(entity);
        return entity;
    } else {
        return this.entityManager.merge(entity);
    }
}
```

What if, I try to directly call EntityManager persist method, instead of spring framework.

```java
@Service
public class UserDetailsService {

    @PersistenceContext
    EntityManager entityManager;

    public void saveUser(UserDetails user) {
        entityManager.persist(user);
    }
}
```

```
jakarta.persistence.TransactionRequiredException Create breakpoint : No EntityManager with actual transaction available for current thread
    at org.springframework.orm.jpa.SharedEntityManagerCreator$SharedEntityManagerInvocationHandler.invoke(SharedEntityManagerCreator.ja
    at jdk.proxy2/jdk.proxy2.$Proxy108.persist(Unknown Source) ~[na:na]
    at com.conceptandcoding.learningspringboot.jpa.service.UserDetailsService.saveUser(UserDetailsService.java:23) ~[classes/:na]
    at com.conceptandcoding.learningspringboot.UserController.getUser(UserController.java:22) ~[classes/:na] <4 internal lines>
```

If I add @Transactional, it works fine now.

```java
@Service
public class UserDetailsService {

    @PersistenceContext
    EntityManager entityManager;

    @Transactional
    public void saveUser(UserDetails user) {
        entityManager.persist(user);
    }
}
```

```
    {
        "id": 1,
        "name": "xyx",
        "email": "xyz@conceptandcoding.com"
    }
]
```


Life cycle of Entity in persitenceContext: