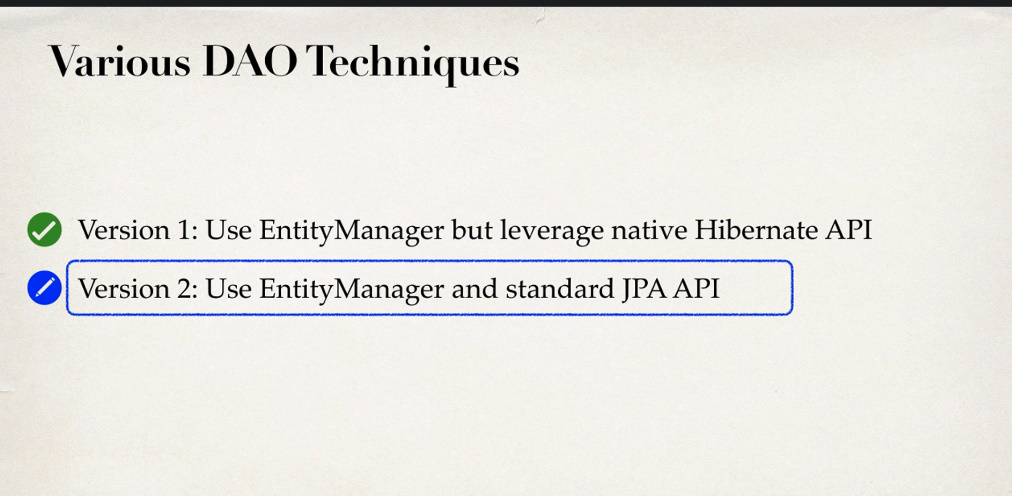
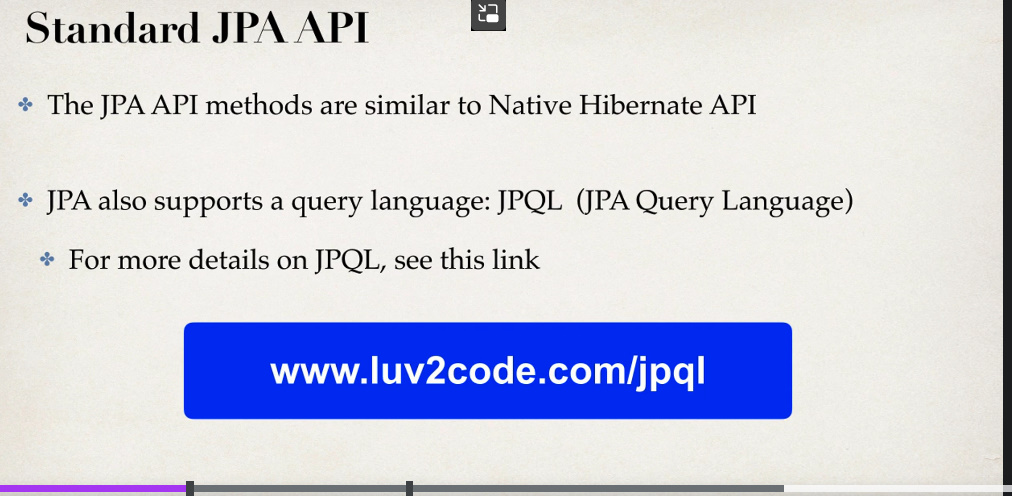
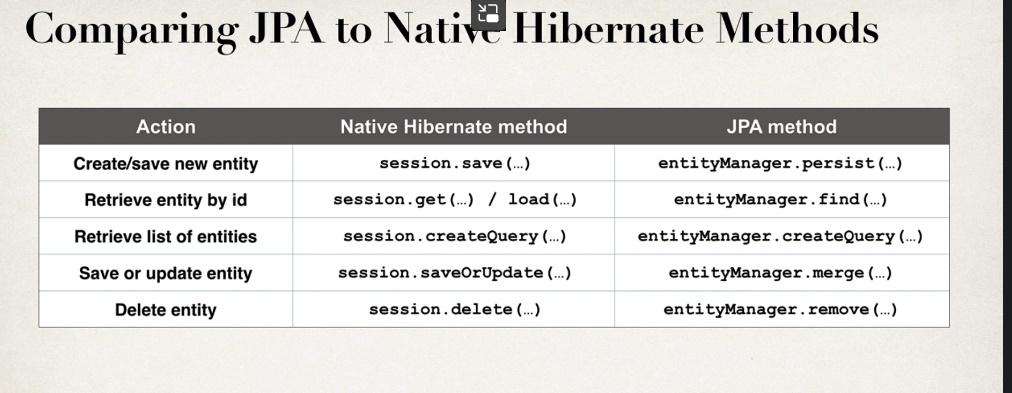
**metoda 2 JPA API**

**ATENTIE! Spring 6 deci Spring BOOT 3 nu folosesc javax! Ci Jakarta!si minim Java17**



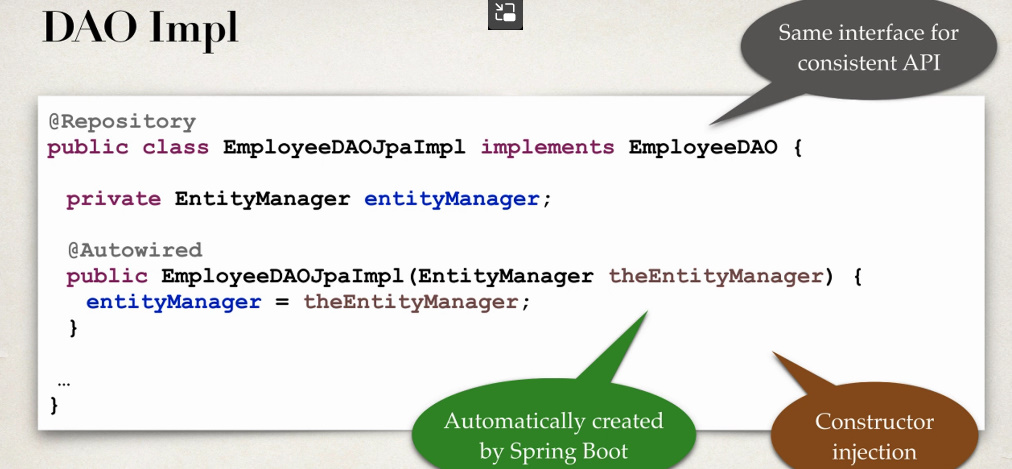


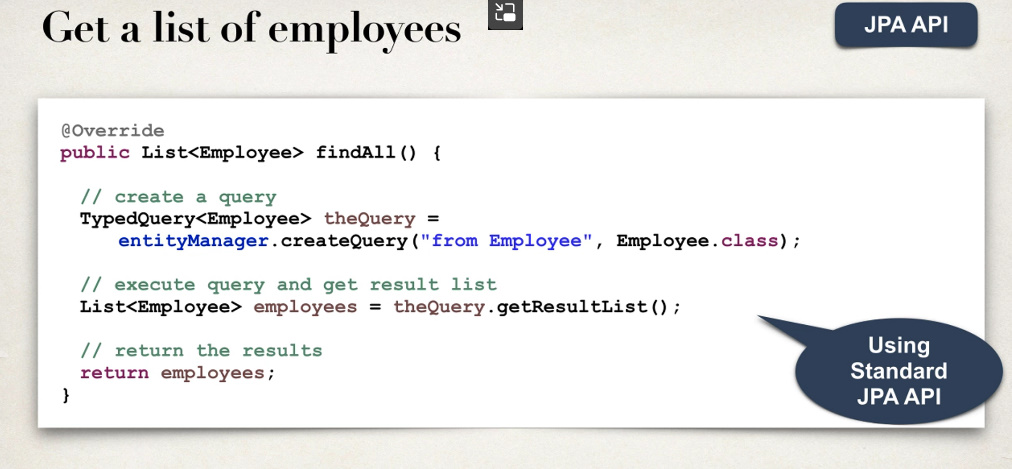
**Metode**



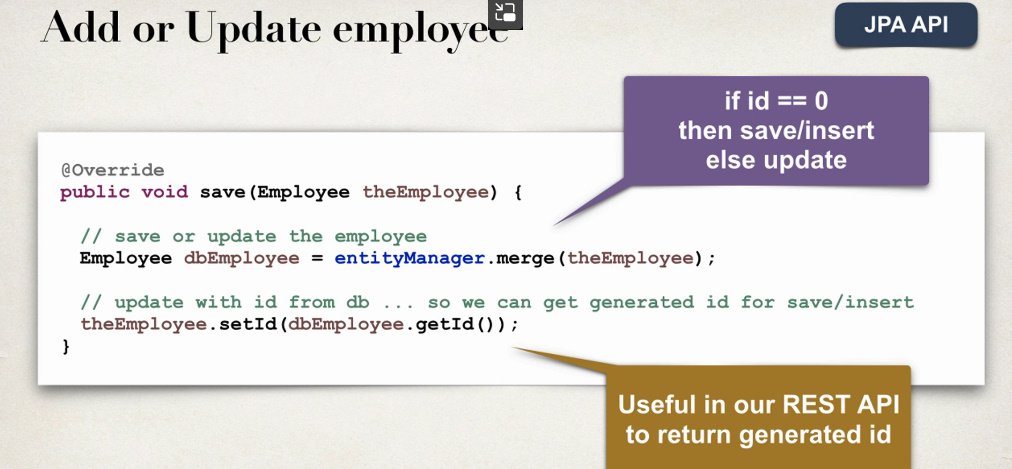
Session implementeaza EntityManager, deci are si toate metodele lui EntityManager, insa EntityManager logic ca nu le are pe ale lui Session

**Creare**







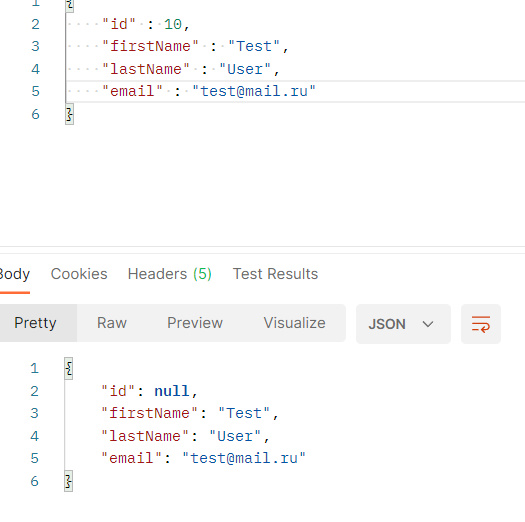


Mare grija!JPA ia referinta la obiect, seteaza un ID, dar nu in referinta data, ci va crea un nou obiect pe baza la cel oferit de noi si dupa merge() il va returna cu tot cu id setat.

Deci, daca facem asa:

public void save(Employee employee) {  
 entityManager.merge(employee);  
}

se va intampla asta





Nu se va salva ID in obiectul trimis din service, ci doar se creaza un nou obiect identic cu acesta, dar cu un id SETAT. Asta e pentru securitate.



Asa cum EntityManager e din JPA si Hibernate implementeaza EntityManager in Session,Session are metodele save, saveOrUpdate etc. dar sunt depreciate,si plus cele ale lui EntityManager si mai bine le folosim pe cele din tabelul de sus pentru EntityManager chiar si cu Hibernate.

**Atentie, nu putem avea 2 implemetari pentru aceeasi interfata pentru a crea DAO!!!**

**@Primary**

* Daca pentru o interfata pentru DAO, de ex EmployeeDAO, avem 2 implementari, ca EmployeeDAOJPA si EmployeeDAOHibernate, atunci Spring nu va sti pe care dintre ele sa le foloseasca cand va injecta in Service, caci ambele implementeaza EmployeeDAO. De aceea, folosim dupa @Repository anotatia @Primary la una dintre ele pentru a spune care Repository(bean) are prioritate.

@Repository  
@Primary  
public class EmployeeJPAImplem implements EmployeeDAO{

**unwrap**

* Spre deosebire de SessionFactory, in Hibernate, JPA nu foloseste asa ceva, ci foloseste EntityManager.
* Partea buna la EntityManager e ca nu trebuie mereu sa returnam sesiunea pentru a crea query, pentru a obtine obiecte din DB etc.

De ex putem face chiar asa:

public List<Employee> findAll() {  
 Query query = entityManager.createQuery("from Employee",Employee.class);  
  
 List<Employee> employees = query.getResultList();  
  
 return employees;  
}

Nu e neaparat necesar sa punem si:

Session session = entityManager.unwrap(Session.class);

* unwrap ne ofera un obiect de tipul specificat pentru a putea obtine control asupra lui.

**Crearea unei noi DataSource din alt .properties file**

Putem sa cream un alt fisier .properties, cum ar fi DB.properties si sa specificam in el toate datele necesare pentru conexiunea la baza de date. In asa caz, vom face asa:

spring.datasource.url = jdbc:mysql://localhost:3306/employee\_directory?useSSL=false&serverTimezone=UTC  
spring.datasource.username = testuser  
spring.datasource.password = Frb2eshox!  
spring.datasource.driver = com.mysql.jdbc.Driver  
spring.jpa.properties.hibernate.show\_sql=true  
  
spring.jpa.properties.hibernate.c3p0.timeout = 3000  
spring.jpa.properties.hibernate.c3p0.idle\_test\_period = 2000  
spring.jpa.properties.hibernate.c3p0.min\_size = 5  
spring.jpa.properties.hibernate.c3p0.initial\_size = 5  
spring.jpa.properties.hibernate.c3p0.max\_size = 20

@SpringBootApplication  
@PropertySource("classpath:DB.properties")  
public class EmployeeApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(EmployeeApplication.class, args);  
 }  
  
 @Autowired  
 private Environment environment;  
  
 @Bean  
 public DataSource dataSource(){  
 return DataSourceBuilder.*create*().username(environment.getProperty("spring.datasource.username"))  
 .password(environment.getProperty("spring.datasource.password")).url(environment.getProperty  
 ("spring.datasource.url"))  
 .driverClassName(environment.getProperty("spring.datasource.driver")).build();  
 }  
  
 @Bean  
 @Primary  
 public ComboPooledDataSource comboPooledDataSource() throws PropertyVetoException, SQLException {  
 ComboPooledDataSource comboPooledDataSource = new ComboPooledDataSource();  
 comboPooledDataSource.setUser(environment.getProperty("spring.datasource.username"));  
 comboPooledDataSource.setPassword(environment.getProperty("spring.datasource.password"));  
 comboPooledDataSource.setJdbcUrl(environment.getProperty  
 ("spring.datasource.url"));  
 comboPooledDataSource.setDriverClass(environment.getProperty("spring.datasource.driver"));  
  
 comboPooledDataSource.setInitialPoolSize(Integer.*parseInt*(environment.getProperty("spring.jpa.properties.hibernate.c3p0.initial\_size")));  
 comboPooledDataSource.setMinPoolSize(Integer.*parseInt*(environment.getProperty("spring.jpa.properties.hibernate.c3p0.min\_size")));  
 comboPooledDataSource.setMaxPoolSize(Integer.*parseInt*(environment.getProperty("spring.jpa.properties.hibernate.c3p0.max\_size")));  
 comboPooledDataSource.setMaxIdleTime(Integer.*parseInt*(environment.getProperty("spring.jpa.properties.hibernate.c3p0.idle\_test\_period")));  
 comboPooledDataSource.setLoginTimeout(Integer.*parseInt*(environment.getProperty("spring.jpa.properties.hibernate.c3p0.timeout")));  
  
 return comboPooledDataSource;  
 }  
  
}

Vedem ca avem 2 metode de a seta Data Source.

Prima e buna daca pur si simplu facem o Data Source simpla, fara properties.

A doua daca vrem sa setam ceva proprietati.