

JPA/Hibernate camp

:docInfo1

Table of Contents

1. O mnie	1
2. Źródła wiedzy	1
3. Hibernate / JPA	1
4. Co to jest w ogóle jest :) ?	1
5. Trendy :)	2
5.1. Nosql vs Hibernate	2
6. Architektura	3
6.1. Zasada działania	3
7. Moduły	6
8. Minimalna konfiguracja projektu	7
9. Hibernate	7
10. Terminy związane z Hibernate	8
11. Symbioza	8
11.1. Lombok	9
11.2. JAXB	9
11.3. JSON	9
11.4. Bean Validation	10
12. Nowości w JPA 2.1	10
13. Connection pooling	12
13.1. Przykład zestawienia puli połączeń dla Springa	14
14. Encja - Trwała klasa	14
15. Cykl życia	14
16. Session Factory ~ EntityManagerFactory	17
17. Session ~ EntityManager (zarządza trwałością obiektów (persistence manager))	18
18. Otwieranie i zamykanie sesji	18
18.1. Ponowne użycie istniejącej już sesji	18
18.2. Otwieranie sesji bezstanowej	19
19. Transaction	19
20. Odświeżanie encji	19
21. Opróżnianie sesji	19
22. Czyszczenie kontekstu (JPA)	20
23. Czy sesja jest zanieczyszczona ?	20
24. Adnotacje	20
24.1. Pola	20
24.2. Metody	20
24.3. Mieszany @Access	20
25. @Entity	21
26. Klucze @Id	21

26.1. Prosty	21
26.2. Ustawianie wartości	21
26.3. Klucz złożony	23
26.4. ElementCollection	24
26.5. AttributeOverride	25
27. @Table	26
27.1. Określenie schematu bazy w persistence.xml	27
27.2. @Index	27
28. @Column	27
29. @Check	29
30. @Transient	29
31. @Basic	30
32. @Embeddable i @Embedded**	30
33. @Enumerated	34
34. @Lob	35
34.1. <code>java.sql.Blob</code>	35
34.2. <code>java.sql.Clob</code>	35
35. Date & time	36
35.1. DATE	36
35.2. TIME	36
35.3. TIMESTAMP.....	37
36. Mapping Java 8 Date/Time Values	37
36.1. DATE	37
36.2. TIME	37
36.3. TIMESTAMP.....	37
37. AttributeConverters	37
38. @Formula	38
39. @SecondaryTable	38
40. @AttributeOverride	39
41. @Version - blokowanie optymistyczne	39
42. @OrderColumn	39
43. @ForeignKey	40
44. @Type (Hibernate only).....	40
45. @ElementCollection** - dla typów prostych lub klas osadzonych	40
46. @OrderBy	41
47. @JoinTable	41
48. Relacje	41
48.1. FetchType	41
49. @Dynamic	44
49.1. @DynamicInsert (false/true).....	44
49.2. @DynamicUpdate (false/true).....	44

50. @Immutable	45
51. Callbacks	45
51.1. @PrePersist	45
51.2. @PreRemove	45
51.3. @PostPersist	46
51.4. @PostRemove	46
51.5. @PreUpdate	46
51.6. @PostUpdate	46
51.7. @PostLoad	46
52. EventListener	46
53. Query, Criteria	47
54. Tożsamość obiektu : Equals & hashCode	47
55. Identyczność obiektów	48
55.1. Identyczność obiektów	48
55.2. Równość obiektów	48
55.3. Tożsamość bazodanowa	48
56. Klucz główny	48
57. Relacje jedno i dwukierunkowe	49
57.1. Charakterystyka	49
58. @OneToOne	49
58.1. Dwukierunkowa	50
58.2. Z dodatkową tabelą łączącą relacje	51
58.3. Ze wspólnym kluczem głównym	51
59. @OneToMany	51
60. @ManyToOne	52
60.1. @ManyToOne z tabelą łączącą	52
61. @ManyToMany	53
61.1. Dwukierunkowa	54
61.2. Dodatkowa kolumna w tabeli złączenia	54
62. Kaskadowość	56
63. Usuwanie sierot - orphanRemoval	56
64. Pobieranie encji	57
65. SAVE/PERSIST	58
66. MERGE vs UPDATE (stackoverflow)	58
67. Trwałość przechodnia	59
68. Trwałość przez osiągalność	59
69. Trwałość kaskadowa	59
70. Zapytania	60
71. Kwerandy	60
72. Możliwe formy zapytań	60
72.1. zapytania HQL, JPQL	60

72.2. Zapytania dynamiczne	60
72.3. Możliwe typy wników kwerend:	60
72.4. Criteria API	62
72.5. native SQL	62
72.6. Hibernate Criteria	62
73. SELECT	62
74. Projekcja / Projection	62
74.1. SQL result-mapping	63
74.2. Projekcja z użyciem konstruktora	63
75. Restriction / where – zawężamy	63
75.1. Like	63
76. Zwiększenie wydajności poprzez agregacje po stronie bazy	64
77. Natywna kwerenda SQL	64
77.1. Nativne / NATIVE	65
78. HQL kwerendy	65
78.1. JPA natywne kwerendy	65
78.2. Nativne kwerendy z aliasami	66
79. Zapytania nazywane / NamedQuery	66
79.1. Nativne w konfiguracja z JOIN	67
79.2. Zapytania nazywane podejście programistyczne	68
79.3. Tuple	69
80. FROM	69
80.1. Native	70
80.2. Hint	70
80.3. Timestamp	71
80.4. Konkadenacja	71
80.5. Porównania	71
80.6. Like	73
81. Kwerenda z wielu podmiotów	73
82. Dynamiczna instancja - przykład	74
83. Dynamiczna mapa - przykład	74
84. Where	75
85. Parametryzacja	75
86. Wstawienie przez kwerende	75
86.1. Dopasowane operacje : Insert, Update , Delete	75
86.2. Przykrywanie operacji na kolekcjach przez adnotacje	76
87. UPDATE	76
87.1. Bulk update	76
88. Delete	77
89. Distinct	77
90. Between	77

91. IS [NOT] EMPTY	78
92. [NOT] MEMBER [OF]	78
93. Podzapytania	79
94. IN	79
95. Operacje na kolekcjach	79
96. Sortowanie	79
97. Agregacje	80
97.1. AVG	80
97.2. COUNT	80
97.3. MAX	81
97.4. MIN	81
97.5. SUM	81
98. GROUP BY	81
99. HAVING	82
100. Stronicowanie	82
101. Pobieranie pojedyńczego wyniku	83
102. JOIN	83
102.1. Join niejawny	84
102.2. Wielokrotny Join	84
103. JOIN LEFT	84
104. JOIN FETCH	85
105. Wyrażenie IN	85
106. JPQL wspieranie standardów	85
106.1. CONCAT - łącznie dwóch lub większej ilości stringów	85
106.2. SUBSTRING - wycinanie części stringa z danego ciągu znaków	85
106.3. UPPER - zamiana na duże litery	86
106.4. LOWER - zamiana na małe litery	86
106.5. TRIM - usuwanie białych znaków	86
106.6. LENGTH - obliczanie długości ciągu znaków	86
106.7. ABS - obliczanie wartości absolutnej	87
106.8. MOD - obliczanie reszty z dzielenia	87
106.9. SQRT - pierwiastek	87
106.10. CURRENT_DATE - bieżąca data	87
106.11. CURRENT_TIME - bieżący czas	87
106.12. CURRENT_TIMESTAMP - bieżąca data i czas z milisek	88
107. HQL functions	88
107.1. CAST - rzutowanie	88
107.2. EXTRACT	88
107.3. YEAR	88
107.4. MONTH	89
108. Typ encji	89

109. Case	89
110. Użycie konstruktora	90
111. Usuwanie obiektów z bazy	90
112. Aktualizacja	90
113. Merge	91
114. Odświeżanie encji (Refreshing Entities)	91
115. O mnie	91
116. Geneza JPA	92
117. Dostawcy trwałości	92
118. Utrwalanie	92
119. Użycie EntityManager'a [JPA]	92
119.1. Zależności	93
119.2. Schemat zależności	93
119.3. EntityManagerFactory(JPA) = SessionFactory(Hibernate)	93
119.4. EntityManager	94
119.5. EntityManager Zdalnie	97
119.6. Metody	98
119.7. persistence.xml	99
120. Persistence Unit	100
120.1. Praca w wieloma jednostkami trwałości	100
121. Persistence Context	101
122. Dostęp do Hibernate API z poziomu JPA	102
123. MetaModel	102
123.1. Generacja	102
124. Autoreferencja	104
125. Dziedziczenie / Polimorfizm	105
126. MappedSuperclass Table per concrete class with implicit polymorphism	105
127. Tabela na każdą hierarchię klas (Table per class hierarchy / Single-Table Strategy)	106
127.1. Discriminator formula	108
128. Tabla na każdą podklasę (Table per subclass/joined strategy)	110
128.1. @PrimaryKeyJoinColumn	110
129. Tabela na klasę konkretną (Table per concrete class)	111
130. Criteria	112
131. CRITERIA API	112
131.1. CriteriaBuilder	112
131.2. CriteriaQuery<T>	112
131.3. Root<T>	113
131.4. TypedQuery<T>	113
132. Wyrażenia / Expression	113
133. Typy / Types	113
134. Kwerendy / guery	113

135. Atrybuty / Attributes.....	113
136. MetaModel	113
137. SELECT.....	114
137.1. Bez potrzeby rzutowania.....	114
137.2. Parametryzacja	114
137.3. Wyrażenie / Expression	115
137.4. Pojedyńcze wartości	115
137.5. Wielokrotne wartości.....	115
137.6. Multiselect.....	116
137.7. Aliasy	116
137.8. Zapytania dynamiczne	116
137.9. Wrapper.....	117
137.10. Tuple.....	118
138. JOIN	119
139. FETCH.....	120
140. Użycie parametrów.....	120
141. GroupBy i Tuple.....	121
142. Predykaty	122
143. Skalary.....	122
144. Funkcje	122
145. Agregacje.....	123
146. Podzapytania	123
147. Logowanie zdarzeń.....	124
148. Hibernate / JPA	124
149. Architektura	125
150. First level cache	125
151. Second level cache	126
151.1. Włączenie	126
151.2. javax.persistence.sharedCache.mode.....	126
151.3. Retrieval Mode -określa jak dane mają być czytane z bufora (odwołania do EntityManagera)	127
151.4. Store Mode - określa jak dane mają być składowane w cache	127
151.5. Dostawcy	127
151.6. Strategie	128
152. Cache dla kwerend.....	129
152.1. Konfiguracja	129
153. Collection cache	131
154. Query level cache	131
154.1. aktywacja.....	131
154.2. JPA	131
154.3. Hibernate native API	132

154.4. Używając JPA	132
155. Natywny Hibernate API.....	132
156. Statystyki.....	132
157. Ehcache	133
157.1. RegionFactory.....	133
158. Przykład użycia	133
159. Zapytania natywne.....	133
160. Walidacja	134
161. Walidacja	134
161.1. Zależności	134
161.2. Validation-mode	135
161.3. Adnotacje.....	135
161.4. Własny validator	136
162. Testy	138
163. Wydajność.....	140
164. Wydajność (Performance)	140
165. Sposoby pobierania rekordów / Fetching	140
165.1. JOIN.....	140
165.2. SELECT	140
165.3. SUBSELECT	140
165.4. BATCH	140
166. Lazy	140
166.1. Sposoby inicjalizacji :	141
166.2. less lazy loading.....	142
166.3. Batching for Performance.....	142
166.4. OpenInView	143
166.5. FETCH JOIN	144
167. FETCH.....	145
167.1. Eager.....	145
167.2. Lazy.....	145
167.3. Fetch Join.....	145
167.4. Batch	146
167.5. Extra lazy.....	146
167.6. Określanie głębi wczytywanych obiektów	146
168. Kartezjan problem	147
169. Kroki optymalizacji.....	148
169.1. Dziennik zdarzeń.....	148
169.2. Analiza przypadków użycia	148
169.3. Dostrajanie parametrów	148
169.4. Gradle	149
169.5. Maven	149

170. readOnly	150
171. Inne możliwe problemy i wskazówki:	151
172. Rady	152
173. Paginacja	153
174. Obsługa wyjątków	153
175. Rozwiązywanie problemów	154
175.1. Problemy z asocjacjami dwukierunkowymi	154
175.2. Kłopoty z pamięcią	154
175.3. Problemy z wydajnością mechanika :	154
175.4. @Basic(lazy)	155
175.5. OutOfMemoryException	155
175.6. Ładowanie klas do kontekstu z poziomu SessionFactory	155
175.7. N+1 problem	155
175.8. Jak pokazać parametryzacje zapytań SQL ?	155
176. Dobre praktyki	157
176.1. Top down (dobre dla już istniejącego kodu)	157
176.2. Bottom up (gdy istnieje baza)	158
176.3. Middle out (dobre przy nowym wytwarzaniu)	158
176.4. Meet in the middle (z JDBC na Hibernate'a)	158
176.5. Architektury	158
177. Blokady	161
178. Lock/Blokowanie	161
178.1. Enabling Optimistic Concurrency Control (Blokowanie optymistyczne)/Blokowanie optymistyczne z wersjonowaniem	161
178.2. Using Pessimistic Concurrency Control / Blokowanie pesymistyczne	162
179. Integracja ze Spring	164
180. Transakcje + Integracja JPA/Hibernate z frameworkiem Spring	164
180.1. Historia	164
180.2. Cechy	167
180.3. Architektura	168
180.4. EntityManager	170
181. Transakcja	170
181.1. Wyjątki	170
181.2. ACID	170
181.3. Transakcje w kodzie	171
181.4. Transakcje deklaratywne	171
181.5. Atrybuty transakcji	171
181.6. Podsumowanie : który poziom na co pozwala :)	175
181.7. read only	175
181.8. timeout	175
181.9. noRollbackFor	176

181.10. rollbackFor	176
182. Konfiguracja :.....	176
182.1. Strategia dla bazy wbudowanej :.....	179
182.2. Baza wbudowana / konfiguracja xml	180
183. H2 w konsoli WEB	180
184. Custom JPA = rozwiązywanie problemów z izolacją transakcji	181
185. TransactionTemplate.....	183
185.1. Użycie	183
186. Tworzenie repozytorium jpa/Hibernate	184
187. Tworzenie repozytorium jpa	184
188. Praca z wieloma manadzerami transakcji.....	184
188.1. Ulepszenia / swoje adnotacje	185
189. Dodatek	185
189.1. Wsparcie JDBC	185
189.2. Tworzenie repozytorium jdbc	185
190. Spring Data	186
191. Spring Data	186
191.1. Najnowsze features:)	186
191.2. Geneza	186
191.3. Charakterystyka.....	186
191.4. Nazwane zapytania.....	194
191.5. Web support	204
192. Envers	207
193. Envers	207
193.1. Konfiguracja	207
194. Alternatywy QueryDSL	208

1. O mnie

- Scalatech
- Architect Solution - RiscoSoftware
- JavaTech trener : Spring ekosystem, JPA , EIP Camel
- Sages trener : JPA , EIP - Apache Camel
- blog <http://przewidywalna-java.blogspot.com>
- twitter przodownikR1

[tUWf7KiC]

2. Źródła wiedzy

- Designing_Data_Intensive_Applications
- Java EE 7 Performance Tuning and Optimization
- Hibernate in Action
- Java Persistence with Hibernate
- Java JEE 6
- Pro JPA 2
- Pro JPA 2: Mastering the Java™ Persistence API (Expert's Voice in Java Technology)
- Hibernate from Novice to Professional
- Spring Data Modern Data Access for Enterprise Java
- Spring Data
- Spring Boot
- Spring Essentials
- Spring in Action
- etc

3. Hibernate / JPA

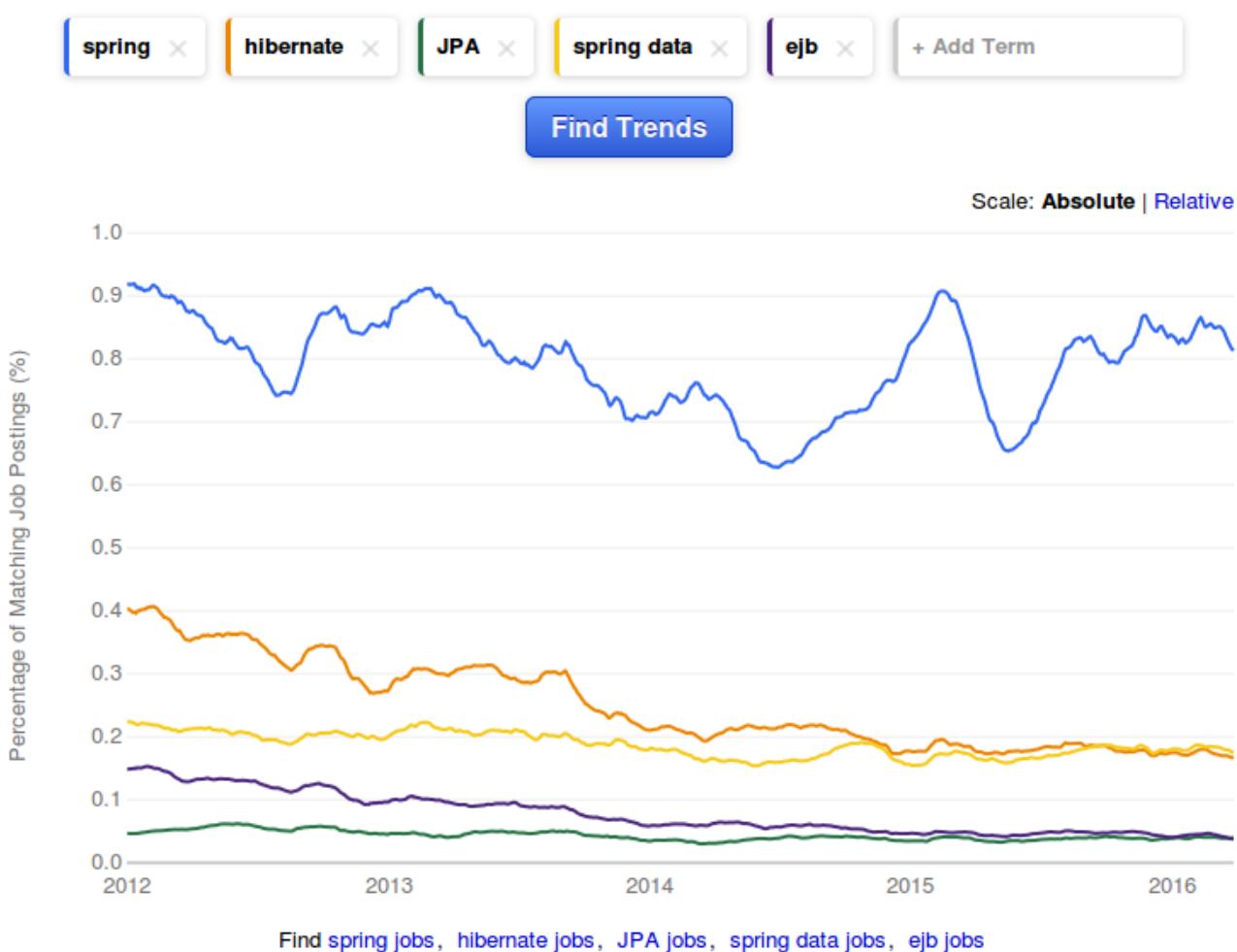
4. Co to jest w ogóle jest :) ?

Deweloperzy korzystają z Hibernate ponieważ czują się niekomfortowo z SQL i RDBS.
Ale zanim przejdą na Hibernate powinni być biegli w SQL'u i JDBC , ponieważ Hibernate korzysta z JDBC
a nie zastępuje go. Koszt mniejszej wydajności to dodatkowa wersja abstrakcji.
Gavin King (twórca Hibernate)

- najpopularniejsza implementacja odwzorowania
- jedna z implemetacji standardu **JPA**
- obsługa asocjacji, kompozycji, dziedziczenia, polimorfizmu
- wysoka wydajność i skalowalność (dwuwarstwowy cache i wparcie dla clusteringu) (**Hibernate Shards**)
- wiele sposobów tworzenia i wydawania zapytań
- nakładka na **JDBC**

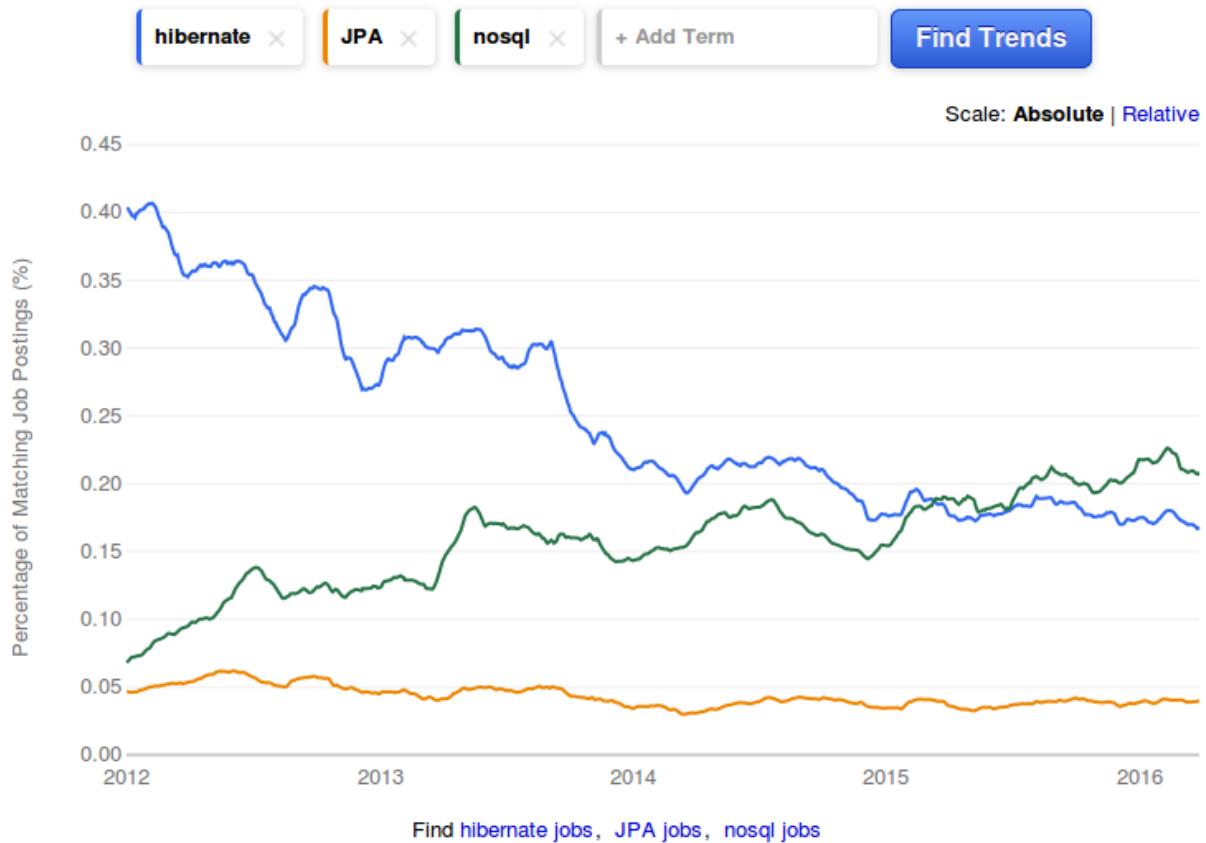
5. Trendy :)

spring, hibernate, JPA, spring data, ejb Job Trends

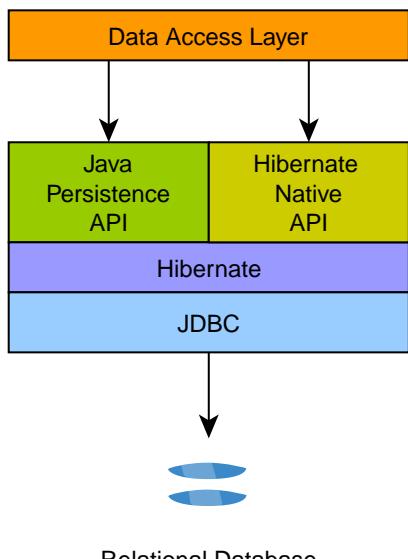


5.1. Nosql vs Hibernate

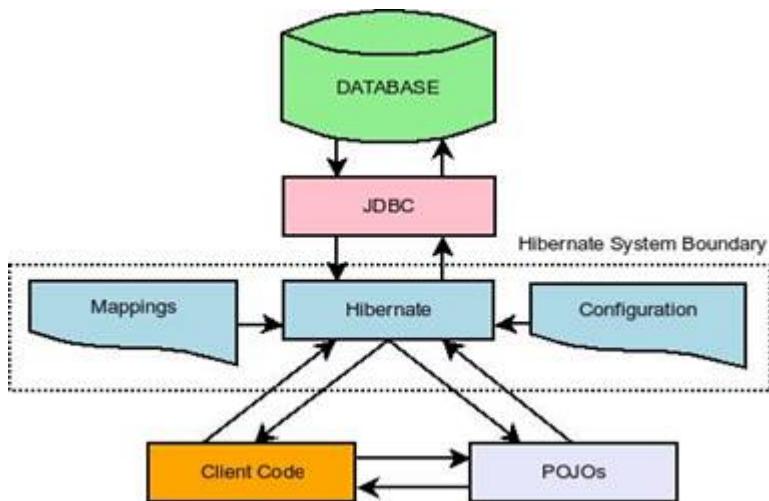
hibernate, JPA, nosql Job Trends



6. Architektura



6.1. Zasada działania



6.1.1. Zalety

Standardowe podejście oparte na JDBC wymaga od programisty następujących czynności:

- Otwarcie połączenia bazodanowego
- Zarządzanie połączeniami w ramach puli
- Budowanie kwerendy
- Wykonanie kwerendy
- Otrzymanie wyniku
- Zmapowanie wyniku na właściwą klasę
- Zamknięcie połączenia bazowanego



Zauważ, że to podejście właściwe dla wzorca template.

Czyli ORM :

- redukuje znaczenie ilość kodu potrzebnego do stworzenia i wykonania zapytania
- ochrona programisty przed czasochłonnym SQLem
- redukuje koszt i czas deploymentu
- redukcja boilerplate JDBC
- skupienie na problemach biznesowych
- mniej synchronizacji kodu z relacyjną bazą danych
- zwiększa szybkość wytwarzania (produktywność - metaDane, query data)
- dostarcza mechanizmy przenośności na inne bazy (nie musimy ograniczać się do danego typu bazy) (portability)

Dostępne dialekty :

- DB2Dialect (supports DB2)
- FrontBaseDialect
- HSQLDialect

- InformixDialect
- IngresDialect
- InterbaseDialect
- MySQLDialect
- Oracle8Dialect
- Oracle9Dialect
- Oracle10Dialect
- PointbaseDialect
- PostgreSQLDialect
- ProgressDialect
- SybaseDialect
- idealnie nadaje się do prototypowania

Dodatkowo otwiera możliwości :

- reużywalności kodu
- zarządzania transakcjami
- wydajnego operowania na kolekcjach relacji
- wbudowany mechanizm cache'u
- wprowadza obiektowe techniki do świata relacyjnych baz danych
- wprowadza runtime'owy mechanizm trzymania i zarządzania grafem zależności w pamięci wraz z synchronizacją z bazą.



Szukaj alternatyw jak naturalne trzymanie grafów obiektów w bazie Neo4j czy dokumentów w MongoDB.

6.1.2. Wady

- krzywa nauki
- dodatkowy narzut na zapytania (overhead)
- w pewnych przypadkach powoduje spadek wydajności z stosunku do zwykłych zapytania JDBC
- wraz ze wzrostem złożoności modelu domenowego występują trudności z mapowaniem, wydajnością. Skutkuje to często wprowadzaniem 'haków'

source : <http://martinfowler.com/bliki/OrmHate.html>

source : <http://blogs.tedneward.com/post/the-vietnam-of-computer-science/>

7. Moduły

▪ **Hibernate Core**

- odpowiada za generowanie natywnych kwerent SQL
- dostarcza dialekty
- dostarcza mechanizmy obsługi i translacji zapytań jak : HQL, Criteria Query czy QBE
- odpowiada za buforowanie i efektywne pobieranie danych - czyli optymalizuje zapytania

▪ **Hibernate Annotation**

- dostarcza znaczki metadanych równorzędne do konfiguracji mapować w XML
- redukcja XML

▪ **Hibernate EntityManager**

- jest implementacją interfejsu JPA

▪ **Hibernate Search**

- dostarcza rozwiązań użycia indeksów Lucene
 - rozwiązania alternatywne : [SOLR](#) , [ElasticSearch](#)

▪ **Hibernate Validator**

- obsługa walidacji danych

▪ **Hibernate OGM**

- rozwiązań NoSql
- np dla Neo4j : [hibernate-ogm-neo4j](#)
- np dla MongoDB : <http://mvnrepository.com/artifact/org.hibernate.ogm/hibernate-ogm-mongodb> [hibernate-ogm-mongodb]

▪ **DataSourcePool**

- [hibernate-c3p0](#) - Integracja C3P0 z Hibernate

▪ Przykład

```
<property name="hibernate.c3p0.min_size">5</property>
<property name="hibernate.c3p0.max_size">10</property>
<property name="hibernate.c3p0.timeout">300</property>
<property name="hibernate.c3p0.max_statements">50</property>
<property name="hibernate.c3p0.acquire_increment">1</property>
<property name="hibernate.c3p0.idle_test_period">3000</property>
```

- min_size = minimalna liczba połączeń dostępna zawsze kiedy jest taka potrzeba
- max_size = max liczba połączeń dostępnych w ramach puli

- timeout = max czas bezczynności po którym połączenie jest usuwane z puli
- idle_test_period = czas w sekundach przed którym połączenie jest automatycznie walidowane
- max_statements = max liczba kwerend które mogą być buforowane w ramach puli
- acquire_increment = liczba nowych połączeń jeśli pula jest wyczerpana
 - hibernate-hikaricp
 - hibernate-proxool
- Cache
 - hibernate-ehcache
 - hibernate-infinispan

8. Minimalna konfiguracja projektu :

9. Hibernate

- Maven

```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.3.5.Final</version>
  </dependency>
</dependencies>
```

- Konfiguracja bazy

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.4.178</version>
</dependency>
```

Możliwe typy konfiguracji : - konfiguracja bezpośrednio w kodzie

```

Configuration configuration = new Configuration().addResource("Book.hbm.xml")
.setProperty("hibernate.dialect", "org.hibernate.dialect.DerbyTenSevenDialect")
.setProperty("hibernate.connection.driver_class",
"org.apache.derby.jdbc.EmbeddedDriver")
.setProperty("hibernate.connection.url", "jdbc:derby://localhost:1527/BookDB")
.setProperty("hibernate.connection.username", "book")
.setProperty("hibernate.connection.password", "book");

ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().applySettings(
configuration.getProperties()).build();
sessionFactory = configuration.buildSessionFactory(serviceRegistry);

```

- konfiguracja xml
- konfiguracja z użyciem plików properties
 - Pojęcia
 - Encja
 - PersistenceUnit

```

@PersistenceUnit(unitName="pu-unit")
private EntityManagerFactory emf;

```

- PersistenceContext

```

@PersistenceContext (unitName = "pu-unit")
EntityManager entityManager;

```

10. Terminy związane z Hibernate

- przezroczyste utrwalanie danych
- mapowania obiektów na tabele
- sprawdzanie zabrudzeń (dirty-checking)
- przechodność utrwalania (transitive persistence)
- przechodność utrwalania (lazy loading/fetching)
- generowanie schematu danych
- symulacja dziedziczenia modelu relacyjnego

11. Symbioza

11.1. Lombok

```
@Entity  
@Data  
@AllArgsConstructor  
@Builder  
@NoArgsConstructor  
public class Person implements Serializable{
```

11.2. JAXB

```
@XmlTransient  
@XmlJavaTypeAdapter
```

11.3. JSON

```
@Convert(converter = JpaConverterJson.class) //on field  
  
public class JpaConverterJson implements AttributeConverter<Object, String> {  
  
    private final static ObjectMapper objectMapper = new ObjectMapper();  
  
    @Override  
    public String convertToDatabaseColumn(Object meta) {  
        try {  
            return objectMapper.writeValueAsString(meta);  
        } catch (JsonProcessingException ex) {  
            return null;  
            // or throw an error  
        }  
    }  
  
    @Override  
    public Object convertToEntityAttribute(String dbData) {  
        try {  
            return objectMapper.readValue(dbData, Object.class);  
        } catch (IOException ex) {  
            // logger.error("Unexpected IOException decoding json from database: " + dbData);  
            return null;  
        }  
    }  
}
```

```

@Entity
public class User {

    @Basic
    @JsonIgnore
    private String password;

    @Basic
    @JsonIgnore
    private Address address;

    // Constructors, getters, setters
}

```

source: <https://github.com/FasterXML/jackson-databind>

11.4. Bean Validation

- @Pattern
- @Size
- @Max -@Min
- etc

12. Nowości w JPA 2.1

- Querying Stored Procedure

```

@Test
public void testCallStoreProcedure() {
    StoredProcedureQuery query = em.createStoredProcedureQuery("my_sum");
    query.registerStoredProcedureParameter("x", Integer.class, ParameterMode.IN);
    query.registerStoredProcedureParameter("y", Integer.class, ParameterMode.IN);
    query.registerStoredProcedureParameter("sum", Integer.class, ParameterMode.OUT);

    query.setParameter("x", 5);
    query.setParameter("y", 4);
    query.execute();
    Integer sum = (Integer) query.getOutputParameterValue("sum");
    assertEquals(sum, new Integer(9));
}

```

source : https://en.wikibooks.org/wiki/Java_Persistence/Advanced_Topics#Stored_Procedures

```

@NamedStoredProcedureQuery(
    name = "ReadAddressById",
    resultClasses = Address.class,
    procedureName = "READ_ADDRESS",
    parameters = {
        @StoredProcedureParameter(mode=javax.persistence.ParameterMode.IN, name=
"P_ADDRESS_ID", type=Long.class)
    }
)
@Entity
public class Address {
    ...
}

StoredProcedureQuery query = em.createNamedStoredProcedureQuery("ReadAddressById");
query.setParameter("P_ADDRESS_ID", 12345);
List<Address> result = query.getResultList();

```

- **Attribute Converter**

```

@Converter
public class PasswordConverter implements AttributeConverter<String, String> {
    @Override
    public String convertToDatabaseColumn(String arg0) {
        if(arg0!=null) {
            return Base64.getEncoder().encodeToString(arg0.getBytes());
        } else {
            return null;
        }
    }

    @Override
    public String convertToEntityAttribute(String arg0) {
        if(arg0!=null) {
            return new String(Base64.getDecoder().decode(arg0));
        } else {
            return null;
        }
    }
}

```

```

@Entity
public class Person {
    @Convert(converter=PasswordConverter.class)
    String password;
}

```

- **Constructor Result Mapping** @ConstructorResult annotation is a handy addition to the already

existing `@SqlResultSetMapping` and can be used to map the result of a query to a constructor call.

```
@Entity
@NamedNativeQuery(name = "findWithTodoResultSetMapper", query = "SELECT id,
description FROM TODO where description like ?1", resultSetMapping =
"TodoResultSetMapper")
@SqlResultSetMapping(name = "TodoResultSetMapper", classes =
@ConstructorResult(targetClass = org.hall.jpa.model.TodoPOJO.class, columns = {
    @ColumnResult(name = "id", type = Long.class),
    @ColumnResult(name = "description") }))
public class Todo {
    private Long id;
    private String summary;
    private String description;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

- **Programmatic Named Queries** `addNamedQuery(String name, Query query)`

```
Query q = this.em.createQuery("SELECT a FROM Book b JOIN b.authors a WHERE b.title
LIKE :title GROUP BY a");
this.em.getEntityManagerFactory().addNamedQuery("selectAuthorOfBook", q);
```

```
TypedQuery<Author> nq = this.em.createNamedQuery("selectAuthorOfBook", Author.class);
nq.setParameter("title", "%Java%");
List<Author> authors = nq.getResultList();
```

- **Named Entity Graph**
- **Java 8 Date Time API**

Hibernate wspiera date i czas z javy 8 ale musimy podłączyć bibliotekę : [hibernate-java8](#)

13. Connection pooling

- Tworzenie połączeń do bazy danych jest kosztowne.
- Utrzymywanie wielu niewykorzystywanych połączeń jest kosztowne
- Hibernate dostarcza gotowe rozwiązanie do poolingu. Jednak rozwiązanie nie jest zalecane w produkcyjnym środowisku.
- Tworzenie konstrukcji przygotowawczych dla niektórych sterowników jest również kosztowne
- Zalecane w środowiskach produkcyjnych jest wykorzystanie zewnętrznych poll poprzez odwołania z JNDI lub konfigurowane zewnętrznie poprzez classpath czy odpowiednie pliki properties.

Przykład zewnętrznej puli połączeń c3p0:

```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-c3p0</artifactId>
    <version>[4.2.6,4.2.9)</version>
  </dependency>
  <dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>[0.9.2.1,)</version>
  </dependency>
</dependencies>
```

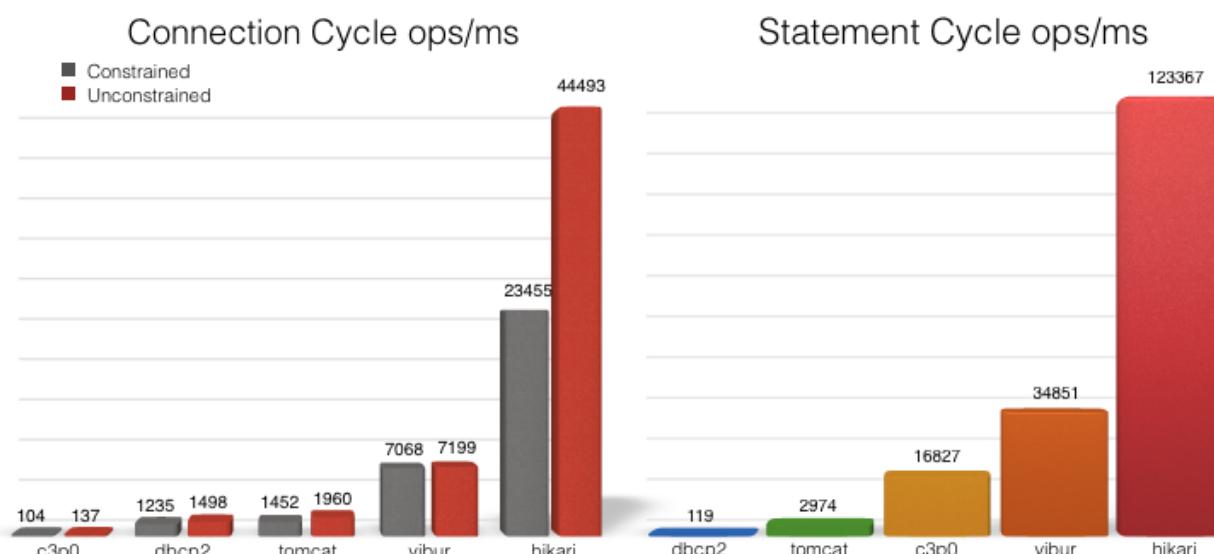
Aby użyć powyższą konfigurację c3p0 wszystko co musimy zrobić to dodać wpis do konfiguracji hibernate:

```
<property name="c3p0.timeout">10</property>
```

W ten sposób Hibernate wyłączy wewnętrzną pulę połączeń i przestawi się na zewnętrzną.

Inne rozwiązania to :

- proxool
- boneCp
- Apache poll
- hikarii
- Spring pool connection
- Server/Container connection pool



13.1. Przykład zestawienia puli połączeń dla Springa

```
<bean id="employeeDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">
    <property name="driverClass" value="${jdbc.driverClassName}" />
    <property name="jdbcUrl" value="${jdbc.employee_db_url}" />
    <property name="user" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
    <property name="maxPoolSize" value="${jdbc.maxPoolSize}" />
    <property name="minPoolSize" value="${jdbc.minPoolSize}" />
    <property name="maxStatements" value="${jdbc.maxStatements}" />
    <property name="testConnectionOnCheckout" value="${jdbc.testConnection}" />
</bean>
```

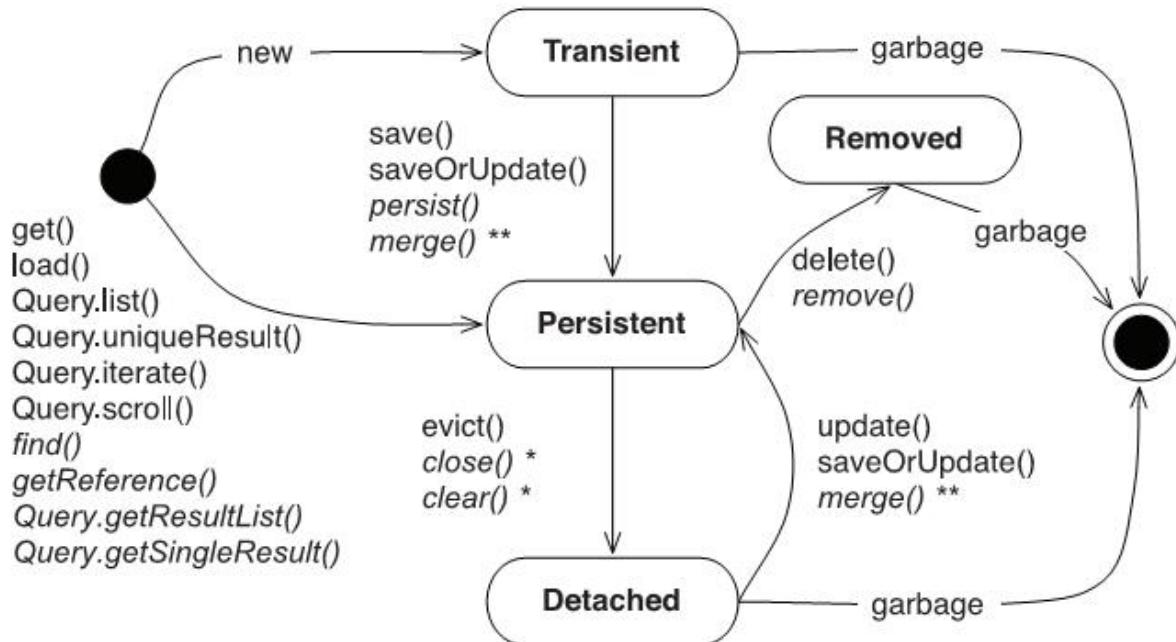
- przykładowy plik ustawień

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.employee_db_url=jdbc:mysql://localhost:3306/testdb
jdbc.username=root
jdbc.password=root
jdbc.maxPoolSize=50
jdbc.minPoolSize=10
jdbc.maxStatements=100
jdbc.testConnection=true
```

14. Encja - Trwała klasa

- klasa nie może być finalna
- klasa musi zawierać bezargumentowy konstruktor
- metody get/set dla trwałych pól
- klasa oznaczona jest adnotacją @Entity
- klasa zawiera unikalny identyfikator @Id

15. Cykl życia



- **Transient** (ulotny) - obiekt istnieje w pamięci i jest rozłączony od kontekstu Hibernate. Taki obiekt nie może być zarządzany przez Hibernate
 - Tworzony za pomocą operatora `new`. Nie skojarzony z sesją.

source:<http://myjourneyonjava.blogspot.com/2014/12/explain-about-hibernate-object-life.html>

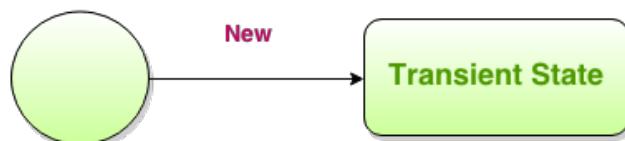


Fig. Converting New object to Transient State

Utrwalanie: `save()`, `persist()`, `saveOrUpdate()`
`save()` i `persist()` -> Insert
`update()` i `merge()` -> Update

- **Persistence** (trwały) - obiekt istnieje w bazie danych. Obiekt jest zarządzany przez Hibernate czyli jest związany z sesją.

EntityManager#persist()

source:<http://myjourneyonjava.blogspot.com/2014/12/explain-about-hibernate-object-life.html>

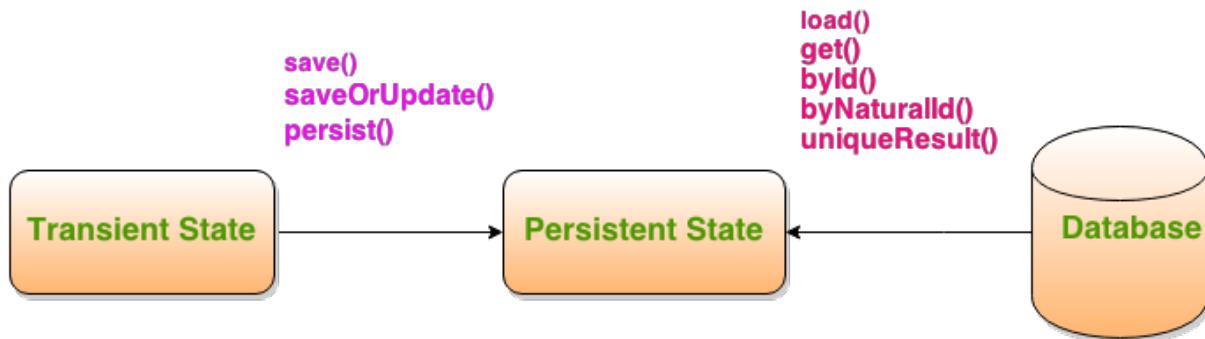


Fig. Converting Transient State to Persistent State

- **Detached** (odłączony) - obiekt ma reprezentacje w bazie danych, ale zmiana wartości obiektu nie ma wpływu na reprezentacje bazodanową i odwrotnie.

Był trwały ale został odłączony od sesji.
Możliwy do modyfikacji poza kontekstem.
Przyłączenie do sesji jest możliwe

source:<http://myjourneyonjava.blogspot.com/2014/12/explain-about-hibernate-object-life.html>

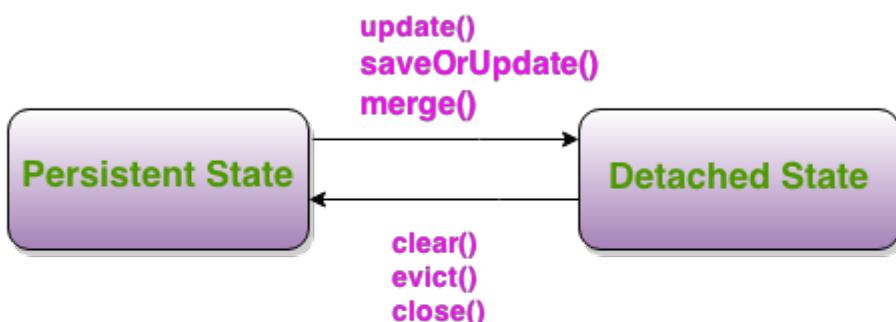


Fig. Converting Persistent State to Detached State

- **Removed** / Usunięcie - obiekty były zarządzane przez Hibernate, ale w wyniku operacji **remove()** zostały skasowane z bazy danych.
 - **delete()**, **EntityManager#remove()****

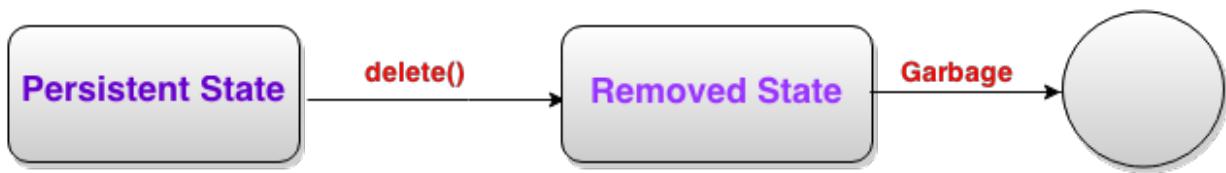


Fig. Converting Persistent State to Removed State

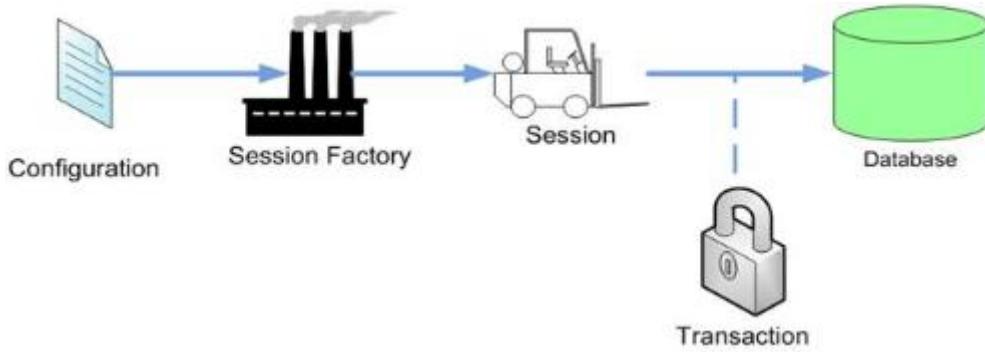
16. Session Factory ~ EntityManagerFactory

- Służy do tworzenia obiektów Session (tworzenie, zarządzanie i pobieranie Session)
- Jedna na kontekst (singleton pattern)
- thread-safe (immutable)
- ciężka i kosztowna do stworzenia
- konfigurowalna programowo lub poprzez konfiguracją xml
- współdzielony przez wiele wątków udostępniający egzemplarze Session

```
SessionFactory factory = configuration.buildSessionFactory(serviceRegistry);
//  
SessionFactory factory = new AnnotationConfiguration().configure().
buildSessionFactory();
```

```
<hibernate-configuration>
<session-factory>
<!-- H2 Configuration -->
<property name="connection.driver_class">org.h2.Driver</property>
<property name="connection.url">jdbc:h2:file:./chapter1</property>
<property name="connection.username">sa</property>
<property name="connection.password"></property>

<property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping resource="Book.hbm.xml"/>
<mapping resource="Publisher.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```



17. Session ~ EntityManager (zarządza trwałości obiektów (persistence manager))

- 'Unit of work' <http://martinfowler.com/eaaCatalog/unitOfWork.html>
- Obsługuje transakcje
- Lekka i szybka do stworzenia
- można ją traktować jako fizyczne buforowalne połączenie z bazą danych
- jedno wątkowy
- krótki okres życia
- zapewnia dostęp do podstawowych operacji CRUD

18. Otwieranie i zamykanie sesji

```

Session session = factory.openSession();
try {
    // Using the session to retrieve objects
} catch(Exception e) {
    e.printStackTrace();
} finally {
    session.close();
}

```

18.1. Ponowne użycie istniejącej już sesji

```

SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
Session session = sessionFactory.getCurrentSession();

```

```
<property name="hibernate.current_session_context_class">Thread</property>
```



Jeśli używasz Springa do zarządzania transakcjami nigdy nie używaj tej opcji chyba, że korzystasz z JTA. Spring domyślnie konfiguruje CurrentSessionContext korzystając z SpringSessionContext

18.2. Otwieranie sesji bezstanowej

- wykorzystywane dla pojedyńczego zadania
- nie wykorzystuje żadnego cache
- nie korzysta z dirty-checking

Przykładowe wykorzystanie to : czytanie danych z pliku i wstawianie do bazy danych. (Cache jest zbyteczny to tego typu działań)

```
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
Session session = sessionFactory.openStatelessSession();
```

19. Transaction

- abstrakcyjna warstwa obsługi transakcji bez względu na jej implementację (JDBC, JTA)
- jedno wątkowy
- określa granice jednej transakcji

```
Company company = new Company(1, "Scalatech");
Transaction tx = session.beginTransaction();
session.save(company);
tx.commit();
session.flush();
session.close();
```

20. Odświeżanie encji

- operacja **refresh()**;
- odświeża stan encji z bazy nadpisując dane wprowadzone do encji
- anuluje zmiany dokonane na instancji w pamięci



Anti-pattern

21. Opróżnianie sesji

- operacja **flush()**;

- synchronizuje persistence context z bazą danych
- wszystkie encje zostaną wstawione/uaktualnione/skasowane z bazie (wysyła instrukcje CRUD do bazy)
- Hibernate zatwierdza wszystkie zmiany będące w kolejce.



Anti-pattern → System.gc()

22. Czyszczenie kontekstu (JPA)

- czyści kontekst utrwalania
- wszystkie encje zostaną odłączone

odśwież stan instancji encji danymi z bazy danych, nadpisując wprowadzone zmiany w encji

23. Czy sesja jest zanieczyszczona ?

- isDirty()

24. Adnotacje

24.1. Pola

Ten rodzaj mapowania oparty jest na typach i nazwach pól.

24.2. Metody

Na poziomie właściwości klasy. Adnotację umieszcza się przy @Getter.

24.3. Mieszany @Access

- Przykład

```
@Entity  
@Access(AccessType.FIELD)  
public class Book{  
    @Id  
    @Setter  
    @Getter  
    private Long id;  
  
    private String name;  
  
    @Access(AccessType.PROPERTY)  
    public String getName()...  
    public void setName(String name)..."  
}
```

25. @Entity

- oznacza klasy klasy, które mają brać udział w procesie utrwalania
- taka klasa może być abstract
- klasa nie może być final
- klasa nie może zawierać pola i metody final
- klasa musi posiadać bezargumentowy konstruktor
- klasa musi posiadać klucz główny @Id



Dla komunikacji (distributed/web/session/serializable) powinna implementować Serializable. Klasa posiada wyróżnioną tożsamość

- Przykład

```
@Entity  
public class Simple {  
    ...  
}
```

26. Klucze @Id

26.1. Prosty

26.2. Ustawianie wartości

26.2.1. ręcznie

26.2.2. automatyczne

- każda klasa encyjna musi posiadać unikalny identyfikator.
 - Przykład

```
@Id  
public Long id;
```

- **Database sequence** - wykorzystuje sekwencje. Wsparcie dla baz DB2, PostgreSQL, Oracle, SAP DB.
- **Native generator** - wybiera jedną ze strategii generowania identyfikatorów : identity, sequence, hilo w zależności od możliwości bazy
 - wsparcie dla kolumn identity w bazach DB2, MySQL, MS SQL Server, Sybase, HypersonicSQL.
- **Increment generator** (Identity)
- **Hilo generator** - identyfikatory są unikalne w ramach całej bazy.
- **UUID** - generuje Stringi (unikalny w sieci adres ip + znacznik czasu). Jest to kosztowne rozwiązanie.
- **assigned** - pozwala aplikacji nadać identyfikator zanim obiekt zostanie zapisany.(persist, save)

26.2.3. @TableGenerator

- Przykład

```
@TableGenerator(name="Book_Gen", table="ID_GEN",pkColumnName="GEN_NAME",  
valueColumnName="GEN_VAL", initialValue=10000, allocationSize=100)  
  
@Id  
@GeneratedValue(strategy = GenerationType.TABLE,generator="Book_Gen")  
private Long id;
```

```

@Entity public class Employee {
    ...
    @TableGenerator(
        name="empGen",
        table="ID_GEN",
        pkColumnName="GEN_KEY",
        valueColumnName="GEN_VALUE",
        pkColumnValue="EMP_ID",
        allocationSize=1)
    @Id
    @GeneratedValue(strategy=TABLE, generator="empGen")
    int id;
    ...
}

```

26.2.4. @SequenceGenerator

- Przykład

```

@SequenceGenerator(name="Book_Gen", sequenceName="Book_Seq", initialValue=10000
,allocationSize=100)

@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE,generator="Book_Gen")
private Long id;

```

26.3. Klucz złożony

26.3.1. Primary key column – @Id and @IdClass

- Przykład

```

public class BookPK implements Serializable{

private String name;
private String isbn;
public int hashCode() {
return ...;
}
public boolean equals(Object obj) {
return ...;
}
}

```

```
@IdClass(BookPK.class)
@Entity
public class Book{
@Id
private String id;
@Id
private String isbn;
}
```

26.3.2. @EmbeddedId

- Przykład

```
@Entity
class User {
    @EmbeddedId
    @AttributeOverride(name="firstName", column=@Column(name="fld_firstname"))
    UserId id;

    Integer age;
}

@Embeddable
class UserId implements Serializable {
    String firstName;
    String lastName;
}
```

NOTE : Może być wykorzystywany przez @ElementCollection

26.4. ElementCollection

- Przykład

```
@Entity
public class Employee {
    @Id
    @Column(name="EMP_ID")
    private long id;
    ...
    @ElementCollection
    @CollectionTable(
        name="PHONE",
        joinColumns=@JoinColumn(name="OWNER_ID")
    )
    private List<Phone> phones;
    ...
}
```

```
@Embeddable
public class Phone {
    private String type;
    private String areaCode;
    @Column(name="P_NUMBER")
    private String number;
    ...
}
```

26.5. AttributeOverride

- Przykład

```

@Entity
public class Employee {
    @Id
    private long id;
    ...
    @Embedded
    @AttributeOverrides({
        @AttributeOverride(name="startDate", column=@Column(name="START_DATE")),
        @AttributeOverride(name="endDate", column=@Column(name="END_DATE"))
    })
    private Period employmentPeriod;
    ...
}

@Entity
public class User {
    @Id
    private long id;
    ...
    @Embedded
    @AttributeOverrides({
        @AttributeOverride(name="startDate", column=@Column(name="SDATE")),
        @AttributeOverride(name="endDate", column=@Column(name="EDATE"))
    })
    private Period period;
    ...
}

```

27. @Table

- domyślnie nazwa tabeli jest taka sama jak nazwa klasy.
- jeśli domyślne ustawienie jest nie wystarczające z różnych powodów możemy użyć @Table
 - Przykład

```

@Table(name = "ITEMS",uniqueConstraints =@UniqueConstraint(name = "UNQ_NAME"
, columnNames = { "ITEM_NAME" })
)
public class Item extends AbstractEntity {

    private static final long serialVersionUID = 5474170031394030929L;
    @Column(name="ITEM_NAME")
    private String name;
}

```

```

create table ITEMS (
    id bigint not null,
    ITEM_NAME varchar(255),
    primary key (id)
)
alter table ITEMS add constraint UNQ_NAME unique (ITEM_NAME)

```

27.1. Określenie schematu bazy w persistence.xml

```

<entity-mappings>
    <persistence-unit-metadata>
        <persistence-unit-defaults>
            <schema name="purchasing"/>
        </persistence-unit-defaults>
    </persistence-unit-metadata>
    ...
</entity-mappings>

```

27.2. @Index

- Przykład

```

@Table(name = "ITEMS",
indexes = {@Index(name = "IDX_USERNAME", columnList = "ITEM_NAME")}
public class Item extends AbstractEntity {

    @Column(name="ITEM_NAME")
    private String name;
}

```

```

create table ITEMS (
    id bigint not null,
    ITEM_NAME varchar(255),
    primary key (id)
)
create index IDX_USERNAME on ITEMS (ITEM_NAME)

```

28. @Column

- analogiczne zachowanie do adnotacji @Table
- insertable/updatable - określa czy dana kolumna będzie brała udział w operacjach insert/update
 - Przykład

```

@Column(name = "retryattempt", columnDefinition = "numeric", nullable = true)
private int retryAttempt = 0;

@Column(name = "messageerror", columnDefinition = "nvarchar")
private String messageError;

@Column(name = "messagebody", length = Integer.MAX_VALUE, columnDefinition =
"nvarchar")
private String body;

@Column(name = "detailstatus", columnDefinition = "nvarchar")
@Enumerated(EnumType.STRING)
private DetailStatus status;

```

- Przykład 1

```

@Column(nullable=false,scale=2,precision=2)
private BigDecimal price;

```

```
price decimal(2,2) not null
```

- Przykład 2

```

@Column
private BigDecimal price;

```

```
price decimal(19,2)
```

- Przykład 3

```

@Column(name="ITEM_NAME",length=20,unique=true)

```

```

create table Item (
    id bigint not null,
    version bigint,
    ITEM_NAME varchar(20),
    price decimal(2,2) not null,
    primary key (id)
)
alter table Item add constraint UK_bjye5lp3xnccmg4ovtumigp3v unique (ITEM_NAME)

```

- Przykład 4

```
@Column(columnDefinition ="varchar(15) not null unique check (not
substring(lower(OWNER), 0, 5) = 'admin')")
private String owner;
```

```
create table Item (
    id bigint not null,
    version bigint,
    ITEM_NAME varchar(20),
    owner varchar(15) not null unique check (not substring(lower(OWNER), 0, 5) =
'admin'),
    price decimal(2,2) not null,
    primary key (id)
```

29. @Check

- Przykład

```
@org.hibernate.annotations.Check(
constraints = "AUCTIONSTART < AUCTIONEND"
)
public class Offer extends AbstractEntity{
@NotNull
protected Date auctionStart;
@NotNull
protected Date auctionEnd;
}
```

```
create table Offer (
    id bigint not null,
    version bigint,
    auctionEnd binary(255) not null,
    auctionStart binary(255) not null,
    offer_value decimal(19,2),
    ITEM_ID bigint not null,
    primary key (id),
    check (AUCTIONSTART < AUCTIONEND)
)
```

30. @Transient

- pole nie podlega procesowi utrwalania

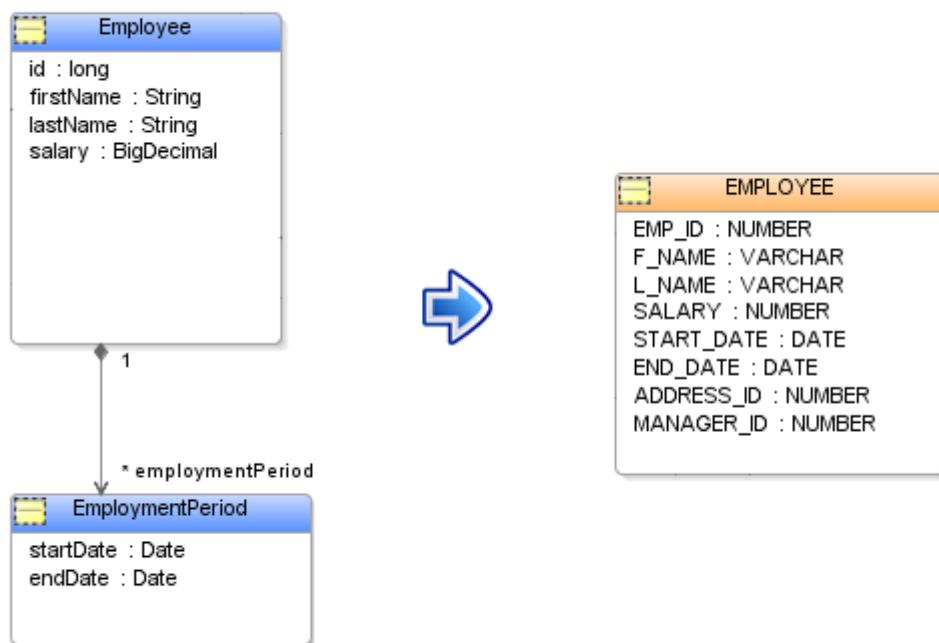
31. @Basic

- określa czy pole ma być opcjonalne (przydatne podczas generowania schematu przez Hibernate).
- określa również sposób pobierania danych, czy pole ma być wypełniane od razu przy odczycie obiektu czy dopiero przy pierwszym odwołaniu.

32. @Embeddable i @Embedded**

- umożliwia osadzanie nieencyjnych obiektów Java w obiektach encyjnych

source: https://en.wikibooks.org/wiki/Java_Persistence/Embeddables



- Przykład

```
@Embeddable
public class EmploymentPeriod {
    @Column(name="START_DATE")
    private java.sql.Date startDate;

    @Column(name="END_DATE")
    private java.sql.Date endDate;
    ...
}
```

```
@Entity
public class Employee {
    @Id
    private long id;
    ...
    @Embedded
    private EmploymentPeriod period;
    ...
}
```

```
@Embeddable
public class Address {

    private String line1;

    private String line2;

    @Embedded
    private ZipCode zipCode;

    ...
    @Embeddable
    public static class Zip {

        private String postalCode;

        private String plus4;

        ...
    }
}

@Entity
public class Person {

    @Id
    private Integer id;

    @Embedded
    private Name name;

    ...
}
```

@Multiple embeddable types

```

@Entity
public class Contact {

    @Id
    private Integer id;

    @Embedded
    private Name name;

    @Embedded
    private Address homeAddress;

    @Embedded
    private Address mailingAddress;

    @Embedded
    private Address workAddress;

    ...
}

```

@AttributeOverride

- Przykład

```

@Entity
public class Contact {

    @Id
    private Integer id;

    @Embedded
    private Name name;

    @Embedded
    @AttributeOverrides(
        @AttributeOverride(
            name = "line1",
            column = @Column( name = "home_address_line1" ),
        ),
        @AttributeOverride(
            name = "line2",
            column = @Column( name = "home_address_line2" )
        ),
        @AttributeOverride(
            name = "zipCode.postalCode",
            column = @Column( name = "home_address_postal_cd" )
        ),
        @AttributeOverride(

```

```

        name = "zipCode.plus4",
        column = @Column( name = "home_address_postal_plus4" )
    )
)
private Address homeAddress;

@Embedded
@AttributeOverrides(
    @AttributeOverride(
        name = "line1",
        column = @Column( name = "mailing_address_line1" ),
    ),
    @AttributeOverride(
        name = "line2",
        column = @Column( name = "mailing_address_line2" )
    ),
    @AttributeOverride(
        name = "zipCode.postalCode",
        column = @Column( name = "mailing_address_postal_cd" )
    ),
    @AttributeOverride(
        name = "zipCode.plus4",
        column = @Column( name = "mailing_address_postal_plus4" )
    )
)
private Address mailingAddress;

@Embedded
@AttributeOverrides(
    @AttributeOverride(
        name = "line1",
        column = @Column( name = "work_address_line1" ),
    ),
    @AttributeOverride(
        name = "line2",
        column = @Column( name = "work_address_line2" )
    ),
    @AttributeOverride(
        name = "zipCode.postalCode",
        column = @Column( name = "work_address_postal_cd" )
    ),
    @AttributeOverride(
        name = "zipCode.plus4",
        column = @Column( name = "work_address_postal_plus4" )
    )
)
private Address workAddress;

...
}

```

33. @Enumerated

- mapowanie enum
 - Przykład

```
@Entity
public class Person {
    @Enumerated
    public Gender gender;
    public static enum Gender {
        MALE,
        FEMALE
    }
}
```

- **@AttribureConverter**

- Przykład

```
public enum Gender {

    MALE('M'),
    FEMALE('F');

    private final char code;

    private Gender( char code ) {
        this.code = code;
    }

    public static Gender fromCode( char code ) {
        if ( code == 'M' || code == 'm' ) {
            return MALE;
        }
        if ( code == 'F' || code == 'f' ) {
            return FEMALE;
        }
        throw...
    }

    public char getCode() {
        return code;
    }
}

@Entity
public class Person {
    ...
}
```

```

@Basic
@Convert( converter = GenderConverter.class )
public Gender gender;
}

@Converter
public class GenderConverter implements AttributeConverter<Character, Gender> {

    public Character convertToDatabaseColumn( Gender value ) {
        if ( value == null ) {
            return null;
        }

        return value.getCode();
    }

    public Gender convertToEntityAttribute( Character value ) {
        if ( value == null ) {
            return null;
        }

        return Gender.fromCode( value );
    }
}

```

34. @Lob

34.1. java.sql.Blob

```

@Entity
public class Step {
    ...
    @Lob
    @Basic
    public byte[] instructions;
    ...
}

```

34.2. java.sql.Clob

```
@Entity  
public class Product {  
    ...  
    @Lob  
    @Basic  
    public Clob description;  
    ...  
}
```

- Przykład

```
@Entity  
public class Product {  
    ...  
  
    @Lob  
    @Basic  
    public Clob description;  
    ...  
  
    @Lob  
    @Basic  
    public char[] description;  
  
    @Lob  
    @Basic  
    public Blob instructions;  
  
    @Lob  
    @Basic  
    public byte[] instructions;  
}
```

35. Date & time

35.1. DATE

- java.sql.Date

35.2. TIME

- java.sql.Time

35.3. TIMESTAMP

- java.sql.Timestamp

36. Mapping Java 8 Date/Time Values

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-java8</artifactId>
    <version>${hibernate.version}</version>
</dependency>
```

36.1. DATE

- java.time.LocalDate

```
INSERT INTO DateEvent( timestamp, id ) VALUES ( '2015-12-29', 1 )
```

36.2. TIME

- java.time.LocalTime
- java.time.OffsetTime

```
INSERT INTO DateEvent( timestamp, id ) VALUES ( '16:51:58', 1 )
```

36.3. TIMESTAMP

- java.time.Instant,
- java.time.LocalDateTime
- java.time.OffsetDateTime
- java.time.ZonedDateTime

```
INSERT INTO DateEvent ( timestamp, id ) VALUES ( '2015-12-29 16:54:04.544', 1 )
```

37. AttributeConverters

- Przykład

```

@Converter
public class PeriodStringConverter implements AttributeConverter<Period, String> {

    @Override
    public String convertToDatabaseColumn(Period attribute) {
        return attribute.toString();
    }

    @Override
    public Period convertToEntityAttribute(String dbData) {
        return Period.parse(dbData);
    }
}

@Entity
public class Event {
    @Convert(converter = PeriodStringConverter.class)
    private Period span;

}

```

38. @Formula

- Przykład

```

@Formula("obj_length * obj_height * obj_width")
private long objectVolume;

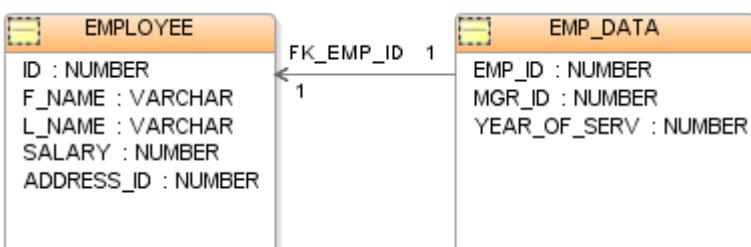
@Formula("UPPER(name)")
private String capitalName;

@Formula("(SELECT c.name FROM category c WHERE c.id=category_id)")
private String categoryName;

```

39. @SecondaryTable

source : https://en.wikibooks.org/wiki/Java_Persistence/Tables



```

@Entity
@Table(name="EMPLOYEE")
@SecondaryTable(name="EMP_DATA",
    pkJoinColumns = @PrimaryKeyJoinColumn(name="EMP_ID",
referencedColumnName="ID")
)
public class Employee {

    ...
    @Column(name="YEAR_OF_SERV", table="EMP_DATA")
    private int yearsOfService;

    @OneToOne
    @JoinColumn(name="MGR_ID", table="EMP_DATA", referencedColumnName="ID")
    private Employee manager;
    ...
}

```

40. @AttributeOverride

- Patrz wyżej w przykładzie z @Embedded.

41. @Version - blokowanie optymistyczne

- Przykład

```

Employee employee = new Employee();
employee.setId(1);
employee.setName("przodownik");
session.saveOrUpdate(employee);

```

Hibernate: `update employee set name=?, version=? where id=? and version=?`

42. @OrderColumn

- Przykład

```

@OrderColumn(name = "index_id")
private List<Change> changes = new ArrayList<>();

```

43. @ForeignKey

- Przykład

```
@Entity  
public class Phone {  
    @ManyToOne  
    @JoinColumn(name = "person_id",  
               foreignKey = @ForeignKey(name = "PERSON_ID_FK"))  
}
```

```
CREATE TABLE Phone (  
    id BIGINT NOT NULL ,  
    number VARCHAR(255) ,  
    person_id BIGINT ,  
    PRIMARY KEY ( id )  
)  
  
ALTER TABLE Phone ADD CONSTRAINT PERSON_ID_FK FOREIGN KEY (person_id) REFERENCES Person
```

44. @Type (Hibernate only)

- Przykład

```
@org.hibernate.annotations.Type( type = "nstring" )  
private String name;  
  
@org.hibernate.annotations.Type( type = "materialized_nclob" )  
private String description;
```

@UniqueConstraint(columnNames = { "id" , "empCode"})

45. @ElementCollection** - dla typów prostych lub klas osadzonych

- Przykład

```

@ElementCollection(fetch=FetchType.LAZY)
@CollectionTable(name = "email")
@IndexColumn(name="email_index")
private List<String> emails;

@ElementCollection(targetClass = CarBrands.class)
@Enumerated(EnumType.STRING)
private List<CarBrands> brands;
}

public enum CarBrands {
SUZUKI, STAR, FERRARI, JAGUAR;
}

```

46. @OrderBy

- kolekcja może zostać uporządkowana według określonych kryteriów
- w przypadku kolekcji uporządkowanej wykorzystać należy typ **List**
 - Przykład

```

@OneToMany(mappedBy="user")
@OrderBy("lastName")
protected List<User> children;

```

47. @JoinTable

- name to nazwa tabeli
- joinColumns – kolumna tabeliłączenia, stanowiąca klucz dla encji
- inverseJoinColumns – kolumna tabeliłączenia, stanowiąca klucz dla encji po drugiej stronie relacji

48. Relacje

48.1. FetchType

Strategia	Domyślny tryb
OneToMany	LAZY
ManyToOne	EAGER
ManyToMany	LAZY

@JoinColumn + @JoinTable

- One-To-One 1:1

```
@Entity
public class Message {
    @Id
    Long id;

    @Column
    String content;

    @OneToOne
    Email email;

    }
    //ommit mutators and accessors
}
```

- One-To-Many 1:N Za pomocą kluczu obcego

- Przykład

```
@Entity
public class Item extends AbstractEntity {
    private String name;
    private BigDecimal price;

    @OneToMany(fetch = FetchType.LAZY) // Defaults to EAGER
    @JoinColumn(name = "ITEM_ID")
    private List<Offer> offers;

}
```

- Przykład

```
@Entity
public class Offer extends AbstractEntity{
    @Column(name="offer_value")
    private BigDecimal value;
}
```

- Generowany SQL :

```

create table Item (
    id bigint not null,
    version bigint,
    name varchar(255),
    price decimal(19,2),
    primary key (id)
)

create table Offer (
    id bigint not null,
    version bigint,
    offer_value decimal(19,2),
    ITEM_ID bigint,
    primary key (id)
)

alter table Offer
    add constraint FKp6fm8wffictppkc0m3ufurbpy
        foreign key (ITEM_ID)
        references Item

```

Za pomocą klucza głównego

- Many-To-One N:1
 - Przykład

```

@Entity
public class Item extends AbstractEntity{
    private String name;
    private BigDecimal price;
}

@Entity
public class Offer extends AbstractEntity{
    @ManyToOne(fetch = FetchType.LAZY) // Defaults to EAGER
    @JoinColumn(name = "ITEM_ID", nullable = false,
    foreignKey = @ForeignKey(name = "FK_ITEM_ID") )
    private Item item;

    @Column(name="offer_value")
    private BigDecimal value;
}

```

```

create table Item (
    id bigint not null,
    version bigint,
    name varchar(255),
    price decimal(19,2),
    primary key (id)
)
create table Offer (
    id bigint not null,
    version bigint,
    offer_value decimal(19,2),
    ITEM_ID bigint not null,
    primary key (id)
)
alter table Offer
    add constraint FK_ITEM_ID
    foreign key (ITEM_ID)
    references Item

```

- Many-To-Many N:M

```

public class ProjectType {
    @Id
    @GeneratedValue
    private long id;
    @ManyToOne
    private Employee employee;
    @Column(name="PROJ_TYPE")
    private String type;
    @ManyToMany
    private List<Project> projects;
}

```

49. @Dynamic

49.1. @DynamicInsert (false/true)

Manipulowanie operacjami Insert na poziomie encji. Wstawiamy tylko wybrane kolumny.



Tuning. Potencjalne przyspieszenie dla dużych tabel w szerszości.

49.2. @DynamicUpdate (false/true)

Manipulowanie operacjami Update na poziomie encji. Uaktualniamy tylko te kolumny, które się zmieniły



Tuning. Potencjalne przyspieszenie dla dużych tabel w szczególności

50. @Immutable

```
@Entity  
@Immutable  
@Cache (usage=CacheConcurrencyStrategy.READ_ONLY)  
@Table(name = "products")  
public class Product extends AbstractEntity {  
  
}
```

==@SubSelect - view

```
@Entity  
@org.hibernate.annotations.Immutable  
@org.hibernate.annotations.Subselect(  
value = "select i.id as userId, i.firstName as name, " +  
"count(a.id) as addressCount " +  
"from ITEM i left outer join Address a on i.ID = a.id " +  
"group by i.id"  
)  
@org.hibernate.annotations.Synchronize({"Item", "Bid"})  
public class UserAddressStats {  
@Id  
protected Long userId;  
protected String name;  
protected long addressCount;  
public ItemBidSummary() {  
}  
}
```

51. Callbacks

51.1. @PrePersist

Wykonanie operacji przed operacją zapisu

51.2. @PreRemove

Wykonanie operacji przez operację usunięcia

51.3. @PostPersist

Wykonanie operacji po operacji zapisu

51.4. @PostRemove

Wykonanie operacji po operacji usunięcia

51.5. @PreUpdate

Wykonanie kodu przed operacją aktualizacji

51.6. @PostUpdate

Wykonanie kodu po operacji aktualizacji

51.7. @PostLoad

Wykonanie akcji po załadowaniu encji z kontekstu trwałości

52. EventListener

- Przykład

```

@Entity
@EntityListeners( LastUpdateListener.class )
public static class Person {

    @Id
    private Long id;

    private String name;

    private Date dateOfBirth;

    @Transient
    private long age;

    private Date lastUpdate;

    @PostLoad
    public void calculateAge() {
        age = ChronoUnit.YEARS.between( LocalDateTime.ofInstant(
            Instant.ofEpochMilli( dateOfBirth.getTime() ), ZoneOffset.UTC ),
            LocalDateTime.now()
        );
    }
}

public static class LastUpdateListener {

    @PreUpdate
    @PrePersist
    public void setLastUpdate( Person p ) {
        p.setLastUpdate( new Date() );
    }
}

```

53. Query, Criteria

- umożliwia dostęp do danych w bazie za pomocą wyspecjalizowanych zapytań czy API

54. Tożsamość obiektu : Equals & hashCode

- brak (Object) - (oparte na nie odłączanych encjach)
- ID tożsamość bazodanowa
- klucz biznesowy
- application managed id - (z bazy danych na aplikacje) (moment poczęcia lub urodzenia)

55. Idenyczność obiektów

55.1. Idenyczność obiektów

- Są identyczne jeśli zajmują tą samą referencję w VM. Sprawdzamy ją za pomocą operatora '=='

55.2. Równość obiektów

- Są równe jeśli operacja **equals** daje wynik pozytywny

55.3. Tożsamość bazodanowa

- Są identyczne z poziomu DB, tzn jeśli dotyczą tego samego wiersza w tabeli i mają ten sam klucz główny

55.3.1. Klucz główny musi spełniać następujące warunki

- jego wartość nigdy nie może być równa null
- każdy wiersz posiada inną wartość
- wartość dla konkretnego wiersza nigdy się nie zmienia

55.3.2. Klucz naturalny

- ma znaczenie biznesowe
- odpowiedni dobor kandydatów / stały w ciągu całego życia



Interfejs IdentifierGenerator do generowania własnych strategii kluczy identyfikujących

56. Klucz główny

- klasa klucza głównego spełnia wymagania :
 - public class
 - klasa musi posiadać domyślny publiczny konstruktor
 - musi implementować metody **equals** i **hashCode**
 - musi być serializowalna
 - klucz złożony musi być reprezentowany i mapowany do wielu pól właściwości klasy encji. Musi być też reprezentowany i mapowany jako klasa osadzona.

57. Relacje jedno i dwukierunkowe

- dwukierunkowa :
 - obie encje posiadają wzajemnie do siebie relacje
 - strona przynależna związku dwukierunkowego musi odnosić się do swojego właściciela za pomocą użycia elementu **mappedBy**, należącego do adnotacji @OneToMany, @ManyToOne, lub @ManyToMany
 - strona wielokrotna dwukierunkowego związku wiele do jednego nie może definiować **mappedBy** - jest zawsze 'posiadającą stroną relacji'
 - w relacji jeden do jeden strona posiadająca pokazuje stronie przynależnej odpowiedni klucz obcy.
 - w relacji wiele do wielu każdej ze stron może być posiadająca.

57.1. Charakterystyka

- owning side - strona właściwska
- inverse side - strona przeciwna
- możliwość nawigacji między powiązanymi encjami. Asocjacja dwukierunkowa jest definiowana jako para asocjacji jednokierunkowych, ze wskazaniem jednej z nich jako głównej.



Wiążemy ze sobą obie strony relacji



Stosujemy jeśli wymaga tego charakterystyka biznesu



Złamanie enkapsulacji



Często tworzone automatycznie gdy stosujemy inżynierie wstępna - generatory !

przykład kodu :



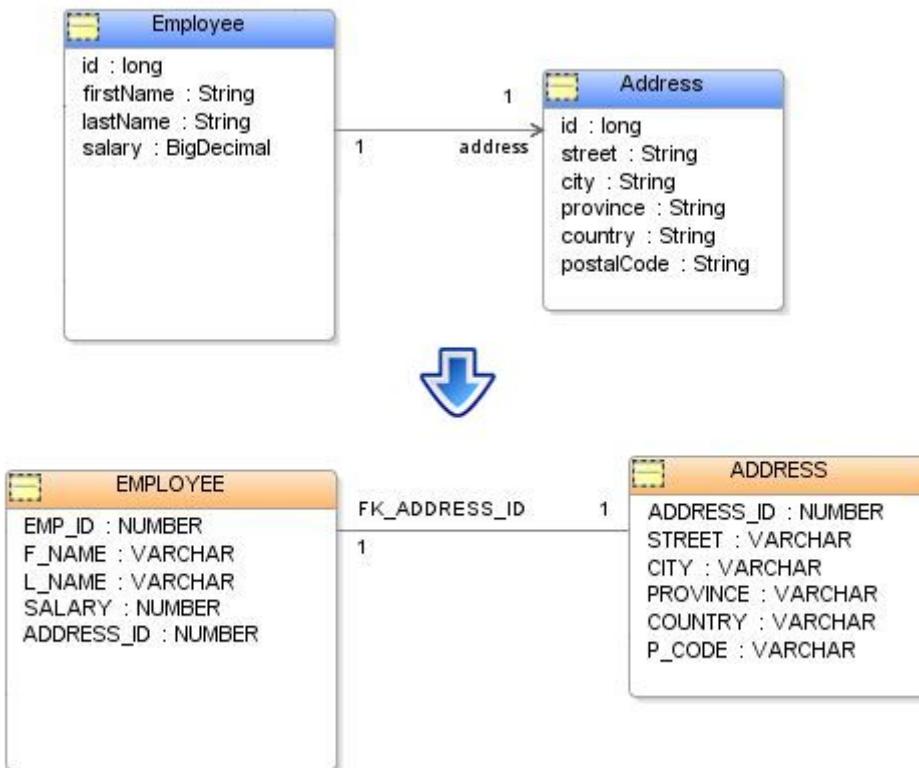
Stosować tylko gdy są uzasadnione biznesowo -Enkapsulacja. Są tworzone przez generatory JPA. Mogą ułatwiać pisanie zapytań ale nie muszą :)

- jednokierunkowe Tylko jedna strona posiada relację do innej encji

58. @OneToOne

- każda ze stron może być właścicielem relacji. Musimy określić stronę bo doprowadzimy do powstania zależności cyklicznej

source : https://en.wikibooks.org/wiki/Java_Persistence/OneToOne



```
@Entity
public class Employee {
    @Id
    @Column(name="EMP_ID")
    private long id;
    ...
    @OneToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="ADDRESS_ID")
    private Address address;
    ...
}
```

58.1. Dwukierunkowa

```
@Entity
public class Address {
    @Id
    @Column(name = "ADDRESS_ID")
    private long id;
    ...
    @OneToOne(fetch=FetchType.LAZY, mappedBy="address")
    private Employee owner;
    ...
}
```

58.2. Z dodatkową tabelą łączącą relacje

```
@OneToOne(fetch=FetchType.LAZY)
@JoinTable(
    name="EMP_ADD",
    joinColumns=
        @JoinColumn(name="EMP_ID", referencedColumnName="EMP_ID"),
    inverseJoinColumns=
        @JoinColumn(name="ADDR_ID", referencedColumnName="ADDRESS_ID"))
private Address address;
...
```

58.3. Ze wspólnym kluczem głównym

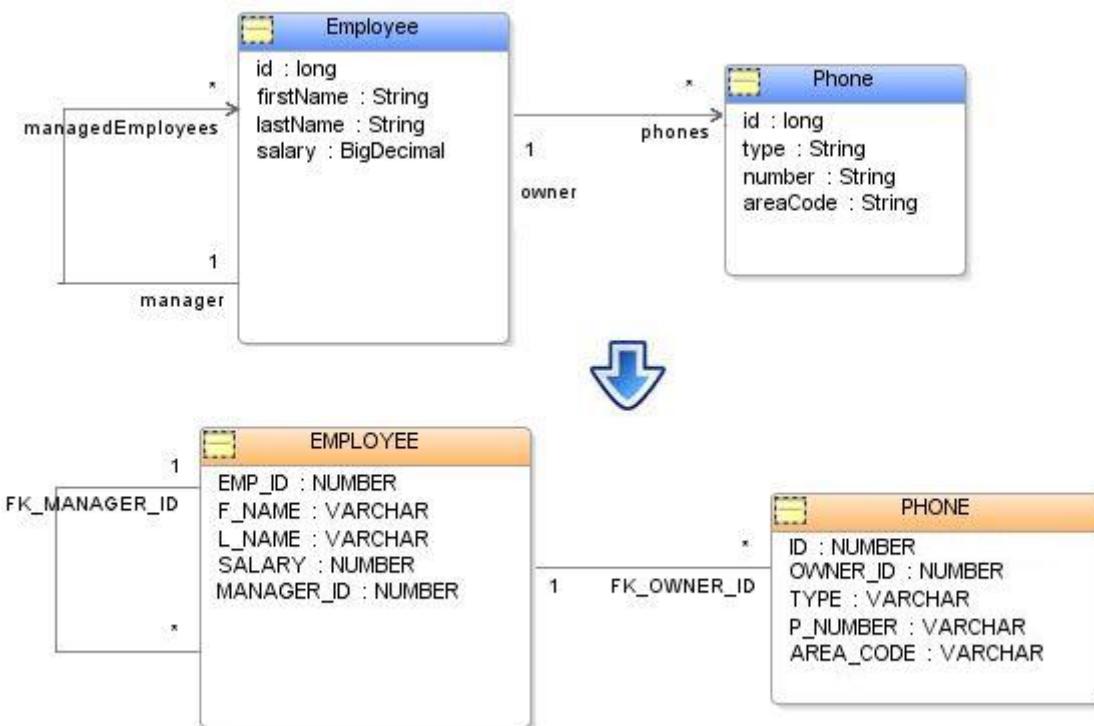
```
@Entity
public class Address {
    @Id
    @GeneratedValue(generator = TableGenerator)
    protected Long id;
    @NotNull
    protected String street;
}
//
@Entity
@Table(name = "USERS")
@Id
protected Long id;

@OneToOne(fetch = FetchType.LAZY, optional = false)
@PrimaryKeyJoinColumn
protected Address address;
```

59. @OneToMany

- strona 'wiele' musi być właściwicielem relacji

source : https://en.wikibooks.org/wiki/Java_Persistence/OneToMany



```
@Entity
public class Employee {
    @Id
    @Column(name="EMP_ID")
    private long id;
    ...
    @OneToMany(mappedBy="owner")
    private List<Phone> phones;
    ...
}
```

60. @ManyToOne

- strona 'wiele' musi być właścicielem relacji. Odwrotność relacji @OneToMany

60.1. @ManyToOne z tabelą łączącą

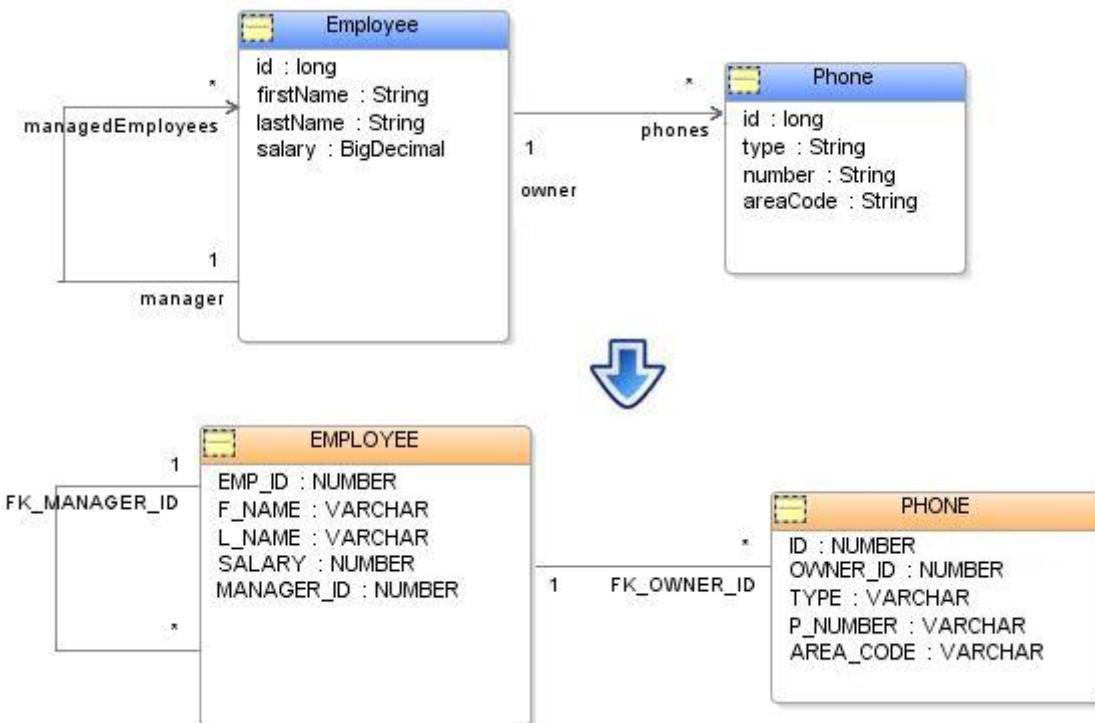
```
@ManyToOne(fetch=FetchType.LAZY)
@JoinTable(name = "owner_phone")
private Employee owner;
```

source : https://en.wikibooks.org/wiki/Java_Persistence/ManyToOne

```

@Entity
public class Phone {
    @Id
    private long id;
    ...
    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="OWNER_ID")
    private Employee owner;
    ...
}

```



61. @ManyToMany

- dowolna strona może być właściwem relacji

source: https://en.wikibooks.org/wiki/Java_Persistence/ManyToMany

```
@Entity
public class Employee {
    @Id
    @Column(name="ID")
    private long id;
    ...
    @ManyToMany
    @JoinTable(
        name="EMP_PROJ",
        joinColumns=@JoinColumn(name="EMP_ID", referencedColumnName="ID"),
        inverseJoinColumns=@JoinColumn(name="PROJ_ID", referencedColumnName="ID"))
    private List<Project> projects;
    ...
}
```

61.1. Dwukierunkowa

```
@Entity
public class Project {
    @Id
    @Column(name="ID")
    private long id;
    ...
    @ManyToMany(mappedBy="projects")
    private List<Employee> employees;
    ...
}
```

61.2. Dodatkowa kolumna w tabeli związku :

```
@Entity
public class Employee {
    @Id
    private long id;
    ...
    @OneToMany(mappedBy="employee")
    private List<ProjectAssociation> projects;
    ...
}
```

```

@Entity
public class Project {
    @Id
    private long id;
    ...
    @OneToMany(mappedBy="project")
    private List<ProjectAssociation> employees;
    ...
    // Add an employee to the project.
    // Create an association object for the relationship and set its data.
    public void addEmployee(Employee employee, boolean teamLead) {
        ProjectAssociation association = new ProjectAssociation();
        association.setEmployee(employee);
        association.setProject(this);
        association.setEmployeeId(employee.getId());
        association.setProjectId(this.getId());
        association.setIsTeamLead(teamLead);

        this.employees.add(association);
        // Also add the association object to the employee.
        employee.getProjects().add(association);
    }
}

```

```

@Entity
@Table(name="PROJ_EMP")
@IdClass(ProjectAssociationId.class)
public class ProjectAssociation {
    @Id
    private long employeeId;
    @Id
    private long projectId;
    @Column(name="IS_PROJECT_LEAD")
    private boolean isProjectLead;
    @ManyToOne
    @PrimaryKeyJoinColumn(name="EMPLOYEEID", referencedColumnName="ID")

    private Employee employee;
    @ManyToOne
    @PrimaryKeyJoinColumn(name="PROJECTID", referencedColumnName="ID")
    private Project project;
    ...
}

```

```

public class ProjectAssociationId implements Serializable {

    private long employeeId;

    private long projectId;

    public int hashCode() {
        return (int)(employeeId + projectId);
    }

    public boolean equals(Object object) {
        if (object instanceof ProjectAssociationId) {
            ProjectAssociationId otherId = (ProjectAssociationId) object;
            return (otherId.employeeId == this.employeeId) && (otherId.projectId == this
.projectId);
        }
        return false;
    }

}

```

62. Kaskadowość

- korzystamy z adnotacji **@CascadeType**
 - ALL (agreguje wszystkie poniższe)
 - DETACH (jeśli encja macieżysta została odłączona od kontekstu wtedy również encja związana jest także odłączana)
 - MERGE (jeśli encja macieżysta została przyłączona do kontekstu również encje powiązane zostaną przyłączone)
 - PERSIST (jeśli encja macieżysta została utrwalona to samo stanie się z encjami przynależnymi)
 - REFRESH (jeśli encja macieżysta została odświeżona w kontekście trwałości również encje powiązane zostaną odświeżone)
 - REMOVE (jeśli encja macieżysta została usunięta z kontekstu trwałości to również encje powiązane zostaną usunięte)

63. Usuwanie sierot - orphanRemoval

Kaskadowe usunięcie tych encji składowych, które usunięto z kolekcji encji głównej

- Przykład

```

@Test
public void orphanRemovalTest() {
    Long id = createLibrary();

    Session session = SessionUtil.getSession();
    Transaction tx = session.beginTransaction();

    Library library = (Library) session.load(Library.class, id);
    assertEquals(library.getBooks().size(), 3);

    library.getBooks().remove(0);
    assertEquals(library.getBooks().size(), 2);

    tx.commit();
    session.close();

    session = SessionUtil.getSession();
    tx = session.beginTransaction();

    Library l2 = (Library) session.load(Library.class, id);
    assertEquals(l2.getBooks().size(), 2);
    Query query = session.createQuery("from Book b");
    List books = query.list();
    assertEquals(books.size(), 2);

    tx.commit();
    session.close();
}

@Entity
public class Library {

    @OneToMany(orphanRemoval = true, mappedBy = "library")
    List<Book> books = new ArrayList<>();
}

@Entity
public class Book {

    @ManyToOne
    Library library;
}

```

64. Pobieranie encji

- `load()`
 - Na podstawie danego Id metoda `load` próbuje pobrać obiekt z bazy danych. Jeśli obiekt nie istnieje wyrzucany jest wyjątek `org.hibernate.ObjectNotFoundException` z metoda `load()`

zwraca też proxy, oznacza to tyle, że nie nastąpi uderzenie do bazy danych do czasu kiedy faktycznie będziemy potrzebować danego obiektu. Proxy zwraca dummy object zamiast uderzyć do db. Jeśli obiekt jest w first-level cache zwróci obiekt. Jeśli obiektu nie ma w first-level-cache uderzy do bazy.

```
public Object load(Class theClass, Serializable id) throws HibernateException  
public Object load(String entityName, Serializable id) throws HibernateException  
public void load(Object object, Serializable id) throws HibernateException
```

- `get()`

- Na podstawie danego Id metoda `get()` próbuje pobrać obiekt z bazy danych. Jeśli obiekt nie istnieje zwraca null. Metoda `get()` w przeciwieństwie do metody `load()` uderza do bazy bezpośrednio.

```
public Object get(Class clazz, Serializable id) throws HibernateException  
public Object get(String entityName, Serializable id) throws HibernateException  
public Object get(Class clazz, Serializable id, LockMode lockMode) throws  
HibernateException  
public Object get(String entityName, Serializable id, LockMode lockMode) throws  
HibernateException
```

przykład :

```
Book book = (Book) session.load(Book.class, isbn);  
Book book = (Book) session.get(Book.class, isbn);
```

65. SAVE/PERSIST

`public Serializable save(Object object) throws HibernateException`

`public Serializable save(String entityName, Object object) throws HibernateException`



Hibernate zrobi zapis do bazy danych w dowolnym momencie, tzn operacja save nie jest natychmiastowo przekładana na bazę.

66. MERGE vs UPDATE (stackoverflow)

Merge Does Following

Merge has intelligence. It has lot of pre-checks before it go actual merge(if required)

if Object is transient, It simply fires INSERT query makes object persistent(attached to session)

if Object is detached, fires select query to check whether data modified or not if modified, fires UPDATE query otherwise just ignore merge task.

where as session.update

throws exception if object is transient.

if Object is detached, it simply fires UPDATE query irrespective of data changes to object.

session.merge is expensive than update

saveOrUpdate() .

- Hibernate aktualnia rekord o danym Id. Jeśli Id nie istnieje w bazie tworzony jest nowy rekord.

67. Trwałość przechodnia

Technika umożliwiająca automatyczną propagację trwałości na podgrupy ulotne i odłączone.

68. Trwałość przez osiągalność

Jeśli z obiektu trwałego dostajemy się do innego obiektu poprzez referencje. Działanie jest rekurencyjne. Dowolny graf obiektów można odtworzyć w całości wczytując jego korzeń.



Pomaga uzyskać trwałość obiektom ulotnym i propagować je do bazy danych bez potrzeby wywołań zarządcy trwałości save()

69. Trwałość kaskadowa

Koncepcja podobna do trwałości przez osiągalność Powiązania są odtwarzane na podstawie asocjacji. Sprawdza asocjacje by określić stan trwałości obiektów.



Do sterowania trwałością kaskadową używamy anotacji **@CascadeType** na kolekcji lub relacji

70. Zapytania

71. Kwerandy

72. Możliwe formy zapytań :

72.1. zapytania HQL, JPQL

- Odpowiednik SQL
- zamiast tabel i pól operuje na obiektach i jego właściwościach.
- przenośny zależny od dostawcy

72.2. Zapytania dynamiczne

- Zalety :
 - nie wiemy jak zapytanie do końca ma wyglądać, w czasie runtime zbierane są dane i tworzone jest zapytanie
- Wady :
 - Koszt związany z tłumaczeniem JPQL na SQL za każdym razem kiedy zapytanie jest wywoływane
 - Większość providers próbuje cache'ować SQL → dynamiczne zapytania. Nie zawsze to działanie osiąga sukces

```
String query = "SELECT d.location " +  
    "FROM Department d " +  
    "WHERE d.name = '" + deptName +  
    "' AND d.company.name = '" + companyName + "'";  
return em.createQuery(query, Long.class).getSingleResult();  
}
```

Konkadenacji podlegają deptName i companyName. Za każdym razem kiedy zapytanie jest wywoływane nowe query jest na nowo generowane



W przypadku dużej liczby takich zapytań mogą wystąpić problemy z wydajnością. Konkadenowany String to nowy obiekt → więcej cykli procesora oraz cykli GC.

72.3. Możliwe typy wników kwerend:

- Typy bazowe : String, int, boolean, JDBC types, etc

- Przykład

```
List<String> lastNames = session.createQuery("SELECT e.firstName FROM "+Employee.class.getSimpleName().list());
```

- Encje

- Przykład

```
TypedQuery<Person> typedQuery = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where p.name like :name", Person.class
);
```

- Tablice obiektów

- Przykład

```
List result = em.createQuery("SELECT e.name, e.department.name FROM Project p JOIN
p.employees e WHERE p.name = ?1 ORDER BY e.name")
.setParameter(1, projectName)
.getResultList();
int count = 0;
for (Iterator i = result.iterator(); i.hasNext();) {
Object[] values = (Object[]) i.next();
...
}
```

- Tuple

- Przykład

```
List<Object[]> tuples = entityManager.createNamedQuery(
"find_person_with_phones_by_name" ).setParameter("name", "%").getResultList();

for(Object[] tuple : tuples) {
    Person person = (Person) tuple[0];
    Phone phone = (Phone) tuple[1];
}
```

- DTO generowane przez konstruktor

- Przykład

```
String sql = "select pl.java.scalatech.BookDTO(b.name, b.id, b.isbn) from Book b";
Query query = entityManager.createQuery(sql);
List<BookDTO> books = query.getResultList();
```

- Przykład

```
Query query = session.createQuery(sql);
```

72.4. Criteria API

72.5. native SQL

72.6. Hibernate Criteria

- Przykład

```
Criteria criteria = session.createCriteria(Employee.class);
criteria.add(Restrictions.eq("department.name", "account"));
List list = criteria.list();
```

73. SELECT

SELECT <select_expression> FROM <from_clause>

74. Projekcja / Projection

- zwracanie tylko interesujących użytkownika zapytań
- redukcja użycia pamięci
- zwiększoa szybkość wykonania
- zwiększoa szybkość przetwarzania
 - Przykład

```
String sql = "select b.name from Book b";
sql = "select b.id, b.name , b.isbn from Book c";
Query query = entityManager.createQuery(sql);
List<Object[]> books = query.getResultList();
```

74.1. SQL result-mapping

- Przykład

```
org.hibernate.SQLQuery query = session.createSQLQuery("select {i.*} from ITEM {i}")
    .addEntity("i", Item.class);
```

74.2. Projekcja z użyciem konstruktora

- Przykład

```
String sql = "select pl.java.scalatech.BookDTO(b.name, b.id, b.isbn) from Book b";
Query query = entityManager.createQuery(sql);
List<BookDTO> books = query.getResultList();
```

75. Restriction / where – zawężamy

- Przykład

```
String sql = "from Book where name='Qua vadis'";
sql = "from Book where name like '%Qu'";
sql = "from Book where price > 40";

Query query = session.createQuery(sql);
List<Book> books = query.getResultList();
```

75.1. Like

- Przykład

```
try {
    Query query = em.createQuery(
        "select p from Person p where firstName like '%a%'"
    );
    Person person = (Person) query.getSingleResult();
    // ...
} catch (NonUniqueResultException ex) {
    // ...
}
```

76. Zwiększenie wydajności poprzez agregacje po stronie bazy

77. Natywna kwerenda SQL

- Przykład 1

```
SQLQuery sqlQuery = session.createSQLQuery("SELECT * FROM product");
List<Object[]> list = sqlQuery.list();
for(Object[] object : list){
    System.out.println("\nId: " + object[0]);
    System.out.println("Name: " + object[1]);
    System.out.println("Price: " + object[2]);
    System.out.println("Category id: " + object[3]);
}

SQLQuery sqlQuery = session.createSQLQuery("SELECT id, name, price,
category_id FROM product");
sqlQuery.addScalar("id", new org.hibernate.type.LongType());
sqlQuery.addScalar("name", new org.hibernate.type.StringType());
sqlQuery.addScalar("price", new org.hibernate.type.DoubleType());
sqlQuery.addScalar("category_id", new
org.hibernate.type.LongType());
sqlQuery.setResultTransformer(Transformers.ALIAS_TO_ENTITY_MAP);
List list = sqlQuery.list();

List<Object[]> persons = entityManager.createNativeQuery(
    "SELECT * FROM person" )
.getResultList();
```

- Przykład 2

```
SQLQuery sqlQuery = session.createSQLQuery("SELECT * FROM category");
sqlQuery.addEntity(Category.class);
List<Category> list = sqlQuery.list();
for(Category category: list){
    System.out.println("\nCategory id: " + category.getId());
    System.out.println("Category name: " + category.getName());
}
```

- Przykład 3

```
List<String> lastNames = session.createQuery("SELECT e.firstName FROM "+Employee.class.getSimpleName()
+ " e e.company c join c.depts d WHERE d.name = :name")
.setParameter("name", "JAVA").list();
og.info("lastNames {}", lastNames);
```

77.1. Natywne / NATIVE

- Przykład

```
List<Object[]> persons = entityManager.createNativeQuery(
    "SELECT id, name FROM person" )
.getResultList();

for(Object[] person : persons) {
    BigInteger id = (BigInteger) person[0];
    String name = (String) person[1];
}
```

78. HQL kwerendy

- Przykład

```
Query query = session.createQuery("FROM Category");
List<Category> list = query.list();
System.out.println("Category size: " + list.size());

Query query = session.createQuery("From Category, Product");
List list = query.list();
System.out.println("Result size: " + list.size());

Query query = session.createQuery("SELECT id, name from Category");
List list = query.list();
System.out.println("Result size: " + list.size());
```

78.1. JPA natywne kwerendy

- Przykład

```
List<Person> persons = entityManager.createNativeQuery("SELECT * FROM person", Person.class ).getResultList();
```

78.2. Natywne kwerendy z aliasami

- Przykład

```
List<Object> entities = session.createSQLQuery(  
    "SELECT {pr.*}, {pt.*} " +  
    "FROM person pr, partner pt " +  
    "WHERE pr.name = pt.name" )  
.addEntity( "pr", Person.class)  
.addEntity( "pt", Partner.class)  
.list();
```

79. Zapytania nazywane / NamedQuery

W celu wygodniejszego używania oraz większej wydajności korzysta się tzw nazwanych zapytań.

Persistence Provider bedzie konwertował named query z JPQL do SQL podczas deploymentu i będzie cache'ował 'poźniej'. Konkadenacja będzie miała tylko narzut podczas deploymentu.

- prekompilacja
- powiązanie z encją
- sprawdzane podczas deployment'u
- łatwiejsze do przeczytania i utrzymania
- zysk wydajnościowy ((model programowy



Unikalne w ramach **Persistence Unit**

@NamedQuery : pojedyńcze natywne zapytanie
@NamedQueries : agregacja kilku natywnych zapytań

- Przykład

```
@NamedQuery(name="getCategoryNameByName", query="FROM Category c WHERE c.name=:name")  
  
session.getNamedQuery("getCategoryNameByName");  
  
@NamedQueries(  
{  
    @NamedQuery(  
        name="getCategoryNameByName",  
        query="FROM Category c WHERE c.name=:name"  
    ),  
    @NamedQuery(  
        name="getCategoryNameById",  
        query="FROM Category c WHERE c.id=:id"  
    ),  
}  
)
```



Zapytania nazwane umieszcza się na klasie encyjnej

- Przykład

```
@NamedQueries({  
    @NamedQuery(name="Company.findAll",query="SELECT c FROM Company c"),  
    @NamedQuery(name="Company.findByPrimaryKey", query="SELECT c FROM Company c WHERE c.id  
= :id")})  
Query q = entityManager.getNamedQuery("Company.findAll");
```

79.1. Natywne w konfiguracja z JOIN

- Przykład

```

@NamedNativeQuery(
    name = "find_person_with_phones_by_name",
    query =
        "SELECT " +
        "    pr.id AS \"pr.id\", " +
        "    pr.name AS \"pr.name\", " +
        "    pr.nickName AS \"pr.nickName\", " +
        "    pr.address AS \"pr.address\", " +
        "    pr.createdOn AS \"pr.createdOn\", " +
        "    pr.version AS \"pr.version\", " +
        "    ph.id AS \"ph.id\", " +
        "    ph.person_id AS \"ph.person_id\", " +
        "    ph.number AS \"ph.number\", " +
        "    ph.type AS \"ph.type\" " +
        "FROM person pr " +
        "JOIN phone ph ON pr.id = ph.person_id " +
        "WHERE pr.name LIKE :name",
    resultSetMapping = "person_with_phones"
)
@SqlResultSetMapping(
    name = "person_with_phones",
    entities = {
        @EntityResult(
            entityClass = Person.class,
            fields = {
                @FieldResult( name = "id", column = "pr.id" ),
                @FieldResult( name = "name", column = "pr.name" ),
                @FieldResult( name = "nickName", column = "pr.nickName" ),
                @FieldResult( name = "address", column = "pr.address" ),
                @FieldResult( name = "createdOn", column = "pr.createdOn" ),
                @FieldResult( name = "version", column = "pr.version" ),
            }
        ),
        @EntityResult(
            entityClass = Phone.class,
            fields = {
                @FieldResult( name = "id", column = "ph.id" ),
                @FieldResult( name = "person", column = "ph.person_id" ),
                @FieldResult( name = "number", column = "ph.number" ),
                @FieldResult( name = "type", column = "ph.type" ),
            }
        )
    }
),
),
)

```

79.2. Zapytania nazywane podejście programistyczne

- Przykład

```
Query findPersonQuery = em.createQuery("select p from Person p");
em.getEntityManagerFactory().addNamedQuery("personQuery", findPersonQuery);
Query query =
em.createNamedQuery("personQuery");
```

79.3. Tuple

- Przykład

```
List<Object[]> tuples = entityManager.createNamedQuery(
"find_person_with_phones_by_name" ).setParameter("name", "J%").getResultList();

for(Object[] tuple : tuples) {
    Person person = (Person) tuple[0];
    Phone phone = (Phone) tuple[1];
}
```

80. FROM

- Przykład

```
String sql = "from Book";
sql = "from Book b";
sql = "from Book as book";
sql = "pl.java.scalatech.Book";

Query query = session.createQuery(sql);
List<Book> books = query.getResultList();
```

- Przykład

```
Query query = entityManager.createQuery(
"select p " +
"from Person p " +
"where p.name like :name"
);

TypedQuery<Person> typedQuery = entityManager.createQuery(
"select p " +
"from Person p " +
"where p.name like :name", Person.class
);
```

80.1. Native

- Przykład

```
@NamedQueries(  
    @NamedQuery(  
        name = "get_person_by_name",  
        query = "select p from Person p where name = :name"  
    )  
)  
  
Query query = entityManager.createNamedQuery( "get_person_by_name" );  
  
TypedQuery<Person> typedQuery = entityManager.createNamedQuery(  
    "get_person_by_name", Person.class  
);
```

80.2. Hint

- Przykład

```
Query query = entityManager.createQuery(  
    "select p " +  
    "from Person p " +  
    "where p.name like :name" )  
// timeout - in milliseconds  
.setHint( "javax.persistence.query.timeout", 2000 )  
// flush only at commit time  
.setFlushMode( FlushModeType.COMMIT );
```

80.2.1. javax.persistence.query.timeout

- definujemy timeout dla kwerendy

80.2.2. javax.persistence.fetchgraph

- definujemy EntityGraph

80.2.3. org.hibernate.cacheMode

- definujemy rodzaj buforowania

80.2.4. org.hibernate.cacheable

- definujemy czy kewrenda ma być buforowana

80.2.5. org.hibernate.cacheRegion

- definujemy nazwę regionu bufora

80.2.6. org.hibernate.comment

- oznaczamy opisowo daną kwerende

80.2.7. org.hibernate.fetchSize

- definujemy fetchSize

80.2.8. org.hibernate.flushMode

- definujemy flushMode dla kwerendy

80.2.9. org.hibernate.readOnly

- definujemy czy interesujący nas obiekt ma być tylko do odczytu

80.3. Timestamp

- Przykład

```
Query query = entityManager.createQuery(  
    "select p " +  
    "from Person p " +  
    "where p.createdOn > :timestamp" )  
.setParameter( "timestamp", timestamp, TemporalType.DATE );
```

80.4. Konkadenacja

- Przykład

```
String name = entityManager.createQuery(  
    "select 'Customer ' || p.name " +  
    "from Person p " +  
    "where p.id = 1", String.class )  
.getSingleResult();
```

80.5. Porównania

- Przykład

```

// numeric comparison
List<Call> calls = entityManager.createQuery(
    "select c " +
    "from Call c " +
    "where c.duration < 30 ", Call.class )
.getResultList();

// string comparison
List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where p.name like 'John%' ", Person.class )
.getResultList();

// datetime comparison
List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where p.createdOn > '1950-01-01' ", Person.class )
.getResultList();

// enum comparison
List<Phone> phones = entityManager.createQuery(
    "select p " +
    "from Phone p " +
    "where p.type = 'MOBILE' ", Phone.class )
.getResultList();

// boolean comparison
List<Payment> payments = entityManager.createQuery(
    "select p " +
    "from Payment p " +
    "where p.completed = true ", Payment.class )
.getResultList();

// boolean comparison
List<Payment> payments = entityManager.createQuery(
    "select p " +
    "from Payment p " +
    "where type(p) = WireTransferPayment ", Payment.class )
.getResultList();

// entity value comparison
List<Object[]> phonePayments = entityManager.createQuery(
    "select p " +
    "from Payment p, Phone ph " +
    "where p.person = ph.person ", Object[].class )
.getResultList();

```

80.6. Like

- Przykład

```
Query query = entityManager.createQuery(  
    "select p " +  
    "from Person p " +  
    "where p.name like ?1" )  
.setParameter( 1, "J%" );
```

81. Kwerenda z wielu podmiotów

- Przykład 1

```
List<Object[]> persons = entityManager.createQuery(  
    "select distinct pr, ph " +  
    "from Person pr, Phone ph " +  
    "where ph.person = pr and ph is not null", Object[].class)  
.getResultList();  
  
List<Person> persons = entityManager.createQuery(  
    "select distinct pr1 " +  
    "from Person pr1, Person pr2 " +  
    "where pr1.id <> pr2.id " +  
    " and pr1.address = pr2.address " +  
    " and pr1.createdOn < pr2.createdOn", Person.class )  
.getResultList();
```

- Przykład 2

```

public class CallStatistics {

    private final long count;
    private final long total;
    private final int min;
    private final int max;
    private final double abg;

    public CallStatistics(long count, long total, int min, int max, double abg) {
        this.count = count;
        this.total = total;
        this.min = min;
        this.max = max;
        this.abg = abg;
    }

    //Getters and setters omitted for brevity
}

CallStatistics callStatistics = entityManager.createQuery(
    "select new org.hibernate.userguide.hql.CallStatistics(" +
    "    count(c), " +
    "    sum(c.duration), " +
    "    min(c.duration), " +
    "    max(c.duration), " +
    "    avg(c.duration)" +
    ") " +
    "from Call c ", CallStatistics.class )
.getSingleResult();

```

82. Dynamiczna instancja - przykład

- Przykład

```

List<List> phoneCallDurations = entityManager.createQuery(
    "select new list(" +
    "    p.number, " +
    "    c.duration " +
    ") " +
    "from Call c " +
    "join c.phone p ", List.class )
.getResultList();

```

83. Dynamiczna mapa - przykład

- Przykład

```
List<Map> phoneCallTotalDurations = entityManager.createQuery(  
    "select new map(" +  
    "    p.number as phoneNumber , " +  
    "    sum(c.duration) as totalDuration, " +  
    "    avg(c.duration) as averageDuration " +  
    ") " +  
    "from Call c " +  
    "join c.phone p ", Map.class )  
.getResultList();
```

84. Where

- Przykład

```
SELECT DISTINCT d FROM Department d, Employee e WHERE d = e.department
```

85. Parametryzacja

- Przykład

```
Query query = em.createQuery("select i from Item i where i.auctionEnd > :endDate")  
.setParameter("endDate", tomorrowDate, TemporalType.TIMESTAMP);
```



Zabezpiecza przed SQL injection

86. Wstawienie przez kwerende

- Przykład

```
int insertedEntities = session.createQuery(  
    "insert into Partner (id, name) " +  
    "select p.id, p.name " +  
    "from Person p ")  
.executeUpdate();
```

86.1. Dopasowane operacje : Insert, Update , Delete

- Przykład

```

@Entity
@Table(name = "CHAOS")
@SQLInsert( sql = "INSERT INTO CHAOS(size, name, nickname, id) VALUES(?,upper(?),?,?)")
@SQLUpdate( sql = "UPDATE CHAOS SET size = ?, name = upper(?), nickname = ? WHERE id = ?")
@SQLDelete( sql = "DELETE CHAOS WHERE id = ?")
@SQLDeleteAll( sql = "DELETE CHAOS")
@Loader(namedQuery = "chaos")
@NamedNativeQuery(name = "chaos", query="select id, size, name, lower( nickname ) as nickname from CHAOS where id= ?", resultClass = Chaos.class)
public class Chaos {
    @Id
    private Long id;
    private Long size;
    private String name;
    private String nickname;

```

86.2. Przykrywanie operacji na kolekcjach przez adnotacje

- Przykład

```

@OneToMany
@JoinColumn(name = "chaos_fk")
@SQLInsert( sql = "UPDATE CASIMIR_PARTICULE SET chaos_fk = ? where id = ?")
@SQLDelete( sql = "UPDATE CASIMIR_PARTICULE SET chaos_fk = null where id = ?")
private Set<CasimirParticle> particles = new HashSet<CasimirParticle>();

```

87. UPDATE

- Przykład

```

int updatedEntities = entityManager.createQuery("update Person p set p.name = :newName
where p.name = :oldName" )
.setParameter( "oldName", oldName )
.setParameter( "newName", newName )
.executeUpdate();

```

87.1. Bulk update

- Przykład

```
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
public void setManager(Department dept, Employee manager) {
    em.createQuery("UPDATE Employee e " + "SET e.manager = :name " + "WHERE e.department =
:dept")
    .setParameter("name", "przodownik")
    .setParameter("dept", "JAVA")
    .executeUpdate();
}
```

88. Delete

- Przykład

```
int deletedEntities = entityManager.createQuery("delete Person p where p.name = :name")
.setParameter("name", name).executeUpdate();
```

```
Query query=session.createQuery("delete from Employee where status=:status");
query.setString("status", "fired");
int rowsDeleted=query.executeUpdate();
```

89. Distinct

- Przykład

```
SELECT DISTINCT mag FROM Magazine AS mag JOIN mag.articles AS art WHERE art.published
= FALSE
```

90. Between

- Przykład

```

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "join p.phones ph " +
    "where p.id = 1L and index(ph) between 0 and 3", Person.class )
.getResultList();

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where p.createdOn between '1999-01-01' and '2001-01-02'", Person.class )
.getResultList();

List<Call> calls = entityManager.createQuery(
    "select c " +
    "from Call c " +
    "where c.duration between 5 and 20", Call.class )
.getResultList();

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where p.name between 'H' and 'M'", Person.class )
.getResultList();

```

91. IS [NOT] EMPTY

- Przykład

```

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where p.phones is empty", Person.class )
.getResultList();

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where p.phones is not empty", Person.class )
.getResultList();

```

92. [NOT] MEMBER [OF]

- Przykład

```

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where 'Home address' member of p.addresses", Person.class )
.getResultList();

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where 'Home address' not member of p.addresses", Person.class )
.getResultList();

```

93. Podzapytania

- Przykład

```
SELECT b FROM Book b WHERE b.price = (SELECT MAX(emp.salary) FROM Employee emp)
```

94. IN

- Przykład

```
SELECT FROM Person p WHERE p.sex IN ('MALE', 'FEMALE')
```

95. Operacje na kolekcjach

```

SELECT e FROM Company c WHERE c.products IS NOT EMPTY
SELECT e FROM Company c WHERE :project MEMBER OF c.products

```

96. Sortowanie

- Przykład 1

```

String sql = "from Book b order by b.name asc";
Query query = entityManager.createQuery(sql);
List<Book> books = query.getResultList();

```

- Przykład 2

```

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "order by p.name", Person.class )
.getResultList();

List<Object[]> personTotalCallDurations = entityManager.createQuery(
    "select p.name, sum( c.duration ) as total " +
    "from Call c " +
    "join c.phone ph " +
    "join ph.person p " +
    "group by p.name " +
    "order by total", Object[].class )
.getResultList();

```

97. Agregacje

97.1. AVG

- Przykład

```

Object[] callStatistics = entityManager.createQuery(
    "select " +
    "    count(c), " +
    "    sum(c.duration), " +
    "    min(c.duration), " +
    "    max(c.duration), " +
    "    avg(c.duration) " +
    "from Call c ", Object[].class )
.getSingleResult();

```

97.2. COUNT

- Przykład

```
Long phoneCount = entityManager.createQuery(  
    "select count( distinct c.phone ) " +  
    "from Call c ", Long.class )  
.getSingleResult();  
  
List<Object[]> callCount = entityManager.createQuery(  
    "select p.number, count(c) " +  
    "from Call c " +  
    "join c.phone p " +  
    "group by p.number", Object[].class )  
.getResultList();
```

97.3. MAX

- Przykład

```
SELECT d, COUNT(e), MAX(e.salary), AVG(e.salary) FROM Department d JOIN d.employees e  
GROUP BY d HAVING COUNT(e) >= 5
```

97.4. MIN

97.5. SUM

98. GROUP BY

- Przykład 1

```
SELECT d.name, COUNT(e)  
FROM Department d JOIN d.employees e  
GROUP BY d.name
```

- Przykład 2

```

Long totalDuration = entityManager.createQuery(
    "select sum( c.duration ) " +
    "from Call c ", Long.class )
.getSingleResult();

List<Object[]> personTotalCallDurations = entityManager.createQuery(
    "select p.name, sum( c.duration ) " +
    "from Call c " +
    "join c.phone ph " +
    "join ph.person p " +
    "group by p.name", Object[].class )
.getResultList();

//It's even possible to group by entities!
List<Object[]> personTotalCallDurations = entityManager.createQuery(
    "select p, sum( c.duration ) " +
    "from Call c " +
    "join c.phone ph " +
    "join ph.person p " +
    "group by p", Object[].class )
.getResultList();

```

99. HAVING

- Przykład

```

List<Object[]> personTotalCallDurations = entityManager.createQuery(
    "select p.name, sum( c.duration ) " +
    "from Call c " +
    "join c.phone ph " +
    "join ph.person p " +
    "group by p.name " +
    "having sum( c.duration ) > 1000", Object[].class )
.getResultList();

```

100. Stronicowanie

- Przykład

```

String sql = "from Book";
Query query = entityManager.createQuery(sql);
query.setFirstResult(10);
query.setMaxResults(25);
List<Book> books = query.getResultList();

```

101. Pobieranie pojedyńczego wyniku

- Przykład

```
String sql = "from Book b where b.id=:id";
Query query = entityManager.createQuery(sql);
query.setLong("id", 1);
Book book = (Book)query.getSingleResult();
```



org.hibernate.NonUniqueResultException gdy metoda zwróci więcej niż jednego obiektu



Zwraca pojedyńczy obiekt lub **null** jeśli takiego obiektu nie ma w bazie

```
String hql = "from Product where price > 21.0";
Query query = session.createQuery(hql);
query.setMaxResults(1);
Product product = (Product) query.uniqueResult();
```

102. JOIN

- Przykład 1

```
SELECT d FROM Employee e JOIN e.department d
```

- Przykład 2

```
List<Person> persons = entityManager.createQuery(
    "select distinct pr " +
    "from Person pr " +
    "join pr.phones ph " +
    "where ph.type = :phoneType", Person.class )
.setParameter( "phoneType", PhoneType.MOBILE )
.getResultList();

// same query but specifying join type as 'inner' explicitly
List<Person> persons = entityManager.createQuery(
    "select distinct pr " +
    "from Person pr " +
    "inner join pr.phones ph " +
    "where ph.type = :phoneType", Person.class )
.setParameter( "phoneType", PhoneType.MOBILE )
.getResultList();
```

102.1. Join niejawny

- Przykład

```
SELECT p.number FROM Employee e, Phone p WHERE e = p.employee AND e.department.name = 'JAVA' AND p.type = 'MOBILE'
```

102.2. Wielokrotny Join

- Przykład

```
SELECT DISTINCT p FROM Department d JOIN d.employees e JOIN e.projects p
```

103. JOIN LEFT

- Przykład

```
List<Person> persons = entityManager.createQuery(  
    "select distinct pr " +  
    "from Person pr " +  
    "left join pr.phones ph " +  
    "where ph is null " +  
    " or ph.type = :phoneType", Person.class )  
.setParameter( "phoneType", PhoneType.LAND_LINE )  
.getResultList();  
  
// functionally the same query but using the 'left outer' phrase  
List<Person> persons = entityManager.createQuery(  
    "select distinct pr " +  
    "from Person pr " +  
    "left outer join pr.phones ph " +  
    "where ph is null " +  
    " or ph.type = :phoneType", Person.class )  
.setParameter( "phoneType", PhoneType.LAND_LINE )  
.getResultList();
```

```
List<Object[]> personsAndPhones = session.createQuery(  
    "select pr.name, ph.number " +  
    "from Person pr " +  
    "left join pr.phones ph with ph.type = :phoneType " )  
.setParameter( "phoneType", PhoneType.LAND_LINE )  
.list();
```

104. JOIN FETCH

- Przykład

```
List<Person> persons = entityManager.createQuery(  
    "select distinct pr " +  
    "from Person pr " +  
    "left join fetch pr.phones ", Person.class )  
.getResultList();
```

105. Wyrażenie IN

- Przykład 1

```
SELECT e FROM Employee e WHERE e.phones.type IN ('MOBILE', 'HOME')
```

- Przykład 2

```
SELECT e FROM Employee e WHERE e.department IN (SELECT DISTINCT d  
FROM Department d JOIN d.employees de JOIN de.projects p  
WHERE p.name LIKE 'VA%')
```

106. JPQL wspieranie standardów

106.1. CONCAT - łącznie dwóch lub większej ilości stringów

- Przykład

```
List<String> callHistory = entityManager.createQuery(  
    "select concat( p.number, ' : ', c.duration ) " +  
    "from Call c " +  
    "join c.phone p", String.class )  
.getResultList();
```

106.2. SUBSTRING - wycinanie części stringa z danego ciągu znaków

- Przykład

```
List<String> prefixes = entityManager.createQuery(  
    "select substring( p.number, 0, 2 ) " +  
    "from Call c " +  
    "join c.phone p", String.class )  
.getResultList();
```

106.3. UPPER - zamiana na duże litery

- Przykład

```
List<String> names = entityManager.createQuery(  
    "select upper( p.name ) " +  
    "from Person p", String.class )  
.getResultList();
```

106.4. LOWER - zamiana na małe litery

- Przykład

```
List<String> names = entityManager.createQuery(  
    "select lower( p.name ) " +  
    "from Person p", String.class )  
.getResultList();
```

106.5. TRIM - usuwanie białych znaków

- Przykład

```
List<String> names = entityManager.createQuery(  
    "select trim( p.name ) " +  
    "from Person p", String.class )  
.getResultList();
```

106.6. LENGTH - obliczanie długości ciągu znaków

- Przykład

```
List<Integer> lengths = entityManager.createQuery(  
    "select length( p.name ) " +  
    "from Person p", Integer.class )  
.getResultList();
```

106.7. ABS - obliczanie wartości absolutnej

- Przykład

```
List<Integer> abs = entityManager.createQuery(  
    "select abs( c.duration ) " +  
    "from Call c ", Integer.class )  
.getResultList();
```

106.8. MOD - obliczanie reszty z dzielenia

- Przykład

```
List<Integer> mods = entityManager.createQuery(  
    "select mod( c.duration, 10 ) " +  
    "from Call c ", Integer.class )  
.getResultList();
```

106.9. SQRT - pierwiastek

- Przykład

```
List<Double> sqrts = entityManager.createQuery(  
    "select sqrt( c.duration ) " +  
    "from Call c ", Double.class )  
.getResultList();
```

106.10. CURRENT_DATE - bieżąca data

- Przykład

```
List<Call> calls = entityManager.createQuery(  
    "select c " +  
    "from Call c " +  
    "where c.timestamp = current_date", Call.class )  
.getResultList();
```

106.11. CURRENT_TIME - bieżący czas

- Przykład

```
List<Call> calls = entityManager.createQuery(  
    "select c " +  
    "from Call c " +  
    "where c.timestamp = current_time", Call.class )  
.getResultList();
```

106.12. CURRENT_TIMESTAMP - bieżaca data i czas z milisek

- Przykład

```
List<Call> calls = entityManager.createQuery(  
    "select c " +  
    "from Call c " +  
    "where c.timestamp = current_timestamp", Call.class )  
.getResultList();
```

107. HQL functions

107.1. CAST - rzutowanie

- Przykład

```
List<String> durations = entityManager.createQuery(  
    "select cast( c.duration as string ) " +  
    "from Call c ", String.class )  
.getResultList();
```

107.2. EXTRACT

- Przykład

```
List<Integer> years = entityManager.createQuery(  
    "select extract( YEAR from c.timestamp ) " +  
    "from Call c ", Integer.class )  
.getResultList();
```

107.3. YEAR

- Przykład

```
List<Integer> years = entityManager.createQuery(  
    "select year( c.timestamp ) " +  
    "from Call c ", Integer.class )  
.getResultList();
```

107.4. MONTH

- użycie analogiczne jak w przykładzie wyżej === DAY
- użycie analogiczne jak w przykładzie wyżej === HOUR
- użycie analogiczne jak w przykładzie wyżej === MINUTE
- użycie analogiczne jak w przykładzie wyżej === SECOND
- użycie analogiczne jak w przykładzie wyżej

108. Typ encji

- Przykład

```
List<Payment> payments = entityManager.createQuery(  
    "select p " +  
    "from Payment p " +  
    "where type(p) = CreditCardPayment", Payment.class )  
.getResultList();  
List<Payment> payments = entityManager.createQuery(  
    "select p " +  
    "from Payment p " +  
    "where type(p) = :type", Payment.class )  
.setParameter( "type", WireTransferPayment.class)  
.getResultList();
```

109. Case

- Przykład

```

List<String> nickNames = entityManager.createQuery(
    "select " +
    "  case p.nickName " +
    "  when 'NA' " +
    "  then '<no nick name>' " +
    "  else p.nickName " +
    "  end " +
    "from Person p", String.class )
.getResultList();

// same as above
List<String> nickNames = entityManager.createQuery(
    "select coalesce(p.nickName, '<no nick name>') " +
    "from Person p", String.class )
.getResultList();

```

110. Użycie konstruktora

- Przykład

```
SELECT NEW pl.java.scalatech.EmployeeReport(e.firstName, e.lastName, e.salary) FROM Employee e
```

111. Usuwanie obiektów z bazy

Poprzez wywołanie metody remove lub kaskadowej operacji remove. Jeśli metoda remove jest wywołana na encji odłączonej zostanie wygenerowany wyjątek **IllegalArgumentException** lub zatwierdzenie transakcji się nie powiedzie. Dane fizycznie zostaną usunięte w momencie zakończenia transakcji lub wykonania metody **flush()**

DELETE FROM employee WHERE id=1;

```
Book book = (Book) session.get(Book.class,new Long(1));
session.delete(book);
```

Metoda ta wyrzuci wyjątek jeśli obiekt o podanym identyfikatorze nie istnieje w bazie (java.lang.IllegalArgumentException)

112. Aktualizacja

UPDATE book SET title='jpa book' WHERE id=2;

113. Merge

- Merge jest odwrotną operacją do operacji refresh()
- Nadpisuje encje w bazie danych wartościami encji odłączonych.
 - Object merge(Object object)
 - Object merge(String entityName, Object object)
- Przykład

```
Session session = factory.openSession();
tx = session.beginTransaction();
Company company = (Company)session.get(Company.class, 1);
company.setName("Scalatech S.p z.o.o");
tx.commit();
session.flush();
session.close();
```

114. Odświeżanie encji (Refreshing Entities)

Metoda **reload** odświeża wartości dla encji wartościami z bazy danych. (odwrotność do merge)

```
public void refresh(Object object) throws HibernateException

public void refresh(Object object, LockMode lockMode) throws HibernateException
```

- Hibernate

```
Object merge(Object object)
```

```
Object merge(String entityName, Object object)
```

115. O mnie

- Architect Solution - RiscoSoftware
- JavaTech trener : Spring ekosystem, JPA , EIP Camel
- Sages trener : JPA , EIP - Apache Camel
- blog <http://przewidywalna-java.blogspot.com>
- twitter przodownikR1

[tUWf7KiC]

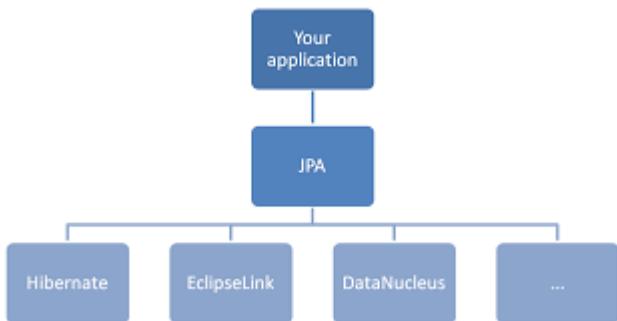
116. Geneza JPA



Java Persistence API (JPA) jest specyfikacją wydaną dla ujednolicenia różnych dostawców trwałości. - start odbył się w 2006 roku (wersja 1.0) - w 2013 roku podstała wersja aktualna t.j 2.1

117. Dostawcy trwałości

- **Hibernate** <http://hibernate.org/orm/>
- **EclipseLink** <http://www.eclipse.org/eclipselink/>
- **Open JPA** <http://openjpa.apache.org/>
- **TopLink** <http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>
- **Kodo** http://docs.oracle.com/html/E13946_01/toc.htm



118. Utrwalanie

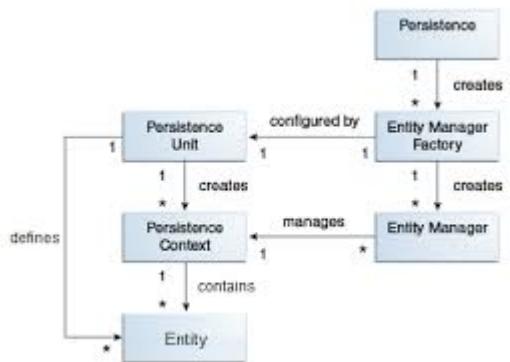
- **@Encje**
- **@MappedSuperClass**
- **@Embeddable**
- **@CollectionElement** (dla typów prostych lub klas osadzonych)

119. Użycie EntityManager'a [JPA]

119.1. Zależności

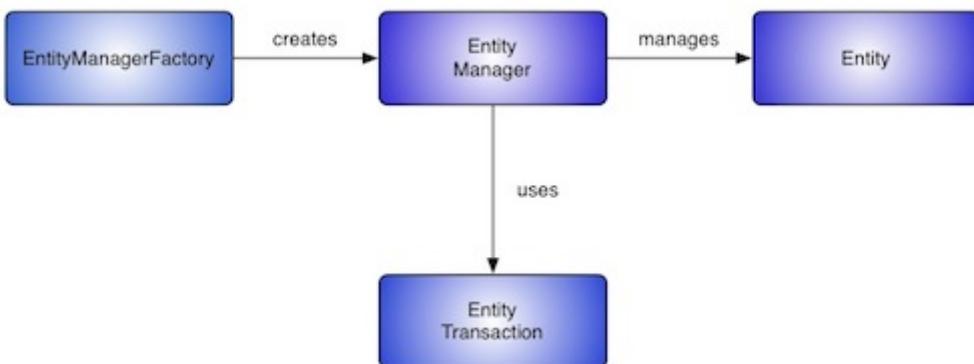
```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.5.Final</version>
</dependency>
```

119.2. Schemat zależności



119.3. EntityManagerFactory(JPA) = SessionFactory(Hibernate)

source : <http://www.java-forums.org/ocpjbcd/53005-tutorial-review-entitymanager-context-component-developer-exam.html>



Mожет być programowalny manualnie lub przy pomocy pliku persistence.xml, który musi znajdować się w classpath projektu.

Każdemu EntityManagerFactory odpowiada jedna jednostka trwałości (Persistence Unit)

119.3.1. Bootstrapping

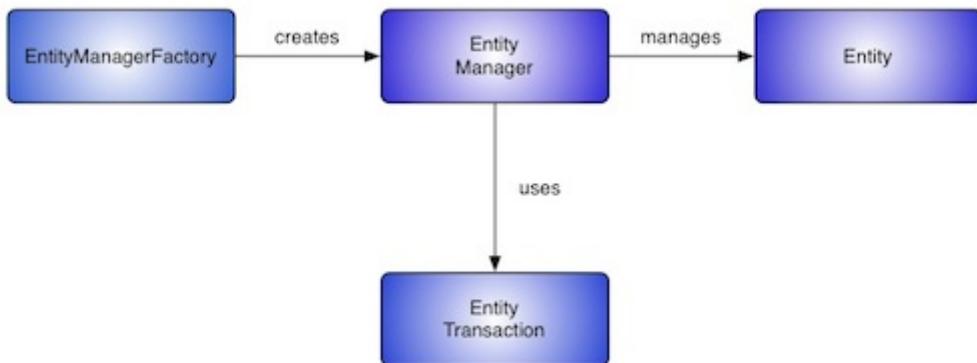
```

EntityManagerFactory emf = Persistence.createEntityManagerFactory("manager1");
//or
Map<String, Object> configOverrides = new HashMap<String, Object>();
configOverrides.put("hibernate.hbm2ddl.auto", "create-drop");
EntityManagerFactory programmaticEmf = Persistence.createEntityManagerFactory(
"manager1", configOverrides);

```

119.4. EntityManager

source: <http://www.java-forums.org/ocpjbcd/53005-tutorial-review-entitymanager-context-component-developer-exam.html>



119.4.1. Cechy

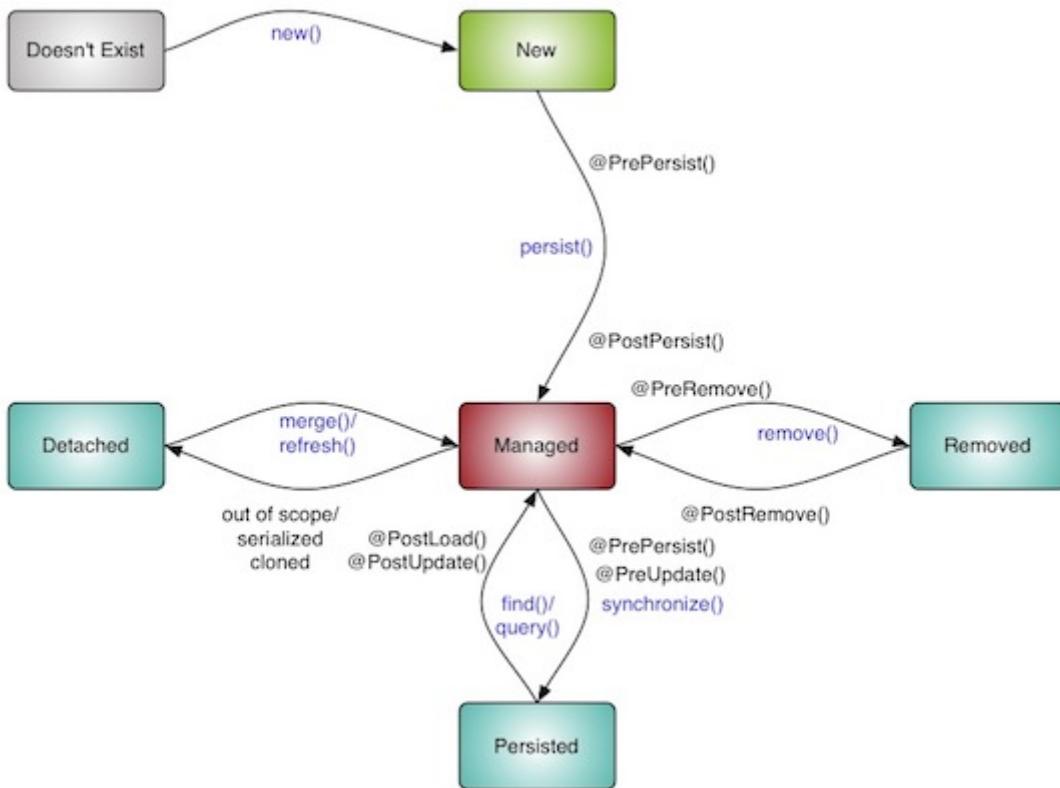
- pozwala wykonywać operacje CRUD na encjach.
- pozwala też na wykonanie zapytań.
- odpowiada mu jeden kontekst trwałości - czyli zbiór obiektów zarządzanych przez EntityManagera w danej chwili
- unikalny obiekt w ramach kontekstu trwałości to taki który posiada unikalny identyfikator

```

public static EntityManager getEntityManager() {
    EntityManagerFactory managerFactory = Persistence.createEntityManagerFactory("myPu");
    EntityManager manager = managerFactory.createEntityManager();
    return manager;
}

```

119.4.2. Cykl życia



119.4.3. Metody

Method	Description
void persist(Object entity)	Saves (persists) an entity into the database, and also makes the entity managed.
<T> T merge(T entity)	Merges an entity to the EntityManager's persistence context and returns the merged entity.
void remove(Object entity)	Removes an entity from the database.
<T> T find(Class<T> entityClass, Object primaryKey)	Finds an entity instance by its primary key.
void flush()	Synchronises the state of entities in the EntityManager's persistence context with the database.
void setFlushMode(FlushModeType flushMode)	Changes the flush mode of the EntityManager's persistent context. The flush mode may either be AUTO or COMMIT. The default flush mode is AUTO, meaning that the EntityManager tries to automatically sync the entities with the database.
FlushMode getFlushMode()	Retrieves the current flush mode.
void refresh(Object entity)	Refreshes (resets) the entity from the database.
Query createQuery(String jpqlString)	Creates a dynamic query using a JPQL statement.
Query createNamedQuery(String name)	Creates a query instance based on a named query on the entity instance.
Query createNativeQuery(String sqlString)	Creates a dynamic query using a native SQL statement.
Query createNativeQuery(String sqlString, Class res	Creates a dynamic query using a native SQL statement.
Query createNativeQuery(String sqlString, String res	Creates a dynamic query using a native SQL statement.
void close()	Closes an application-managed EntityManager.
boolean isOpen()	Checks whether an EntityManager is open.
EntityTransaction getTransaction()	Retrieves a transaction object that can be used to manually start or end a transaction.
void joinTransaction()	Asks an EntityManager to join an existing JTA transaction.

119.4.4. Uzyskiwanie EntityManagera wersja numer 1

```
@PersistenceContext  
EntityManager em;
```

119.4.5. Uzyskiwanie EntityManagera wersja numer 2

```
@PersistenceUnit  
EntityManagerFactory emf;  
EntityManager em = emf.createEntityManager();
```

WARNING:Menadżery encji zarządzane przez aplikację nie propagują automatycznie kontekstu transakcji JTA. Musimy to robić manualnie

- Przykład 1

```

EntityManagerFactory entityManagerFactory = Persistence
.createEntityManagerFactory("pu");
EntityManager em = entityManagerFactory.createEntityManager();
EntityTransaction userTransaction = em.getTransaction();
userTransaction.begin();

User user = new User("przodownik", "slawek");
em.persist(user);
userTransaction.commit();
em.close();
entityManagerFactory.close();

```

- Przykład 2

```

@PersistenceUnit(unitName = "pu")
private EntityManagerFactory entityManagerFactory;
@Resource
private UserTransaction userTransaction;
private Book book;

public String saveBook() {
    String returnValue = "BookAdded";
    try {
        userTransaction.begin();
        EntityManager em = entityManagerFactory.createEntityManager();
        em.persist(book);
        userTransaction.commit();
        em.close();
        returnValue = "BookAddedConfirmation";
    } catch (Exception e) {
        e.printStackTrace();
    }
    return returnValue;
}

```

119.4.6. EntityManager Lokalnie

- Przykład

```

EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("pu");
EntityManager entityManager = entityManagerFactory.createEntityManager();

```

119.5. EntityManager Zdalnie

```
@PersistenceContext(unitName = "pu")
EntityManager manager;
```

119.6. Metody

119.6.1. utrwalenie

- em.persist(object);

119.6.2. wyszukanie

- Employee e = em.find(Employee.class, new Long(10));

119.6.3. usuwanie

- em.remove(object);

119.6.4. getReference()

- zwraca proxy zamiast zainicjalizowanego obiektu. Encja nie będzie ładowana jeśli nie była w buforze EntityManager'a
- jeśli obiekt nie istnieje wyrzuci EntityNotFoundException

```
public class PersonServiceImpl implements PersonService {

    public void changeAge(Integer personId, Integer newAge) {
        Person person = em.getReference(Person.class, personId);

        // person is a proxy
        person.setAge(newAge);
    }

}
```

```
UPDATE PERSON SET AGE = ? WHERE PERSON_ID = ?
```

119.6.5. find()

- zwraca zainicjalizowany obiekt. Jeśli nie był załadowany w EntityManager'rze, pozyska go z bazy
- jeśli obiekt nie istnieje zwróci null

```

public class PersonServiceImpl implements PersonService {

    public void changeAge(Integer personId, Integer newAge) {
        Person person = em.getReference(Person.class, personId);

        // person is a proxy
        person.setAge(newAge);
    }

}

```

SELECT NAME, AGE FROM PERSON WHERE PERSON_ID = ?

UPDATE PERSON SET AGE = ? WHERE PERSON_ID = ?

119.7. persistence.xml

Powinnien znajdować się w classpath w katalogu **META-INF** Plik **persistence.xml** jest unikalny dla danego kontekstu **persistence unit**.

- Przykład

```

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd" version="2.0">
    <persistence-unit name="myPu" transaction-type="RESOURCE_LOCAL">
        <mapping-file>Author.hbm.xml</mapping-file>
        <mapping-file>Book.hbm.xml</mapping-file>

        <class>domain.Author</class>
        <class>domain.Book</class>

        <properties>
            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
            <property name="javax.persistence.jdbc.user" value="sa"/>
            <property name="javax.persistence.jdbc.password" value="" />
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:file:~/testjpa"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
            <property name="hibernate.hbm2ddl.auto" value="create"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>

```

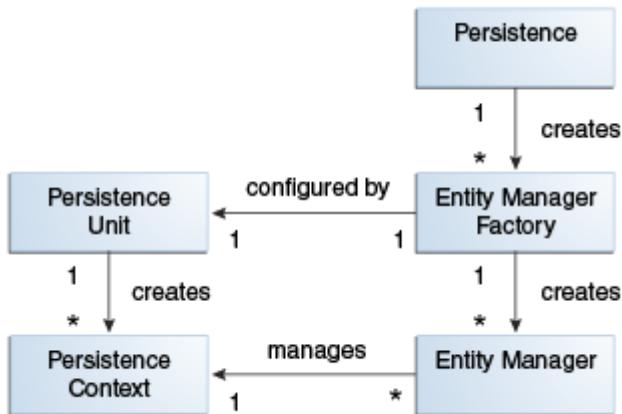
- **RESOURCE_LOCAL transaction** - sama aplikacja zarządza transakcjami .

- JTA transaction - transakcjami zarządza kontener na serwerze aplikacyjnym

120. Persistence Unit

Jednostka trwałości - sposób na komunikowanie się z bazą. Jest skonfigurowany w pliku persistence.xml

source : eclipse.org



120.1. Praca w wieloma jednostkami trwałości

- Przykład

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="pu1" transaction-type="RESOURCE_LOCAL">
    <!-- details -->
  </persistence-unit>
  <persistence-unit name="pu2" transaction-type="RESOURCE_LOCAL">
    <!-- details -->
  </persistence-unit>
</persistence>
// code...
EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu1");
EntityManager em = emf.createEntityManager();
em.persist(entity);
  
```

120.1.1. Programowy persistence.xml

- Przykład

```

Map<String, String> prop = Maps.newHashMap();

prop.put("javax.persistence.transactionType", "RESOURCE_LOCAL");
prop.put("javax.persistence.jtaDataSource", "");
prop.put("javax.persistence.jdbc.driver", "org.h2.Driver");
prop.put("javax.persistence.jdbc.url", "jdbc:h2:mem:");

EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu", prop);
EntityManager em = emf.createEntityManager();

```

120.1.2. Mapowanie klas

- Przykład

```

<persistence-unit name="unit1" transaction-type="RESOURCE_LOCAL">
<provider>org.hibernate.ejb.HibernatePersistence</provider>
<class>pl.java.scalatech.domain.Employee</class>
<class>pl.java.scalatech.domain.Department</class>
<class>pl.java.scalatech.domain.Poject</class>
.....

```

121. Persistence Context

- służy do tworzenia obiektów klasy EntityManagerFactory
- zbiór obiektów zarządzanych przez entityManagera w danej chwili

```
//include::{resourcedir}META-INF/persistence.xml[]
```

- Przykład

```

EntityManagerFactory emf = Persistence.createEntityManagerFactory("HelloWorldPU");

UserTransaction tx = TM.getUserTransaction();
tx.begin();
EntityManager em = emf.createEntityManager();
Message message = new Message()
message.setText("Hello World!");
em.persist(message);
tx.commit();
// INSERT into MESSAGE (ID, TEXT) values (1, 'Hello World!')
em.close();

```

122. Dostęp do Hibernate API z poziomu JPA

- Przykład

```
Session session = entityManager.unwrap( Session.class );
SessionImplementor sessionImplementor = entityManager.unwrap( SessionImplementor.class
);

SessionFactory sessionFactory = entityManager.getEntityManagerFactory().unwrap(
SessionFactory.class );
```

123. MetaModel

123.1. Generacja

Konfiguracja w Gradle :

- Przykład

```

sourceSets {
    intTest
    generated.java.srcDirs=['src/main/generated']
    main {
        java { srcDir 'src/main/java' }
        resources { srcDir 'src/main/resources' }
    }
    test {
        java { srcDir 'src/test/java' }
        resources { srcDir 'src/test/resources' }
    }
}

repositories {
    mavenCentral()
}

configurations {
    providedRuntime
    jpamodel
}
dependencies {
    jpamodel ("org.hibernate:hibernate-jpamodelgen:4.3.1.Final")
}

task generateMetaModel(type: JavaCompile, group: 'build', description: 'metamodel generate') {

    source = sourceSets.main.java
    classpath = configurations.compile + configurations.jpamodel
    options.compilerArgs = ["-proc:only"]
    destinationDir = sourceSets.generated.java.srcDirs.iterator().next()

    doFirst {
        logger.warn("Prepare dictionary structures " +sourceSets.generated.java.srcDirs)
        //delete(sourceSets.generated.java.srcDirs)
        //sourceSets.generated.java.srcDirs.mkdirs()
    }
}
compileJava.dependsOn generateMetaModel
compileJava.source sourceSets.generated.java, sourceSets.main.java

```

- Przykład

```

@Entity
public class Pet {
    @Id
    protected Long id;
    protected String name;
    protected String color;
    @ManyToOne
    protected Set<Owner> owners;
    ...
}

@Static Metamodel(Pet.class)
public class Pet_ {

    public static volatile SingularAttribute<Pet, Long> id;
    public static volatile SingularAttribute<Pet, String> name;
    public static volatile SingularAttribute<Pet, String> color;
    public static volatile SetAttribute<Pet, Owner> owners;
}

```

- Przykład 2

```

EntityManager em = ...;
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
EntityType<Pet> Pet_ = pet.getModel();

```

```

EntityManager em = ...;
Metamodel m = em.getMetamodel();
EntityType<Pet> Pet_ = m.entity(Pet.class);

```

124. Autoreferencja

- relacja jaka zachodzi pomiędzy polami relacji tej samej encji

```
@Entity  
class Category{  
  
    @ManyToOne  
    private Category parent;  
  
    @OneToMany(mappedBy="parent")  
    private List<Category> child;  
  
}
```

125. Dziedziczenie / Polimorfizm

126. MappedSuperclass Table per concrete class with implicit polymorphism

Właściwości klasy oznaczonej tą adnotacją będą propagowane na klasy pochodne. W relacji bazodanowej oznacza to, że kolumny będą propagowane do tabel pośredniczących tej strategii. Taka klasa nie będzie miała odwzorowania w bazie danych.



Abstract class



Klasy z adnotacją **@MappedSuperclass** nie mogą być używane do wyszukiwania oraz w zapytaniach JPQL.

- Przykład

```

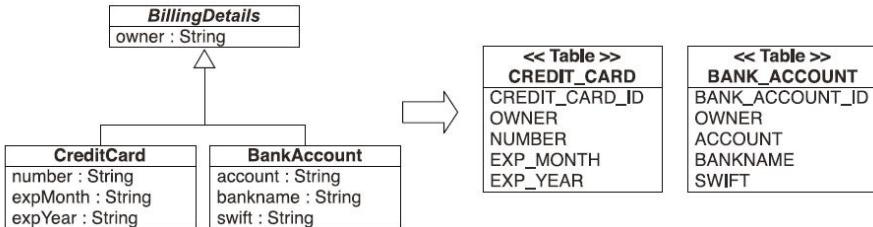
@MappedSuperclass
public abstract class BillingDetails {
    @Column(name = "OWNER", nullable = false)
    private String owner;
    ...
}

@Entity
@AttributeOverride(name = "owner", column =@Column(name = "CC_OWNER", nullable =
false))
public class CreditCard extends BillingDetails {
    @Id @GeneratedValue
    @Column(name = "CREDIT_CARD_ID")
    private Long id = null;

    @Column(name = "NUMBER", nullable = false)
    private String number;
    ...
}

```

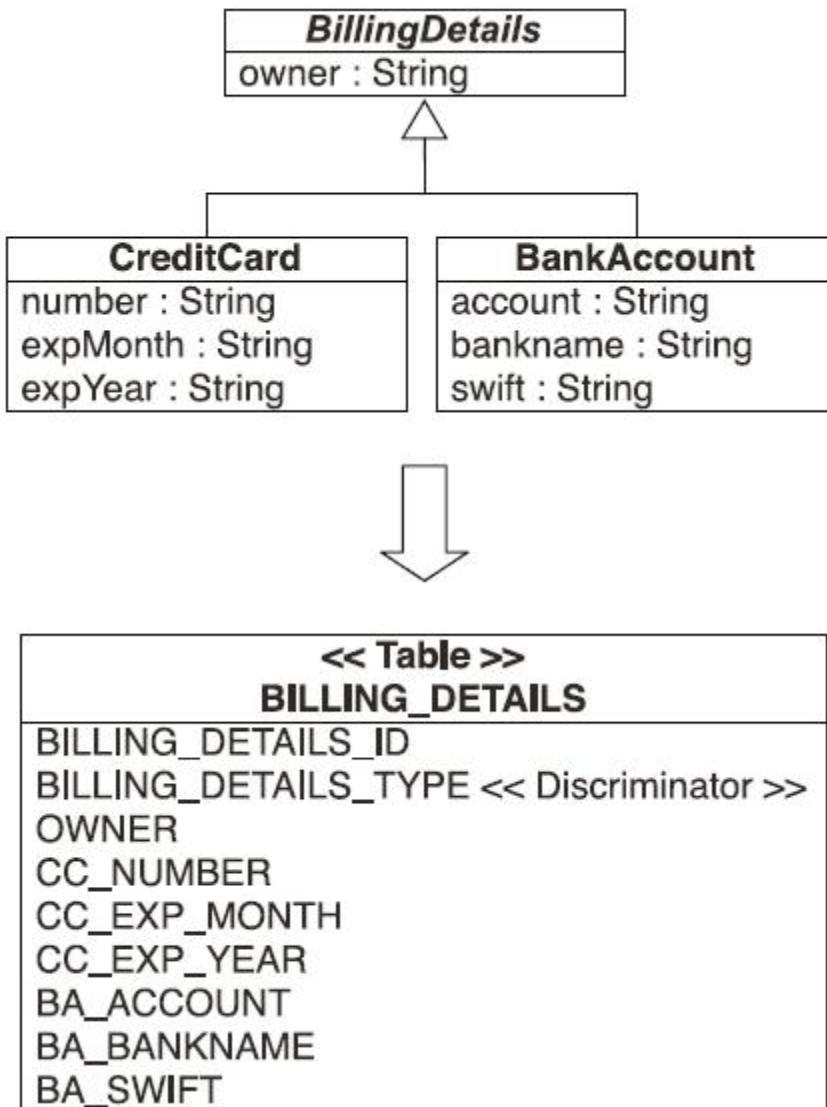
- source <https://detailfocused.wordpress.com>



127. Tabela na każdą hierarchię klas (Table per class hierarchy / Single-Table Strategy)

Martin Fowler Pattern : <http://martinfowler.com/eaaCatalog/singleTableInheritance.html>

- Symulacja dziedziczenia oparta jest na jednej tabeli.
- Tabela zawiera kolumny dotyczące wszystkich właściwości klas
- Kolumny zadeklarowane w podklasach muszą dopuszczać wartości **NULL**
- do odróżnienia podklas stosuje się dodatkową kolumnę - dyskryminator (wyłącznie wartość logiczna) -
- source <https://detailfocused.wordpress.com>



Najbardziej wydajna oraz najprostsza ze wszystkich strategii. Jeśli potrzebujemy korzystać z zapytań polimorficznych do klasy bazowej i podklasy zawierają niewiele własnych właściwości - wybierz tą strategię



Dopuszczanie wartości **NULL** może stanowić zagrożenie integralności danych. Straty miejsca w bazie. Brak normalizacji danych.



Zbyt wiele kolumn w tabeli może mieć wpływ na wydajność.

- Przykład

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="billing_type", discriminatorType=DiscriminatorType.STRING,
length=2)
@DiscriminatorValue(value="BD")
public class BillingDetails {

}

@Entity
@Table(name = "employee")
@DiscriminatorValue(value="CC")
public class CreditCard extends BillingDetails {
}
```

- **Możliwe wartości dyskriminatora to :** String, char, int, byte, short, boolean(including yes_no, true_false).



Dla każdej z podklas dyskryminator musi być unikalny. Jeśli go nie określmy będzie się nazywał jak podklasa

127.1. Discriminator formula

- Przykład

```

@Entity(name = "Account")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorFormula(
    "case when debitKey is not null " +
    "then 'Debit' " +
    "else ( " +
    "    case when creditKey is not null " +
    "    then 'Credit' " +
    "    else 'Unknown' " +
    "end ) " +
    "end "
)
public static class Account {

    @Id
    private Long id;
    private String owner;
    private BigDecimal balance;
    private BigDecimal interestRate;

@Entity(name = "DebitAccount")
@DiscriminatorValue(value = "Debit")
public static class DebitAccount extends Account {

    private String debitKey;
    private BigDecimal overdraftFee;

}

@Entity(name = "CreditAccount")
@DiscriminatorValue(value = "Credit")
public static class CreditAccount extends Account {

    private String creditKey;
    private BigDecimal creditLimit;
}

```

```

CREATE TABLE Account (
    id int8 NOT NULL ,
    balance NUMERIC(19, 2) ,
    interestRate NUMERIC(19, 2) ,
    owner VARCHAR(255) ,
    debitKey VARCHAR(255) ,
    overdraftFee NUMERIC(19, 2) ,
    creditKey VARCHAR(255) ,
    creditLimit NUMERIC(19, 2) ,
    PRIMARY KEY ( id )
)

```

128. Tabla na każdą podklasę (Table per subclass/joined strategy.)

Martin Fowler Pattern : <http://martinfowler.com/eaaCatalog/classTableInheritance.html>

- wykorzystanie relacji do emulacji dziedziczenia.
- realizacja za pomocą kluczy obcych
- osobną tabelą dla każdej klasy
- wspólny klucz główny (klucz główny głównej tabeli jest kluczem obcym dla tabeli pochodnych)
- najłatwiejsze zarządzanie bazą



główna zaleta tej strategii to pełna normalizacja modelu relacyjnego. Jeśli potrzebujemy korzystać z zapytań polimorficznych do klasy bazowej i podklasy zawierają wiele własnych właściwości - wybierz tą strategię



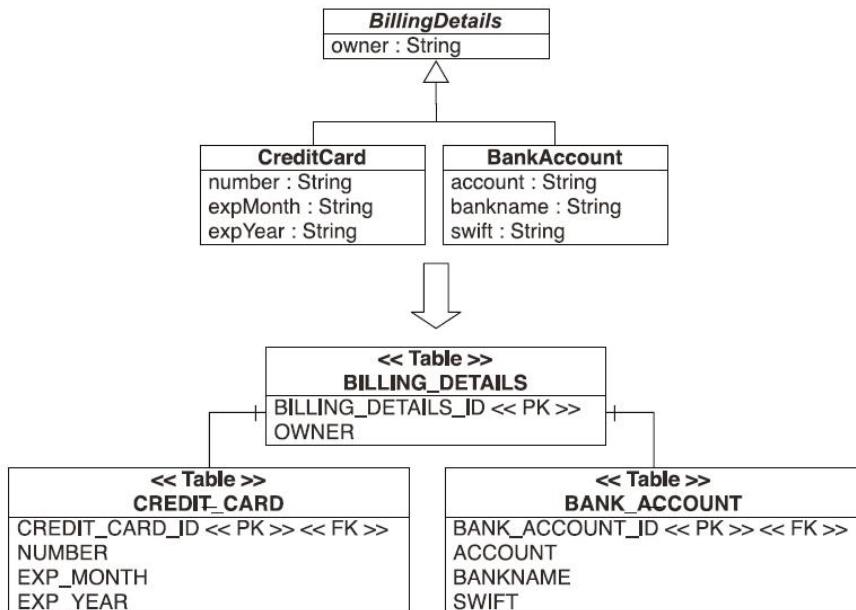
W przypadku złożonych hierarchii wydajność nie jest do zaakceptowania (joins). Zapytania wymagająłączenia wielu tabel, albo wielu sekwencyjnych odczytów

- Przykład

```
@Entity  
@Inheritance(strategy=InheritanceType.JOINED)  
public class BillingDetails {  
}  
  
@Entity  
@Table  
@PrimaryKeyJoinColumn(name="billing_id")  
public class CreditCard extends BillingDetails {  
}
```

128.1. @PrimaryKeyJoinColumn

- source :<https://detailfocused.wordpress.com>



129. Tabela na klasę konkretną (Table per concrete class)

Martin Fowler Pattern <http://martinfowler.com/eaaCatalog/concreteTableInheritance.html>

- Hibernate tworzy osobną tabelę na każdą podklasę.

CAUTION: * Słaba obsługa asocjacji polimorficznych.



Jeśli asocjacje i zapytania polimorficzne nie są potrzebne - wybierz tę strategię. Bardzo dobra wydajność jeśli będziemy pobierać dane jedynie z danego węzła relacji.



Zapytania polimorficzne zwracające obiekty wszystkich klas pasujących stwarzają duże problemy. Zapytanie dotyczące klasy bazowej musi zostać robione na n-operacji **SELECT** po czym dane grupowane są za pomocą operacji bazodanowej **UNION**. Unie są z kolei słabo przenośne.



zmiana typu jednej właściwości klasy propaguje się na zmiany kolumny w tabelach pochodnych. Każda operacja **CRUD** na tabeli 'root' pociąga zmianę w sub-tabelach

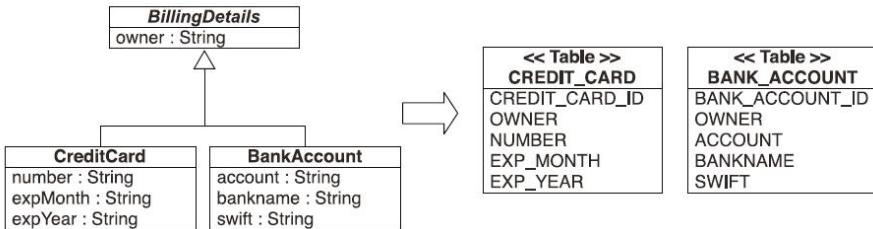
- Przykład

```

@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class BillingDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    @Column(name="id")
    private long id;
}

@Entity
@AttributeOverrides({
    @AttributeOverride(name="id", column = @Column(name="id")),
    @AttributeOverride(name="name", column = @Column(name="name"))
})
public class CreditCard extends BillingDetails {
}

```



130. Criteria

131. CRITERIA API

- Criteria API ma działanie analogiczne do JPQL z tą różnicą że działały na obiektach i bytach mocno typowalnych.
- Model w pełni obiektowy
- Silnie generyczny
- Wsparcie MetaModelu
- dynamicznie budowanie zapytań

131.1. CriteriaBuilder

Tworzony z EntityManagera. Punkt wyjściowy do funkcjonowania interfejsu Criteria API.

131.2. CriteriaQuery<T>

Reprezentuje wszystkie warunki użyte w zapytaniu

131.3. Root<T>

Zmienna wskazująca na korzeń zapytania

- Przykład

```
<X> Root<X> from( Class<X> );
<X> Root<X> from( EntityType<X> );
//example
CriteriaBuilder builder = entityManager.getCriteriaBuilder();
CriteriaQuery<Person> criteria = builder.createQuery( Person.class );
Root<Person> root = criteria.from( Person.class );
```

131.4. TypedQuery<T>

132. Wyrażenia / Expression

[UML expressions] | *UML-expressions.gif*

133. Typy / Types

[UML types] | *UML-types.gif*

134. Kwerendy / query

[UML query] | *UML-query.gif*

135. Atrybuty / Attributes

[UML attrs] | *UML-attrs.gif*

136. MetaModel

- klasa metamodelu jest tworzona przez generator metamodelu dla każdej klasy encyjnej.
- nazwa takiej klasy jest zakończona '_'
- wszystkie atrybuty odpowiadające polom i właściwościom trwałości klasy encji
- pozwala uzyskać bezpieczeństwo typologiczne i refaktoryzacyjne modelu Criteria API

137. SELECT

- Przykład

```
CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Person> criteria = builder.createQuery( Person.class );
Root<Person> root = criteria.from( Person.class );
criteria.select( root );
criteria.where( builder.equal( root.get( Person_.name ), "Przodownik" ) );

List<Person> persons = entityManager.createQuery( criteria ).getResultList();
```

137.1. Bez potrzeby rzutowania

- Przykład

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Person> criteria = cb.createQuery(Person.class);
Root<Person> i = criteria.from(Person.class);
criteria.select(i).where(cb.equal(i.get("id"), PERSON_ID));
TypedQuery<Person> query = em.createQuery(criteria);
Person result = query.getSingleResult();
```

137.2. Parametryzacja

- Przykład

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery criteria = cb.createQuery();
Root<Person> i = criteria.from(Person.class);
Query query = em.createQuery(
criteria.select(i).where(cb.equal(i.get("name"),cb.parameter(String.class, "personName")))).setParameter("personName", "slawek");
```

137.2.1. Parametryzacja z mocną kontrolą typów

- Przykład

```
CriteriaQuery criteria = cb.createQuery(Person.class);
Root<Person> i = criteria.from(Item.class);
ParameterExpression<String> personNameParameter = cb.parameter(String.class);
Query query = em.createQuery(criteria.select(i).where(cb.equal(i.get("name"),
, personNameParameter))).setParameter(personNameParameter, "slawek");
```

137.3. Wyrażenie / Expression

- Przykład

```
CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<String> criteria = builder.createQuery( String.class );
Root<Person> root = criteria.from( Person.class );
criteria.select( root.get( Person_.nickName ) );
criteria.where( builder.equal( root.get( Person_.name ), "John Doe" ) );

List<String> nickNames = entityManager.createQuery( criteria ).getResultList();
```

137.4. Pojedyncze wartości

- Przykład

```
CriteriaQuery<String> c = cb.createQuery(String.class);
Root<Employee> emp = c.from(Employee.class);
c.select(emp.<String>get("name"));
```

137.5. Wielokrotne wartości

- Przykład

```
CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Object[]> criteria = builder.createQuery( Object[].class );
Root<Person> root = criteria.from( Person.class );

Path<Long> idPath = root.get( Person_.id );
Path<String> nickNamePath = root.get( Person_.nickName );

criteria.select( builder.array( idPath, nickNamePath ) );
criteria.where( builder.equal( root.get( Person_.name ), "przodownik" ) );

List<Object[]> idAndNickNames = entityManager.createQuery( criteria ).getResultList();
```

137.6. Multiselect

- Przykład 1

```
CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Object[]> criteria = builder.createQuery( Object[].class );
Root<Person> root = criteria.from( Person.class );

Path<Long> idPath = root.get( Person_.id );
Path<String> nickNamePath = root.get( Person_.nickName );

criteria.multiselect( idPath, nickNamePath );
criteria.where( builder.equal( root.get( Person_.name ), "przodownik" ) );

List<Object[]> idAndNickNames = entityManager.createQuery( criteria ).getResultList();
```

- Przykład 2

```
CriteriaQuery<Tuple> c= cb.createTupleQuery();
Root<Employee> emp = c.from(Employee.class);
c.select(cb.tuple(emp.get("id"), emp.get("name")));
CriteriaQuery<Object[]> c = cb.createQuery(Object[].class);
Root<Employee> emp = c.from(Employee.class);
c.multiselect(emp.get("id"), emp.get("name"));
```

137.7. Aliasy

- Przykład

```
CriteriaQuery<Tuple> c= cb.createTupleQuery();
Root<Employee> emp = c.from(Employee.class);
c.multiselect(
    emp.get("id").alias("id"),
    emp.get("name").alias("fullName"));
```

137.8. Zapytania dynamiczne

- Przykład

```

public List<Employee> findEmployees(String name, String deptName, String projectName)
{
    StringBuffer query = new StringBuffer();
    query.append("SELECT DISTINCT e ");
    query.append("FROM Employee e LEFT JOIN e.projects p ");
    query.append("WHERE ");

    List<String> criteria = new ArrayList<String>();
    if (name != null) { criteria.add("e.name = :name"); }
    if (deptName != null) { criteria.add("e.dept.name = :dept"); }
    if (projectName != null) { criteria.add("p.name = :project"); }

    if (criteria.size() == 0) {
        throw new RuntimeException("no criteria");
    }
    for (int i = 0; i < criteria.size(); i++) {
        if (i > 0) { query.append(" AND "); }
        query.append(criteria.get(i));
    }

    Query q = em.createQuery(query.toString());
    if (name != null) { q.setParameter("name", name); }
    if (deptName != null) { q.setParameter("dept", deptName); }
    if (projectName != null) { q.setParameter("project", projectName); }
    return (List<Employee>)q.getResultList();
}

```

137.9. Wrapper

- Przykład

```

public class PersonWrapper {

    private final Long id;

    private final String nickName;

    public PersonWrapper(Long id, String nickName) {
        this.id = id;
        this.nickName = nickName;
    }
}

CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<PersonWrapper> criteria = builder.createQuery( PersonWrapper.class );
Root<Person> root = criteria.from( Person.class );

Path<Long> idPath = root.get( Person_.id );
Path<String> nickNamePath = root.get( Person_.nickName );

criteria.select( builder.construct( PersonWrapper.class, idPath, nickNamePath ) );
criteria.where( builder.equal( root.get( Person_.name ), "przodownik" ) );

List<PersonWrapper> wrappers = entityManager.createQuery( criteria ).getResultList();

```

137.10. Tuple

- Przykład

```

CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Tuple> criteria = builder.createQuery( Tuple.class );
Root<Person> root = criteria.from( Person.class );

Path<Long> idPath = root.get( Person_.id );
Path<String> nickNamePath = root.get( Person_.nickName );

criteria.multiselect( idPath, nickNamePath );
criteria.where( builder.equal( root.get( Person_.name ), "John Doe" ) );

List<Tuple> tuples = entityManager.createQuery( criteria ).getResultList();

for ( Tuple tuple : tuples ) {
    Long id = tuple.get( idPath );
    String nickName = tuple.get( nickNamePath );
}

//or using indices
for ( Tuple tuple : tuples ) {
    Long id = (Long) tuple.get( 0 );
    String nickName = (String) tuple.get( 1 );
}

```

138. JOIN

- Przykład 1

```

CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Phone> criteria = builder.createQuery( Phone.class );
Root<Phone> root = criteria.from( Phone.class );

// Phone.person is a @ManyToOne
Join<Phone, Person> personJoin = root.join( Phone_.person );
// Person.addresses is an @ElementCollection
Join<Person, String> addressesJoin = personJoin.join( Person_.addresses );

criteria.where( builder.isNotEmpty( root.get( Phone_.calls ) ) );

List<Phone> phones = entityManager.createQuery( criteria ).getResultList();

```

- Przykład 2

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
Join<Pet, Owner> owner = pet.join(Pet_.owners);

CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
Join<Owner, Address> address = cq.join(Pet_.owners).join(Owner_.addresses);
```

```
Join<Employee,Employee> directs = emp.join("directs");
Join<Employee,Project> projects = directs.join("projects");
Join<Employee,Department> dept = directs.join("dept");

Join<Employee,Project> project = dept.join("employees").join("projects");
```

139. FETCH

- Przykład

```
CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Phone> criteria = builder.createQuery( Phone.class );
Root<Phone> root = criteria.from( Phone.class );

// Phone.person is a @ManyToOne
Fetch<Phone, Person> personFetch = root.fetch( Phone_.person );
// Person.addresses is an @ElementCollection
Fetch<Person, String> addressesJoin = personFetch.fetch( Person_.addresses );

criteria.where( builder.isNotEmpty( root.get( Phone_.calls ) ) );

List<Phone> phones = entityManager.createQuery( criteria ).getResultList();
```

140. Użycie parametrów

- Przykład

```

CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Person> criteria = builder.createQuery( Person.class );
Root<Person> root = criteria.from( Person.class );

ParameterExpression<String> nickNameParameter = builder.parameter( String.class );
criteria.where( builder.equal( root.get( Person_.nickName ), nickNameParameter ) );

TypedQuery<Person> query = entityManager.createQuery( criteria );
query.setParameter( nickNameParameter, "JD" );
List<Person> persons = query.getResultList();

```

141. GroupBy i Tuple

- Przykład 1

```

CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Tuple> criteria = builder.createQuery( Tuple.class );
Root<Person> root = criteria.from( Person.class );

criteria.groupBy(root.get("address"));
criteria.multiselect(root.get("address"), builder.count(root));

List<Tuple> tuples = entityManager.createQuery( criteria ).getResultList();

for ( Tuple tuple : tuples ) {
    String name = (String) tuple.get( 0 );
    Long count = (Long) tuple.get( 1 );
}

```

- Przykład 2

```
SELECT e, COUNT(p) FROM Employee e JOIN e.projects p GROUP BY e HAVING COUNT(p) >= 2
```

- Przykład 3

```

CriteriaQuery<Object[]> c = cb.createQuery(Object[].class);
Root<Employee> emp = c.from(Employee.class);
Join<Employee,Project> project = emp.join("projects");
c.multiselect(emp, cb.count(project)).groupBy(emp).having(cb.ge(cb.count(project),2));

```

142. Predykaty

- IS EMPTY - **isEmpty()**
- IS NOT EMPTY - **isNotEmpty()**
- MEMBER OF - **isMember()**
- NOT MEMBER OF - **isNotMember()**
- LIKE - **like()**
- NOT LIKE - **notLike()**
- IN - **in()**
- NOT IN - **not(in())**
 - Przykład

```
Predicate criteria = cb.conjunction();
if (name != null) {
    ParameterExpression<String> p = cb.parameter(String.class, "name");
    criteria = cb.and(criteria, cb.equal(employee.get("name"), p));
}
if (deptName != null) {
    ParameterExpression<String> p = cb.parameter(String.class, "dept");
    criteria = cb.and(criteria, cb.equal(employee.get("dept").get("name"), p));
}
```

143. Skalary

- ALL - **all()**
- ANY - **any()**
- SOME - **some()**
- -- **neg()**, **diff()**
- + - **sum()**
- * - **prod()**
- / - **quot()**
- COALESCE - **coalesce()**
- NULLIF - **nullif()**
- CASE - **selectCase()**

144. Funkcje

- ABS - **abs()**

- CONCAT - **concat()**
- CURRENT_DATE - **currentDate()**
- CURRENT_TIME - **currentTime()**
- CURRENT_TIMESTAMP - **currentTimestamp()**
- LENGTH - **length()**
- LOCATE - **locate()**
- LOWER - **lower()**
- MOD - **mod()**
- SIZE - **size()**
- SQRT - **sqrt()**
- SUBSTRING - **substring()**
- UPPER - **upper()**
- TRIM - **trim()**

145. Agregacje

- AVG - **avg()**
- SUM - **sum()**
- MIN - **min(), least()**
- MAX - **max(), greatest()**
- COUNT - **count()**
- COUNT DISTINCT - **countDistinct()**

146. Podzapytania

- Przykład

```

public List<Employee> findEmployees(String name, String deptName, String projectName)
{
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Employee> c = cb.createQuery(Employee.class);
    Root<Employee> emp = c.from(Employee.class);
    c.select(emp);

    // ...

    if (projectName != null) {
        Subquery<Employee> sq = c.subquery(Employee.class);
        Root<Project> project = sq.from(Project.class);
        Join<Project, Employee> sqEmp = project.join("employees");
        sq.select(sqEmp)
            .where(cb.equal(project.get("name"),
                cb.parameter(String.class, "project")));
        criteria.add(cb.in(emp).value(sq));
    }
}

```

- Analogia do :

```

SELECT e
FROM Employee e
WHERE e IN (SELECT emp
    FROM Project p JOIN p.employees emp
    WHERE p.name = :project)

```

147. Logowanie zdarzeń

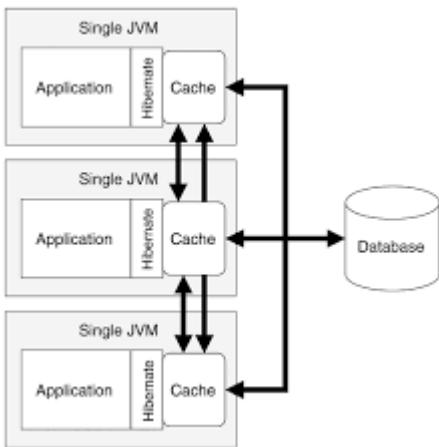
```
<property name="show_sql">true</property>
```

- Włączenie Live Statistics

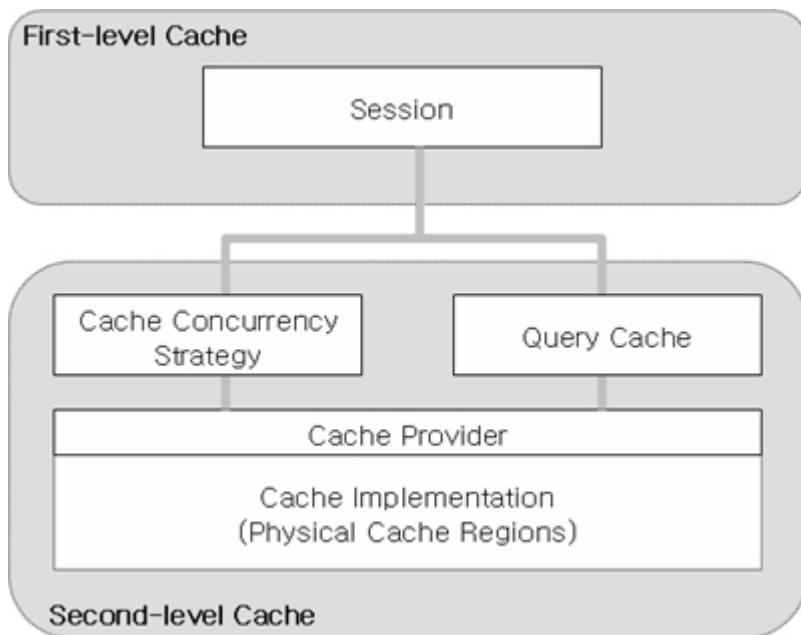
```
<property name="hibernate.generate_statistics">true</property>
```

148. Hibernate / JPA

Cache - zwiększenie wydajności zapytań poprzez eliminację ponownego zapytania w bazie



149. Architektura



- source : <https://powerdream5.wordpress.com>

150. First level cache

Pierwsza czynność wykonywana przez Hibernate to sprawdzenie tego obszaru pod kątem optymalizacji

- włączone domyślnie. Nie można go wyłączyć
- dane są umieszczane w kontekście sesji
- ograniczony w ramach **Session**
- niszczony wraz z **Session**
- jeśli hibernate szuka obiektu to najpierw w cache first level jeśli go tam nie ma uderza do bazy
- Session.evict(Object object)** -usuwa pojedyńczy obiekt z cache

- **Session.clear()** - czyści wszystkie obiekty znajdujące się aktualnie w cache
- operacja jak **Save Update Get Load List** wstawiają obiekt do first-cache
- optymalizuje operacje **EntityManager** - w obrębie **unit of work**
- wielokrotna operacja **find()** → jedna operacja **SELECT**
- wielokrotne operacja **merge()** → jedna operacja **UPDATE**

151. Second level cache

- skojarzony z **EntityManagerFactory** lub **SessionFactory**
- optymalizuje dostęp do encji lub kolekcji na poziomie całego kontekstu entityManagerFactory. Co sprawia, że operacja find() uderza do bazy tylko jeden raz



Dla bardzo dużych woluminów danych: Wyjątkowo nieefektywne → **Out of memory exception**



Wyjątkowo słabe skalowanie dla równoległych lub częstych aktualizacji danych

151.1. Włączenie

- Przykład

```
@Entity
@Cacheable
public class Employee {
    ...
}
```

151.2. javax.persistence.sharedCache.mode

- **ALL**: wszystkie encje są cachowane
- **NONE**: odwrotność do **ALL**
- **ENABLE_SELECTIVE**: Buforowaniem objęte są tylko encje oznaczone **@Cacheable(true)**
- **DISABLE_SELECTIVE**: Buforowaniem objęte są wszystkie encje z wyjątkiem tych oznaczonych **@Cacheable(false)**
- **UNSPECIFIED**: zależna od dostawcy JPA.

```
<persistence-unit name="ACME">
    <shared-cache-mode>NONE</shared-cache-mode>
</persistence-unit>
```

151.3. Retrieval Mode - określa jak dane mają być czytane z bufora (odwołania do EntityManagera)

- BYPASS: ignoruje cache. Buduje obiekt bezpośrednio z bazy danych
- USE: Jeśli dane są w cache pobiera je z bufora w przeciwnym wypadku uderza do bazy

151.4. Store Mode - określa jak dane mają być składowane w cache

- BYPASS: nie wstawia nic do bufora
- REFRESH: jeśli dane są w buforze wtedy odświeża , zamienia z danymi z bazy
- USE: dane pochodzą z cache
- Opcjonalny
- Cache na poziomie sessionFactory

151.5. Dostawcy

- Ehcache
- OSCache
- SwarmCache
- JBoss Cache
 - Przykład

```
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache-core</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-ehcache</artifactId>
</dependency>
```

Włączenie cache

- Przykład

```

<property name="hibernate.cache.use_second_level_cache">true </property>
<property name="hibernate.cache.region.factory_class">
    net.sf.ehcache.hibernate.EhCacheRegionFactory</property>

<session-factory>
    <property name="connection.driver_class">org.h2.Driver</property>
    <property name="connection.url">jdbc:h2:file:./chapter12</property>
    <property name="hibernate.dialect">org.hibernate.dialect.HSQLDialect</property>
    <property name="hibernate.hbm2ddl.auto">create</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.discriminator.ignore_explicit_for_joined">true</property>
    <property name="hibernate.generate_statistics">true</property>
    <property name="connection.username"></property>
    <property name="connection.password"></property>
    <property name="hibernate.cache.region.factory_class">
        org.hibernate.cache.ehcache.EhCacheRegionFactory
    </property>
    <mapping class="com.apress.hibernate.recipes.chapter12.recipe2.Book2"/>
</session-factory>
</hibernate-configuration>

```

- Przykład

```

@Entity(name = "Person")
@Cacheable
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public static class Person {
}

```

151.6. Strategie

- Read-only** - Najbardziej wydajna - Encje są często czytane ale nigdy modyfikowane (**CacheConcurrencyStrategy.READ_ONLY**)
- Nonstrict read-write** - Encje są rzadko modyfikowane (**CacheConcurrencyStrategy.NONSTRICT_READ_WRITE**)
- Read-write** - Większy narzut Encje są modyfikowane (**CacheConcurrencyStrategy.READ_WRITE**)
- Transactional** : Dostępna jedynie w środowisku zarządzanym. Gwarantuje pełną izolację transakcyjną aż do trybu powtarzalnego odczytu. Cache wspierany przez transakcyjne cache'ę jak JBOSS TreeCache (**CacheConcurrencyStrategy.TRNSACTIONAL**)
 - Przykład

```
@Entity  
 @Table(name="employee")  
 @Cache(usage=CacheConcurrencyStrategy.READ_ONLY)  
 public class Employee {  
  
}
```

152. Cache dla kwerend

152.1. Konfiguracja

- Przykład

```
<property name="hibernate.cache.use_query_cache" value="true"/>
```



Należy zawsze stosować z L2 cache : Query cache nie przechowuje wartości a przechowuje jedynie **id**



Włączenie **Query cache** ma sens dla zapytań często wykonywalnych, tak samo sparametryzowanych

- Przykład

```

Session session1 = SessionManager.openSession();
try {
Query query = session1.createQuery("from Book5 b where b.name like ?");
query.setString(0, "%Hibernate%");
List books = query.list();
} finally {
session1.close();
}
Session session2 = SessionManager.openSession();
try {
Query query = session2.createQuery("from Book5 b where b.name like ?");
query.setString(0, "%Hibernate%");
List books = query.list();
} finally {
session2.close();
}

```

```

<hibernate-configuration>
<session-factory>
...
<property name="hibernate.cache.use_query_cache">true</property>
...
</session-factory>
</hibernate-configuration>

```

```

@Entity
@Data
@Cacheable
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
public class Book5 {
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
int id;
String title;
}

```

The test that shows the cache in action uses a method to execute the queries to reduce code duplication:

Enabling a query cache:

```
<property name="hibernate.cache.use_query_cache">true</property>
```

- Przykład

```

Session session = sessionFactory.openSession();
for (int i = 0; i < 5; i++) {
/* Line 3 */ Criteria criteria = session.createCriteria(Employee.class).setCacheable(true);
List<Employee> employees = criteria.list();
System.out.println("Employees found: " + employees.size());
}
session.close();

```

153. Collection cache

- Przykład

```

@OneToOne(mappedBy = "person", cascade = CascadeType.ALL)
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.
NONSTRICT_READ_WRITE)
private List<Phone> phones = new ArrayList<>();

...
Person person = entityManager.find( Person.class, 1L );
person.getPhones().size();

```

154. Query level cache

aktywowany poprzez dyrektywę : hibernate.cache.use_query_cache = true przetrzymuje całkowite wyniki zapytania w pamięci cache.

154.1. aktywacja

```
<property name="hibernate.cache.use_query_cache" value="true" />
```

154.2. JPA

- Przykład

```

List<Person> persons = entityManager.createQuery(
    "select p " +
    "from Person p " +
    "where p.name = :name", Person.class)
.setParameter( "name", "Przodownik pracy")
.setHint( "org.hibernate.cacheable", "true")
.getResultList();

```

154.3. Hibernate native API

- Przykład

```
List<Person> persons = session.createQuery(  
    "select p from Person p where p.name = :name").setParameter( "name", "Przodownik  
pracy").setCacheable(true).list();
```

154.4. Używając JPA

- Przykład

```
List<Person> persons = entityManager.createQuery(  
    "select p " +  
    "from Person p " +  
    "where p.id > :id", Person.class)  
    .setParameter( "id", 0L)  
    .setHint( QueryHints.HINT_CACHEABLE, "true")  
    .setHint( QueryHints.HINT_CACHE_REGION, "query.cache.person" )  
    .getResultList();
```

155. Natywny Hibernate API

- Przykład

```
List<Person> persons = session.createQuery(  
    "select p " +  
    "from Person p " +  
    "where p.id > :id")  
    .setParameter( "id", 0L)  
    .setCacheable(true)  
    .setCacheRegion( "query.cache.person" )  
    .list();
```

156. Statystyki

- Przykład

```
Statistics statistics = session.getSessionFactory().getStatistics();
SecondLevelCacheStatistics secondLevelCacheStatistics = statistics
    .getSecondLevelCacheStatistics( "query.cache.person" );
long hitCount = secondLevelCacheStatistics.getHitCount();
long missCount = secondLevelCacheStatistics.getMissCount();
double hitRatio = (double) hitCount / ( hitCount + missCount );
```

157. Ehcache

157.1. RegionFactory

Regiony to pojemniki na dane.

157.1.1. EhCacheRegionFactory



Konfigurujemy CacheManager dla każdego SessionFactory, CacheManager nie jest współdzielony dla wszystkich instancji SessionFactory w obrębie tego samego JVM.

```
<property name="hibernate.cache.region.factory_class" value=
"org.hibernate.cache.ehcache.EhCacheRegionFactory"/>
```

SingletonEhCacheRegionFactory



Konfigurujemy CacheManager współdzielony na wielu instancji SessionFactory na tej samej maszynie wirtualnej JVM

```
<property
  name="hibernate.cache.region.factory_class"
  value="org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory"/>
```

158. Przykład użycia

159. Zapytania natywne

- SqlResultSetMapping

```

@SqlResultSetMapping(
    name      = "myResultMapping",
    entities  = {@EntityResult(...), ...},
    classes   = {@ConstructorResult(...), ...},
    columns   = {@ColumnResult(...), ...}
)

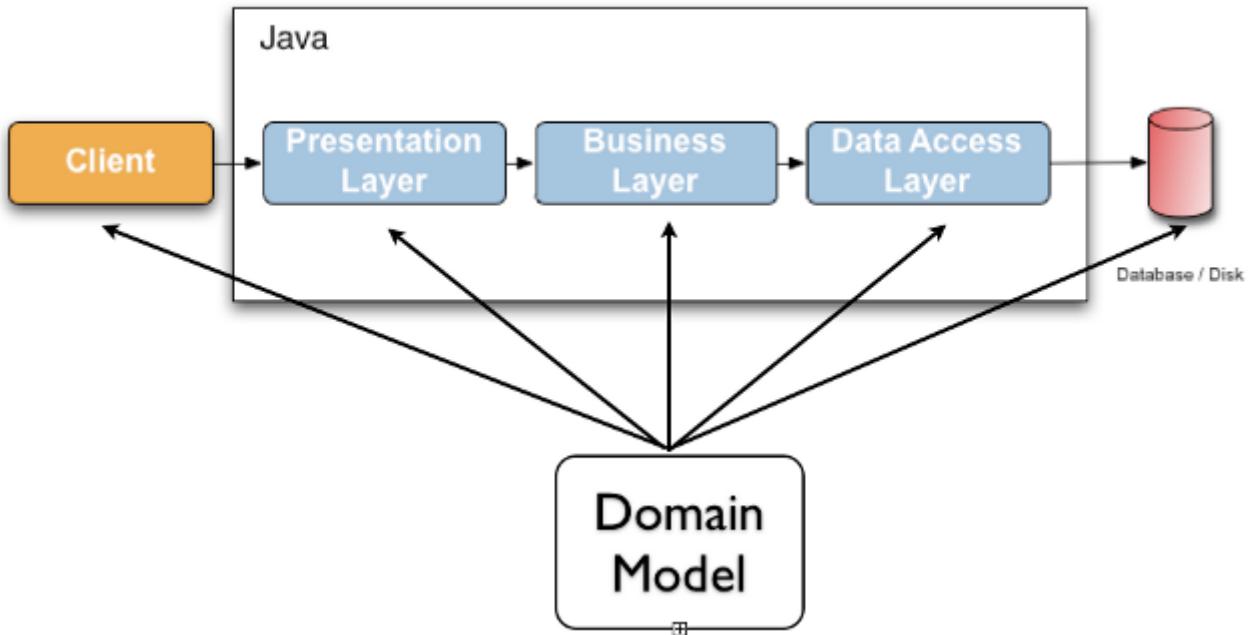
this.em.createNativeQuery("Select ...", "myResultMapping")

```

160. Walidacja

161. Walidacja

source: http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/



161.1. Zależności

```

<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-validator</artifactId>
<version>5.1.0.Alpha1</version>
</dependency>
<!-- these are only necessary if not in a Java EE environment -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-validator-cdi</artifactId>
<version>5.1.0.Alpha1</version>
</dependency>
<dependency>
<groupId>javax.el</groupId>
<artifactId>javax.el-api</artifactId>
<version>2.2.4</version>
</dependency>

<dependency>
<groupId>org.glassfish.web</groupId>
<artifactId>javax.el</artifactId>
<version>2.2.4</version>
</dependency>

```



Validator nie generuje ograniczeń na bazie !

161.2. Validation-mode

- domyślnie włączony
- Przykład konfiguracji w persistence.xml

```

<property name="javax.persistence.validation.mode">
    dd
</property>

```

161.3. Adnotacje

161.3.1. @DecimalMax

- Numeric

161.3.2. @DecimalMin

- Numeric

161.3.3. @Min

- Numeric

161.3.4. @Max

- Numeric

161.3.5. @Null

- obiekt musi być null

161.3.6. @NotNull

- obiekt nie może być null'em

161.3.7. @Digits

- Numeric

161.3.8. @Past

- data musi być w przeszłości

161.3.9. @Future

- data musi być w przyszłości

161.3.10. @AssertFalse

- boolean musi być false

161.3.11. @Size

- String | Collection | Map | Arrray długość/rozmiar

161.3.12. @AssertTrue

- boolean musi być true

161.3.13. @Pattern

- tworzymy własny pattern

161.4. Własny validator

- Przykład

```
@PasswordsEqualConstraint(field = "confirmPassword")
@NoArgsConstructor
public abstract class BaseUser extends EntityCommonVersioning {

    @Transient
    @XmlTransient
    @JsonIgnore
    private String confirmPassword;

    @Transient
    @XmlTransient
    @JsonIgnore
    private String password;
}
```

```
@Target({ ElementType.FIELD, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Size(min = 6)
@NotEmpty
@Constraint(validatedBy = PasswordsEqualConstraintValidator.class)
public @interface PasswordsEqualConstraint {
    String message() default "Wrong password";

    String field();

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}
```

```

public class PasswordsEqualConstraintValidator implements ConstraintValidator<PasswordsEqualConstraint, BaseUser> {

    @Override
    public void initialize(PasswordsEqualConstraint constraintAnnotation) {
    }

    @Override
    public boolean isValid(BaseUser user, ConstraintValidatorContext context) {
        if (!user.getPassword().equals(user.getConfirmPassword())) {
            return false;
        }
        return true;
    }

}

```

162. Testy

```

public class Car {

    @NotNull
    private String manufacturer;

    @NotNull
    @Size(min = 2, max = 14)
    private String licensePlate;

    @Min(2)
    private int seatCount;

    public Car(String manufacturer, String licencePlate, int seatCount) {
        this.manufacturer = manufacturer;
        this.licensePlate = licencePlate;
        this.seatCount = seatCount;
    }

    //getters and setters ...
}

```

```

public class CarTest {

    private static Validator validator;

    @BeforeClass
    public static void setUpValidator() {

```

```

        ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
        validator = factory.getValidator();
    }

    @Test
    public void manufacturerIsNull() {
        Car car = new Car( null, "DD-AB-123", 4 );

        Set<ConstraintViolation<Car>> constraintViolations =
            validator.validate( car );

        assertEquals( 1, constraintViolations.size() );
        assertEquals( "may not be null", constraintViolations.iterator().next()
            .getMessage() );
    }

    @Test
    public void licensePlateTooShort() {
        Car car = new Car( "Morris", "D", 4 );

        Set<ConstraintViolation<Car>> constraintViolations =
            validator.validate( car );

        assertEquals( 1, constraintViolations.size() );
        assertEquals(
            "size must be between 2 and 14",
            constraintViolations.iterator().next().getMessage()
        );
    }

    @Test
    public void seatCountTooLow() {
        Car car = new Car( "Morris", "DD-AB-123", 1 );

        Set<ConstraintViolation<Car>> constraintViolations =
            validator.validate( car );

        assertEquals( 1, constraintViolations.size() );
        assertEquals(
            "must be greater than or equal to 2",
            constraintViolations.iterator().next().getMessage()
        );
    }

    @Test
    public void carIsValid() {
        Car car = new Car( "Morris", "DD-AB-123", 2 );

        Set<ConstraintViolation<Car>> constraintViolations =
            validator.validate( car );
    }
}

```

```
        assertEquals( 0, constraintViolations.size() );
    }
}
```

163. Wydajność

164. Wydajność (Performance)

165. Sposoby pobierania rekordów / Fetching

165.1. JOIN

- Tworzymy join'a w obrębie instrukcji SELECT

165.2. SELECT

- Tworzony jest dodatkowy SELECT oprócz bazowego zapytania SELECT
 - N+1 problem jest możliwy
 - Lazy

165.3. SUBSELECT

- Tworzony jest dodatkowy SELECT w obrębie już istniejącego polecenie SELECT w celu dociągnięcia kolekcji.
 - możliwy dla relacji *-to-Many

165.4. BATCH

- Tworzone dodatkowe polecenia SELECT w obrębie w celu przyspieszenia dociągania części kolekcji

166. Lazy

- leniwe ładowanie to jeden z features Hibernate. Pozwala ładować grafy obiektów w momencie faktycznej potrzeby skorzystania z tych danych.
- w wielu wypadkach zapewnia znaczny wzrost wydajności poprzez ograniczenie niepotrzebnych strzałów do bazy
- zmniejsza czas odpowiedzi
- zapobiega zbyt dużemu zużyciu pamięci

- problem LazyLoadingException

166.1. Sposobyinicjalizacji :

166.1.1. Fizyczne 'dotknięcie' kolekcji

- Przykład

```
Person person = this.em.find(Person.class, id);
person.getPhones().size();
```

166.1.2. Fetch Join



Cartesian problem : Nie możemy używać join fetch dla wielu kolekcji ponieważ tworzymy kartezjana !! Złączenie będzie zawierało wszystkie możliwe kombinacje.

- Przykład

```
Query q = this.em.createQuery("SELECT p FROM Person p JOIN FETCH p.phones ph WHERE
p.id = :id");
q.setParameter("id", id);
person = (Person) q.getSingleResult();
```

166.1.3. Fetch Join z Criteria

- Przykład

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery q = cb.createQuery(Person.class);
Root p = q.from(Person.class);
p.fetch("phones", JoinType.INNER);
q.select(p);
q.where(cb.equal(p.get("id"), personId));

Person person = (Person) this.em.createQuery(q).getSingleResult();
```

166.1.4. Named Entity Graph

- Przykład

```
@Entity  
@NamedEntityGraph(name = "graph.Person.phones",  
    attributeNodes = @NamedAttributeNode("phones"))  
public class Person implements Serializable {
```

- Przykład

```
EntityGraph graph = this.em.getEntityGraph("graph.Person.phones");  
  
Map hints = new HashMap();  
hints.put("javax.persistence.fetchgraph", graph);  
  
Person person = this.em.find(Person.class, personId, hints);
```

166.1.5. Dynamiczny graf

- Przykład

```
EntityGraph graph = this.em.createEntityGraph(Person.class);  
Subgraph phonesGraph = graph.addSubgraph("phones");  
  
Map hints = new HashMap();  
hints.put("javax.persistence.loadgraph", graph);  
  
Person person = this.em.find(Person.class, personId, hints);
```

166.2. less lazy loading

- Przykład

```
@ManyToMany  
@Fetch(FetchMode.JOIN)  
public Set<ArtEntity> getArtEntities() {  
    return artEntities;  
}
```

166.3. Batching for Performance

- Przykład

```
@ManyToMany  
 @BatchSize(size = 10)  
 public Set<ArtEntity> getArtEntities() {  
 return artEntities;  
 }
```

166.4. OpenInView

- Pojedyncza instancja EntityManagera na HttpRequest
- Tworzy nową transakcję na początku każdego HttpRequest'u
- Komituje lub Rollback'uje transakcję na końcu requestu



Lock transakcji na połączenie bazodanowe. Opóźniona reakcja bazodanowa dopóki nie zostanie wyredendowany widok.



File upload requests

- Przykład

```
<bean name="openEntityManagerInViewInterceptor" class=  
"org.springframework.orm.jpa.support.OpenEntityManagerInViewInterceptor" />
```

- Przykład

```
<filter>  
 <filter-name>OpenEntityManagerInViewFilter</filter-name>  
 <filter-class>  
 org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-  
 class>  
 </filter>  
 <!--Map the EntityManager Filter to all requests -->  
 <filter-mapping>  
 <filter-name>OpenEntityManagerInViewFilter</filter-name>  
 <url-pattern>/*</url-pattern>  
 </filter-mapping>
```

- Analiza logów Hibernate - poprawa wolnych zapytań
 - Analiza statystyk
 - wolne zapytania
 - zbyt wiele zapytań n+1 problem
 - trafienia w cache
- Wybór strategii pobierania danych : fetchType

166.5. FETCH JOIN

- Przykład

```
SELECT DISTINCT a FROM Author a JOIN FETCH a.books b
```

@NamedEntityGraph

```
@NamedEntityGraph(name = "graph.AuthorBooksReviews", attributeNodes = @NamedAttributeNode(value = "books"))
```

- Ciężkie zapytania np raporty powinny być wykonywane po stronie bazy

@NamedStoredProcedureQuery

- Użyj cache
 - pierwszy poziom cache aktywowany jest domyślnie
 - drugi poziom cache jest włączany świadomie przez programistę. Konfigurowany dla klasy i kolekcji
 - buforowanie zapytań. Jest użyteczny podczas częstego wykonywania zapytań z takimi samymi parametrami.
- Masowe operacje UPDATE i DELETE CriteriaUpdate and CriteriaDelete
 - Przykład

```
CriteriaBuilder cb = this.em.getCriteriaBuilder();

// create update
CriteriaUpdate<Order> update = cb.createCriteriaUpdate(Order.class);

// set the root class
Root e = update.from(Order.class);

// set update and where clause
update.set("amount", newAmount);
update.where(cb.greaterThanOrEqualTo(e.get("amount"), oldAmount));

// perform update
this.em.createQuery(update).executeUpdate();
```

- Strategie pobierania (Fetching Strategies)

167. FETCH

167.1. Eager

- natychmiastowe
- czasem wygodne do użycia
- znaczący przyrost danych pobieranych z bazy
- sekwencyjne odczyty z bazy lub bufora danych
- FetchType.EAGER - domyślne dla @OneToOne i @ManyToOne

167.2. Lazy

- opóźniony/odroczony dostęp do danych
- proxy
- tworzy nowe zapytanie do bazy danych jeśli obiekt nie istnieje w buforze
- FetchType.LAZY - domyślne dla @OneToMany i @ManyToMany
- jest możliwe jedynie, gdy podstawowa encja jest w stanie managed
- pobranie encji w stanie **Detached** spowoduje wyrzucenie wyjątku - LazyInitializationException

167.2.1. Zapobieganie LazyInitializationException

- ponowne utrwalenie encji
- pobieranie przy pomocy Fetch JOIN
- wybór Eager zamiast Lazy
- openSessionInView pattern
- EntityGraph
- isInitialized() - sprawdzamy czy pośrednik jest zainicjalizowany
- initialize() - programowe wymuszenie inicjalizacji

167.3. Fetch Join

- obiekt czy kolekcja zostaje pobrana razem z obiektem głównym przez zastosowanie złączenia JOIN FETCH

167.3.1. INNER JOIN FETCH** - dla pobrania pojedynczych obiektów

167.3.2. LEFT JOIN FETCH** – dla pobrania kolekcji

167.4. Batch

- poprawa wydajności dla strategii lazy poprzez pobranie grupy obiektów. To samo dotyczy się poprawy strategii eager.



To tak naprawdę nie strategia a wskazówka mająca na celu zwiększenia wydajności innych strategii jak : lazy czy eager. To dobra strategia dla mniej doświadczonych developerów , którzy chcą osiągnąć zadowalającą wydajność bez potrzeby wnikliwej analizy kodu SQL.

167.5. Extra lazy

- tylko dla kolekcji
- nie dociąga całej kolekcji
- @LazyCollection(LazyCollectionOption.EXTRA)
- niektóre operacje jak : size(), contains(), get(), etc. nie odpalają pełnej inicjalizacji kolekcji

167.5.1. EXTRA

- .size() , .contains() etc nie inicjalizują pełnej kolekcji

167.5.2. TRUE

- inicjalizacja pełnej kolekcji przy pierwszym odwołaniem do niej

167.5.3. FALSE

- Eager loading

167.6. Określanie głębi wczytywanych obiektów

Sterowanie max liczbą złączonach tabel w jednym zapytaniu SQL.



parametr odpowiedzialny za to ustawienie to : **hibernate.max_fetch_depth**

```
List<Author> authors = this.em.createQuery(  
        "SELECT DISTINCT a FROM Author a JOIN FETCH a.books b",  
        Author.class).getResultList();
```

+ Relationships gets loaded in same query - Requires a special query for each use case - Creates cartesian product

- @NamedEntityGraph Declaratively defines a graph of entities which will be loaded

```
@NamedEntityGraph(  
    name = "graph.AuthorBooksReviews",  
  
    attributeNodes =  
    @NamedAttributeNode(value = "books")  
)
```

168. Kartezjan problem

- Omówienie
- Przykład

```
@Entity  
public class Person extends AbstractEntity{  
  
    private static final long serialVersionUID = -4106601879598237198L;  
    private String firstName = null;  
    private String lastName = null;  
  
    @OneToMany(cascade = CascadeType.ALL, fetch=FetchType.EAGER)  
    // @Fetch(FetchMode.SELECT)  
    @JoinColumn(name="PERSON_ID")  
    private List<Address> addresses;  
  
    @OneToMany(cascade = CascadeType.ALL, fetch=FetchType.EAGER)  
    // @Fetch(FetchMode.SELECT)  
    @JoinColumn(name="PERSON_ID")  
    private List<Phone> phones;  
}
```

```

select
    person0_.id as id1_1_0_,
    person0_.version as version2_1_0_,
    person0_.firstName as firstNam3_1_0_,
    person0_.lastName as lastName4_1_0_,
    addresses1_.PERSON_ID as PERSON_I4_0_1_,
    addresses1_.id as id1_0_1_,
    addresses1_.id as id1_0_2_,
    addresses1_.version as version2_0_2_,
    addresses1_.CITY as CITY3_0_2_,
    phones2_.PERSON_ID as PERSON_I4_2_3_,
    phones2_.id as id1_2_3_,
    phones2_.id as id1_2_4_,
    phones2_.version as version2_2_4_,
    phones2_.phoneNumber as phoneNum3_2_4_
from
    Person person0_
left outer join
    Address addresses1_
        on person0_.id=addresses1_.PERSON_ID
left outer join
    Phone phones2_
        on person0_.id=phones2_.PERSON_ID
where
    person0_.id=?

```

169. Kroki optymalizacji

169.1. Dziennik zdarzeń

- trafienia w bufor
- koszty złączenia czy może dwa osobne selecty
- czas wykonywania zapytań

169.2. Analiza przypadków użycia

- próby wykrycia problemu n+1
- analiza wywołań zapytań w celu zmniejszenia liczby i złożoności dla danej akcji biznesowej

169.3. Dostrajanie parametrów

- hibernate.max_fetch_depth
- hibernate batch fetch
- dobór najlepszego stylu kaskadowego dla każdej relacji w celu zmniejszenia wywołania liczby

transakcji i zapytań do bazy poprzez zarządcę transakcji



org.hibernate.SQL → DEBUG, org.hibernate.type → TRACE



Patrz rozdział : Jak pokazać parametryzacje zapytań SQL ?

169.4. Gradle

- Przykład

```
ext {  
    hibernateVersion = 'hibernate-version-you-want'  
}  
  
buildscript {  
    dependencies {  
        classpath "org.hibernate:hibernate-gradle-plugin:$hibernateVersion"  
    }  
}  
  
hibernate {  
    enhance {  
        enableLazyInitialization= false  
        enableDirtyTracking = false  
        enableAssociationManagement = false  
    }  
}
```

169.5. Maven

- Przykład

```

<build>
  <plugins>
    [...]
    <plugin>
      <groupId>org.hibernate.orm.tooling</groupId>
      <artifactId>hibernate-enhance-maven-plugin</artifactId>
      <version>$currentHibernateVersion</version>
      <executions>
        <execution>
          <configuration>
            <failOnErrors>true</failOnErrors>
            <enableLazyInitialization>true</enableLazyInitialization>
            <enableDirtyTracking>true</enableDirtyTracking>
            <enableAssociationManagement>
              true</enableAssociationManagement>
            
```

```

            </configuration>
            <goals>
              <goal>enhance</goal>
            
```

```

            </goals>
          
```

```

        </execution>
      
```

```

      </executions>
    
```

```

  </plugin>
  [...]

```

```

  </plugins>
</build>

```

170. readOnly

source : *java persistence with hibernate*

- Przykład

```

em.unwrap(Session.class).setDefaultReadOnly(true);
Item item = em.find(Item.class, ITEM_ID);
item.setName("New Name");
em.flush(); // no update

```

- Przykład

```

Item item = em.find(Item.class, ITEM_ID);
em.unwrap(Session.class).setReadOnly(item, true);
item.setName("New Name");
em.flush() //no update

```

- Przykład

```

org.hibernate.Query query = em.unwrap(Session.class).createQuery("select i from Item
i");
query.setReadOnly(true).list();
List<Item> result = query.list();
for (Item item : result)
    item.setName("New Name");
em.flush(); // no update

```

- Przykład

```

Query query = em.createQuery(queryString).setHint(org.hibernate.annotations.
QueryHints.READ_ONLY,true );

```

171. Inne możliwe problemy i wskazówki:

- aktualizowanie encji jedna-po-drugiej zamiast zrobienie tego w pojedynczym kwerendzie
- ciężkie procesowanie po stronie Javy zamiast bardziej wydajnego procesowania po stronie bazy
- dla małych woluminów danych Eager loading zawsze sens.
- opcja **hibernate.max_fetch_depth** - dostosowuje ilość możliwych złączeń w systemie. (optymalna wartość to 1-5)
- tam gdzie to możliwe wybieraj implementację Set zamiast List (cartesian problem)
- dla dużych wartości kolekcji używaj późnego ładowania
- zastosuj **_@Version** aby uzyskać poziom uniknąć niepowtarzalnych odczytów na domyślnym poziomie izolacji jakim jest odczyt zatwierdzonych (Read Committed) (*scalability issue*) czyli podnosimy prawie za darmo poziom na powtarzalne odczyty (Repeatable Read)
- Praca z wielkimi kolekcjami - stosuj @BatchSize w celu poprawy wydajności , @LazyCollection = Extra lazy
- Wolny insert lub update (wiersz po wierszu → narzut na tworzenie kwerendy, przepustowość sieci) : stosuj przetwarzania wsadowe , jako półrodekk hibernate.jdbc.batch_size=100
- OutOfMemory - batch insert/update

```
//bad code :)

Session session = SessionManager.openSession();
Transaction tx = session.beginTransaction();

for(int i=0;i<1_000_000;i++) {
    Book book = new Book();
    book.setName("MyBook "+i);
    book.setPrice(i);
    session.save(book);
}

tx.commit();
session.close();
```

- flush() i clear() w odniesieniu do cache first level
- Session.setCacheMode (CacheMode.IGNORE) dla second level cache

```
Transaction tx = session.beginTransaction();
session.setCacheMode(CacheMode.IGNORE);
// ...
```

- globalnie

```
<property name="hibernate.jdbc.batch_size">25</property> <!-- 5-30 -->
<property name="hibernate.cache.use_second_level_cache">false</property>
```

172. Rady

- Listener vs Converter jak i gdzie go stosować ?
- Formuła
- ColumnTransformer
- DynamicInsert i DymanicUpdate a optymalizacja
- Immutable jak i kiedy ?
- @Subselect i symbioza z Immutable
- Lazy i rozwiązywanie problemów
- Filters vs View - centralizacja zarządzania (on/off)

```
Unresolved directive in performance.adoc - include:::/src/test/java/pl/java/scalatech
/exercise/filter/FilterTest.java[tags=contains,indent=0]
```

- DTO (Fowler : rozprysk, równoległa hierarchia klas (one2one))
 - składacz (LazyInitializationException)
 - spłaszczenie domeny
 - transfer obiektów - integracja systemowa

173. Paginacja

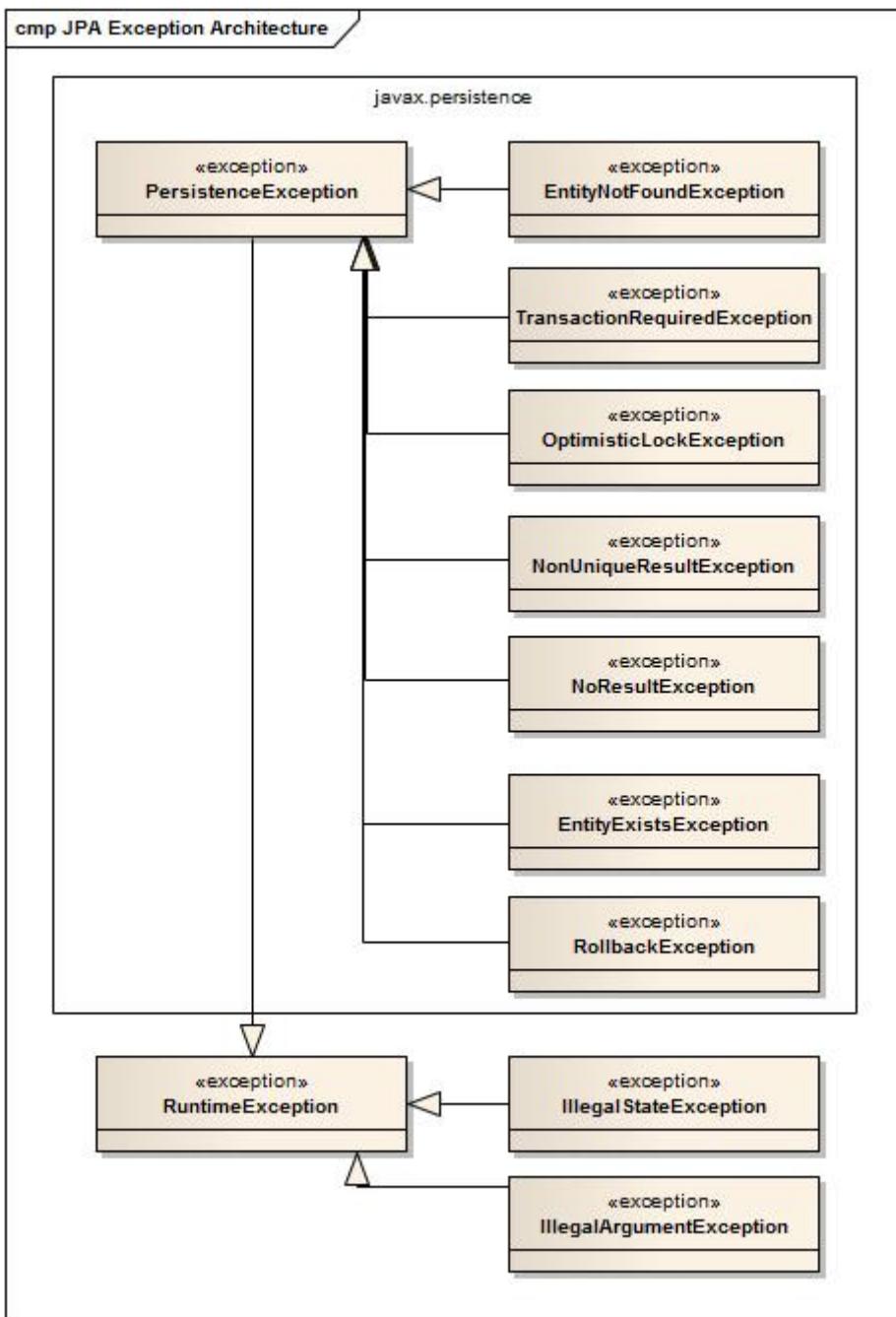
```
public List<Employee> findEmployees(int offset, int limit) {
    TypedQuery<Employee> query = entityManager.createQuery("SELECT e FROM Employee e
    ORDER BY e.name", Employee.class);
    query.setFirstResult(offset);
    query.setMaxResults(limit);
    return query.getResultList();
}
```

174. Obsługa wyjątków

- **HibernateException / PersistenceException** - generyczny wyjątek Hibernate
- **SQLException** - wyrzucany przez wewnętrzny sterownik JDBC
- **ConstraintViolationException** - naruszenie węzłów referencyjnych
- **NoResultException** - wyrzucany przez metodę **getSingleResult** (w kontekscie : TypedQuery,Query) jeśli nie zwróci żadnych wyników
- **NonUniqueResultException** - wyrzucany przez metodę **getSingleResult** (w kontekscie : TypedQuery,Query) jeśli zwróci więcej niż jeden wynik
- **QueryTimeoutException** - wyrzucany przez zapytania w kontekscie TypedQuery i Query jeśli kwerenda trwa za długo
- **LockTimeoutException** - wyrzucany w przypadku blokowania pesymistycznego
- **EntityNotFoundException** - wyrzucany przez metodę refresh lub lock jeśli encja po odświeżeniu kontekstu nie istnieje w bazie
- **OptimisticLockException** - modyfikacja encji chronionej przez **@Version**
- **EntityExistsException** - zapis encji już zarządzanej
- **LazyInitializationException** - próba dostępu do leniwych danych , gdy skojarzona sesja jest już zamknięta.

source

www.bhaveshthaker.com



175. Rozwiązywanie problemów

175.1. Problemy z asocjacjami dwukierunkowymi

175.2. Kłopoty z pamięcią

175.3. Problemy z wydajnością mechanika :

- sprawdź wygenerowane SQL
- sprawdź execution plan
- sprawdź poprawność indeksów bazodanowych

próba optymalizacji zapytania

- próba rozważenia zapytania natywnego
- jpql wspiera tylko niektóre podzbiory features z bazy danych
- SQL dla danej bazy może być wysoce wyspecjalizowany

175.4. @Basic(lazy)

- Wymagane narzędzie do instrumentalizacji bytecode

175.5. OutOfMemoryException

- czyszczenie cache
- flush dla kontekstu
- dla przetwarzania wsadowego : hibernate.jdbc.batch.size = 50
- wyłączenie cache drugiego poziomu hibernate.cache.use_second_level_cache = false

175.6. Ładowanie klas do kontekstu z poziomu SessionFactory

```
sessionFactory = new AnnotationConfiguration()
    .addPackage("test.animals")
    .addAnnotatedClass(Flight.class)
    .addAnnotatedClass(Sky.class)
    .addAnnotatedClass(Person.class)
    .addAnnotatedClass(Dog.class);
```



JPA to nie zadziała

175.7. N+1 problem

- Redukcja do $N/batch_size + 1$ - query
- JOIN FETCH
- EntityGraph
- Fetch.SubSelect

175.8. Jak pokazać parametryzacje zapytań SQL ?

175.8.1. Log4jdbc

Zależność :

-

```
compile 'org.lazyluke:log4jdbc-remix:????'
```

Konfiguracja :

```
@Bean  
public Log4JdbcCustomFormatter logFormater() {  
    Log4JdbcCustomFormatter formatter = new Log4JdbcCustomFormatter();  
    formatter.setLoggingType(LoggingType.SINGLE_LINE);  
    formatter.setSqlPrefix("SQL:\r");  
    return formatter;  
}
```

```
log4jdbc.sqltiming.warn.threshold=1000  
log4jdbc.dump.sql.maxline length=0  
log4jdbc.dump.sql.addsemicolon=true  
log4jdbc.trim.sql=true  
log4jdbc.trim.sql.extra blanklines=false
```

175.8.2. P6Spy

Link do biblioteki : <https://sourceforge.net/projects/p6spy/>

Dodajemy zależność :

```
<dependency>  
    <groupId>p6spy</groupId>  
    <artifactId>p6spy</artifactId>  
    <version>??</version>  
</dependency>
```

- Przykład

```
<session-factory>  
    <property name="hibernate.connection.driver_class">com.p6spy.engine.spy.P6SpyDriver  
    </property>  
    <property name="hibernate.connection.password">password</property>  
    <property  
name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>  
    <property name="hibernate.connection.username">test</property>  
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>  
</session-factory>
```

Konfigurujemy właściwości : spy.properties

```
realDriver=com.mysql.jdbc.Driver
```

```
#specifies another driver to use  
realDriver2=
```

175.8.3. Log4J, Sl4j

```
log4j.logger.org.hibernate.type=trace
```

176. Dobre praktyki

- Ochrona przed **SQL Injection**



Patrz kod poniżej

```
String searchString = getValueEnteredByUser();  
  
Query query = em.createQuery(  
    "select i from Item i where i.name = '" + searchString + "'"  
)
```

- zachowanie właściwych poziomów abstrakcji podczas modelowania relacji
- unikanie jawnego operacji save() - wzorzec 'unit of work'
- load vs get Lepiej na początku jest sprawdzić czy obiekt jest null czy nie jeśli chcemy użyć metody get(). (NullPointerException problem)

```
-unikaj relacji dwukierunkowych  
- tight coupling  
- cykliczność  
- utrzymanie spójności (musimy pamiętać aby obsługiwać dwie strony relacji)  
- paginacja  
- DDD agregacja  
- unikaj merge  
- unikaj obiektów odłączonych  
- pobieranie zbyt dużych ilości danych powoduje marnowanie pamięci w warstwie aplikacji  
- identyczność obiektów a ich tożsamość w bazie
```

176.1. Top down (dobre dla już istniejącego kodu)

- implementujemy model obiektowy

tworzymy encję i generujemy schemat danych

176.2. Bottom up (gdy istnieje baza)

- zaczynamy pracę z istniejącym modelem danych w bazie
- generujemy encje i relację dzięki **reverse engineering**

176.3. Middle out (dobre przy nowym wytwarzaniu)

- startujemy od strony modelu encyjnego. Tworzymy encje i relacje między nimi od podstaw

176.4. Meet in the middle (z JDBC na Hibernate'a)

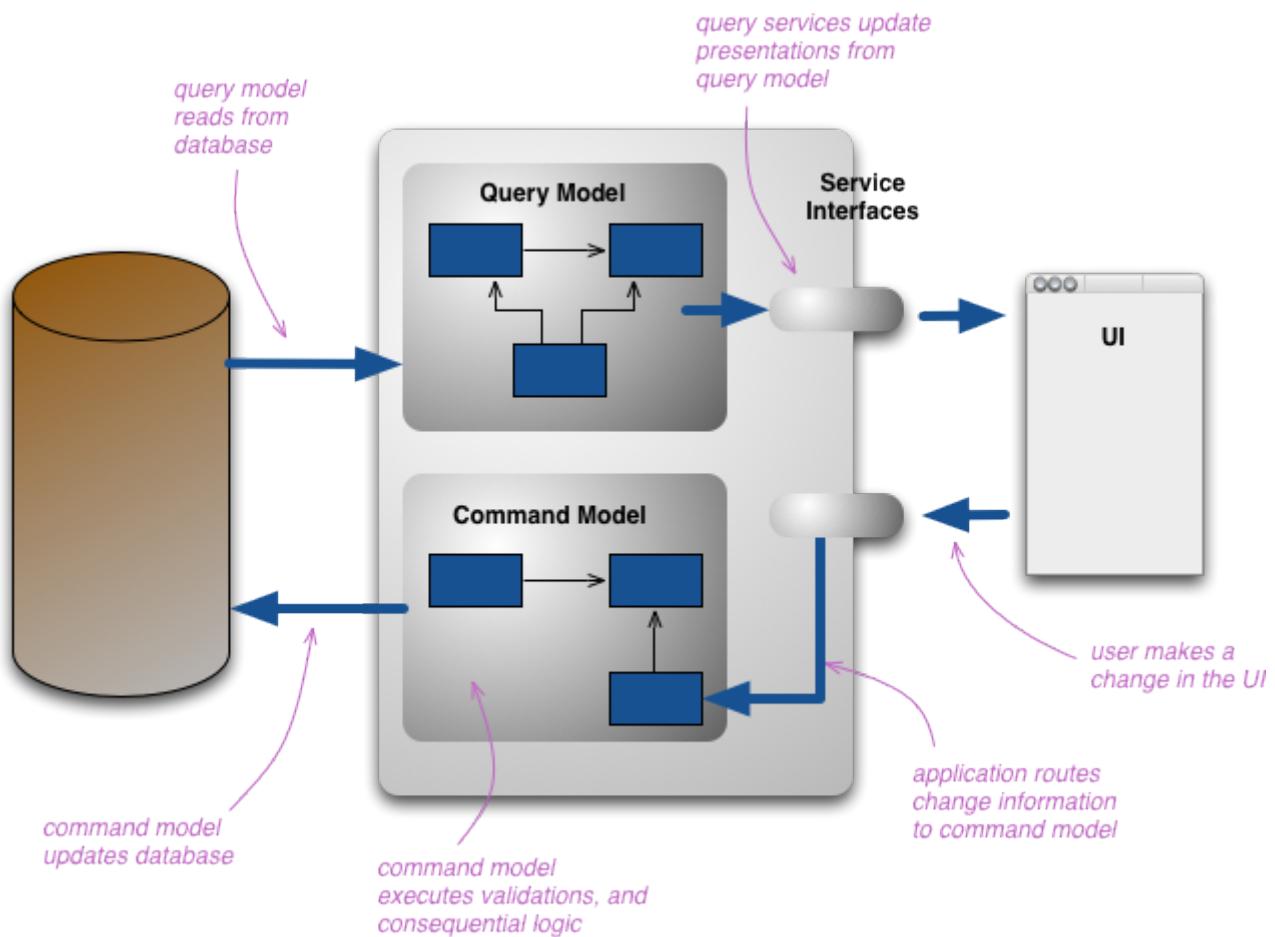
- startujemy z istniejącym modelem danych i gotowymi w pewnym stopniu plikami POJO po czym przekształcamy je na pełnoprawne encje.

176.5. Architektury

176.5.1. CQRS Command–query separation

- cqrs <http://martinfowler.com/bliki/CQRS.html>
 - skalowalne systemy
 - większy nakład pracy
 - dane do odczytu są dostępne z opóźnieniem

•



176.5.2. DDD

source: wikipedia

Jest to podejście do tworzenia oprogramowania kładące nacisk na takie definiowanie obiektów i komponentów systemu oraz ich zachowań, aby wiernie odzwierciedlały rzeczywistość. Dopiero po utworzeniu takiego modelu należy rozważyć zagadnienia związane z techniczną realizacją.

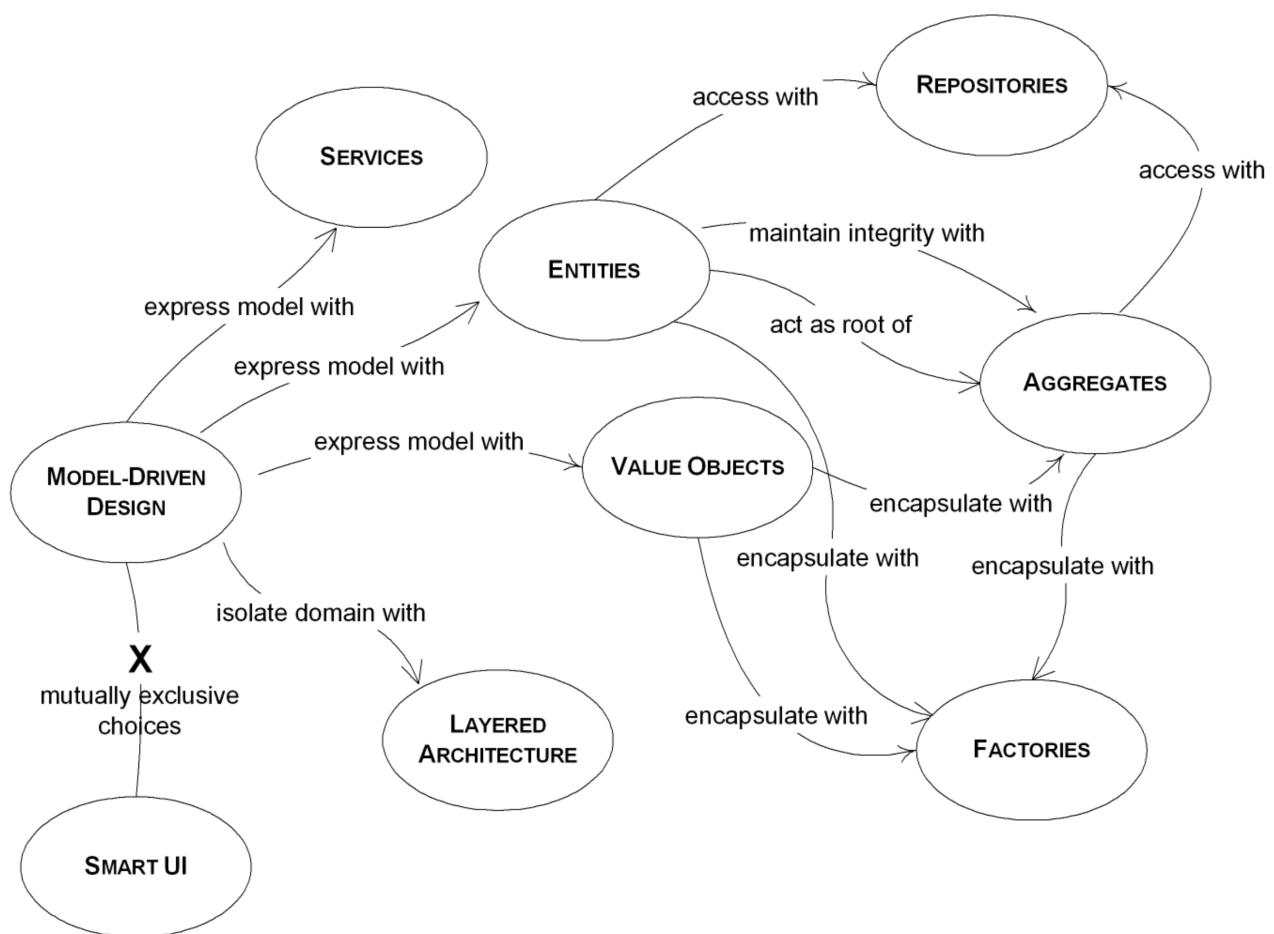
Podejście to umożliwia modelowanie systemów informatycznych przez ekspertów, którzy znają specyfikę problemu lecz nie muszą znać się na projektowaniu architektury systemów informatycznych.

Domain-Driven Design zaleca stosowanie określonych wzorców projektowych i architektonicznych.

source: <http://static.olivergierke.de/lectures/ddd-and-spring/>

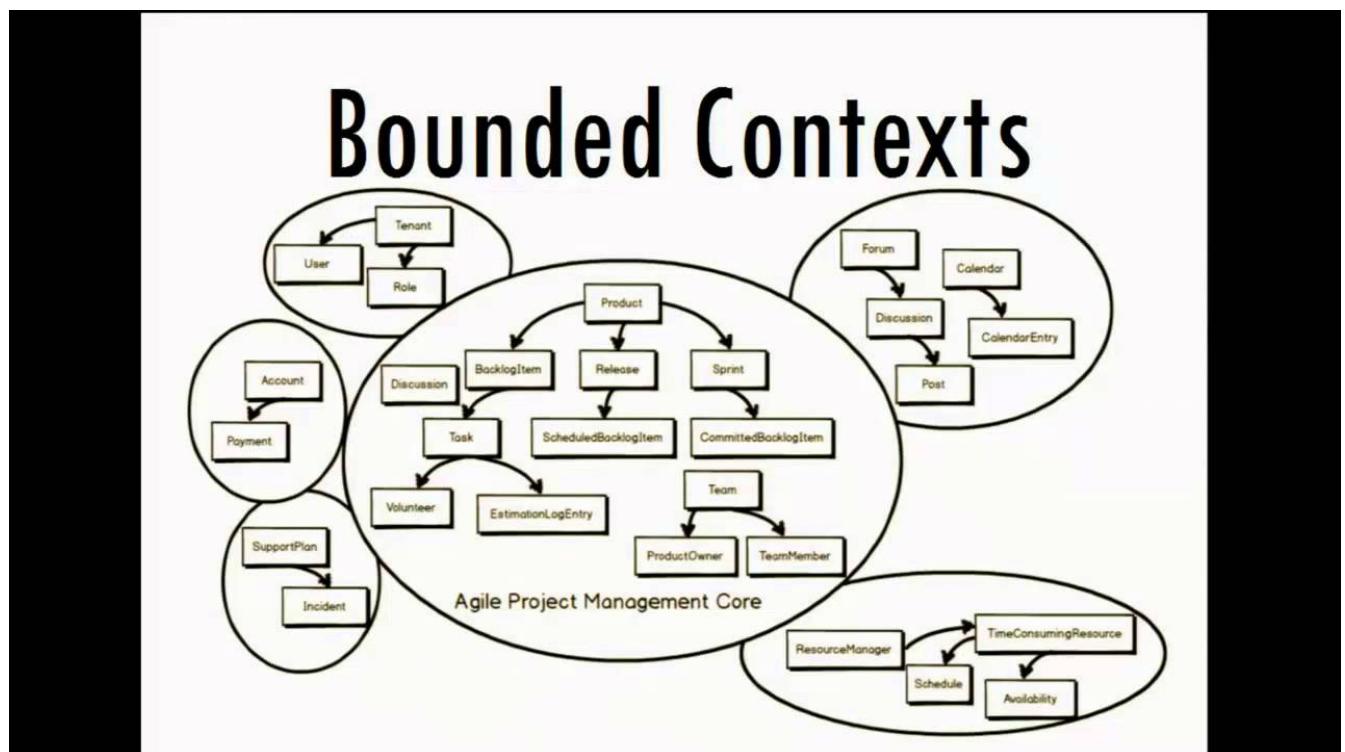
source: <http://www.infoq.com/minibooks/domain-driven-design-quickly>

Zastosowanie Springa dla architektury DDD



- DDD kontekst:

source : <https://www.youtube.com/watch?v=aieoAWXNjlo>



177. Blokady

178. Lock/Blokowanie

178.1. Enabling Optimistic Concurrency Control (Blokowanie optymistyczne)/Blokowanie optymistyczne z wersjonowaniem

- zakładamy, że obecnie wykonywana transakcja na jakimś zasobie jest jedyną, która dokonuje zmian.
- hibernate tworzy dodatkową kolumnę **@Version** możliwe typy :
 - int
 - Integer
 - short
 - Short
 - long
 - Long
 - java.sql.Timestamp
- podczas zatwierdzania zmian porównywalny jest orginalny numer wersji z bieżącą wartością. W razie różnicy, zatwierdzenie transakcji jest wycofane i wyrzucany jest wyjątek **OptimisticLockException**
- nie chroni przed **phantom read**



Wydajność

178.1.1. LockMode

OPTIMISTIC – w czasie zatwierdzania wykonywane jest zapytanie **SELECT FOR UPDATE**.

Tylko jedna transakcja ma chwilowy dostęp do zasobu. Jeśli druga transakcja chciałaby zmodyfikować obiekt zostanie wyrzucony wyjątek

OPTIMISTIC_FORCE_INCREMENT – to samo co wyżej. Działa nawet dla nie zmienionej wersji.

- Przykład

```

@Entity
public class Book {

    @Id
    @GeneratedValue (strategy=GenerationType.TABLE)
    @Column (name="ISBN")
    private long isbn;

    @Version
    @Column (name="version")
    private Integer version;

    @Column (name="book_Name")
    private String bookName;

}

```

178.2. Using Pessimistic Concurrency Control / Blokowanie pesymistyczne

- mechanizm umiemożliwiający współbieżny dostęp do konkretnego zasobu.
- zakładana w momencie odczytu danych aż do zakończenia transakcji



Nie jest to najlepszy wybór dla wysoce współbieżnej aplikacji. Powoduje istotny spadek wydajności. Możemy liczyć się z zakleszczeniami (deadlock).

****READ**** - blokowanie na odczyt (ochrona przed dirty reads i unrepeatable reads)
 Zakładany automatycznie gdy Hibernate odczytuje dane przy poziomie izolacji Repeatable Read or Serializable isolation level.

****WRITE**** - blokowanie na zapis (ochrona przed dirty reads i unrepeatable reads)
 Zakładany automatycznie, gdy Hibernate wstawia lub aktualizuje wiersz

178.2.1. LockMode - blokada dla konkretnego elementu

LockMode.NONE

- nie wykonuj zapytania chyba, że obiektu nie ma w buforze. Pominięcie poziomów buforowania i przejście do bazy danych.
- Czytaj z bazy danych tylko wtedy gdy obiekt nie istnieje w buforze

LockMode.READ

- pomiń oba poziomy buforowania i dokonaj sprawdzenia wersji , aby sprawdzić czy dane w

pamięci są takie same jak bazie

- Czytaj z bazy niezależnie od zawartości bufora

LockMode.WRITE

- uzyskiwany automatycznie , gdy zarządcą trwałości zapisał dane do wiersza aktualnej transakcji
- **LockMode.WRITE** jest zakładany automatycznie, gdy Hibernate wstawia lub aktualizuje wiersz

LockMode.UPGRADE

- pomija poziomy buforowania, dokonuje sprawdzenia wersji i uzyskuje blokadę pesymistyczną na poziomie bazy
- **LockMode.UPGRADE** może być założony, gdy użytkownik użyje SELECT ... FOR UPDATE w bazie wspierającej tą składnię.
- żaden inna transakcja nie może zmodyfikować rekordu

LockMode.UPGRADE_NOWAIT

- do samo co przy **UPGRADE** ale stosuje zapytanie SELECT ... FOR UPDATE NOWAIT . Wyłącza to czekanie na zwolnienie blokad dotyczących wpółbieżności i natychmiastowe zgłoszenie wyjątku , jeśli blokady nie udało się uzyskać.
- LockMode.UPGRADE_NOWAIT może być założony, gdy użytkownik użyje SELECT ... FOR UPDATE NOWAIT w bazie Oracle.

```
public Object load(Class theClass, Serializable id, LockMode lockMode) throws  
HibernateException  
public Object load(String entityName, Serializable id, LockMode lockMode) throws  
HibernateException
```

178.2.2. Blokowanie pojedyńczych encji

- Przykład

```
User user = em.find(User.class, 4L);  
em.lock(user, LockModeType.PESSIMISTIC_WRITE);  
//lub  
User user = em.find(User.class, 4L, LockModeType.PESSIMISTIC_WRITE);
```

178.2.3. Blokowanie wszystkich encji zwracanych przez kwerende

- Przykład

```
String statement = ....  
TypedQuery<User> query = em.createQuery(statement, User.class);  
query.setLockMode(LockModeType.PESSIMISTIC_READ);  
List<User> users = query.getResultList();
```

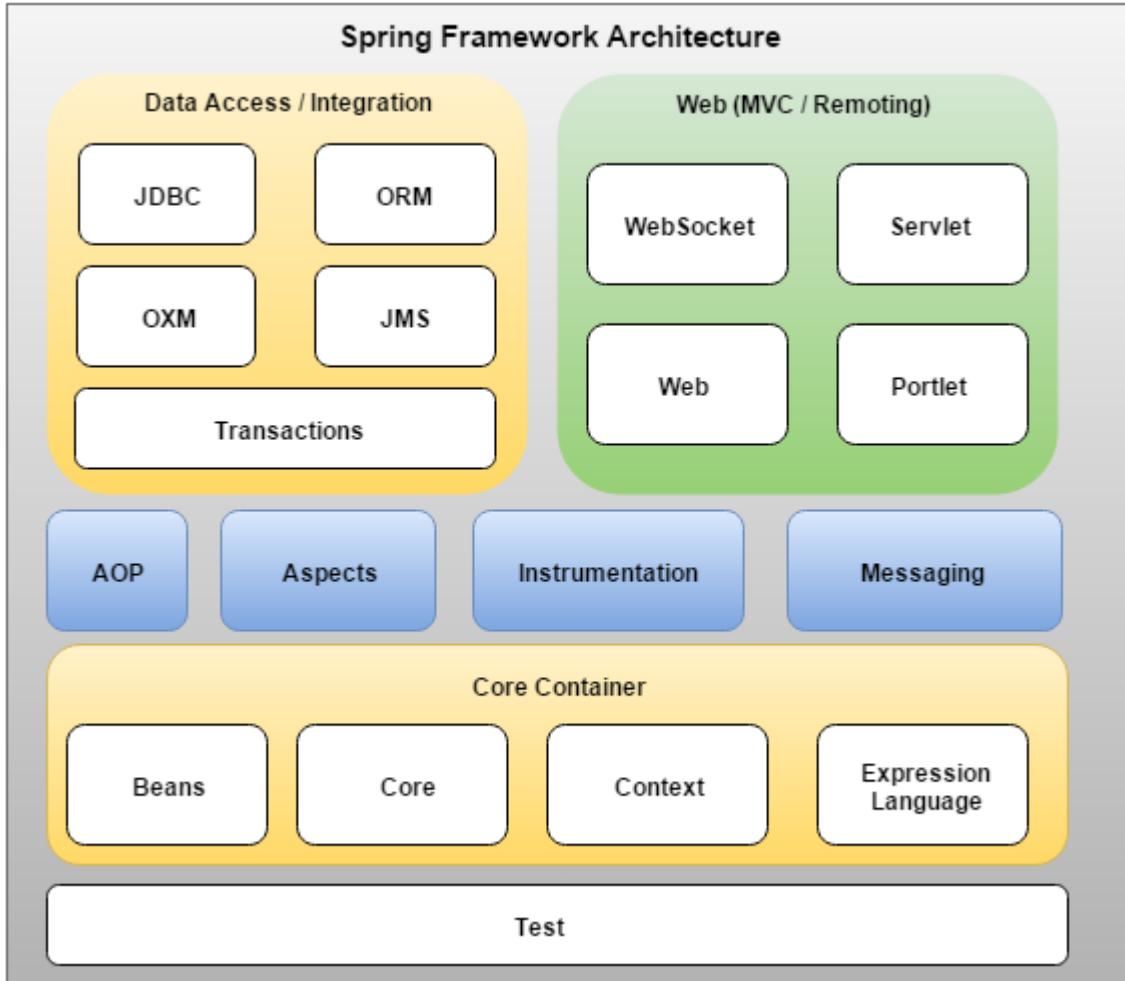
179. Integracja ze Spring

180. Transakcje + Integracja JPA/Hibernate z frameworkiem Spring

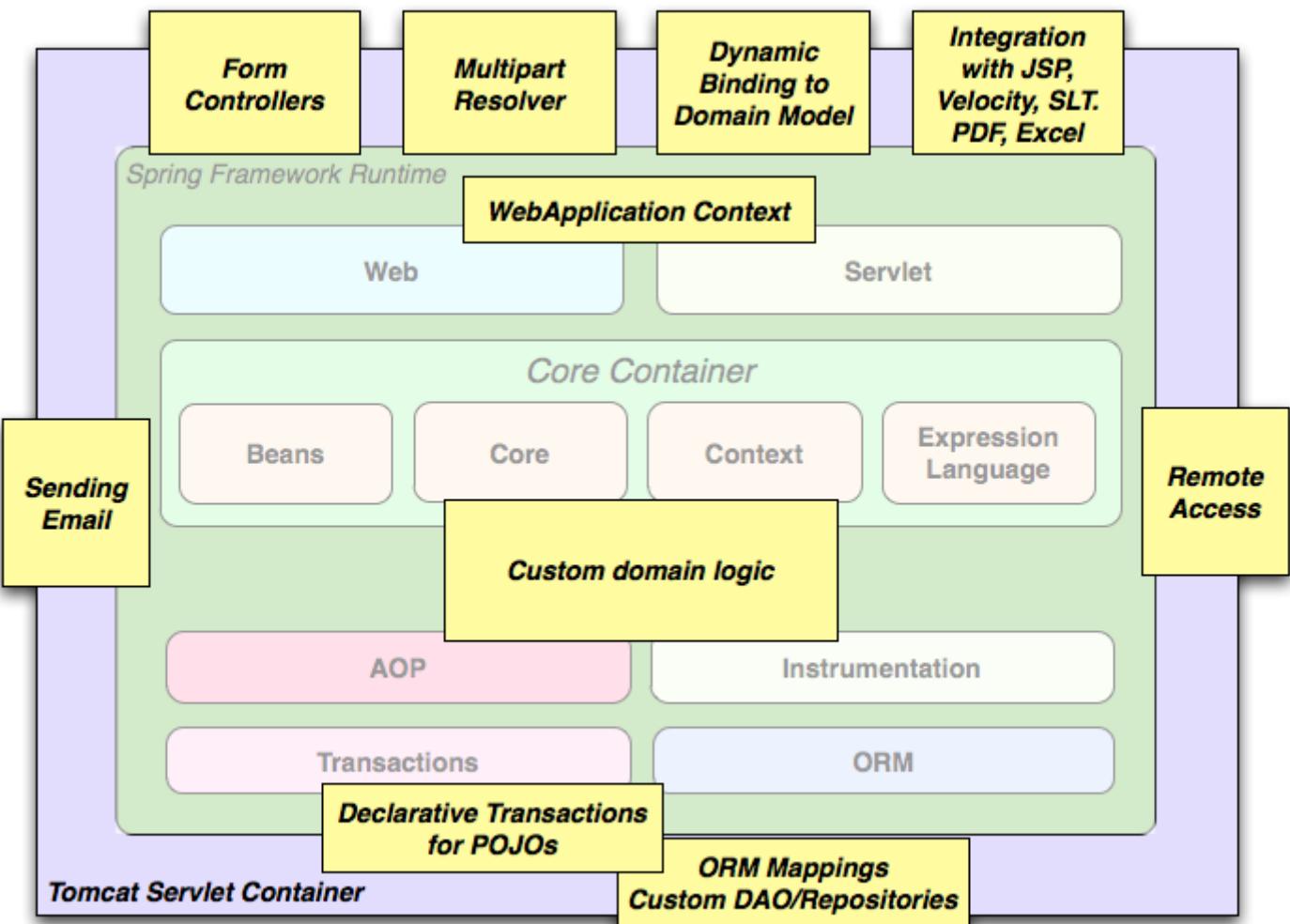
180.1. Historia

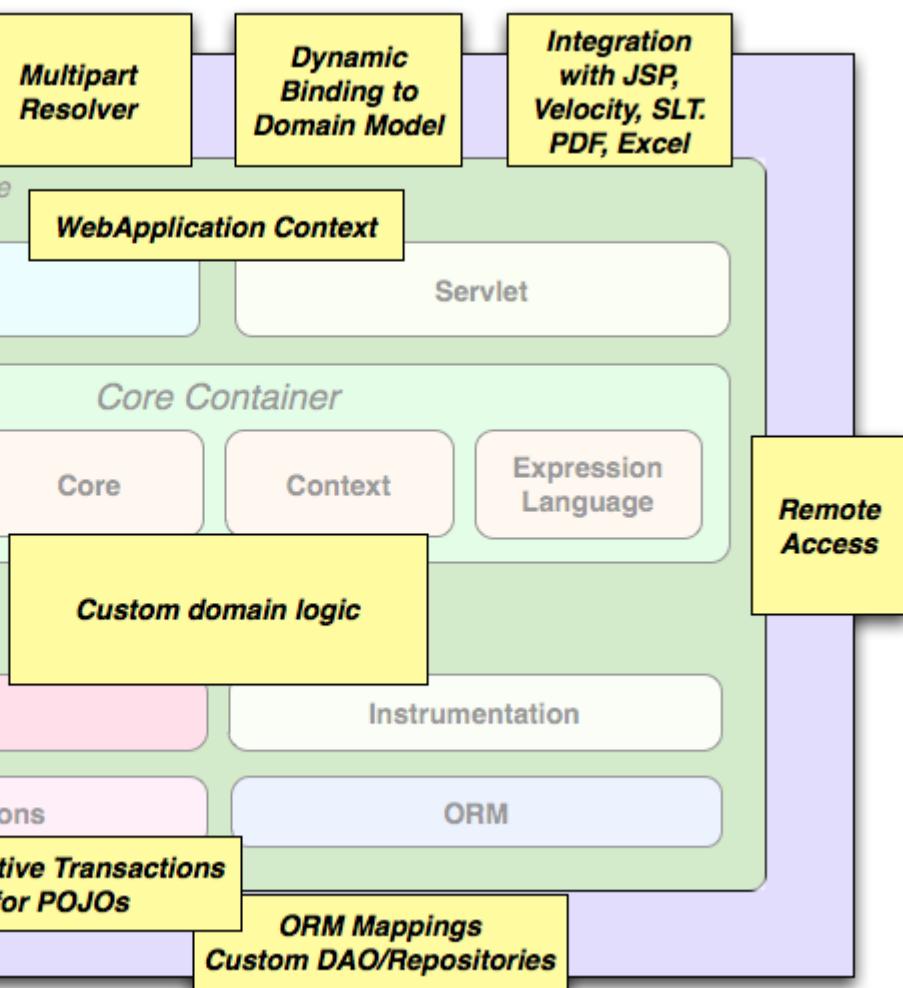
- Spring 1.0 - Marzec 2004
- Spring 2.0 - Październik 2006
- Spring 2.5 - Grudzień 2007
- Spring 3.0 - Grudzień 2009
- i dalsze wersje
- aktualna 4.2.5.Release

Stworzony przez Rod Johnson jako alternatywa dla ciężkiego EJB Bazuje na IoC



- zarys

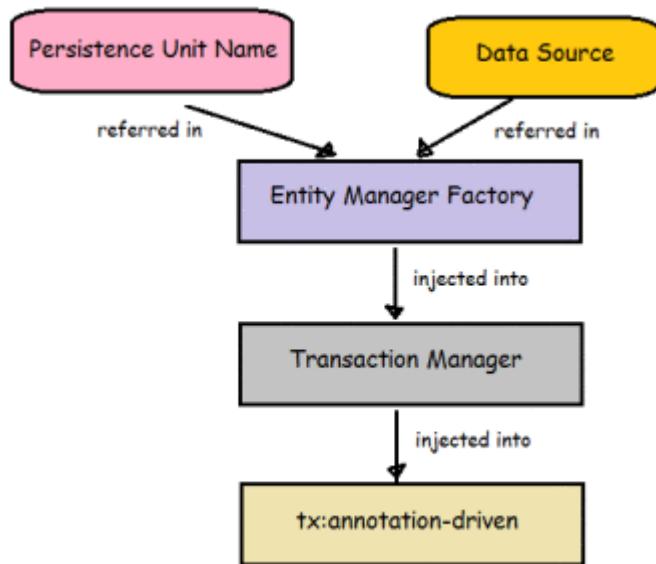




180.2. Cechy

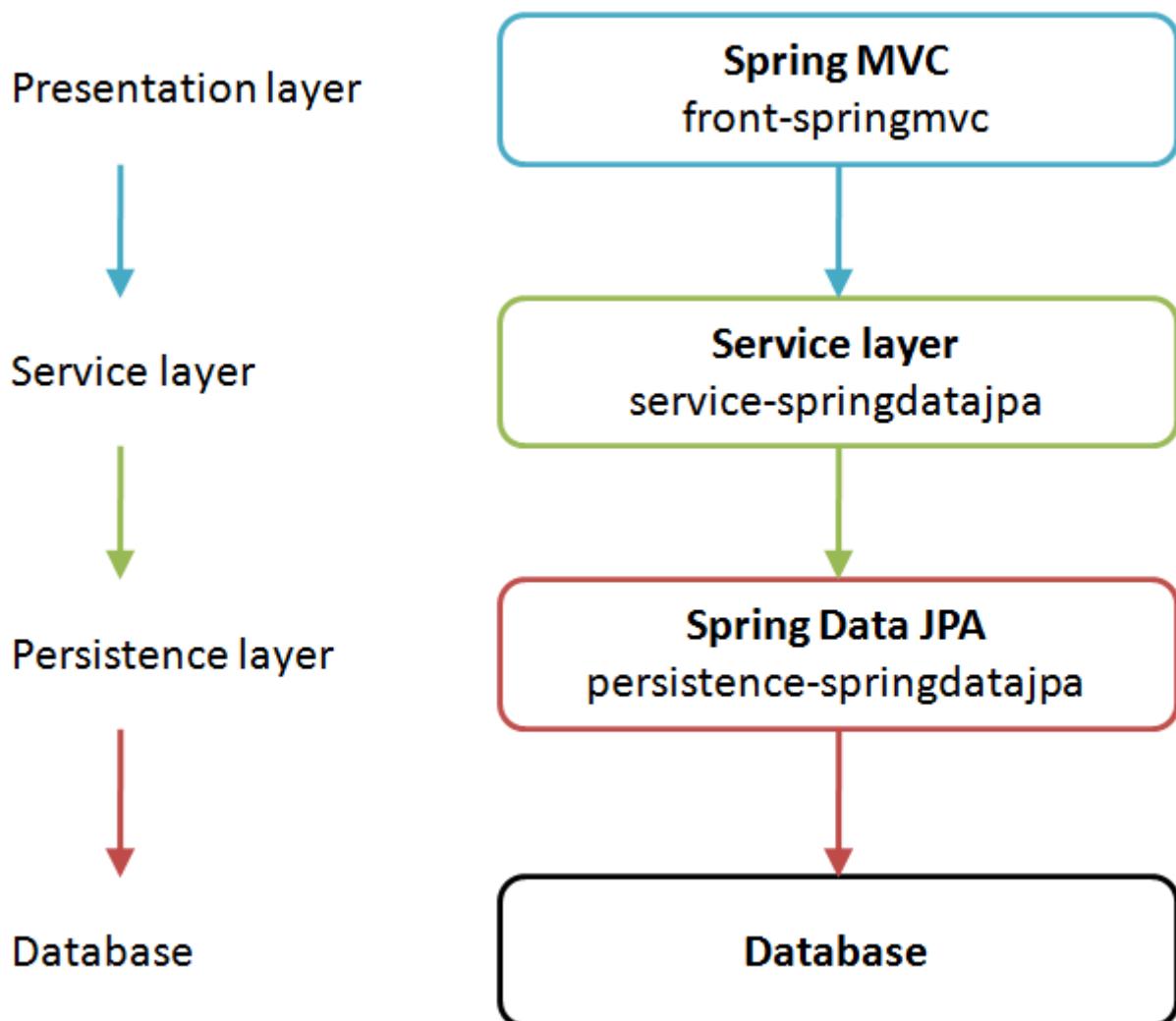
- testowalność
- loose coupling
- IoC / dependency injection
- programowanie deklaratywne
- wspracie dla AOP, Web, JMS, Batch, Security itd
- eliminacje boilerplace code
- szybkość wytwarzania
- łatwość utrzymania

source: <https://aishwaryavaishno.wordpress.com/category/technical-posts/java/>

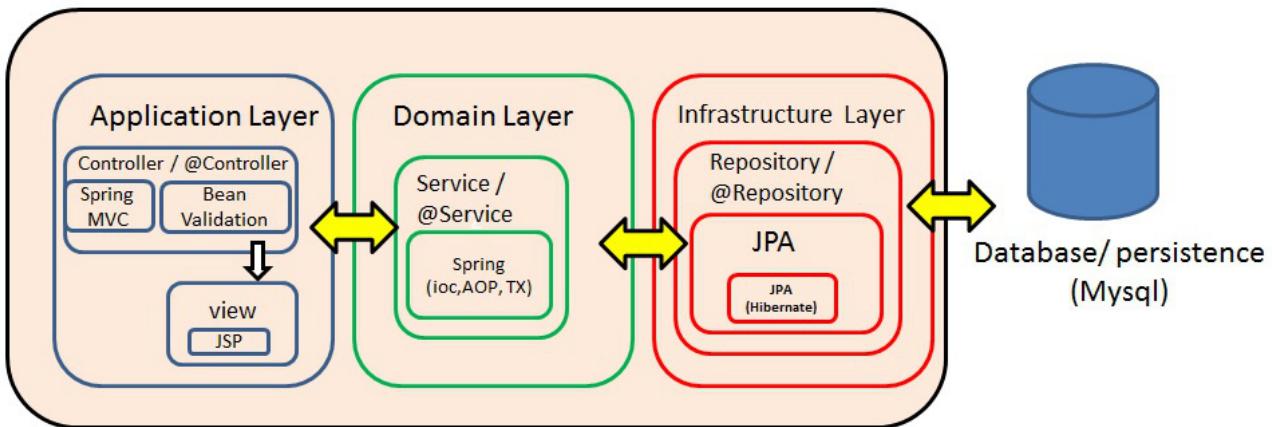


180.3. Architektura

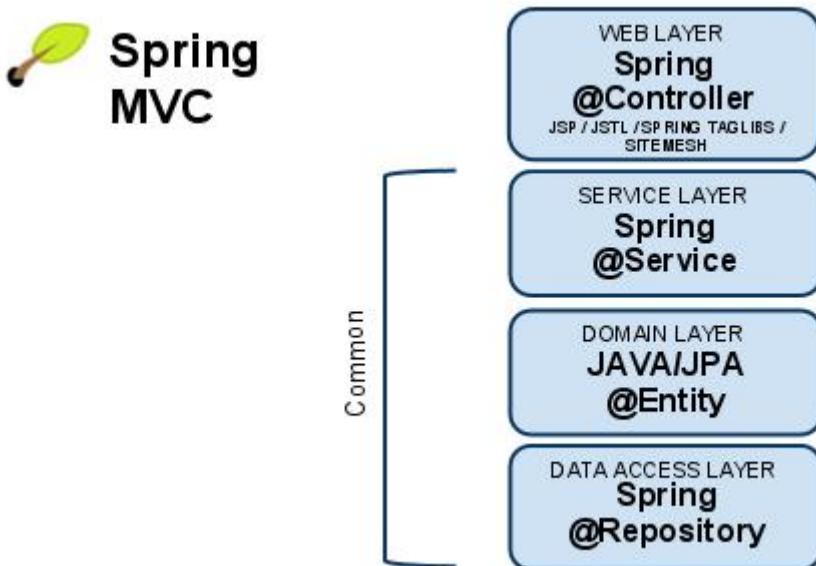
source : <https://sites.google.com/site/telosystutorial/springmvc-jpa-springdatajpa>



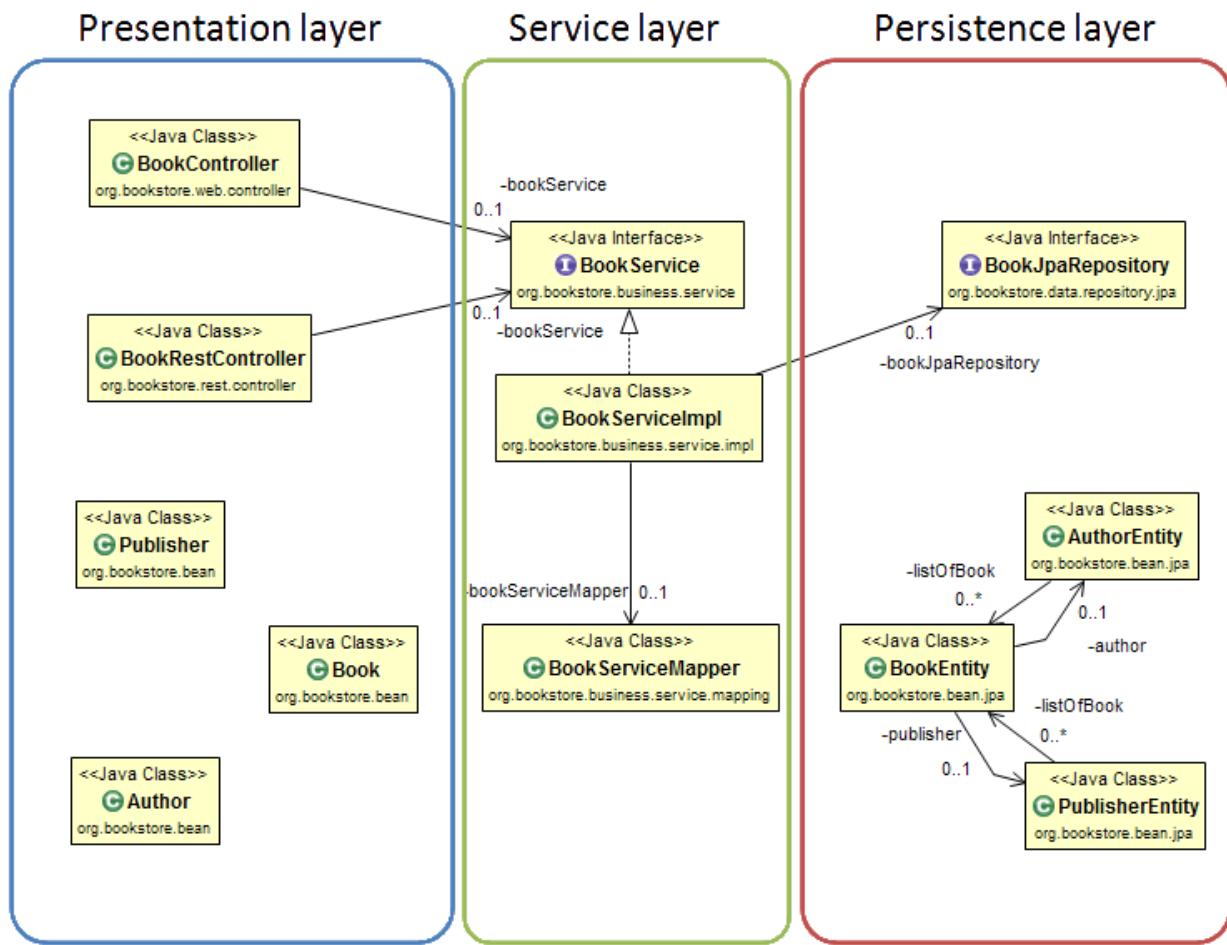
source : <https://fndong.wordpress.com/category/spring-framework/>



source : www.genuitec.com



180.3.1. Przykładowy podglądowy diagram klas w projekcie



180.4. EntityManager

- wstrzyknięcie jest zawsze wątkowo bezpieczne

181. Transakcja

- grupa operacji jest postrzegana jako jedna niezależna operacja
- grupuje operacje w całość, albo są akceptowane wszystkie albo żadna
- operacje w ramach transakcji mogą pochodzić z różnych źródeł i brać udział w wielu składnicach danych (2PC)

181.1. Wyjątki



Domyślnie tylko niesprawdzane wyjątki : `RuntimeException` + `Error` prowadzą do wycofania transakcji

181.2. ACID

- atomowość/Atomicity** - akceptowane są wszystkie w ramach procesu transakcji. W

przeciwnym wypadku są wycofywane.

- **spójność/Consistency** - koniec transakcji pozostawia system w stanie spójnym i stabilnym
- **izolacja/Isolation** - inne transakcje są nie widoczne dla danej w tej chwili wykonywanej
- **trwałość/Durability** - zakończone transakcje pozostają w stanie trwałym, to znaczy że awaria systemu nie będzie stanowiła problemu dla danych

181.3. Transakcje w kodzie

- uciążliwe kodowanie
- drogie utrzymanie
- boilerplate code

181.4. Transakcje deklaratywne

- większa elastyczność
- sterowane adnotacją
- sterowane plikami konfiguracyjnymi
- sterowane AOP
- spójny model JTA , JPA, JDBC , Hibernate
 - Przykład

```
@Transactional  
public class BookServiceImpl implements BookService {  
    @Transactional  
    public Book getBook(Long id) {  
        return repository.getBook(id);  
    }  
    @Transactional  
    public Book createBook(Book book) {  
        return repository.createBook(book);  
    }  
}
```

181.5. Atrybuty transakcji

181.5.1. Propagacja

- PROPAGATION_MANDATORY

source : http://docstore.mik.ua/oreilly/java-ent/ebeans/ch08_02.htm

[mandatory] | *mandatory.gif*

- metoda musi działać w ramach transakcji. Jeśli nie istnieje uruchomiona transakcja zgłoszany jest wyjątek

- **PROPAGATION_NESTED**

- pojedyńcza fizyczna transakcja z wieloma savepointami

- **PROPAGATION_NEVER**

- jeśli istnieje transakcja wyrzuć wyjątek.

- System typowo beztransakcyjny

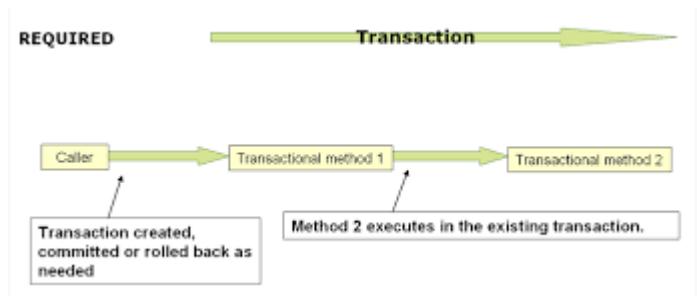
- **PROPAGATION_NOT_SUPPORTED**

source : http://docstore.mik.ua/oreilly/java-ent/ebeans/ch08_02.htm

[not supported] | *not_supported.gif*

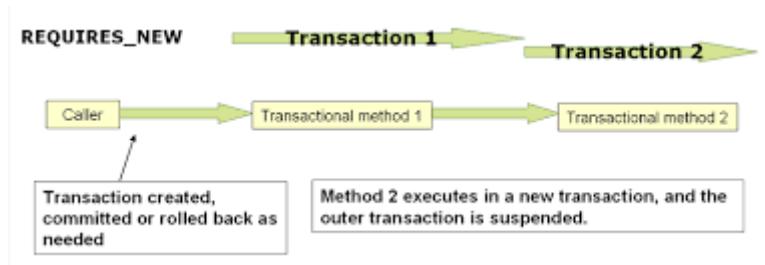
- **PROPAGATION_REQUIRED**

source : <http://docs.spring.io/spring/docs/3.1.x/spring-framework-reference/html/transaction.html>



- **PROPAGATION_REQUIRES_NEW**

source : <http://docs.spring.io/spring/docs/3.1.x/spring-framework-reference/html/transaction.html>



- **PROPAGATION_SUPPORTS**

source : http://docstore.mik.ua/oreilly/java-ent/ebeans/ch08_02.htm

[supported] | *supported.gif*

- jeśli istnieje transakcja metoda działa w jej ramach , w przeciwnym razie metoda jest wykonywana poza kontekstem transakcyjnym

181.5.2. izolacja

Stopień uniezależnienia od siebie poszczególnych transakcji
Wyszy poziom zapewnia lepszą separację - kosztem wydajności
Poprzez separacje rozumiemy, że jedna niezależna transakcja nie będzie wpływała na inną równoległą w systemie.

181.5.3. Problemy związane z izolacją

Utrata aktualizacji

Brudny odczyt

Czas	Akcja
T1	T1 się rozpoczęła
T2	T2 się rozpoczęła
T3	T1 uaktualnia rekord ROW1
T4	T2 czyta niezakomitetowany rekord ROW1
T5	T1 wycofuje transakcję
T6	T2 komituje

Niepowtarzalny odczyt

Czas	Akcja
T1	T1 się rozpoczęła
T2	T1 czyta rekord ROW1
T3	T2 się rozpoczęła
T4	T2 uaktualnia rekord ROW1
T5	T2 komituje
T6	T1 czyta rekord ROW1 który teraz jest inny niż ostatnio
T7	T1 komituje

Problem utraty wcześniejszego zapisu

Otrzymanie fantomu

Czas	Akcja
T1	T1 się rozpoczęła

Czas	Akcja
T2	T1 czyta zbiór rekordów
T3	T2 się rozpoczęła
T4	T2 wstawia nowy rekord
T5	T2 komituje
T6	T1 czyta zbiór rekordów który teraz ma inny rozmiar
T7	T1 komituje

Odczyt niezatwierdzonych (Read uncommited)

- brak izolacji
- Jedna transakcja ma dostęp do danych modyfikowanych przez inne transakcji
- Dopuszcza brudne odczyty, ale uniemożliwia utratę aktualizacji

```
@Transactional(isolation = Isolation.READ_UNCOMMITTED)
public Order getOrder(Long orderId) {
    return repository.getOrder(orderId);
}
```

Odczyt zatwierdzonych (Read commited)



Domyśla dla większości baz DBMSs

- bieżąca transakcja widzi tylko dane zatwierdzone.
- problem z długimi transakcjami, gdyż bieżąca transakcja będzie odczytywała dane zatwierdzone chwilowo przez inne.
- transakcja odczytuje dwa razy te same dane może zwrócić inne wyniki
- nie zapewnia powtarzalnego odczytu ale zapobiega brudnym odczytom

```
@Transactional(isolation = Isolation.READ_COMMITTED)
public Order getOrder(Long orderId) {
    return repository.getOrder(orderId);
}
```

Odczyt powtarzalny (Repeatable read)

- bieżąca transakcja widzi zmiany zatwierdzone już po jej rozpoczęciu przez inne równoległe transakcje.
- ma zapewnioną powtarzalność odczytów
- rozwiązuje problemy brudnego i niepowtarzalnego odczytu

- mogą wystąpić fantomy

Szeregowalność (Serializable)

- transakcja szeregowana. Możemy te sposób izolacji traktować synchroniczne wykonywane krok po kroku zamiast opcji zrównoleglenia.
- bardzo możliwe są konflikty.

181.5.4. Wybór poziomu

- odrzucamy poziom odczytu niezatwierdzonego.(niezatwierdzone zmiany z innych transakcji są bardzo groźne)
- odrzucamy również górny poziom serializable. Najtrudniej poddaje się współbieżności.
- powtarzalny - eliminacja nadpisywania przez inną transakcję. Zwiększa powtarzalność wyników zapytań.
- zatwierdzony + wersjonowanie wydajne się być w większości przypadków najlepszą praktyką.

181.6. Podsumowanie : który poziom na co pozwala :)

Poziom izolacji	Brudny odczyt	Niepowtarzalny odczyt	Fantomowy Odczyt
Szeregowalny			
Powtarzalny			zezwolenie
Odczyt zatwierdzonych		zezwolenie	zezwolenie
Odczyt niezatwierdzonych	zezwolenie	zezwolenie	zezwolenie

181.7. read only

- Przykład

```
@Transactional(readOnly = true)
public Account getAccount(Long accountId) {
    return repository.getAccount(accountId);
}
```

181.8. timeout

- Przykład

```
@Transactional(timeout = 60)
public List<Order> getActiveOrders(String from, String to) {
}
```

181.9. noRollbackFor

- wyjątki dla których wyrzucenie przez metodę nie spowoduje wycofanie transakcji
 - Przykład

```
@Transactional(noRollbackFor = MailException.class)
public void sendJobSuccessMessage(String jobName, String message);
```

181.10. rollbackFor

- wyjątki dla których wyrzucenie przez metodę może spowodować wycofanie transakcji
 - Przykład

```
@Transactional(rollbackFor = Exception.class)
void createUser(final RegisterForm registerForm) throws Exception;
```

182. Konfiguracja :

- Przykład

```
@EnableJpaRepositories(basePackages = "pl.java.scalatech.repository")
@PropertySource("classpath:spring-data.properties")
@Slf4j
@Import(Metrics2Config.class)
public abstract class JpaConfig {

    @Autowired
    Optional<MetricRegistry> metricRegistry;

    @Autowired
    private Environment env;

    @Value("${dataSource.driverClassName}")
    protected String driver;

    @Value("${dataSource.url}")
    protected String url;
```

```

    @Value("${dataSource.username}")
    protected String username;

    @Value("${dataSource.password}")
    protected String password;

    @Value("${hibernate.dialect}")
    protected String dialect;

    @Value("${hibernate.hbm2ddl.auto}")
    protected Boolean hbm2ddlAuto;

    @Value("${hibernate.show.sql}")
    protected Boolean showSql;

    @Value("${jpa.package}")
    protected String jpaPackage;

    @Value("${jpa.hikariMaxPoolSize}")
    protected int maxPoolSize;

    @Value("${jpa.hikariConnectionTimeoutMs}")
    protected long connectionTimeoutMs;

    @Value("${jpa.hikariIdleTimeoutMs}")
    protected long idleTimeoutMs;

    @Value("${jpa.hikariMaxLifetimeMs}")
    protected long maxLifetimeMs;

    @Value("${jpa.hikariRegisterMbeans}")
    protected boolean registerMbeans;

    public abstract void dataSourceConfigure(HikariConfig hikariConfig) throws
SQLException;
    public abstract Database DataBase();

    @Bean
    public DataSource datasource() throws SQLException{
        HikariConfig config = new HikariConfig();
        dataSourceConfigure(config);
        config.setMaximumPoolSize(maxPoolSize);
        // config.setConnectionTimeout(connectionTimeoutMs);
        config.setIdleTimeout(idleTimeoutMs);
        config.setMaxLifetime(maxLifetimeMs);
        config.setRegisterMbeans(registerMbeans);
    }

```

```

        config.setPoolName("pool");
        if(metricRegistry.isPresent()){
            config.setMetricRegistry(metricRegistry.get());
        }
        HikariDataSource dataSource = new HikariDataSource(config);
        return dataSource;
    }

    @Bean
    public PlatformTransactionManager transactionManager() {
        return new JpaTransactionManager();
    }

    @Bean
    public PersistenceExceptionTranslationPostProcessor exceptionTranslation() {
        return new PersistenceExceptionTranslationPostProcessor();
    }

    public Map<String, Object> jpaProperties() {
        Map<String, Object> props = new HashMap<>();
        /*
         * props.put("hibernate.cache.use_query_cache", "true");
         * props.put("hibernate.cache.region.factory_class",
         "org.hibernate.cache.ehcache.EhCacheRegionFactory");
         * props.put("hibernate.cache.provider_class",
         "org.hibernate.cache.ehcache.EhCacheRegionFactory");
         * props.put("hibernate.cache.use_second_level_cache", "true");
         */
        return props;
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory() throws
SQLException {
        log.info("++ entityManagerFactory started ...");
        LocalContainerEntityManagerFactoryBean lef = new
LocalContainerEntityManagerFactoryBean();
        lef.setJpaDialect(customJpaDialect());
        lef.setDataSource(datasource());
        lef.setJpaVendorAdapter(jpaVendorAdapter());
        lef.setJpaPropertyMap(jpaProperties());
        lef.setPackagesToScan(jpaPackage); // eliminate persistence.xml
        return lef;
    }

    @Bean
    public JpaVendorAdapter jpaVendorAdapter() {
        HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new
HibernateJpaVendorAdapter();
        hibernateJpaVendorAdapter.setShowSql(showSql);
        hibernateJpaVendorAdapter.setGenerateDdl(hbm2ddlAuto);
    }
}

```

```

        hibernateJpaVendorAdapter.setDatabase(dataBase());
        hibernateJpaVendorAdapter.setDatabasePlatform(dialect);
        return hibernateJpaVendorAdapter;
    }

    @Bean
    public Log4JdbcCustomFormatter logFormater() {
        Log4JdbcCustomFormatter formatter = new Log4JdbcCustomFormatter();
        formatter.setLoggingType(LoggingType.SINGLE_LINE);
        formatter.setSqlPrefix("SQL:\r");
        return formatter;
    }

    public JpaDialect customJpaDialect() {
        return new CustomHibernateJpaDialect();
    }
}

```

182.1. Strategia dla bazy wbudowanej :

- Przykład

```

@Configuration
@Slf4j
@Profile(value = "test")
@Order(10001)
public class JpaEmbeddedConfig extends JpaConfig {

    @Override
    public Database dataBase() {
        return Database.H2;
    }

    @Override
    public void dataSourceConfigure(HikariConfig config) throws SQLException {
        config.setDataSourceClassName("org.h2.jdbcx.JdbcDataSource");
        config.setConnectionTestQuery("VALUES 1");
        config.addDataSourceProperty("URL", "jdbc:h2:~/test");
        config.addDataSourceProperty("user", "sa");
        config.addDataSourceProperty("password", "");

    }

}

```

182.2. Baza wbudowana / konfiguracja xml

- Przykład

```
<jdbc:embedded-database id="dataSource" type="HSQL|H2|Derby">
<jdbc:script location="classpath:db-schema.sql"/>
<jdbc:script location="classpath:test-data.sql"/>
</jdbc:embedded-database>

<bean class="pl.java.scalatech.repository.jdbc.BookJdbcRepository">
<property name="dataSource" ref="dataSource"/>
</bean>
```

183. H2 w konsoli WEB

- Przykład

```
@Configuration
@Profile(value="h2")
@Order(10001)
public class H2Database extends JpaConfig{

    @Bean(destroyMethod = "close")
    @DependsOn(value = "h2Server")
    DataSource dataSource(Server h2Server) throws SQLException {
        HikariConfig hikariConfig = new HikariConfig();
        dataSourceConfigure(hikariConfig);
        hikariConfig.setMaximumPoolSize(maxPoolSize);
        hikariConfig.setConnectionTimeout(connectionTimeoutMs);
        hikariConfig.setIdleTimeout(idleTimeoutMs);
        hikariConfig.setMaxLifetime(maxLifetimeMs);
        hikariConfig.setRegisterMbeans(registerMbeans);
        hikariConfig.setConnectionTestQuery("VALUES 1");
        hikariConfig.addDataSourceProperty("useServerPrepStmts", username);
        HikariDataSource dataSource = new HikariDataSource(hikariConfig);

        createTcpServer();
        // CodaHaleMetricsTracker cmt = new CodaHaleMetricsTracker(pool,
        dataSource.getMetricRegistry());
        return dataSource;
    }

    @Bean(name = "h2Server", initMethod = "start", destroyMethod = "stop")
    @DependsOn(value = "h2WebServer")
    public org.h2.tools.Server createTcpServer() throws SQLException {
        return org.h2.tools.Server.createTcpServer("-tcp,-tcpAllowOthers,-
tcpPort,9092".split(","));
    }
}
```

```

    }

    @Bean(name = "h2WebServer", initMethod = "start", destroyMethod = "stop")
    public org.h2.tools.Server createWebServer() throws SQLException {
        return org.h2.tools.Server.createWebServer("-web,-webAllowOthers,-
webPort,8082".split(","));
    }

    @Override
    public Database DataBase() {
        return Database.H2;
    }

    @Override
    public void dataSourceConfigure(HikariConfig hikariConfig) throws SQLException {
        hikariConfig.addDataSourceProperty("url", url);
        hikariConfig.setUsername(username);
        hikariConfig.setPassword(password);
        hikariConfig.setDataSourceClassName(driver);
    }
}

```

184. Custom JPA = rozwiązywanie problemów z izolacją transakcji

- Przykład

```

public class CustomHibernateJpaDialect extends HibernateJpaDialect {

    private static final long serialVersionUID = 1L;

    /*
     * This method is overridden to set custom isolation levels on the connection
     * (non-Javadoc)
     * @see
     org.springframework.orm.jpa.vendor.HibernateJpaDialect#beginTransaction(javax.persistence.EntityManager, org.springframework.transaction.TransactionDefinition)
     */
    @Override
    public Object beginTransaction(final EntityManager entityManager,
        final TransactionDefinition definition)
        throws PersistenceException, SQLException, TransactionException {

        Session session = (Session) entityManager.getDelegate();
        if (definition.getTimeout() != TransactionDefinition.TIMEOUT_DEFAULT) {
            getSession(entityManager).getTransaction().setTimeout(
                definition.getTimeout());
        }
    }
}

```

```

    }

    final TransactionData data = new TransactionData();

    session.doWork(new Work() {
        @Override
        public void execute(Connection connection) throws SQLException {
            Integer previousIsolationLevel = DataSourceUtils
                .prepareConnectionForTransaction(connection, definition);
            data.setPreviousIsolationLevel(previousIsolationLevel);
            data.setConnection(connection);
        }
    });
}

entityManager.getTransaction().begin();

Object springTransactionData = prepareTransaction(entityManager,
    definition.isReadOnly(), definition.getName());

data.setSpringTransactionData(springTransactionData);

return data;
}

@Override
public void cleanupTransaction(Object transactionData) {
    super.cleanupTransaction(((TransactionData) transactionData)
        .getSpringTransactionData());
    ((TransactionData) transactionData).resetIsolationLevel();
}

private static class TransactionData {

    private Object springTransactionData;
    private Integer previousIsolationLevel;
    private Connection connection;

    public TransactionData() {
    }

    public void resetIsolationLevel() {
        if (this.previousIsolationLevel != null) {
            DataSourceUtils.resetConnectionAfterTransaction(connection,
                previousIsolationLevel);
        }
    }

    public Object getSpringTransactionData() {
        return this.springTransactionData;
    }
}

```

```

public void setSpringTransactionData(Object springTransactionData) {
    this.springTransactionData = springTransactionData;
}

public void setPreviousIsolationLevel(Integer previousIsolationLevel) {
    this.previousIsolationLevel = previousIsolationLevel;
}

public void setConnection(Connection connection) {
    this.connection = connection;
}

}

```

185. TransacionTemplate

- Przykład

```

<bean id="transactionManager" class=
"org.springframework.orm.jpa.JpaTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>

<bean id="transactionTemplate" class=
"org.springframework.transaction.support.TransactionTemplate">
    <property name="transactionManager" ref="transactionManager"/>
</bean>

<bean id="bookService" class="pl.java.scalatech.BookServiceImpl">
    <property name="transactionTemplate" ref="transactionTemplate" />
</bean>

```

185.1. Użycie

```

@Override
public void deleteBooks(final List<Book> books) {
    transactionTemplate.execute(new TransactionCallback() {
        @Override
        public Object doInTransaction(TransactionStatus status) {
            ...
            ...
        }
    });
}

```

186. Tworzenie repozytorium jpa/Hibernate

- Przykład

```
@Repository  
@Repository  
public class BookHibernateRepository implements BookRepository {  
    @Autowired  
    private DataSource dataSource;  
    private HibernateTemplate hibernate;  
    @Autowired  
    public BookHibernateRepository(DataSource dataSource) {  
        super  
        this.hibernateTemplate = new HibernateTemplate(dataSource);  
    }  
}
```

187. Tworzenie repozytorium jpa

```
@Repository  
@Repository  
public class BookJpaRepository implements BookRepository {  
  
    @Autowired  
    private EntityManager em;  
  
}
```

188. Praca z wieloma manadzerami transakcji

- Przykład

```
@Transactional(value = "txManager1")  
public BigDecimal getAndSavePrice (Book book) {}  
  
@Transactional(value = "txManager2")  
public Book merge(Book book ) {}
```

188.1. Ulepszenia / swoje adnotacje

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("txManager1")
public @interface BookTx {}  
  

@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("txManager2")
public @interface BookSecondTx {}  
  

@OrderTx
public BigDecimal getAndSavePrice (Book book) {}  
  

@BookSecondTx
public Book merge(Book book ) {}
```

189. Dodatek

189.1. Wsparcie JDBC

- JdbcTemplate
- NamedParameterJdbcTemplate
- SimpleJdbcTemplate
- SimpleJdbcInsert
- SimpleJdbcCall

189.2. Tworzenie repozytorium jdbc

- Przykład

```
@Repository  
@Repository  
public class BookJdbcRepository implements BookRepository {  
    @Autowired  
    private DataSource dataSource;  
    private JdbcTemplate jdbcTemplate;  
    @Autowired  
    public BookJdbcRepository(DataSource dataSource) {  
        super  
        this.jdbcTemplate = new JdbcTemplate(dataSource);  
    }  
}
```

190. Spring Data

191. Spring Data

191.1. Najnowsze features:)

- Support for Projections in repository query methods.
- Support for Query by Example.
- The following annotations have been enabled to build own, composed annotations: @EntityGraph, @Lock, @Modifying, @Query, @QueryHints and @Procedure.
- Support for Contains keyword on collection expressions.
- AttributeConverters for ZoneId of JSR-310 and ThreeTenBP.
- Upgrade to Querydsl 4, Hibernate 5, OpenJPA 2.4 and EclipseLink 2.6.1.

191.2. Geneza

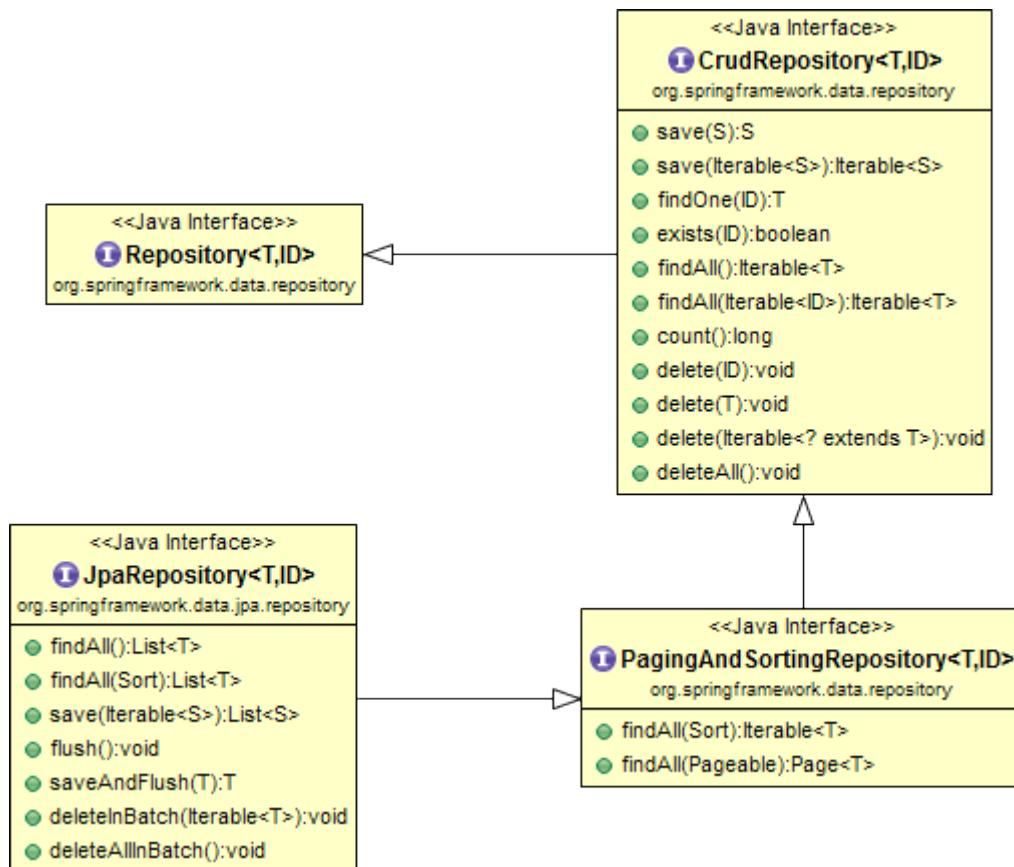
Powstał 2010 podczas sesji Roda Jonsona z Emilem Eifrem (Neo Technologies), jako próba integracji Spring z Neo4j

191.3. Charakretyстика

- generyczna impl DAO
- postawowa impl CRUD
- kod dao generowany poprzez definicję metody np findByName , findByAgeBetween
- paging i sortowanie w standardzie
- wsparcie dla JPA, Mongo, Neo4j, Redis itd

- wsparcie dla QueryDSL

source : www.patrick-gotthard.de



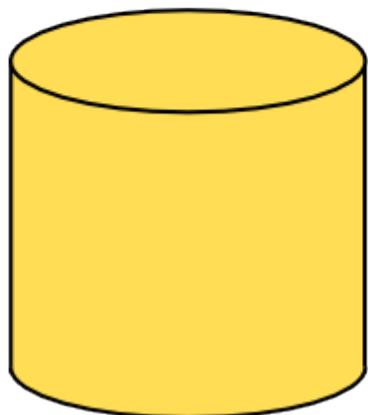
source: <https://visola.github.io>

Spring Data Common
(Abstract Repository Layer)

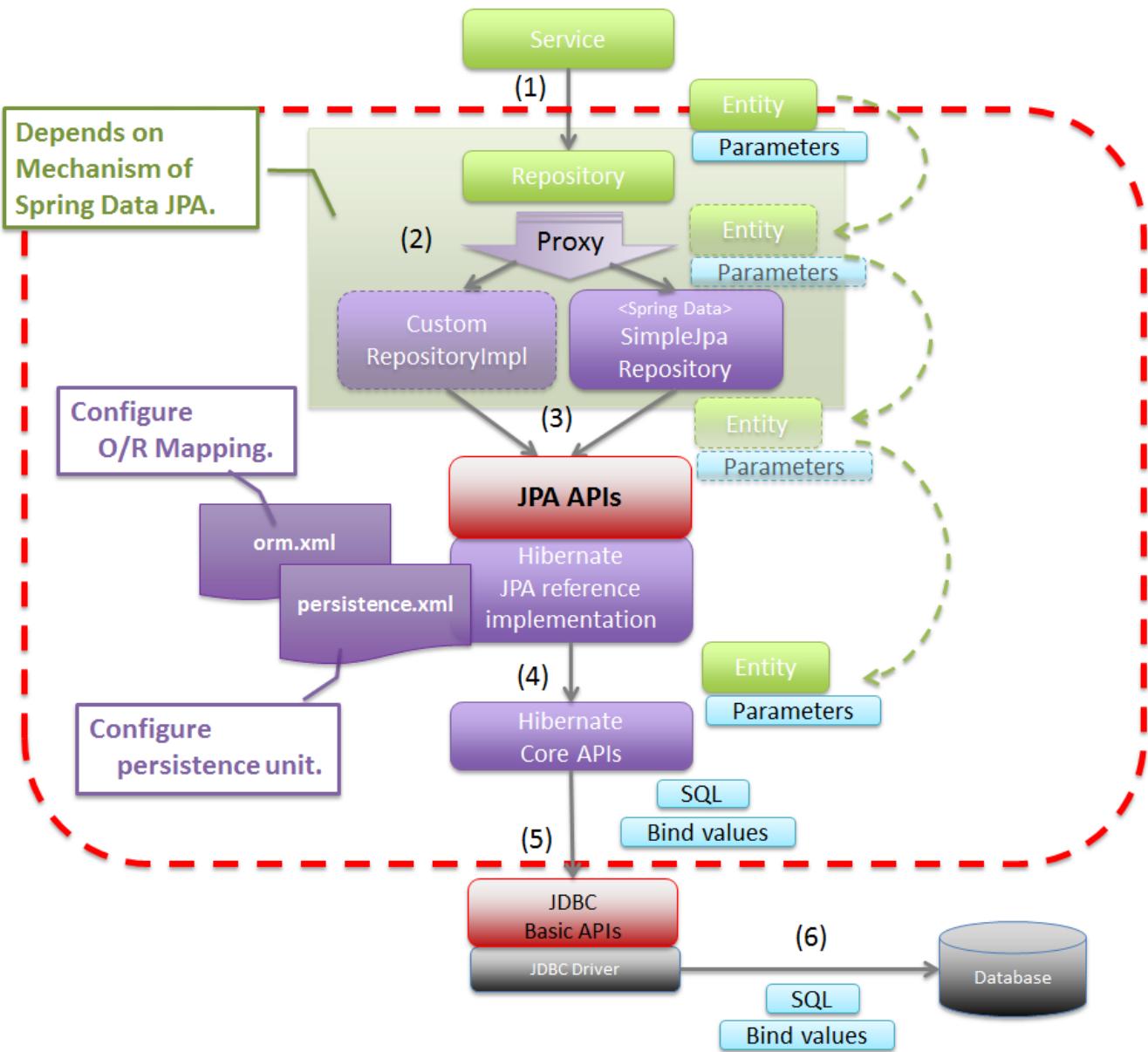
Spring Data JPA
(Specific Repository Management)

JPA Repository
(javax.persistence)

Data Source
(java.sql and javax.sql)



source: <http://luxosuseadicto.blogspot.com>



191.3.1. Przykład całej minimalnej konfiguracji

- Przykład

```

@Configuration
@EnableJpaRepositories
@EnableTransactionManagement
class ApplicationConfig {

    @Bean
    public DataSource dataSource() {

        EmbeddedDatabaseBuilder builder = new EmbeddedDatabaseBuilder();
        return builder.setType(EmbeddedDatabaseType.HSQL).build();
    }

    @Bean
    public EntityManagerFactory entityManagerFactory() {

        HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        vendorAdapter.setGenerateDdl(true);

        LocalContainerEntityManagerFactoryBean factory = new
        LocalContainerEntityManagerFactoryBean();
        factory.setJpaVendorAdapter(vendorAdapter);
        factory.setPackagesToScan("com.acme.domain");
        factory.setDataSource(dataSource());
        factory.afterPropertiesSet();

        return factory.getObject();
    }

    @Bean
    public PlatformTransactionManager transactionManager() {

        JpaTransactionManager txManager = new JpaTransactionManager();
        txManager.setEntityManagerFactory(entityManagerFactory());
        return txManager;
    }
}

```

191.3.2. Tworzenie proxy

- Przykład

```

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableJpaRepositories
class Config {}

```

191.3.3. Annotation-driven configuration

- Przykład

```
@EnableJpaRepositories(basePackages = "com.acme.repositories.jpa")
@EnableMongoRepositories(basePackages = "com.acme.repositories.mongo")
interface Configuration { }
```

191.3.4. Standalone

- Przykład

```
RepositoryFactorySupport factory = ... // Instantiate factory here
UserRepository repository = factory.getRepository(UserRepository.class);
```

191.3.5. Wybór metod CRUD

- Przykład

```
@NoRepositoryBean
interface MyBaseRepository<T, ID extends Serializable> extends Repository<T, ID> {

    T findOne(ID id);

    T save(T entity);
}

interface UserRepository extends MyBaseRepository<User, Long> {
    User findByEmailAddress(EmailAddress emailAddress);
}
```

191.3.6. Wykorzystanie Pageable, Slice, Sort

- Przykład

```
Page<User> findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

191.3.7. Ograniczenie wyników zapytań

- Przykład

```
User findFirstByOrderByLastnameAsc();  
  
User findTopByOrderByAgeDesc();  
  
Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);  
  
Slice<User> findTop3ByLastname(String lastname, Pageable pageable);  
  
List<User> findFirst10ByLastname(String lastname, Sort sort);  
  
List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

191.3.8. Streaming

- Przykład

```
@Query("select u from User u")  
Stream<User> findAllByCustomQueryAndStream();  
  
Stream<User> readAllByFirstnameNotNull();  
  
@Query("select u from User u")  
Stream<User> streamAllPaged(Pageable pageable);  
  
///  
  
try (Stream<User> stream = repository.findAllByCustomQueryAndStream()) {  
    stream.forEach(...);  
}
```

191.3.9. Asynchroniczność

- Przykład

```
@Async  
Future<User> findByFirstname(String firstname);  
  
@Async  
CompletableFuture<User> findOneByFirstname(String firstname);  
  
@Async  
ListenableFuture<User> findOneByLastname(String lastname);
```

191.3.10. Dostrajanie do swoich potrzeb

- Przykład

```
interface UserRepositoryCustom {  
    public void someCustomMethod(User user);  
}  
  
class UserRepositoryImpl implements UserRepositoryCustom {  
  
    public void someCustomMethod(User user) {  
        // Your custom implementation  
    }  
}  
  
interface UserRepository extends CrudRepository<User, Long>, UserRepositoryCustom {  
  
    // Declare query methods here  
}
```

191.3.11. DSL

- Przykład

```
public interface QueryDslPredicateExecutor<T> {

    T findOne(Predicate predicate);

    Iterable<T> findAll(Predicate predicate);

    long count(Predicate predicate);

    boolean exists(Predicate predicate);

    // ... more functionality omitted.

}

interface UserRepository extends CrudRepository<User, Long>,
QueryDslPredicateExecutor<User> {

}

}
```

- Przykład

```
Predicate predicate = user.firstname.equalsIgnoreCase("dave")
    .and(user.lastname.startsWithIgnoreCase("mathews"));

userRepository.findAll(predicate);
```

191.4. Nazwane zapytania

- Przykład

```

@Entity
@NamedQuery(name = "User.findByEmailAddress",
    query = "select u from User u where u.emailAddress = ?1")
public class User {

}

public interface UserRepository extends JpaRepository<User, Long> {

    List<User> findByLastname(String lastname);

    User findByEmailAddress(String emailAddress);
}

```

191.4.1. @Query

- Przykład

```

public interface UserRepository extends JpaRepository<User, Long> {

    @Query("select u from User u where u.emailAddress = ?1")
    User findByEmailAddress(String emailAddress);
}

public interface UserRepository extends JpaRepository<User, Long> {

    @Query("select u from User u where u.firstname like %?1")
    List<User> findByFirstnameEndsWith(String firstname);
}

```

191.4.2. Natywne zapytania

- Przykład

```

public interface UserRepository extends JpaRepository<User, Long> {

    @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1", nativeQuery = true)
    User findByEmailAddress(String emailAddress);
}

public interface UserRepository extends JpaRepository<User, Long> {

    @Query("select u from User u where u.firstname = :firstname or u.lastname = :lastname")
    User findByLastnameOrFirstname(@Param("lastname") String lastname,
                                    @Param("firstname") String firstname);
}

```

191.4.3. SpEL expressions

- Przykład

```

@Entity
public class User {

    @Id
    @GeneratedValue
    Long id;

    String lastname;
}

public interface UserRepository extends JpaRepository<User, Long> {

    @Query("select u from #{#entityName} u where u.lastname = ?1")
    List<User> findByLastname(String lastname);
}

```

191.4.4. Modyfikacja danych

```

@Modifying
@Query("update User u set u.firstname = ?1 where u.lastname = ?2")
int setFixedFirstnameFor(String firstname, String lastname);

```

191.4.5. Hint

- Przykład

```

public interface UserRepository extends Repository<User, Long> {

    @QueryHints(value = { @QueryHint(name = "name", value = "value") },
                forCounting = false)
    Page<User> findByLastname(String lastname, Pageable pageable);
}

```

191.4.6. Fetch load EntityGraph

- Przykład

```

@Entity
@NamedEntityGraph(name = "GroupInfo.detail",
    attributeNodes = @NamedAttributeNode("members"))
public class GroupInfo {

    // default fetch mode is lazy.
    @ManyToMany
    List<GroupMember> members = new ArrayList<GroupMember>();

    ...

}

@Repository
public interface GroupRepository extends CrudRepository<GroupInfo, String> {

    @EntityGraph(value = "GroupInfo.detail", type = EntityGraphType.LOAD)
    GroupInfo getByName(String name);

}

@Repository
public interface GroupRepository extends CrudRepository<GroupInfo, String> {

    @EntityGraph(attributePaths = { "members" })
    GroupInfo getByName(String name);

}

```

191.4.7. Projection

- Przykład

```

@Entity
public class Person {

    @Id @GeneratedValue
    private Long id;
    private String firstName, lastName;

    @OneToOne
    private Address address;
    ...
}

@Entity
public class Address {

    @Id @GeneratedValue
    private Long id;
    private String street, state, country;

    ...
}

interface PersonRepository extends CrudRepository<Person, Long> {

    Person findPersonByFirstName(String firstName);
}

interface AddressRepository extends CrudRepository<Address, Long> {}

interface NoAddresses {

    String getFirstName();

    String getLastName();
}

```

191.4.8. Procedury składowane

- Przykład

```

/;
DROP procedure IF EXISTS plus1inout
/;
CREATE procedure plus1inout (IN arg int, OUT res int)
BEGIN ATOMIC
    set res = arg + 1;
END
/;

```

```

@Entity
@NamedStoredProcedureQuery(name = "User.plus1", procedureName = "plus1inout",
parameters = {
    @StoredProcedureParameter(mode = ParameterMode.IN, name = "arg", type = Integer
.class),
    @StoredProcedureParameter(mode = ParameterMode.OUT, name = "res", type = Integer
.class) })
public class User {}

@Procedure("plus1inout")
Integer explicitlyNamedPlus1inout(Integer arg);

@Procedure(procedureName = "plus1inout")
Integer plus1inout(Integer arg);

@Procedure(name = "User.plus1IO")
Integer entityAnnotatedCustomNamedProcedurePlus1IO(@Param("arg") Integer arg);

@Procedure
Integer plus1(@Param("arg") Integer arg);

```

191.4.9. Specifications

- Przykład

```

public interface CustomerRepository extends CrudRepository<Customer, Long>,
JpaSpecificationExecutor {
...
}

List<T> findAll(Specification<T> spec);

public interface Specification<T> {
    Predicate toPredicate(Root<T> root, CriteriaQuery<?> query,
        CriteriaBuilder builder);
}

public class CustomerSpecs {

    public static Specification<Customer> isLongTermCustomer() {
        return new Specification<Customer>() {
            public Predicate toPredicate(Root<Customer> root, CriteriaQuery<?> query,
                CriteriaBuilder builder) {

                LocalDate date = new LocalDate().minusYears(2);
                return builder.lessThan(root.get(_Customer.createdAt), date);
            }
        };
    }

    public static Specification<Customer> hasSalesOfMoreThan(MontaryAmount value) {
        return new Specification<Customer>() {
            public Predicate toPredicate(Root<T> root, CriteriaQuery<?> query,
                CriteriaBuilder builder) {

                // build query here
            }
        };
    }
}

// using

List<Customer> customers = customerRepository.findAll(isLongTermCustomer());

```

- Przykład 2

```

public class UserSpecifications {

    public static Specification<User> getUserByLogin(final String str) {
        return new Specification<User>() {
            @Override
            public Predicate toPredicate(Root<User> personRoot, CriteriaQuery<?>
query, CriteriaBuilder cb) {
                return cb.equal(personRoot.<String> get(User_.login), str);
            }
        };
    }

    public static Specification<User> getUsersWhoEarMoreThan(final BigDecimal salary)
{
        return new Specification<User>() {
            @Override
            public Predicate toPredicate(Root<User> personRoot, CriteriaQuery<?>
query, CriteriaBuilder cb) {
                return cb.greaterThan(personRoot.<BigDecimal> get(User_.salary),
salary);
            }
        };
    }

}
//
```

- Przykład

```

@Test
public void shouldSpecificationsPredicateWork() {
    assertThat(userRepository.findAll(getUserByLogin("przodownik"))).hasSize(1)
        .containsOnly(User.builder().login("przodownik").name("borowiec").salary
(new BigDecimal(120)).build());
    assertThat(userRepository.findAll(getUsersWhoEarMoreThan(new BigDecimal(300)))
).hasSize(2);
}
```

191.4.10. Query by Example

*** Przykład

```

public class Person {

    @Id
    private String id;
    private String firstname;
    private String lastname;
    private Address address;

    // ... getters and setters omitted
}

Person person = new Person();
person.setFirstname("Dave");

Example<Person> example = Example.of(person);

public interface QueryByExampleExecutor<T> {

    <S extends T> S findOne(Example<S> example);

    <S extends T> Iterable<S> findAll(Example<S> example);

    // ... more functionality omitted.
}

//example

Person person = new Person();
person.setFirstname("Dave");

ExampleMatcher matcher = ExampleMatcher.matching()
    .withIgnorePaths("lastname")
    .withIncludeNullValues()
    .withStringMatcherEnding();

Example<Person> example = Example.of(person, matcher);

```

191.4.11. Transakcyjność

- Przykład

```

public interface UserRepository extends CrudRepository<User, Long> {

    @Override
    @Transactional(timeout = 10)
    public List<User> findAll();

    // Further query method declarations
}

@Transactional(readOnly = true)
public interface UserRepository extends JpaRepository<User, Long> {

    List<User> findByLastname(String lastname);

    @Modifying
    @Transactional
    @Query("delete from User u where u.active = false")
    void deleteInactiveUsers();
}

```

191.4.12. Locking

```

interface UserRepository extends Repository<User, Long> {

    // Plain query method
    @Lock(LockModeType.READ)
    List<User> findByLastname(String lastname);
}

```

191.4.13. Audyt / Audit

- Przykład

```

class Customer {

    @CreatedBy
    private User user;

    @CreatedDate
    private DateTime createdDate;

    // ... further properties omitted
}

```

191.4.14. AuditorAware

- Przykład

```
class SpringSecurityAuditorAware implements AuditorAware<User> {

    public User getCurrentAuditor() {

        Authentication authentication = SecurityContextHolder.getContext()
            .getAuthentication();

        if (authentication == null || !authentication.isAuthenticated()) {
            return null;
        }

        return ((MyUserDetails) authentication.getPrincipal()).getUser();
    }
}
```

```
@Entity
@EntityListeners(AuditingEntityListener.class)
public class MyEntity {
```

```
}
```

```
@Configuration
@EnableJpaAuditing
class Config {

    @Bean
    public AuditorAware<AuditableUser> auditorProvider() {
        return new AuditorAwareImpl();
    }
}
```

191.5. Web support

191.5.1. Konfiguracja

- Przykład

```
@Configuration  
@EnableWebMvc  
@EnableSpringDataWebSupport  
class WebConfiguration { }
```

DomainClassConverter

- Przykład

```
@Controller  
@RequestMapping("/users")  
public class UserController {  
  
    @RequestMapping("/{id}")  
    public String showUserForm(@PathVariable("id") User user, Model model) {  
  
        model.addAttribute("user", user);  
        return "userForm";  
    }  
}
```

HandlerMethodArgumentResolver dostęp do Pageable i Sort z poziomu parametrów requesta

- Przykład

```
@Controller  
@RequestMapping("/users")  
public class UserController {  
  
    @Autowired UserRepository repository;  
  
    @RequestMapping  
    public String showUsers(Model model, Pageable pageable) {  
  
        model.addAttribute("users", repository.findAll(pageable));  
        return "users";  
    }  
}
```

Hypermedia wsparcie dla Pageables

- Przykład

```
@Controller
class PersonController {

    @Autowired PersonRepository repository;

    @RequestMapping(value = "/persons", method = RequestMethod.GET)
    HttpEntity<PagedResources<Person>> persons(Pageable pageable,
        PagedResourcesAssembler assembler) {

        Page<Person> persons = repository.findAll(pageable);
        return new ResponseEntity<>(assembler.toResources(persons), HttpStatus.OK);
    }
}
```

Querydsl web support / QuerydslPredicateArgumentResolver.

- Przykład

```
?firstname=Dave&lastname=Matthews

=>

QUser.user.firstname.eq("Dave").and(QUser.user.lastname.eq("Matthews"))
```

- Przykład

```

@Controller
class UserController {

    @Autowired UserRepository repository;

    @RequestMapping(value = "/", method = RequestMethod.GET)
    String index(Model model, @QuerydslPredicate(root = User.class) Predicate predicate,
                 Pageable pageable, @RequestParam MultiValueMap<String, String> parameters) {

        model.addAttribute("users", repository.findAll(predicate, pageable));

        return "index";
    }
}

interface UserRepository extends CrudRepository<User, String>,
    QueryDslPredicateExecutor<User>,
    QuerydslBinderCustomizer<QUser> {

    @Override
    default public void customize(QuerydslBindings bindings, QUser user) {

        bindings.bind(user.username).first((path, value) -> path.contains(value))
        bindings.bind(String.class)
            .first((StringPath path, String value) -> path.containsIgnoreCase(value));
        bindings.excluding(user.password);
    }
}

```

192. Envers

193. Envers

Pozwala na zarządzanie historią zmian w bazie. Możemy traktować to jako narzędzie podobne do SVN'a czy Git'a

193.1. Konfiguracja

```

<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-envers</artifactId>
<version>4.3.10.Final</version>
</dependency>

```

```
<listener class="org.hibernate.envers.event.AuditEventListener" type="post-insert"/>
<listener class="org.hibernate.envers.event.AuditEventListener" type="post-update"/>
<listener class="org.hibernate.envers.event.AuditEventListener" type="post-delete"/>
```

@Audited = poziom klasy. Hibernate uaktywni audyt dla danej encji. Jeśli obiekt encyjny będzie się zmieniał spowoduje to również zmiany w tabeli revision.

NotAudited != @Audited

194. Alternatywy QueryDSL

INFO == info adoc

Built-in

asciidocdoctor-version

1.5.4

safe-mode-name

unsafe

docdir

/home/przodownik/blog/springJpaCamp/docsPdf

docfile

/home/przodownik/blog/springJpaCamp/docsPdf/hib_jpa_pl.adoc

imagesdir

./img

Custom

sourcedir

./src/main/java/