



JPA / Hibernate Model Design & CRUD

Αθ. Ανδρούτσος



Παράδειγμα

Hibernate

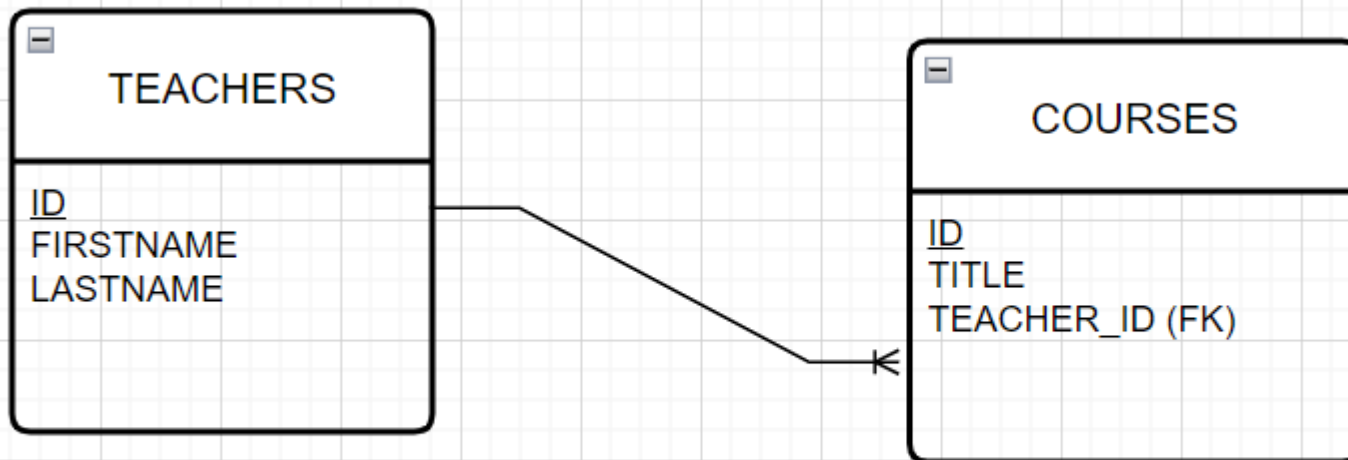
- Θα αναλύσουμε ένα αρχικό παράδειγμα για το πώς μπορούμε να υλοποιήσουμε ένα σχήμα μίας ΒΔ σε όρους JPA / Hibernate
- Καθώς και πως μπορούμε να υλοποιήσουμε τις βασικές CRUD πράξεις
- Θα δούμε τη βασική δομή του project και όλα τα βήματα και θα εξηγούμε κάθε βήμα



Simple School Domain

Hibernate

- Έστω ένα απλό *domain model* με Teachers και Courses. Έστω ένας Teacher σχετίζεται με πολλά Courses ενώ ένα Course σχετίζεται με ένα Teacher





Νέο Maven Project

Hibernate

New Project

Generators

- Maven Archetype
- Jakarta EE
- Spring Initializr
- JavaFX
- Quarkus
- Micronaut
- Ktor
- Compose for Desktop
- HTML
- React
- Express
- Angular CLI
- Vue.js

To create a general Maven project, go to the [New Project](#) page.

Name:

Location:

Project will be created in: ~\IdeaProjects\hiber6-dev1

☐ Create Git repository

JDK:

Catalog: [Manage catalogs...](#)

Archetype:

Version:

Additional Properties

+	-
junitVersion	5.11.0
javaCompilerVersion	17

Advanced Settings

GroupId:

ArtifactId:

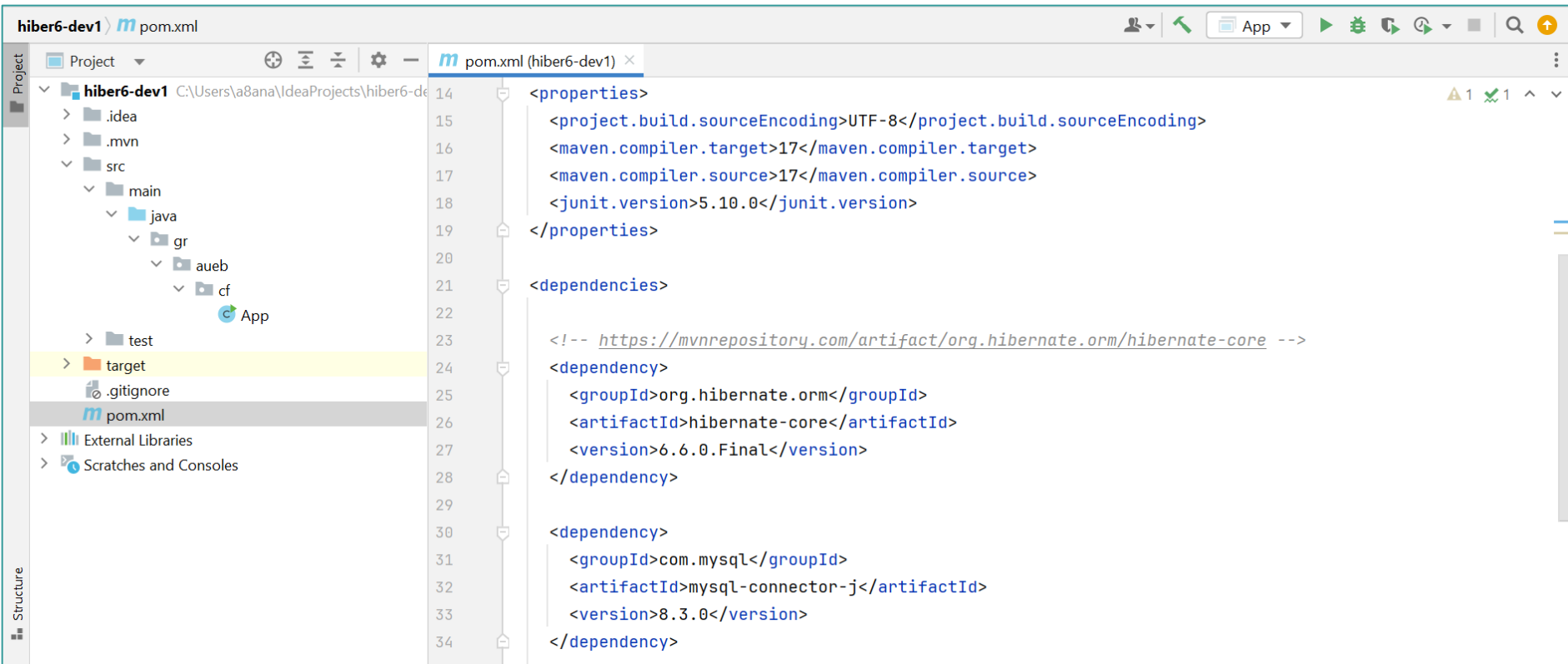
Version:

- Δημιουργούμε ένα νέο Maven quickstart project



POM.xml

Hibernate

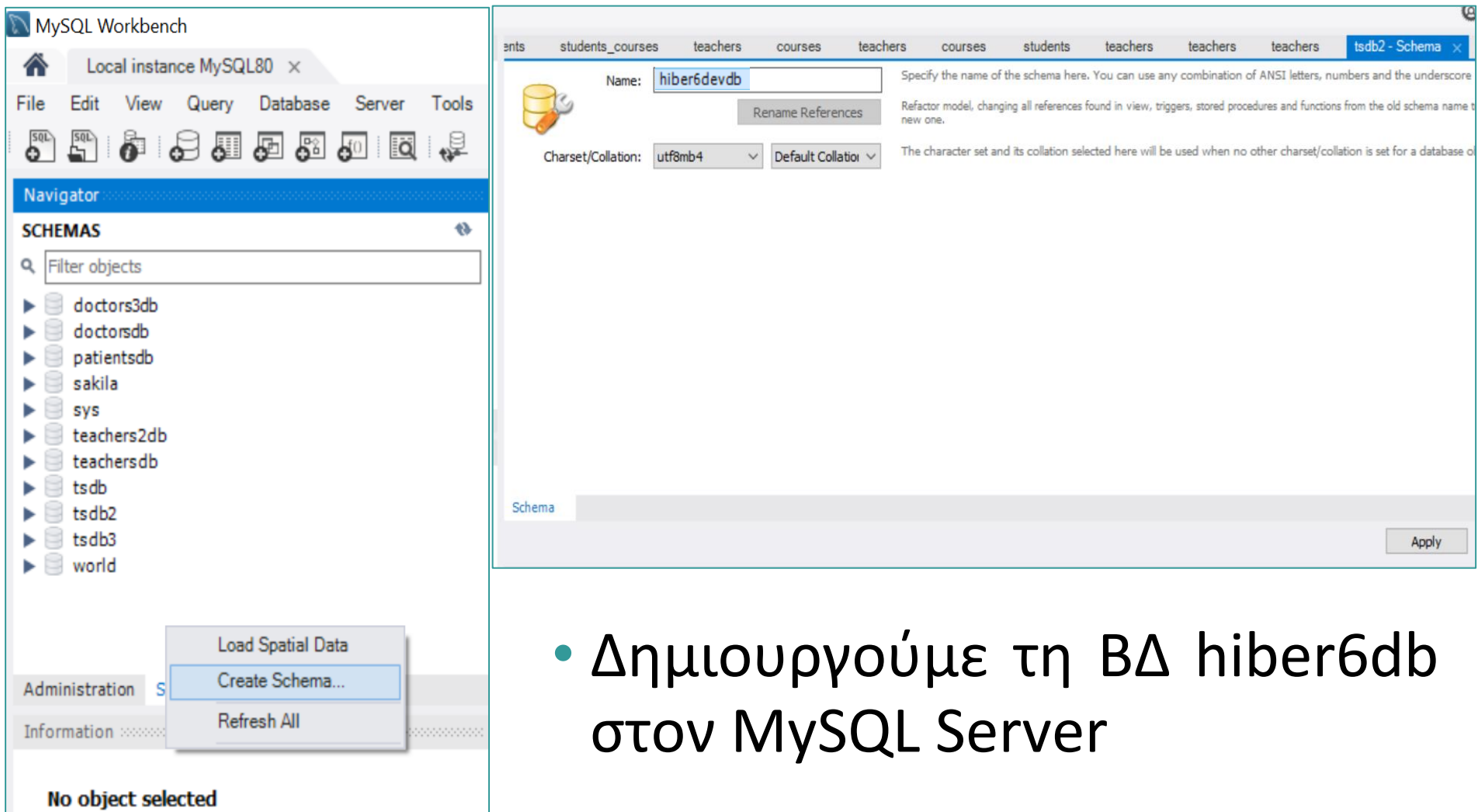


- Στο POM.xml εισάγουμε dependencies για το Hibernate (που περιλαμβάνει και τα JPA specs) καθώς και τον mysql-connector



MySQL

Hibernate



MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools

Navigator

SCHEMAS

Filter objects

- doctors3db
- doctorsdb
- patientsdb
- sakila
- sys
- teachers2db
- teachersdb
- tsdb
- tsdb2
- tsdb3
- world

Administration S Information

No object selected

Load Spatial Data

Create Schema...

Refresh All

Schema

Apply

- Δημιουργούμε τη ΒΔ hiber6db στον MySQL Server



Δημιουργία χρήστη (1)

Hibernate

Navigation: teachers teachers courses teachers courses teachers teachers courses Administration - Users and Privileges

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

Administration Schemas

Schema: hiber6devdb

Local instance MySQL80
Users and Privileges

User Accounts

User	From Host
codingplustest	%
docadmin	%
doctors5	%
favdbuser	%
hellouser	%
hiber6devuser	%
hiber6user	%
hiberuser5	%
kyduser	%
movdbuser	%
mydbuser	%
mydbuser6	%
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost
school4dbuser	%
school5appuser	%
school5user	%
school8dbuser	%

Details for account hiber6user@%

Login Account Limits Administrative Roles Schema Privileges

Login Name: hiber6user You may create multiple accounts with the same name to connect from different hosts.

Authentication Type: caching_sha2_password For the standard password and/or host based authentication, select 'Standard'.

Limit to Hosts Matching: % % and _ wildcards may be used

Password: ***** Type a password to reset it.

Consider using a password with 8 or more characters with mixed case letters, numbers and punctuation marks.

Confirm Password: ***** Enter password again to confirm.

Expire Password

Authentication String: \$A\$005\$11)pH}%[MuW]□□&□?□ Authentication plugin specific parameters.

Add Account Delete Refresh

Revert Apply

- Δημιουργούμε μέσα από τον MySQL Server ένα χρήστη hiber6user με authentication type caching-sha2-password και του δίνουμε δικαιώματα owner στη ΒΔ (βλ. επόμενη διαφάνεια)



Δημιουργία Χρήστη (2)

Hibernate

Local instance MySQL80
Users and Privileges

User Accounts

User	From Host
codingplustest	%
docadmin	%
doctors5	%
favdbuser	%
hellouser	%
hiber6devuser	%
hiber6user	%
hiberuser5	%
kyduser	%
movdbuser	%
mydbuser	%
mydbuser6	%
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost
school4dbuser	%
school5appuser	%
school5user	%
school8dbuser	%

Details for account hiber6user@%

Login Account Limits Administrative Roles Schema Privileges

Schema	Privileges
hiber6devdb	ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE VIEW, DELETE, EVENT, EX

Schema and Host fields may use % and _ wildcards.
The server will match specific entries before wildcarded ones.

The user 'hiber6user'@'%' will have the following access rights to the schema 'hiber6devdb':

Object Rights

- ☒ SELECT
- ☒ INSERT
- ☒ UPDATE
- ☒ DELETE
- ☒ EXECUTE
- ☒ SHOW VIEW

DDL Rights

- ☒ CREATE
- ☒ ALTER
- ☒ REFERENCES
- ☒ INDEX
- ☒ CREATE VIEW
- ☒ CREATE ROUTINE
- ☒ ALTER ROUTINE

Other Rights

- ☐ GRANT OPTION
- ☒ CREATE TEMPORARY TABLES
- ☒ LOCK TABLES

Revoke All Privileges Delete Entry Add Entry...

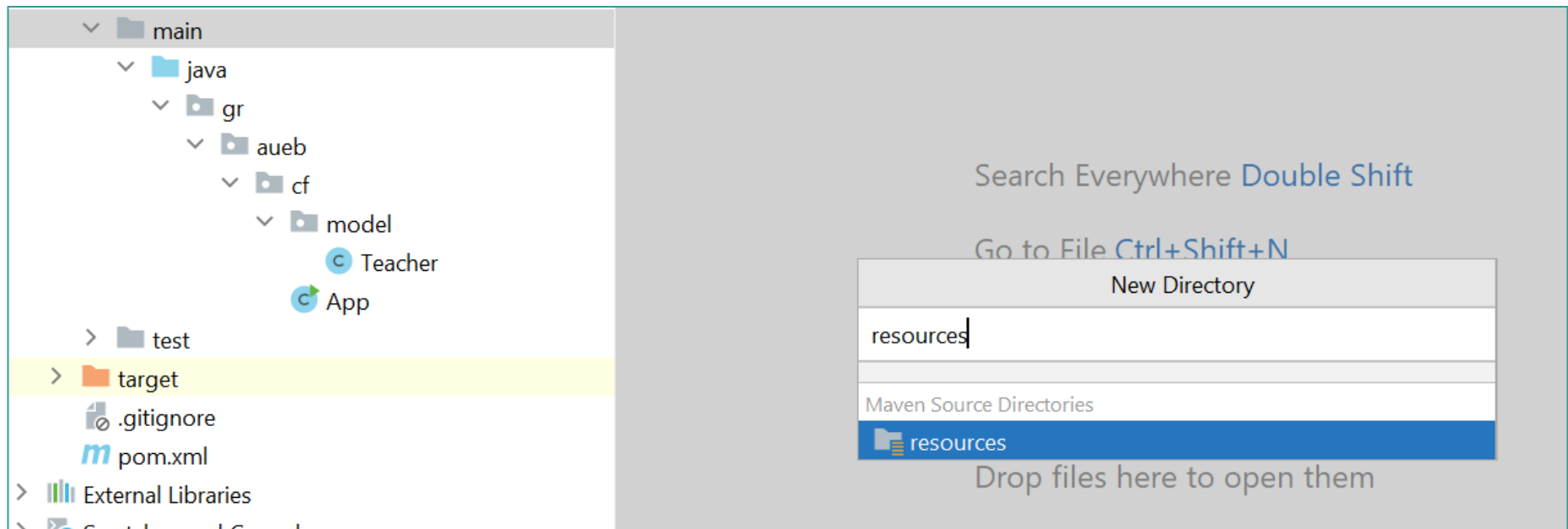
Add Account Delete Refresh Revert Apply

- Στο schema privileges επιλέγουμε Add Entry και τη ΒΔ hiber6devdb και επιλέγουμε όλα τα δικαιώματα εκτός από GRAND OPTION που είναι δικαίωμα απόδοσης δικαιωμάτων σε άλλους χρήστες



Néo directory resources

Hibernate



- Στο main εισάγουμε ένα νέο directory με όνομα resources (δεν το δίνει by default το quickstart archetype) με δεξί κλικ / new directory



Persistence.xml (1)

Hibernate

```
<?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="school1PU"
    transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <properties>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.connection.driver" value="com.mysql.cj.jdbc.Driver" />
      <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/hello6devdb?serverTimezone=UTC" />
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
      <property name="hibernate.connection.username" value="hellouser" />
      <property name="hibernate.connection.password" value="12345" />
      <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>
  </persistence-unit>
</persistence>
```

- Στο φάκελο resources εισάγουμε νέο φάκελο META-INF και μέσα στον META-INF εισάγουμε ένα αρχείο persistence.xml που είναι ένα config αρχείο του Hibernate



Persistence.xml (2)

Hibernate

- Μέσα στο persistence.xml, η βασική δομή είναι το **persistence-unit** (γρ. 8) όπου ορίζουμε όλα τα στοιχεία της σύνδεσης με τη ΒΔ. Στο Persistence-unit δίνουμε ένα όνομα (π.χ. **name = "school1PU"**)
- Η ιδιότητα **transaction-type** μπορεί να είναι είτε **RESOURCE_LOCAL** (η εφαρμογή μας είναι υπεύθυνη για τη δημιουργία του EntityManager) ή **JTA** (ο Application Server είναι υπεύθυνος για τη δημιουργία του EntityManager οπότε ο EntityManager μετά πρέπει να γίνει inject με το annotation @PersistenceContext ενώ ο EntityManagerFactory πρέπει να γίνει inject με @PersistenceUnit)



Entities Auto-scan

```
persistence.xml x
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
5          http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
6      version="2.0">
7
8      <persistence-unit name="school1PU"
9          transaction-type="RESOURCE_LOCAL">
10         <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
11
12         <properties>
13             <property name="hibernate.show_sql" value="true" />
14             <property name="hibernate.format_sql" value="true" />
15             <property name="hibernate.connection.driver" value="com.mysql.cj.jdbc.Driver" />
16             <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/hello6devdb?serverTimezone=UTC" />
17             <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
18             <property name="hibernate.connection.username" value="hellouser" />
19             <property name="hibernate.connection.password" value="12345" />
20             <property name="hibernate.hbm2ddl.auto" value="update" />
21         </properties>
22     </persistence-unit>
23 </persistence>
```

- Δεν έχουμε ορίσει συγκεκριμένες κλάσεις, οπότε όλες οι κλάσεις με annotation **@Entity** ανήκουν αυτόματα στο Persistent Unit.



Properties (1)

- Τα *JPA* και *Hibernate options* μπορούν να γίνουν set ως properties στο persistent unit
- Οι Hibernate properties ορίζονται με *hibernate.**
- Για παράδειγμα:
 - **hibernate.show.sql** - Αν true, ενεργοποιεί το logging των Hibernate generated SQL statements στην κονσόλα. Βλέπουμε δηλαδή τις εντολές που εκτελούνται
 - **hibernate.format.sql** – Αν true, μορφοποιεί τις Hibernate generated SQL statements στην κονσόλα και τις κάνει πιο readable, καταλαμβάνοντας περισσότερο χώρο



Properties (2)

- **hibernate.connection.driver_class**
 - Αναπαριστά την κλάση του JDBC driver
- **hibernate.connection.url**
 - Το JDBC URL για την ΒΔ
- **hibernate.dialect**
 - Η ιδιότητα αυτή ορίζει την συγκεκριμένη SQL version, ώστε το hibernate να δημιουργήσει τις κατάλληλες SQL εντολές για την επιλεγμένη ΒΔ



Properties (3)

- **hibernate.connection.username**
 - Το username για τη σύνδεση στη ΒΔ
- **hibernate.connection.password**
 - Το password για τη σύνδεση στη ΒΔ



Hibernate.hbm2ddl.auto

Hibernate

- Η ιδιότητα **hibernate.hbm2ddl.auto** αυτόματα παράγει τα DDL Statements και επιβεβαιώνει (validates) το DB Schema με βάση τα Entity mappings
- Μπορεί να λάβει τις ακόλουθες τιμές:
 - **validate**: Επιβεβαιώνει (validates) το DB Schema, δεν κάνει αλλαγές στην ΒΔ
 - **update**: κάνει update το DB schema (Για παραγωγικές εφαρμογές)
 - **create**: Δημιουργεί (creates) το DB Schema διαγράφοντας το προηγούμενο σχήμα και τα δεδομένα (Για testing)
 - **create-drop**: Κάνει drop το DB schema όταν γίνεται close το EntityManagerFactory (ρητά) ή όταν κλείνει η εφαρμογή) (Για testing)
- Αν δεν κάνουμε set το **hibernate.hbm2ddl.auto**, το default είναι το validate. Σε περιβάλλον test μπορούμε να έχουμε create ή create-drop ή update, ενώ σε περιβάλλον παραγωγής αναλόγως είτε update ή validate



Teacher Entity (1)

```
5  @Entity
6  @Table (name = "teachers")
7  public class Teacher {
8
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     private Long id;
12
13     @Column(name = "firstname", length = 255, unique =false, nullable = true)
14     private String firstname;
15
16     @Column(name = "lastname", length = 255, unique =false, nullable = true)
17     private String lastname;
18 }
19
```

- Τα Entities (@Entity) είναι Domain Model classes που αντιστοιχούν σε ένα πίνακα στη ΒΔ. Μπορούμε να ορίσουμε το όνομα του πίνακα με @Table
- Επομένως, μέρος του mapping από το Domain Model στο Database Schema είναι και η **αντιστοίχιση των ονομάτων των Entities σε ονόματα πινάκων και πεδίων των πινάκων**. Τα πεδία αντιστοιχούνται με @Column. Τα properties που φαίνονται στο παραπάνω παράδειγμα είναι τα default και μπορούν να παραληφθούν. Αν κάτι δεν είναι default, π.χ. ένα πεδίο είναι unique, τότε θα πρέπει να εισαχθεί ρητά



Teacher Entity (2)

Hibernate

```
5  @Entity
6  @Table (name = "teachers")
7  public class Teacher {
8
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     private Long id;
12
13     @Column(name = "firstname", length = 255, unique = false, nullable = true)
14     private String firstname;
15
16     @Column(name = "lastname", length = 255, unique = false, nullable = true)
17     private String lastname;
18 }
19
```

- Το **@Id annotation** ορίζει τον Entity Identifier. Ένα Entity πρέπει να ορίζει έναν identifier που είναι μοναδικός και μοναδικοποιεί ένα instance στο PersistenceContext και αντιστοιχεί στο primary key του αντίστοιχου πίνακα στη ΒΔ
- Η αρίθμηση του Id μπορεί να γίνεται explicitly από τη ΒΔ (GenerationType.IDENTITY) ή αυτόματα από το ORM (GenerationType.AUTO). Το AUTO δημιουργεί προβλήματα στην αρίθμηση γιατί γίνεται από το Hibernate με bulk τρόπο (από το 1 μπορεί να πάει στο 31)



Generated Ids

- Τα Generated Ids (γνωστά και ως surrogate ids) είναι sequential IDs που δημιουργούνται από το JPA Implementation και εκχωρούνται στα instances
- Το πλεονέκτημα των surrogate ids είναι ότι εγγυώνται ότι είναι μοναδικά ενώ επίσης όλα τα άλλα πεδία μπορούν να γίνουν update εύκολα (ενώ αν ορίζαμε ως Id το natural primary key, αν θέλαμε να αλλάξουμε το Id ενός entity θα έπρεπε να αλλάξει και το ξένο κλειδί όλων των entities και tables που σχετίζονταν με το Entity). Επομένως, τα Natural Ids δεν είναι τόσο ευέλικτα και τόσο efficient. Γιαυτό χρησιμοποιούμε surrogate ids
- Μπορούμε να ορίσουμε την 'στρατηγική' δημιουργίας surrogate ids ως παράμετρο **strategy** του **@GeneratedValue** με τιμές *IDENTITY*, *SEQUENCE*, *TABLE* και *AUTO*



@GeneratedValue (1)

Hibernate

javax.persistence

Enum GenerationType

java.lang.Object

- ↳ java.lang.Enum<[GenerationType](#)>
 - ↳ javax.persistence.GenerationType

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable<[GenerationType](#)>

public enum GenerationType
extends java.lang.Enum<[GenerationType](#)>

Defines the types of primary key generation strategies.

Since:
Java Persistence 1.0

See Also:
[GeneratedValue](#)

Enum Constant Summary	
AUTO	Indicates that the persistence provider should pick an appropriate strategy for the particular database.
IDENTITY	Indicates that the persistence provider must assign primary keys for the entity using a database identity column.
SEQUENCE	Indicates that the persistence provider must assign primary keys for the entity using a database sequence.
TABLE	Indicates that the persistence provider must assign primary keys for the entity using an underlying database table to ensure uniqueness.

- Το default GeneratedValue είναι το AUTO αλλά αυτό δεν τρέχει πάντα σωστά γιατί ελέγχεται από το Hibernate το οποίο κάνει προεργασία για bulk inserts και δεν χρησιμοποιούνται σωστά τα auto increment ids



@GeneratedValue (2)

Hibernate

```
1 package gr.aueb.cf.model;  
2  
3 import jakarta.persistence.*;  
4  
5 @Entity  
6 @Table (name = "teachers")  
7 public class Teacher {  
8  
9     @Id  
10    @GeneratedValue(strategy = GenerationType.IDENTITY)  
11    private Long id;
```

- Μπορούμε να χρησιμοποιήσουμε το IDENTITY strategy της MySQL και να βελτιστοποιήσουμε τη διαδικασία στον MySQL Server
- Ως Id χρησιμοποιούμε συνήθως Long που είναι wrapper κλάση της Java και αντιστοιχεί σε bigint στην MySQL



Persistence class

Hibernate

- Εισάγουμε default και υπερφορτωμένο Constructor καθώς και getters και setters, κανονικά αφού πρόκειται για Java Beans

```
5  @Entity
6  @Table (name = "teachers")
7  public class Teacher {
8
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     private Long id;
12
13     @Column(name = "firstname", length = 255, unique =false, nullable = true)
14     private String firstname;
15
16     @Column(name = "lastname", length = 255, unique =false, nullable = true)
17     private String lastname;
18
19     public Teacher() {}
20
21     public Teacher(Long id, String firstname, String lastname) {...}
22
23     public Long getId() { return id; }
24
25     public void setId(Long id) { this.id = id; }
26
27     public String getFirstname() { return firstname; }
28
29     public void setFirstname(String firstname) { this.firstname = firstname; }
30
31     public String getLastname() { return lastname; }
32
33     public void setLastname(String lastname) { this.lastname = lastname; }
34
35 }
```



Course Entity

Hibernate

- Με τον ίδιο τρόπο ορίζουμε το Course entity

```
1 package gr.aueb.cf.model;  
2  
3 import jakarta.persistence.*;  
4  
5 @Entity  
6 @Table(name = "courses")  
7 public class Course {  
8  
9     @Id  
10    @GeneratedValue(strategy = GenerationType.IDENTITY)  
11    private Long id;  
12    private String title;  
13  
14    public Course() {}  
15  
16    public Course(Long id, String title) {  
17        this.id = id;  
18        this.title = title;  
19    }  
20  
21    public Long getId() { return id; }  
24    public void setId(Long id) { this.id = id; }  
27    public String getTitle() { return title; }  
30    public void setTitle(String title) { this.title = title; }  
33 }
```



Σχέσεις μεταξύ κλάσεων και πινάκων (Associations)

Hibernate

- Οι Entities έχουν μεταξύ τους σχέσεις που μπορεί να είναι
 - OneToOne
 - OneToMany
 - ManyToOne
 - ManyToMany
- Οι σχέσεις αυτές αντιστοιχούνται στη Βάση Δεδομένων με βάση κάποιους κανόνες του Hibernate
- Επίσης, η εφαρμογή μας θα πρέπει να εγγυάται το consistency των σχέσεων του domain model. **Δεν αρκεί δηλαδή να δηλώσουμε τις σχέσεις πρέπει και να συντηρούμε την ακεραιότητα των σχέσεων** παρέχοντας μεθόδους που να κάνουν κάτι τέτοιο για κάθε πεδίο μίας σχέσης



Teachers – Courses @OneToMany

Hibernate

```
8      @Entity
9      @Table (name = "teachers")
10     public class Teacher {
11
12         @Id
13         @GeneratedValue(strategy = GenerationType.IDENTITY)
14         private Long id;
15
16         private String firstname;
17         private String lastname;
18
19         @OneToMany(mappedBy = "teacher")
20         private Set<Course> courses = new HashSet<>();
21     }
```

- Η σχέση Teacher με Courses είναι 1:N και απεικονίζεται στην πλευρά του Teacher ως ένα Collection Set<Course> (Τα Set έχουν καλύτερους χρόνους αναζήτησης όταν υλοποιούνται ως HashSet)
- Εφόσον ο courses είναι ο πίνακας με το ξένο κλειδί, το ξένο κλειδί θα βρίσκεται στον Courses και εδώ απλά θα γίνεται mappedBy το αντίστοιχο πεδίο στον Courses



Overloaded Constructor

Hibernate

```
25 public Teacher(Long id, String firstname, String lastname, Set<Course> courses) {  
26     this.id = id;  
27     this.firstname = firstname;  
28     this.lastname = lastname;  
29     this.courses = courses;  
30 }
```

- Ο overloaded constructor αλλάζει ώστε να περιλαμβάνεται και το courses



Courses Getters & Setters

Hibernate

```
50     protected Set<Course> getCourses() {  
51         return courses;  
52     }  
53  
54     public Set<Course> getAllCourses() {  
55         return Collections.unmodifiableSet(courses);  
56     }  
57  
58     public void setCourses(Set<Course> courses) {  
59         this.courses = courses;  
60     }  
61 }
```

- Την `getCourses()` την κάνουμε `protected` για να μην επιστρέφουμε references και αντ' αυτού παρέχουμε μία `public` μέθοδο `getAllCourses` που επιστρέφει `unmodifiable Set`



Course Entity @ManyToOne

Hibernate

```
5  @Entity
6  @Table(name = "courses")
7  public class Course {
8
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     private Long id;
12     private String title;
13
14     @ManyToOne(fetch = FetchType.LAZY)
15     @JoinColumn(name = "teacher_id")
16     private Teacher teacher;
17 }
```

- Η σχέση σε αυτή την πλευρά είναι @ManyToOne. Έχουμε το πεδίο Teacher που είναι @JoinColumn (ξένο κλειδί). Δίνουμε το name ως **teacher_id** όπως θα είναι το όνομα του πεδίου στον πίνακα της ΒΔ. Το default fetch για Collections είναι LAZY, ωστόσο το τονίζουμε στον κώδικα



Overloaded Constructor

Hibernate

```
20 public Course(Long id, String title, Teacher teacher) {  
21     this.id = id;  
22     this.title = title;  
23     this.teacher = teacher;  
24 }
```

- Ο overloaded constructor αλλάζει και προσθέτουμε το teacher



Getters & Setters

Hibernate

```
38      public Teacher getTeacher() {  
39          return teacher;  
40      }  
41  
42      public void setTeacher(Teacher teacher) {  
43          this.teacher = teacher;  
44      }  
45  }
```



Convenient methods

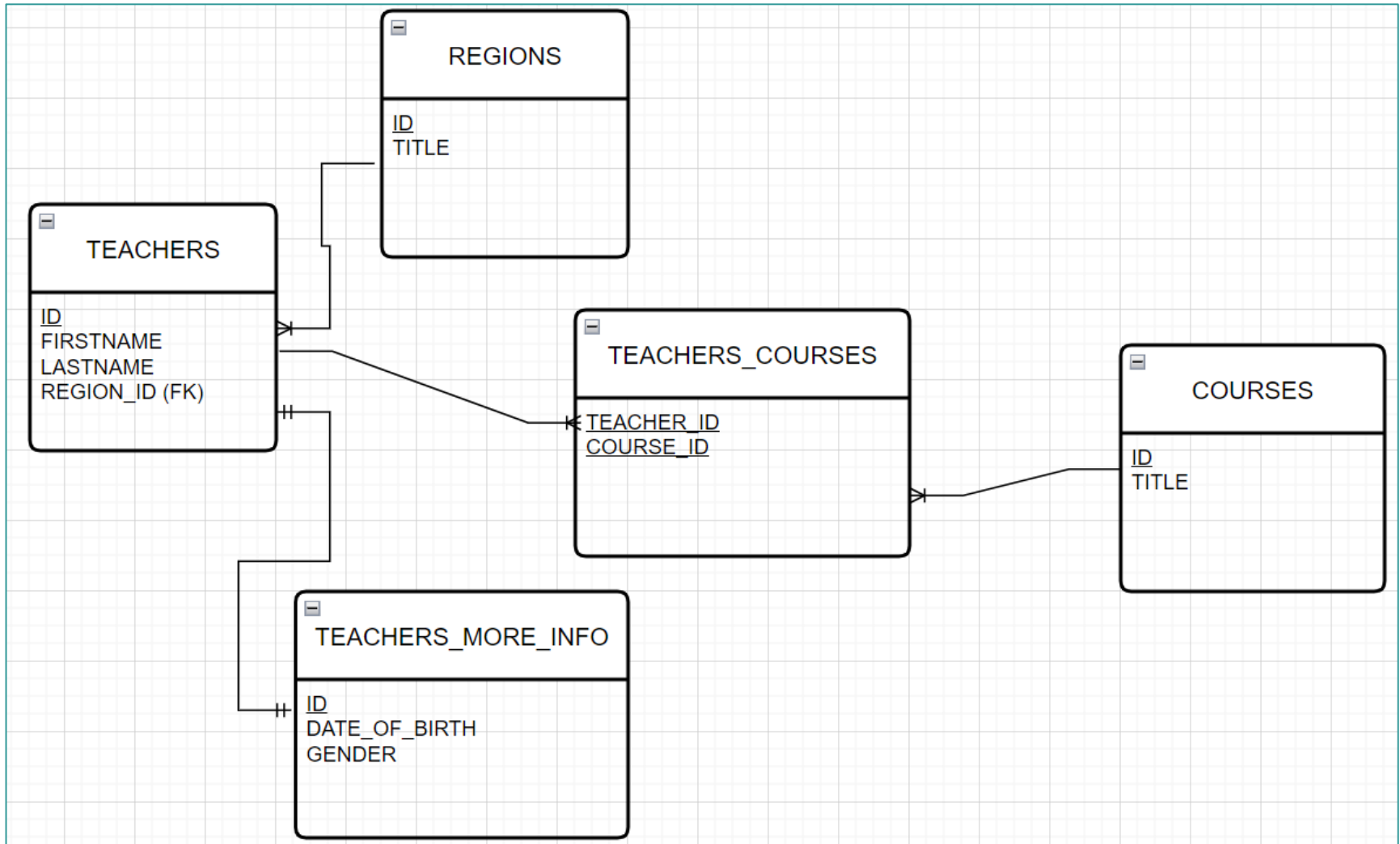
- Τα models θα πρέπει να παρέχουν μεθόδους για να κάνουμε add/remove στοιχεία σε collections. Θα πρέπει ταυτόχρονα να συντηρούνται και οι δύο πλευρές της σχέσης

```
62      public void addCourse(Course course) {  
63          if (courses == null) courses = new HashSet<>();  
64          courses.add(course);  
65          course.setTeacher(this);  
66      }  
67  
68      public void removeCourse(Course course) {  
69          courses.remove(course);  
70          course.setTeacher(null);  
71      }
```



Επόμενο Παράδειγμα

Hibernate





Néo Project

Hibernate

New Project

Empty Project

Generators

Maven Archetype

Jakarta EE

Spring Initializr

JavaFX

Quarkus

Micronaut

Ktor

Compose for Desktop

HTML

React

Express

Angular CLI

Vue.js

Vite

To create a general Maven project, go to the [New Project](#) page.

Name:

Location:

Project will be created in: ~\IdeaProjects\hiber6-dev2

☐ Create Git repository

JDK:

Catalog: [Manage catalogs...](#)

Archetype:

Version:

Additional Properties

<div><div></div><div></div></div>	
junitVersion	5.11.0
javaCompilerVersion	17

Advanced Settings

GroupId:



Teacher

Hibernate

```
13 @Id
14 @GeneratedValue(strategy = GenerationType.IDENTITY)
15 private Long id;
16
17 private String firstname;
18 private String lastname;
19
20 @ManyToMany(mappedBy = "teachers")
21 private Set<Course> courses = new HashSet<>();
22
23 @ManyToOne(fetch = FetchType.LAZY)
24 @JoinColumn(name = "region_id")
25 private Region region;
26
27 @OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)
28 @JoinColumn(name = "teacher_more_info_id")
29 private TeacherMoreInfo teacherMoreInfo;
```

Το **CascadeType.ALL** αφορά το γεγονός ότι το teacherMoreInfo δεν μπορεί να υπάρξει χωρίς τον Teacher. Αν ο teacher γίνει persist, merge, remove, κλπ θα πρέπει και η teacherMoreInfo να γίνει persist, merge, remove, κλπ. Επίσης, αν ο teacher γίνει null, η teacherMoreInfo θα πρέπει να γίνει delete (orphanRemoval, τυπικά να γίνει delete στη ΒΔ)

- Οι σχέσεις πολλά-προς-πολλά μπορούν να θεωρήσουν τη μία πλευρά ως την προς πολλά (ισχυρή πλευρά) και την άλλη ως να ήταν προς ένα
- Ο Teacher έστω ότι είναι η προς-ένα οπότε και κάνει mappedBy
- Η σχέση ένα-προς-ένα πάλι θεωρεί τη μία πλευρά ως να ήταν προς πολλά και την άλλη ως να ήταν προς-ένα.
- Ο Teacher έστω ότι είναι προς πολλά μιας και είναι η ισχυρή οντότητα



Constructors

```
31 public Teacher() {}  
32  
33 public Teacher(Long id, String firstname, String lastname,  
34                 Set<Course> courses, Region region, TeacherMoreInfo teacherMoreInfo) {  
35     this.id = id;  
36     this.firstname = firstname;  
37     this.lastname = lastname;  
38     this.courses = courses;  
39     this.region = region;  
40     this.teacherMoreInfo = teacherMoreInfo;  
41 }
```



Getters, Setters, Convenient Methods in Teacher

Hibernate

```
58     protected Set<Course> getCourses() { return courses; }
61
62     public Set<Course> getAllCourses() {
63         return Collections.unmodifiableSet(courses);
64     }
65
66     public void setCourses(Set<Course> courses) { this.courses = courses; }
69
70     public Region getRegion() { return region; }
73
74     public void setRegion(Region region) { this.region = region; }
77
78     public void addCourse(Course course) {
79         if (courses == null) courses = new HashSet<>();
80         courses.add(course);
81         course.getTeachers().add(this);
82     }
83
84     public void removeCourse(Course course) {
85         courses.remove(course);
86         course.getTeachers().remove(this);
87     }
88 }
```



@ManyToMany in Course

Hibernate

```
10  @Entity
11  @Table(name = "courses")
12  public class Course {
13
14      @Id
15      @GeneratedValue(strategy = GenerationType.IDENTITY)
16      private Long id;
17      private String title;
18
19      @ManyToMany
20      @JoinTable(name = "courses_teachers",
21                joinColumns = @JoinColumn(name = "course_id"),
22                inverseJoinColumns = @JoinColumn(name = "teacher_id"))
23      private Set<Teacher> teachers = new HashSet<>();
24  }
```

- Η μία πλευρά, εδώ η Course, ορίζει το 'config' του ενδιάμεσου πίνακα, όπως name, joinColumns, inverseJoinColumns



Constructors

Hibernate

```
25     public Course() {}
26
27     public Course(Long id, String title, Set<Teacher> teachers) {
28         this.id = id;
29         this.title = title;
30         this.teachers = teachers;
31     }
```



Getters / Setters in Course

Hibernate

```
45     protected Set<Teacher> getTeachers() {  
46         return teachers;  
47     }  
48  
49     public Set<Teacher> getAllTeachers() {  
50         return Collections.unmodifiableSet(teachers);  
51     }  
52  
53     public void setTeachers(Set<Teacher> teachers) {  
54         this.teachers = teachers;  
55     }  
56  
57     public void addTeacher(Teacher teacher) {  
58         if (teachers == null) teachers = new HashSet<>();  
59         teachers.add(teacher);  
60         teacher.getCourses().add(this);  
61     }  
62  
63     public void removeTeacher(Teacher teacher) {  
64         teachers.remove(teacher);  
65         teacher.getCourses().remove(this);  
66     }  
67 }
```



Region

```
1 package gr.aueb.cf.model;
2
3 import ...
4
5
6
7
8
9 @Entity
10 @Table(name = "regions")
11 public class Region {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16     private String title;
17
18     @OneToMany(mappedBy = "region", fetch = FetchType.LAZY)
19     private Set<Teacher> teachers;
20
21     public Region() {}
22
23     public Region(Long id, String title, Set<Teacher> teachers) {
24         this.id = id;
25         this.title = title;
26         this.teachers = teachers;
27     }
28 }
```

- Το Region entity ορίζει μία σχέση @OneToMany με τους teachers
- Κάνει mappedBy το region από τον Teacher
- Στον Teacher υπάρχει το @JoinColumn



Enum

hiber6-dev2 C:\Users\asana\IdeaProjects\hiber6-de

- > .idea
- > .mvn
- src
 - main
 - java
 - gr
 - aueb
 - cf
 - core
 - enums

```
1 package gr.aueb.cf.core.enums;  
2   
3 public enum GenderType {  
4     MALE, FEMALE  
5 }
```

E GenderType

- Ορίζουμε ένα GenderType enum που θα χρησιμοποιήσουμε στο TeacherMoreInfo
- Το MALE, FEMALE είναι instances ενώ ως string τα παίρνουμε με .name() (π.χ. GenderType.MALE.name())



TeacherMoreInfo

Hibernate

```
1 package gr.aueb.cf.model;  
2  
3 import ...  
4  
5  
6  
7  
8 @Entity  
9 @Table(name = "teacher_more_info")  
10 public class TeacherMoreInfo {  
11  
12     @Id  
13     @GeneratedValue(strategy = GenerationType.IDENTITY)  
14     private Long id;  
15  
16     private LocalDateTime dateOfBirth;  
17  
18     @Enumerated(EnumType.STRING)  
19     private GenderType genderType;  
20  
21     @OneToOne(mappedBy = "teacherMoreInfo")  
22     private Teacher teacher;  
23 }
```

- Με Enumerated (EnumType.STRING) ορίζουμε το GenderType να αποθηκεύεται ως String
- Η σχέση @OneToOne είναι η bidirectional ένα-προς-ένα σχέση με τον Teacher.
- Δεδομένου ότι η TeacherMoreInfo είναι εξαρτώμενη πλήρως θα μπορούσε εδώ να μην ορίζεται @OneToOne και η σχέση να είναι απλά one-directional από τον Teacher προς εδώ



Constructors

```
24 public TeacherMoreInfo() {}
25
26 public TeacherMoreInfo(Long id, LocalDateTime dateOfBirth, GenderType genderType,
27     Teacher teacher) {
28     this.id = id;
29     this.dateOfBirth = dateOfBirth;
30     this.genderType = genderType;
31     this.teacher = teacher;
32 }
```



1ο Παράδειγμα CRUD

Hibernate

- Να δούμε το API που μας δίνει το JPA/Hibernate για CRUD πράξεις
- Θα θεωρήσουμε το 1^ο παράδειγμα με Teachers και Courses



CRUD με JPA/Hibernate

Hibernate

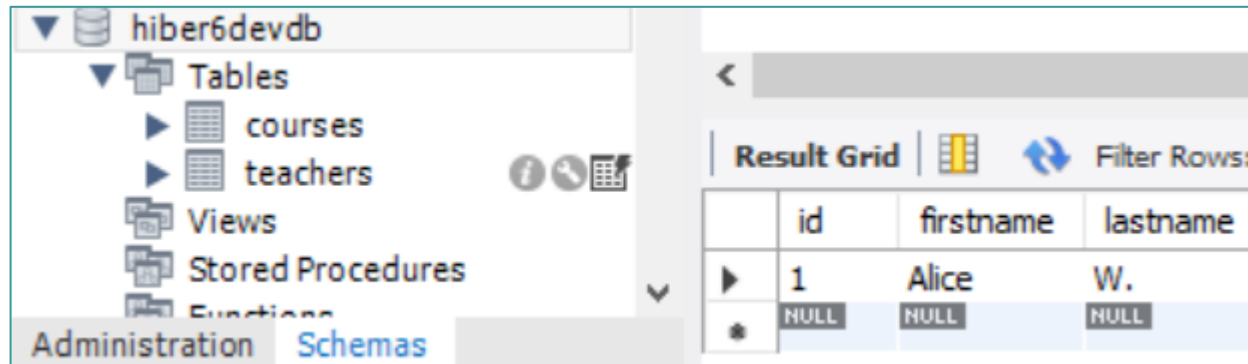
```
3 import gr.aueb.cf.model.Course;
4 import gr.aueb.cf.model.Teacher;
5 import jakarta.persistence.EntityManager;
6 import jakarta.persistence.EntityManagerFactory;
7 import jakarta.persistence.Persistence;
8
9 public class App {
10     public static void main(String[] args) {
11         EntityManagerFactory emf = Persistence.createEntityManagerFactory("school1PU");
12         EntityManager em = emf.createEntityManager();
13
14         Teacher alice = new Teacher(null, "Alice", "W.", null);
15         Course java = new Course(null, "Java", null);
16         alice.addCourse(java);
17
18         em.getTransaction().begin();
19
20         em.persist(alice);
21         em.persist(java);
22
23         em.getTransaction().commit();
24
25         em.close();
26         emf.close();
27     }
28 }
```

- Πρώτα κάνουμε register το PU (school1PU) στον emf, μετά δημιουργούμε τον EntityManager, που είναι ο βασικός τύπος που διαχειρίζεται το persistence-context
- Όλα τα CRUD τρέχουν μέσα σε transactions (getTransaction.begin() – getTransaction.commit()) με τον Entity Manager

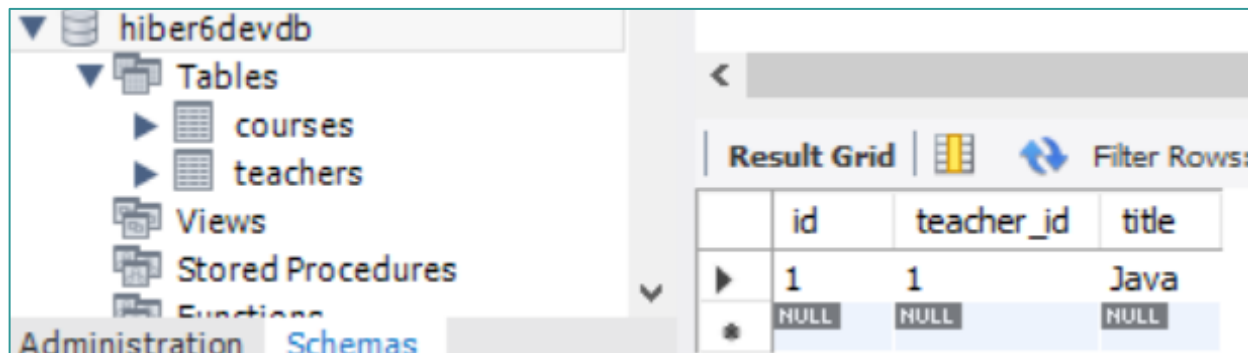


MySQL DB

Hibernate



	id	firstname	lastname
▶	1	Alice	W.
★	NULL	NULL	NULL



	id	teacher_id	title
▶	1	1	Java
★	NULL	NULL	NULL

- Έχουν τρέξει τα persist και έχουν γίνει τα αντίστοιχα insert



Merge

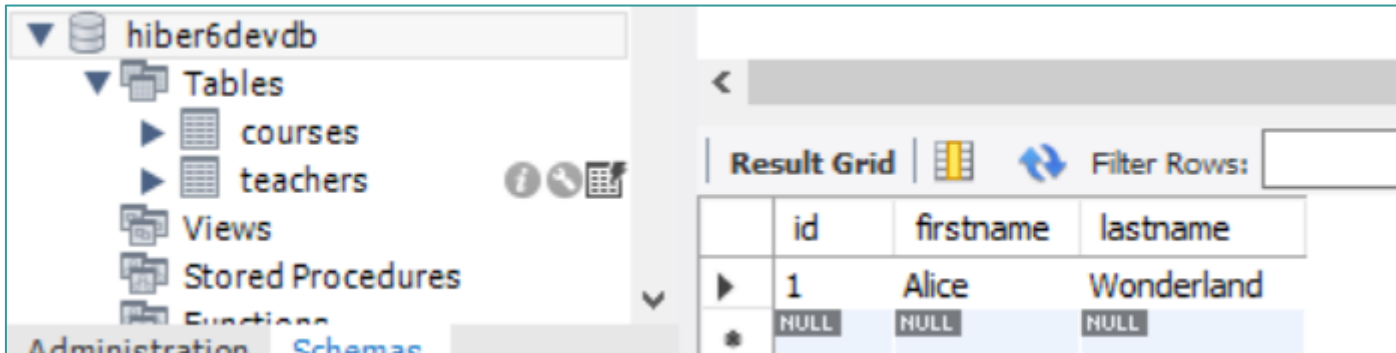
```
18 Teacher alice = em.find(Teacher.class, 1L);
19 alice.setLastname("Wonderland");
20 Course databases = new Course(null, "Databases");
21 alice.addCourse(databases);
22
23 em.getTransaction().begin();
24
25 em.persist(databases);
26 em.merge(alice);
```

- Αλλάζουμε τα στοιχεία της Alice, lastname και courses (αφού δημιουργήσαμε και νέο course και το προσθέσαμε στην alice)

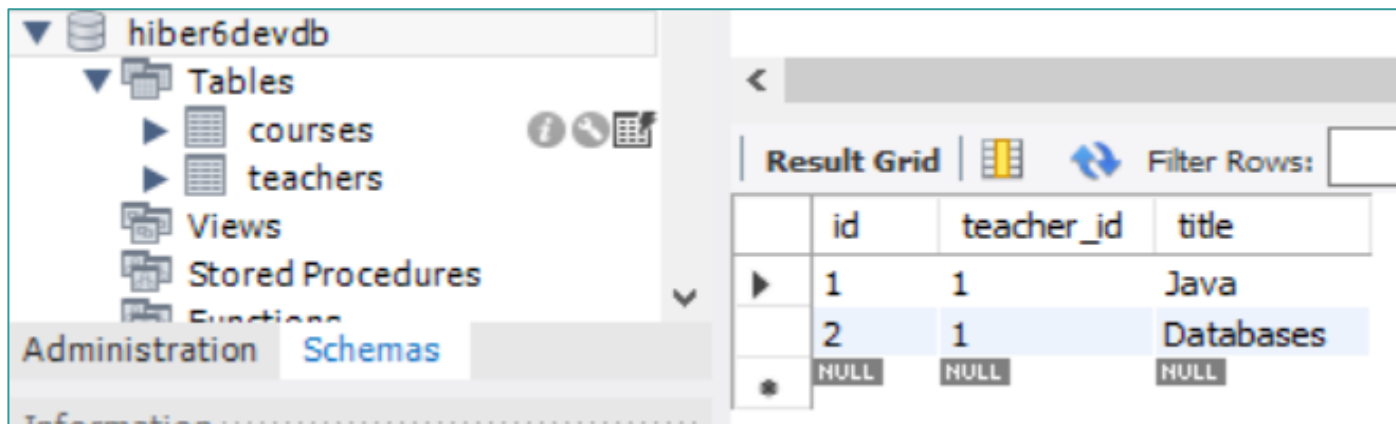


MySQL view

Hibernate



	id	firstname	lastname
▶	1	Alice	Wonderland
*	NULL	NULL	NULL



	id	teacher_id	title
▶	1	1	Java
	2	1	Databases
*	NULL	NULL	NULL



remove

```
23      Course course = em.find(Course.class, 2L);  
24  
25      em.getTransaction().begin();  
26  
27      em.remove(course);
```

- Όλες αυτές οι εντολές εκτελούνται στο Persistence Context και μόνο όταν γίνει commit εκτελούνται στη ΒΔ



MySQL μετά το remove

Hibernate

The screenshot shows the MySQL Workbench interface. On the left, the 'hiber6devdb' database is expanded, showing a tree view with 'Tables' (containing 'courses' and 'teachers'), 'Views', 'Stored Procedures', and 'Functions'. The 'Administration' tab is selected, and the 'Schemas' sub-tab is active. On the right, the 'Result Grid' is displayed, showing the results of a query. The grid has four columns: 'id', 'teacher_id', and 'title'. The first row shows '1', '1', and 'Java'. The second row shows 'NULL', 'NULL', and 'NULL'.

	id	teacher_id	title
▶	1	1	Java
★	NULL	NULL	NULL



Queries

- Εκτός από insert, update, delete και απλό select με το id (persist, merge, remove, find) μας δίνεται η δυνατότητα από το JPA / Hibernate να κάνουμε queries
- Παρέχονται δύο βασικοί μηχανισμοί οι **JPQL** και το **Criteria API**
 - **JPQL** (Java Persistence Query Language). Πρόκειται για παρόμοια με την SQL γλώσσα που εκτελείται στο Domain Model
 - **Criteria Queries**. Πρόκειται για προγραμματιστικό τρόπο έκφρασης Queries με Java interfaces και κλάσεις
 - **Native Queries**. Παρέχονται επίσης και κανονικά SQL Queries που εκτελούνται στο Domain Model



- Η **JPQL (Java Persistence Query Language)** ορίζεται στο JPA Specification
- Πρόκειται για μια γλώσσα παρόμοια με την SQL με τη διαφορά ότι **εφαρμόζεται στο Domain Model και στα Entities** και όχι στο Database Schema
- Έχει το πλεονέκτημα ότι είναι παρόμοια με την SQL, αλλά και το μειονέκτημα ότι παρέχει μηχανισμούς για static queries, γνωστά @compile time
- Η JPQL δεν είναι Type-Safe ενώ όπως αναφέραμε έχει **περιορισμούς στην κατασκευή δυναμικών queries τα οποία αλλάζουν structurally**



toString()

Hibernate

```
73      @Override
74      public String toString() {
75          return "Teacher{" +
76              "id=" + id +
77              ", firstname='" + firstname + '\'' +
78              ", lastname='" + lastname + '\'' +
79              '}';
80      }
81  }
```

```
46      @Override
47      public String toString() {
48          return "Course{" +
49              "id=" + id +
50              ", title='" + title + '\'' +
51              '}';
52      }
53  }
```



Select (1)

- Έστω ένα **Entity Teacher**. Έστω το παρακάτω query στην JPQL: **SELECT t FROM Teacher s**
- Όπως και στην SQL έτσι και στην JPQL το **FROM** ορίζει από ποια Entities θα επιλεγούν τα δεδομένα (πεδία των Entities)
- Το **Teacher** είναι ένα entity reference και μάλιστα **Root Entity Reference** μιας και από εδώ ξεκινά το scope (η εμβέλεια) του Query.
- Το **t** είναι ένα ψευδώνυμο (alias) που επέχει θέση *identification variable*, δηλαδή το **Teacher t** είναι δήλωση μίας μεταβλητής **t**, που αναφέρεται σε όλα τα instances της Entity και για αυτό ονομάζεται και **range variable**



Select (2)

- Επομένως το **SELECT t FROM teacher t** θα επιστρέψει όλα τα Teacher instances (σε ένα List<Teacher>) και στη συνέχεια θα μπορούμε να επεξεργαστούμε το List
- Στο SELECT μπορούμε να επιλέξουμε είτε το instance, ή πεδία του instance



Queries

- Στην JPA ένα query αναπαρίσταται από το **`jakarta.persistence.Query`** ή **`jakarta.persistence.TypedQuery`**
- Λαμβάνουμε ένα *Query* ή *TypedQuery* με την εκτέλεση από τον **`EntityManager`** της μεθόδου **`createQuery`** (ή **`createNamedQuery`**, αν έχουμε δημιουργήσει query με συγκεκριμένο όνομα)
- Ο τύπος ***Query*** δεν είναι compile type-safe γιατί η Java δεν γνωρίζει τον επιστρεφόμενο τύπο
- Ο τύπος ***TypedQuery*** είναι compile type-safe και παρέχεται μία **overloaded μέθοδος `createQuery`** που επιστρέφει *TypedQuery* και περιλαμβάνει στις παραμέτρους της και την κλάση που επιστρέφεται, οπότε ο τύπος επιστροφής μπορεί να ελεγχθεί



createQuery

Hibernate

- Η **createQuery** δημιουργεί ένα JPQL query και έχει δύο βασικές μορφές:
 - Με μία παράμετρο String –το String είναι η εντολή Select- **που επιστρέφει Query object**, και δεν είναι **type safe** γιατί όταν εκτελείται επιστρέφει Object και
 - Με δύο παραμέτρους, String και Class<T> **που επιστρέφει TypedQuery** όπου το String είναι το Select string και το Class<T> είναι το επιστρεφόμενο Entity τύπου T, και επομένως είναι type-safe

createQuery

```
Query createQuery(String qlString)
```

Create an instance of Query for executing a Java P

Parameters:

qlString - a Java Persistence query string

Returns:

the new query instance

Throws:

IllegalArgumentException - if the query str

createQuery

```
<T> TypedQuery<T> createQuery(String qlString,  
                                Class<T> resultClass)
```

Create an instance of TypedQuery for executing a Java Persis
resultClass argument.

Parameters:

qlString - a Java Persistence query string

resultClass - the type of the query result

Returns:

the new query instance

Throws:

IllegalArgumentException - if the query string is fo

Since:

Java Persistence 2.0



Path Expressions (1)

Hibernate

- Στο SELECT συνήθως έχουμε Entity range variables αλλά μπορούμε να έχουμε και πεδία των Entities (Paths ή Path expressions)
- Τα *path expressions* πάντα ξεκινάνε από ένα instance ενός user defined class, π.χ. το Root Entity variable και χρησιμοποιούν την τελεία για την 'πλοήγηση' (navigation) μέσω των πεδίων των entities σε άλλα objects και τιμές
- Για παράδειγμα στο: `"SELECT s.name FROM Student s "`
- το **s.name** είναι ένα **path**, όπου το **s** αναπαριστά ένα Student entity object και το **name** είναι ένα persistent field του Student



Path Expressions (2)

Hibernate

- Τα path expressions μπορούν να γίνουν extend και να κάνουμε navigate και σε συνδεδεμένες entities του domain model
- Αν για παράδειγμα ο Student είχε ένα πεδίο *teacher* μέσω του οποίου συνδεόταν με ένα Teacher entity και ο Teacher είχε ένα πεδίο *lastname*, τότε θα μπορούσαμε να έχουμε το παρακάτω TypedQuery:

```
("SELECT  s.teacher.lastname FROM Student s
WHERE s.teacher.lastname = :lastname", Student.class)
.setParameter("lastname", lastname).getResultList();
```

- Αυτό είναι ένα implicit join



WHERE - parameters

Hibernate

```
99 TypedQuery<Student> studentQuery = em.createQuery(  
100     "SELECT s " +  
101     "FROM Student s " +  
102     "WHERE s.name = :name", Student.class);  
103  
104 studentQuery.setParameter("name", "alice");  
105  
106 Student student = studentQuery.getSingleResult();  
107 System.out.println(student.getName());
```

```
99 TypedQuery<Student> studentQuery = em.createQuery(  
100     "SELECT s " +  
101     "FROM Student s " +  
102     "WHERE s.name = ?1", Student.class);  
103  
104 studentQuery.setParameter( 1 , "alice" );  
105  
106 Student student = studentQuery.getSingleResult();  
107 System.out.println(student.getName());
```

- Στο `where` μπορεί να έχουμε `named parameters`
- Το `:name` είναι `placeholder` και πρέπει στη συνέχεια με την `setParameter` να δώσουμε τιμή

- Στο `where` μπορούμε να έχουμε και `positional parameters`
- Το `?1` είναι πάλι `placeholder` και πρέπει στη συνέχεια με την `setParameter` να δώσουμε τιμή



WHERE clause & Paths

Hibernate

- Τα Paths είναι περισσότερο χρήσιμα στα predicates του WHERE clause. Στο WHERE ενός JPQL statement μπορεί να έχουμε συγκρίσεις με διάφορους τελεστές όπως =,<>,LIKE κλπ.
- Οι συγκρίσεις γίνονται ανάμεσα σε **paths** και **τιμές**
- Τα path expressions, όπως είδαμε είναι η διαδρομή από την Root entity μέχρι το πεδίο που θέλουμε να συγκρίνουμε για παράδειγμα s.name
- Ο τελεστής . είναι τελεστής εμβέλειας



JOIN

- Μπορούμε να έχουμε ***Implicit Join*** όταν στο SELECT επιλέγουμε path (s.teacher) όπου το teacher είναι Entity και επομένως δημιουργείται αυτόματα join:
- "SELECT s.teacher FROM Student s"
- Μπορούμε όμως να έχουμε και Explicit Join:
- "SELECT t FROM Student s JOIN s.teacher t"
- Τα αποτελέσματα είναι ίδια



Explicit Join

Hibernate

- Implicit Joins παίρνουμε αν ζητάμε στο `SELECT association` σε *Entity* ή αν το `association` βρίσκεται στο `WHERE`.
- Διαφορετικά, αν το join δεν υπονοείται στο `WHERE`, πρέπει να κάνουμε ρητά Join. Για παράδειγμα:
- `"SELECT t FROM Student s JOIN s.teacher t WHERE s.name LIKE 'A%'"`



Explicit Join με Collection

Hibernate

- Αν θέλουμε να πάρουμε collection και να κάνουμε filter στο WHERE δεν μπορούμε γιατί δεν μπορούμε να έχουμε **path expression** σε **collections** (δεν μπορεί να μπει ο τελεστής τελεία μετά το **collection**), οπότε αναγκαστικά κάνουμε join
 - "SELECT c FROM Student s JOIN s.courses c WHERE c.title LIKE 'Computing%'"



Native Queries

Hibernate

- Υπάρχουν και τα native queries που εκτελούνται μέσω του interface NativeQuery
- Αν είχαμε ένα Entity Student με δύο persistent πεδία, π.χ. firstname και lastname, τότε θα μπορούσαμε να έχουμε ένα native query όπως το παραπάνω

```
111 List<Object[]> students = em.createNativeQuery("SELECT * FROM Student").getResultList();
112 for(Object[] student : students) {
113     System.out.println(student[0]);
114     System.out.println(student[1]);
115 }
```



JPQL Queries (1)

Hibernate

```
em.getTransaction().begin();

// Select all teachers
String jpql = "SELECT t FROM Teacher t";
TypedQuery<Teacher> query = em.createQuery(jpql, Teacher.class);
List<Teacher> teachers = query.getResultList();
teachers.forEach(System.out::println);

// Select all courses
String jpql2 = "SELECT c FROM Course c";
List<Course> courses = em.createQuery(jpql2, Course.class).getResultList();
courses.forEach(System.out::println);

// Select the courses that are taught by Alice
String jpql3 = "SELECT c FROM Course c WHERE c.teacher.firstname = :firstname";
TypedQuery<Course> query3 = em.createQuery(jpql3, Course.class);
query3.setParameter("firstname", "Alice");
List<Course> courses3 = query3.getResultList();

em.getTransaction().commit();
```



JPQL Queries (2)

Hibernate

```
// Select Teachers and course titles they teach
String jpql4 = "SELECT t, c.title FROM Teacher t JOIN t.courses c";
TypedQuery<Object[]> query4 = em.createQuery(jpql4, Object[].class);
List<Object[]> results = query4.getResultList();

for (Object[] result : results) {
    Teacher teacher = (Teacher) result[0];
    String courseTitle = (String) result[1];
    System.out.println("Teacher: " + teacher.getFirstname() + ", Course: " + courseTitle);
}

// Select teachers that teach Java
String jpql5 = "SELECT t FROM Teacher t JOIN t.courses c WHERE c.title = :courseTitle";

TypedQuery<Teacher> query5 = em.createQuery(jpql5, Teacher.class);
query5.setParameter("courseTitle", "Mathematics");
List<Teacher> results5 = query5.getResultList();
results5.forEach(System.out::println);
```



JPQL Queries (3)

```
// Select teacher's firstname and the count of courses they teach
String jpql6 = "SELECT t.firstname, COUNT(c) FROM Teacher t JOIN t.courses c GROUP BY t.firstname";

TypedQuery<Object[]> query6 = em.createQuery(jpql6, Object[].class);
List<Object[]> results6 = query6.getResultList();

for (Object[] result : results6) {
    String firstname = (String) result[0];
    Long courseCount = (Long) result[1];
    System.out.println("Teacher: " + firstname + ", Courses: " + courseCount);
}

// Select teachers who teach more than one course
String jpql7 = "SELECT t FROM Teacher t JOIN t.courses c GROUP BY t HAVING COUNT(c) > 1";

TypedQuery<Teacher> query7 = em.createQuery(jpql7, Teacher.class);
List<Teacher> results7 = query7.getResultList();
results7.forEach(System.out::println);
```



JPQL Queries (4)

Hibernate

```
// Select teachers and the courses they teach ordered by lastname, and course title
String jpql8 = "SELECT t, c FROM Teacher t JOIN t.courses c ORDER BY t.lastname ASC, c.title ASC";

TypedQuery<Object[]> query8 = em.createQuery(jpql8, Object[].class);
List<Object[]> results8 = query8.getResultList();

for (Object[] result : results8) {
    Teacher teacher = (Teacher) result[0];
    Course course = (Course) result[1];
    System.out.println("Teacher: " + teacher.getLastname() + ", Course: " + course.getTitle());
}

// Select Teachers that do not teach any course
String jpql9 = "SELECT t FROM Teacher t LEFT JOIN t.courses c WHERE c IS NULL";

TypedQuery<Teacher> query9 = em.createQuery(jpql9, Teacher.class);
List<Teacher> results9 = query9.getResultList();
results9.forEach(System.out::println);
```



JPQL Queries (5)

Hibernate

```
// Select the most popular courses by teacher's count
String jpql10 = "SELECT c.title, COUNT(t) FROM Course c JOIN c.teacher t GROUP BY c.title ORDER BY COUNT(t) DESC";

TypedQuery<Object[]> query10 = em.createQuery(jpql10, Object[].class);
List<Object[]> results10 = query10.getResultList();

for (Object[] result : results10) {
    String courseTitle = (String) result[0];
    Long teacherCount = (Long) result[1];
    System.out.println("Course: " + courseTitle + ", Teacher Count: " + teacherCount);
}
```

- Όλα αυτά ήταν παραδείγματα χρήσης της JPQL η οποία είναι χρήσιμη σε στατικά όπως είπαμε queries, δηλαδή queries που αλλάζουν @runtime μόνο οι τιμές και όχι η δομή τους
- Για δυναμικά queries όπως εκείνα που χρησιμοποιούνται για filtering, παρέχεται το Criteria API



Παραμετρικοί Πίνακες

Hibernate

- Υπάρχουν δεδομένα που θεωρούνται στατικά, ή αλλιώς και παραμετρικά γιατί αφενός μεν δεν μεταβάλλονται συχνά ενώ επίσης χρησιμοποιούνται σε άλλους πίνακες
- Όπως για παράδειγμα, περιφέρειες, νομοί, δήμοι, χώρες, ΔΟΥ, κλπ.
- Αυτά τα δεδομένα τα εισάγουμε κατευθείαν στη ΒΔ. Ενώ επίσης στη συνέχεια εισάγουμε και indexes όπου χρειάζεται πάλι με SQL στη ΒΔ



- Η εισαγωγή στατικών / παραμετρικών δεδομένων δεν αναιρεί την ανάγκη δημιουργίας διαχειριστικών φορμών από όπου θα μπορεί να γίνεται εισαγωγή, διαγραφή, επεξεργασία των στατικών δεδομένων



Δεδομένα excel

Hibernate

- Έστω ότι θέλουμε να εισάγουμε Regions από όπου προέρχονται οι Teachers και Courses και Καθηγητές
- Συνήθως τα δεδομένα δίνονται από τους φορείς σε μορφή excel

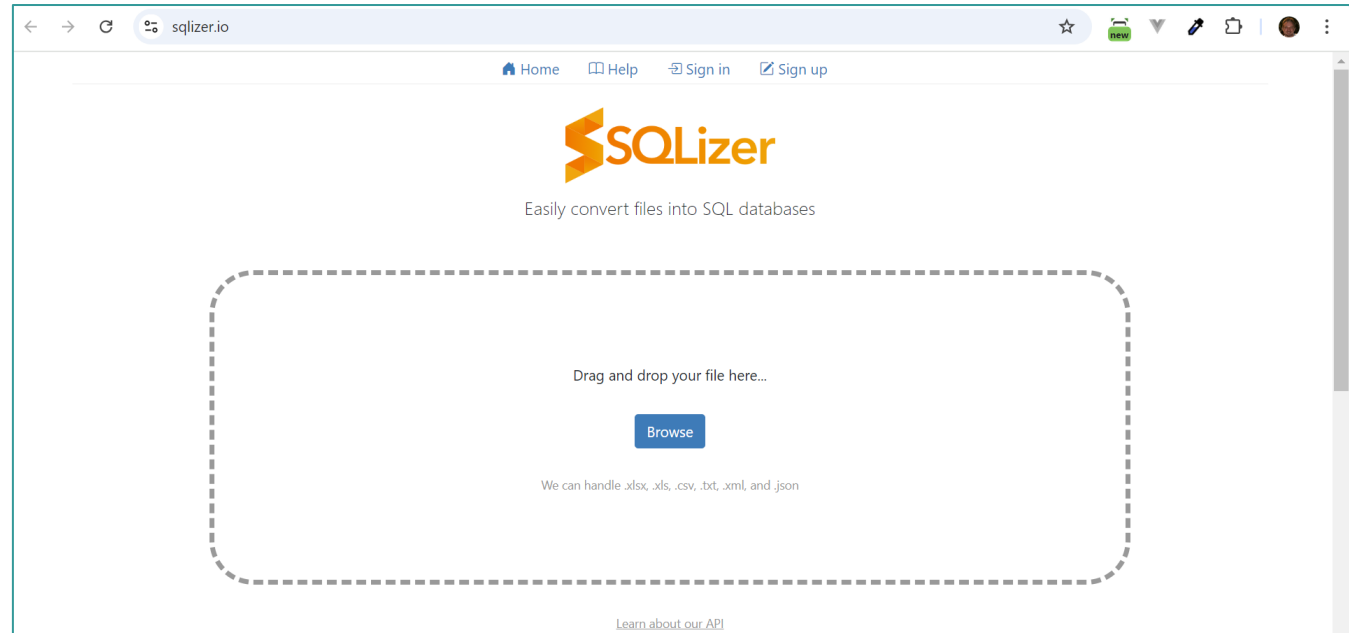


Excel -> SQL Insert

Hibernate

	A	B
1	id	title
2		1 Αθήνα
3		2 Θεσσαλονίκη
4		3 Βόλος
5		4 Τρίπολη
6		5 Καλαμάτα
7		6 Δράμα
8		7 Καβάλα
9		8 Πάτρα
10		9 Λάρισα
11		10 Χανιά
12		
13		
14		
15		
16		
17		
18		
19		
20		

< > regions



- Με online tools όπως ο SQLizer (<https://sqlizer.io/>) μπορούμε να μετατρέψουμε αυτόματα τα δεδομένα από excel σε SQL INSERT εντολές





Μετατροπή δεδομένων (1)


Hibernate


← → ↻ sqlizer.io ☆ 📄 ▼ ✎ 📁 👤 ⋮

File Types


☒


☐


☐


☐

☒ Has Header Row

☒ Active Worksheet

☒ Whole Sheet

Advanced Settings ⬆

Table Name

☒ Check Table Exists

Treat this file as

Convert



Μετατροπή δεδομένων (2)

Hibernate

The screenshot shows the sqlizer.io website interface. On the left, there are five buttons: 'Download regions.sql' (green), 'Copy to Clipboard' (white with green border), 'Change Settings' (white with green border), 'Convert Another File' (white with green border), and 'Tweet about SQLizer!' (orange). On the right, the generated SQL code is displayed in a text area:

```
CREATE TABLE IF NOT EXISTS `regions` (  
  `id` INT,  
  `title` VARCHAR(11) CHARACTER SET utf8  
);  
INSERT INTO `regions` VALUES (1,'Αθήνα'),  
  (2,'Θεσσαλονίκη'),  
  (3,'Βόλος'),  
  (4,'Τρίπολη'),  
  (5,'Καλαμάτα'),  
  (6,'Δράμα'),  
  (7,'Καβάλα'),  
  (8,'Πάτρα'),  
  (9,'Λάρισα'),  
  (10,'Χανιά');
```

- Παίρνουμε μόνο το INSERT copy / paste



Εκτέλεση σε MySQL

Hibernate

```
regions x
1 • use hiber6dev2db;
2
3 • INSERT INTO regions (id, title) VALUES
4     (1, 'Αθήνα'),
5     ('2', 'Θεσσαλονίκη'),
6     ('3', 'Βόλος'),
7     ('4', 'Τρίπολη'),
8     ('5', 'Καλαμάτα'),
9     ('6', 'Δράμα'),
10    ('7', 'Καβάλα'),
11    ('8', 'Πάτρα'),
12    ('9', 'Λάρισα'),
13    ('10', 'Χανιά');
14
15 • ALTER TABLE regions AUTO_INCREMENT = 11;
```

1 • SELECT * FROM hiber6dev2db.regions;

Result Grid | Filter Rows: | Edit:

	id	title
▶	1	Αθήνα
	2	Θεσσαλονίκη
	3	Βόλος
	4	Τρίπολη
	5	Καλαμάτα
	6	Δράμα
	7	Καβάλα
	8	Πάτρα
	9	Λάρισα
	10	Χανιά
•	NULL	NULL

- Εκτελούμε το query (αριστερά) και εισάγονται τα δεδομένα (δεξιά). Παρατηρήστε το AUTO_INCREMENT = 11 όπου θέτουμε σωστά το auto increment ώστε οποιοδήποτε insert γίνει στη συνέχεια να ξεκινήσει από το 11



Courses

Hibernate

id	title
1	Java
2	C#
3	Databases
4	JavaScript
5	UX/UI
6	Git
7	Python
8	Angular
9	React
10	Vue



Easily convert files into SQL databases

```
INSERT INTO `courses` VALUES (1,'Java'),  
  (2,'C#'),  
  (3,'Databases'),  
  (4,'JavaScript'),  
  (5,'UX/UI'),  
  (6,'Git'),  
  (7,'Python'),  
  (8,'Angular'),  
  (9,'React'),  
  (10,'Vue');
```

- Αντίστοιχα μετατρέπουμε τα courses από excel σε SQL Insert



MySQL Courses insert

Hibernate

```
1 • use hiber6dev2db;
2
3 • INSERT INTO `courses` VALUES (1, 'Java'),
4     (2, 'C#'),
5     (3, 'Databases'),
6     (4, 'JavaScript'),
7     (5, 'UX/UI'),
8     (6, 'Git'),
9     (7, 'Python'),
10    (8, 'Angular'),
11    (9, 'React'),
12    (10, 'Vue');
13
14 • ALTER TABLE teachers AUTO_INCREMENT = 11;
```

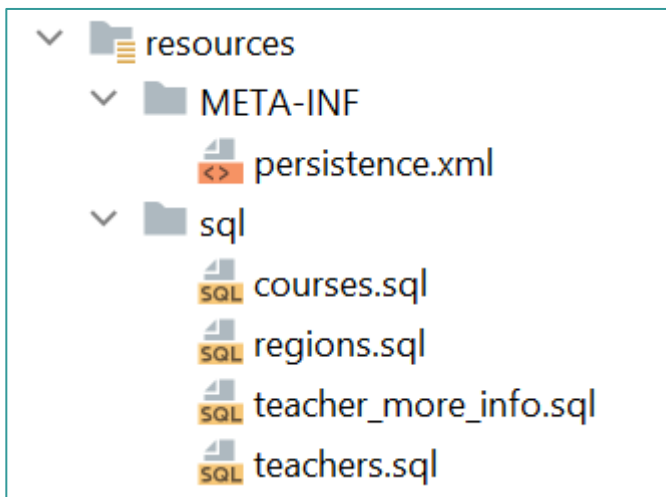
- Εκτελούμε σε περιβάλλον MySQL



Linux MySQL

Hibernate

- Σε περιβάλλον Linux θα συνδεόμασταν στον MySQL Server με: `mysql -u root -p`
- Και θα εκτελούσαμε: `SOURCE /path/to/your/script.sql;`



Αν έχουμε κατεβάσει με Git στο Linux τότε τα SQL Scripts είναι στο φάκελο sql μέσα στο resources



Teachers

Hibernate

```
1 • use hiber6dev2db;
2
3 • INSERT INTO `teachers` VALUES (1,1,NULL,'Αθανάσιος','Ανδρούτσος'),
4     (2,1,NULL,'Μάκης','Καπέτης'),
5     (3,2,NULL,'Άννα','Γιαννούτσου'),
6     (4,3,NULL,'Μάρκος','Καραμπάτσης'),
7     (5,4,NULL,'Παναγιώτης','Μόσχος'),
8     (6,4,NULL,'Σοφοκλής','Στουραϊτης');
9
10 • ALTER TABLE teachers AUTO_INCREMENT = 7;
```

- Μπορούμε να εισάγουμε και teachers γνωρίζοντας και τη συσχέτιση με τα regions και αφήνοντας null τα more info που θα εισάγουμε αργότερα και θα κάνουμε εδώ update



Teacher more info

Hibernate

```
1 • use hiber6dev2db;
2
3 • INSERT INTO `teacher_more_info` VALUES (1,'1991-01-12 09:30:00','MALE'),
4      (2,'2000-11-10 21:10:00','FEMALE');
5
6 • ALTER TABLE `teacher_more_info` AUTO_INCREMENT = 3;
7
```

Result Grid					
Filter Rows: <input type="text"/>					
	id	region_id	teacher_more_info_id	firstname	lastname
	1	1	1	Αθανάσιος	Ανδρούτσος
	2	1	2	Μάκης	Καπέτης
	3	2	NULL	Άννα	Γιαννούτσου

- Εισάγουμε teacher_more_info και κάνουμε και update τον πίνακα των teachers, ώστε να συσχετίζουμε τους δύο πρώτους καθηγητές με τα teacher_more_info



indexes

```
1 • USE hiber6dev2db;
2
3 • CREATE INDEX idx_courses_title ON courses (title);
4 • CREATE INDEX idx_regions_title ON regions (title);
5 • CREATE INDEX idx_more_info_date_of_birth ON teacher_more_info (date_of_birth);
6 • CREATE INDEX idx_more_info_gender ON teacher_more_info (date_of_birth);
7 • CREATE INDEX idx_teachers_firstname ON teachers (firstname);
8 • CREATE INDEX idx_teachers_lastname ON teachers (lastname);
```

- Σε ένα .sql script εισάγουμε τα indexes ώστε αναζητήσεις, join, group by να γίνονται με indexes. Διαφορετικά όλα τα παραπάνω θα γίνονται πολύ αργά



Ασκήσεις

Hibernate

- Τρέξτε τα παρακάτω queries με JPQL στα δεδομένα που έχουμε εισάγει:
 - Get all teachers
 - Get all courses with titles
 - All teachers from a specific region
 - All teachers and region from a specific region (π.χ. Αθήνα)
 - List Teachers with More Info
 - Find All Teachers Who Teach a Specific Course (π.χ. Java)
 - Count the Number of Teachers per Region
 - Find All Female Teachers with Their Courses
 - List Teachers Without Any Courses
 - Find Teachers Who Teach More Than One Course
 - List All Teachers and Their Region, Including Those Without Courses
 - List Courses with Number of Teachers
 - List Courses with Number of Teachers even with 0
 - Teachers who are Female, teach more than one course, and are from a specific region (π.χ. Αθήνα)