**ORM - HIBERNATE**

ORM מציעה למפות את ה-DB שלנו לאובייקטים ולעבוד מולם.

ל-ORM יש מספר רב של ספריות הממשות אותה. הספריות הנפוצות ביותר בימינו בשפת JAVA מאפשרות לנו להמיר את הרשומה בטבלה בDB לאובייקט במספר דרכים.

**דרך מיושנת לעשות זאת,** זה עם קובץ מיפוי מסוג XML, אשר מכיל את כל פרטי האובייקט והקשרים שלו בין הטבלאות לעמודות. הנה דוגמא למיפוי בעזרת XML:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**שימו לב** כי לא נשתמש בשיטה זו, אך חשוב שתכירו אותה כי יש מקומות שעדיין משתמשים בה וגם למקרה שתיתקלו בה במהלך חיפוש באינטרנט אחר בעיות בORM, ותוכלו לקשר בין שיטה זו לשיטה שבה אנו נשתמש.

שיטה נוספת (בה נשתמש) היא מיפוי האובייקט ממש בתוך מחלקת האובייקט וזאת באמצעות שימוש באנוטציות (Annotations).   
אנוטציות אלה באות מתוך API שנקרא JPA - **J**ava **P**ersistence **A**PI.

[יש לקרוא פה על JPA](https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html)

**שימו לב שכל האנוטציות בהמשך הם case sensitive!**

**קישור לפרויקט**על מנת לקשר את הJPA לפרויקט, נקשר עוד תלות לpom.xml שלנו:

[**(וודאו שזה הdependancy העדכני)**](https://mvnrepository.com/)

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

**מימוש Hibernate**  
נקשר את התלות של הJPA. זוהי התלות היחידה שנצטרך לקשר כי באופן דיפולטי המימוש של JPA בSpring משתמש בHibernate.  
כמו שאמרנו, אנחנו רוצים לעבוד בצורה הקלה ביותר מול הDB. גישת הORM מאפשרת לנו את זה כי היא עובדת עם אובייקטים שמדמים לנו טבלה בDB. לכן דבר ראשון שנעשה זה ליצור Model – מחלקה שמייצגת את אותה הטבלה. המחלקה תהיה כשם הטבלה והDMים יהיו כשם השדות.   
כדי לקשר בין הטבלה למחלקה, נוסיף אנוטציות מתאימות.

@Entity

@Table(name="t\_empl")

**public** **class** Employee {

בואו נראה מה כל אנוטציה אומרת:

**@Entity** – מייצגת שהמחלקה היא ישות שאנו ממפים באמצעות JPA. כלומר מעל כל מחלקה שתייצג לנו טבלה נכתוב את האנוטציה הזאת, וכך המחלקה תזוהה על ידי התוכנית כישות שמשויכת לטבלה.

**@Table(name=”table\_name”)** – האנוטצייה הזאת מייצגת איזו טבלה נרצה לקשר למחלקה הזאת כאשר המאפיין name הוא פרמטר שהאנוטציה מקבלת והוא שם הטבלה ב-DB.

@Column(name="FIRSTNAME")

@JsonProperty

**private** String firstname;

**@Column(name=”column\_name”)** – מעל כל מאפיין נגדיר את שם העמודה שאליו המאפיין מקושר. למשל בדוגמה של הקוד שלנו עמודת השם הפרטי שבטבלה תקושר עם המאפיין firstname בתוך המחלקה

**@JsonProperty** – נגדיר אנוטציה זאת כפי שלמדנו בשיעור Spring + REST. אם שכחתם, תוכלו למצוא בשיעור Spring + Rest את ההסבר המורחב על האנוטציה הזאת ותפקידה.

@Id

@Column(name="EMPNO")

@JsonProperty

**private** long id;

**@id** – נרשום אנוטציה זו מעל המאפיין שמייצג את המפתח הראשי בטבלה ב-DB. המטרה היא שנוכל לשלוף רשומות ספציפיות לפי המפתח הראשי שלהם ובצורה יותר נוחה ולמחוק רשומות במידה ונרצה באותו האופן.

**אז מה בעצם ראינו עד עכשיו?**ראינו שכל טבלה בDB תבוטא על ידי מחלקה, ושדות הטבלה יהיו מבוטאים על ידי DMים של המחלקה. בנוסף הבנו שכדי למפות ולשייך כל DM לשדה מתאים בטבלה בDB נצטרך להשתמש באנוטציות המתאימות.

ניתן לכתוב שאילתות בקוד שלנו בSpring בעזרת JPA ללא ספריית ORM חיצונית בעזרת אנוטציה שנקראת Query@

@Query("SELECT u FROM User u WHERE u.status = 1")

**public** Collection<User> findAllActiveUsers();

בדוגמה הזאת כתבנו ישירות שאילתה וקישרתי אותה לפונקציה מסוימת.   
בדרך זאת נוכל לכתוב שאילתות פשוטות ומורכבות. **נדגיש כי אנחנו לא נשתמש בדרך זו, אלא נשתמש בHibernate ובפעולות שהספרייה מייחצנת לנו.**

שכבות ה**Hibernate**

תמונה שמכילה טקסט, שלט

התיאור נוצר באופן אוטומטי**Repository**

ממשק אשר מייצג לנו את שכבת הDAL (Data Access Layer) בSpring. הוא מאפשר לי מימוש אבסטרקטי לכל הגישה לDB באופן אוטומטי. זה בעיקר חוסך שכפול קוד שמתבצע שליפות דומות עבור כמה אובייקטים.

ברגע שאצור את הinterface הזה, מתאפשר לי האפשרות ליצור באופן אוטומטי שליפות בסיסיות.

**Service**

מחלקה אשר תייצג לנו את שכבת הBL בSpring. כל הלוגיקה של המערכת שלנו נמצאת בשכבת הBL (Business Logic). מטרות הService היא ביצוע לוגיקה על המידע שנשלוף מהDB או על פרמטרים שהתקבלו מהמשתמש.

**Controller**

מחלקה אשר אחראית על ניתוב בין בקשות הלקוח לפונקציות שמבצעות את הבקשות. הController מקשר בין שכבת הBL (Service) לבין שכבת הUI. עוד הסבר על Controller אפשר למצוא בשיעור Spring + REST.

**אז איך הכול מתקשר?**

הלקוח ניגש לURL מסוים עבור הבקשה אותה הוא רוצה. הController מזהה לפי הURL איזה מידע הלקוח צריך. הController פונה לService המתאים לצורך ביצוע הלוגיקה ובקשת המידע. הService פונה לRepository שמבצע את השליפה מהDB. המידע שהוחזר לService, עובר את הלוגיקה הרצויה וחוזר לController. הController מחזיר את המידע לצד הלקוח. כך מסתיימת הבקשה.

**נזכיר כי החיבור בין הDB לRepository מתבצע באמצעות המודלים שלנו המיצגים טבלאות בDB בעזרת האנוטציות עליהן הסברנו קודם.**

**והמימוש?**

כדי לבצע שליפות של המידע CRUD (Create, Read, Update and Delete) נצטרך ליצור עץ תיקיות וקבצים שייצג לנו את השכבות עליהן הסברנו. עבור כל מודל ניצור קובץ של Repository ושל Service.

**מימוש Repository**

כמו שאמרנו, הRepository הוא ממשק. עלינו ליצור ממשק ולדאוג שהוא יורש מהJpaRepository. ניתן לו כפרמטרים את שם המודל (שם המחלקה, במקרה שלנו Employee) ואת הטיפוס המורכב של המפתח הראשי (אם המפתח הוא int אז נרשום Integer).   
האנוטציה Repository@ מראה לSpring שהממשק הזה הוא אכן Repository.

@Repository

public interface EmloyeesRepository extends JpaRepository<Employee, Long> {

}

הJpaRepository מכיל בתוכו כמה פונקציות כתובות מראש שנועדו לעבודה עם הDB. למשל הפונקציה findAll() היא מתודה שמחזירה List של כל האובייקטים של המודל שלי. כלומר, מחזירה את כל הרשומות מהטבלה בDB. ([תוכלו לראות כאן עוד פעולות שמגיעות לנו עם JpaRepository](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)) העובדה שאנחנו יורשים מJpaRepository זה מה שמאפשר לנו לעשות את השליפות הבסיסיות באופן אוטומטי. יכולים לנסות בעצמכם ולראות מה ההשלמה האוטומטית מציעה לכם!

**מימוש Service**

בכדי להשתמש בService אנחנו יוצרים מחלקה רגילה ושים מעליה אנוטציה של @Service.

על מנת לשלוף בService, נצטרך להשתמש בRepository שבנינו, לשם כך נשים מופע של הRepository כמאפיין של מחלקת Service. כדי שלא נצטרך בכל פעם לאתחל את המאפיין הזה, נשים מעליו אנוטציה @Autowired והמחלקה תראה כך:

@Service

public class EmployeesService {

@Autowired

private EmployeesRepository employeesRepository;

public List<Employee> getAll() {

return this.employeesRepository.findAll();

}

}

**@Autowired** - בפועל האנוטציה אומרת שהמשתנה שנשים מעליו את האנוטציה הזאת יהיה מאותחל כבר במופע שלו ולא נצטרך לאתחל אותו כלומר לא נצטרך לעשות לו new או השמה של ערך, המופע נוצר לבד לפי הצורך בעזרת האנוטציה. (משתמש במנגנון הזרקת תלויות של Spring. למעשה, משתמש לנו בDependency Injection כפי שלמדנו בSOLID). [אפשר לקרוא פה עוד על הAutowired](https://www.baeldung.com/spring-autowire)

**מימוש Controller**

הController ממומש באותו האופן כמו בשיעור Spring + REST רק שיש להוסיף לו מופע של Service מתאים. בדומה למימוש של Service, נשתמש באנוטציה Autowired@ מעל המופע של הService.

@RestController

@RequestMapping(value = "/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService

@GetMapping

public getAllEmployees() {

return this.employeeService.findAll();

}

}

**שליפות מורכבות**

בכדי ליצור שליפות יותר מורכבות מהCrud (create, read, update, delete) הבסיסי יש כמה אפשרויות שאנחנו יכולים לעשות.

הדרך הראשונה והכי פשוטה היא שימוש באפשרות האוטומטית שהRepository מייחצנת לנו.

בדרך זו ניתן לבצע שליפות מורכבות מבלי לממש את השליפות – השליפה תתבצע רק באמצעות כתיבת **חתימה** לשגרה! הJPA יודע להמיר את החתימה לשאילתה המתאימה מאחורי הקלעים!

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);

}

הפונקציה findByEmailAddressAndLastName אוטומטית שולפת מהDB לפי הפרמטרים ששלחתי לה! לא צריך לממש כלום, זה נעשה באופן אוטומטי. יש מילים שמורות שעל פיהם הRepository יודע לבד לשלוף.

במקרה זה, לאחר הfindBy נכניס את שמות הDM של מה שנרצה לשלוף מהDB וביניהם את המילה השמורה And שאוטומטית תמיר את זה לשאילתה:

"WHERE x.emailaddress = ?1 AND x.lastname = ?2"

**שימו לב!** ה?1 ו?2 הם הפרמטרים שלנו, הEmailAddress והLastName הם **השמות של הDM** שיש לנו באובייקט User **ולא** השמות של העמודות בDB.

כמו And יש לנו עוד הרבה מילים שמורות שאפשר לתת לRepository על מנת ליצור שליפות מורכבות

בצורה אוטומטית.

יש לעבור על [הקישור](https://www.javaguides.net/2018/11/spring-data-jpa-query-creation-from-method-names.html) הבא, שם תמצאו טבלה המכילה דוגמאות לשימוש במילים שמורות לצורך שליפות מורכבות באמצעות חתימה בלבד.