* Querying for Java Objects

TypedQuery<Student> aQuery = entityManager.createQuery(“from student”, Student.class);

List<Student> list = aQuery.getResultList();

Above code returns the list of student from Database.

* We can notice that we don’t have to do any kind of low level SQL coding

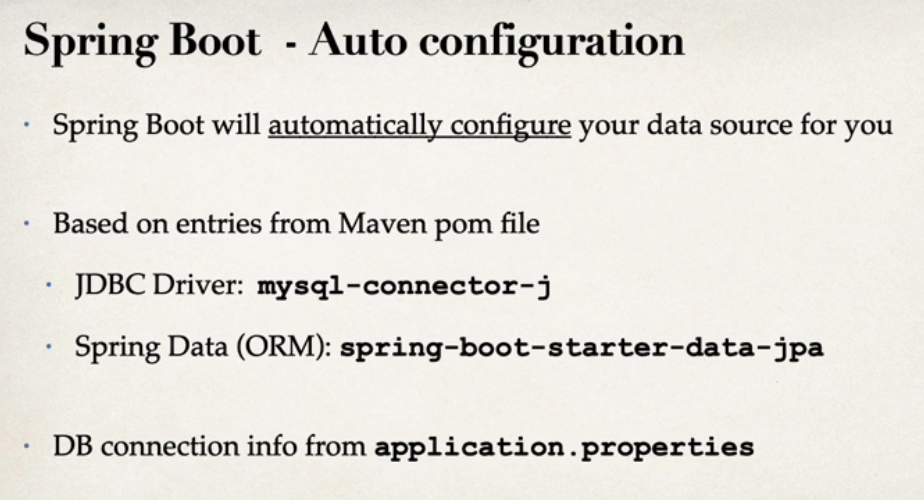
**Relation between Hibernate JPA and JDBC**

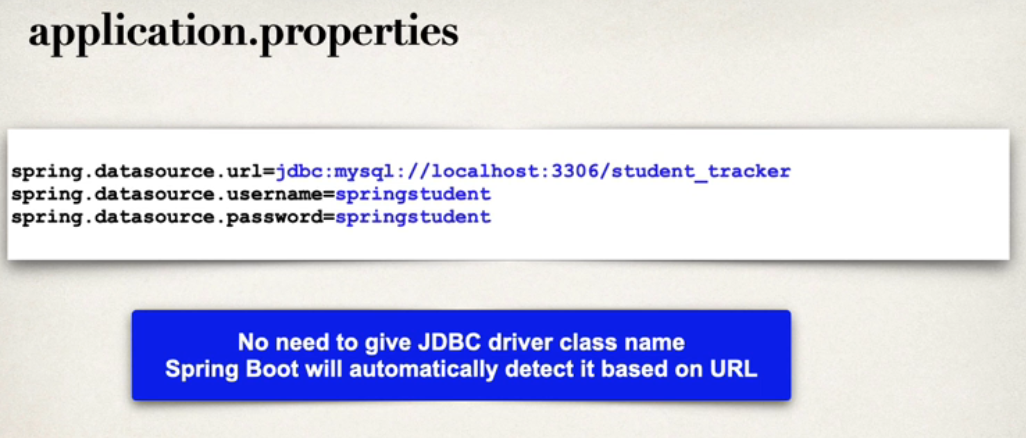
Hibernate framework is build on top of JDBC layer. So Hibernate makes many things easier for us at high level, but behind the scenes Hibernate also uses JDBV APIs.

* In Spring boot Hibernate is the default implementation of JPA
* Entity Manager is main component for creating Queries
* Entity Manager comes from Jakarta Persistentce API
* Based on configs, Spring will automatically create the beans – DataSource, EntityManager

Dependencies:

* MySql driver: mysql-conector-j
* Spring Data Jpa – spring-boot-starter-data-jpa



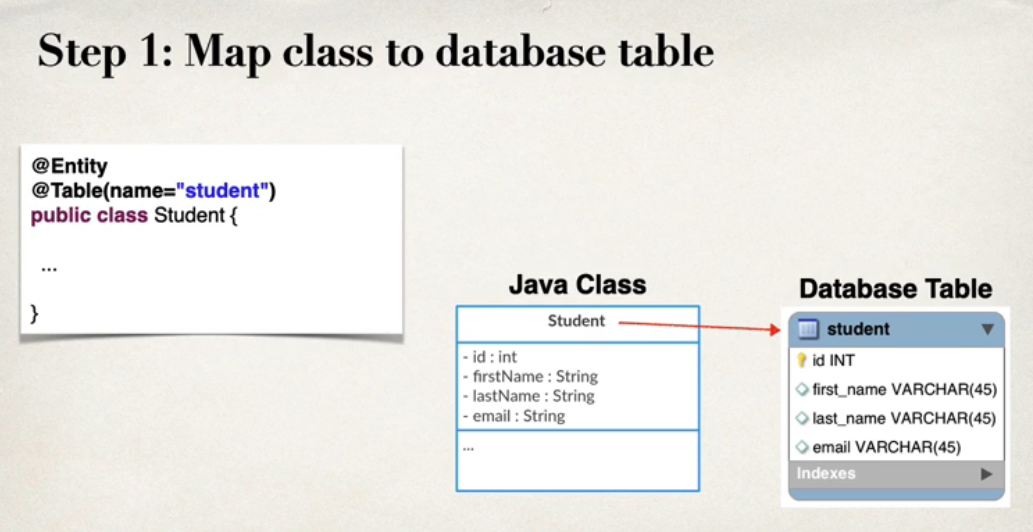


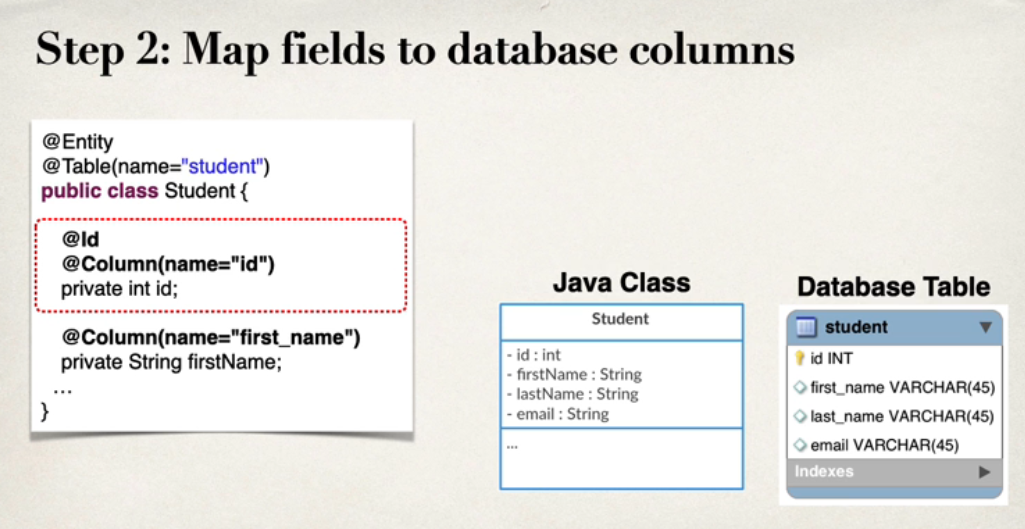
Terminologies:

* Entity – A Java class that is mapped to a table in Database

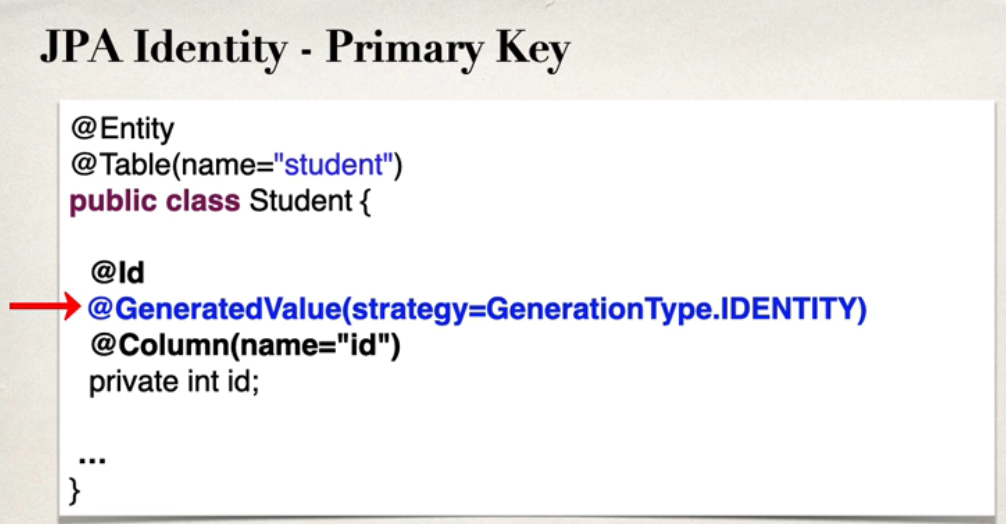
Must be annotated with @Entity

Must have a public/protected No-Arg constructor, and can’t have any other ctor





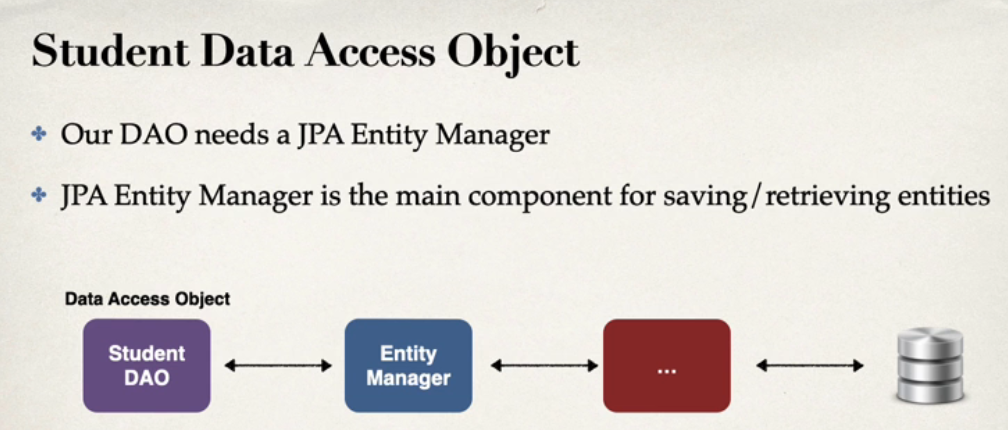
* Use of @Column is optional, if we ignore it the name of database column will be same as name of that property 🡪 this is not recommended
* Same applies for @Table



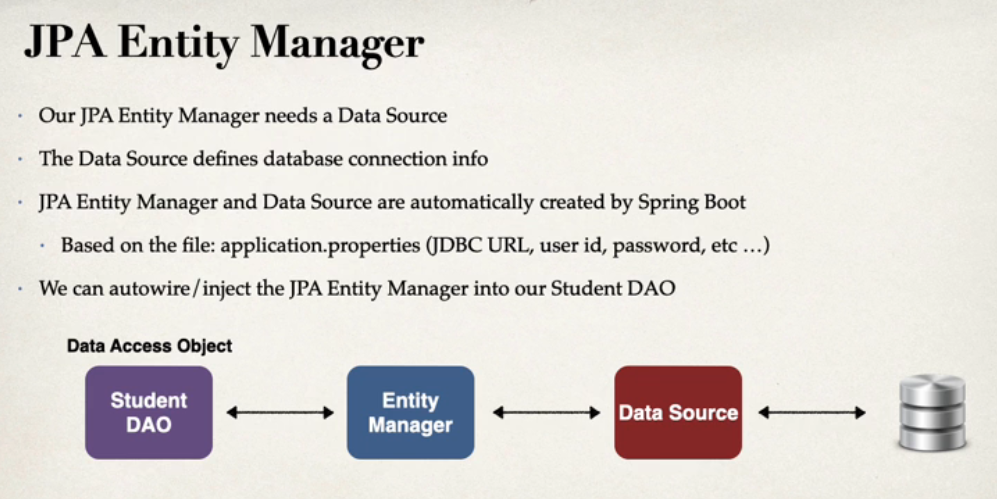
In Spring boot , we can define our own custom generation strategy

For this we have to Implement **org.Hibernate.id.IdentifierGenerator**

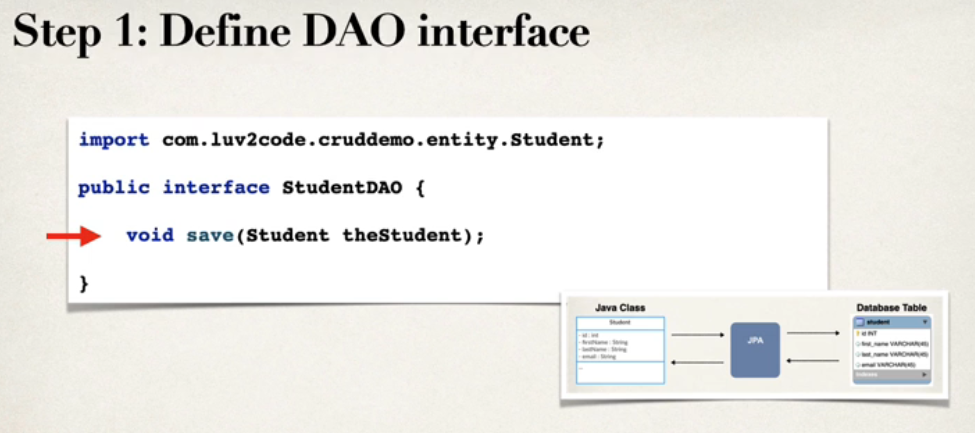
**Override the method public Serializable Generate(…)**

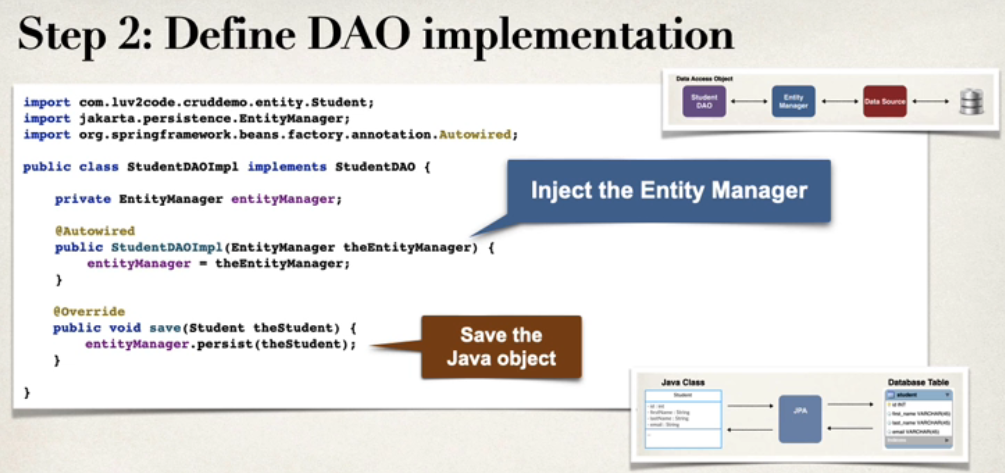


* Our EntityManager needs a DataSource object
* DataSource is basically the info to our Database
* JPA Entityt Manager DataSource obj is auto created by Spring boot
* Then we have to wire the EntityManager object inside our DAO object



* Steps to Perform Save





* **@Transactional**

Provided by Spring Framework. This automatically begins and end a Transaction for our JPA Code so we no need to do Transaction explicitly

**Specialized Annotation For DAOs**

**@Repository**

* @Repository is a Spring annotation that indicates that the decorated class is a repository. A repository is a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects. It is a specialization of the @Component annotation allowing for implementation classes to be autodetected through classpath scanning.
* @ComponentScan ensures that the classes decorated with @Component and their derivatives including @Repository are found and registered as Spring beans. @ComponentScan is automatically included with @SpringBootApplication.