ORM - JPA - Hibernate

Codecool, 2021

Agenda

What is ORM?

JPA

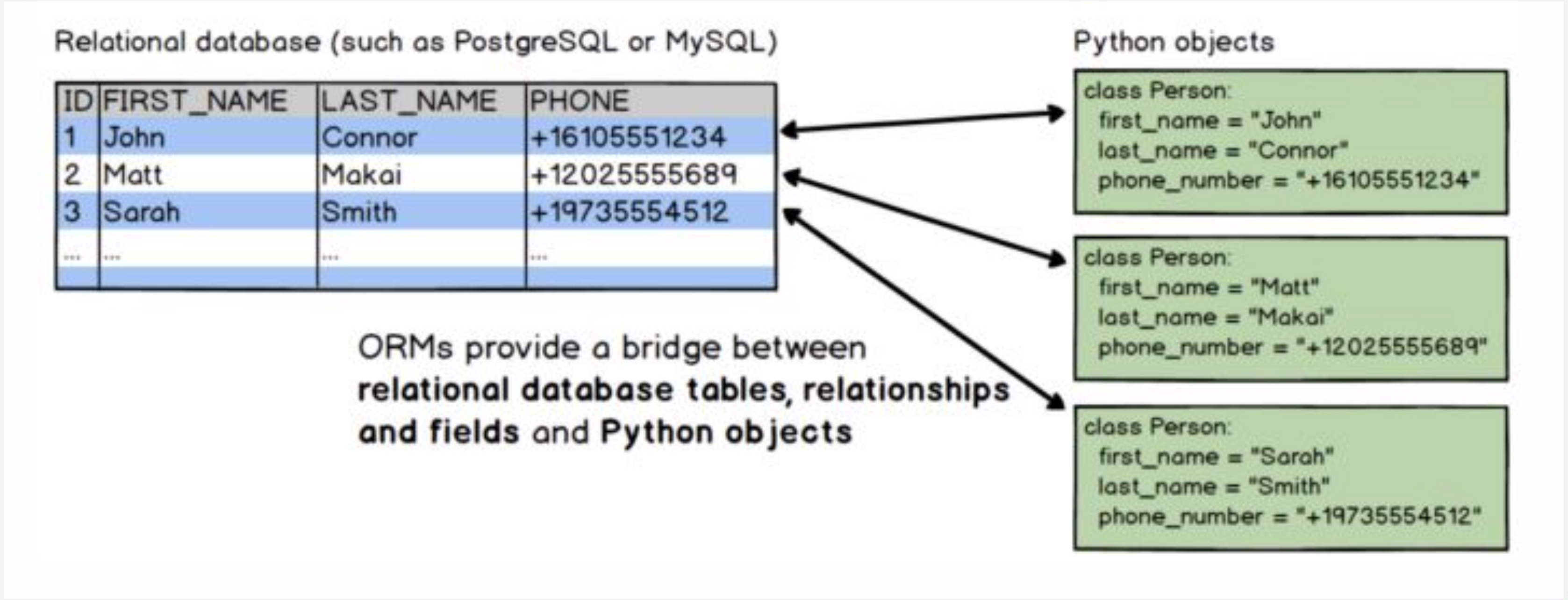
Hibernate

Annotations

ORM

Object–relational mapping

Converts data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language.


Relational database (such as PostgreSQL or MySQL)

ID	FIRST_NAME	LAST_NAME	PHONE
1	John	Connor	+16105551234
2	Matt	Makai	+12025555689
3	Sarah	Smith	+19735554512
...	...	...	...

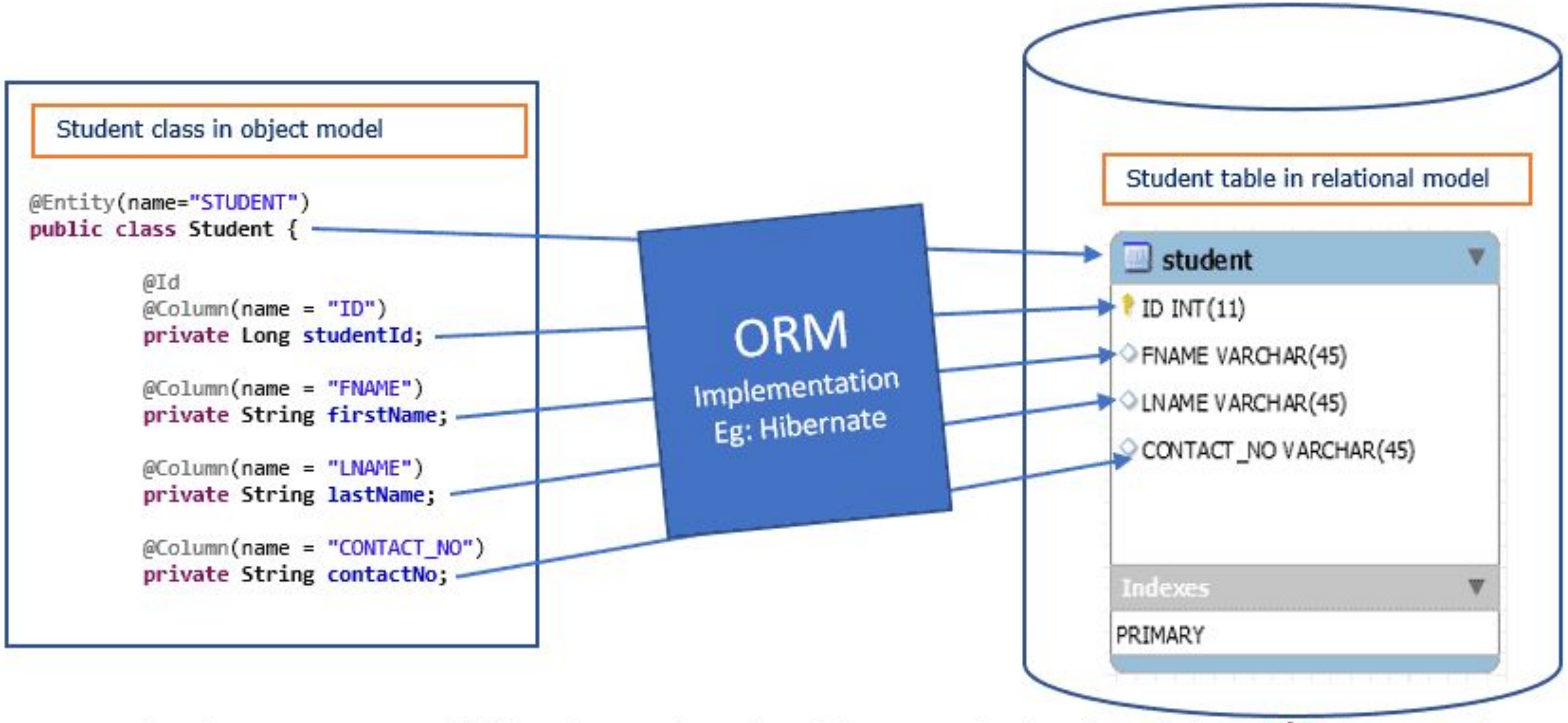
Python objects

```
class Person:
  first_name = "John"
  last_name = "Connor"
  phone_number = "+16105551234"
```

```
class Person:
  first_name = "Matt"
  last_name = "Makai"
  phone_number = "+12025555689"
```

```
class Person:
  first_name = "Sarah"
  last_name = "Smith"
  phone_number = "+19735554512"
```

ORMs provide a bridge between **relational database tables, relationships and fields** and **Python objects**


Student class in object model
Student table in relational model
ORM
Implementation
Eg: Hibernate
student
Indexes
PRIMARY

```
@Entity(name="STUDENT")
public class Student {

        @Id
        @Column(name = "ID")
        private Long studentId;

        @Column(name = "FNAME")
        private String firstName;

        @Column(name = "LNAME")
        private String lastName;

        @Column(name = "CONTACT_NO")
        private String contactNo;
```

```
ID INT(11)
FNAME VARCHAR(45)
LNAME VARCHAR(45)
CONTACT_NO VARCHAR(45)
```

ORM implements responsibility of mapping the Object to Relational Model.

JavaByDeveloper

Hibernate

- Hibernate ORM (or simply Hibernate)
- object–relational mapping tool for the Java programming language.
- Framework
- mapping an object-oriented domain model to a relational database.

CODECOOL

Java Persistence (API) - JPA

It is a Jakarta EE application programming interface specification that describes the management of relational data in enterprise Java applications.

The API itself, defined in the **javax.persistence** package

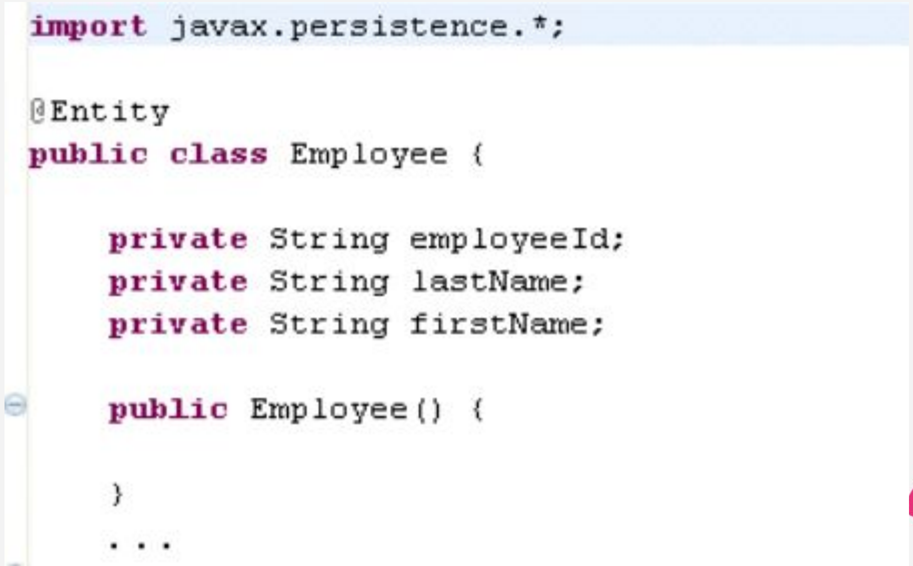
https://dzone.com/articles/all-jpa-annotations-mapping-annotations

Entity

- lightweight Java class with its state typically persisted to a table in a relational database.
- Instances of such an entity correspond to individual rows in the table.
- Entities typically have relationships with other entities, and these relationships are expressed through object/relational metadata. This metadata may be specified directly in the entity class file by using annotations or in a separate XML descriptor file distributed with the application.

@Entity

It is used to specify that the currently annotate class represents an entity type.


```
import javax.persistence.*;

@Entity
public class Employee {

    private String employeeId;
    private String lastName;
    private String firstName;

    public Employee() {

    }
    ...
```

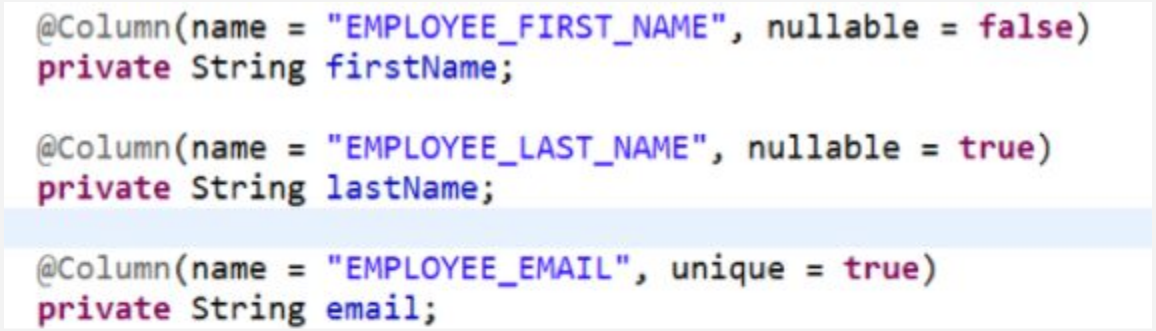
@Table

It is used to specify the primary table of the currently annotated entity.

- @Enity(name) vs @Table
https://stackoverflow.com/questions/18732646/name-attribute-in-entity-and-table

@Column

It is used to specify the mapping between a basic entity attribute and the database table column.



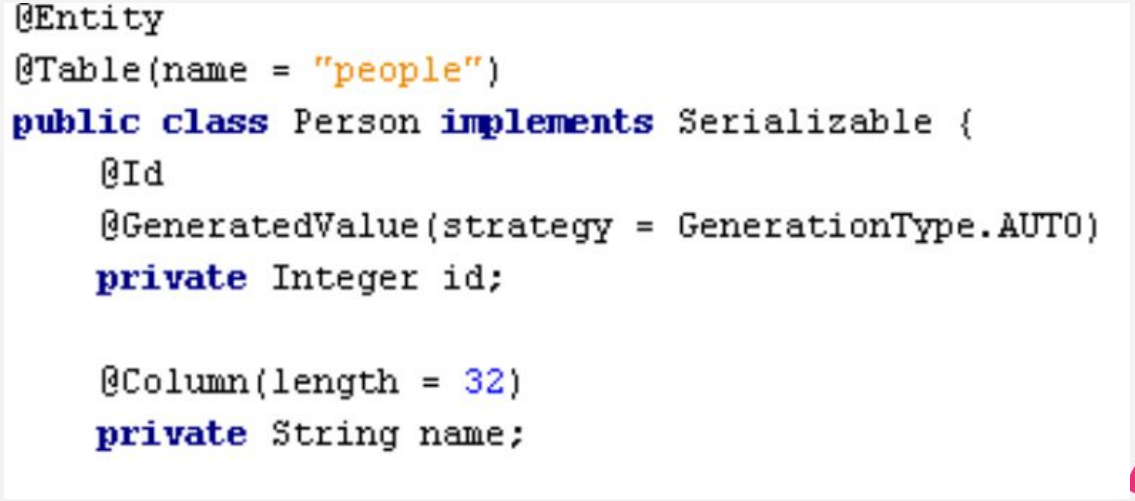
```
@Column(name = "EMPLOYEE_FIRST_NAME", nullable = false)
private String firstName;

@Column(name = "EMPLOYEE_LAST_NAME", nullable = true)
private String lastName;

@Column(name = "EMPLOYEE_EMAIL", unique = true)
private String email;
```

@Id

It specifies the entity identifier. An entity must always have an identifier attribute, which is used when loading the entity in a given Persistence Context.



```
@Entity
@Table(name = "people")
public class Person implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    @Column(length = 32)
    private String name;
```

CODECOOL