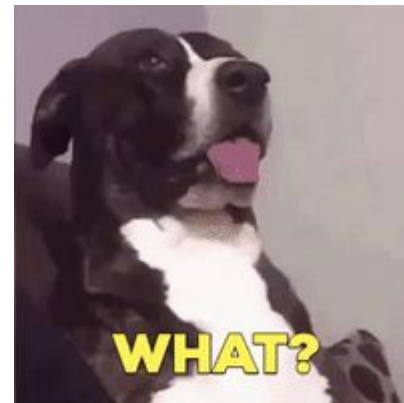


Módulo 05

Tainá Medeiros



O que são Informações?

- Dados são fatos em seu estado bruto
 - Não possuem utilidade alguma isoladamente
- Informações são o resultado do processamento dos dados
 - A análise de um conjunto de dados pode gerar informações
 - Possuem utilidade

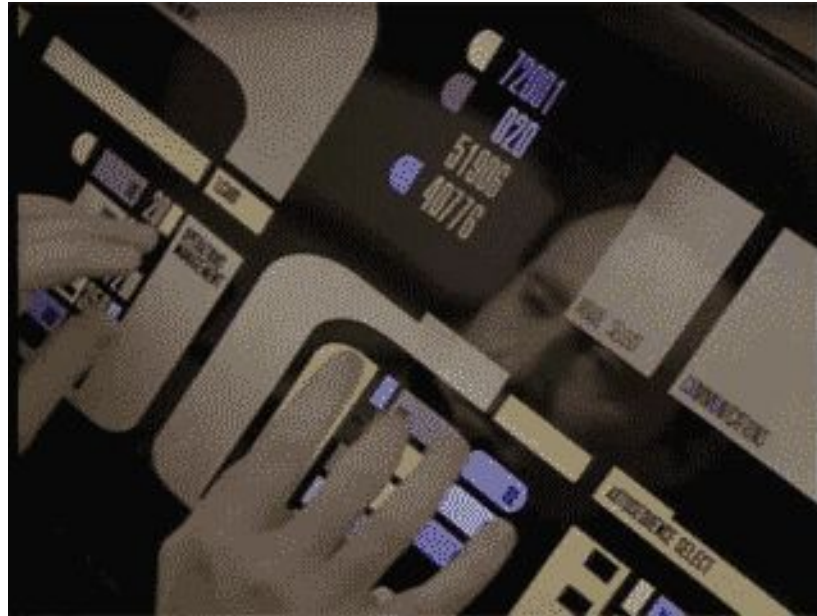


De onde surgem as informações?

2019 *This Is What Happens In An Internet Minute*



Onde são Armazenadas as Informações?



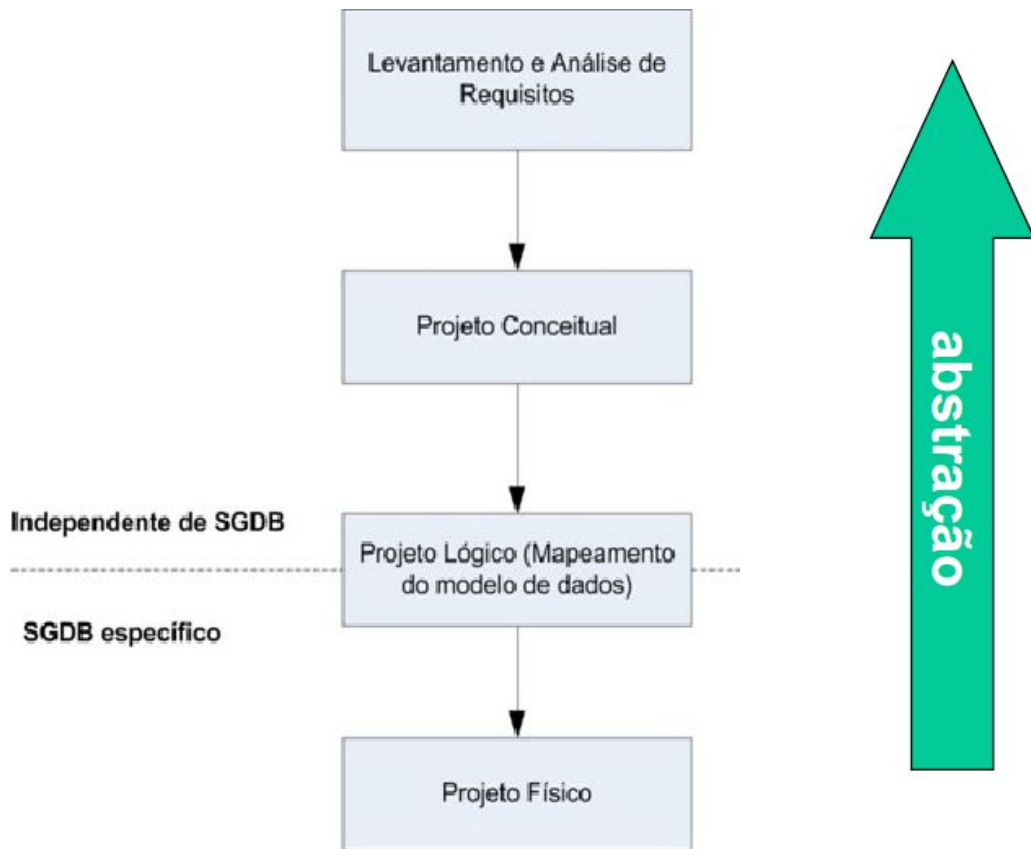
Banco de Dados

- Um banco de dados é um software que armazena um conjunto de dados inter-relacionados, representando informações sobre um domínio específico, ou seja, sempre que for possível agrupar informações que se relacionam e tratam de um mesmo assunto, posso dizer que tenho um banco de dados (KORTH, 1994).

Onde são Armazenadas as Informações?

- Existem vários tipos de bancos de dados, no entanto, os principais são:
 - Banco de dados relacional: modelo que usa relações para representar e armazenar os dados. Relação é um conceito da álgebra relacional que é utilizado para modelar banco de dados relacionais. Não se preocupe com o conceito de relações e entidades, pois será detalhado nesta aula e nas aulas seguintes.
 - Banco de dados orientado a objetos: esse modelo usa a mesma ideia de objetos da programação orientada a objetos para representar e armazenar os dados. Esse tipo de banco de dados é muito empregado em aplicações que demandam georeferenciamento.
 - Banco de dados NoSQL: Os bancos de dados NoSQL usam diversos modelos para acessar e gerenciar dados, como documento, gráfico, chave-valor, em memória e pesquisa,

Projeto de um Banco de Dados



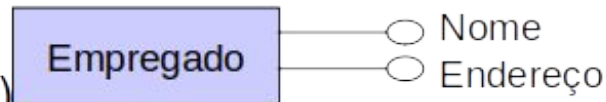
Modelos de Dados

- **Modelo conceitual** (projeto conceitual)

- Modelo de dados abstrato que descreve a estrutura de um banco de dados independente de um SGBD

- **Modelo lógico** (projeto lógico)

- Modelo de dados que representa a estrutura dos dados de um banco de dados
 - Dependente do modelo do SGBD



Empregado (Nome, Endereço)

- **Modelo físico** (projeto físico)

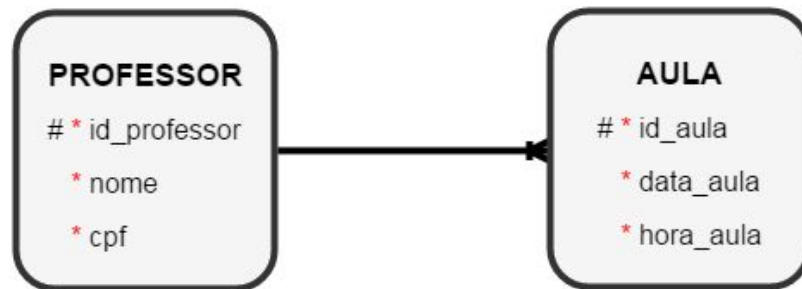
- Nível de Implementação
- Depende do SGBD
- ênfase na eficiência de acesso

Projeto de um Banco de Dados

Nível conceitual



Nível Lógico



Modelo Entidade Relacionamento



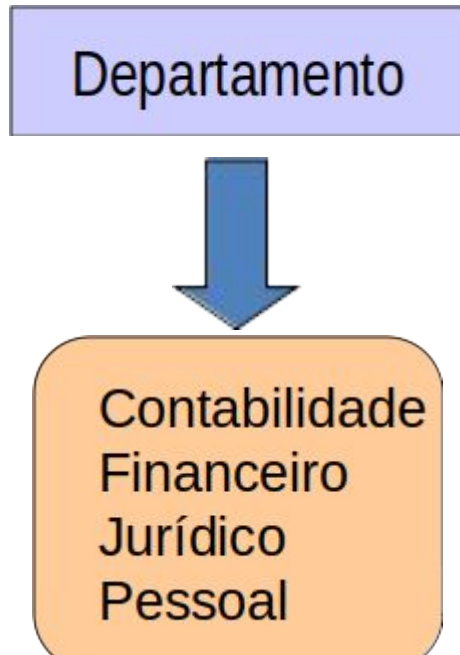
Entidade



Entidade

Entidade

- É um conjunto de objetos do mundo real sobre os quais se deseja manter informações no banco de dados
- É distinguível de outros objetos
- Representada através de um retângulo
- Pode representar:
 - objetos concretos (uma pessoa)
 - objetos abstratos (um departamento)



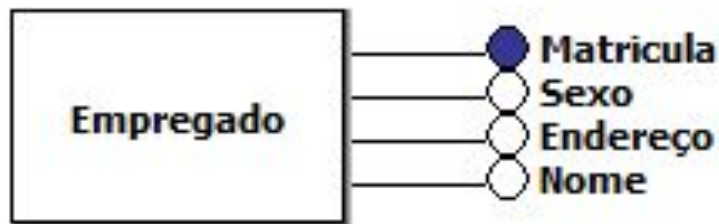
Atributos



Atributos

Para saber quais são os atributos de uma entidade, você deve perguntar que informações são necessárias guardar sobre aquela entidade.

- CPF, Nome, Sexo, Endereço, Data de Nascimento etc.



- Representados por um círculo com o nome do atributo dentro.

Relacionamentos



Relacionamentos

Representa a forma como as entidades se relacionam no modelo ER

Representado com um losango e o nome que especifica o relacionamento no meio

Possui uma ligação para as entidades envolvidas no relacionamento



Cardinalidade



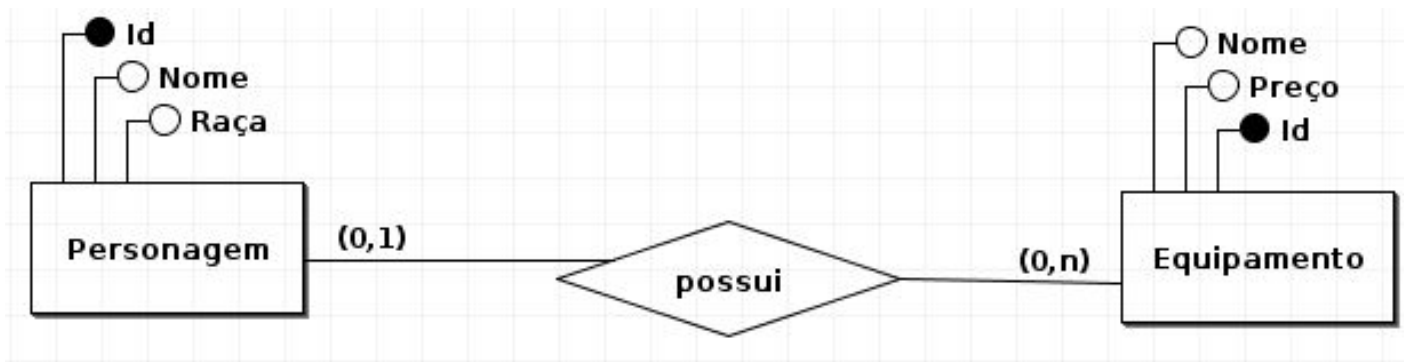
Cardinalidade

Existem dois tipos de cardinalidades:

- Máxima.
- Mínima.

Cardinalidades Possíveis:

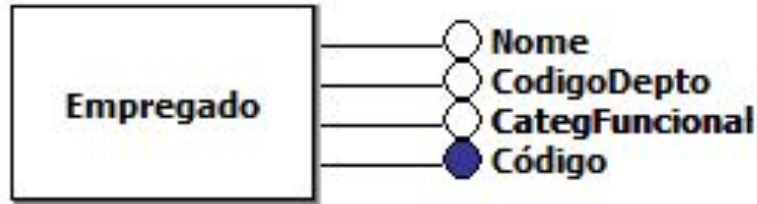
- (1,1);
- (1,N);
- (0,1);
- (0,N);
- (N,N).



Composição de um Banco de Dados Relacional

Tabelas

- Linhas;
- Colunas;
- Chaves;



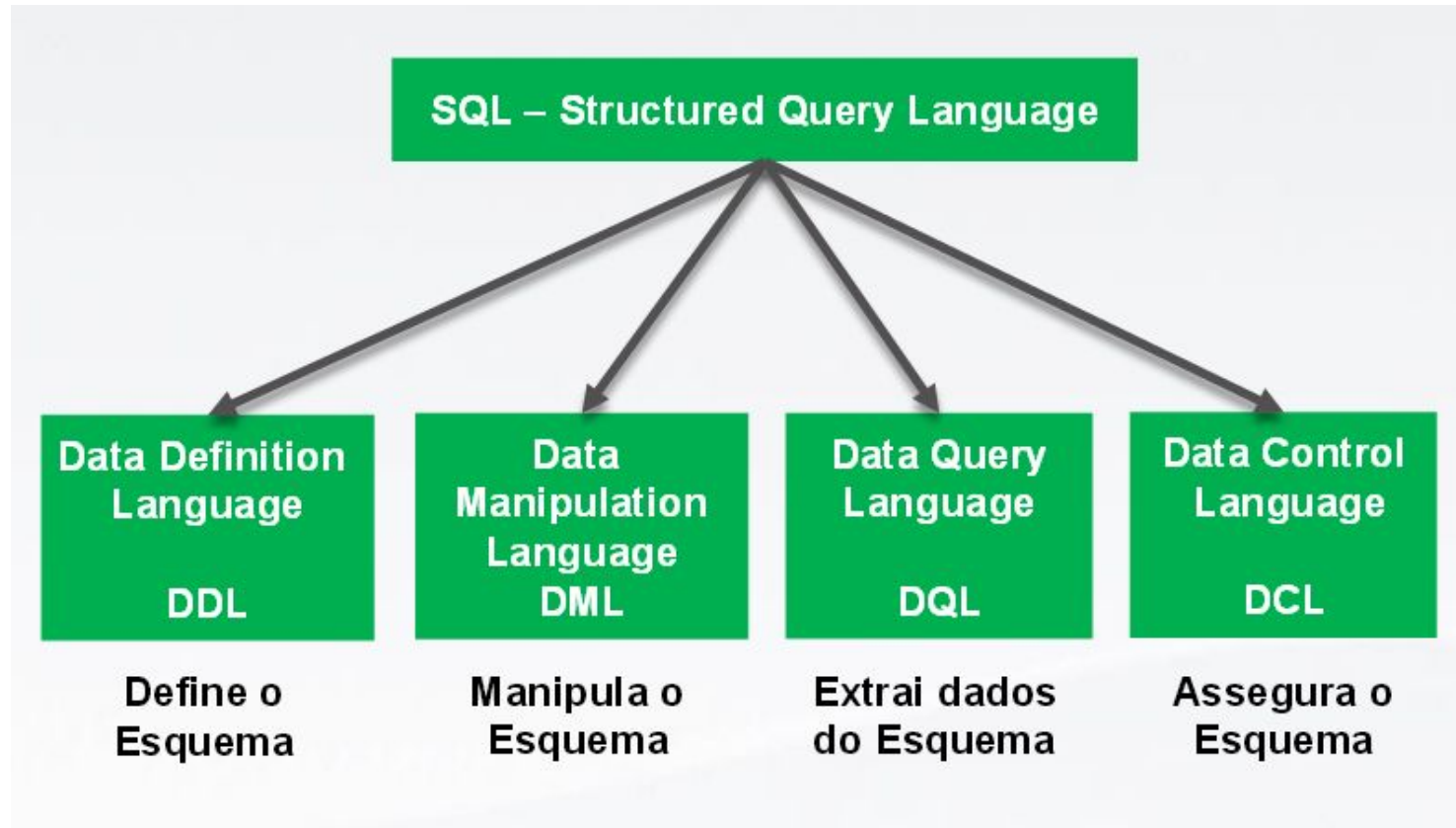
Emp:

CodigoEmp	Nome	CodigoDepto	CategFuncional
E5	Souza	D1	C5
E3	Santos	D2	C5
E2	Silva	D1	C2
E1	Soares	D1	—

Estrutura da SQL



Estrutura da SQL



Data Definition Language- DDL

Principais comandos:

- CREATE TABLE
 - Cria uma nova tabela em um banco de dados existente
- ALTER TABLE
 - Altera uma tabela em um banco de dados existente
- DROP TABLE
 - Delete uma tabela em um banco de dados existente

CREATE TABLE – Sintaxe

```
13 CREATE TABLE IF NOT EXISTS table_name
14 (
15     column_name1 data_type(size) constraint_name,
16     column_name2 data_type(size) constraint_name,
17     column_name3 data_type(size) constraint_name,
18 ) engine=table_type;
```

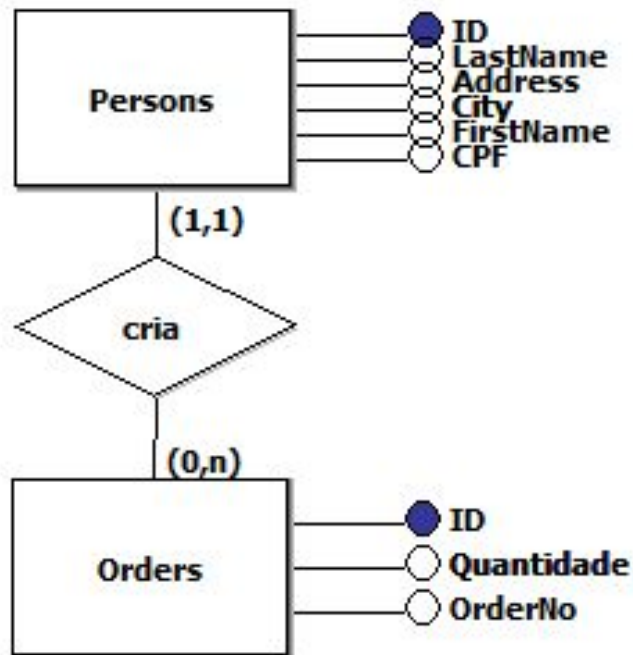
DDL Constraint

- NOT NULL
 - Garante que a coluna não pode ter valor NULL.
- UNIQUE
 - Garante que todos os valores em uma coluna são diferentes.
- PRIMARY KEY
 - A combinação de um NOT NULL e UNIQUE. Garante que uma coluna (ou combinação de duas ou mais colunas) tem uma identidade única, que ajuda a encontrar um registro específico de uma tabela mais fácil e rapidamente.
- FOREIGN KEY
 - Garantir a integridade referencial dos dados em uma tabela para coincidir com os valores em outra tabela.

DDL Constraint

- CHECK
 - A restrição CHECK garante que todos os valores em uma coluna de satisfazer determinadas condições.
- DEFAULT
 - Fornece um valor padrão para uma coluna quando nenhum é especificado.
- INDEX
 - Use para criar e recuperar dados do banco de dados muito rapidamente.
- AUTO_INCREMENT
 - A coluna é incrementada com algum valor.

DDL Constraint



```
20 • CREATE TABLE Empresa.Persons
21   (
22     P_Id int AUTO_INCREMENT,
23     CPF int NOT NULL UNIQUE,
24     LastName varchar(255) ,
25     FirstName varchar(255) NOT NULL,
26     Address varchar(255),
27     City varchar(255) DEFAULT 'Sandnes',
28     UNIQUE (CPF),
29     PRIMARY KEY (P_Id),
30     CHECK (P_Id>0));
31
32 • CREATE TABLE Empresa.Orders
33   (
34     O_Id int NOT NULL,
35     OrderNo int NOT NULL,
36     Quantity int NOT NULL,
37     P_Id int,
38     PRIMARY KEY (O_Id),
39     FOREIGN KEY (P_Id) REFERENCES Persons(P_Id),
40     CHECK(Quantity >= 1 AND Quantity <= 1000));
```

ALTER TABLE - Sintaxe

```
61 #Selecionando um banco para fazer as transações
62 • Use Empresa;
63 #Renomeando a tabela "teste" para "ievolution":
64 • ALTER TABLE teste RENAME TO Empregado;
65 #Vamos adicionar uma coluna do tipo varchar nessa tabela:
66 • ALTER TABLE Empregado ADD COLUMN telefone VARCHAR(50) NOT NULL;
67 #cria a nova coluna antes da coluna endereco
68 • ALTER TABLE Empregado ADD COLUMN Id VARCHAR(50) FIRST;
69 #cria a nova coluna depois da coluna Id
70 • ALTER TABLE Empregado ADD COLUMN nome VARCHAR(50) AFTER Id;
71 #Vamos renomear essa coluna "teste" para "nacionalidade" e mudar seu tipo:
72 • ALTER TABLE Empregado CHANGE COLUMN teste nacionalidade VARCHAR(50);
73 #Podemos adicionar uma chave primária nessa tabela:
74 • ALTER TABLE Empregado ADD PRIMARY KEY (Id);
75 #Podemos adicionar também chave estrangeiras nessa tabela:
76 • ALTER TABLE Empregado ADD FOREIGN KEY (IdEmpresa) REFERENCES Empresa (IdEmpresa);
77 #Para eliminar uma coluna da tabela basta fazer:
78 • ALTER TABLE Empregado DROP COLUMN nacionalidade;
```

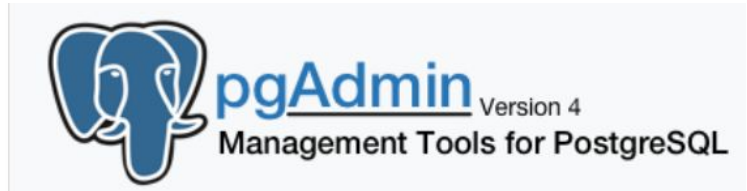
DROP TABLE - Sintaxe

```
83      #Deletando a chave primaria
84 •    ALTER TABLE Empregado DROP PRIMARY KEY;
85      #Deletando a chave estrangeira
86 •    ALTER TABLE Empregado DROP FOREIGN KEY IdEmpresa;
87      #Deletando a tabela
88 •    DROP TABLE Empregado;
```

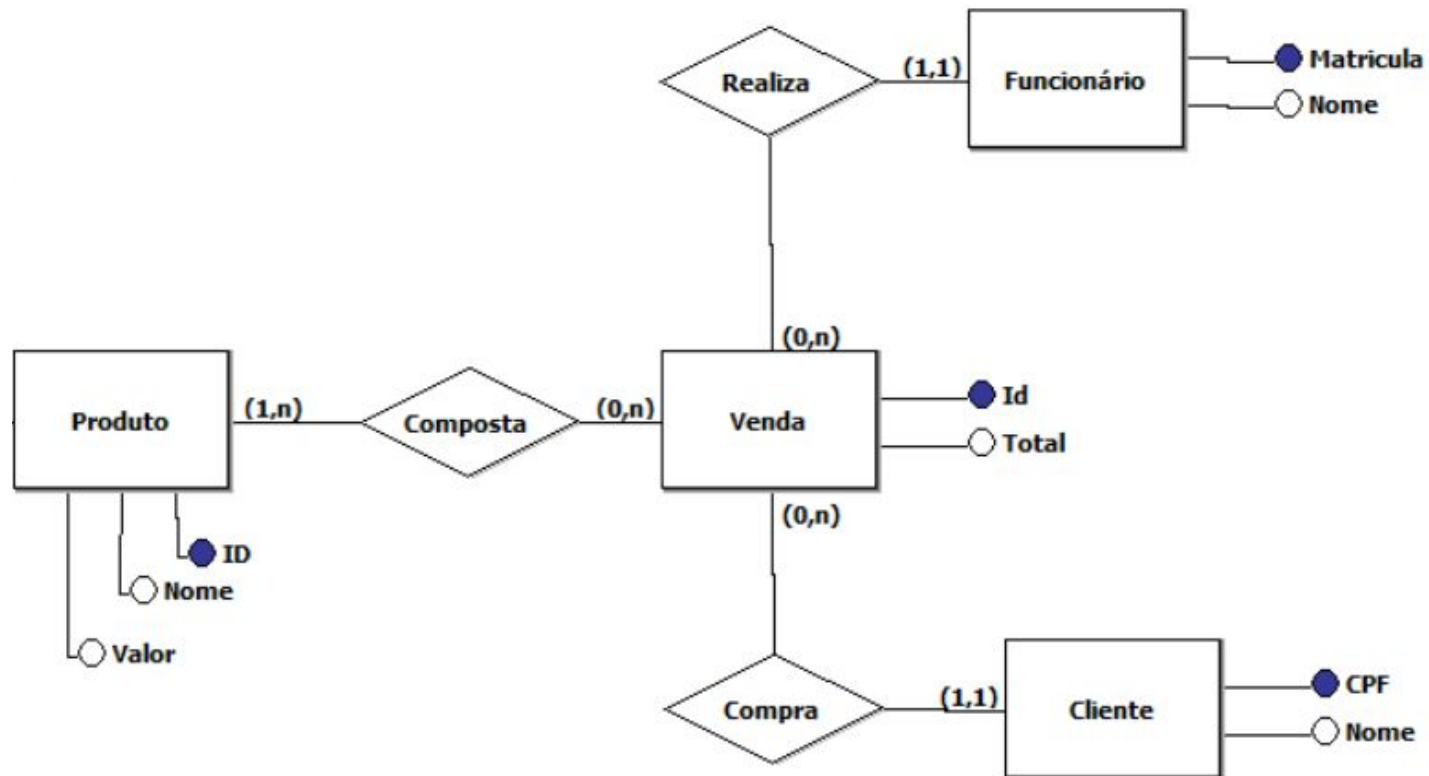


"To start, press any key."
Where's the "any" key?

Instalando e configurando o SQL



Criando nosso banco...



Criando as tabelas..

```
create table funcionario (  
  matricula int NOT NULL UNIQUE,  
  nome varchar(255) NOT NULL,  
  PRIMARY KEY (matricula)  
);
```

```
create table cliente (  
  cpf int NOT NULL UNIQUE,  
  nome varchar(255) NOT NULL,  
  PRIMARY KEY (cpf)  
);
```

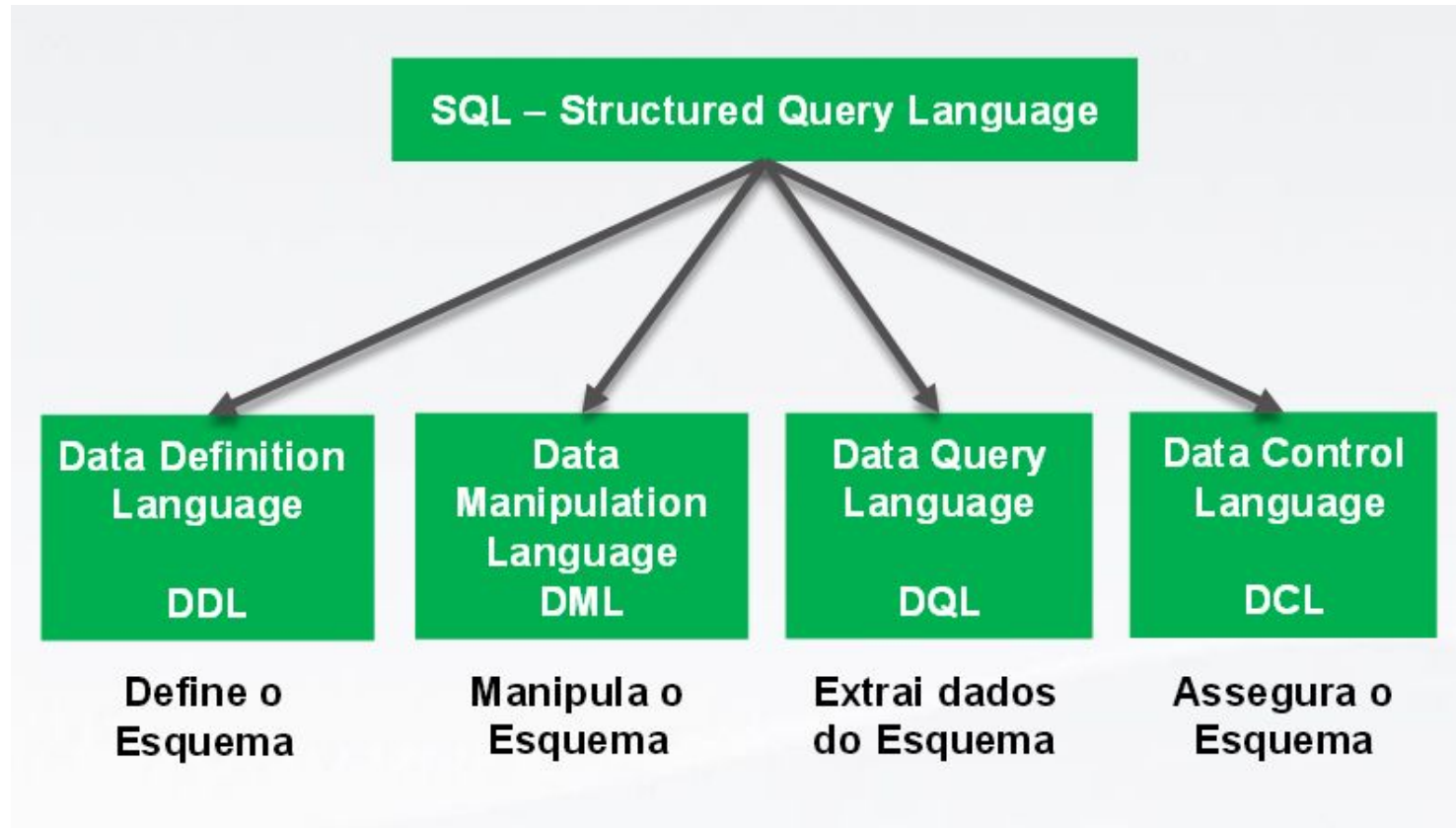
```
create table venda (  
  id SERIAL,  
  total float,  
  PRIMARY KEY (id)  
);
```

```
create table produto (  
  id SERIAL,  
  nome varchar(255) NOT NULL,  
  valor float,  
  PRIMARY KEY (id)  
);
```


E se tudo deu certo....



Estrutura da SQL



Data Manipulation Language - DML

Principais comandos:

- **INSERT**
 - Instrução para inserir algum valor no banco.
- **UPDATE**
 - Instrução para alterar algum valor no banco
- **DELETE**
 - Instrução para deletar algum valor no banco

INSERT - Sintaxe

```
19 • INSERT INTO table_name  
20 VALUES (value1,value2,value3, etc);  
21  
22 • INSERT INTO table_name (column1,column2,column3, etc)  
23 VALUES (value1,value2,value3, etc);
```

INSERT

```
18 • use Empresa;
19
20 • INSERT INTO empresa
21   VALUES (1, 'Mercadinho do Jose');
22
23 • INSERT INTO empresa (Nome)
24   VALUES ('Mercadinho da Maria');
25
26 • INSERT INTO setor
27   VALUES (1,1, 'Almoxarifado', 500);
28
29 • INSERT INTO setor (IdEmpresa,IdSetor,Nome, Tamanho)
30   VALUES (1,2, 'Financeiro', 500);
31
32 • INSERT INTO setor (IdEmpresa,Nome, Tamanho)
33   VALUES (1, 'Distribuidor', 1500);
34
35 • INSERT INTO setor (IdEmpresa,Nome, Tamanho)
36   VALUES (1, 2, 'Administrativo', 1500);
```

```
3 • CREATE TABLE IF NOT EXISTS Empresa.Empresa (
4   IdEmpresa INT AUTO_INCREMENT,
5   Nome VARCHAR(255),
6   PRIMARY KEY (IdEmpresa));
7
```

UPDATE - Sintaxe

```
145 • UPDATE nome_tabela  
146 SET Campo1 = "novo_valor", Campo2 = "novo_valor", CampoN = "novo_valor"  
147 WHERE some_column=some_value;
```

UPDATE

```
149 • use Empresa;
150
151 • UPDATE setor
152   SET Nome = "Cafezinho"
153   where IdSetor = 1;
154
155 • UPDATE setor
156   SET Tamanho = 600, IdEmpresa = 2, Nome = "Cafezinho"
157   where Nome = "Financeiro";
158
159 • UPDATE setor
160   SET Nome = "Financeiro"
161   where Nome = "Cafezinho";
```

DELETE - Sintaxe

```
171 • DELETE FROM table_name  
172 WHERE some_column=some_value;
```


DELETE

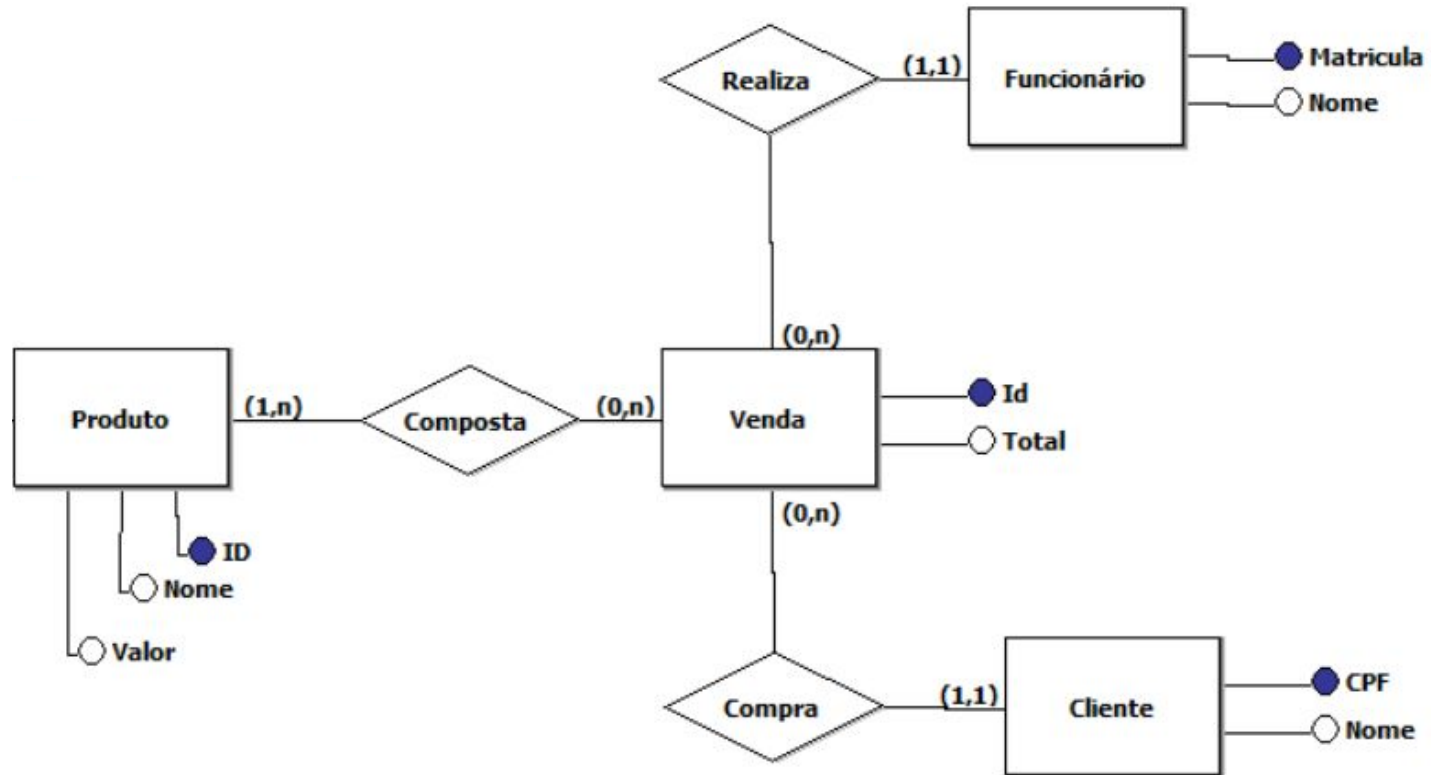
```
175 • DELETE FROM setor
176 WHERE Nome='Cafezinho' AND IdEmpresa='5';
177
178 • DELETE FROM setor
179 WHERE IdEmpresa='10';
```

DELETE

182 • `DELETE FROM table_name;`



Manipulando nosso banco...



Inserindo registro

```
insert into funcionario  
values (1, 'Maria');
```

```
insert into funcionario values  
(2, 'Joao'),  
(3, 'Ana');
```

E se tudo deu certo....

	matricula [PK] integer	nome character varying(255)
1	1	Maria
2	2	Joao
3	3	Ana
*		



Atualizando registro

```
update funcionario  
set nome = 'Murilo'  
where matricula = 2;
```

E se tudo deu certo....

	matricula [PK] integer	nome character varying(255)
1	1	Maria
2	2	Joao
3	3	Ana
*		

	matricula [PK] integer	nome character varying(255)
1	1	Maria
2	2	Murilo
3	3	Ana
*		



Deletar registro

```
delete from funcionario  
where nome = 'Murilo';
```


E se tudo deu certo....

	matricula [PK] integer	nome character varying(255)
1	1	Maria
2	2	Joao
3	3	Ana
*		

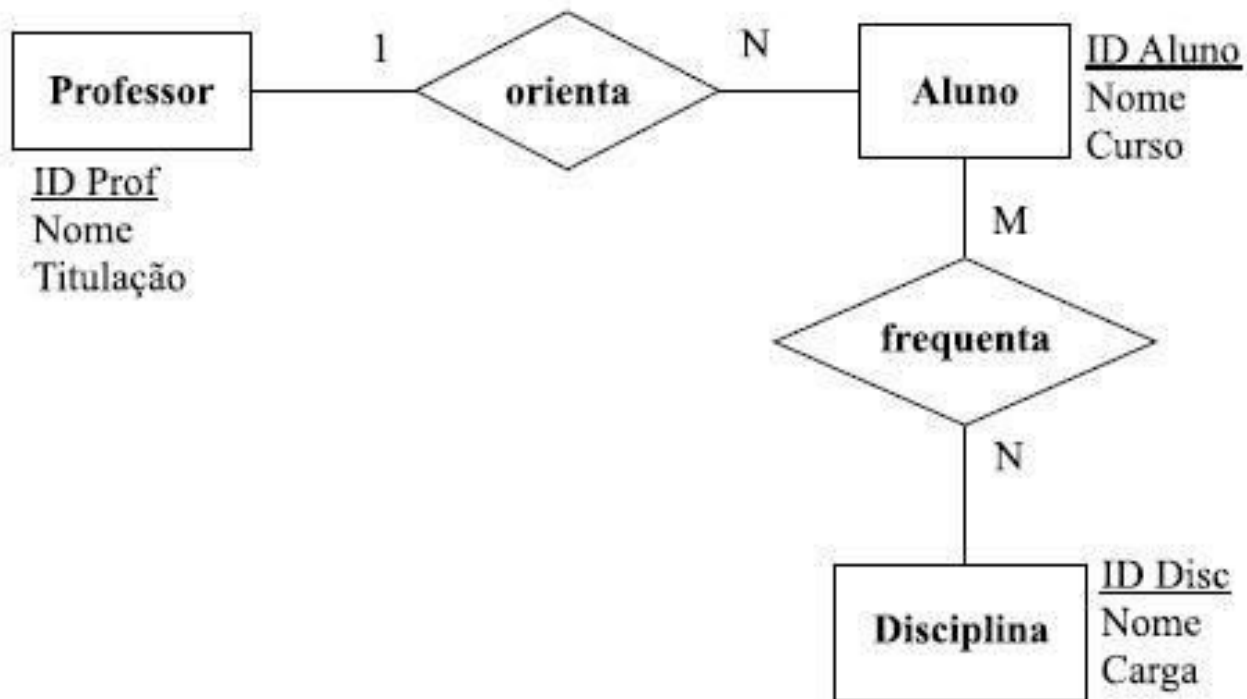
	matricula [PK] integer	nome character varying(255)
1	1	Maria
2	2	Murilo
3	3	Ana
*		

	matricula [PK] integer	nome character varying(255)
1	1	Maria
2	3	Ana
*		





Praticando



Praticando

- Crie a tabela professore
- Crie a tabela aluno
- Crie a tabela disciplina
- Crie um novo atributo de idade na tabela de aluno depois do atributo curso
- Crie 4 professores
- Crie 10 alunos
- Crie 3 disciplinas
- Delete 1 professor
- Altere o nome de 1 aluno

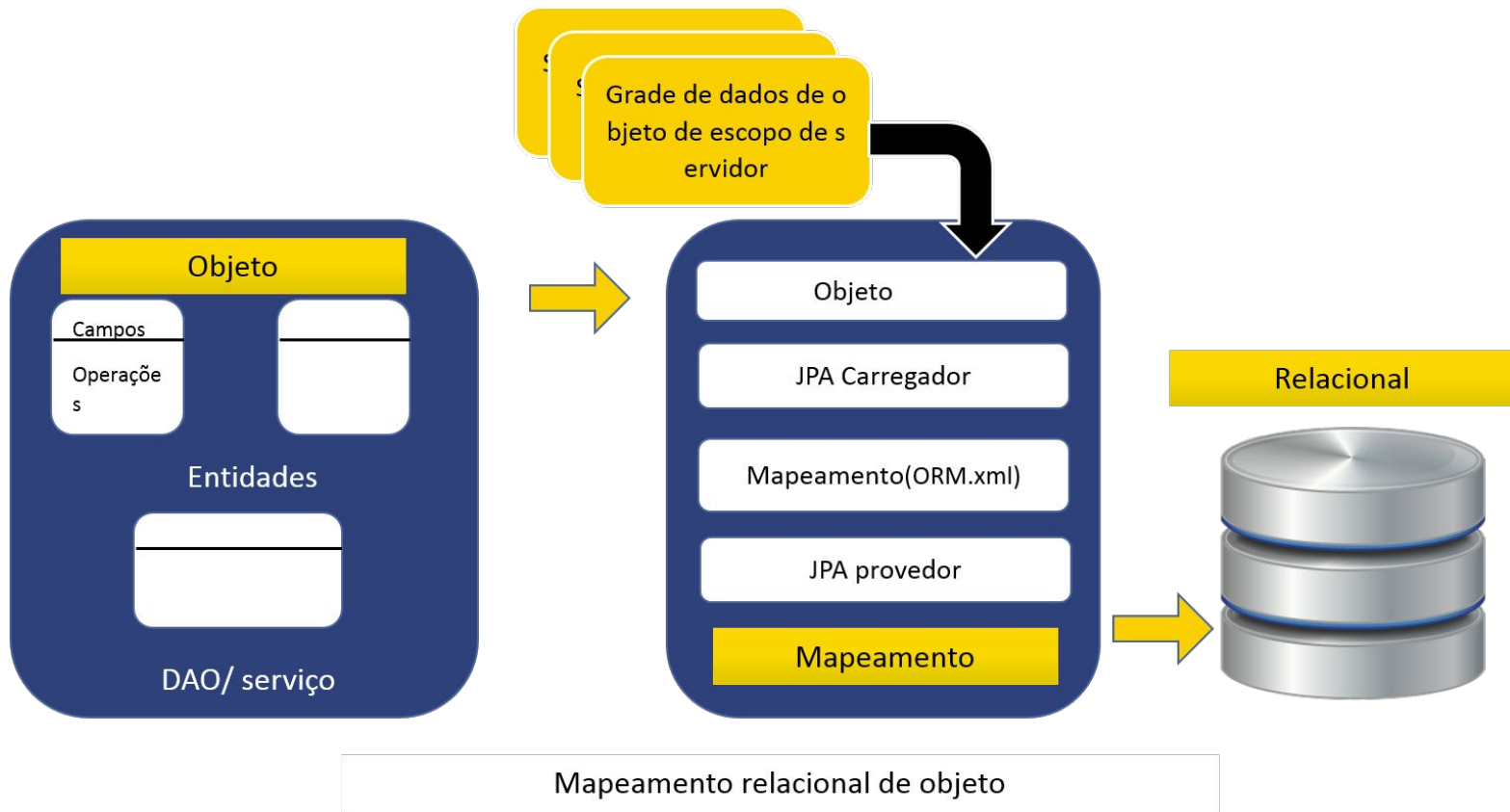
Integrando o BD no projeto java



ORM

- Mapeamento objeto-relacional (ou ORM, do inglês: Object-relational mapping) é uma técnica de desenvolvimento utilizada para reduzir a diferença da programação orientada aos objetos utilizando bancos de dados relacionais.
 - As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes.
 - Com esta técnica, o programador não precisa se preocupar com os comandos em linguagem SQL; ele irá usar uma interface de programação que faz todo o trabalho de persistência.

Arquitetura ORM



JPA

- Java Persistence API (ou simplesmente JPA) é uma API padrão da linguagem Java que descreve uma interface comum para frameworks de persistência de dados.
 - Define um meio de mapeamento objeto-relacional para objetos Java simples e comuns (POJOs), denominados beans de entidade.
 - Qualquer objeto com um construtor default, que não dependem da herança de interfaces ou classes de frameworks externos.
- A Java Persistence API, diferente do que muitos imaginam, não é apenas um framework para Mapeamento Objeto-Relacional (ORM - Object-Relational Mapping), ela também oferece diversas funcionalidades essenciais em qualquer aplicação corporativa.

Java Persistence API e Frameworks ORM

- O Hibernate é uma ferramenta ORM open source e é a líder de mercado, sendo a inspiração para a especificação Java Persistence API (JPA).
 - O Hibernate nasceu sem JPA mas hoje em dia é comum acessar o Hibernate pela especificação JPA.
 - Como toda especificação, ela deve possuir implementações. Entre as implementações mais comuns, podemos citar:
 - Hibernate da Red Hat, EclipseLink da Eclipse Foundation e o OpenJPA da Apache.
 - Apesar do Hibernate ter originado a JPA, o EclipseLink é a implementação referencial.

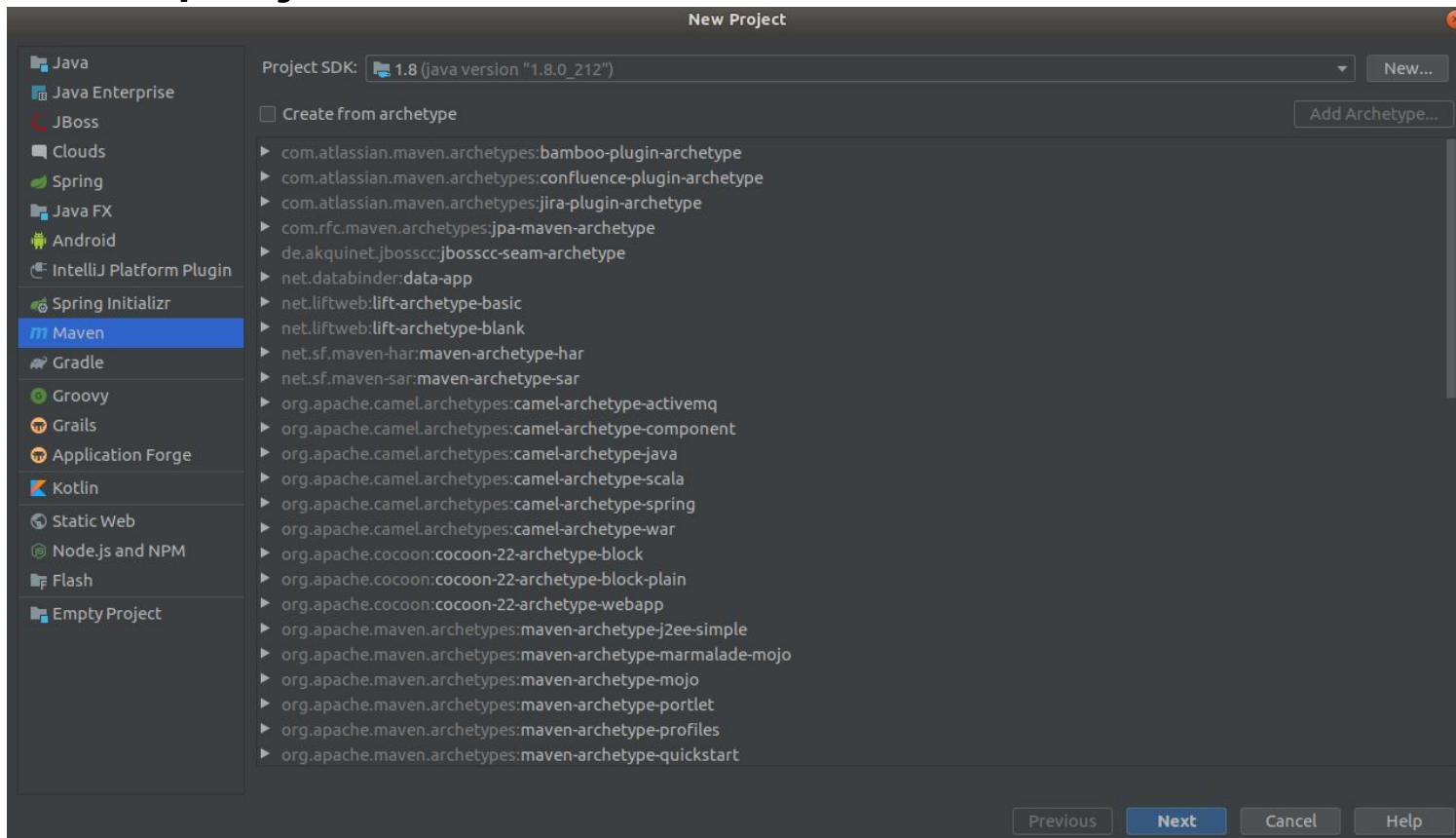
Java Persistence API e Frameworks ORM

- O Hibernate abstrai o seu código SQL, toda a camada JDBC e o SQL será gerado em tempo de execução.
- Mais que isso, ele vai gerar o SQL que serve para um determinado banco de dados, já que cada banco fala um "dialetto" diferente dessa linguagem.
 - Assim há também a possibilidade de trocar de banco de dados sem ter de alterar código Java, já que isso fica como responsabilidade da ferramenta.
- Como usaremos JPA abstraímos mais ainda, podemos desenvolver sem conhecer detalhes sobre o Hibernate e até trocar o Hibernate com uma outra implementação como OpenJPA:

Vamos criar?



Criando o projeto



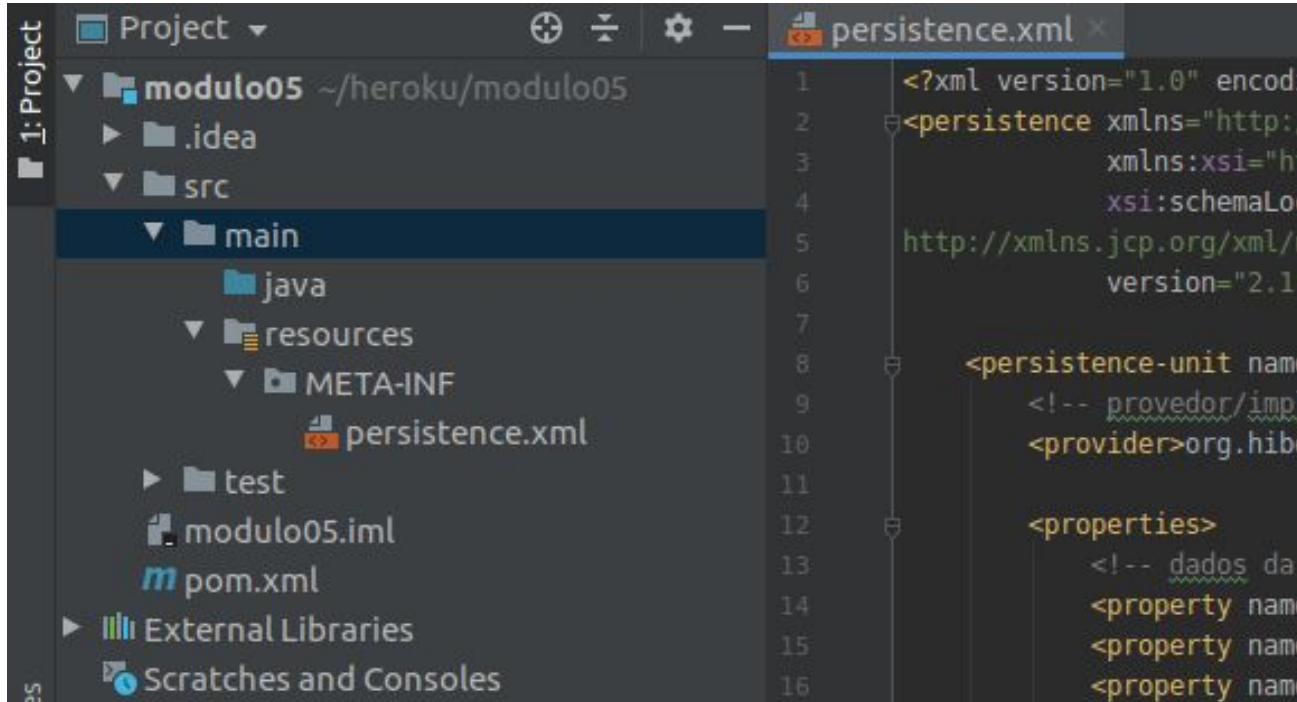
Add lib de Hibernate e Postgresql

<https://mvnrepository.com/>

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.12.Final</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.10</version>
  </dependency>
</dependencies>
```

Configuração do JPA com o BD



persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
    version="2.1">

    <persistence-unit name="Sales" transaction-type="RESOURCE_LOCAL">
        <!-- provedor/implementacao do JPA -->
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

        <properties>
            <!-- dados da conexao -->
            <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" /> <!-- DB Driver -->
            <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost/sales" /> <!-- BD Name -->
            <property name="javax.persistence.jdbc.user" value="postgres" /> <!-- DB User -->
            <property name="javax.persistence.jdbc.password" value="123" /> <!-- DB Password -->

            <!-- propriedades do hibernate -->
            <property name="hibernate.hbm2ddl.auto" value="update" /> <!-- create / create-drop / update -->

            <property name="hibernate.show_sql" value="true" /> <!-- Show SQL in console -->
            <property name="hibernate.format_sql" value="true" /> <!-- Show SQL formatted -->

        </properties>
```

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">

  <persistence-unit name="Sales" transaction-type="RESOURCE_LOCAL">
    <!-- provedor/implementacao do JPA -->
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
```

Configuração do postgres

```
    <properties>
    <!-- dados da conexao -->
    <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" /> <!-- DB Driver -->
    <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost/sales" /> <!-- BD Name -->
    <property name="javax.persistence.jdbc.user" value="postgres" /> <!-- DB User -->
    <property name="javax.persistence.jdbc.password" value="123" /> <!-- DB Password -->
```

```
    <!-- propriedades do hibernate -->
    <property name="hibernate.hbm2ddl.auto" value="update" /> <!-- create / create-drop / update -->
```

```
    <property name="hibernate.show_sql" value="true" /> <!-- Show SQL in console -->
    <property name="hibernate.format_sql" value="true" /> <!-- Show SQL formatted -->
```

```
</properties>
```


persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">

  <persistence-unit name="Sales" transaction-type="RESOURCE_LOCAL">
    <!-- provedor/implementacao do JPA -->
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
```

```
    <properties>
      <!-- dados da conexao -->
      <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" /> <!-- DB Driver -->
      <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost/sales" /> <!-- BD Name -->
      <property name="javax.persistence.jdbc.user" value="postgres" /> <!-- DB User -->
      <property name="javax.persistence.jdbc.password" value="123" /> <!-- DB Password -->
```

```
      <!-- propriedades do hibernate -->
      <property name="hibernate.hbm2ddl.auto" value="update" /> <!-- create / create-drop / update -->

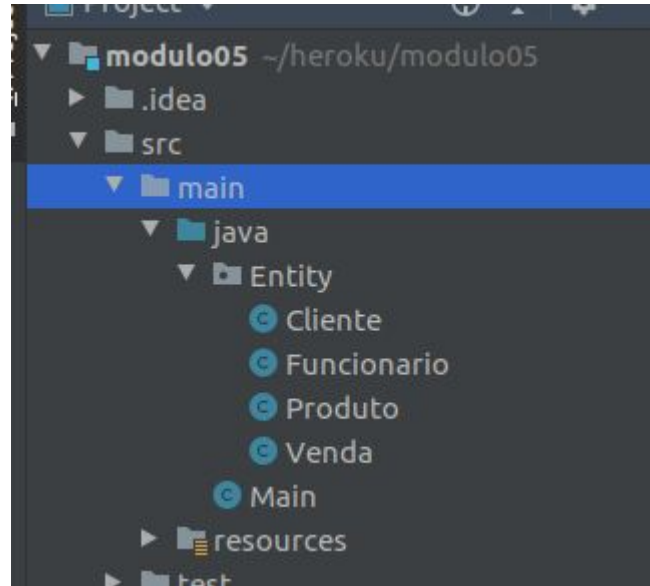
      <property name="hibernate.show_sql" value="true" /> <!-- Show SQL in console -->
      <property name="hibernate.format_sql" value="true" /> <!-- Show SQL formatted -->
```

```
</properties>
```

Configuração do Hibernate

- Iniciando BD e verificando atualizações
- Controla se o Hibernate deve exibir o SQL executado no console.
- Controlar a formatação.

Criando as entidades



Definindo os atributos

```
funcionario.java x venda.java x cliente.java x
1  import com.sun.istack.NotNull;
2  import javax.persistence.Column;
3  import javax.persistence.Entity;
4  import javax.persistence.Id;
5  import javax.persistence.Table;
6
7  @Entity
8  @Table(name = "funcionario")
9  public class funcionario {
10
11      @Id
12      private int matricula;
13      @Column
14      @NotNull
15      private String nome;
16      @Column
17      @NotNull
18      private String cpf;
19
20      public int getMatricula() {
21          return matricula;
22      }
23      public void setMatricula(int matricula) {
24          this.matricula = matricula;
25      }
26  }
```

Annotation

Anotação	Descrição
@Entity	Declara a classe como uma entidade ou uma tabela.
@Table	Declara nome da tabela.
@Basic	Especifica os campos não-restrição explícita.
@Embedded	Especifica as propriedades de classe ou de uma entidade cujo valor é uma instância de uma classe incorporável.
@Id	Especifica a propriedade, o uso de uma identidade (chave primária de uma tabela) da classe.
@GeneratedValue	Identidade Especifica como o atributo pode ser inicializado como automático, manual, ou o valor de uma sequência.
@Transient	Especifica a propriedade que não é persistente, ou seja, o valor nunca é armazenada no banco de dados.
@Coluna	Especifica o atributo de coluna para a persistência.

Annotation

@SequenceGenerator	Especifica o valor da propriedade que é especificado no @GeneratedValue anotação. Ele cria uma seqüência.
@TableGenerator	Especifica o valor gerador da propriedade especificada no @GeneratedValue anotação. Ele cria uma tabela para geração de valor.
@AccessType)	Este tipo de comentário é usado para definir o tipo de acesso. Se você definir @AccessType(CAMPO), então o acesso ocorre Domínio sábio. Se você definir @AccessType (PROPRIEDADE), então o acesso ocorre imóvel sábio.
@JoinColumn	Especifica uma entidade associação ou entidade coleção. Isso é usado em muitos - para um e um para muitas associações.
@UniqueConstraint	Especifica os campos e as únicas restrições para o primário ou o secundário.
@ColumnResult	As referências ao nome de uma coluna da consulta SQL utilizando cláusula select.

Annotation

@ManyToMany	Define um relacionamento muitos-para-muitos entre a juntar tabelas.
@ManyToOne	Define um muitos-para-um relacionamento entre a juntar tabelas.
@OneToMany	Define um relacionamento um-para-muitos entre a juntar tabelas.
@OneToOne	Define uma relação de um para um entre o juntar tabelas.
@NamedQueries	Especifica uma lista de consultas nomeadas.
@NamedQuery	Especifica uma consulta usando nome estático.

Construir a classe DAO responsável por fazer a comunicação com o banco de dados

Nossa classe `FuncionarioJpaDAO` segue o padrão de projeto Singleton que garante que apenas uma instância dessa classe será criada durante toda a aplicação.

Ao realizar a criação da classe pela primeira vez o método `getEntityManager()` é chamado, responsável por criar uma instância de `EntityManager`.

Construir a classe DAO responsável por fazer a comunicação com o banco de dados

A linha `Persistence.createEntityManagerFactory("Sales")` usa as configurações presentes no arquivo `persistence.xml` para criar uma instância de `EntityManagerFactory`.

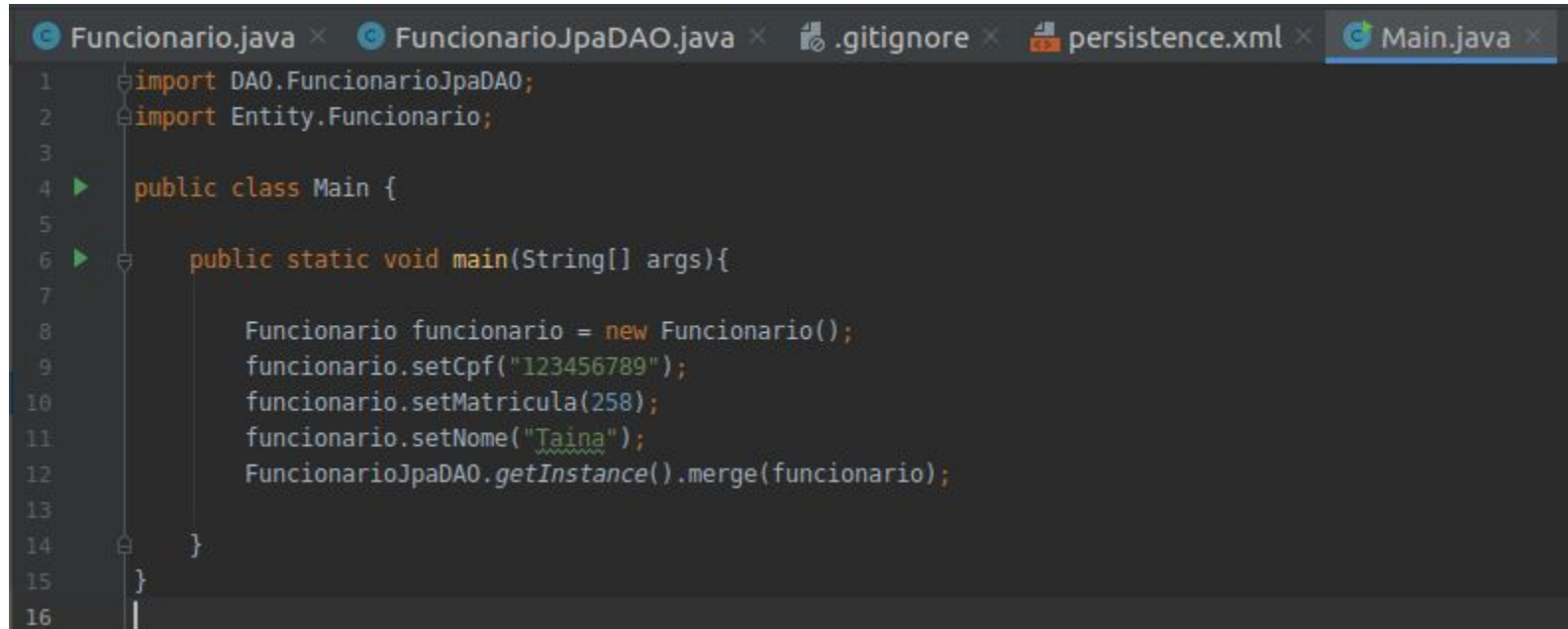
Depois disso verificamos se o atributo `entityManager` é nulo, ou seja, nunca foi criado, sendo assim usamos o `createEntityManager()` para criar uma instância de `EntityManager` que é responsável por realizar as operações de CRUD no banco de dados.

Construir a classe DAO responsável por fazer a comunicação com o banco de dados

Tudo gira em torno do EntityManager, este é o nosso objeto principal para o CRUD.

Feito isso e entendido para que precisamos dele, podemos começar a criar os métodos que usarão tão atributo.

Salvando o objeto no BD



The screenshot shows an IDE with five tabs: `Funcionario.java`, `FuncionarioJpaDAO.java`, `.gitignore`, `persistence.xml`, and `Main.java`. The `Main.java` tab is active, displaying the following code:

```
1  import DAO.FuncionarioJpaDAO;
2  import Entity.Funcionario;
3
4  public class Main {
5
6  public static void main(String[] args){
7
8      Funcionario funcionario = new Funcionario();
9      funcionario.setCpf("123456789");
10     funcionario.setMatricula(258);
11     funcionario.setNome("Taina");
12     FuncionarioJpaDAO.getInstance().merge(funcionario);
13
14 }
15 }
16
```



<https://github.com/tainajmedeiros/modulo5Codenation>