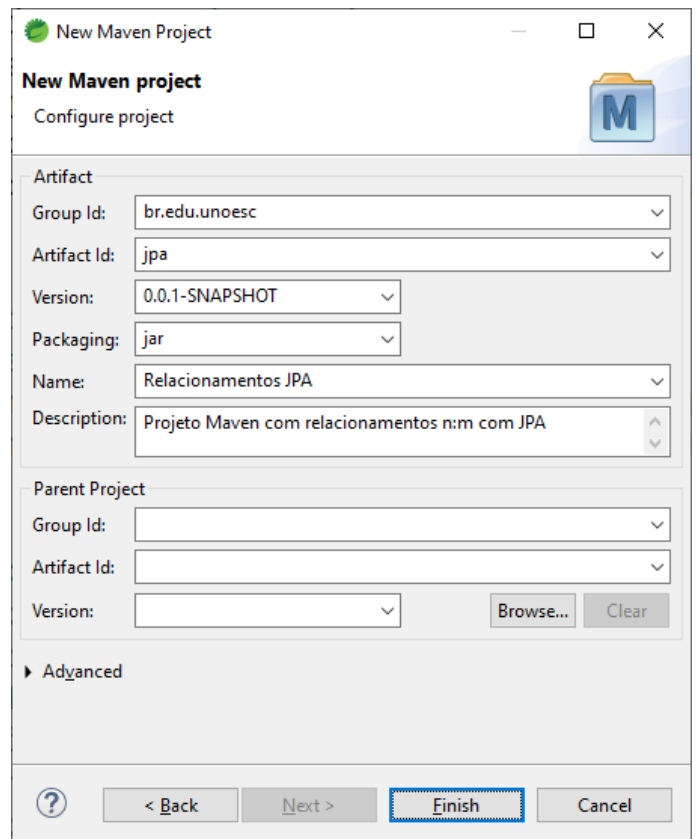
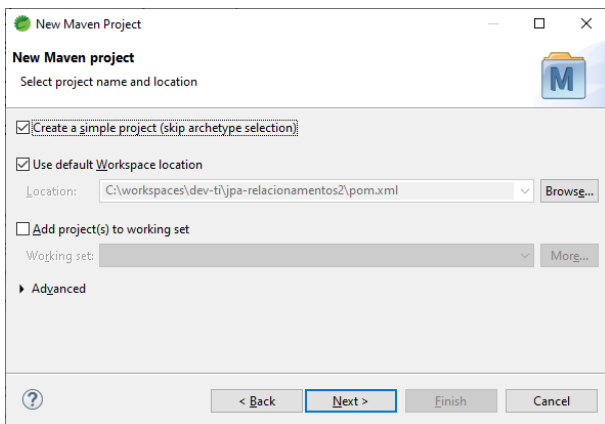


Relacionamentos $n:m$ em JPA

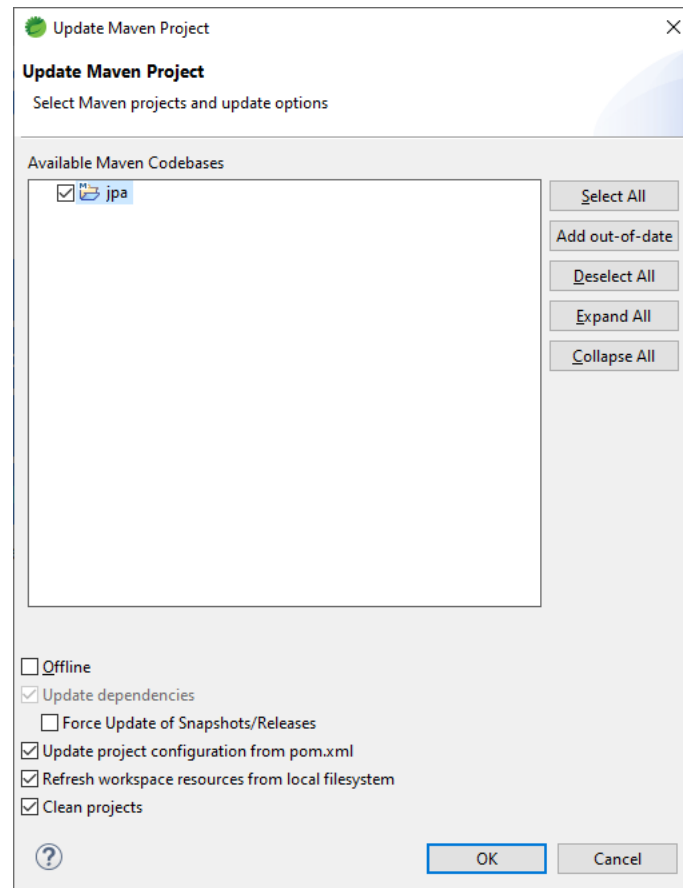
1. No STS crie um novo projeto do tipo Maven com as configurações abaixo:



2. Configure o arquivo *pom.xml* como mostrado a seguir:

```
1<project xmlns="http://maven.apache.org/POM/4.0.0"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4  <modelVersion>4.0.0</modelVersion>
5  <groupId>br.edu.unoesc</groupId>
6  <artifactId>jpa</artifactId>
7  <version>0.0.1-SNAPSHOT</version>
8  <name>Relacionamentos JPA</name>
9  <description>Projeto Maven com relacionamentos n:m com JPA</description>
10
11  <properties>
12    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
13    <java.version>17</java.version>
14    <maven.compiler.source>${java.version}</maven.compiler.source>
15    <maven.compiler.target>${java.version}</maven.compiler.target>
16  </properties>
17
18  <dependencies>
19    <dependency>
20      <groupId>org.hibernate</groupId>
21      <artifactId>hibernate-core</artifactId>
22      <version>6.1.2.Final</version>
23    </dependency>
24    <dependency>
25      <groupId>mysql</groupId>
26      <artifactId>mysql-connector-java</artifactId>
27      <version>8.0.29</version>
28    </dependency>
29  </dependencies>
30</project>
```

3. Clique com o botão direito sobre o projeto e escolha *Maven → Update Project ...*



Confira se as dependências foram baixadas e a versão do Java atualizada para a 17.

- ▼ jpa
 - > src/main/java
 - > src/main/resources
 - > src/test/java
 - > src/test/resources
 - > JRE System Library [JavaSE-17]
 - ▼ Maven Dependencies
 - > hibernate-core-6.1.2.Final.jar - C:\Users\Her
 - > jakarta.persistence-api-3.0.0.jar - C:\Users\H
 - > jakarta.transaction-api-2.0.0.jar - C:\Users\H
 - > jboss-logging-3.4.3.Final.jar - C:\Users\Herc
 - > hibernate-commons-annotations-6.0.2.Final
 - > jandex-2.4.2.Final.jar - C:\Users\Herculano [
 - > classmate-1.5.1.jar - C:\Users\Herculano De
 - > byte-buddy-1.12.9.jar - C:\Users\Herculano
 - > jakarta.xml.bind-api-3.0.1.jar - C:\Users\Herc
 - > jakarta.activation-2.0.1.jar - C:\Users\Hercul
 - > jaxb-runtime-3.0.2.jar - C:\Users\Herculano
 - > jaxb-core-3.0.2.jar - C:\Users\Herculano De
 - > txw2-3.0.2.jar - C:\Users\Herculano De Biasi
 - > istack-commons-runtime-4.0.1.jar - C:\Users
 - > jakarta.inject-api-2.0.0.jar - C:\Users\Hercul
 - > antlr4-runtime-4.10.jar - C:\Users\Herculanc
 - > mysql-connector-java-8.0.29.jar - C:\Users\H
 - > protobuf-java-3.19.4.jar - C:\Users\Herculano
 - > src
 - > target
 - pom.xml

4. Configure o arquivo *persistence.xml* como mostrado a seguir. Verifique no material ou em projetos anteriores em que pasta este arquivo deve ser criado.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <persistence version="3.0"
3   xmlns="https://jakarta.ee/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">
6
7   <!-- unidade de persistencia com o nome unoesc -->
8   <persistence-unit name="devti_jpa" transaction-type="RESOURCE_LOCAL">
9     <!-- Implementação do JPA, no nosso caso Hibernate -->
10    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
11
12    <!-- Aqui são listadas todas as entidades
13    <class>br.edu.unoesc.financas.modelo.Conta</class> -->
14
15    <properties>
16      <!-- Propriedades JDBC -->
17      <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver" />
18      <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://localhost/devti_jpa?createDatabaseIfNotExist=true" />
19      <property name="jakarta.persistence.jdbc.user" value="root" />
20      <property name="jakarta.persistence.jdbc.password" value="root" />
21
22      <!-- Configurações específicas do Hibernate -->
23      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
24      <property name="hibernate.hbm2ddl.auto" value="create" />
25      <property name="hibernate.show_sql" value="true" />
26      <property name="hibernate.format_sql" value="true" />
27    </properties>
28  </persistence-unit>
29 </persistence>
```

5. Dentro do pacote `br.edu.unoesc.jpa.util` crie a classe `JPAUtil` com o seguinte conteúdo:

```
1 package br.edu.unoesc.jpa.util;
2
3 import jakarta.persistence.EntityManager;
4 import jakarta.persistence.EntityManagerFactory;
5 import jakarta.persistence.Persistence;
6
7 public class JPAUtil {
8   private static final String UNIDADE_PERSISTENCIA = "devti_jpa";
9   private static final EntityManagerFactory EMF;
10
11   static {
12     try {
13       EMF = Persistence.createEntityManagerFactory(UNIDADE_PERSISTENCIA);
14     } catch (Exception e) {
15       throw new RuntimeException(e);
16     }
17   }
18
19   private JPAUtil() {}
20
21   public static EntityManager getEntityManager() {
22     return EMF.createEntityManager();
23   }
24
25   public static void closeEntityManagerFactory() {
26     EMF.close();
27   }
28 }
```

6. Dentro do pacote `br.edu.unoesc.jpa.modelo` crie a classe `Funcionario` obedecendo o seguinte:

- Transforme-a em uma entidade JPA com a anotação `@Entity`.
- Use a anotação `@Table` para configurar que o nome da tabela será `funcionarios`.
- Implemente a *interface* `Serializable` e crie o UID padrão.
- Crie um atributo do tipo `Integer` chamado `codigo` e defina-o como um identificador do JPA utilizando a anotação correta.
- Defina que o valor do atributo `codigo` será gerado de forma automática com estratégia `IDENTITY`.
- Crie um atributo do tipo `String` chamado `nome` e use a anotação `@Column` para definir o tamanho máximo de 50 caracteres e não aceitar que o campo seja nulo.
- Defina um atributo chamado `dataNascimento` do tipo `LocalDate`. Use a anotação `@Column` para informar que o campo na tabela se chamará `data_nascimento` e não poderá ser nulo.
- Defina um atributo chamado `salario` do tipo `BigDecimal`. Use a anotação `@Column` para informar que o tamanho total será 12 dígitos com 2 casas decimais e que não poderá ser nulo.
- Crie um construtor padrão (vazio).
- Crie um construtor com parâmetros. Os parâmetros deverão ser: `nome`, `dataNascimento` e `salario`.

7. Dentro do pacote `br.edu.unoesc.jpa.modelo` crie a classe `Projeto` obedecendo o seguinte:
- Transforme-a em uma entidade JPA com a anotação `@Entity`.
 - Use a anotação `@Table` para configurar que o nome da tabela será `projetos`.
 - Implemente a *interface* `Serializable` e crie o UID padrão.
 - Crie um atributo do tipo `Integer` chamado `codigo` e defina-o como um identificador do JPA utilizando a anotação correta.
 - Defina que o valor do atributo `codigo` será gerado de forma automática com estratégia `IDENTITY`.
 - Crie um atributo do tipo `String` chamado `nome` e use a anotação `@Column` para definir o tamanho máximo de 50 caracteres e não aceitar que o campo seja nulo.
 - Crie um construtor padrão (vazio).
 - Crie um construtor que receba somente o parâmetro `nome`.
8. Dentro do pacote `br.edu.unoesc.jpa.app` crie a classe `App` com o seguinte:
- Uma variável pública estática chamada `em` do tipo `EntityManager`.
 - 3 métodos privados e estáticos com retorno `void`, chamados `adicionarDados()`, `listarPorProjeto()` e `listarPorFuncionario()`.
 - A função `main()` tradicional do Java.
 - Chame esses métodos acima de dentro da função `main()`.

9. A função `adicionarDados()` deverá iniciar obtendo um gerenciador de entidades e finalizar fechando-o:

```
private static void adicionarDados() {
    em = JPAUtil.getEntityManager();

    em.close();
}
```

10. Execute o programa, neste ponto o banco de dados `devti_jpa`, assim como as tabelas `funcionarios` e `projetos` devem ser criadas:

Hibernate:

```
create table funcionarios (
    codigo integer not null auto_increment,
    data_nascimento date not null,
    nome varchar(50) not null,
    salario decimal(12,2) not null,
    primary key (codigo)
) engine=InnoDB
```

```
set. 22, 2022 3:51:37 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator
$ConnectionProviderJdbcConnectionAccess@74024f3] for (non-JTA) DDL execution was not in auto-commit mode; the Connection 'local transaction'
will be committed and the Connection will be set into auto-commit mode.
```

Hibernate:

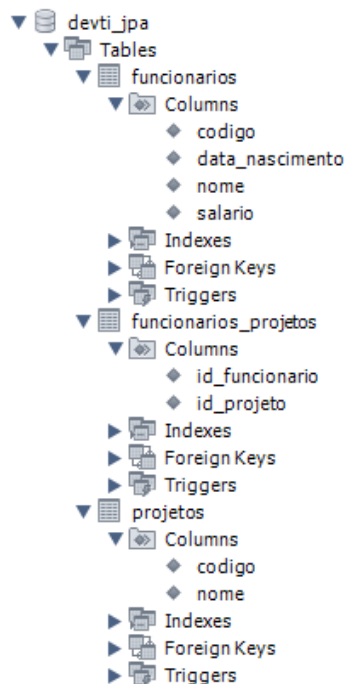
```
create table projetos (
    codigo integer not null auto_increment,
    nome varchar(50) not null,
    primary key (codigo)
) engine=InnoDB
```

```
set. 22, 2022 3:51:37 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
```

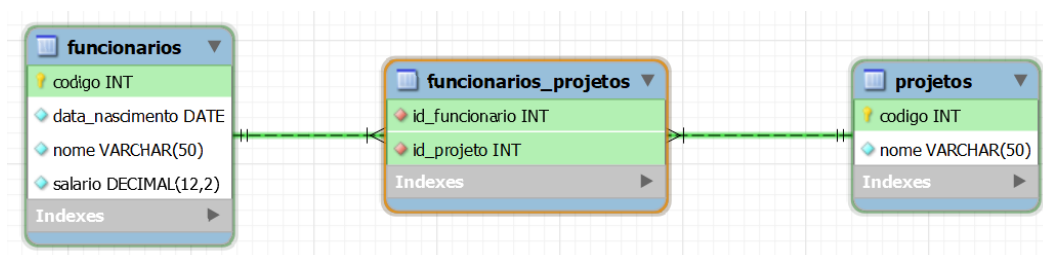
11. Para iniciar a implementação do relacionamento **bidirecional** ($m:n$) entre funcionários e projetos, proceda com as seguintes modificações na classe `Funcionario`:

- Crie uma objeto chamado `projetos` do tipo `List<Projeto>`.
- Anote este atributo com `@ManyToMany` informando também a opção `ALL` como tipo estratégia de aplicações de modificações em cascata e `EAGER` como estratégia de busca de dados.
- Anote este mesmo atributo com `@JoinTable` informando `id_funcionario` no parâmetro `joinColumns` (use a anotação `@JoinColumn` para isso). Informe `id_projeto` no parâmetro `inverseJoinColumns` usando para isso novamente a anotação `@JoinColumn`.
- Adicione de forma automatizada o método `getProjetos()`. Modifique este método de forma que, antes dele retornar o valor do atributo `projetos`, ele teste se o atributo é nulo. Em caso afirmativo, o método deve inicializar o atributo `projetos` com uma lista vazia.
- Adicione automaticamente o método `toString()` mas desmarque o atributo `projetos` a fim de evitar problemas com chamadas recursivas mais adiante.

12. Para continuar a implementação do relacionamento **bidirecional** ($m:n$) entre funcionários e projetos, proceda com as seguintes modificações na classe `Projeto`:
- Crie um objeto chamado `funcionarios` do tipo `List<Funcionario>`.
 - Anote este atributo com `@ManyToMany` informando também a opção `ALL` como tipo estratégia de aplicações de modificações em cascata e `EAGER` como estratégia de busca de dados. Utilize também o parâmetro `mappedBy` e informe nele o valor "projetos" a fim de fazer o mapeamento reverso.
 - Adicione de forma automatizada o método `getFuncionarios()`. Modifique este método de forma que, antes dele retornar o valor do atributo `funcionarios`, ele teste se o atributo é nulo. Em caso afirmativo, o método deve inicializar o atributo `funcionarios` com uma lista vazia.
 - Adicione automaticamente o método `toString()` mas desmarque o atributo `funcionarios` a fim de evitar problemas com chamadas recursivas mais adiante.
13. Com essas modificações feitas, o JPA deve ter criado agora uma terceira tabela, *funcionarios_projetos*, cuja função é relacionar os dados das outras duas.



Tente utilizar o recurso de engenharia reversa do MySQL Workbench para gerar o diagrama abaixo.



14. Para encerrar a amarração entre as entidades, implemente, na classe `Funcionario`, o método abaixo:

```
public void adicionarProjeto(Projeto projeto) {
    if (projeto != null && !this.getProjetos().contains(projeto)) {
        this.projetos.add(projeto);

        if (!projeto.getFuncionarios().contains(this)) {
            projeto.getFuncionarios().add(this);
        }
    }
}
```

15. Na classe `Projeto` implemente o método `adicionarFuncionario()`, semelhante ao da questão anterior, mas que adicione um funcionário passado por parâmetro na lista de funcionários do projeto atual. Ele também deve adicionar o projeto atual na lista de projetos do funcionário.
16. Na classe `App`, no método `adicionarDados()`, faça o seguinte:
- Crie dois projetos: 'Projeto Manhattan' e 'Projeto Genoma Humano'.
 - Crie 3 funcionários: Fulano, Beltrano e Sicrano.
 - Utilize o método `adicionarFuncionario()` para acrescentar Fulano e Beltrano ao Projeto Manhattan.
 - Utilize o método `adicionarProjeto()` para acrescentar o Projeto Genoma Humano aos funcionários Fulano e Sicrano.
 - Abra uma transação e persista os 3 funcionários. Isso deve ser suficiente para salvar também os projetos nas tabelas, assim como quais funcionários estão alocados a quais projetos.
17. Na classe `App`, no método `listarPorFuncionario()`, faça o seguinte:
- Crie uma consulta em JPQL simples que selecione todos os funcionários.
 - Percorra a lista de resultados com um `for()` e imprima o código e nome do funcionário.
 - Dentro deste `for()`, para cada funcionário, execute outro `for()` que percorra a lista de projetos deste (utilize para isso o método `getProjetos()`).
 - Imprima duas tabulações e a seguir o código e nome do projeto.

A saída deverá se parecer com a seguinte:

```
1 - Fulano
    1 - Projeto Manhattan
    2 - Projeto Genoma Humano

2 - Beltrano
    1 - Projeto Manhattan

3 - Sicrano
    2 - Projeto Genoma Humano
```

18. Na classe `App`, no método `listarPorProjeto()`, faça o seguinte:
- Crie uma consulta em JPQL simples que selecione todos os funcionários.
 - Percorra a lista de resultados com um `for()` e imprima o código e nome do projeto.
 - Dentro deste `for()`, para cada projeto, execute outro `for()` que percorra a lista de funcionários deste (utilize para isso o método `getFuncionarios()`).
 - Imprima duas tabulações e a seguir o código e nome do funcionário.

A saída deverá se parecer com a seguinte:

```
1 - Projeto Manhattan
    1 - Fulano
    2 - Beltrano

2 - Projeto Genoma Humano
    1 - Fulano
    3 - Sicrano
```