* Primeiramente popular as dependências do hibernate e JPA do projeto
* Configurar um arquivo persistence com todas as configurações que precisamos dentro de uma unidade de persistência para acessarmos o banco
* Criar a class Aluno com os atributos e seus tipos definidos com a anotação @Entity para o hibernate mapear a class como uma tabela dentro do banco de dados juntamente com um id anotado com as anotações @Id e @GeneratedValue(strategy = GenerationType.***IDENTITY***) para que esse id seja um identificador da tabela uma PK.
* Utilizar **EntityManager** que nos permitirá fazer todas as operações do banco.
* Criar a class Avaliações com os atributos e seus tipos definidos com a anotação @Entity para o hibernate mapear a class como uma tabela dentro do banco de dados juntamente com um id anotado com as anotações @Id e @GeneratedValue(strategy = GenerationType.***IDENTITY***) para que esse id seja um identificador da tabela uma PK.
* Criar a class de avaliações cognitive walkthrough com o atributo resposta com a anotação @Entity para o hibernate mapear a class como uma tabela dentro do banco de dados juntamente com um id anotado com as anotações @Id e @GeneratedValue(strategy = GenerationType.***IDENTITY***) para que esse id seja um identificador da tabela uma PK.
* Adicionar os relacionamentos com as tabelas de avaliações e aluno com a anotação @OneToMany para aluno e @OneToOne para avaliações.
* Criar a class de avaliação pessoal com o atributo nota pessoal com a anotação @Entity para o hibernate mapear a class como uma tabela dentro do banco de dados juntamente com um id anotado com as anotações @Id e @GeneratedValue(strategy = GenerationType.***IDENTITY***) para que esse id seja um identificador da tabela uma PK.
* Adicionar o relacionamentos com as tabela de avaliações cognitive walkthrough com a anotação e @OneToOne
* Criar um novo aluno realizando os sets dos atributos de aluno.
* Usar EntityManager com método createEntityManager do EntityManagerFactory para criar um objeto EntityManager
* Com o objeto EntityManager usar o persist() passando o objeto aluno que queremos persistir no banco de dados.
* Abrir uma transação com getTransaction() e iniciar a transação chamando o método **begin()**. Depois, fazer o **commit()** para salvar um aluno.
* Criar uma nova avaliação realizando os sets dos atributos de avaliação.
* Usar EntityManager com método createEntityManager do EntityManagerFactory para criar um objeto EntityManager
* Com o objeto EntityManager usar o persist() passando o objeto avaliaçao que queremos persistir no banco de dados.
* Abrir uma transação com getTransaction() e iniciar a transação chamando o método **begin()**. Depois, fazer o **commit()** para salvar uma avaliação.
* Criar uma nova resposta realizando os sets dos atributos de resposta.
* Passar o id de aluno referente da resposta e o id da avaliação referente a resposta
* Usar EntityManager com método createEntityManager do EntityManagerFactory para criar um objeto EntityManager
* Com o objeto EntityManager usar o persist() passando o objeto resposta que queremos persistir no banco de dados.
* Abrir uma transação com getTransaction() e iniciar a transação chamando o método **begin()**. Depois, fazer o **commit()** para salvar uma resposta.
* Criar uma nova auto correção realizando os sets dos atributos de aluno.
* Passar o id da avaliação referente a essa auto correção.
* Usar EntityManager com método createEntityManager do EntityManagerFactory para criar um objeto EntityManager
* Com o objeto EntityManager usar o persist() passando o objeto auto correção que queremos persistir no banco de dados.
* Abrir uma transação com getTransaction() e iniciar a transação chamando o método **begin()**. Depois, fazer o **commit()** para salvar uma auto correção.
* Utilizaria JPQL para realizar a query de busca do nome do aluno, usar : para referenciar o objeto dentro da query, informar o **TypedQuery com a class que query ira receber .**
* Utilizaria JPQL para realizar a query de busca das respostas de um aluno, usar : para referenciar o objeto dentro da query para trazer uma lista de resposta, informar o **TypedQuery com a class que query ira receber .**



**Java e JPA: Persista seus objetos com a JPA2 e Hibernate**

|  |
| --- |
| **Cenário:** |
| Somos muitos alunos e alunas aqui no Bootcamp. Em função de todas as avaliações respondidas, muitos dados são gerados. E agora temos alguns desafios para você:   * Dado que todo(a) aluno(a) tem um email (máximo de 30 caracteres),nome (máximo de 30 caracteres) e idade (entre 1 e 100). Como você vai modelar essa classe e configurá-la para que possa ser utilizada pelo Hibernate? * Dentro do bootcamp temos também um conjunto de avaliações que são respondidas pelas pessoas. Toda avaliação tem um título e uma descrição do que precisa ser feito. Como você vai modelar essa classe e configurá-la para que possa ser utilizada pelo Hibernate? * Cada aluno(a) responde uma ou mais avaliações que chamamos de cognitive walkthrough, ela tem que descrever os passos da solução dela para determinada situação problema. Toda resposta tem um campo aberto para que a pessoa consiga descrever a solução dela. É necessário que toda resposta seja linkada com a pessoa que a respondeu e também com a avaliação relativa àquela resposta. Como você vai modelar essa classe e configurá-la para que possa ser utilizada pelo Hibernate? * Além de responder sua avaliação, a pessoa também responde um outro formulário onde ela corrige sua resposta em função da resposta de um mentor(a). Essa correção sempre tem uma nota de 1 a 10 e está linkada com a avaliação respondida pela própria pessoa. Como você vai modelar essa classe e configurá-la para que possa ser utilizada pelo Hibernate? * Agora que você modelou e mapeou as classes para que possam ser utilizadas pelo hibernate, temos desafios extras. * Quais são os passos para salvar para salvar um(a) aluno(a)? * Quais são os passos para salvar para salvar uma avaliação? * Quais são os passos para salvar uma resposta de um(a) aluno(a)? * Quais são os passos para salvar a auto correção de um(a) aluno(a)? * Caso você precise carregar uma auto correção e tenha que descobrir o nome do(a) aluno(a) que fez, como você faria? * Caso você carregar as respostas de um(a) aluno(a) a partir de um objeto do tipo Aluno, qual query você executaria? |

|  |
| --- |
| **O que seria bom ver nessa resposta?** |

* **Peso 0.25:**Mapeamento simples para as classes Aluno e Avaliação
* **Peso 1:**Mapeamento como @ManyToOne para as classes Resposta e AutoCorrecao
* **Peso 0.25:**Criação de EntityManagerFactory, EntityManager e chamada do persist para salvar um objeto do tipo Aluno
* **Peso 0.25:**Criação/Reaproveitamento de EntityManagerFactory, EntityManager e chamada do persist para salvar um objeto do tipo Avaliação
* **Peso 2:**Criação/Reaproveitamento de EntityManagerFactory, EntityManager, carregamento do aluno, da avaliação e posterior persist da resposta
* **Peso 2:**Criação/Reaproveitamento de EntityManagerFactory, EntityManager, carregamento da resposta e posterior persist da auto correção.
* **Peso 2.25:**Carregamento da auto correção e navegação até o nome do(a) aluno(a) tomando cuidado com o lazy load
* **Peso 2:**Carregamento de todas respostas de um(a) aluno(a) via query a partir de um(a) determinado(a) aluno(a).

|  |
| --- |
| **Resposta do Especialista:** |
| * Crio a classe Aluno, adiciono os atributos necessários e também adiciono um id. Adiciono o Id para mapear como chave primária para o banco de dados. * Mapeio a classe Aluno para ser entendida pelo HIbernate. Uso o @Entity em cima da classe e o @Id em cima do atributo id. Poderia colocar também o @Id no email. A @Entity explica para o Hibernate que aquela classe vai ser uma tabela no banco e o @Id explica que aquele atributo vai ser a chave primaria na tabela. Os outros atributos eu não preciso mapear pq ele já vai mapear para colunas do mesmo nome. * Crio a classe Avaliacao, adiciono os atributos necessários e também adiciono um id. * Mapeio a classe Avaliacao para ser entendida pelo HIbernate. Uso o @Entity em cima da classe e o @Id em cima do atributo id. * Crio a classe Respota, adiciono os atributos necessários. Um dos atributos aqui é do tipo Avaliação para fazer o link, além do atributo do tipo Aluno. Além disso adiciono o id. Uso o atributo do tipo Avaliacao e Aluno para manter o domínio rico e tirar proveito do ORM. * Mapeio a classe Resposta para ser entendida pelo HIbernate. Uso o @Entity em cima da classe e o @Id em cima do atributo id. Além disso preciso usar o @ManyToOne em cima do atributo que representa a Avaliacao, assim como no atributo Aluno. Dessa forma o Hibernate vai entender que tal atributo precisa ser mapeado como chave estrangeira na tabela. * Crio a classe AutoAvaliacao e adiciono os atributos necessários. Um dos atributos é do tipo Resposta, já que a autoavaliação é relativa a resposta que o(a) próprio(a) aluno(a) deu.  Além disso adiciono o id  como atributo extra para ser chave primária. * Mapeio a classe AutoAvaliacao para ser entendida pelo HIbernate. Uso o @Entity em cima da classe e o @Id em cima do atributo id. Além disso preciso usar o @ManyToOne em cima do atributo que representa a Resposta. Dessa forma o Hibernate vai entender que tal atributo precisa ser mapeado como chave estrangeira na tabela. * Para salvar um aluno eu crio um EntityManagerFactory, construo um EntityManager, crio o objeto do tipo Aluno, abro uma transação, invoco o método persist e faço um commit. Preciso da EMF porque ela é a fábrica de EM. O EM representa a ideia de conexão com o banco de dados. * Para salvar uma avaliacao eu crio um EntityManagerFactory, construo um EntityManager, crio o objeto do tipo Avaliacao, abro uma transação, invoco o método persist e faço um commit. Preciso da EMF porque ela é a fábrica de EM. O EM representa a ideia de conexão com o banco de dados. * Para salvar uma resposta eu crio um EntityManagerFactory, construo um EntityManager, carrego o objeto do tipo Avaliacao, carrego o objeto do tipo Aluno, crio o objeto do tipo Reposta, abro uma transação, invoco o método persist e faço um commit. Preciso da EMF porque ela é a fábrica de EM. O EM representa a ideia de conexão com o banco de dados. Preciso carregar os objetos do tipo Aluno e Avaliacao para construir o vínculo com a Resposta. * Para salvar uma auto correcao eu crio um EntityManagerFactory, construo um EntityManager, carrego o objeto do tipo Resposta, crio o objeto do tipo AutoCorrecao, abro uma transação, invoco o método persist e faço um commit. Preciso da EMF porque ela é a fábrica de EM. O EM representa a ideia de conexão com o banco de dados. Preciso carregar o objeto do tipo Resposta para fazer o vínculo com a AutoCorrecao. * Para descobrir o nome do(a) aluno(a) a partir da AutoCorrecao eu preciso navegar pelos relacionamentos @ManyToOne até chegar onde preciso. Isso pode ser feito através de métodos getters ou posso criar um método na AutoCorrecao que já me retorne o aluno em questão. * Para descobrir as respostas do aluno a partir do objeto do tipo Aluno é necessário fazer uma query planejada usando a JPQL ou navegar por relacionamentos. A ideia é realizar uma query sobre os objetos do tipo Resposta buscando todas que sejam daquele determinado(a) aluno(a). Algo como select r from Resposta r where r.aluno =:aluno . |