LU10b - Objekte vs. Fremdschlüssel

Grundsätzlich kann man bei den JPA-Klassen entweder den Fremdschlüssel oder direkt das Zielobjekt selbst als Feld verwenden.

Datenbankorientiertes Modell	Objektorientiertes Modell

`@Entity`
`@Getter`
`@Setter`
`public class Account {`
`  ...`
`  @Column(name = "project_id", nullable = false)`
`  private String projectId;`
`  ...`
`}`
	`@Entity`
	`@Getter`
	`@Setter`
	`public class Account {`
	`  ...`
	`  @ManyToOne`
	`  @JoinColumn(name = "project_id", nullable = false)`
	`  private Project project;`
	`  ...`
	`}`

`String projectName = projectRepository.findById(account.getProjectId()).getProjectName();`	`String projectName = account.getProject().getProjectName();`

Der „objektorientierte“ Ansatz ermöglicht es im Code von einer Entität eine andere aufzurufen, ohne, dass diese jedes Mal manuell geladen werden müssen. Das erleichtert die Schreibweise als auch die Abhängigkeiten, da man im obigen Beispiel zusätzlich noch Zugriff auf `projectRepository` benötigt.

Besonders bei längeren Ketten von Entitäten, wird der Code bei der „datenbankorientierten“ Variante aufgrund der manuellen „findById“-Methoden schnell komplex und unübersichtlich.

Lazy Loading

Beim „objektorientierten“ Ansatz wird ein sogenanntes `Lazy Loading` angewendet. Das heisst, es werden nicht alle verknüpften Objekte zu Beginn miteinander geladen, sondern nur die tatsächlich benötigten Objekte nach und nach.

```
Account account = accountRepository.findById(accountId).get(); // Holt das Account-Objekt inklusive dem FK project_id
String projectName = account.getProject().getProjectName(); // Kein nachladen nötig, weil project_name dem Fremdschlüssel entspricht
String passwordHash = account.getProject().getPasswordHash(); // Hier wird im Hintergrund automatisch das Project-Objekt geladen
```

References

Um ein Objekt mittels einem Objekt zu suchen oder zu löschen, muss nicht das zwingend das ganze Objekte erstellt werden, sondern lediglich die Werte der Fremdschlüssel.

Operation	Via Join	Via Reference-Object
find	`List<Account> accounts = accountRepository.findByProject_ProjectName(projectName);`	`Project projectRef = entityManager.getReference(Project.class, projectName);` `List<Account> accounts = accountRepository.findByProject(project);`
delete	`accountRepository.deleteByAccountNumberAndProject_ProjectName(accountNumber, projectName);`	`Project projectRef = entityManager.getReference(Project.class, projectName);` `accountRepository.deleteByAccountNumberAndProject(accountNumber, projectRef);`

Sofern man nicht eh schon ein Project-Object beziehungsweise ein Reference-Objekt hat, ist die „Project_ProjectName“-Variante empfohlen. Hat man aber bereits ein solches Objekt, kann man dieses direkt mitgeben.

Die Reference-Objekte sind insbesondere praktisch, wenn neue Entitäten erstellt werden. Dies wird in folgendem Beispiel gezeigt.

```
Project projectRef = entityManager.getReference(Project.class, projectName);

Account account = accountRepository
        .findByAccountNumberAndProject(accountNumber, projectRef) //
Reference-Objekt kann benutzt werden, weil es eh bereits existiert.
        .orElse(new Account());
account.setAccountNumber(accountNumber);
account.setName(accountName);
account.setProject(projectRef); // Hier kann ein Reference-Objekt mitgegeben
werden, ohne dass das Project-Objekt aus der DB geladen werden muss.
accountRepository.save(account);
```

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/de/modul/ffit/3-jahr/java/learningunits/lu10/b?rev=1762734536

Last update: **2025/11/10 01:28**