07. JPA

## Inleiding

Java Persistence API (JPA) is een Java-specificatie voor ORM en biedt een abstractielaag bovenop JDBC. Het doel van JPA is om het gemakkelijker te maken om Java-applicaties te schrijven die gebruikmaken van een relationele database. Met behulp van JPA kunnen Java-ontwikkelaars database-entiteiten definiëren als Java-klassen en deze entiteiten op een objectgeoriënteerde manier gebruiken in hun code.

## Entities

Entities zijn Java-klassen die overeenkomen met tabellen in een database. Elke instantie van een entity klasse vertegenwoordigt een rij in de bijbehorende tabel. Entities worden meestal gedefinieerd met behulp van annotaties, zoals @Entity, @Table en @Column.

## Relationships

Relaties worden gebruikt om de verbanden tussen entities te definiëren. Er zijn verschillende soorten relaties, zoals One-to-One, One-to-Many en Many-to-Many. Relaties worden gedefinieerd met behulp van annotaties, zoals @OneToOne, @OneToMany en @ManyToMany.

## Queries

Queries worden gebruikt om gegevens op te halen uit de database. Met JPA kunnen ontwikkelaars queries opstellen met behulp van de Java Persistence Query Language (JPQL), een op SQL-gebaseerde querytaal voor JPA. JPQL is vergelijkbaar met SQL, maar maakt gebruik van entity-namen in plaats van tabelnamen.

## Transacties

Transacties worden gebruikt om meerdere databasebewerkingen te groeperen en ervoor te zorgen dat deze als één geheel worden uitgevoerd. Met JPA kunnen ontwikkelaars transacties definiëren met behulp van de Java Transaction API (JTA). JTA biedt een uniforme manier om transacties te beheren, ongeacht de onderliggende database.

## Voorbeelden

Hieronder volgen enkele voorbeelden van scenario's waarin JPA kan worden gebruikt:

### E-Commerce

Een e-commerce website heeft een database met klantinformatie en bestellingen. De ontwikkelaar kan JPA gebruiken om entities te definiëren voor klanten en bestellingen, en deze op een objectgeoriënteerde manier te gebruiken in de code. Zo kan bijvoorbeeld een query worden geschreven om alle bestellingen op te halen die zijn geplaatst door een bepaalde klant.

### Bedrijfsbeheersysteem

Een bedrijf heeft een database met werknemersinformatie en projecten. De ontwikkelaar kan JPA gebruiken om entities te definiëren voor werknemers en projecten, en relaties tussen deze entities te definiëren. Zo kan bijvoorbeeld een query worden geschreven om alle werknemers op te halen die werken aan een specifiek project.

### Mobiele App voor SMS

Een mobiele app heeft een database met gebruikersinformatie en berichten. De ontwikkelaar kan JPA gebruiken om entities te definiëren voor gebruikers en berichten, en deze te gebruiken in de code om nieuwe berichten toe te voegen aan de database of bestaande berichten op te halen.

## JPA vs JDBC

In al deze scenario's kan JPA de ontwikkeling vereenvoudigen door een hogere abstractielaag te bieden en de complexiteit van JDBC te verbergen. Ontwikkelaars kunnen zich concentreren op het definiëren van entities en relaties in Java-code, in plaats van SQL-statements te schrijven en zich zorgen te maken over de onderliggende databasestructuur. JPA zorgt voor de conversie van de objectgeoriënteerde code naar de relationele databasestructuur en stelt ontwikkelaars in staat om efficiënter en effectiever te werken.

In de bovenstaande scenario's zou het gebruik van JDBC en JPA enkele belangrijke verschillen hebben. Hieronder staan enkele voorbeelden:

### SQL-Kennis

Met JDBC zou de ontwikkelaar SQL-statements moeten schrijven om gegevens uit de database op te halen, te wijzigen of toe te voegen. Dit vereist kennis van SQL-syntax en databasestructuren. Met JPA kan de ontwikkelaar entities definiëren voor de klant- en bestellingstabellen en relaties tussen deze entities definiëren. Vervolgens kunnen queries worden geschreven in JPQL (Java Persistence Query Language) om gegevens op te halen. Deze aanpak vereist minder kennis van SQL en maakt het gemakkelijker om objectgeoriënteerde code te schrijven.

### Relaties tussen tabelen

Met JDBC zou de ontwikkelaar SQL-statements moeten schrijven om gegevens op te halen uit de werknemer- en projecttabellen en om relaties tussen deze tabellen te definiëren. Met JPA kan de ontwikkelaar entities definiëren voor werknemers en projecten en relaties tussen deze entities definiëren. Dit kan het schrijven van complexe SQL-statements vermijden en maakt het gemakkelijker om de objectgeoriënteerde code te schrijven.

### Gegevens uit de tabellen behandelen

Met JDBC zou de ontwikkelaar SQL-statements moeten schrijven om gegevens uit de gebruikers- en berichtentabellen op te halen, te wijzigen of toe te voegen. Met JPA kan de ontwikkelaar entities definiëren voor gebruikers en berichten en deze op een objectgeoriënteerde manier gebruiken in de code. Dit maakt het gemakkelijker om de gegevens te beheren en te wijzigen en kan het aantal SQL-statements verminderen dat nodig is om de database te wijzigen.

### Algemene verschillen

Over het algemeen kan worden gesteld dat JPA het werken met databases gemakkelijker en efficiënter maakt voor ontwikkelaars, omdat het een hogere abstractielaag biedt dan JDBC. JPA verbergt de complexiteit van de onderliggende databasestructuur en biedt een objectgeoriënteerde manier om met de gegevens in de database om te gaan. Dit kan het ontwikkelingsproces versnellen en de code eenvoudiger en onderhoudbaarder maken.

## Samenvatting

Door deze concepten te begrijpen, kunnen Java-ontwikkelaars JPA gebruiken om hun applicaties op een objectgeoriënteerde manier te schrijven, zonder zich zorgen te hoeven maken over de complexiteit van JDBC. JPA vereenvoudigt het ontwikkelingsproces door een hogere abstractielaag te bieden, waardoor het gemakkelijker wordt om database-entiteiten te definiëren, te gebruiken en te onderhouden.