DEZSYS ORM

Projektbeschreibung

This exercise is intended to demonstrate the interaction between a programming language (Java) and a persistance layer (MySQL, PostgreSQL).

First you should follow the Spring tutorial "Accessing data with MySQL" and document all important steps in your protocol. Don't forget to make notes about all problems occured during the setup. Afterwards you should extend the data model of the example and adapt it for a Data Warehouse application (data structure see below). One relation between the entities Datawarehouse and Products is required in this example. Please read the documentation how you implementation entity relations using the ORM model.

Document all individual implementation steps and any problems that arise in a log (Markdown).

Create a GITHUB repository for this project and add the link to it in the comments.

Projekteinführung

In diesem Projekt habe ich einen Webshop mit Zugriff auf eine MySQL Datenbank erstellt. Dabei benutze ich Docker zum starten von meiner Datenbank und Rest für meine Funktionalität.

Theorie

Die Theorie zu diesem Kapitel haben wir im Unterricht und von den Links auf Elearning gelernt.

Arbeitsschritte

Wir müssen uns als erstes eine MySQL Datenbank aufsetzen, dafür machen wir ein docker pull mysql und ein docker exec -it mysql, aber am besten gleich in der Docker App selbst. Danach kann ich das Projekt von elearning klonen und einmal austesten. Dafür muss ich die Verbindung in die application.properties

Datei schreiben und kann das Programm ausprobieren. Die nächste Aufgabe ist dann die Erweiterung von den Funktionen auf alle CRUD Operationen. Wenn ich das gemacht habe, kann ich das Programm starten und die Funktionen austesten.

Crud Operationen

Die Crud Operationen, die ich mir ausgesucht habe, sind die folgenden 5:

Find

Bei der klassischen Get Methode muss man nichts weiter als den Namen der Tabelle und man kann direkt den ganzen Inhalt ausgeben: `db.warehouses.find().pretty()`

Insert

Die Insert Methode schaut folgendermaßen aus: `db.warehouses.insertOne({warehouseid: 2,warehousename: "Lager München",warehouseaddress: "Lagerstraße 5",warehousepostalcode: "80331",warehousecity: "München",warehousecountry: "Deutschland",timestamp: "2025-05-27T10:00:00Z",productdata: [{productid: 201,productname: "Monitor",price: 149.99},{productid: 202,productname: "Laptop-Ständer", price: 39.99}]})`

Hier füge ich ein warehouse hinzu, wobei in diesem warehouse direkt zwei produkte enthalten sind.

Delete

Bei Delete kann ich ein bestimmtes Warehouse nach einem Kriterium löschen. Das sieht foglendermaßen aus:

`db.warehouses.deleteOne({ warehouseid: 2 })`

Hier habe ich nach der ID gelöscht.

Find mit Filter

Bei der find Funktion kann ich auch Filter benutzen. Diese werden dann danach ungefähr so:

`db.products.find({productQuantity: { $gt: 3000 }}).pretty()`

aus. Dabei kann ich nach greater than, less than, greater than or equal (gte) oder auch gleich filtern.

Update

Bei der Update Funktion, kann man einen bestimmten Wert ersetzen. Das sieht so aus: `db.products.updateOne( { productName: "Bio Apfelsaft Gold" }, { $set: { productQuantity: 4000 } })`

EK

Mehrere Lagerstandorte

Um die Möglichkeit für mehrere Lagerstandorte anzubieten, muss man nur ein Attribut an die Produkte anhängen, dass das Warehouse angibt. Damit kann man das Produkt an zwei verschiedenen Standorten angeben.

Neue Mongo Shell Befehle

```
db.products.aggregate([ { $group: { _id: "$warehouseID", productCount: { $sum: 1 } } } ])
```

Dieser Befehl sucht mir die Anzahl von Objekten an einem Standort aus.

```
db.products.countDocuments({ productName: "Screen" })
```

Hier suche ich nach allen Produkten, die den Namen Screen haben

```
db.products.find().sort({ productQuantity: -1 })
```

Hier sortiere ich alle Produkte absteigend

Zusammenfassung

Durch diese Projekt habe ich die Fähigkeit erlernt, die Daten, die ich auf einer Website eingeben kann, auch in einer Datenbank abspeichern.

Quellen

https://www.w3schools.com/MySQL/default.asp