DEZSYS_GK81_WAREHOUSE_ORM

@author: David Socaciu 4DHIT

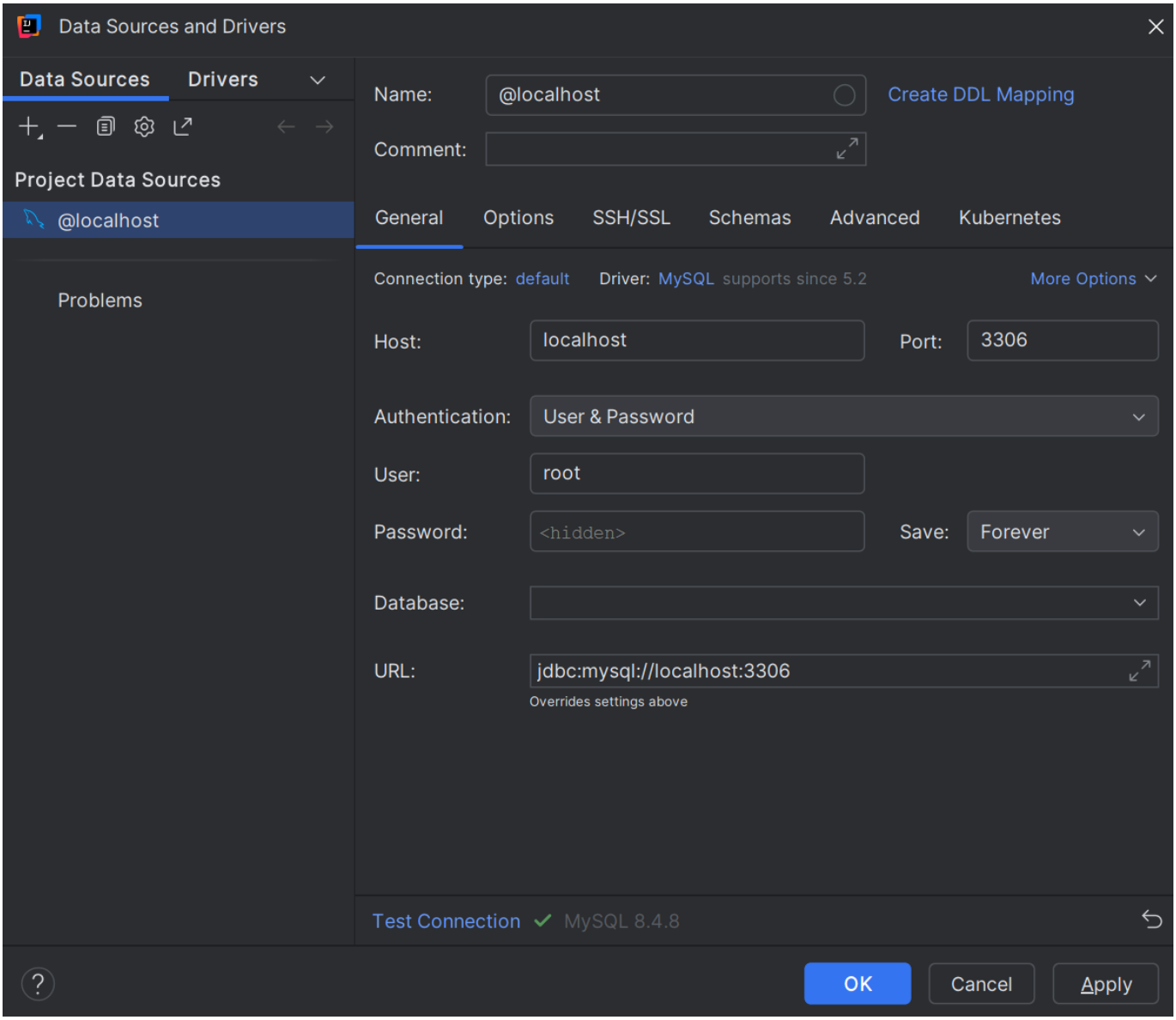
@version: 2026-02-24

Grundlagen

First, I cloned the project from GitHub and prepared everything for execution. Afterwards, I directly took care of MySQL and started a Docker container running MySQL on the specified port.

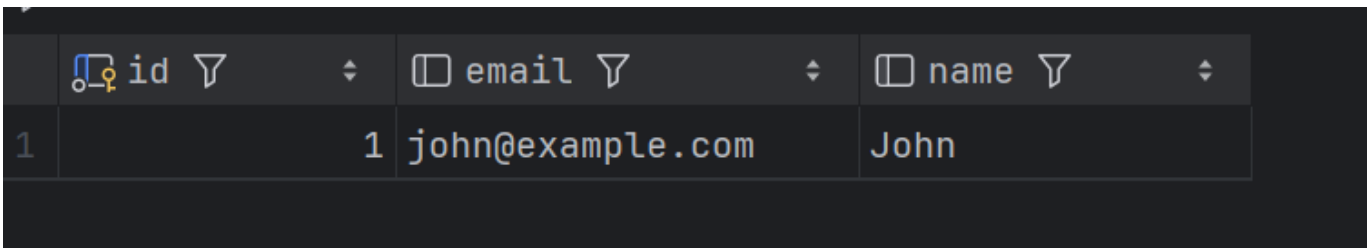
```
PS C:\Users\Socac> docker run --name mysql-spring -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=example
p 3306:3306 -d mysql:8
```

Now the container is running and when I connect to the database in IntelliJ, it can be seen that the connection works successfully.



Afterwards, I added a user using curl and it can also be seen in the database that the entry for the newly created user was successfully inserted.

```
C:\Users\Socac>curl -X POST "http://localhost:8080/demo/add" -d "name=John" -d "email=john@example.com"
Saved
```



First i started by creating the classes:

`WarehouseRepository`

`ProductRepository`

`Warehouse` (Entity)

`Product` (Entity)

`DataService` in which i create those warehouses & products

In the entity classes i labeled those with `@Entity` and table name which is optional but is responsible for how the table is called in the db. And marked them each with a id which automatically increments

Here is a snippet from how it looks:

```
@Entity
@Table(name = "warehouse")
public class Warehouse {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;


    private String warehouseID;
    private String warehouseName;
    private String warehouseAddress;
    private String warehousePostalCode;
    private String warehouseCity;
    private String warehouseCountry;

    private LocalDateTime timestamp;
```

And those entity also need a relation

A warehouse can contain multiple products.

The `@OneToMany` annotation defines this relationship.

```
    @OneToMany(mappedBy = "warehouse", cascade = CascadeType.ALL)
    private List<Product> products;
```

And the one in the Product entity:

```java
@ManyToOne
@JoinColumn(name = "warehouse_id")
private Warehouse warehouse;
```

Each product belongs to exactly one warehouse.
The `@ManyToOne` annotation defines that many products can reference one warehouse.

And last i created in the Service class the Products and warehouses:

Here are two examples:

```java
Warehouse warehouse1 = new Warehouse();
warehouse1.setWarehouseID("001");
warehouse1.setWarehouseName("Linz Bahnhof");
warehouse1.setWarehouseAddress("Bahnhofsstrasse 27/9");
warehouse1.setWarehousePostalCode("4020");
warehouse1.setWarehouseCity("Linz");
warehouse1.setWarehouseCountry("Austria");
warehouse1.setTimestamp(LocalDateTime.now());
warehouseRepository.save(warehouse1);
```

```java
Product p1 = new Product();
p1.setProductID("00-443175");
p1.setProductName("Bio Orangensaft Sonne");
p1.setProductCategory("Getraenk");
p1.setProductQuantity(2500);
p1.setProductUnit("Packung 1L");
p1.setWarehouse(warehouse1);
productRepository.save(p1);
```

And when i run the program we can see that those are in the database:

WHERE ORDER BY

	id	timestamp	warehouse_address	warehouse_city	warehouse_country	warehouseid
1	1	2026-02-24 14:52:38.442444	Bahnhofsstrasse 27/9	Linz	Austria	001
2	2	2026-02-24 14:52:38.554567	Südtirolerplatz	Wien	Austria	002

WHERE ORDER BY

	id	product_category	productid	product_name	product_quantity	product_unit	wareho
1	1	Getraenk	00-443175	Bio Orangensaft Sonne	2500	Packung 1L	
2	2	Getraenk	00-871895	Bio Apfelsaft Gold	3420	Packung 1L	
3	3	Getraenk	00-123456	Mineralwasser	5000	Flasche 1L	
4	4	Getraenk	00-654321	Cola	4200	Dose 0.5L	
5	5	Getraenk	00-998877	Eistee	3100	Flasche 1L	
6	6	Milchprodukte	00-112233	Milch	2800	Packung 1L	
7	7	Milchprodukte	00-445566	Butter	1500	250g	
8	8	Milchprodukte	00-778899	Joghurt	3600	Becher 200g	
9	9	Backwaren	00-667788	Vollkornbrot	1800	Laib	
10	10	Backwaren	00-334455	Croissant	2200	Stueck	

Questions:

1. What is ORM and how is JPA used?
ORM maps Java objects to database tables.
JPA is used to define and manage these mappings in Java applications.

2. What is application.properties used for and where is it stored?
It contains configuration settings such as database connection and Hibernate options.
It must be stored in `src/main/resources`.

3. Which annotations are frequently used for entity types?
Common annotations are `@Entity`, `@Table`, `@Id`, `@GeneratedValue`, `@OneToMany`, and `@ManyToOne`.
Every entity must have a primary key and correctly defined relationships.

4. What methods are needed for CRUD operations?
`save()`, `findById()`, `findAll()`, `delete()`, and `count()`.

Erweiterte Grundlagen

To complete the extended requirements, I first analyzed the available methods of `CrudRepository` and `JpaRepository`.
These include methods such as `save()`, `findById()`, `findAll()`, `delete()`, and `count()`.

After that, I extended the repositories with custom query methods.

In the `WarehouseRepository`, I added:

```
Optional<Warehouse> findByWarehouseID(String warehouseID);
```

This method allows retrieving a specific warehouse by its business ID.

In the `ProductRepository`, I added:

```
Optional<Product> findByWarehouse_WarehouseIDAndProductID(String warehouseID, String
productID);
```

Optional findByWarehouse_WarehouseIDAndProductID(String warehouseID, String productID);

Data Model:

To extend the data model, I created a new entity called `Purchase`.

This entity represents customer purchases and contains:

- amount
- purchaseTime
- relation to Product
- relation to Warehouse

The relationships were implemented using:

```
@ManyToOne
@JoinColumn(name = "product_id")
private Product product;

@ManyToOne
@JoinColumn(name = "warehouse_id")
private Warehouse warehouse;
```

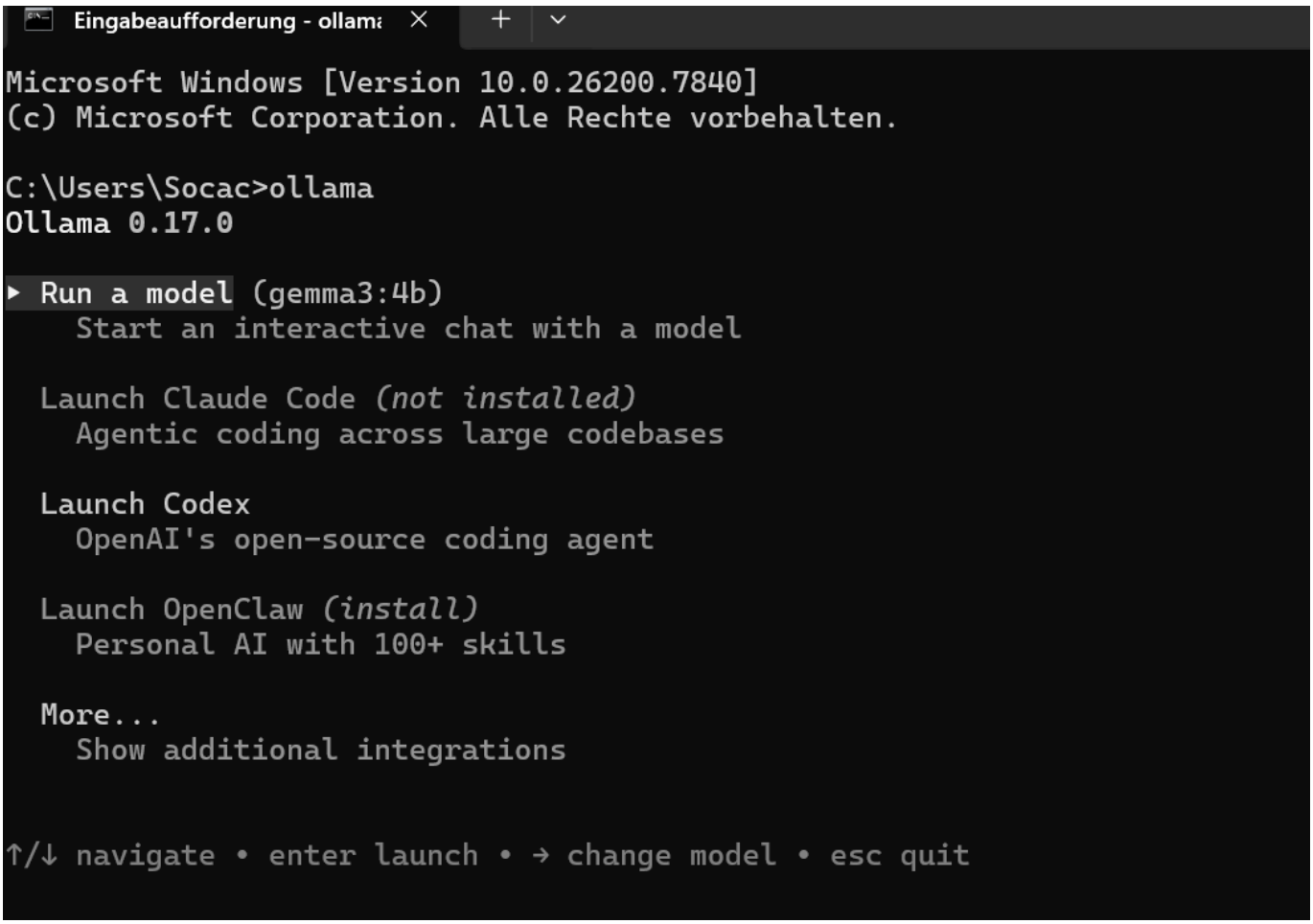
Finally, I inserted 30 purchase records inside the `DataService` class using a loop.
After running the application, the purchase records were visible in the database, and the total number of purchases was verified using `purchaseRepository.count()`.

Here is the table:

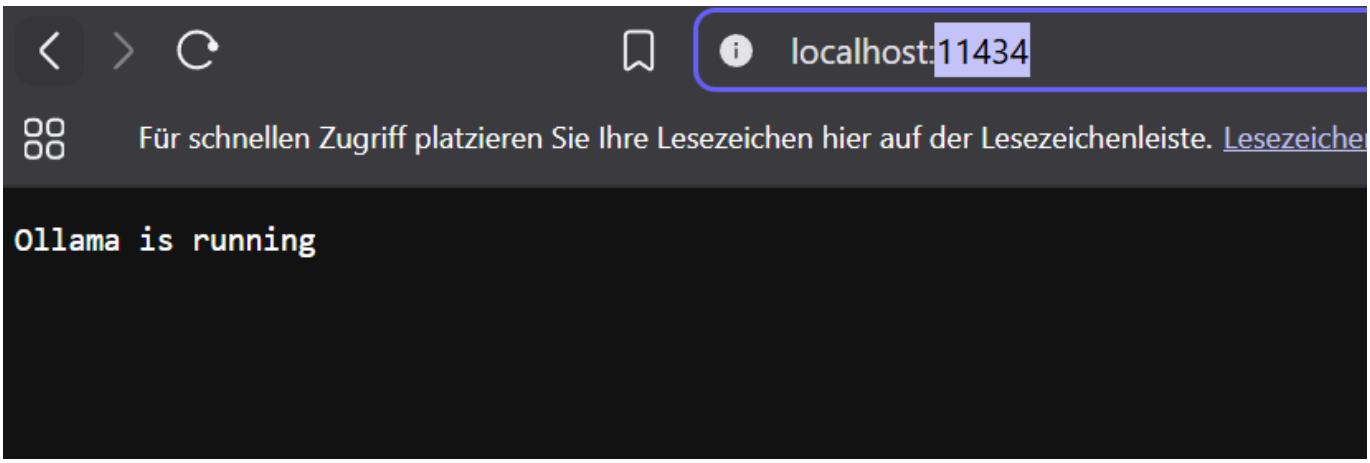
	amount	id	product_id	purchase_time	warehouse_id
1	6	1	1	2026-02-24 15:27:36.602903	1
2	7	2	1	2026-02-24 15:27:36.614869	1
3	8	3	1	2026-02-24 15:27:36.627842	1
4	9	4	1	2026-02-24 15:27:36.643570	1
5	10	5	1	2026-02-24 15:27:36.664053	1
6	11	6	1	2026-02-24 15:27:36.684459	1
7	12	7	1	2026-02-24 15:27:36.699897	1
8	13	8	1	2026-02-24 15:27:36.714646	1
9	14	9	1	2026-02-24 15:27:36.728052	1
10	15	10	1	2026-02-24 15:27:36.743626	1
11	16	11	1	2026-02-24 15:27:36.762029	1
12	17	12	1	2026-02-24 15:27:36.781949	1
13	18	13	1	2026-02-24 15:27:36.799862	1
14	19	14	1	2026-02-24 15:27:36.824801	1
15	20	15	1	2026-02-24 15:27:36.845304	1
16	21	16	1	2026-02-24 15:27:36.866713	1
17	22	17	1	2026-02-24 15:27:36.891409	1
18	23	18	1	2026-02-24 15:27:36.914293	1

Vertiefung:

For this i first downloaded Ollama and installed an model. Then i runned Ollama in the terminal with ollama and then run model:



Afterwards i checked if its really runnung on port 11434 and this was my response:



So now i need to implement the code:

FIrst i created a class:

private final RestTemplate restTemplate = new RestTemplate();

public String generateForecast(String salesData) {

```
String url = "http://localhost:11434/api/generate";

Map<String, Object> request = new HashMap<>();
request.put("model", "gemma3:4b");
request.put("prompt",
        "Based on the following sales data, predict the sales for the next 3 months
just with the data you received" +
                "no questions or anything clean structured and everything:\n"
                + salesData);
request.put("stream", false);

HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);

HttpEntity<Map<String, Object>> entity =
        new HttpEntity<>(request, headers);

ResponseEntity<String> response =
        restTemplate.postForEntity(url, entity, String.class);

return response.getBody();
```

}

Now first you see the model which i used, then the prompt which i then create

Afterwards i get the Response with response.getbody

Here i implemented the 300 purchases randomized:

```
        for (int i = 1; i <= 300; i++) {

            Purchase purchase = new Purchase();

            purchase.setAmount((int)(Math.random() * 20) + 1);
            purchase.setPurchaseTime(
                    LocalDateTime.now().minusDays((int)(Math.random() * 180))
            );

            purchase.setProduct(p1);
            purchase.setWarehouse(warehouse1);

            purchaseRepository.save(purchase);
        }
```

And lastly i need an endpoint in my controller:

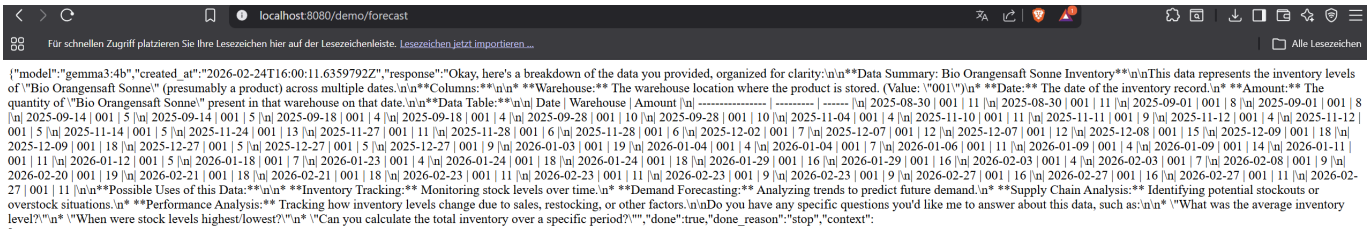
```java
@GetMapping("/forecast")
@ResponseBody
public String forecast() {

    StringBuilder salesData = new StringBuilder();

    purchaseRepository.findAll().forEach(p ->
            salesData.append(
                    "Warehouse: " + p.getWarehouse().getWarehouseID()
                            + ", Product: " + p.getProduct().getProductName()
                            + ", Amount: " + p.getAmount()
                            + ", Date: " + p.getPurchaseTime()
                            + "\n"
            )
    );

    return llmService.generateForecast(salesData.toString());
```

Here i put every purchase which will go in to the prompt and this is the respond i got:



{"model":"gemma3:4b","created_at":"2026-02-24T16:00:11.6359792Z","response":"Okay, here's a breakdown of the data you provided, organized for clarity:\n\n**Data Summary: Bio Orangensaft Sonne Inventory**\n\nThis data represents the inventory levels of \"Bio Orangensaft Sonne\" (presumably a product) across multiple dates.\n\n**Columns:**\n\n* **Warehouse:** The warehouse location where the product is stored. (Value: \"001\")\n* **Date:** The date of the inventory record.\n* **Amount:** The quantity of \"Bio Orangensaft Sonne\" present in that warehouse on that date.\n\n**Data Table:**\n\n| Date | Warehouse | Amount |\n| ---------- | --------- | ------ |\n| 2025-08-30 | 001 | 11 |\n| 2025-08-30 | 001 | 11 |\n| 2025-09-01 | 001 | 8 |\n| 2025-09-01 | 001 | 8 |\n| 2025-09-14 | 001 | 5 |\n| 2025-09-14 | 001 | 5 |\n| 2025-09-18 | 001 | 4 |\n| 2025-09-18 | 001 | 4 |\n| 2025-09-28 | 001 | 10 |\n| 2025-09-28 | 001 | 10 |\n| 2025-11-04 | 001 | 4 |\n| 2025-11-10 | 001 | 11 |\n| 2025-11-11 | 001 | 9 |\n| 2025-11-12 | 001 | 4 |\n| 2025-11-12 | 001 | 5 |\n| 2025-11-14 | 001 | 5 |\n| 2025-11-24 | 001 | 13 |\n| 2025-11-27 | 001 | 11 |\n| 2025-11-28 | 001 | 6 |\n| 2025-11-28 | 001 | 6 |\n| 2025-12-02 | 001 | 7 |\n| 2025-12-07 | 001 | 12 |\n| 2025-12-07 | 001 | 12 |\n| 2025-12-08 | 001 | 15 |\n| 2025-12-09 | 001 | 18 |\n| 2025-12-09 | 001 | 18 |\n| 2025-12-27 | 001 | 5 |\n| 2025-12-27 | 001 | 5 |\n| 2025-12-27 | 001 | 9 |\n| 2026-01-03 | 001 | 19 |\n| 2026-01-04 | 001 | 4 |\n| 2026-01-04 | 001 | 7 |\n| 2026-01-06 | 001 | 11 |\n| 2026-01-09 | 001 | 4 |\n| 2026-01-09 | 001 | 14 |\n| 2026-01-11 | 001 | 11 |\n| 2026-01-12 | 001 | 5 |\n| 2026-01-18 | 001 | 7 |\n| 2026-01-23 | 001 | 4 |\n| 2026-01-24 | 001 | 18 |\n| 2026-01-24 | 001 | 18 |\n| 2026-01-29 | 001 | 16 |\n| 2026-01-29 | 001 | 16 |\n| 2026-02-03 | 001 | 4 |\n| 2026-02-03 | 001 | 7 |\n| 2026-02-08 | 001 | 9 |\n| 2026-02-20 | 001 | 19 |\n| 2026-02-21 | 001 | 18 |\n| 2026-02-21 | 001 | 18 |\n| 2026-02-23 | 001 | 11 |\n| 2026-02-23 | 001 | 11 |\n| 2026-02-23 | 001 | 9 |\n| 2026-02-23 | 001 | 9 |\n| 2026-02-27 | 001 | 16 |\n| 2026-02-27 | 001 | 16 |\n| 2026-02-27 | 001 | 11 |\n| 2026-02-27 | 001 | 11 |\n\n**Possible Uses of this Data:**\n\n* **Inventory Tracking:** Monitoring stock levels over time.\n* **Demand Forecasting:** Analyzing trends to predict future demand.\n* **Supply Chain Analysis:** Identifying potential stockouts or overstock situations.\n* **Performance Analysis:** Tracking how inventory levels change due to sales, restocking, or other factors.\n\nDo you have any specific questions you'd like me to answer about this data, such as:\n\n* \"What was the average inventory level?\"\n* \"When were stock levels highest/lowest?\"\n* \"Can you calculate the total inventory over a specific period?\"","done":true,"done_reason":"stop","context":

I formatted it to get a better result:

Here’s a breakdown of the data provided, focusing on key aspects and potential uses:\n\n**Data Overview**\n\n* **Product:** “Bio Orangensaft Sonne” (Sun Orange Juice)\n* **Warehouse:** 001\n* **Data Structure:** The data is presented as a series of records, each detailing a specific quantity of the product and its associated date.\n\n**Key Observations \u0026 Potential Uses**\n\n1. **Time Series Data:** This is a time series dataset, where the data is recorded at specific points in time. This is *extremely* valuable for:\n\n * **Demand Forecasting:** Analyzing the frequency of product movement over time can help predict future demand. You could look for patterns (e.g., higher demand on certain days of the week, seasonal trends).\n * **Inventory Management:** The data directly informs inventory levels. You can determine how much stock to keep on hand based on past sales.\n * **Trend Analysis:** Identify if sales are increasing, decreasing, or stable over time.\n\n2. **Frequency Analysis:**\n * Count the number of occurrences of each quantity (1, 2, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20). This would reveal which quantities are most commonly sold.\n\n3. **Date Analysis:**\n * **Day-of-Week Analysis:** Determine if sales vary by day of the week.\n * **Month-of-Year Analysis:** Identify seasonal patterns (e.g., higher sales during the summer months).\n * **Lead Time:** Calculate the time between when a product is recorded at the warehouse and when it is sold. This is important to understand fulfillment times.\n\n4. **Potential Insights (Based on the initial data):**\n\n * The quantity of ‘20’ appears frequently. This likely represents a larger order or a significant sales event.\n * The quantity of ‘1’ appears frequently, indicating potentially a small-scale purchase or a promotional item.\n\n**Possible Analytical Techniques**\n\n* **Descriptive Statistics:** Calculate measures like average sales quantity, standard deviation, minimum, and maximum.\n* **Moving Averages:** Smooth out fluctuations in the time series data to identify trends more clearly.\n* **Regression Analysis:** Explore the relationship between sales quantity and factors like date (day of week, month, year) to build a predictive model.\n* **Time Series Decomposition:** Separate the data into trend, seasonal, and residual components.\n\n**Important Note:** To get the most out of this data, you would need *much* more data (e.g., additional warehouses, product variations, promotional information, external factors like weather) to build robust and accurate models