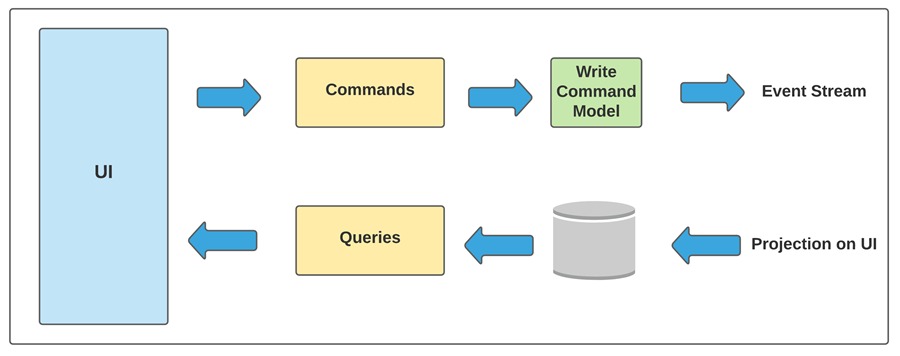
# **Problem Statement**

We must separate the operations for reading the data from the operations for writing or updating the data, according to the Command and Query Responsibility Segregation (CQRS) paradigm. This indicates that separate interfaces or classes are not maintained for data reading and writing functions.

We must bear in mind, nevertheless, that this design pattern works better for larger systems where the demands and stress levels for read and write operations differ. The typical CRUD pattern, which is frequently automatically produced by ORM technologies, is adequate for a straightforward and modest application.

Our goal is to create an application with two distinct repositories. There are two: one for writing operations and one for read operations. The corresponding middle-tier components will use these repositories when needed. There will be two repositories: one for reads (queries) and one for commands (writes).

# **Working of CQRS**



# **Methodology**

First of all, we will create two repositories one for reads (queries) and another for writes (commands) in the repositories folder. After creating repositories, we will write read and write operations in respective repositories. Next we will have to make two middle-tier components again one for reads (queries) and another for writes (commands) in the commands and queries folder. These middle-tier components will handle the commands and queries operations.

Next we have to make a class which represents the data means in this class we will define our attributes and we will also define getter and setter for each of these attributes. After this we will have to make a class which will be used by the queries (reads) operations to return data in a shape or object required by the requestor or client.

Lastly to call or utilize these writes (commands) and reads (queries) operation from main program we will need some dependency injection framework to instantiate objects of the repositories we defined.

# **Conclusion**

In comparison to conventional CRUD activities, the CQRS design pattern takes a holistic approach that dramatically simplifies the architecture and gets rid of redundancies and complexity. To minimize complications, we must adopt the CQRS pattern when developing intricate enterprise systems.