# **Learning Journal - Unit 7**

Computer Science, University of the People

CS 2203-01 Databases 1 - AY2024-T3

Instructor, Irfan Rashid Thoker

March 21, 2024

This week I was given the task of building and seeding a database for a hospital system. This included writing the scripts to generate the tables using DDL (Data Definition Language) commands for creating the tables and DML (Data Manipulation Language) to seed and populate the tables. I was also asked to retrieve and query the data using select commands. It was important while building the scripts to make sure that I chose the correct data types to use in each column and that I made the correct connections between the tables using foreign keys.

The normalization concept and methods of course played a part in these processes. Since I had to make sure that the resulting database had as little redundancy as possible and that it was able to support the level of data integrity that I was looking for. Another concept that was part of the week’s tasks was the ACID properties (Atomic, Consistency, Isolation, and Durability) which when designing a large and critical system such as a hospital system are crucial to the solution.

Learning and exploring database programming I came across two approaches that stood out to me:

1. **Embedded SQL** – High performance and tight control over the transactions. These benefits of course come at a price, being, the application code and database become tightly embedded with one another making it hard to change one or the other out later. This would also most likely make maintenance harder.
2. **API interfaces / ORM’s (Object Relational Mapping)** – These options give us an abstraction layer between our database and our code. Meaning that it will be easier for us to swap out our database engine with another with relative ease. The maintainability of our application and database has also improved. These, however, come at a price, that being the opposite of the above. Here we are more likely to see a slight performance decrease. Also, it’s important to note that some teams and people might find the use of an ORM a bit more complex and might involve a steeper learning curve.

Looking over these options I again find myself in the same position of not having a clear option. But again, we need to way each option again for each project and that project needs and requirements. So, for instance for a system that I might find needs and warrants a more performance heavy solution I would most likely go for the embedded SQL approach. But for systems where the performance takes a back seat to maintainability and flexibility, I would most likely choose the API/ORM based options. Of course, there is always the option to also mix and match both approaches in a single solution that way getting the best of both worlds at a more granular level so parts of the system that need more performance can use embedded SQL and parts with less performance heavy operations can utilize the ORM’s benefits.

## References

* Learning Guide Unit 1-7  
  <https://my.uopeople.edu/course/view.php?id=7455>
* Sharma, N., Perniu, L., Chong, R. F., Iyer, A., Nandan, C., Mitea, A. C., Nonvinkere, M. & Danubianu, M. (2010). Database fundamentals. IBM Canada.  
  <https://my.uopeople.edu/pluginfile.php/1827130/mod_book/chapter/484065/Database_Fundamentals.pdf>
* Watt, A., & Eng, N. (2014). Database design,  2nd ed. BCcampus, BC Open Textbook Project.   
  <https://opentextbc.ca/dbdesign01/>  
  <https://my.uopeople.edu/pluginfile.php/1827130/mod_book/chapter/484065/Database-Design-2nd-Edition-1560272109.pdf>