Contents

Databases

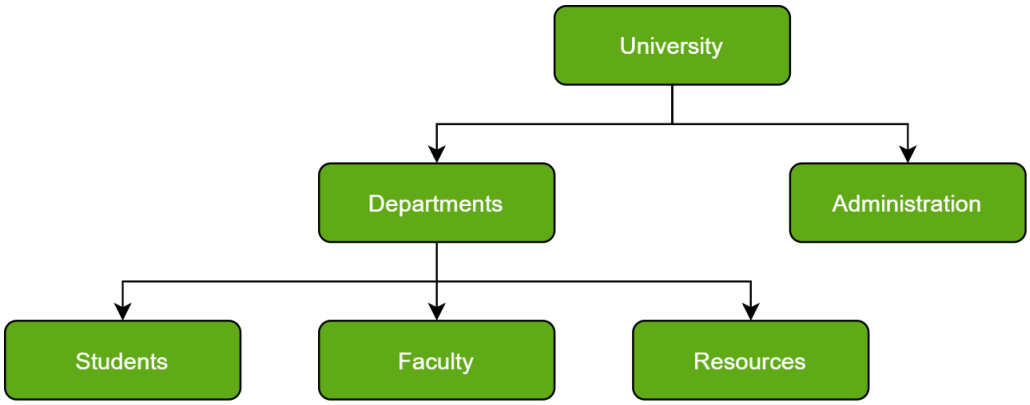
In my search for a database, I have come to realize I never quite knew what the difference is between the database options. They all store data, right? Last semester I used Microsoft SQL server hosted on Azure for my IP and GP without much thought. It was something I heard of before thus I picked it. This semester however, I wanted to put some more thought in.

Database Types

If I were asked the type of DB I used last semester, I would not be able to answer. I want to first look at the type of databases before looking at actual databases. Not shockingly, I have found there are many ways to store data. When it comes to databases however, there are some that are more used than others.

Hierarchical

As the name suggests, the hierarchical databases are structured similarly to a hierarchy. The data is organized in a tree-like structure where each record has a parent-child relationship. Each child record can have only one parent, but a parent record can have multiple child records.



Relational

A relational database organizes data into tables with rows and columns. The tables represent entities, and the relationships between entities are defined using keys. Data is stored in a structured manner, ensuring consistency and integrity. The SQL (Structured Query Language) is commonly used to query and manipulate relational databases. Relational databases are flexible, scalable, and widely adopted for various applications.

Non-Relational

Non-relational databases, also known as NoSQL databases, are designed to handle unstructured, semi-structured, or large volumes of data that don't fit well into the rigid structure of a relational database. NoSQL databases use various data models, including key-value, document, columnar, and graph-based. They provide flexible schemas, horizontal scalability, and high-performance. NoSQL databases are commonly used for web applications, real-time analytics, and handling big data.

Object oriented

Object-oriented databases store data as objects, like object-oriented programming concepts. Objects encapsulate data and behaviour into a single entity. These databases support the storage of complex data types, such as images, audio, and video. They offer features like inheritance, encapsulation, and

polymorphism. Object-oriented databases are used when there is a need to directly store and manipulate objects without mapping them to relational tables.

Database services

Having a small understanding of the different types of databases, I need to decide what structure I want for my application. I personally like structure and a good overview like the relational databases. With that choice made, I will have to look in the different types of available databases.

MySQL

Pros

High performance: MySQL is known for its excellent performance and can handle large volumes of data efficiently.

Scalability: MySQL is designed to scale horizontally by adding more servers to distribute the load.

Open-source: MySQL is open-source software, which means it's free to use, and there's a vast community of developers contributing to its development.

Large community support: The MySQL community is extensive, and there are many resources available, including tutorials, documentation, and support forums.

Cross-platform compatibility: MySQL runs on various platforms, including Windows, Linux, and Mac OS.

Ease of use: MySQL is relatively easy to set up and use, even for non-experts.

Cons

Limited functionality: Compared to some other relational databases, MySQL has a limited set of advanced features, such as stored procedures and triggers.

Limited security features: MySQL has some security features, but they are not as comprehensive as those of some other databases.

Data integrity: MySQL's default storage engine, MyISAM, does not guarantee referential integrity, which can lead to data inconsistencies.

Compatibility issues: Some third-party tools may not be fully compatible with MySQL, causing potential issues when using these tools.

Configuration complexity: Although MySQL is relatively easy to set up, configuring advanced features can be complex and time-consuming, requiring knowledge of the database internals.

PostgreSQL

Pros

Advanced features: PostgreSQL offers a wide range of advanced features, including support for complex queries, stored procedures, triggers, full-text search, and more.

Extensibility: PostgreSQL allows users to define custom data types, operators, and functions, providing flexibility and extensibility to meet specific requirements.

Reliability and data integrity: PostgreSQL ensures data integrity through the implementation of ACID (Atomicity, Consistency, Isolation, Durability) properties, making it a reliable choice for critical applications.

Scalability: PostgreSQL supports horizontal scalability through sharding and provides features like logical replication and streaming replication for high availability and fault tolerance.

Community support: PostgreSQL has an active and vibrant community that contributes to its development, provides support, and shares knowledge through forums, mailing lists, and conferences.

Cross-platform compatibility: PostgreSQL is available for multiple platforms, including Windows, Linux, and macOS, allowing developers to choose their preferred operating system.

Cons:

Complexity for beginners: PostgreSQL can have a steeper learning curve compared to simpler databases due to its rich set of features and advanced functionality.

Memory management: PostgreSQL can require more memory to perform optimally, especially for larger datasets and complex queries.

Performance trade-offs: While PostgreSQL offers advanced features, certain operations may be slower compared to other databases. Fine-tuning and optimization may be required for specific use cases.

Less widespread usage: PostgreSQL, while popular, is not as widely adopted as some other relational databases like MySQL or SQL Server, which may result in fewer available resources or third-party tools.

Microsoft SQL Server

Pros

Comprehensive feature set: Microsoft SQL Server offers a comprehensive set of features for managing and analysing data. It includes advanced capabilities for data integration, analytics, reporting, and business intelligence.

Integration with Microsoft ecosystem: SQL Server integrates seamlessly with other Microsoft products and technologies, such as .NET framework and Visual Studio, making it a preferred choice for developers using Microsoft technologies.

Excellent performance: SQL Server is known for its performance and optimization capabilities. It includes features like indexing, query optimization, and in-memory technologies to deliver efficient query execution and data processing.

Scalability and high availability: SQL Server provides scalability options such as partitioning, replication, and Always On availability groups for building highly available and scalable database solutions.

Strong security features: SQL Server offers robust security features, including encryption, authentication mechanisms, and fine-grained access control, to protect sensitive data.

Strong community and support: SQL Server has a large and active community, providing access to resources, forums, documentation, and support from Microsoft.

Cons

Licensing costs: While there is a free Express edition available, the full-featured versions of SQL Server can involve significant licensing costs, especially for larger deployments or enterprise-level usage.

Windows dependency: SQL Server is primarily designed to run on Windows operating systems, which may limit its usage in cross-platform or non-Windows environments.

Complexity: SQL Server has a wide range of features and configurations, which can make it complex to set up and manage, especially for inexperienced users.

Resource requirements: SQL Server can require significant system resources, including memory and storage, particularly for larger databases or high transaction volumes.

Limited availability of some features: Certain advanced features, such as partitioning or advanced analytics, may only be available in specific editions or require additional licensing.

Object Relational Mapping

What is an ORM?

Object-Relational Mapping is a technique or a framework that communicates with *relational-database* using object-oriented programming concepts. The main purpose of an ORM is to bridge the gap between the object-oriented world and the relational database world, allowing developers to work with objects in their code while transparently persisting and retrieving data from the database.

ORMs provide a way to define classes or models that represent database tables, where each instance of the class represents a row in the table. You can define relationships between different models, such as one-to-one, one-to-many, or many-to-many relationships, and the ORM handles the database operations required to maintain those relationships.

With an ORM, you can perform standard CRUD operations using object-oriented syntax, rather than writing your own raw SQL queries.

Pros and Cons

An ORM is not a requirement for each program, nor is it per-se standard to use. I want to take a good look at what could be the benefits and downsides of an ORM for my project.

Pros

Productivity: Using an ORM can simplify your data access code and reduce the amount of boilerplate code you need to write. ORMs provide higher-level abstractions that allow you to work with objects rather than writing raw SQL queries. This can speed up development and make your code more maintainable.

Object-Oriented Approach: ORMs enable you to work with objects in your code, making it easier to map database tables to classes and perform operations on them using object-oriented programming concepts. This can align well with your Vue front-end, as you can work with data in a more consistent and intuitive manner across your entire application.

Database Independence: ORMs provide a layer of abstraction between your code and the underlying database. This allows you to switch between different database providers without having to rewrite your data access code. If you ever need to migrate your application to a different database engine, an ORM can make the transition smoother.

Query Optimization: Most ORMs offer query optimization capabilities, automatically generating efficient SQL queries based on your object-oriented code. This can help improve the performance of your application by minimizing unnecessary database roundtrips and optimizing data retrieval.

Community Support and Resources: Popular ORM frameworks like Entity Framework (EF) have a large and active community, which means you can find extensive documentation, tutorials, and community support if you encounter any issues or have questions.

Cons

Learning Curve: ORM frameworks often have a learning curve associated with understanding their concepts, APIs, and best practices. It may take time for developers to become proficient in using an ORM effectively, especially if they are new to the framework.

Performance Overhead: ORM frameworks add a layer of abstraction between your code and the database, which can introduce some performance overhead. While most ORM frameworks are optimized for performance, there may be cases where writing raw SQL queries can be more efficient for specific complex scenarios.

Increased Complexity for Simple Queries: ORM frameworks excel at handling complex relationships and query scenarios. However, for simple queries or when working with large datasets, the ORM's abstraction layer might introduce unnecessary complexity. In these cases, writing direct SQL queries or using a lightweight micro-ORM might be more straightforward.

Increased Complexity in Debugging: When using an ORM, the SQL queries generated by the framework might not always be transparent. This can make debugging more complex, as you need to understand the underlying SQL being executed by the ORM.

Conclusion

Database

Having a better understanding between the differences and available options for databases, I have a good idea on what I want. For my project I think a relational database is the best option, as I personally like structure and it works well with an object-oriented programming language like c#, what I will use for the web API.

Knowing the type of database, I still need to pick a DB service. I researched three commonly used services and wrote down some pros and cons for them. My project is a small and personal project; I don't need comprehensive features or the top-notch security. What is important for me is a simple to learn and understand service with good resources for any potential issues. Knowing this, I will be using MySQL as a database service. It's one I have not used before, so it's something I will learn throughout the project. It does however appear to have a good community which will help me with any upcoming issues. I also have personal friends who have used this service, making it even easier for me to stay clear of issues.

ORM

Many comments and opinions on ORM's (that I found) seemed very mixed. More often than not I found people preferred raw SQL over ORM's. I suppose it boils down to personal preference. Considering I have worked with raw SQL before but not an ORM, I have decided to include an ORM in my project. As I am mostly using an ORM to gain experience first-hand, I will be using Entity Framework as it is by far the most used ORM for C#.