ECON485 – Homework Assignment 4

Course Registration System – NoSQL Concepts and Applications

Task 1 – Seat Availability Lookups (Key-Value Database)

In a traditional SQL-based course registration system, seat availability is usually calculated by counting how many students are registered in a section and comparing this number with the section capacity. During busy registration periods, many students check seat availability at the same time. This causes the database to run the same aggregation queries again and again, which creates heavy load and slows down the system.

A key-value database such as Redis can solve this problem more efficiently. Instead of calculating availability each time, the system stores the number of available seats as a simple value in memory. When a student registers for a section, this value is decreased by one. When a student drops the section, the value is increased by one. These operations are very fast and reduce pressure on the main SQL database.

Redis also supports atomic operations, which means updates happen safely even when many users act at the same time. This helps prevent problems such as registering more students than the allowed capacity. However, the system must ensure that Redis data stays consistent with the SQL database. If Redis fails or restarts, seat values may need to be recalculated.

Task 2 – Prerequisite Eligibility Caching

Prerequisite checks require the system to verify whether a student has completed required courses with sufficient grades. In an SQL system, this usually involves JOIN operations between prerequisite tables and completed course records. During registration periods, students may check their eligibility many times, which leads to repeated execution of the same queries.

Using a key-value database, the system can cache eligibility results for each student and course. Once eligibility is calculated, the result can be stored and reused. This avoids unnecessary database queries and improves performance. If student grades do not change during registration, cached results remain valid.

A document database such as MongoDB can also be used for this purpose. Instead of storing only a simple eligibility result, the system can store a document containing prerequisite details, student grades, and the final decision. This approach is more flexible and useful when prerequisite structures are complex. Cache expiration or invalidation strategies should be applied to ensure that outdated information is not shown.

Task 3 – Storing Complex Historical Actions (Document Database)

A course registration system must record many types of actions such as adding courses, dropping courses, withdrawals, prerequisite overrides, and approval decisions. These actions often have different types of information attached to them. In a relational database, storing this variety of data usually requires many tables and complicated relationships.

A document-oriented database provides a simpler solution for storing historical actions. Each student can have a document that contains all registration-related actions as a list of events. Each event can store different fields depending on the action type. This allows the system to grow and change without modifying a fixed schema.

Document databases work well for append-heavy data, where new records are added frequently and existing records are rarely updated. Registration history fits this pattern. Although document databases may provide fewer relational constraints than SQL, they are well suited for audit logs and administrative review processes.