I hereby confirm that I am choosing not to complete the full paper, and I request you to enter **82.315** as my paper grade, which will be the average of my other two examinations

Course Reflection

1. What this class is really about

The very first thing I learned after taking the Advanced Data Management course is how it is different from traditional database management. I quickly realized this course focused on both managing data in general and managing databases specifically. For example, we learnt the importance of data integration processes and transforming data into a desired format in the desired place through ETL milestone. As the course progressed, I felt there was a strong emphasis on how different kinds of data stores can be used to deal with different kinds of data for different purposes More importantly, we successfully learned advanced techniques used in various databases, like ETL, caching techniques, denormalization, materialized views, database triggers, stored procedures, and transactions.

I enjoyed that the course encompassed much more than playing around with data alone. As we progressed through the course and project, I was able to see the patterns and anomalies in the data. For example, the ETL milestone taught me that data is never clean, and with experience, finding patterns in the data also becomes intuitive. The course not only taught me data management and querying, but also enabled me to think from a data security standpoint, It was highly interesting to me to learn how prepared statements can handle SQL injection attacks.

Throughout the course, I loved the emphasis on Infrastructure as Code approach and DevOps way of creating EC-2 instances and installing MariaDB through bash scripts. This automation approach taught me how in the real world, we should always focus on consistency and repeatability if we are doing the same things over and again. This approach enabled us to see the different components that are usually wrapped around the database itself in a real-life production scenario.

A lot of Linux-related components like Putty, SSH, SSH ports, and how we should control them were taught even before diving into the database itself. Extensive knowledge of industry-relevant things like Linux operating Systems, Yum package management utilities, fork versions, and repo files can help us set up infrastructure and working environments for not just relational databases, but can extend to many more tools in our future projects. Public and Private key authentication taught us how Linux users in a distributed computing environment can interact with different servers and different directory levels in those servers.

What really felt like a Proof of concept database when beginning with the project, gradually started giving me a taste of the issues and complexities one can face with a real-world database and its architecture. Triggers milestone was the turning point for me in this course. It really gave me an idea of how the data can be updated in real-time whenever there is a change in related database tables, without any manual intervention. But, it also quickly made me imagine the complexities one may have to deal with, when this is scaled out to a production-level application database where

anywhere between thousands to millions of users try to access and update the data constantly, depending upon the application and business.

Strategies and techniques for managing concurrent user access and consistency in the data started making sense to me, and I was able to comprehend and appreciate them by the time we were taught data replication and clustering techniques. Advanced topics like Sharding, Replication, and Clustering with the real-life examples given in class gave me a good idea of how the scaling and availability issues are solved by the industry. I could relate theoretical concepts like ACID and CAP theorem with the project we built in a relational database and different NoSQL database models we explored as part of group presentations.

We discussed how NoSQL is both evolutionary and revolutionary in its own ways and how the lines are blurring between traditional relational databases and NoSQL databases with time. The NoSQL part of the course not only taught me different kinds of NoSQL models, but also some highly complex issues faced by engineers and businesses like impedance mismatch, and the developer productivity affected by it. A handful of things that I observed at an application level in the past make more sense and meaning to me after this course. I now understand things related to application development and data storage at a deeper and grassroots level. For example, I knew Object Relational Mapping was a thing earlier, but now I also realize the history of how NoSQL developed and the ways it enhances productivity and development speeds.

Overall, with this course, I took away more than I expected before the beginning of the course. I only thought I would be learning how to deal with databases in this course. But, It broadened my horizons of how data should be handled in a secure and smoother way in the world of distributed computing. I now realize, that with each decision we take in data management, there are plenty other areas of it which can get impacted in both positive and negative ways. With these learnings instilled in me, during the winter break, I look forward to exploring and trying some data visualization concepts and tools with the similar datasets we used in the project, alongside some cloud data services.

2. The best part about participating in the class

The best part about participating in the class is the ability to get insights from my peers, and the different experiences they had with the database systems in their previous projects and organizations. Perusall helped us with discussion and preparations and provided us with a learning exchange platform wherein we could share our understanding of the course contents. it was very helpful to interact with other students and ask questions when I felt like I was stuck.

Also, Perusall made me feel like I was able to give back through the means of comments and questions. I was able to answer small and simple questions related to topics like version control, automation, and scripting in my capacity, which I gained from my past experience in DevOps engineering. Milestone reviews in class encouraged me to share and gain insights from alternative approaches to writing efficient code.

Perusall review sections at the end of each video not only served as a quick recap of everything covered in the video, but also helped me in doing a quick revision before exams.

During the milestone reviews in class, Oftentimes, Dr.G's code demonstration helped me think about how an alternative approach to writing the code can make the database faster even though the goal was to achieve the same end result with the same code.

When my peers sought clarifications as to why their approach did not work in project milestones, I took those scenarios and made them as testing grounds in my database. I think breaking things helps me learn better and equip myself better for when I run into complex issues myself. So, participating and being present in class when students discussed what did not work for them, allowed me to learn more about what not to do as well. Also, being present in the class during these discussions helped me to be cautious about the issues that I could likely run into while solving the milestones.

The class was interactive, with Dr.G asking us questions on a regular basis and taking our opinions to gauge our understanding of various topics. This kept me active and interested throughout the course.

Through the group presentations for NoSQL databases, we were able to see a lot more databases that were presented to us by our peers. It definitely helped us appreciate the diverse application needs out there in the real world and presented various innovative solutions to deal with them. The various use cases and applications showcased in the presentations improved my awareness of how each database is unique in it's own way and tailored for the specific use case or problems it is designed to deal with.

I liked Dr.G's idea of making the NoSQL group presentation as a business sales presentation with compelling reasons to choose our database.

NoSQL Group presentation provided me with an opportunity to explore an additional database of my choice at a deeper level. In hindsight, I do feel that there is a bit more research I could have done about the database my group chose. Though there are parts in my presentation that I would have liked to be better, while giving the presentation in class, the questions which my peers asked helped me see the expectations a typical business would ask before choosing any database.

3. How to select the best persistence layer

One of the key takeaways from this course for me is that there should not be a hard and fast rule to consider any one persistent layer as the best. When choosing a persistence layer, we should always go with the best possible option for our application needs and use case, and this best possible option will be our best persistence layer. This can change on a case-to-case basis.

For any of my future projects, I will first analyze my application needs, the kind of data I store in it, and my possible data access patterns, and then start shortlisting the appropriate choices before zeroing down to my final option. To be sure about my access patterns, and the performance of my application and database, I highly lean towards prototyping to determine any possible issues and finalize my persistence layer selection.

After this course, I learned that there is a unique set of requirements and advantages with each of the following persistence layers.

Files

I prefer file systems, usually for any personal productivity files or documents. When it comes to handling any large or media files without a worry for concurrent data access issues, My first go-to choice will be a file system. If I were to build a small application like word processing software that would run locally on a user's computer alone, the file system would be my persistence layer to save the work done in the software. The file system follows a hierarchical structure allowing users the option for storage and retrieval of data in files and directories. Each word document coming out of any word processing software is essentially a file containing the textual and formatting information, and it can be organized within folders on users' computer's file systems. For filesystem, I would consider the most used one in our computers like NTFS(New Technology File System) used in windows machines or Apple File System for Mac.

Alternatively, If I need to collaborate with multiple users and a remote access to the file system is a must, then I would go with any appropriate modern-day distributed file system like Google File System or Azure Files based on the project needs and ease of access.

Relational Database

Relational databases are rigid about the structure in which we hold the data. It stores the data in form of tables and relates those tables with each other based on primary key – foreign key relations. They use relations between each tables and joins them together to fetch the data from the tables of our interest. Essentially, They form the links between data and tables on the fly when querying to retrieve data from our target tables. So, I prefer relational databases for structured and normalized data, where duplication of data will hurt the scope of the project or not add any advantage.

If I am tasked with a government information system, where the super speed querying or storing is not a requirement, but ACID compliance, and data integrity constraints are of utmost priority, my go-to choice will be a relational database. My choice of relational databases would be MySQL for traditional needs and it's huge community support. I would choose Microsoft SQL server if the application relies on Microsoft stack heavily.

Key-Value Datastores

When my project requirements demand me to come up with a persistence layer that will allow super-fast retrievals and high write throughputs, and the provision to store various types of data, My choice will be a Key-Value store. Key value stores can store any kind of data as the value and assign a key to that value using which we can perform any kind of data operations on it. For example, I am planning to apply a visitor counter to a website that I will be building soon. I plan to use DynamoDB as the key-value store for this. Upon users request to open my website in their browser, my application logic is programmed to hit my Key value database to increase the existing visitor count in the Key value store and retrieve it back to the website through my

API. I plan to store visitorID as the key and the number of times the website is opened as its value. Each time the website is opened or upon refreshing, I plan to make the API hit my database, update the old value for my visitor key, and quickly retrieve it. A key value datastore can help me achieve all of this happening within a fraction of a second. Redis key-value store is my second option to achieve this.

Document Databases

Document databases are useful when we need a NoSQL database that will support flexible schema, also to handle complex data structures. They can deal with both semi-structured and unstructured data and involve scenarios where the data schema may evolve from time to time. They can handle diverse and dynamic data structures, thus being adaptable for a variety of applications. If I have to write a mobile or web application to handle different kinds of data from users like text, images, and videos, with unique attributes, Document databases can store these without enforcing a rigid schema.

Additionally, If I have to choose a NoSQL database with the ability to handle complex queries through the internal structure of the document, the document database fits right in. Document databases offer the flexibility to add or remove fields within documents without affecting the entire database. Document databases are also known to be used for denormalized data due to their support for nested structures. This supports the storage of complex, hierarchical data within a single document.

I would choose MongoDB for this, as it has advanced features like auto sharding, horizontal scaling abilities, supports indexing and a strong querying language. Other options I would consider would be CouchDB, or Firebase for its serverless architecture and fully managed service.

Column-Family Databases

Column family database imposes structure on the aggregates residing in database to an extent in form of columns and column families, which can be taken as data units within the row aggregate present in the database. Each row can have a different number of columns, and this offers schema flexibility. Since columns are grouped together as column families, when we target specific column families to read from, it is faster and more efficient. Since clustering is peer-to-peer, it is easily scalable.

If I have to build a counter using my database, I would most likely choose a column-family database. By choosing this option, I can just target the specific column in a row where instead of accessing and updating the whole row to increment the count. This can be useful for any event logging or monitoring user interactions like page clicks too. I would choose Apache Cassandra or Google H Base, if going with a column-family database. Also, the atomic updates offered at single row level ensures that the updates are done without affecting any other operations.

Graph Databases

In my opinion, Among all the NoSQL models, Graph databases stand apart. Graph model supports ACID compliance and is good for not so large datasets with complex relations. A graph database typically contains

nodes and edges depicting the relations between those nodes, which together form a graph. We traverse through this graph to make connections between nodes. Because of these graph traversals, it usually gets trickier to do data sharding. So, it is an usual convention to do sharding at application level. However, there are distributed graph databases like Amazon Neptune which will take care of partitioning and clustering for us.

If I have to build a social media application with user recommendations based on the feed they interact with, graph databases will be my go to choice. Imagine your friend posting some activity on social media, and you liking it. Your friend, the activity, and yourself will be treated as nodes here. You liking his activity, and your friend posting it will be edges depicting the relationships with activity itself. Based on graph traversal, I can suggest you similar activities in your feed with my algorithm. Amazon Neptune and OrientDB are the graph database options I would consider for a recommendation engine or location based services etc.

Elevator Pitch

In my Advanced Data Management course, I explored the data and database management techniques deeply including advanced SQL techniques to handle large-scale databases where we deal with data security, replication, and clustering and address concurrent data access challenges. The course had a strong emphasis on the learning-by-doing approach. We built a production scale point of sale database system using the infrastructure hosted on Amazon web services, cleaned and loaded raw data to the database, created caching solutions, then focused on data security to prevent SQL injection attacks, and finally developed solutions to get real-time updates in database tables affected by changes in related tables using database triggers, before moving on to creating replication and clustering strategies. I have learned the strategies and methods to split and couple data for better performance through sharding and clustering. Apart from purely technical and theoretical concepts, this course taught me how to strategize data management at an organizational level. I learned every organization and application has a unique need, and there is no such thing as the perfect data management solution, it all depends on the application, business, and user needs. I have seen how different tools or persistence layers can be mixed and matched to make the best solution possible for a business. As I enter the professional world, I believe in my capabilities to optimize and build data solutions in a secure way, making me a valuable asset to organizations looking for experts in advanced data management.