Steven Moody

CSC 263

**Question 1. (10 pts) Compare and contrast the following three unique approaches to database management being leveraged by today's organizations.**

1. Relational databases
2. Memory Cache
3. NoSQL databases

Just to begin this rather large final essay, databases in general have been created and widely used as an alternative to traditional file processing as a means of storing and querying data much more efficiently and securely than ever before. While excel may be a plausible way of storing data similarly to a database, it is also much slower and requires much more manipulation than databases of any kind. The two primary database types that I am aware of are Relational databases and NoSQL databases. I am truthfully not sure what you mean by memory caching but I will do my best to attempt to explain that.

Relational databases are databases that exist in terms of relations. What I mean by this is that all data is somehow related within the database using a variety of rules and constraints to guide how exactly this is to occur. Within this database approach we have terms such as entities, attributes, and relationships to help define how all data is structured. Entities are the objects or primary topics that each table consists of (for example in a school database, an entity may be Students). Then each of these entities has one or more attributes that is within it to make up the columns of the entity table (such as a students first/last name etc.). An interesting fact about these attributes is that they can hold one or more values and also be used within other entities. Then, we also have relationships, these are defined words or terms that relate entities to each other and can also hold attributes (Such as the relationship of “take” in which a student entity is related to a class entity because they “take” a class). Not within these tables, there needs to be some sort of identifier that links all the attributes within that entity table to eachother in a way that would not allow for duplicate entries. This is known as a Primary Key. What a primary key does/is is a key that we as users can use to uniquely identify each entry within a table and distinguish it from another. They primary keys can consist of one value, multiple values, be a composite value of two other keys, or even be a primary key from another entity (we call this a foreign key). To start in the creation of a Relational database, we need to first create what is called an ERD (Entity Relationship Diagram). What this does is use a set of visual items to display how each piece of data relates within a database from a visual perspective. ERDs use certain visual graphics/shapes/fonts/etc. to show all entities within a relational database, all attributes that these entities have, each entities primary key, each relationship between two or more entities, as well as what we call the cardinality ratio between two entities. A cardinality ratio is a quantifiable number that defines how two entities interact. An example of this is that we know that many a student can take many classes and that a class can have many students. This means that within our defined relationship between the entities of Student and Class there is a MANY to MANY ratio. The different ratios that are possible are: 1 to 1, 1 to many, many to 1, and many to many. One final note here is that there are two ways to complete an ERD, the first is chen notation with is what I described above and the other is known as Crowe’s foot which also includes any foreign keys and is arranged using boxes similarly shaped to those use in a UML. Both display the same information but one is more visually representing while the other is more of logical/scientific showing.

Once we have an ERD for our database with all of its contents, we then have to develop a schema. A schema is sort of a list of each entity in table form with its attributes with arrows points to other tables that those attributes are within as well. This gives a logical breakdown of how each entity interacts with other entities within a database and also shows all of the tables that will be created and manipulated within a database. An important note here is that there may be more tables created than entities of an ERD, this is because certain relationships and cardinality ratios may require that additional attributes and tables be used to ensure relational functionality. This schema will also help define what constraints need to exist within a database to which I will get to now.

Once a schema is created and we have our ERD in hand, we can then go about scripting our database creation. The language we will be using is known as SQL and is divided into two categories, the first is known as DDL or Data Definition Language. This category includes keywords that are used to create(define) all of the tables within a database as well as the constraints within them. The second category is known as DML or Data Manipulation Language and is primary used to make changes to the database or the contents within it. For now, we will talk about DDL as that is the category we will be using at this stage to define our database and its constraints. We use DDL here to create each table, as well as each tables attributes along with a description of the data type that an attribute will hold, any length constraints, and also any other constraints on that specific attribute as well as the table itself. Constraints can be broken down into the following categories: Domain constraints, entity constraints, and referential constraints. Domain constraints are those that define what each attribute can hold as a data type as well as length. For example, if we were to set the STUDENT attribute of firstName to only except letters and that the length of the character string could only be 10 characters long, this would be an example of a domain constraint. An entity constraint is one that determines the rules of an entity itself. An example of this would be setting the attribute of Student Id within the Student entity to be the primary key for that table. Referential Constraints are those that define how references to objects within the database are defined and used. An example of this is a foreign key constraint which defines a key of a table to be the primary key from another table and is used as a reference.

Once all of that is complete, Congratulations you have just created a relational database, now use DML to make modification, as necessary. The advantages of using relational databases include: less redundancy, an increase in security and compartmentalization over traditional file processing, the ability to scale much more easily than traditional methods, as well as an overall increase in usability by end-state users. However, this approach may not be ideal if you are dealing with very precise, small projects that are time sensitive and do not require the complexity or overhead that relational databases have.

Moving to the other primary type of database I am familiar with, NoSQL (Not only SQL) or as I knew it before this class (Non SQL) databases are another very viable option when deciding what database type to use or even if you should be using a database. NoSQL database if I am not mistaken are actually more widely used that relational databases at this point because of what they offer and how easily they are implemented.

NoSQL databases follow the same steps as Relational Databases, however within each step different representations exist. For example, while a NoSQL database can absolutely store relational information it does not have to. So the ERD portion of this is generally replaced by a lit of the information to be stores within the database. Second, the schema is very different as a NoSQL database does not necessarily store data on different tables but this is not necessarily the case for NoSQL, NoSQL databases store all of the information about an object in the same place rather than separating it into various tables. All of this sounds quite enticing, as it is. However, NoSQL databases have a few pros as well as cons. First, NoSQL databases generally scale horizontally rather than vertically if I remember our reading correctly. This means that essentially when scaling is needed companies will just buy more servers to do so rather than increases the performance of the servers that exist and this can be costly. However, NoSQL is generally less intensive with overhead and also offers more efficient data retrieval than traditional relational databases and MUCH more efficient than memory caching. Overall, NoSQL has been slowly overtaking the market when it comes to database types and implementations every since the creation of the modern internet and while the approach is similar in some ways, in others a NoSQL offers a very viable solution over the lackluster relational database model and especially over memory caching for any major company.

In conclusion, when considering what database type to use we first have to consider if it is even worth it to use a database. The overhead and cost associated with any database is going to be higher than traditional methods and if the product is relatively small or time sensitive it may make more sense to use traditional methods. However, when dealing with most modern data storage tables, it is ideal and even encouraged as well as some people even stating necessary to use one of the database types I discussed here.