20042768

Data Management Fundamentals

UFCFLR-15-M

DMF Exam

Answer ALL questions in this section.

Consider the following flat file database:

cust-id	**name**	**address**	**product-id**	**desc**	**quantity**	**price**	**order-id**	**order-date**
12	Mr Joe Bloggs	5 Muller Rd ..	75	widget	7	14.00	34	01/3/10
12	Mr Joe Bloggs	5 Muller Rd ..	12	nibbet	3	9.00	34	01/3/10
13	Mr Harry Hill	43 Dry Av..	75	widget	4	8.00	36	01/03/10
13	Mr Harry Hill	43 Dry Av..	23	gibbet	2	3.50	36	01/03/10
14	Ms Jean Grey	34 Tree Hill…	12	nibbet	5	12.50	43	03/03/10

Note: Only 5 records are shown and the address field is concatenated.

QUESTION 1

Explain the major problems associated with holding and using data in this format.

(12 marks)

The major problem with holding data in this format is that it does not comply with the relational database fundamentals. Beginning with the point of holding data in the first place; the reason that relational databases exist is to later retrieve the data in the most efficient way possible.
Here this is not observed due to the lack of normalization. With no normalization there is no referential integrity, and no relationship rules. In other words, I can alter any ID in this table, and there would be no constraints to stop the occurrence. This is where the idea of creating separate tables and joining through relationships comes into play. For example, if product ID was a primary key in a separate table, but referenced back to the customer ID table as a foreign key creating a one to many relationship, this would prevent one from altering freely or even accidentally. This helps to keep the database consistent and accurate. The foreign keys create referential constraints and make the table to do the work in preventing, as we do not want to worry about having bad data in a table.
The first normal form states that values in each cell should be atomic and tables should have no repeating groups. This is violated with the first four rows as the customer ID is referenced as duplicates. This would create issues in the future when we come back to query based on Mr. Joe Bloggs or Harry Hill. How do we know which record we are looking for because there is nothing to uniquely identify these people.
Second notational form states comes into the conversation when one key is dependent on a portion of a composite key. All values in the table that are not a key must fully rely on the whole

primary or composite key. In this case there are no composite keys present, so this is not an issue.

Third notational form states that values should not be stored if they can be calculated from another non-key field. This applies to all columns that relate to product ID, and order ID. These ID's are not considered primary keys in the table and their attribute fields are dependent upon them. Hence the need to create another table. In addition, the redundancy of the data causes issues with cascading deletes. If we delete a specific piece, it causes the table to delete all records associated within that table. Last but not least this table does not comply with the acronym ACID, atomic, consistency, isolation, and durability.

QUESTION 2

Identify and explain the major advantages a RDMS offers over a flat-file database when holding and processing large amounts of data.

(12 marks)

The major advantage that a RDBMS offers over a flat-file database begins with the normalization seen above. With that nature of normalization comes relational benefits. There are no longer duplicates of data, and all records are uniquely identified. Indexing or assessing relationships in a flat file database is impossible because there is no organization or specific arrangement. Relational databases follow a specific schema which then works in unison with SQL. When we process a large amount of data we often want to group it, analyze it, and make sense of it. This is extremely limited in a flat-file database, whereas a relational model can make specific joins, and aggregates from the structured query language. When we have a relational database we can also backup our database to the cloud. Then it can be accessed, and updated

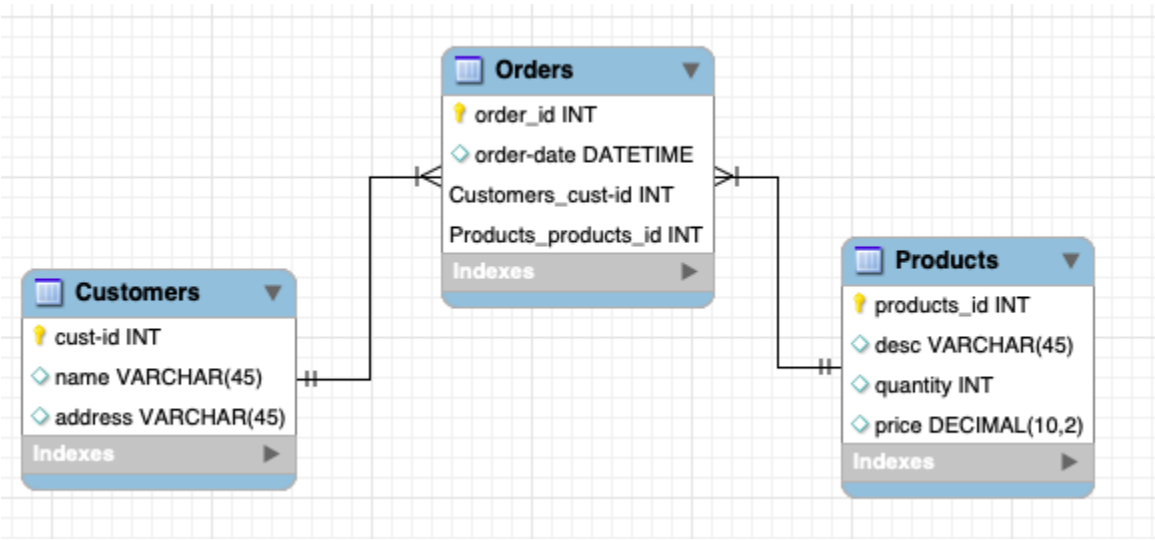
from any platform. This leads me to my next point of updating a RDBMS at the same time. This would violate the ACID properties, and each update would take place one at a time which maintains database integrity. The relational model can also be vertically scaled so that read and write queries are performing at top speed. Compared to a flat-file database, the relations are implicitly present in relational databases. These act as another contributor towards top speed queries, as well as indexing. In addition, another high level feature of a relational database are triggers. For example, one can create an automated trigger that sends an email every time a customer is added to the database. In a flat-file database it simply holds the information.

QUESTION 3

Analyse & normalize the above data to 3NF, create any entities (including implied entities) and draw a low-level ER diagram showing all entities, attributes, keys and foreign keys (these must be clearly marked).

NOTE: You do need to show your Normalization steps but only the final E-R model.

(16 marks)



Section B

Q2

a) Identify and briefly explain the *four* different types of NoSQL data stores, giving examples of typical usage and available products.

(16 marks)

- Document

  In comparison to key-value storage, a document database may draw similarities. The way that document databases store data is through structured key-value pairs such as XML or JSON formats. In other words, they are the evolution of simple key-value databases. The main benefit of these documents is increased query performance by grouping schema-less data into collections. In comparison to SQL documents, collections are that of a table and documents are that of a record. To identify a specific document database we can look at MongoDB. MongoDB stores all of its data in a JSON like format, and then shards data horizontally across multiple database servers. The benefit of sharding boasts faster read and write queries. Put simply, data is classified further into servers, and within those servers the data is classified even further. To combat the question of a fixed amount of servers, we classify the data even further. This is known as hierarchical sharding, for it increases flexibility by kicking proper requests to a manager who then directs the request accordingly. Because of this process MongoDB is ideal for real time insertion, and can handle spatial data much better than a RDBMS. Another key benefit of MongoDB is that it supports replications and failure recovery through master slave architecture. All write

requests are sent to the master server, and the slave servers read from the master. In the case that the master server were to fail, the slaves would choose a master amongst themselves, and the process would continue. MongoDB is an ACID compliant document database that is a top of the line option for real time unstructured data. Document databases can be an ideal option if one is looking to move away from relations.

- Columnar

  Column oriented databases are specific to columns in the way that they query for information. For this same reason they are great at compressing, because within each column they all maintain similarity. Columnar databases are typically used in analytics, because they have great usage in aggregating data (OLAP). Also querying on columns allows for optimized retrieval, inherently reducing our input output requirements in the amount of data that we are loading. For a specific example of a columnar database we can look at the architecture of Apache Cassandra. Cassandra works in clustering. Therefore the data it receives may go to multiple server nodes. For example, when someone makes a request in the form of an ID it must first be hashed. Once that request is hashed it is designated to a specific node, where the following node will also replicate the data. In this way, the data is also horizontally scaled where if one of our server nodes fails, we can still read and write queries from the following server. In the case of a server node failing, the way that the other server nodes come to a conclusion about a particular data instance is known as quorum. Quorum ensures that we maintain integrity when returning queries to the end user. The advanced architecture paired with fast specific

column queries marks columnar databases the preferred analytical database when storing and aggregating big data.

- Graph

  A graph database capitalizes on the blemishes of relational databases in terms of deep relationships. In its base structure, a graph database still follows key values relationships, but they are known as nodes that are connected through relationships or edges. The nodes may have many attributes inside them, and then are connected through relational verbs. For example, person 10 is connected to person 22 because they took the same survey. Conceptually speaking it is a much simpler model regarding far out relationships than a relational database model. What is found today, is that the typical usage of a graph database is found as an additional component to a RDBMS. A graph database may spark connections, and create relationships that would have otherwise been unknown by a relational model. A concrete example of the most popular graph database is Neo4J. As mentioned above, the main difference between a RDBMS and a graph is the underlying architecture being highly connected. Whereas doing a query to find a deep relationship with joins and indexes takes a lot of power with relational models. The edges and entities that connect nodes in Neo4J allow the user to make advanced queries in a couple lines of code. Just as other database models, Neo4J shards its data across multiple database servers, and replicates data on the basis of requests. The company horizontally scales their data across multiple servers to maintain peak efficiency. A great example where a

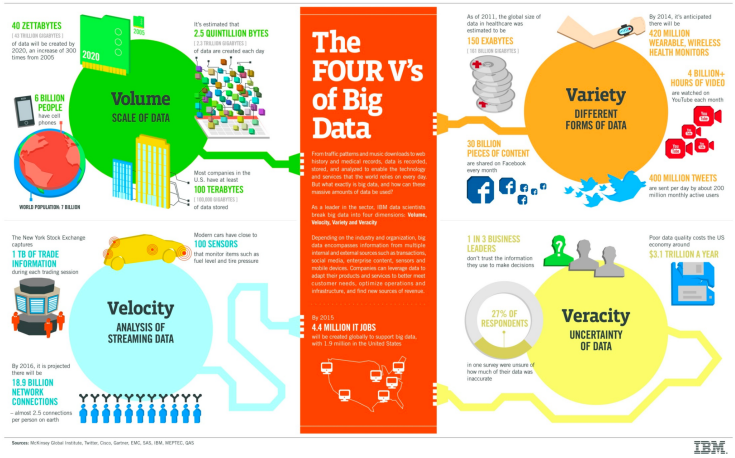
graph database would do well is social media. As posts, hashtags, replies, quotes, mentions, and re-posts all are related to each other in one way or another.

- Key-Value

  A key-value database maintains simplicity, and one can say that the database inputs are similar to that of a python dictionary. The way that data is input removes the need for underlying structured relations that utilize multiple tables. Rather the way that a key-value database maintains classification amongst its records is through the composite keys. A great modern example of a key-value database is Amazon's DynamoDB. As mentioned, the composite keys play a huge role in keeping records unique. These composites are created through a partition key and a sort key. The sort key is what classifies the records into their SQL comparison of "tables". While the partition key serves an explicit identifier in order to keep all inputs unique. The main benefits of Amazon's DynamoDB lie in its simplicity, but that is also its downfall. It is a cost-effective plug in and play that is ideal for a small or middle sized business that do not need to make advanced queries or table joins. The database runs on a horizontal scale, meaning that your data is replicated over multiple servers. The scalability is based on need, and realistically it is limitless. Also the functionality of a key-value is beneficial for caching and provides performance gains in that sense. In essence, it does the job of classifying, tracking, and grouping data. At the end of the day you know you are able to look up the value of something based on a unique key or grouping.

b) Explain how the key business drivers of *Volume*, *Velocity*, *Variability* and *Agility* are driving commercial and public service based organizations to adopt NoSQL based systems and solutions.

(14 marks)



In the past, utilizing RDBMS was a no brainer, in fact it was basically the only thing. In today's day and age it still has its place in the world, but it is all situational cases. Relational databases are essential for transactional systems, and will always maintain accuracy. Not only that, but they are built for joins across multiple tables, and merge relationships between data very well. However, in the modern world, we have modern situations. Data is fueled through volume, velocity, variety, and agility.

The amount of big data that commercial and public service based organizations create everyday is growing exponentially. As shown in week 1 slides, "Most companies in the U.S. have at least 100 Terabytes of data stored. In today's day and age, we ask what qualifies as the correct volume for big data? The classification of a Terabyte proves to be right on the cusp, as many organizations are now reaching the petabyte landmark. Not only do they have the proper volume of data, but it is also generating at a reputable velocity. The general speed at which data is created is known as the velocity. Usually volume and velocity go hand in hand, for you can not

have one without the other. The variety of data refers to the different types of information that is coming in on a daily basis due to the velocity. The agility of a database relies on the ability to absorb data in real time, and then turn around to make sense of it. A relational database would have to work extremely hard to access multiple tables and store accordingly. Many companies are now harvesting complex data, which is absolutely unstructured. RDBMS' can not keep up with the three V's plus agility, which is why we have a need for No SQL. No SQL does not require one to classify data, and put each specific item in columns. There is no predefined schema that the data must fit, rather this data is stored in document based systems, graph databases, and columnar databases. The best part is these transactions are stored all at one time. The data engineer does not have to be concerned about dividing the database into tables or identifying relationships. The demand to understand the volume, variety, variability, and agility on the fly has propelled No SQL forward.

The types of businesses that are using No SQL also have a need for scalability. With the increase in data the scale must move in correlation. Rather than having all information on a single server, they distribute their data in clusters across multiple servers, hence MongoDB, Apache Cassandra. This allows unlimited access from the cloud, and classifying big data in real time.

Question 7

a) With reference to the notion of "*Polyglot Persistence*", describe and explain the reasons why organizations in both the private and public sectors are taking up NoSQL based solutions to meet their data storage and processing needs.

(8 marks)

It is no mystery why organizations are utilizing Polyglot Persistence to make use of their data. It is simply because there is not a one platform solution that works best on all types of data. It all depends on classifying what is the best for the operation, that way we can get the most production possible, and choose something that aligns directly with our end goal. For example, if we were looking to find deep relationships with our customers and get some insight about their click patterns then graph databases would be the way to go. If we wanted to make classify transactions, make joins across tables, and enforce relationships across our database than RDBMS would be our go to. Or perhaps we would have a need to digest advanced complex data in real time. Well then No SQL would be our choice.

The idea is this: organizations need all of these things, so why not utilize the best tool for the job. Assuming that these sectors have the funds to utilize these tools it is a hole in one. If the tools were at your disposal to make sense of different types of data it would hold efficiency to a higher standard in these organizations.

b) There are a range of parameters used to measure the strength and weaknesses of NoSQL & SQL databases. Scalability is one of these measures. What is the difference between horizontal and vertical scaling in and of these databases?

(10 marks)

The comparison of buying bigger machines, verus buying more machines. Simply put, that is what we are talking about when it comes to vertical and horizontal scaling. If you think about it, vertical scaling has always been around since storage came about. This type of scaling is unique in that it utilizes inter process communication, meaning that the computer is only relying on itself to pull data. Because of this reason it is fast, and cache consistent. The immediate downfall of this vertical scaling is that it has a single point of failure. If that computer or server goes down that is it. Where else are you going to be able to get the information from. This is where the idea of horizontal scaling came in. With horizontal scaling spreading out, and replicating information across multiple servers, in a sense the scaling becomes uncapped. If memory was ever an issue, the solution would be to just get another computer. However, there are downfalls to this horizontal scaling, and this is typically what we see with No SQL databases. The RPC's or remote procedure calls work across the network in order to query for information. This can be slow in comparison to inter-process communication. Another issue and difference that horizontal scaling may face is data inconsistency. As data is passed along from server to server, there is potential for inconsistency. However, that same weakness is also a strength because if one server goes down, we are able to reference other servers as backup.

c) You are creating a web-based interface to allow for the ad-hoc querying and reporting on a large (and growing) collection of various types of customer invoice, where the main key is an invoice ID (but the customers and products purchased are also of high interest). With reference to key-value, document and graph store NoSQL systems, evaluate each as a potential persistence layer for the application.

Beginning with the base key-value for this application it can get the job done, but not at the level that we want it done. Sure the key-value database would be able to classify our data for us, but it would not be able to make advance queries or table joins amongst the data that we want. The main question with the document database becomes what is the velocity and variety that this data is coming in at. If it is true big data, where we are talking terabytes then the document database is a no-brainer here. We have the ability to store all of this information regarding customers and products in single transactions. We do not have to stress about splitting the database into multiple tables, querying relationships, or following explicit schemas. In a document styled database we would have a cluster friendly platform, with the ability to store scale horizontally across a countless amount of servers. Our data can be replicated, and accessed with ease. In regards to the graph store, I do not believe it would be the best fit in this instance. The two things that we are piecing together are small in the scope of a graph database. We have customer invoices, customers, and products. Yes a graph database would be able to get the job done, but it is best when looking for patterns, and underlying connections. In this case this does not seem like the primary goal. At the end of the day it would really come down the volume, velocity, variability, and agility of our data.