USN | | | | | | | | | | |

Internal Assessment Test 1 – May 2022

Sub:	NoSql database					Sub Code:	18CS823	Branch:	CSE
Date:	14/05/22	Duration:	90 mins	Max Marks:	50	Sem / Sec:	VIII/A,B,C		OBE

Answer any 5 FIVE FULL Questions

		MARKS	CO	RBT
1	What is the version stamp? What are the ways to create version stamps for single server and peer to peer architecture? Version stamp - 1 marks Ways to create version stamp in single server model - 5 marks Ways to create version stamp in distributed model-4 marks	[10]	CO1	L2
2	Describe i) Impedance mismatch with suitable examples ii) Graph database iii) CAP theorem Impedance mismatch with suitable examples- 3 marks Graph database and diagram-4 marks CAP theorem-3 marks	[10]	CO1	L2
3	Summarize the ways to relax consistency. Explain the trade-offs in various distribution models after relaxing consistency with examples. (4+6 marks) Ways to relax consistency-4 marks Various tradeoffs-6 marks	[10]	CO1	L2
4	What do you mean by write-write conflict/consistency and read-write consistency? Explain with suitable examples. write-write consistency with example - 3+2 marks read-write consistency with example- 3+2 marks	[10]	CO1	L2
5	Discuss advantages and disadvantages of Schemalessness? Explain Materialised views. advantages and disadvantages of Schemalessness- 5 marks Materialised view- 5 marks	[10]	CO1	L2
6	Assume an organization needs to create an online platform for its employees to create posts and add various images, videos, audio, and comments. Any employee can comment on these posts and rate them. Employees can join different groups as per their interests. The landing page will have a feed of posts that employees can share and interact with. For the given scenario, suggest which type of database implementation (SQL / NoSQL) would be most suitable and specify appropriate reasons for your choice of the database implementation. (tip: write sample table names to show connections) Type of database implementation - 2 marks Specify appropriate reason- 7 marks	[10]	CO2	L4

1. What is the version stamp? What are the ways to create version stamps for single server and peer to peer architecture?

   Version stamps help you detect concurrency conflicts. When you read data, then update it, you can check the version stamp to ensure nobody updated the data between your read and write.
   • Version stamps can be implemented using counters, GUIDs, content hashes, timestamps, or a combination of these.
   • With distributed systems, a vector of version stamps allows you to detect when different nodes have conflicting updates.
   Ways to create version stamp:
   1. Using a counter, always incrementing it when you update the resource. Counters are useful since they make it easy to tell if one version is more recent than another. On the other hand, they require the server to generate the counter value,and also need a single master to ensure the counters aren't duplicated.
   2. Another approach is to create a GUID, a large random number that's guaranteed to be unique. These use some combination of dates, hardware information, and whatever other sources of randomness. The nice thing about GUIDs is that they can be generated by anyone and never get duplicated; a disadvantage is that they are large and can't be compared directly for recentness.
   3. The third approach is to make a hash of the contents of the resource. With a big enough hash key size, a content hash can be globally unique like a GUID and can also be generated by anyone; the advantage is that they are deterministic—anynode will generate the same content hash for the same resource data. However, like GUIDs they can't be directly compared for recentness, and they can be lengthy.
   4. A fourth approach is to use the timestamp of the last update. Like counters, they are reasonably short and can be directly compared for recentness, yet have the advantage of not needing a single master. Multiple machines can generate timestamps—but to work properly, their clocks have to be kept in sync. There's also a danger that if the timestamp is too granular you can get duplicates—it's no good using timestamps of a millisecond precision if you get many updates per millisecond.
   5. You can blend the advantages of these different version stamp schemes by using more than one of them to create a composite stamp. For example, CouchDB uses a combination of counter and content hash.

   Assume an organization needs to create an online platform for its employees to create posts and add various images, videos, audio, and comments. Any employee can comment on these posts and rate them. Employees can join different groups as per their interests. The landing page will have a feed of posts that employees can share and interact with.
   For the given scenario, suggest which type of database implementation (SQL / NoSQL) would be most suitable and specify appropriate reasons for your choice of the database implementation.

2. Describe i) Impedance mismatch with suitable examples ii) Graph database iii) CAP theorem

Impedance Mismatch

● For application developers, the biggest frustration has been what's commonly called the impedance mismatch: the difference between the relational model and the in-memory data structures.

● The relational data model organizes data into a structure of tables and rows, or more properly, relations and tuples.

● In the relational model, a tuple is a set of name-value pairs and a relation is a set of tuples. (The relational definition of a tuple is slightly different from that in mathematics and many programming languages with a tuple data type, where a tuple is a sequence of values.)

● All operations in SQL consume and return relations, which leads to the mathematically elegant relational algebra.

● This foundation on relations provides a certain elegance and simplicity, but it also introduces limitations.

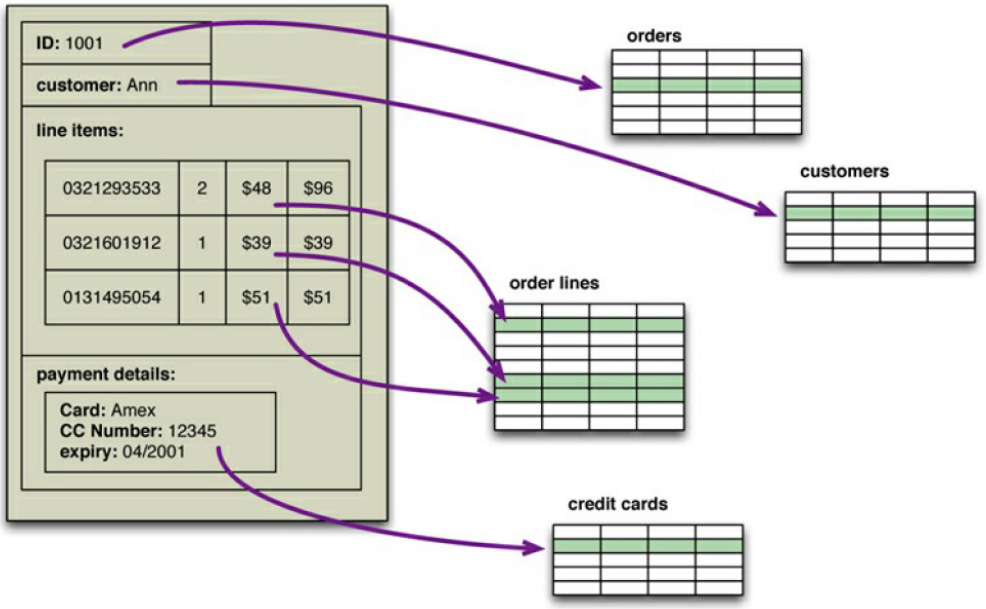
In particular, the values in a relational tuple have to be simple—they cannot contain any structure, such as a nested record or a list. This limitation isn't true for in-memory data structures, which can take on much richer structures than relations.

● As a result, if you want to use a richer in memory data structure, you have to translate it to a relational representation to store it on disk. Hence the impedance mismatch—two different representations that require translation.

● Impedance mismatch has been made much easier to deal with by the wide availability of object relational mapping frameworks, such as Hibernate and iBATIS that implement well-known mapping patterns [Fowler PoEAA], but the mapping problem is still an issue.

● Object-relational mapping frameworks remove a lot of grunt work, but could not solve problem of impedance mismatch

● Relational databases continued to dominate the enterprise computing world



ii) Graph database

• A characteristic of graph is high flexibility.

• Any number of nodes and any number of edges can be added to expand a graph.

• The complexity is high and the performance is variable with scalability.

• Data store as series of interconnected nodes.

• Graph with data nodes interconnected provides one of the best database system when

relationships and relationship types have critical values**.**

This is ideal for capturing any data consisting of complex relationships such as social networks, product preferences, eligibility rules

● Once you have built up a graph of nodes and edges, a graph database allows you to query that network with query operations designed with this kind of graph in mind.

● Doing same with lot of joins in RDBMS is very expensive process

● Graph databases make traversal along the relationships very cheap.

This is majorly because graph databases shift most of the work of navigating relationships from query time to insert time.

● Ideal for situations where querying performance is more important than insert speed.

● The emphasis on relationships makes graph databases very different from aggregate-oriented databases.

● Databases are more likely to run on a single server rather than distributed across clusters.

● ACID transactions need to cover multiple nodes and edges to maintain consistency.

● The only thing they have in common with aggregate-oriented databases is their rejection of the relational model.

iii) CAP Theorem

• Any two properties must be satisfied

• Consistency means all copies have the same value like in traditional DBs.

• Availability means at least one copy is available in case a partition becomes inactive or fails. For example, in web applications, the other copy in the other partition is available.

• Partition means parts which are active but may not cooperate (share) as in distributed DBs.

1.Consistency in distributed databases

• All nodes observe the same data at the same time. Therefore, the operations in one partition of the database should reflect in other related partitions in case of distributed database.

• Operations, which change the sales data from a specific showroom in a table should also reflect in changes in related tables which are using that sales data.

2.Availability

• Availability means that during the transactions, the field values must be available in other partitions of the database so that each request receives a response on success as well as failure. (Failure causes the response to request from the replicate of data).

• Distributed databases require transparency between one another.

• Network failure may lead to data unavailability in a certain partition in case of no replication.

• Replication ensures availability.

3.Partition

• Partition means division of a large database into different databases without affecting the operations on them by adopting specified procedures

3. Summarize the ways to relax consistency. Explain the trade-offs in various distribution models after relaxing consistency with examples

Relaxing Consistency : Consistency is a Good Thing—but,it comes with sacrifices.

●It is always possible to design a system to avoid inconsistencies, but often impossible to do so without making unbearable sacrifices in other characteristics of the system.

●As a result, we often have to tradeoff consistency for something else.

●Furthermore, different domains have different tolerances for inconsistency, and we need to take this tolerance into account as we make our decisions.

● **Trading off consistency is a familiar concept even in single-server relational database systems.** Here, our principal tool **to enforce consistency is the transaction, and transactions can provide strong consistency guarantees.**

● However, **transaction systems usually come with the ability to relax isolation levels, allowing queries to read data that hasn't been committed yet.**

● In practice we see **most applications relax consistency down from the highest isolation level (serialized) in order to get effective performance**.

● We most commonly see people u**sing the read-committed transaction level, which eliminates some read-write conflicts but allows others**.

●Many systems forgo transactions entirely because the performance impact of transactions is too high.

On a small scale, **we saw the popularity of MySQL during the days when it didn't support transactions. Many websites liked the high speed of MySQL and were prepared to live without transactions.**

● At the other end of the scale, **some very large websites, such as eBay, have had to forgo transactions in order to perform acceptably**— this is particularly true when you need to introduce sharding.

● Even without these constraints, many application builders **need to interact** with **remote systems that are outside a transaction boundary,** so updating outside of transactions is a quite common occurrence for enterprise applications.

4. What do you mean by write-write conflict/consistency and read-write consistency? Explain with suitable examples.

Take an example of considering updating a telephone number. Coincidentally, Martin and Pramod are looking at the company website and notice that the phone number is out of date. **They both have update**

access, so they both go in at the same time to update the number. We'll assume **they update it using a slightly different format**. **This issue is called a write-write conflict: two people updating the same data item at the sametime**.

When the writes reach the server, the server will serialize them—decide to apply one,then the other. Let's assume it uses alphabetical order and picks Martin's update first,then Pramod's. **Without any concurrency control,** Martin's update would be applied and immediately overwritten by Pramod's. In this case **Martin's is a lost update**. We see this as a failure of consistency because Pramod's **update was based on the state** beforeMartin's update, yet was applied after it.

Having a data store that maintains update consistency is one thing, but it doesn't guarantee that readers of that data store will always get consistent responses to their requests.

● Let's imagine we have an order with line items and a shipping charge. The Shipping charge is calculated based on the line items in the order. If we add a line item, we thus also need to recalculate and update the shipping charge. In a relational database, the shipping charge and line items will be in separate tables.

● The danger of inconsistency is that Martin adds a line item to his order, Pramodthen reads the line items and shipping charge, and then Martin updates the shipping charge. This is an inconsistent read or read-write conflict: Pramod has done a read in the middle of Martin's write.

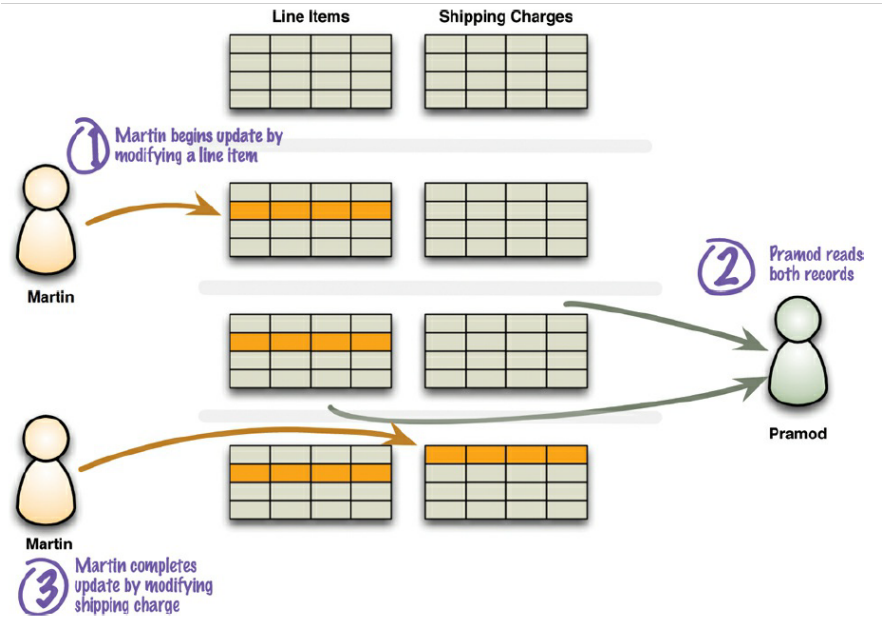


Figure 5.1. A read-write conflict in logical consistency

\

5. Discuss advantages and disadvantages of Schemalessness? Explain Materialised views.

Schemaless Databases

● A common theme across all the forms of NoSQL databases is that they are schemaless.

● In a relational database, first have to define a schema—a defined structure for the database which says what tables,which columns exist, and what data types each column can hold.

● Before you store some data, you have to have the schema defined for it.

● With NoSQL databases, storing data is much more casual. A key-value store allows you to store any data you like under a key.

● A document database effectively does the same thing, since it makes no restrictions on the structure of the documents you store.

● Column-family databases allow you to store any data under any column you like.Graph databases allow you to freely add new edges and freely add properties to nodes and edges as you wish.

Advocates of schemaless is freedom and flexibility.

With a schema, you have to figure out in advance what you need to store, but that can be hard to do.

Without a schema binding you, you can easily store whatever you need. This Allows you to easily change your data storage as you learn more about your project. You can easily add new things as you discover them.

Furthermore, if you find you don’t need some things anymore, you can just stop storing them, without worrying about losing old data as you would if you delete columns in a relational schema.

Schemaless store also makes it easier to deal with nonuniform data: data where each record has a different set of fields.

Problems of Schemalessness

Schemalessness is appealing, but it brings some problems of its own.

If all you are doing is storing some data and displaying it in a report as a simple list of fieldName:value lines then a schema is only going to get in the way.

Fact is that whenever we write a program that accesses data, that program almost always relies on some form of implicit schema.

This implicit schema is a set of assumptions about the data’s structure in the code that manipulates the data. Having the implicit schema in the application code results in some problems. It means that in order to understand what data is present you have to dig into the application code.

If that code is well structured you should be able to find a clear place from which to deduce the schema. But there are no guarantees; it all depends on how clear the application code is.

Furthermore, the database remains can't use the schema to help it decide how to store and retrieve data efficiently. It can't apply its own validations upon that data to ensure that different applications don't manipulate data in an inconsistent way.

Schemaless DB: shifting of schema to code

Essentially, a schemaless database shifts the schema into the application code that accesses it. This becomes problematic if multiple applications, developed by different people, access the same database.

Materialized Views

● In aggregate-oriented data models, if you want to access orders, it's useful to have all the data for an order contained in a single aggregate that can be stored and accessed as a unit.

● In scenarios where if a product manager wants to know how much a particular item has sold over the last couple of weeks? Aggregate oriented data model force to check all the orders in that aggregate, indexing on product works but works against aggregate structure.

● In relational db can access data in different ways using views views. A view is like a relational table (it is a relation) but it's defined by computation over the base tables.

● Views provide a mechanism to hide from the client whether data is derived data orbase data—but can't avoid the fact that some views are expensive to compute.

● To cope with this, materialized views were invented, which are views that are computed in advance and cached on disk. Materialized views are effective for data that is read heavily but can stand being somewhat stale.

● Although NoSQL databases don't have views, they may have precomputed and cached queries, and they reuse the term "materialized view" to describe them.

● It's also much more of a central aspect for aggregate-oriented databases than it is for relational systems. Eg MapReduce

There are two rough strategies to building a materialized view.

❏ The first is the eager approach where you update the materialized view at the same time you update the base data for it.

❏ In this case, adding an order would also update the purchase history aggregates for each product.

❏ This approach is good when you have more frequent reads of the materialized view than you have writes and you want the materialized views to be as fresh as possible.

❏ The application database approach is valuable here as it makes it easier to ensure that any updates to base data also update materialized views.

Second approach: If you don't want to pay that overhead on each update,you can run batch jobs to update the materialized views at regular intervals.

● You'll need to understand your business requirements to assess how stable your materialized views can be.

● You can build materialized views outside of the database by reading the data,computing the view, and saving it back to the database.

● More often databases will support building materialized views themselves.

● In this case, you provide the computation that needs to be done, and the database executes the computation when needed according to some parameters that you configure.

● This is particularly handy for eager updates of views with incrementalmap-reduce.

● Materialized views can be used within the same aggregate. An order document might include an order summary element that provides summary information about the order so that a query for an order summary does not have to transfer the entire order document.

● Using different column families for materialized views is a common feature of column-family databases.

● An advantage of doing this is that it allows you to update the materialized view within the same atomic operation.

6. Assume an organization needs to create an online platform for its employees to create posts and add various images, videos, audio, and comments. Any employee can comment on these posts and rate them. Employees can join different groups as per their interests. The landing page will have a feed of posts that employees can share and interact with.

   For the given scenario, suggest which type of database implementation (SQL / NoSQL) would be most suitable and specify appropriate reasons for your choice of the database implementation

   solution:

   Best suitable database implementation would be NoSQL databases.

   Following are the reasons:

   For the given scenario, there are several entities such as employee, post, comments, images, audios, videos, rating and so on.

You have several relationships that link these entities as shown in the table below:

Entity	Relationship	Entity
post	*written by*	employee
post	*with many*	comments
comments	*created by*	employee
rating	*assigned by*	employee
post	*with many*	rating
post	*with many*	images

Use NoSQL database implementation because,

NoSQL databases are very flexible in structure and can store all types of related data in one place

User can retrieve the whole post with a single query avoiding joins thus increasing the performance

Data on NoSQL databases scale out naturally and hence able to deal with the continuous streaming of posts

Using SQL databases is not suggested as,

Several joins are used to display the post containing various forms of data which is very time consuming

Data is existing in heterogeneous forms that SQL does not support

Continuous streaming of posts that are dynamically loaded onto the screen require thousands of queries to be performed