USN

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 2 question paper – April 2024

Sub:	NoSQL database			Sub Code:	18CS823	Branch:	CSE	
Date:	13/4/2024	Duration:	90 mins	Max Marks: 50	Sem/Sec:	VIII/A,B,C		OBE

	Answer any 5 FIVE FULL Questions	MARKS	CO	RBT
1	Explain incremental map reduce process in detail. Explain how scaling is handled in key-value store with example. Incremental Map-Reduce process -5 marks scaling in key-value store-5 marks	[10]	CO2	L2
2	Describe Map-reduce process to compare the sales of products for each month in 2011 to the prior year. Use suitable diagrams. Map-reduce process with description -8 marks Diagrams-2 marks	[10]	CO2	L3
3	Suppose we want to return all the documents in an order collection (all rows in the order table. Write down SQL and MongoDB queries for: i) Selecting the orders for a single customerId of 883c2c5b4e5b (3 marks) ii) Selecting orderId and orderDate for one customer (3 marks) iii) Query for all the orders where one of the items ordered has a name like Refactoring (2+2 marks)	[10]	CO3	L3
4	Explain key-value store with respect to consistency and transactions. Explain Bucket creation in key-value store. Consistency in key-value store, bucket creation – 3+3 marks Transaction processing in key-value store- 4 marks	[10]	CO3	L2
5	Identify the situations where document databases are i) applicable ii) not advisable. Justify your answer. document databases are applicable with justification - 5 marks Not advisable with justification -5 marks	[10]	CO3	L2
6	What is key-value store? Explain with an example. List and explain any two features of key-value store. key-value store with example- 3+3 marks two features of key-value store - 4 marks	[10]	CO3	L3

Scheme

1	Explain incremental map reduce process in detail. Explain how scaling is handled in key-value store with example. Incremental Map-Reduce process -5 marks scaling in key-value store-5 marks	[10]
2	Describe Map-reduce process to compare the sales of products for each month in 2011 to the prior year. Use suitable diagrams. Map-reduce process with description -8 marks Diagrams-2 marks	[10]
3	Suppose we want to return all the documents in an order collection (all rows in the order table. Write down SQL and MongoDB queries for: iv) Selecting the orders for a single customerId of 883c2c5b4e5b (3 marks) v) Selecting orderId and orderDate for one customer (3 marks) vi) Query for all the orders where one of the items ordered has a name like Refactoring (2+2 marks)	[10]
4	Explain key-value store with respect to consistency and transactions. Explain Bucket creation in key-value store. Consistency in key-value store, bucket creation – 3+3 marks Transaction processing in key-value store- 4 marks	[10]
5	Identify the situations where document databases are i) applicable ii) not advisable. Justify your answer. document databases are applicable with justification - 5 marks Not advisable with justification -5 marks	[10]
6	What is key-value store? Explain with an example. List and explain any two features of key-value store. key-value store with example- 3+3 marks two features of key-value store - 4 marks	[10]

Solution

Q1. Explain incremental map reduce process in detail. Explain how scaling is handled in key-value store with example.
ans:

Many map-reduce computations take a while to perform, even with clustered hardware, and new data keeps coming in which means we need to rerun the computation to keep the output up to date.

- Starting from scratch each time can take too long, so often it's useful to structure a map-reduce computation to allow incremental updates, so that only the minimum computation needs to be done.

- The map stages of a map-reduce are easy to handle incrementally—only if the input data changes does the mapper need to be rerun.

- Since maps are isolated from each other, incremental updates are straightforward.

- The more complex case is the reduce step, since it pulls together the outputs from many maps and any change in the map outputs could trigger a new reduction.

- This re-computation can be lessened depending on how parallel the reduce step is. If we are partitioning the data for reduction, then any partition that's unchanged does not need to be re-reduced.

- Similarly, if there's a combiner step, it doesn't need to be rerun if its source data hasn't changed.

- If our reducer is combinable, there's some more opportunities for computation avoidance.

- If the changes are additive—that is, if we are only adding new records but are not changing or deleting any old records—then we can just run the reduce with the existing result and the new additions.

- If there are destructive changes, that is updates and deletes, then we can avoid some recomputation by breaking up the reduce operation into steps and only recalculating those steps whose inputs have changed—essentially, using a Dependency Network [Fowler DSL] to organize the computation.

Q2. Describe Map-reduce process to compare the sales of products for each month in 2011 to the prior year. Use suitable diagrams
ans:

Consider an example where **we want to compare the sales of products for each month in 2011 to the prior year**. To do this, we'll break the calculations down into two stages.

- The first stage will produce records showing the aggregate figures for a single product in a single month of the year.
- The second stage then uses these as inputs and produces the result for a single product by comparing one month's results with the same month in the prior year (see Figure 7.8).

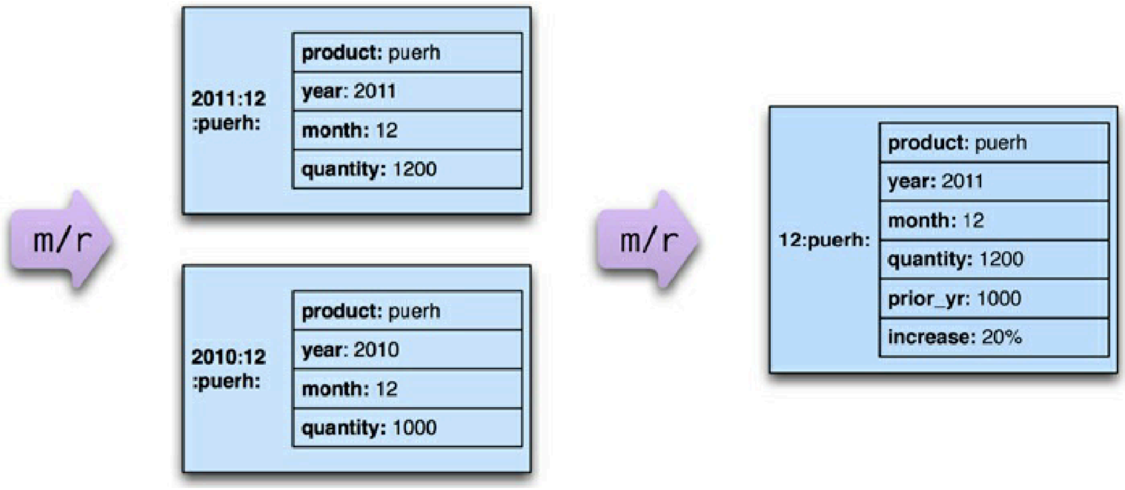

Figure 7.8. A calculation broken down into two map-reduce steps, which will be expanded in the next three figures

- A first stage (Figure 7.9) would read the original order records and output a series of key-value pairs for the sales of each product per month.

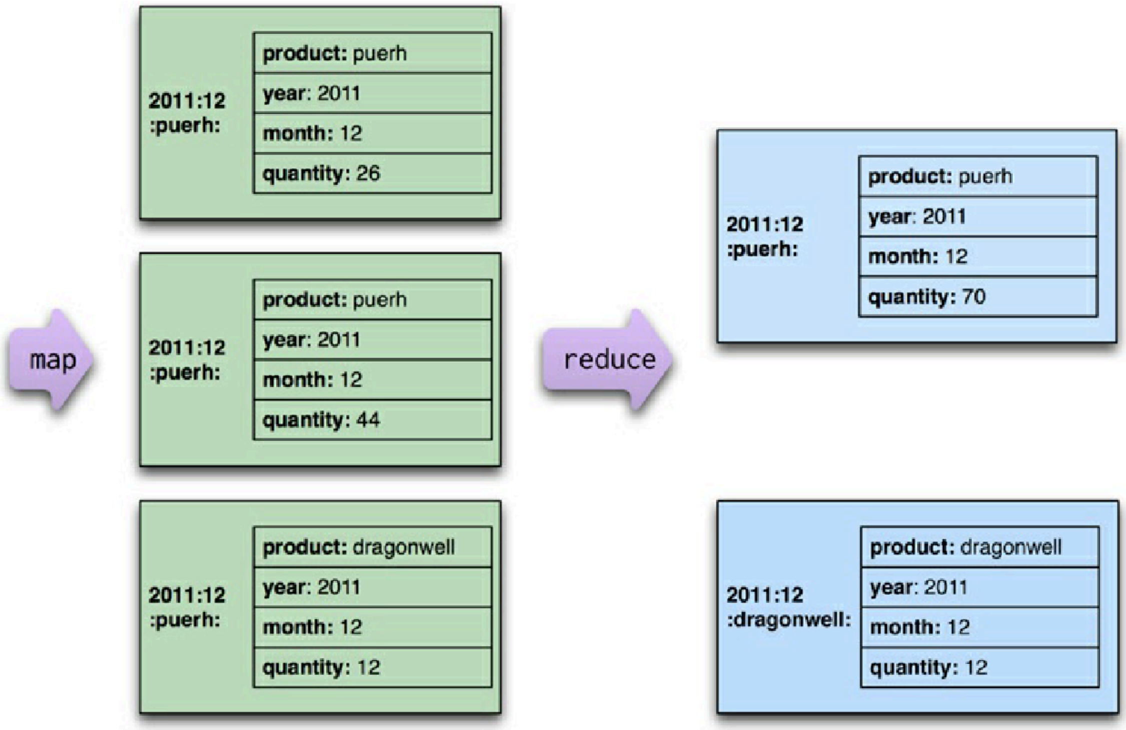

Figure 7.9. Creating records for monthly sales of a product

- The only new feature is using a composite key so that we can reduce records based on the values of multiple fields.
- The second-stage mappers (Figure 7.10) **process this output depending on the year.** A 2011 record populates the current year quantity while a 2010 record populates a prior year quantity. Records for earlier years (such as 2009) don't result in any mapping output being emitted.

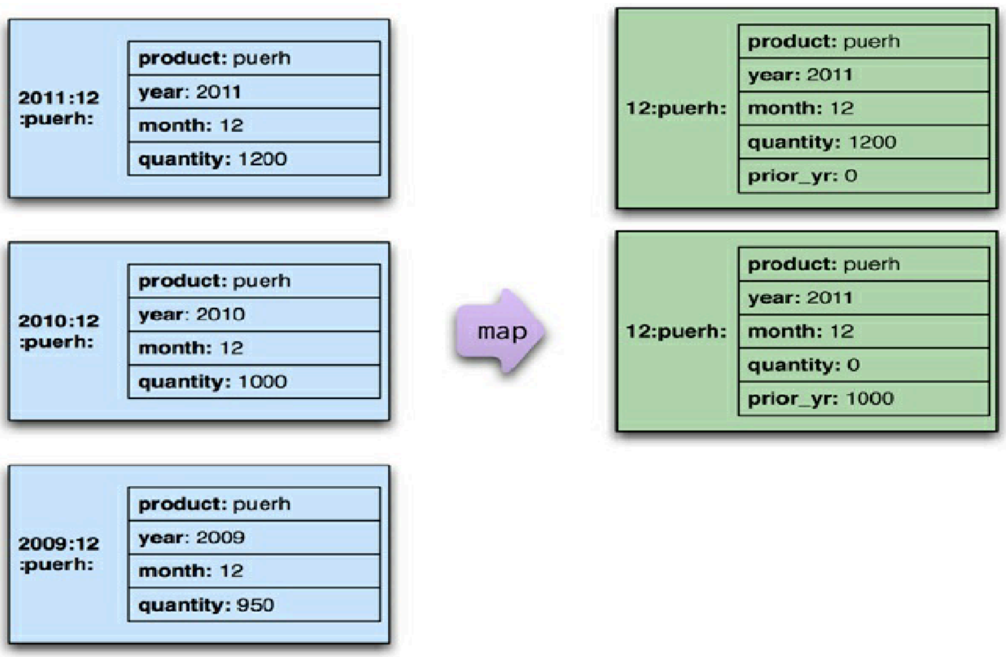

Figure 7.10. The second stage mapper creates base records for year-on-year comparisons.

- The reduce in this case (Figure 7.11) is a merge of records, where combining the values by summing allows two different year outputs to be reduced to a single value (with a calculation based on the reduced values thrown in for good measure).

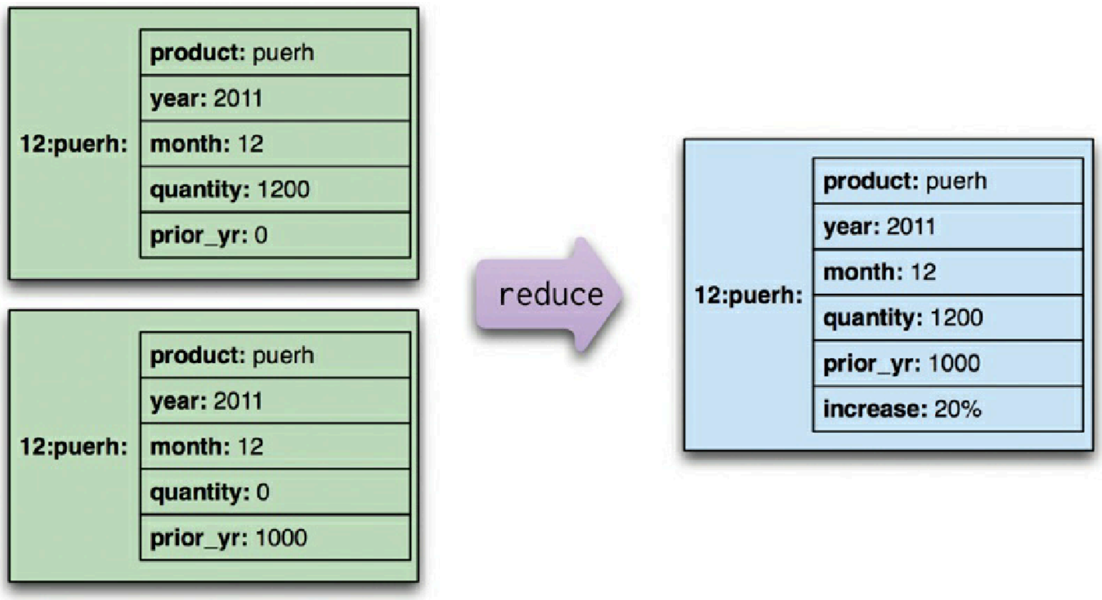

Figure 7.11. The reduction step is a merge of incomplete records.

Q3. Suppose we want to return all the documents in an order collection (all rows in the order table. Write down SQL and MongoDB queries for:

i) Selecting the orders for a single customerId of 883c2c5b4e5b

ii) Selecting orderId and orderDate for one customer

iii) Query for all the orders where one of the items ordered has a name like Refactoring

ans:
SELECT * FROM order WHERE customerId = "883c2c5b4e5b"
The equivalent query in Mongo to get all orders for a single customerId of 883c2c5b4e5b:
db.order.find({"customerId":"883c2c5b4e5b"})

Similarly, selecting orderId and orderDate for one customer in SQL would be:
SELECT orderId,orderDate FROM order WHERE customerId = "883c2c5b4e5b"
and the equivalent in Mongo would be:
db.order.find({customerId:"883c2c5b4e5b"},{orderId:1,orderDate:1})

SELECT * FROM customerOrder, orderItem, product
WHERE
customerOrder.orderId = orderItem.customerOrderId
AND orderItem.productId = product.productId
AND product.name LIKE '%Refactoring%'
and the equivalent Mongo query would be:
db.orders.find({"items.product.name":/Refactoring/})

Q4. Explain key-value store with respect to consistency and transactions. Explain Bucket creation in key-value store.
ans:

Consistency

1. Consistency is applicable only for operations on a single key, since these operations are either a get, put, or delete on a single key. Across key writes are very expensive.

2. In distributed key-value store implementations like Riak, **the eventually consistent model** of consistency is implemented.

3. Since the value may have already been replicated to other nodes, **Riak has two ways of resolving update conflicts: either the newest write wins and older writes loose, or**
both (all) values are returned allowing the client to resolve the conflict.

4. In Riak, th**ese options can be set up during the bucket creation**.

5. **Buckets are just a way to namespace keys so that key collisions can be reduced**—for example, **all customer keys may reside in the customer bucket**.

6. **When creating a bucket, default values for consistency can be provided,** for example that a write is considered good only when the data is consistent across all the nodes where the data is stored.

Bucket bucket = connection.createBucket(bucketName).withRetrier(attempts(3))

```
.allowSiblings(siblingsAllowed)
.nVal(numberOfReplicasOfTheData)
.w(numberOfNodesToRespondToWrite)
.r(numberOfNodesToRespondToRead)
.execute();
```

7. If we need data in every node to be consistent, increase the numberOfNodesToRespondToWrite set by w to be the same as nVal.
8. Of course doing that will decrease the write performance of the cluster.
9. To improve on write or read conflicts, change the allowSiblings flag during bucket creation: If it is set to false, we let the last write to win and not create siblings

Transactions

 Different products of the key-value store kind have different specifications of transactions. There are no guarantees on the writes.
 Many data stores do implement transactions in different ways.
 Riak uses the concept of quorum implemented by using the W value—replication factor—during the write API call.
 Assume we have a Riak cluster with a replication factor of 5 and supply the W value of 3. When writing, the write is reported as successful only when it is written and reported as a success on at least three of the nodes.
 This allows Riak to have write tolerance; in our example, with N equal to 5 and with a W value of 3, the cluster can tolerate N - W = 2 nodes being down for write operations, though we would still have lost some data on those nodes for read

Bucket creation:
When creating a bucket, default values for consistency can be provided, for example that a write is considered good only when the data is consistent across all the nodes where the data is stored.

```
Bucket bucket = connection.createBucket(bucketName).withRetrier(attempts(3))
.allowSiblings(siblingsAllowed)
.nVal(numberOfReplicasOfTheData)
.w(numberOfNodesToRespondToWrite)
.r(numberOfNodesToRespondToRead)
.execute();
```

Q5. Identify the situations where document databases are i) applicable ii) not advisable. Justify your answer

ans:

Applicable

1.Storing Session Information

- Generally, every web session is unique and is assigned a unique sessionid value.
- Applications that store the sessionid on disk or in an RDBMS will greatly benefit from moving to a key-value store, since everything about the session can be stored by a single PUT request or retrieved using GET.
- This single-request operation makes it very fast, as everything about the session is stored in a single object.
- Solutions such as Memcached are used by many web applications, and Riak can be used when availability is important.

.2. User Profiles, Preferences

- Almost every user has a unique userId, username, or some other attribute, as well as preferences such as language, color, timezone, which products the user has access to, and so on.
- This can all be put into an object, so getting preferences of a user takes a single GET operation.
- Similarly, product profiles can be stored.

3. Shopping Cart Data

- E-commerce websites have shopping carts tied to the user.
- As we want the shopping carts to be available all the time, across browsers, machines, and sessions, all the shopping information can be put into the value where the key is the userid.
- A Riak cluster would be best suited for these kinds of applications.

## When Not to Use

There are problem spaces where key-value stores are not the best solution.

.1. Relationships among Data

To express relationships between different sets of data, or correlate the data between different sets of keys, key-value stores are not the best solution to use, even though some key-value stores provide link-walking features.

2. Multi Operation Transactions

For saving multiple keys and if there is a failure to save any one of them, and you want to revert or roll back the rest of the operations, key-value stores are not the best solution to be used.

3. Query by Data

- If you need to search the keys based on something found in the value part of the key-value pairs, then key-value stores are not going to perform well for you.
- There is no way to inspect the value on the database side, with the exception of some products like Riak Search or indexing engines like Lucene[Lucene] or Solr [Solr].

4. Operations by Sets

Since operations are limited to one key at a time, there is no way to operate upon multiple keys at the same time. If you need to operate upon multiple keys, you have to handle this from the client side.

Q6. What is key-value store? Explain with an example. List and explain any two features of key-value store.

ans:

A key-value store is a type of NoSQL database that organizes data as a collection of key-value pairs. In this database model, each data item (or record) is stored as a key-value pair, where the key is a unique identifier that is used to retrieve or access the associated value. Key-value stores are designed to be simple and efficient for storing and retrieving data based on keys.

Example of Key-Value Store:

Imagine a simple key-value store used to store user profiles:

- Key: User ID (e.g., "123")

- Value: User Profile Information (e.g., {"name": "John Doe", "age": 30, "email": "johndoe@example.com"})

In this example, the user ID serves as the key, and the associated user profile information (name, age, email) is the value. Using the user ID as the key, we can quickly retrieve the corresponding user profile data from the key-value store.

Features of Key-Value Stores:

1. **High Performance Retrieval**:

Key-value stores are optimized for fast retrieval of data based on keys. This performance is achieved by using efficient data structures (like hash tables) to store and index key-value pairs. Retrieval operations typically have a constant time complexity (O(1)), making key-value stores suitable for use cases where rapid data access is crucial, such as caching and session management.

2. **Schema-less Design**:

Key-value stores typically have a schema-less or schema-flexible design, meaning that each key-value pair can have a different structure or format without requiring a predefined schema for the entire database. This flexibility allows developers to store heterogeneous data types together in the same database and adapt the data model as requirements evolve. It also simplifies data modeling and avoids the overhead of maintaining complex schemas.

Key-value stores are widely used in various applications, including distributed caching (e.g., Redis), session stores, user profiles, metadata storage, and real-time data processing. They provide a scalable and efficient solution for managing large volumes of data with predictable performance characteristics.