**MATH CLUB**

**HACKATHON ROUND-3**

**Team Details: DMV GIRLS**

**23BCE9315 - P.Durgamma**

**23BCE9319 - J.Vineetha**

**23BCE9318 - K.Meghana**

**4. Database**

**Problem Statement**: Create an innovative, scalable, and highly-available database solution optimized for modern applications. Your database should support ACID transactions,efficient indexing, and flexible querying with a focus on performance. Additionally, provide multi-tenant capabilities, backup and restore functionalities, and horizontal scaling across distributed nodes. Consider adding support for NoSQL or hybrid (SQL + NoSQL) models while ensuring data consistency, fault tolerance, and high availability across clusters.

**CODE:**

import java.util.\*;

import java.util.stream.Collectors;

// Transaction Management

class TransactionManager {

private Map<String, List<String>> transactionLogs = new HashMap<>();

public String startTransaction() {

String transactionId = UUID.randomUUID().toString();

transactionLogs.put(transactionId, new ArrayList<>());

return transactionId;

}

public void logOperation(String transactionId, String operation) {

transactionLogs.get(transactionId).add(operation);

}

public void commitTransaction(String transactionId, TenantDB tenantDB) {

List<String> logs = transactionLogs.get(transactionId);

for (String log : logs) {

tenantDB.applyOperation(log);

}

transactionLogs.remove(transactionId);

}

public void rollbackTransaction(String transactionId) {

transactionLogs.remove(transactionId); // Discard all operations

}

}

// Basic Key-Value Store for a Tenant's Database

class TenantDB {

private Map<String, String> keyValueStore = new HashMap<>();

private Map<String, List<String>> index = new HashMap<>();

// Apply a logged operation (used in transaction commit)

public void applyOperation(String operation) {

String[] parts = operation.split(" ");

String key = parts[0];

String value = parts[1];

keyValueStore.put(key, value);

addToIndex(key, value);

}

// Add a key-value pair

public void put(String key, String value) {

keyValueStore.put(key, value);

addToIndex(key, value);

}

// Get a value by key

public String get(String key) {

return keyValueStore.get(key);

}

// Add indexing for efficient lookups

public void addToIndex(String key, String value) {

index.computeIfAbsent(key, k -> new ArrayList<>()).add(value);

}

// Search by index

public List<String> searchByIndex(String key) {

return index.getOrDefault(key, Collections.emptyList());

}

// Querying mechanism

public List<String> query(String condition) {

return keyValueStore.entrySet().stream()

.filter(entry -> entry.getKey().startsWith(condition))

.map(Map.Entry::getValue)

.collect(Collectors.toList());

}

}

// SQL/NoSQL Hybrid Support

class HybridDB {

// SQL-like structured data

private Map<String, Map<String, String>> tables = new HashMap<>();

// NoSQL-like unstructured data

private Map<String, String> documentStore = new HashMap<>();

// SQL: Create Table

public void createTable(String tableName) {

tables.put(tableName, new HashMap<>());

}

// SQL: Insert Row

public void insertRow(String tableName, String rowId, String data) {

tables.get(tableName).put(rowId, data);

}

// NoSQL: Add Document

public void addDocument(String docId, String jsonDocument) {

documentStore.put(docId, jsonDocument);

}

// NoSQL: Get Document

public String getDocument(String docId) {

return documentStore.get(docId);

}

// SQL: Query Table

public String queryTable(String tableName, String rowId) {

return tables.getOrDefault(tableName, Collections.emptyMap()).get(rowId);

}

}

// Horizontal Scaling with Consistent Hashing

class ConsistentHashing {

private TreeMap<Integer, String> hashRing = new TreeMap<>();

private int numberOfReplicas;

public ConsistentHashing(List<String> nodes, int numberOfReplicas) {

this.numberOfReplicas = numberOfReplicas;

for (String node : nodes) {

addNode(node);

}

}

public void addNode(String node) {

for (int i = 0; i < numberOfReplicas; i++) {

hashRing.put((node.hashCode() + i), node);

}

}

public String getNodeForKey(String key) {

if (hashRing.isEmpty()) {

return null;

}

int hash = key.hashCode();

if (!hashRing.containsKey(hash)) {

SortedMap<Integer, String> tailMap = hashRing.tailMap(hash);

hash = tailMap.isEmpty() ? hashRing.firstKey() : tailMap.firstKey();

}

return hashRing.get(hash);

}

}

// Simulate Data Replication

class ReplicationManager {

private List<String> replicationNodes;

public ReplicationManager(List<String> nodes) {

this.replicationNodes = nodes;

}

// Simulate replication by sending data to multiple nodes

public void replicateData(String key, String value) {

for (String node : replicationNodes) {

System.out.println("Replicating " + key + " to node: " + node);

}

}

}

public class ScalableDatabase {

public static void main(String[] args) {

// Simulate tenant and transaction management

TenantDB tenantDB = new TenantDB();

TransactionManager txManager = new TransactionManager();

HybridDB hybridDB = new HybridDB();

List<String> nodes = Arrays.asList("Node1", "Node2", "Node3");

ConsistentHashing consistentHashing = new ConsistentHashing(nodes, 3);

ReplicationManager replicationManager = new ReplicationManager(nodes);

// Start a transaction

String txId = txManager.startTransaction();

txManager.logOperation(txId, "key1 value1");

txManager.logOperation(txId, "key2 value2");

// Commit transaction (ACID: Atomicity, Consistency)

txManager.commitTransaction(txId, tenantDB);

// Querying data

System.out.println("Query Result: " + tenantDB.query("key"));

// SQL-like table operations

hybridDB.createTable("users");

hybridDB.insertRow("users", "user1", "John Doe");

// NoSQL-like document operations

hybridDB.addDocument("doc1", "{ 'name': 'Jane Doe', 'age': 25 }");

// Query SQL Table

System.out.println("SQL Query: " + hybridDB.queryTable("users", "user1"));

// Query NoSQL Document

System.out.println("NoSQL Query: " + hybridDB.getDocument("doc1"));

// Horizontal scaling: Get node for key (Consistent Hashing)

System.out.println("Node for key 'key1': " + consistentHashing.getNodeForKey("key1"));

// Simulate replication

replicationManager.replicateData("key1", "value1");

}

}

**DESCRIPTION OF CODE:**

Overview: Scalable Database System

The provided code implements a scalable and versatile database system combining multiple database features and concepts. It includes transaction management, a hybrid SQL/NoSQL database, horizontal scaling with consistent hashing, and data replication.

**1. Transaction Management** (TransactionManager Class):

Purpose: Manages database transactions to ensure ACID properties (Atomicity, Consistency, Isolation, Durability).

Key Methods:

**startTransaction()**: Begins a new transaction and returns a unique transaction ID.

**logOperation(String transactionId, String operation):** Logs database operations (e.g., insert/update) for a transaction.

**commitTransaction(String transactionId, TenantDB tenantDB):** Applies all operations in the transaction log to the TenantDB and commits the transaction.

**rollbackTransaction(String transactionId):** Discards all operations in the transaction log, effectively rolling back the transaction.

**2. Tenant Database** (TenantDB Class):

Purpose: Represents a basic key-value store for managing tenant-specific data with optional indexing.

Key Methods:

**put(String key, String value):** Adds or updates a key-value pair in the database.

**get(String key):** Retrieves the value associated with a given key.

**addToIndex(String key, String value):** Adds key-value pairs to an index for efficient searching.

**searchByIndex(String key):** Retrieves values based on the indexed key.

**query(String condition):** Searches the key-value store for entries matching the given condition.

**3. Hybrid Database** (HybridDB Class):

Purpose: Provides a combination of SQL-like and NoSQL-like functionalities.

Key Methods:

**createTable(String tableName)**: Creates a new SQL table.

**insertRow(String tableName, String rowId, String data):** Inserts a row into an SQL table.

**addDocument(String docId, String jsonDocument):** Adds a NoSQL document.

**getDocument(String docId):** Retrieves a NoSQL document by its ID.

**queryTable(String tableName, String rowId):** Retrieves a row from an SQL table by its ID.

**4. Consistent Hashing** (ConsistentHashing Class):

Purpose: Implements horizontal scaling using consistent hashing to distribute data across multiple nodes.

Key Methods:

**addNode(String node):** Adds a node to the hash ring with multiple replicas.

**getNodeForKey(String key):** Determines the node responsible for a given key using the hash ring.

**5. Replication Manager** (ReplicationManager Class):

Purpose: Simulates data replication to ensure high availability.

Key Method:

**replicateData(String key, String value)**: Simulates the process of replicating data to all nodes.

**6. Main Application** (ScalableDatabase Class):

Purpose: Demonstrates the use of all classes by setting up a database system and performing various operations.

Key Actions:

1.Initializes instances of TenantDB, TransactionManager, HybridDB, ConsistentHashing, and ReplicationManager.

2.Starts and commits transactions using TransactionManager.

3.Performs SQL and NoSQL operations using HybridDB.

4.Uses ConsistentHashing to determine the node for a given key.

5.Simulates data replication using ReplicationManager.