BANKING SYSTEM PROJECT - DAY 4: MONGODB INTEGRATION

====================================================

OVERVIEW

--------

Day 4 focuses on integrating MongoDB as the primary database for the banking system. This implementation provides ACID-like consistency, efficient data storage, and robust query capabilities for customer, account, and transaction data management.

MONGODB CONFIGURATION

---------------------

1. DEPENDENCY CONFIGURATION

- Spring Boot Starter Data MongoDB

- MongoDB Java Driver

- Spring Data MongoDB

- MongoDB Embedded (for testing)

2. APPLICATION PROPERTIES

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=banking\_system

spring.data.mongodb.auto-index-creation=true

spring.data.mongodb.options.connections-per-host=10

spring.data.mongodb.options.threads-allowed-to-block-for-connection-multiplier=5

spring.data.mongodb.options.max-wait-time=120000

spring.data.mongodb.options.connect-timeout=10000

spring.data.mongodb.options.socket-timeout=0

3. CONNECTION POOL SETTINGS

- Maximum connections per host: 10

- Connection timeout: 10 seconds

- Socket timeout: 0 (infinite)

- Max wait time: 120 seconds

- Thread multiplier: 5

DATABASE SCHEMA DESIGN

----------------------

1. CUSTOMERS COLLECTION

Document Structure:

{

"\_id": ObjectId("..."),

"firstName": "Rajesh",

"lastName": "Kumar",

"email": "rajesh.kumar@example.com",

"phoneNumber": "9876543210",

"address": "123 MG Road",

"city": "Mumbai",

"state": "Maharashtra",

"pincode": "400001",

"panNumber": "ABCDE1234F",

"aadharNumber": "123456789012",

"dateOfBirth": ISODate("1990-05-15T00:00:00.000Z"),

"createdDate": ISODate("2024-01-01T10:00:00.000Z"),

"lastModifiedDate": ISODate("2024-01-01T10:00:00.000Z"),

"status": "ACTIVE"

}

Indexes:

- \_id: Primary key (automatic)

- email: Unique index

- phoneNumber: Unique index

- panNumber: Unique index (sparse)

- aadharNumber: Unique index (sparse)

- status: Regular index

- createdDate: Regular index

2. ACCOUNTS COLLECTION

Document Structure:

{

"\_id": ObjectId("..."),

"accountNumber": "ACC123456789",

"customerId": ObjectId("..."),

"accountType": "SAVINGS",

"balance": NumberDecimal("50000.00"),

"minimumBalance": NumberDecimal("1000.00"),

"interestRate": NumberDecimal("4.50"),

"status": "ACTIVE",

"openedDate": ISODate("2024-01-01T10:00:00.000Z"),

"lastTransactionDate": ISODate("2024-01-15T14:30:00.000Z"),

"createdDate": ISODate("2024-01-01T10:00:00.000Z"),

"lastModifiedDate": ISODate("2024-01-15T14:30:00.000Z")

}

Indexes:

- \_id: Primary key (automatic)

- accountNumber: Unique index

- customerId: Regular index

- accountType: Regular index

- status: Regular index

- balance: Regular index

- openedDate: Regular index

3. TRANSACTIONS COLLECTION

Document Structure:

{

"\_id": ObjectId("..."),

"transactionId": "TXN20240115001",

"accountId": ObjectId("..."),

"transactionType": "DEPOSIT",

"amount": NumberDecimal("10000.00"),

"description": "Salary deposit",

"sourceAccountId": ObjectId("..."),

"destinationAccountId": ObjectId("..."),

"status": "COMPLETED",

"transactionDate": ISODate("2024-01-15T14:30:00.000Z"),

"processedDate": ISODate("2024-01-15T14:30:05.000Z"),

"referenceNumber": "REF123456789",

"userId": "USER001",

"createdDate": ISODate("2024-01-15T14:30:00.000Z"),

"lastModifiedDate": ISODate("2024-01-15T14:30:05.000Z")

}

Indexes:

- \_id: Primary key (automatic)

- transactionId: Unique index

- accountId: Regular index

- transactionType: Regular index

- status: Regular index

- transactionDate: Regular index

- userId: Regular index

- sourceAccountId: Regular index

- destinationAccountId: Regular index

4. AUDITLOGS COLLECTION

Document Structure:

{

"\_id": ObjectId("..."),

"actionId": "ACT20240115001",

"userId": "USER001",

"actionType": "DEPOSIT",

"entityType": "TRANSACTION",

"entityId": "TXN20240115001",

"actionDetails": "Deposit of Rs. 10000.00 processed successfully",

"timestamp": ISODate("2024-01-15T14:30:00.000Z"),

"ipAddress": "192.168.1.100",

"userAgent": "Mozilla/5.0...",

"status": "SUCCESS",

"errorMessage": null,

"createdDate": ISODate("2024-01-15T14:30:00.000Z")

}

Indexes:

- \_id: Primary key (automatic)

- actionId: Unique index

- userId: Regular index

- actionType: Regular index

- entityType: Regular index

- timestamp: Regular index

- status: Regular index

REPOSITORY IMPLEMENTATION

-------------------------

1. CUSTOMER REPOSITORY

@Repository

public interface CustomerRepository extends MongoRepository<Customer, String> {

Optional<Customer> findByEmail(String email);

Optional<Customer> findByPhoneNumber(String phoneNumber);

Optional<Customer> findByPanNumber(String panNumber);

Optional<Customer> findByAadharNumber(String aadharNumber);

List<Customer> findByStatus(CustomerStatus status);

List<Customer> findByCity(String city);

List<Customer> findByState(String state);

@Query("{ 'firstName': { $regex: ?0, $options: 'i' } }")

List<Customer> findByFirstNameContainingIgnoreCase(String firstName);

@Query("{ 'lastName': { $regex: ?0, $options: 'i' } }")

List<Customer> findByLastNameContainingIgnoreCase(String lastName);

}

2. ACCOUNT REPOSITORY

@Repository

public interface AccountRepository extends MongoRepository<Account, String> {

Optional<Account> findByAccountNumber(String accountNumber);

List<Account> findByCustomerId(String customerId);

List<Account> findByAccountType(AccountType accountType);

List<Account> findByStatus(AccountStatus status);

List<Account> findByBalanceGreaterThan(BigDecimal amount);

List<Account> findByBalanceLessThan(BigDecimal amount);

@Query("{ 'customerId': ?0, 'status': 'ACTIVE' }")

List<Account> findActiveAccountsByCustomerId(String customerId);

@Query("{ 'balance': { $gte: ?0, $lte: ?1 } }")

List<Account> findByBalanceBetween(BigDecimal minBalance, BigDecimal maxBalance);

}

3. TRANSACTION REPOSITORY

@Repository

public interface TransactionRepository extends MongoRepository<Transaction, String> {

Optional<Transaction> findByTransactionId(String transactionId);

List<Transaction> findByAccountId(String accountId);

List<Transaction> findByTransactionType(TransactionType type);

List<Transaction> findByStatus(TransactionStatus status);

List<Transaction> findByUserId(String userId);

List<Transaction> findByTransactionDateBetween(LocalDateTime startDate, LocalDateTime endDate);

List<Transaction> findBySourceAccountId(String sourceAccountId);

List<Transaction> findByDestinationAccountId(String destinationAccountId);

@Query("{ 'accountId': ?0, 'transactionDate': { $gte: ?1, $lte: ?2 } }")

List<Transaction> findByAccountIdAndTransactionDateBetween(String accountId, LocalDateTime startDate, LocalDateTime endDate);

@Query("{ 'transactionType': ?0, 'status': 'COMPLETED' }")

List<Transaction> findCompletedTransactionsByType(TransactionType type);

}

4. AUDITLOG REPOSITORY

@Repository

public interface AuditLogRepository extends MongoRepository<AuditLog, String> {

Optional<AuditLog> findByActionId(String actionId);

List<AuditLog> findByUserId(String userId);

List<AuditLog> findByActionType(String actionType);

List<AuditLog> findByStatus(AuditStatus status);

List<AuditLog> findByTimestampBetween(LocalDateTime startDate, LocalDateTime endDate);

@Query("{ 'userId': ?0, 'timestamp': { $gte: ?1, $lte: ?2 } }")

List<AuditLog> findByUserIdAndTimestampBetween(String userId, LocalDateTime startDate, LocalDateTime endDate);

@Query("{ 'actionType': ?0, 'status': 'SUCCESS' }")

List<AuditLog> findSuccessfulActionsByType(String actionType);

}

ACID TRANSACTION IMPLEMENTATION

-------------------------------

1. TRANSACTION MANAGEMENT

@Service

@Transactional

public class TransactionService {

@Transactional(rollbackFor = Exception.class)

public Transaction processDeposit(String accountNumber, BigDecimal amount, String description, String userId) {

// Start MongoDB transaction

TransactionSession session = mongoTemplate.startSession();

session.startTransaction();

try {

// Validate account

Account account = accountRepository.findByAccountNumber(accountNumber)

.orElseThrow(() -> new AccountNotFoundException("Account not found"));

// Update balance

account.setBalance(account.getBalance().add(amount));

account.setLastTransactionDate(LocalDateTime.now());

accountRepository.save(account);

// Create transaction record

Transaction transaction = new Transaction();

transaction.setTransactionId(generateTransactionId());

transaction.setAccountId(account.getId());

transaction.setTransactionType(TransactionType.DEPOSIT);

transaction.setAmount(amount);

transaction.setDescription(description);

transaction.setStatus(TransactionStatus.COMPLETED);

transaction.setUserId(userId);

transactionRepository.save(transaction);

// Log audit trail

auditService.logSuccess(userId, "DEPOSIT", "TRANSACTION", transaction.getId(),

"Deposit of " + amount + " processed successfully");

// Commit transaction

session.commitTransaction();

return transaction;

} catch (Exception e) {

// Rollback transaction

session.abortTransaction();

throw e;

} finally {

session.close();

}

}

}

2. TRANSFER TRANSACTION IMPLEMENTATION

@Transactional(rollbackFor = Exception.class)

public Transaction processTransfer(String sourceAccountNumber, String destinationAccountNumber,

BigDecimal amount, String description, String userId) {

TransactionSession session = mongoTemplate.startSession();

session.startTransaction();

try {

// Validate source account

Account sourceAccount = accountRepository.findByAccountNumber(sourceAccountNumber)

.orElseThrow(() -> new AccountNotFoundException("Source account not found"));

// Validate destination account

Account destinationAccount = accountRepository.findByAccountNumber(destinationAccountNumber)

.orElseThrow(() -> new AccountNotFoundException("Destination account not found"));

// Check sufficient balance

if (sourceAccount.getBalance().compareTo(amount) < 0) {

throw new InsufficientFundsException("Insufficient balance");

}

// Update source account balance

sourceAccount.setBalance(sourceAccount.getBalance().subtract(amount));

sourceAccount.setLastTransactionDate(LocalDateTime.now());

accountRepository.save(sourceAccount);

// Update destination account balance

destinationAccount.setBalance(destinationAccount.getBalance().add(amount));

destinationAccount.setLastTransactionDate(LocalDateTime.now());

accountRepository.save(destinationAccount);

// Create transaction record

Transaction transaction = new Transaction();

transaction.setTransactionId(generateTransactionId());

transaction.setAccountId(sourceAccount.getId());

transaction.setTransactionType(TransactionType.TRANSFER);

transaction.setAmount(amount);

transaction.setDescription(description);

transaction.setSourceAccountId(sourceAccount.getId());

transaction.setDestinationAccountId(destinationAccount.getId());

transaction.setStatus(TransactionStatus.COMPLETED);

transaction.setUserId(userId);

transactionRepository.save(transaction);

// Log audit trail

auditService.logSuccess(userId, "TRANSFER", "TRANSACTION", transaction.getId(),

"Transfer of " + amount + " from " + sourceAccountNumber + " to " + destinationAccountNumber);

// Commit transaction

session.commitTransaction();

return transaction;

} catch (Exception e) {

// Rollback transaction

session.abortTransaction();

throw e;

} finally {

session.close();

}

}

QUERY OPTIMIZATION

------------------

1. INDEX STRATEGY

- Primary indexes on \_id fields (automatic)

- Unique indexes on business keys (accountNumber, email, etc.)

- Compound indexes for common query patterns

- Sparse indexes for optional fields

- Text indexes for search functionality

2. QUERY PERFORMANCE

- Use of projection to limit returned fields

- Pagination for large result sets

- Aggregation pipelines for complex queries

- Explain plans for query optimization

- Connection pooling for performance

3. AGGREGATION PIPELINES

db.transactions.aggregate([

{

$match: {

"transactionDate": {

$gte: ISODate("2024-01-01T00:00:00.000Z"),

$lte: ISODate("2024-01-31T23:59:59.999Z")

}

}

},

{

$group: {

\_id: "$transactionType",

totalAmount: { $sum: "$amount" },

count: { $sum: 1 }

}

},

{

$sort: { totalAmount: -1 }

}

])

DATA CONSISTENCY

----------------

1. REFERENTIAL INTEGRITY

- Application-level foreign key validation

- Cascade operations for related entities

- Orphan record prevention

- Data integrity checks

2. CONCURRENCY CONTROL

- Optimistic locking with version fields

- Pessimistic locking for critical operations

- Race condition prevention

- Deadlock avoidance

3. BACKUP AND RECOVERY

- Regular database backups

- Point-in-time recovery

- Replica set configuration

- Disaster recovery procedures

MONITORING AND MAINTENANCE

--------------------------

1. PERFORMANCE MONITORING

- Query performance tracking

- Index usage analysis

- Connection pool monitoring

- Memory usage tracking

2. HEALTH CHECKS

- Database connectivity checks

- Replica set status monitoring

- Disk space monitoring

- Performance metrics collection

3. MAINTENANCE TASKS

- Index rebuilding

- Collection compaction

- Log rotation

- Statistics updates

This MongoDB integration provides a robust, scalable, and performant data storage solution for the banking system with proper ACID compliance and data consistency.