10. Liệt kê các các đặc tính chất lượng mong muốn đạt được với thiết kế Event Sourcing trong bài lab 03. Liệt kê các công cụ có thể sử dụng và các bước cần thực hiện, để kiểm tra các đặc tính chất lượng này. Vẽ sơ đồ lưu trữ của thiết kế đề xuất? Liệt kê các công cụ có thể sử dụng và các bước cần thực hiện để cài đặt sơ đồ lưu trữ đề xuất. Giải thích mã nguồn ghi và đọc các sự kiện từ hệ thống lưu trữ các sự kiện.

**Trả lời**

1. **Các đặc tính chất lượng mong muốn đạt được với thiết kế Event Sourcing trong lab03.**

|  |  |
| --- | --- |
| **Đặc tính chất lượng** | **Ý nghĩa trong hệ thống** |
| **Functionality** | Hệ thống quản lý đơn hàng dựa trên Event Sourcing + CQRS, hỗ trợ đầy đủ Create, Update, Cancel, Rollback, View, Audit. |
| **Auditability**  (Khả năng kiểm toán) | * **Mô tả**: Lưu trữ toàn bộ lịch sử thay đổi dưới dạng events immutable. * **Giá trị:** Cung cấp audit trail đầy đủ cho compliance, tracking mọi business action * **Ứng dụng**: Phù hợp cho hệ thống tài chính, y tế, pháp lý yêu cầu transparency cao. |
| **Data Integrity**  (Tính toàn vẹn dữ liệu) | * **Immutable Events:** Events không bao giờ bị sửa đổi hoặc xóa * **Version Control:** Mỗi event có version number để đảm bảo ordering * **Rollback Protection:** Hệ thống ngăn chặn rollback về các version không hợp lệ |
| **Traceability**  (Khả năng truy vết) | * **Event History**: Có thể xem toàn bộ chuỗi events của một aggregate * **Timestamp Tracking**: Mỗi event có timestamp chính xác * **State Reconstruction**: Rebuild state tại bất kỳ thời điểm nào |
| **Scalability**  (Khả năng mở rộng) | * **Append-only Events**: Hiệu suất cao cho write operations * **Stateless Application**: Dễ dàng horizontal scaling * **Event Replay**: Có thể parallel processing events |
| **Flexibility**  (Tính linh hoạt) | * **Schema Evolution**: Dễ dàng thêm event types mới * **Business Logic Changes**: Có thể thay đổi cách interpret events mà không làm mất dữ liệu. * **Multi-purpose Events**: Cùng event stream cho nhiều use cases |
| **Recoverability**  (Khả năng phục hồi) | * **Time Travel**: Rollback về bất kỳ thời điểm nào * **Zero Data Loss**: Rollback không phá hủy dữ liệu gốc * **Disaster Recovery**: Rebuild toàn bộ hệ thống từ event store |

1. **Các công cụ có thể sử dụng và các bước cần thực hiện để kiểm tra các đặc tính chất lượng này.**

**Công cụ đề xuất**

* **Postman / Newman** – kiểm thử chức năng, rollback, data integrity.
* **JMeter / k6** – kiểm thử tải, stress test (scalability & performance).
* **Database client (psql, MongoDB Compass, Kafka UI)** – kiểm tra dữ liệu & event log.
* **Logging & Tracing cơ bản (console log, timestamp, correlation ID)** – kiểm tra auditability & traceability.
* **Backup & Restore script (dump & restore DB)** – kiểm tra recoverability.

**Các bước kiểm tra cho từng đặc tính**

* **Functionality** → Dùng **Postman** gọi API (Create/Update/Cancel/Rollback/View). So sánh kết quả với expected state.
* **Auditability** → Kiểm tra **log file / DB event collection** đảm bảo events không bị sửa/xóa.
* **Data Integrity** → Thử update đồng thời, rollback sai version bằng **Postman**, rồi check lại dữ liệu bằng **DB client**.
* **Traceability** → Mỗi event có timestamp + correlation ID, kiểm tra trực tiếp trong **log file/DB**.
* **Scalability** → Dùng **JMeter/k6** giả lập nhiều request song song, đo throughput & latency.
* **Recoverability** → Dùng **backup & restore DB** (pg\_dump/restore, mongodump/restore) + replay events để xác thực dữ liệu khớp.
  1. **Performance Testing Tools:**
     1. **Artillery.io / K6 - Load Testing**

# Cài đặt Artillery

npm install -g artillery

# Test create order performance

artillery run performance-test.yml

**Test Script Example**

config:

  target: 'http://localhost:3001'

  phases:

    - duration: 60

      arrivalRate: 10

scenarios:

  - name: "Create Orders"

    requests:

      - post:

          url: "/api/orders"

          json:

            customerId: "load-test-{{ $randomString() }}"

            items: [{"productId": "p1", "productName": "Test", "quantity": 1, "price": 100}]

* + 1. **PostgreSQL Performance Monitoring**

-- Check event store performance

EXPLAIN ANALYZE SELECT \* FROM events WHERE aggregate\_id = 'order-123' ORDER BY version;

-- Index effectiveness

SELECT \* FROM pg\_stat\_user\_indexes WHERE relname = 'events';

-- Query performance stats

SELECT query, mean\_time, calls FROM pg\_stat\_statements WHERE query LIKE '%events%';

* + 1. **Data Integrity Testing Tools:**

**Custom Test Scripts**

// Event immutability test

describe('Event Immutability', () => {

  it('should never modify existing events', async () => {

    const orderId = await createOrder();

    const originalEvents = await eventStore.getEvents(orderId);

    await updateOrderStatus(orderId, 'SHIPPED');

    const newEvents = await eventStore.getEvents(orderId);

    // Verify original events unchanged

    expect(newEvents.slice(0, originalEvents.length))

      .toEqual(originalEvents);

  });

});

**Database Constraints Testing**

-- Test unique version constraint

INSERT INTO events (aggregate\_id, event\_type, event\_data, version)

VALUES ('test-id', 'TestEvent', '{}', 1);

-- This should fail (duplicate version)

INSERT INTO events (aggregate\_id, event\_type, event\_data, version)

VALUES ('test-id', 'TestEvent2', '{}', 1);

* + 1. **Concurrency Testing**

**Jest với Concurrent Operations**

describe('Concurrency Tests', () => {

  it('should handle concurrent order updates', async () => {

    const orderId = await createOrder();

    const promises = [

      updateOrderStatus(orderId, 'CONFIRMED'),

      addOrderItem(orderId, newItem),

      removeOrderItem(orderId, 'product-1')

    ];

    await Promise.all(promises);

    const events = await eventStore.getEvents(orderId);

    expect(events).toHaveLength(4); // 1 create + 3 updates

  });

});

* + 1. **Rollback Testing Tools**

**Time Travel Verification**

describe('Rollback Testing', () => {

  it('should maintain data integrity during rollback', async () => {

    const orderId = await createTestOrder();

    // Create multiple events

    await updateStatus(orderId, 'CONFIRMED');

    await addItem(orderId, item1);

    await addItem(orderId, item2);

    const beforeRollback = await getOrder(orderId);

    // Rollback to version 2

    await rollbackOrder(orderId, 2);

    const afterRollback = await getOrder(orderId);

    const allEvents = await getOrderEvents(orderId);

    // Verify rollback event exists

    expect(allEvents).toContainEventType('OrderRolledBack');

    // Verify no data loss

    expect(allEvents.filter(e => e.type !== 'OrderRolledBack'))

      .toHaveLength(4);

  });

});

* 1. **Các bước kiểm tra chi tiết**

**Bước 1: Setup Testing Environment**

# Clone và setup project

git clone <repo>

cd Event-Sourcing-Demo

npm run install:all

# Start test database

docker run -d --name test-postgres -e POSTGRES\_PASSWORD=test -p 5433:5432 postgres

# Setup test schema

cd Order-management/database

psql -U postgres -h localhost -p 5433 -f schema.sql

**Bước 2: Auditability Testing**

# Test 1: Create order and verify event generation

curl -X POST http://localhost:3001/api/orders -H "Content-Type: application/json" -d '{

  "customerId": "audit-test-001",

  "items": [{"productId": "p1", "productName": "Test Product", "quantity": 1, "price": 100}]

}'

# Test 2: Verify event immutability

curl http://localhost:3001/api/debug/orders/{orderId}/events

# Test 3: Verify complete audit trail

curl http://localhost:3001/api/debug/events

**Kỳ vọng**: Mọi thao tác sinh event, event có timestamp & payload đầy đủ, không bị thay đổi.

**Bước 3: Performance Testing**

# Install performance testing tools

npm install -g autocannon

# Test create order performance

autocannon -c 10 -d 30 -m POST -H 'content-type=application/json' -b '{"customerId":"perf-test","items":[{"productId":"p1","productName":"Test","quantity":1,"price":100}]}' http://localhost:3001/api/orders

# Test query performance

autocannon -c 5 -d 30 http://localhost:3001/api/orders/{orderId}

**Kỳ vọng**: Write < 100ms, Read < 50ms, CPU/memory ổn định.

**Bước 4: Scalability Testing**

// Horizontal scaling test

const cluster = require('cluster');

const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {

  for (let i = 0; i < numCPUs; i++) {

    cluster.fork();

  }

} else {

  // Start application instance

  require('./src/index.js');

}

**Kỳ vọng**: Load phân bổ đều, không lỗi connection.

**Bước 5: Data Integrity Verification**

-- Verify referential integrity

SELECT

  aggregate\_id,

  COUNT(\*) as event\_count,

  MAX(version) as max\_version,

  MIN(version) as min\_version

FROM events

GROUP BY aggregate\_id

HAVING COUNT(\*) != MAX(version);

-- Check for version gaps

SELECT \* FROM (

  SELECT

    aggregate\_id,

    version,

    LAG(version) OVER (PARTITION BY aggregate\_id ORDER BY version) as prev\_version

  FROM events

) t

WHERE version != prev\_version + 1 AND prev\_version IS NOT NULL;

**Kỳ vọng**: Không có gap version, duplicate version bị chặn.

**Bước 6: Concurrency Testing**

describe('Concurrency Tests', () => {

  it('should handle concurrent updates', async () => {

    const orderId = await createOrder();

    await Promise.all([

      updateOrderStatus(orderId, 'CONFIRMED'),

      addOrderItem(orderId, newItem),

      removeOrderItem(orderId, 'product-1')

    ]);

    const events = await eventStore.getEvents(orderId);

    expect(events.length).toBe(4); // 1 create + 3 update

  });

});

**Kỳ vọng**: Không mất event, version tăng đúng.

**Bước 7: Rollback Testing**

describe('Rollback Test', () => {

  it('should rollback correctly', async () => {

    const id = await createTestOrder();

    await updateStatus(id, 'CONFIRMED');

    await addItem(id, item1);

    await addItem(id, item2);

    await rollbackOrder(id, 2);

    const events = await getOrderEvents(id);

    expect(events).toContainEqual(expect.objectContaining({ type: 'OrderRolledBack' }));

    expect(events.filter(e => e.type !== 'OrderRolledBack')).toHaveLength(4);

  });

});

**Kỳ vọng**: Rollback không xóa dữ liệu gốc, có log event rollback.

**Checklist kiểm tra đặc tính chất lượng:**

* **Auditability Checklist:**
  + Mọi business operation tạo events
  + Events có timestamp chính xác
  + Event payload chứa đầy đủ thông tin
  + Có thể trace back mọi thay đổi
* **Performance Checklist:**
  + Create operations < 100ms
  + Query operations < 50ms
  + Event store queries có index hiệu quả
  + Memory usage ổn định
* **Scalability Checklist:**
  + Application stateless
  + Database connection pooling
  + Horizontal scaling test passed
  + Event replay performance acceptable
* **Data Integrity Checklist:**
  + Event immutability verified
  + Version ordering correct
  + No data loss during rollback
  + Concurrent operations handle correctly

1. **Sơ đồ lưu trữ của thiết kế đề xuất.**
   1. **Kiến trúc lưu trữ tổng thể:**

A screenshot of a computer

AI-generated content may be incorrect.

* 1. **Chi tiết Event Store Schema:**

**1. Bảng events - Core Event Store**

Bảng trung tâm lưu trữ tất cả các sự kiện của hệ thống.

CREATE TABLE events (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

aggregate\_id VARCHAR(255) NOT NULL,

aggregate\_type VARCHAR(100) NOT NULL DEFAULT 'Order',

event\_type VARCHAR(100) NOT NULL,

event\_data JSONB NOT NULL,

version INTEGER NOT NULL,

timestamp TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

correlation\_id UUID DEFAULT NULL,

causation\_id UUID DEFAULT NULL,

metadata JSONB DEFAULT '{}',

*-- Constraints*

UNIQUE(aggregate\_id, version),

CHECK (version > 0),

CHECK (jsonb\_typeof(event\_data) = 'object')

);

**Mô tả các cột:**

| **Cột** | **Kiểu dữ liệu** | **Mô tả** | **Ví dụ** |
| --- | --- | --- | --- |
| id | UUID | Primary key duy nhất cho mỗi event | f47ac10b-58cc-4372-a567-0e02b2c3d479 |
| aggregate\_id | VARCHAR(255) | ID của đối tượng nghiệp vụ (Order ID) | order-123 |
| aggregate\_type | VARCHAR(100) | Loại đối tượng nghiệp vụ | Order, Customer, Product |
| event\_type | VARCHAR(100) | Loại sự kiện đã xảy ra | OrderCreated, OrderStatusUpdated |
| event\_data | JSONB | Dữ liệu chi tiết của sự kiện | {"orderId": "order-123", "customerId": "cust-001"} |
| version | INTEGER | Số thứ tự sự kiện trong aggregate | 1, 2, 3, ... |
| timestamp | TIMESTAMPTZ | Thời gian sự kiện xảy ra | 2025-08-16 10:30:00+07 |
| correlation\_id | UUID | ID để nhóm các events liên quan | corr-456 |
| causation\_id | UUID | ID của event gây ra event này | event-789 |
| metadata | JSONB | Metadata bổ sung (user, IP, etc.) | {"userId": "user-001", "source": "web"} |

**Constraints và Validation:**

* **UNIQUE(aggregate\_id, version)**: Đảm bảo không có 2 events cùng version cho 1 aggregate
* **CHECK (version > 0)**: Version phải là số dương
* **CHECK (jsonb\_typeof(event\_data) = 'object')**: event\_data phải là JSON object

**2. Bảng snapshots - Performance Optimization**

Bảng lưu trữ "ảnh chụp nhanh" trạng thái aggregate để tối ưu hóa performance.

CREATE TABLE snapshots (

aggregate\_id VARCHAR(255) PRIMARY KEY,

aggregate\_type VARCHAR(100) NOT NULL DEFAULT 'Order',

version INTEGER NOT NULL,

snapshot\_data JSONB NOT NULL,

timestamp TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

CHECK (version > 0),

FOREIGN KEY (aggregate\_id) REFERENCES events(aggregate\_id) DEFERRABLE

);

**Mục đích:**

* Giảm thời gian rebuild state cho aggregates có nhiều events
* Thay vì replay từ event đầu tiên, chỉ cần load snapshot + replay events sau đó
* Đặc biệt hữu ích cho aggregates có hàng trăm/nghìn events

1. **Các công cụ có thể sử dụng và các bước cần thực hiện để cài đặt sơ đồ lưu trữ đề xuất.**
   1. **Công cụ cần thiết**

 **PostgreSQL**: Hệ quản trị CSDL chính.

 **pgAdmin hoặc DBeaver**: Quản lý & xem dữ liệu.

 **Docker**: Khởi chạy Postgres nhanh.

 **psql CLI**: Chạy script tạo bảng.

 **Schema Migration Tool**: Flyway hoặc Prisma Migrate (tùy dự án).

* 1. **Các bước cài đặt chi tiết**

**Bước 1 – Cài PostgreSQL**

* **Docker cách nhanh nhất**:

docker run -d --name eventstore-db \

-e POSTGRES\_USER=postgres \

-e POSTGRES\_PASSWORD=123456 \

-e POSTGRES\_DB=order\_management \

-p 5432:5432 postgres

* Hoặc cài local từ <https://www.postgresql.org/download/>

**Bước 2 – Tạo schema lưu trữ**

Dùng file schema.sql trong thư mục Order-management/database, nội dung:

-- Order Management Event Store Schema

-- This script creates the necessary tables for the Event Sourcing pattern

-- Create database (run this separately if needed)

-- CREATE DATABASE order\_management;

-- Connect to the database

\c order\_management;

-- Create extension for UUID generation

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Event Store table

CREATE TABLE IF NOT EXISTS events (

    id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

    aggregate\_id VARCHAR(255) NOT NULL,

    aggregate\_type VARCHAR(100) NOT NULL DEFAULT 'Order',

    event\_type VARCHAR(100) NOT NULL,

    event\_data JSONB NOT NULL,

    version INTEGER NOT NULL,

    timestamp TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

    -- Ensure version uniqueness per aggregate

    UNIQUE(aggregate\_id, version)

);

-- Index for better query performance

CREATE INDEX IF NOT EXISTS idx\_events\_aggregate\_id ON events(aggregate\_id);

CREATE INDEX IF NOT EXISTS idx\_events\_aggregate\_type ON events(aggregate\_type);

CREATE INDEX IF NOT EXISTS idx\_events\_event\_type ON events(event\_type);

CREATE INDEX IF NOT EXISTS idx\_events\_timestamp ON events(timestamp);

-- Snapshots table (optional, for performance optimization)

CREATE TABLE IF NOT EXISTS snapshots (

    aggregate\_id VARCHAR(255) PRIMARY KEY,

    aggregate\_type VARCHAR(100) NOT NULL DEFAULT 'Order',

    version INTEGER NOT NULL,

    snapshot\_data JSONB NOT NULL,

    timestamp TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

-- Create a view for easy event querying

CREATE OR REPLACE VIEW event\_stream AS

SELECT

    id,

    aggregate\_id,

    aggregate\_type,

    event\_type,

    event\_data,

    version,

    timestamp

FROM events

ORDER BY aggregate\_id, version;

-- Insert some sample data for testing

INSERT INTO events (aggregate\_id, event\_type, event\_data, version) VALUES

('order-001', 'OrderCreated', '{"customerId": "customer-001", "items": [{"productId": "product-001", "productName": "Laptop Dell XPS", "quantity": 1, "price": 1500}]}', 1),

('order-001', 'OrderStatusUpdated', '{"status": "CONFIRMED"}', 2),

('order-002', 'OrderCreated', '{"customerId": "customer-002", "items": [{"productId": "product-002", "productName": "Mouse Wireless", "quantity": 2, "price": 25}]}', 1);

-- Function to get next version for an aggregate

CREATE OR REPLACE FUNCTION get\_next\_version(p\_aggregate\_id VARCHAR(255))

RETURNS INTEGER AS $$

DECLARE

    next\_version INTEGER;

BEGIN

    SELECT COALESCE(MAX(version), 0) + 1

    INTO next\_version

    FROM events

    WHERE aggregate\_id = p\_aggregate\_id;

    RETURN next\_version;

END;

$$ LANGUAGE plpgsql;

-- Function to append event with automatic versioning

CREATE OR REPLACE FUNCTION append\_event(

    p\_aggregate\_id VARCHAR(255),

    p\_event\_type VARCHAR(100),

    p\_event\_data JSONB,

    p\_expected\_version INTEGER DEFAULT NULL

)

RETURNS TABLE(event\_id UUID, version INTEGER) AS $$

DECLARE

    v\_event\_id UUID;

    v\_version INTEGER;

    v\_current\_version INTEGER;

BEGIN

    -- Get current version

    SELECT COALESCE(MAX(events.version), 0)

    INTO v\_current\_version

    FROM events

    WHERE aggregate\_id = p\_aggregate\_id;

    -- Check expected version if provided (for optimistic concurrency)

    IF p\_expected\_version IS NOT NULL AND v\_current\_version != p\_expected\_version THEN

        RAISE EXCEPTION 'Concurrency conflict: expected version %, current version %',

            p\_expected\_version, v\_current\_version;

    END IF;

    -- Calculate next version

    v\_version := v\_current\_version + 1;

    -- Insert the event

    INSERT INTO events (aggregate\_id, event\_type, event\_data, version)

    VALUES (p\_aggregate\_id, p\_event\_type, p\_event\_data, v\_version)

    RETURNING id INTO v\_event\_id;

    RETURN QUERY SELECT v\_event\_id, v\_version;

END;

$$ LANGUAGE plpgsql;

-- Grant permissions (adjust as needed)

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO postgres;

GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO postgres;

GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO postgres;

**Bước 3 – Import schema vào PostgreSQL**

# Cách 1: Dùng psql

|  |
| --- |
| psql -U postgres -h localhost -d order\_management -f schema.sql |

# Cách 2: Dùng pgAdmin/DBeaver → Run SQL Script

**Bước 4 – Kết nối ứng dụng**

# PostgreSQL connection string (alternative to individual DB\_\* variables)

DATABASE\_URL=postgresql://username:password@localhost:5432/database\_name

# Individual database configuration (used if DATABASE\_URL is not set)

DB\_HOST=localhost

DB\_PORT=5432

DB\_NAME=order\_management

DB\_USER=postgres

DB\_PASSWORD=your\_password\_here

**Bước 5 – Kiểm tra**

* Chạy migration test:

|  |
| --- |
| psql -U postgres -d order\_management -c "\dt" # Liệt kê bảng |

* Insert thử một event để kiểm tra:

|  |
| --- |
| INSERT INTO events (id, aggregate\_id, event\_type, payload, version)  VALUES (gen\_random\_uuid(), gen\_random\_uuid(), 'OrderCreated', '{"customerId":"test"}', 1); |

1. **Giải thích mã nguồn ghi và đọc các sự kiện từ hệ thống lưu trữ các sự kiện.**
   1. **Giai đoạn ghi – Từ API Request → Database**
      1. **API Controller (Entry Point)**

// File: Order-management/src/api/controller.ts

async createOrder(req, res) {

  try {

    const { customerId, items } = req.body;

    // Gọi Command Handler

    const orderId = await this.commandHandlers.handleCreateOrder({

      customerId, items

    });

    res.status(201).json({ success: true, data: { orderId } });

  } catch (error) {

    res.status(400).json({ success: false, error: error.message });

  }

}

* Đây là entry point của API, nhận request từ frontend.
* Gọi tới **Command Handler** để thực hiện logic nghiệp vụ và sinh sự kiện.
  + 1. **Command Handler (Business Logic)**

// File: Order-management/src/commands/handlers.ts

async handleCreateOrder(command) {

  // Tạo domain object

  const order = new Order(command.customerId, command.items);

  // Tạo event

  const event = {

    type: 'OrderCreated',

    aggregateId: order.id,

    version: 1,

    timestamp: new Date(),

    data: {

      orderId: order.id,

      customerId: order.customerId,

      items: order.items,

      status: order.status,

      totalAmount: order.totalAmount

    }

  };

  // Lưu event vào Event Store

  await this.eventStore.saveEvent(event);

  return order.id;

}

* Chuyển yêu cầu thành **Domain Object** (Order).
* Sinh ra một **Event** chứa toàn bộ thông tin cần lưu.
* Gửi Event tới tầng lưu trữ (Event Store).
  + 1. **PostgreSQL Event Store (Storage Layer)**

// File: Order-management/src/infrastructure/postgres-event-store.ts

async saveEvent(event) {

  const client = await this.pool.connect();

  try {

    await client.query('BEGIN');

    // Gọi database function để lưu event

    const result = await client.query(

      'SELECT \* FROM append\_event($1, $2, $3, $4)',

      [

        event.aggregateId,           // ID của aggregate

        event.type,                  // Loại event

        JSON.stringify(event.data),  // Dữ liệu dạng JSONB

        null                         // Expected version (auto)

      ]

    );

    await client.query('COMMIT');

    const { event\_id, version } = result.rows[0];

    console.log(`Event saved: ${event.type} v${version}`);

  } catch (error) {

    await client.query('ROLLBACK');

    throw error;

  } finally {

    client.release();

  }

}

* Sử dụng transaction để đảm bảo atomicity.
* Gọi PostgreSQL function append\_event để thêm event với version tăng dần.
* Ghi log kết quả lưu.
  + 1. **Database Function (SQL Level)**

-- File: Order-management/database/schema.sql

CREATE OR REPLACE FUNCTION append\_event(

    p\_aggregate\_id VARCHAR(255),

    p\_event\_type VARCHAR(100),

    p\_event\_data JSONB,

    p\_expected\_version INTEGER DEFAULT NULL

) RETURNS TABLE(event\_id UUID, version INTEGER) AS $$

DECLARE

    v\_event\_id UUID;

    v\_version INTEGER;

    v\_current\_version INTEGER;

BEGIN

    -- Lock để tránh concurrency issues

    PERFORM pg\_advisory\_xact\_lock(hashtext(p\_aggregate\_id));

    -- Tính version mới

    SELECT COALESCE(MAX(events.version), 0)

    INTO v\_current\_version

    FROM events WHERE aggregate\_id = p\_aggregate\_id;

    v\_version := v\_current\_version + 1;

    -- Insert event

    INSERT INTO events (aggregate\_id, event\_type, event\_data, version)

    VALUES (p\_aggregate\_id, p\_event\_type, p\_event\_data, v\_version)

    RETURNING id INTO v\_event\_id;

    RETURN QUERY SELECT v\_event\_id, v\_version;

END;

$$ LANGUAGE plpgsql;

* **pg\_advisory\_xact\_lock**: lock theo aggregate\_id để tránh race condition.
* Lấy **version** mới dựa trên version cao nhất hiện tại.
* Chèn event vào bảng events và trả về event\_id, version.
  1. **Giai đoạn đọc (READ) – Từ Event Store → Current State**
  2. **API Controller (Query Handler)**

async getOrder(req, res) {

  try {

    const { id } = req.params;

    // Load tất cả events của aggregate

    const events = await this.eventStore.getEvents(id);

    if (events.length === 0) {

      return res.status(404).json({ success: false, error: 'Not found' });

    }

    // Tái tạo state từ events

    const order = this.rebuildOrderFromEvents(events);

    res.json({ success: true, data: order });

  } catch (error) {

    res.status(500).json({ success: false, error: error.message });

  }

}

* Lấy toàn bộ events theo aggregateId.
* Nếu không có event → 404.
* Replay events để dựng lại trạng thái hiện tại.
  1. **Event Store Read Operation**

async getEvents(aggregateId) {

  const client = await this.pool.connect();

  try {

    const result = await client.query(

      `SELECT aggregate\_id, event\_type as type, event\_data as data,

              version, timestamp

       FROM events

       WHERE aggregate\_id = $1

       ORDER BY version ASC`,  // Quan trọng: sắp xếp theo version

      [aggregateId]

    );

    // Transform dữ liệu

    return result.rows.map(row => ({

      type: row.type,

      aggregateId: row.aggregate\_id,

      data: row.data,              // JSONB tự động parse

      timestamp: new Date(row.timestamp),

      version: row.version

    }));

  } finally {

    client.release();

  }

}

* Lấy danh sách event theo thứ tự **version tăng dần** để replay chính xác.
* PostgreSQL tự parse jsonb thành object.
  1. **State Reconstruction (Event Replay)**

rebuildOrderFromEvents(events) {

  // Sắp xếp events theo version

  const sortedEvents = [...events].sort((a, b) => a.version - b.version);

  // Xử lý logic rollback (nếu có)

  const rollbackEvents = sortedEvents.filter(e => e.type === 'OrderRolledBack');

  let eventsToProcess = sortedEvents;

  if (rollbackEvents.length > 0) {

    // Logic phức tạp để xử lý rollback...

    // Có thể rollback theo version hoặc timestamp

  }

  // Replay events để tái tạo state

  let order = null;

  for (const event of eventsToProcess) {

    switch (event.type) {

      case 'OrderCreated':

        order = new Order(

          event.data.customerId,

          event.data.items,

          event.data.status,

          event.data.orderId

        );

        break;

      case 'OrderStatusUpdated':

        if (order) {

          order = order.updateStatus(event.data.newStatus);

        }

        break;

      case 'OrderItemAdded':

        if (order) {

          order = order.addItem(event.data.item);

        }

        break;

      case 'OrderItemRemoved':

        if (order) {

          order = order.removeItem(event.data.productId);

        }

        break;

    }

  }

  return order;

}

* Replay events theo thứ tự version.
* Mỗi event gọi method tương ứng trong Order Aggregate.
* Nếu có OrderRolledBack thì áp dụng logic khôi phục trạng thái.

export class Order {

  public readonly id: string;

  public readonly customerId: string;

  public readonly items: OrderItem[];

  public readonly status: OrderStatus;

  public readonly totalAmount: number;

  public readonly createdAt: Date;

  public readonly updatedAt: Date;

  constructor(

    customerId: string,

    items: OrderItem[],

    status: OrderStatus = OrderStatus.PENDING,

    id?: string,

    createdAt?: Date,

    updatedAt?: Date

  ) {

    this.id = id || uuidv4();

    this.customerId = customerId;

    this.items = items;

    this.status = status;

    this.totalAmount = this.calculateTotal();

    this.createdAt = createdAt || new Date();

    this.updatedAt = updatedAt || new Date();

  }

  private calculateTotal(): number {

    return this.items.reduce((total, item) => total + (item.price \* item.quantity), 0);

  }

  public updateStatus(newStatus: OrderStatus): Order {

    return new Order(

      this.customerId,

      this.items,

      newStatus,

      this.id,

      this.createdAt,

      new Date()

    );

  }

  public addItem(item: OrderItem): Order {

    const updatedItems = [...this.items, item];

    return new Order(

      this.customerId,

      updatedItems,

      this.status,

      this.id,

      this.createdAt,

      new Date()

    );

  }

  public removeItem(productId: string): Order {

    const updatedItems = this.items.filter(item => item.productId !== productId);

    return new Order(

      this.customerId,

      updatedItems,

      this.status,

      this.id,

      this.createdAt,

      new Date()

    );

  }

}