Chapter 07 Implementing Quries

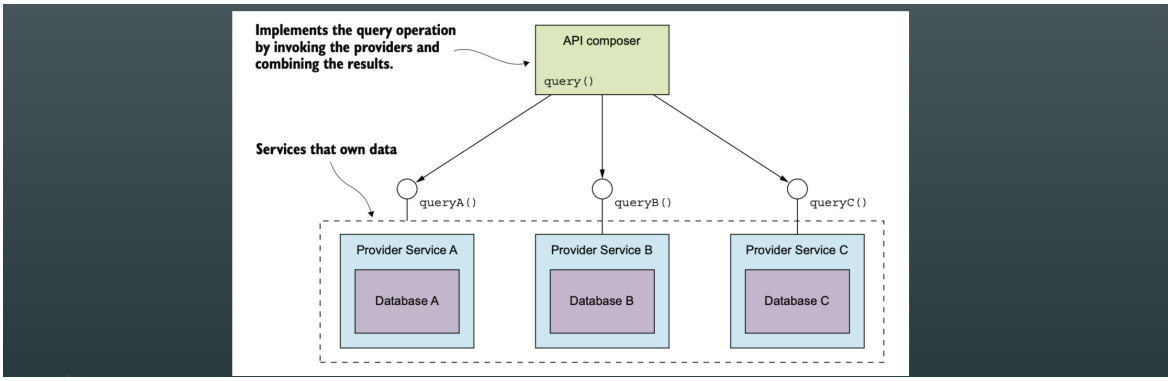
Overview of Querying Data

Querying data ใน Microservices นั้นยุ่งยากเพราะแต่ละ service มีหลาย database (มีการ join data และความ private ของ data ด้วย)

1. **API Composition Pattern** >> simplest
2. **Command Query Responsibility Segregation (CQRS) Pattern** >> more powerful but complex

API Composition Pattern

- Query data โดยเรียกแต่ละ service และนำ data ที่ได้มา combine กัน
- Structure ประกอบไปด้วย
  1. **API Composer** >> ทำ query operation (client หรือ API gateway)
  2. **API Provider** >> service ต่าง ๆ ที่ส่ง data กลับไปให้



- Implementation of query operation มีสิ่งที่ต้องคำนึงถึงอยู่
  1. How the **data is partitioned**
  2. The capabilities of the **APIs** >> ว่า API ของ service นั้นง่ายต่อการ query ขนาดไหน
  3. The capabilities of **databases** >> database บางประเภทอาจมีข้อจำกัด
  4. **In-memory join** of large datasets (Aggregator อาจต้องทำ)
- Design Issues and Solutions
  1. ตัดสินใจว่า**ให้ API Composer อยู่ที่ไหน**
     a. Client >> ภาระอยู่ที่ client ซึ่งเบา app เรา แต่มีปัญหาด้าน security และ performance
     b. API Gateway >> ปกติใช้อันนี้
     c. Stand-alone service >> ค่อนข้าง complex สำหรับ API gateway
  2. How to write **efficient aggregation logic**
     a. เขียน API Composer ด้วย **Reactive Programming Model**>> rely on events instead of order of lines in code (หมายความว่าการเรียก API Provider ควรเรียกแบบ parallel)
- Drawbacks of API Composition Pattern
  1. **Increased overhead** >> เพราะใช้ทรัพยากรทั้ง computing และ network ในการคำนวนและส่ง data กลับไปให้ client
  2. **Risk of reduced availability** >> service เยอะ โอกาสล่มสูง
  3. **Lack of transactional data consistency**

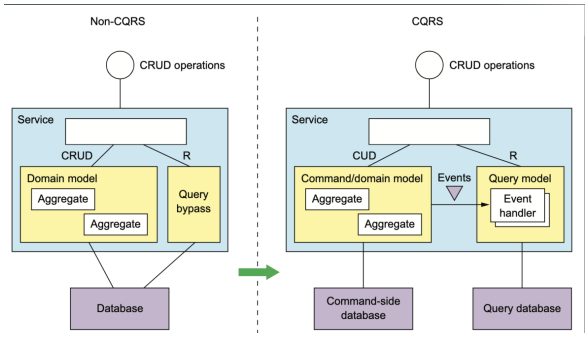
Querying Using CQRS Pattern

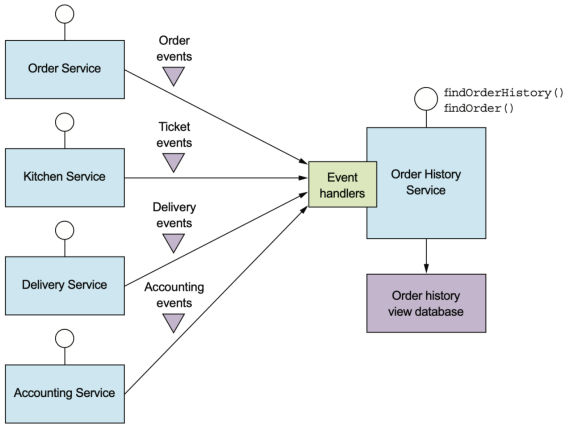
Motivation for Using CQRS Pattern

ใช้ **CQRS Pattern** เพราะ

- บางกรณีที่ต้องการ query แล้วมีการทำ filter หรือ sort ด้วย ซึ่งอาจมีกรณีที่ข้อมูลในบาง service ไม่สามารถทำ filter หรือ sort ด้วย field นั้น ๆ ได้
  1. **Do an in-memory join** >> fetch ทุกอย่างแล้วมา join ใน API composer (inefficient เพราะต้องมา join large datasets)
  2. **Matching then request by fetching ID** >> ทำทีละ steps (inefficient เพราะใช้ network traffic เยอะ)
- บางกรณีทำ query ใน single/local service ก็เป็นเรื่องยาก เพราะอาจเป็นเรื่องที่ไม่เหมาะสม ที่จะให้ service นั้น ๆ ทำ query นั้นเอง (เช่น return ข้อมูล log ทั้งหมด) หรือ data model อาจไม่ support

What is CQRS?

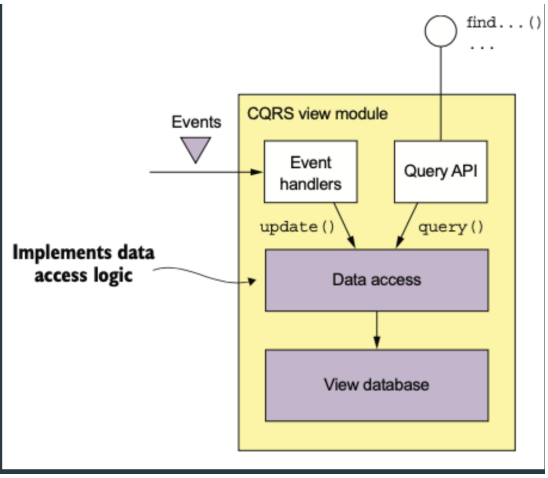




- **CQRS** แยกการ read/write โดยให้ write ทำกับ database นึง read ทำกับอีก database นึง
  - up-to-date ได้โดย subscribe event ที่เกิดขึ้นกับ services
  - write >> **RBDMS**, read >> **NoSQL**
- ข้อดี
  - Query มีประสิทธิภาพมากขึ้น
  - เขียน Query ได้หลากหลายขึ้น
  - ไม่ต้องทำ
- ข้อเสีย
  - More Complex Architecture >> developer ต้องเขียน query side เอง แถมอาจใช้ database คนละประเภท ทำให้จัดการได้ยาก
  - Replication lag >> event delay ทำให้มีปัญหากับ view update
    - Supply client with version information
    - Update its model using data returned by command

Design CQRS

- A CQRS view module consists of a view database and three submodules.
  - The data access module implements the database access logic.
  - The event handlers module subscribes to events and updates the database.
  - The query API module implements the query API.



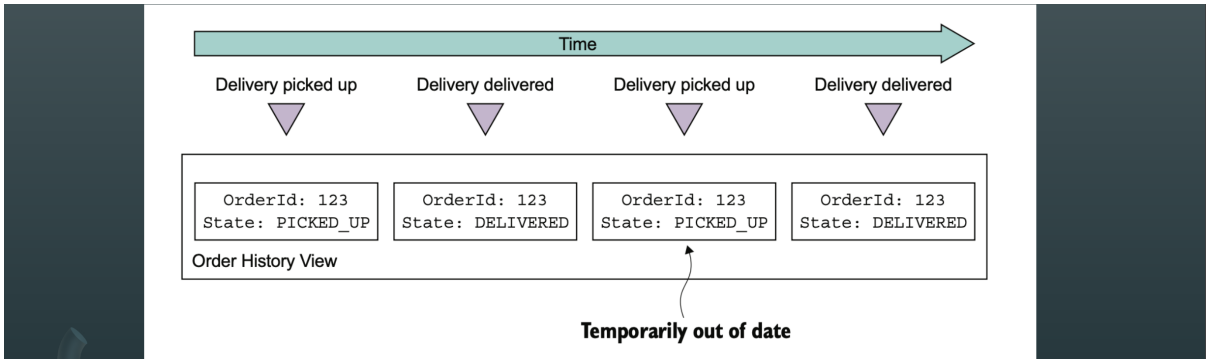
View module ประกอบไปด้วย 3 submodules และ database

- **Data access** >> implement database access logic
- **Event handlers** >> subscribes event
- **Query API**

(1) Choosing view datastore

- **SQL vs. NoSQL** >> NoSQL for CQRS view
- **Supporting update operation** >> เลือก database ที่ support PK, FK หรือ Indexing

(2) Data access module design



- **Handling concurrency** >> data access object (DAO) จะต้องจัดการ data consistency ให้ได้
- **Idempotent event handlers** >> handler โดนเรียกด้วย same event ดังนั้นจึงจะต้อง auto-discard ไม่ก็ไปเช็คเพื่อบอก user ว่า event ที่อยากได้นั้นไปถึงไหนแล้ว
- **Enable a client to use eventually consistent view** >> เพิ่ม token ให้กับ command-side operation เพื่อใช้ token นี้ในการเช็คว่า view update แล้วหรือยังตอนทำ query-side operation

(3) Adding and Updating CQRS view

- **Create a New View** >> Develop query-side module → Set up DB → Deploy service
- **Update an Existing View** >> Change event handler → Rebuild view or Update incrementally