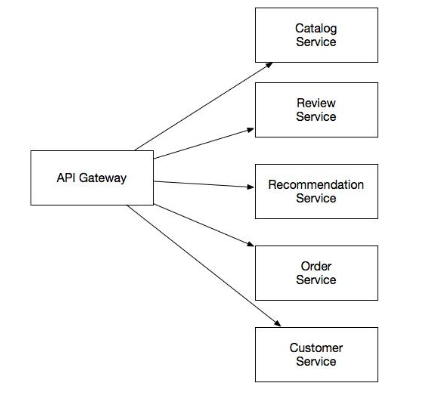
Microservices

* Arsitektur *Microservice* secara fungsional memecah menjadi bagian-bagian aplikasi kecil sesuai dengan kemampuan bisnis masing-masing service.
* Tantangan utama ketika mengembangkan arsitekture microservice adalah transaksi, domain model dan query resist decomposition.
* Domain model dapat didekomposisi menjadi agregat desain berbasis domain.
* Setiap logika bisnis layanan adalah model domain yang terdiri dari satu atau lebih agregat desain berbasis domain.
* Dalam satu layanan setiap transaksi membuat atau memperbaharui satu agregat.
* Acara digunakan untuk menjaga konsistensi antara agregat dan services.

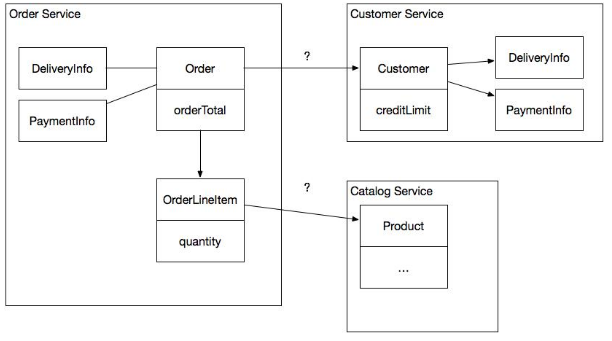
Microservices adalah pendekatan modularitas yang secara fingsional menguraikan atau membagi sebuah aplikasi menjadi bagian aplikasi kecil (Service kecil). Setiap aplikasi tersebut berfokus untuk menangani servicesnya sendiri.

Beberapa pendekatan yang digunakan untuk mengembangkan aplikasi microservices.

* Domain Driven Design.
* Event Sourcing and
* Command Query Responsibility Segregation (CQRS).



* Masalah pertama #1 - Decomposing a Domain Model



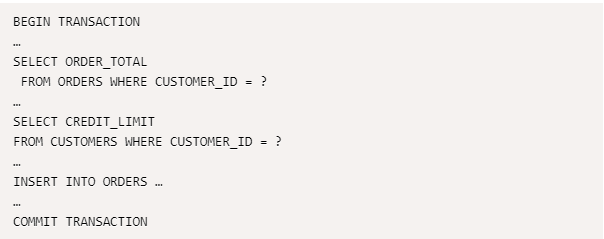
Tantangan dengan mendekomposisi model domain adalah bahwa kelas sering merujuk satu sama lain. Misalnya order mereferensikan customer dan OrderLineItem mereferensikan sebuah produk.

Microservices dan database

Dekomposisi fungsional dari basis data membuatnya sulit untuk mempertahankan konsistensi data dan untuk mengimplementasikan berbagai jenis queries.

* Masalah pertama #2 - Melaksanakan Transaksi Yang Meliputi Layanan

Pada arsitektur monolitik dapat mengandalkan ACID (*atomicity*, *consistency*, *isolation*, *durability*) untuk bisnis transaksnya. Misal pada aplikasi ecommerce saat transaksi ada pengecekan batas limit kartu kredit yang harus di check terlebih dahulu sebelum membuat transaksi. Jika pesanan (order) dan pelanggan (customer) di dalam database yang sama maka akan lebih mudah menggunakan transaksi ACID.

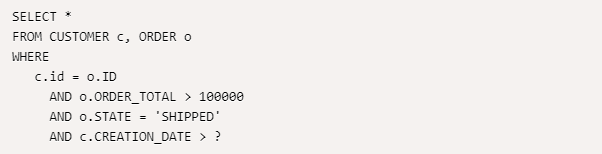


Sayangnya pengecekan menggunakan metode ACID tersebut tidak dapat diimplementasikan langsung pada arsitektur microservices. Karena terpisah nya table dari pesanan (order) dan pelanggan (customer) dipisahkan dalam database yang berbeda dan hanya dapat diakses melalui API.

Nanti Anda akan melihat bahwa solusinya adalah menggunakan arsitektur berbasis **event** berdasarkan teknik yang dikenal sebagai **event sourcing**.

* Masalah ketiga #3 - Querying and Reporting

Jika pada arsitektur monolith apabila ada kebutuhan untuk reporting misal untuk menemukan pelanggan baru dan pesanan besar menggunakan query berikut:



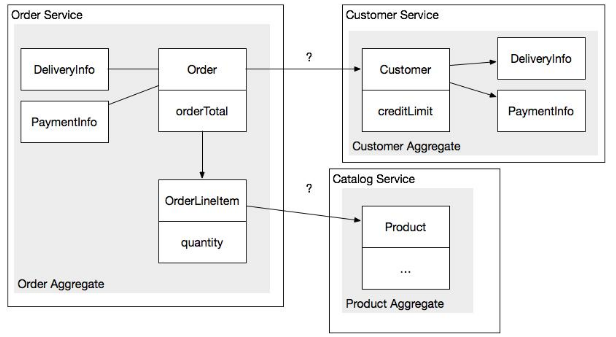
solusinya adalah *materialized views* menggunakan pendekatan yang dikenal sebagai Command Query Responsibility Segregation (CQRS)

DDD Aggregates are the Building Blocks of Microservices.

Apa yang dimaksud dengan aggregates?

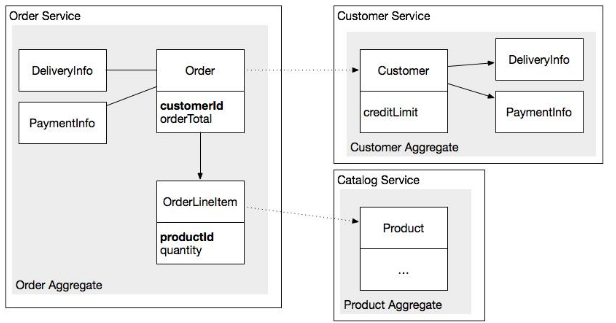
Agregat adalah sekelompok objek domain yang diperlukan sebagai satu unit. Terdiri dari entitas root dan mungkin satu atau lebih entitas terkait launya dan objek nilai.

Misalnya, model domain untuk toko online berisi agregat seperti Pesanan dan Pelanggan. Agregat Pemesanan terdiri dari entitas Pesanan (root), satu atau beberapa objek nilai OrderLineItem bersama dengan objek nilai lainnya seperti Alamat pengiriman dan Informasi Pembayaran. Agregat Pelanggan terdiri dari entitas root Pelanggan bersama dengan objek nilai lainnya seperti DeliveryInfo dan PaymentInformation.



* Inter-Aggregate References Must Use Primary Keys

Aturan pertama adalah yang mengagregasi referensi satu sama lain berdasarkan foreign key (mis. Kunci utama) penggati referensi objek. Misalnya, Pesanan merujuk Pelanggannya menggunakan CustomerId daripada referensi ke objek Pelanggan. Demikian pula, sebuah OrderLineItem mereferensikan suatu Produk menggunakan productId.



Pendekatan ini sangat berbeda dari pemodelan objek tradisional, yang menganggap kunci asing dalam model domain. Penggunaan identitas daripada referensi objek berarti bahwa agregat secara longgar digabungkan. Anda dapat dengan mudah menempatkan agregat berbeda di berbagai layanan. Bahkan, logika bisnis layanan terdiri dari model domain yang merupakan kumpulan agregat. Misalnya, OrderService berisi agregat Order dan CustomerService berisi agregat Pelanggan.

* One Transaction Creates or Updates One Aggregate

Aturan kedua yang harus dipatuhi oleh agregat adalah bahwa transaksi hanya dapat membuat atau memperbarui agregat tunggal.

Misalnya, sebelumnya saya menjelaskan bagaimana dalam model domain toko online, Order dan Pelanggan adalah agregat terpisah. Desain alternatif adalah menjadikan Pesanan sebagai bagian dari agregat Pelanggan. Manfaat agregat Pelanggan yang lebih besar adalah bahwa aplikasi dapat memberlakukan pemeriksaan kredit secara atomis. Kelemahan dari pendekatan ini adalah bahwa ia menggabungkan fungsi pesanan dan manajemen pelanggan ke dalam layanan yang sama. Ini juga mengurangi skalabilitas karena transaksi yang memperbarui berbagai pesanan untuk pelanggan yang sama akan diserialisasi. Demikian pula, dua pengguna mungkin konflik jika mereka mencoba mengedit pesanan yang berbeda untuk pelanggan yang sama. Selain itu, seiring dengan meningkatnya jumlah pesanan, semakin mahal untuk memuat agregat Pelanggan. Karena masalah-masalah ini, yang terbaik adalah membuat agregat sehalus mungkin.

Meskipun transaksi hanya dapat membuat atau memperbarui agregat tunggal, aplikasi tetap harus mempertahankan konsistensi antara agregat. Layanan Pemesanan harus, misalnya, memverifikasi bahwa agregat Pesanan baru tidak akan melebihi batas kredit agregat Pelanggan. Ada beberapa cara berbeda untuk menjaga konsistensi. Salah satu opsi adalah untuk menipu dan membuat dan / atau memperbarui beberapa agregat dalam satu transaksi. Ini hanya mungkin jika semua agregat dimiliki oleh layanan yang sama dan bertahan dalam RDBMS yang sama. Pilihan lainnya, yang lebih tepat adalah mempertahankan konsistensi antara kelompok unsur kehidupan dengan menggunakan pendekatan yang akhirnya konsisten dan didorong oleh peristiwa.