

NoSQL

Gold - Chapter 4 - Topic 5

Selamat datang di **Chapter 4 Topic 5**
online course **Back End Java** dari
Binar Academy!

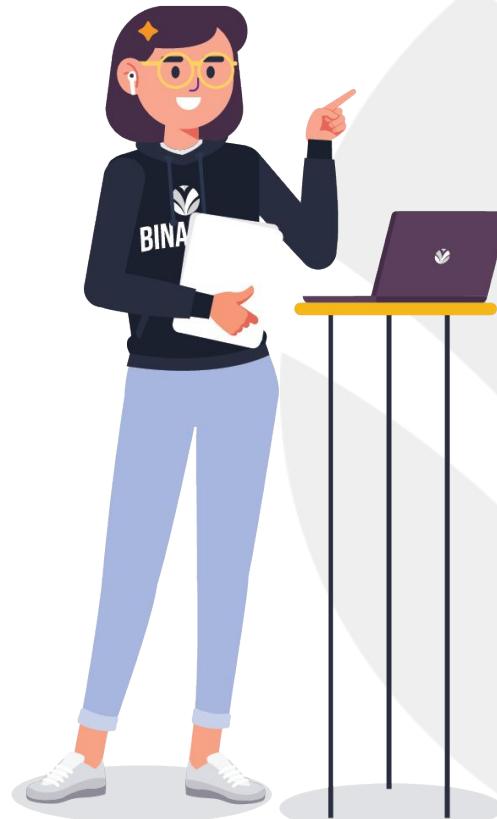


Dear diary, MySQL~

Siapa yang di sini masih inget tentang MySQL? serupa tapi tak sama pada topic baru ini, kita bakal mengelaborasi tentang lawan dari SQL yaitu **NoSQL**.

Merupakan kependekan dari Not Only SQL, ternyata doi punya sifat yang berkaitan sama penggunaan bahasa berbeda. Tapi sifat kayak gimana sih yang dimaksud?

Yuk, kita geser dulu materinya~



Dari sesi ini, kita bakal bahas hal-hal berikut:

- Jenis database NoSQL
- CRUD menggunakan implementasi MongoDB
- Konsep Caching
- Caching Strategy
- Caching dengan Redis



Kamu udah nggak asing sama istilah SQL dan NoSQL, kan?

Yap, sebelumnya kita udah pernah bahas di chapter 4.

Tapi di pembahasan kali ini, kita bakal pelajari lebih dalam lagi tentang **NoSQL types of Database**.



Sebelum bahas ke materi, mari kita flashback dulu~

Kamu masih ingat kan tentang SQL?

Kalau kita recall lagi, SQL adalah akronim dari Structured Query Language yang secara sederhana menggunakan SQL sebagai bahasanya, gengs.

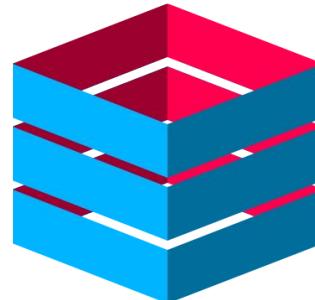


Ada SQL, ada juga NoSQL~

NoSQL ini merupakan kependekan dari Not Only SQL.

Not Only ya sob, bukan No atau Not SQL.

Istilah ini dipakai untuk menunjukkan database selain relational database SQL.

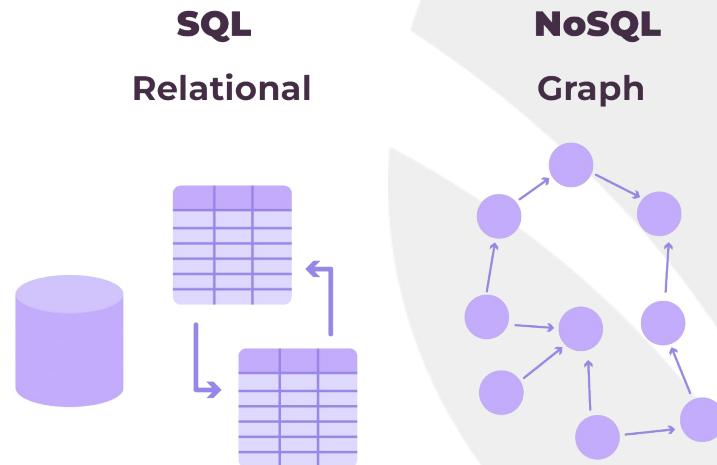
The word "nosql" in a large, lowercase, sans-serif font. The letters are colored in a gradient from red to blue. A soft gray shadow of the text is cast onto the surface below it.

“Jelasin bedanya SQL sama NoSQL, dong!”

NoSQL ini kebalikan dari SQL.

NoSQL nggak pakai bahasa SQL dalam melaksanakan query, atau bisa dibilang kalau **setiap database NoSQL punya bahasanya sendiri**.

Jadi, bahasanya nggak ada yang sama ya, gengs~



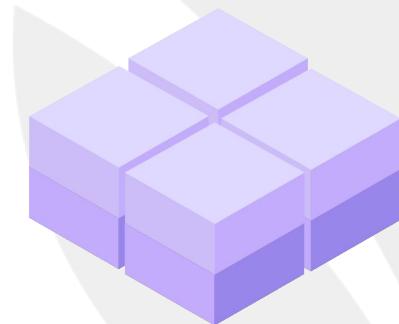
“Kenapa sih harus pakai database NoSQL?”

Jadi gini, pas menerapkan microservices, database NoSQL bisa dipakai untuk berbagai keperluan supaya bisa meningkatkan performance, robustness dan juga scalability.

SQL
Vertical Scalability



NoSQL
Horizontal Scalability



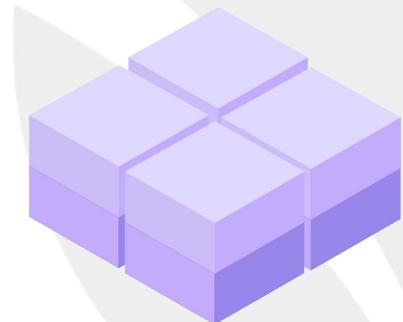
Selain itu, ada beberapa kasus yang lebih mudah diselesaikan dengan struktur database selain relational database.

Makanya, penting banget nih bagi seorang back end engineer untuk tahu database selain relational database~

SQL Vertical Scalability



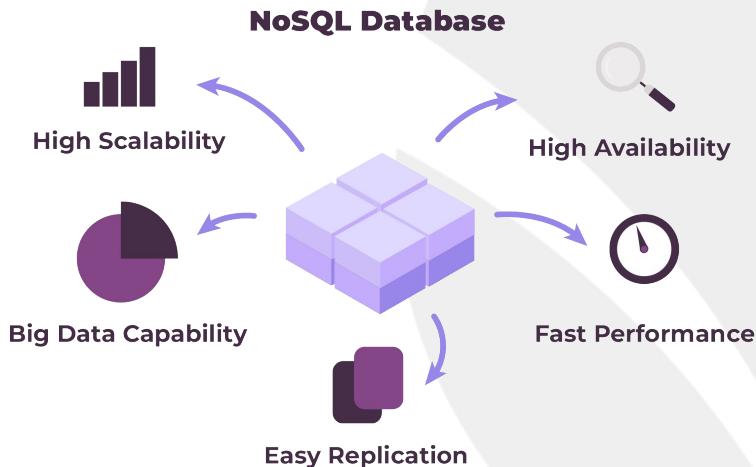
NoSQL Horizontal Scalability



“Terus apakah NoSQL bisa menggantikan relational database sepenuhnya?”

Jawabannya adalah enggak! Biasanya NoSQL ini dipakai untuk kebutuhan yang spesifik banget, contohnya:

- Menyimpan data secara sementara, tapi punya performance yang cepat.
- Menyimpan data yang performancenya cepat dengan data yang nggak terlalu kompleks.
- Menyimpan data dengan kolom yang nggak terhingga.



Berikut contoh-contoh jenis NoSQL!

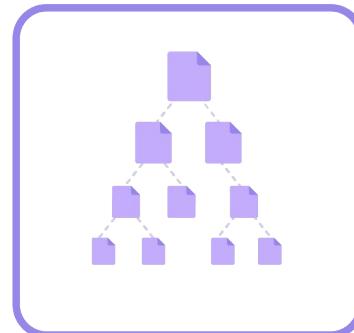
- **Document Oriented Database**

Merupakan database yang datanya disimpan dalam bentuk file document yang disimpan di dalam suatu storage.

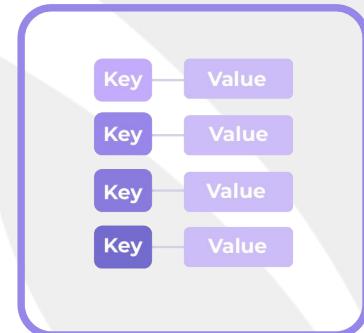
- **Key-Value Database**

Merupakan database yang menyimpan datanya di dalam suatu memory dalam bentuk key and value dan menyimpan data secara sementara.

Document Store



Key-Value Store



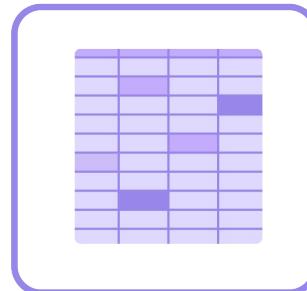
- **Wide Column Database**

Merupakan database yang memungkinkan penggunaan kolom secara dinamis.

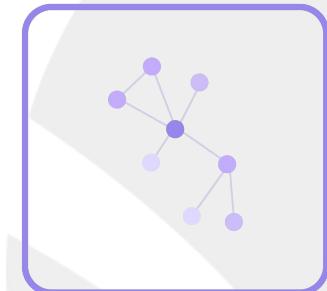
- **Search Database**

Merupakan database yang memungkinkan pencarian dengan menggunakan [Regex](#).

Wide Column Store



Graph Store



Search Database



Kalau ini adalah contoh software database yang berbasis NoSQL ya!

- MongoDB
- Redis
- Apache Cassandra
- Apache HBase
- Bigtable
- Elasticsearch



Software database NoSQL kalau dikelompokkan menurut jenisnya, kategorinya kayak gini nih~

Jenis NoSQL	Software
Document Oriented Database	MongoDB
Key-Value Database	Redis
Wide Column Database	Apache Cassandra
Search Database	Elasticsearch

NoSQL merupakan lawan dari MySQL dilihat dari penggunaan bahasa yang berbeda.

Terus, tadi disebutin tentang software NoSQL yang salah satunya adalah **MongoBD**.

Nyambung nih ke pembahasan kita, selanjutnya ada **CRUD Menggunakan Implementasi MongoDB**.



MongoDB itu apa, sih?

MongoDB adalah salah satu jenis database NoSQL yang cukup populer dipakai dalam pengembangan website.

Beda sama database jenis SQL yang menyimpan data pakai relasi tabel, **MongoDB pakai dokumen dan collection dengan format JSON**.



MongoDB punya beberapa keunggulan, lho!

Keunggulan MongoDB tuh kayak gini:

- Nggak ada downtime.
- Performanya cepat.
- Mudah buat dipakai.
- Mampu menyimpan data dengan bentuk yang kompleks.
- Scalability yang baik.



Tapi dibalik kelebihannya, MongoDB juga punya kekurangan, sob!

Kekurangan dari MongoDB tuh begini:

- **Nggak bisa menyimpan relasi.**
- **Nggak mendukung transaksi.**
- **MongoDB juga punya size yang terbatas.**

Jadi, buat database yang nggak memerlukan relasi yang begitu kompleks dan nggak butuh transaksi, mongoDB bisa jadi database alternatif.



Buat menambahkan MongoDB di project, kita bisa menambahkan dependency berikut, sob.

```
● ● ●  
<dependency>  
    <groupId>org.springframework.data</groupId>  
    <artifactId>spring-data-mongodb</artifactId>  
    <version>3.0.3.RELEASE</version>  
</dependency>
```

Buat pakai MongoDB di local, kita harus meng-install MongoDB di local dulu.

Caranya kayak gimana? Coba simak referensi buat menginstall mongoDB berikut, ya.

[Install MongoDB](#)



Setelah melakukan instalasi, jangan lupa buat menambahkan konfigurasi berikut, ya!

```
spring.data.mongodb.host=localhost  
spring.data.mongodb.port=27017  
spring.data.mongodb.database=namaDB
```

Kalau menggunakan password, kamu bisa menambahkan code berikut~



```
spring.data.mongodb.username  
spring.data.mongodb.password
```

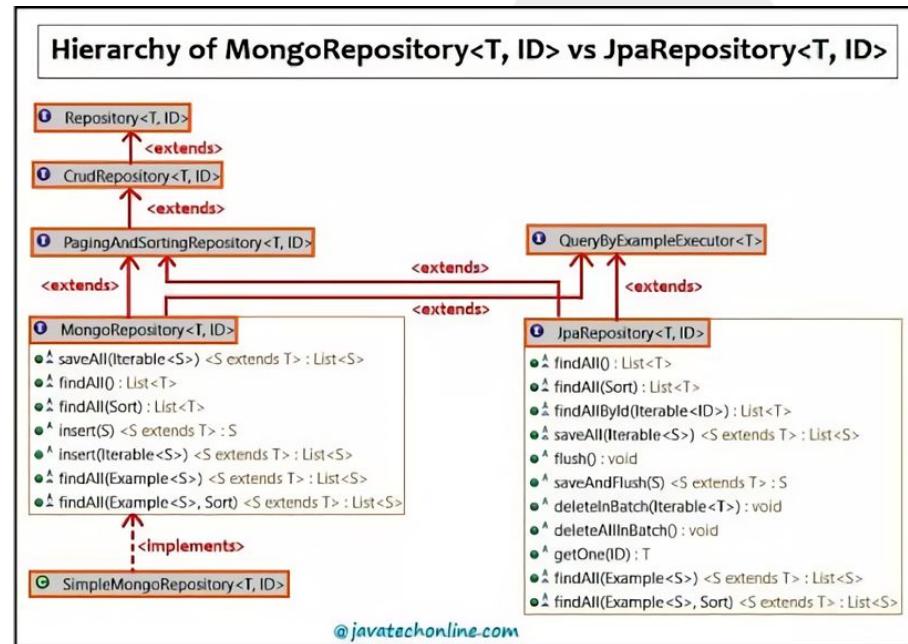
Terus, kamu bisa bikin entity kayak contoh di bawah, gengs~

Dilirik dulu, ya~

```
● ● ●  
  
@Data  
@Document           // Maps Entity class objects to JSON formatted Documents  
public class Book {  
  
    @Id                // making this variable as ID, will be auto-generated by MongoDB  
    private String id;  
  
    @NonNull  
    private Integer bookId;  
    @NonNull  
    private String bookName;  
    @NonNull  
    private String bookAuthor;  
    @NonNull  
    private Double bookCost;  
}
```

Karena pakai Spring Data, buat bikin DAO nya kita bisa pakai MongoRepository yang mirip kayak JpaRepository.

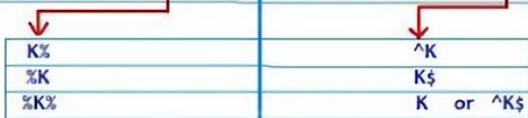
Kalau pakai repository ini, kita udah bisa menjalankan operasi CRUD dasar.



Berikut adalah contoh dari repository MongoDB ya, bestie~

```
● ● ●  
public interface BookRepository extends MongoRepository<Book, String> {  
}
```

Kalau kita mau pakai query customized, kita bisa menggunakan annotation `@Query` kayak contoh berikut:

MongoDB Query vs SQL Query	
SQL Query	MongoDB Query
Select * from BOOK where id = ?	<code>@Query("{id :?o}")</code>
Select * from BOOK where pages = ?	<code>@Query("{ pages :?o }")</code>
Select * from BOOK where pages < ?	<code>@Query("{pages : {\$lt: ?o}}")</code>
Select * from BOOK where pages >= ?	<code>@Query("{ pages : { \$gte: ?o } }")</code>
Select * from BOOK where author = ? and cost = ?	<code>@Query("{ \$and :[{author: ?o}, {cost: ?1}] }")</code>
Select count(*) from BOOK where author = ?	<code>@Query(value = "{author: ?o}", count=true)</code>
Select * from BOOK where author LIKE ''	<code>@Query("{ author : { \$regex : ?o } })"</code>
 <p>Diagram illustrating the equivalence between SQL's LIKE operator and MongoDB's \$regex operator. It shows two rows of patterns: 'K%' and '%K'. Red arrows point from these patterns to their MongoDB equivalents: '^K' and 'K\$' respectively, with the additional note 'or' between them.</p>	

Setelah bahas tentang MongoDB, kita bakal beranjak ke **Konsep Caching**.

Caching ini masih berkaitan sama istilah cache, lho.

Iya, yang kalau aplikasi di smartphone kita error atau penuh, kita disuruh hapus cache. Iya, yang kayak gitu!

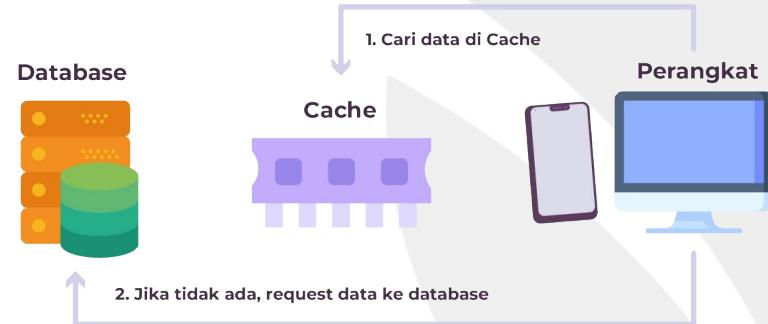


Kita bahas tentang Cache dulu, ya~

Cache merupakan tempat penyimpanan dengan performa yang cepat banget disimpan di dalam suatu memori.

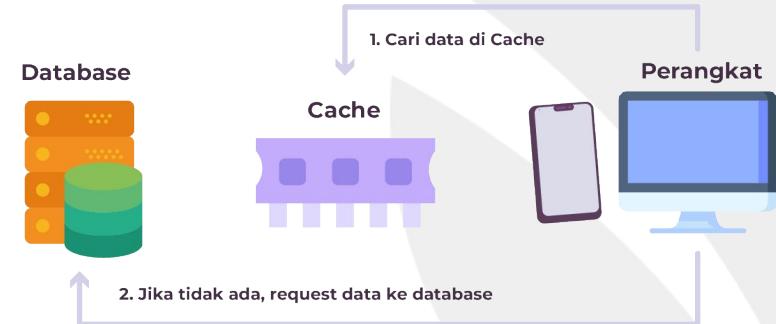
Cache menyimpan data secara sementara. Artinya, kalau aplikasi mati, bisa aja data tersebut hilang.

Oleh karena itu, cache menggunakan memori RAM.



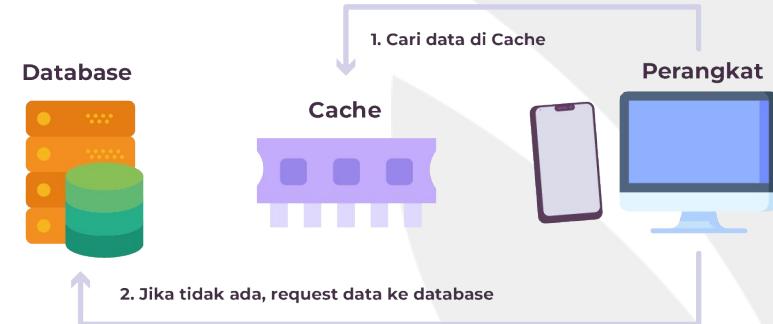
Tujuan dari penggunaan cache biasanya supaya data yang berasal dari sumber aslinya bisa disimpan sementara.

Nggak perlu juga tuh mengambil data yang sama lagi di sumber aslinya sehingga bisa meningkatkan performance dari suatu program.



Selain itu, cache juga bisa dipakai untuk keperluan menyimpan data secara sementara di local aja.

Proses buat menyimpan suatu data supaya jadi cache dinamakan caching.



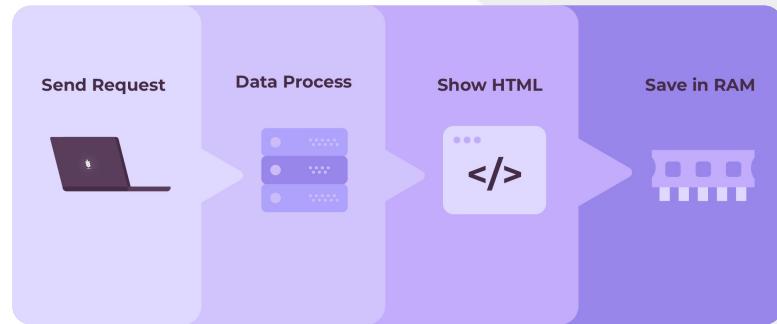
Bahas konsepnya udah, lanjut ke contoh penerapannya ya!

- **Contoh caching pada tampilan**

Request dikirim melalui front-end layer, terus dilakukan pemrosesan data oleh back end layer.

Setelah itu, data tersebut disajikan oleh front end layer dalam bentuk HTML.

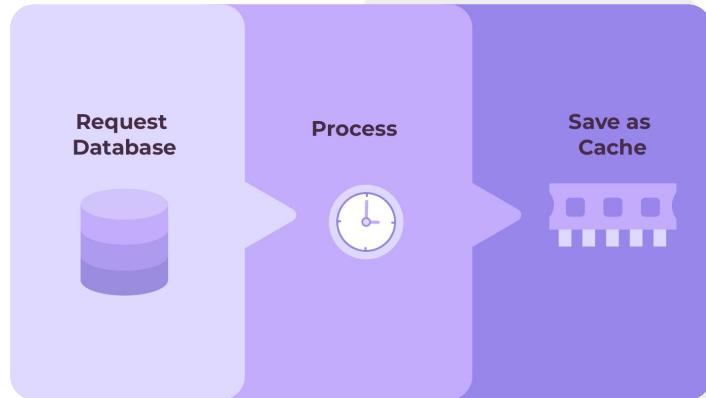
Karena cache pada proses caching diaktifkan, maka browser bakal menyimpan file HTML dalam RAM (Random Access Memory) sehingga pas mengakses website tersebut lagi, performanya bakal lebih cepat.



- **Contoh caching untuk data dari database**

Misalnya kita mengambil data dari sebuah database, nih.

Data tersebut memakan waktu yang cukup lama untuk dipanggil. Supaya data tersebut bisa dipakai lagi tanpa harus memakan waktu yang lama ke database utama, kita bisa menyimpan data tersebut dalam bentuk cache



Caching yang bakal dibahas di dalam topic ini adalah database caching ya, sob~

Untuk melakukan database Caching ada beberapa pattern yang biasa dipakai nih, yaitu:

1. Cache Aside
2. Write through cache
3. Read through cache
4. Write Back Cache



Kita bahas satu-satu mengenai database caching pattern di **Caching Strategy**.

Biar nggak makin penasaran nih, langsung kita intip aja, yuk! Berangkattt~

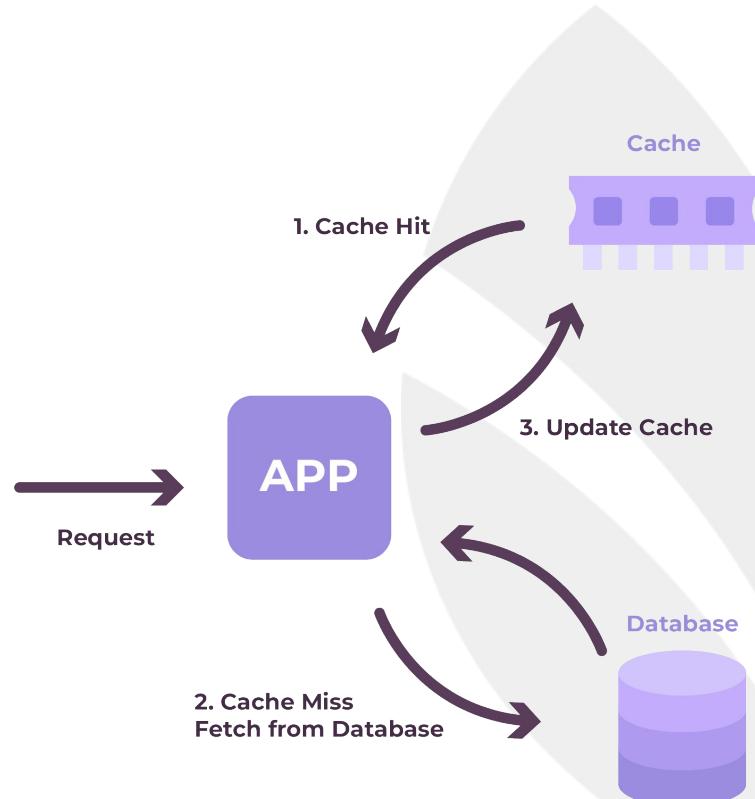


Yang pertama adalah Cache Aside~

Pada strategi Cache Aside, aplikasi bakal mengecek cache dulu untuk tahu apakah data pada cache tersedia atau nggak.

Kalau tersedia, maka data bakal diambil dari cache. Sedangkan kalau nggak tersedia, maka data bakal diambil dari database yang jadi sumbernya.

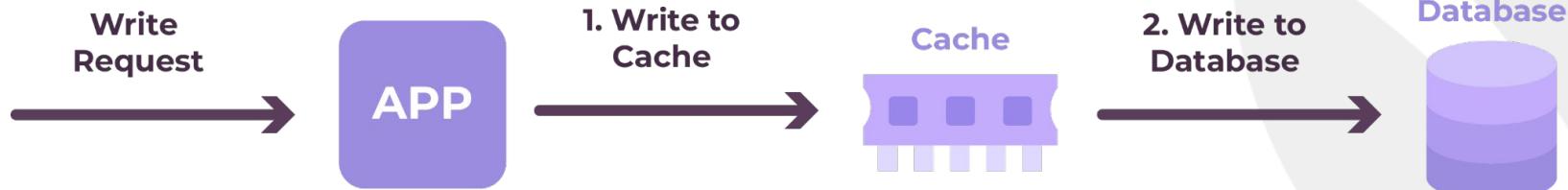
Meskipun membaca dari database sumber, cache tersebut bakal di-update sesuai dengan database.



Yang kedua yaitu Write Through Cache~

Pada write through cache, cache bakal di-update setiap kali ada data yang dituliskan di database sumber atau utama. Sehingga nantinya cache bakal selalu ter-update.

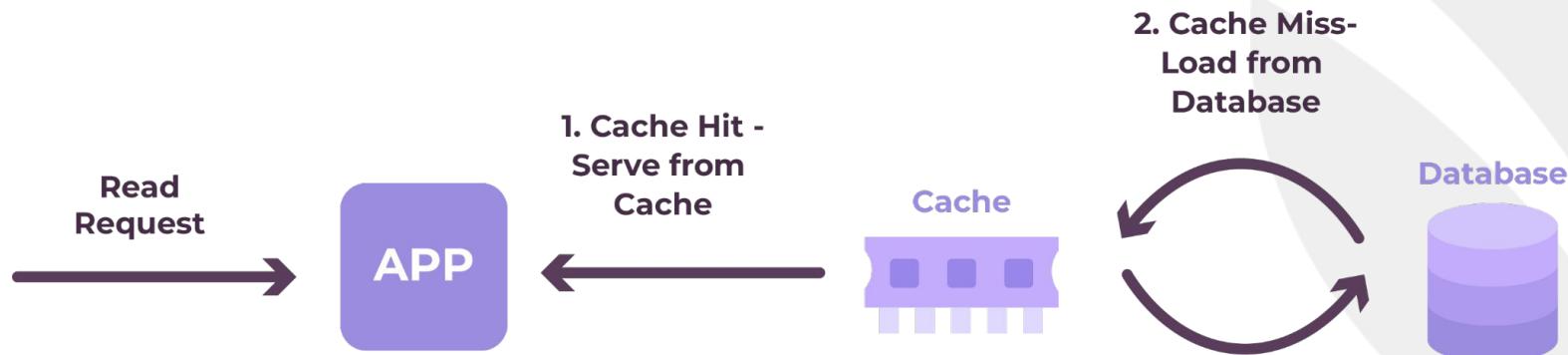
Proses ini terjadi secara sequential.



Yang ketiga ada Read Through Cache~

Read through cache mirip kayak cache aside.

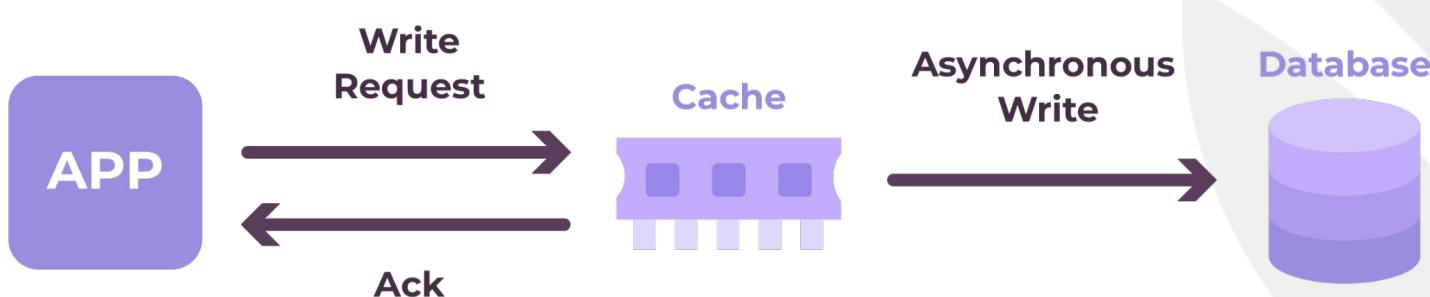
Bedanya, pas data nggak ditemukan pada cache, cache bakal di-update dulu dari database dan pengambilan data tetap menggunakan cache, bukan database sumber.



Yang terakhir ada Write Back Cache~

Di write back cache, program bakal menuliskan data pada cache, kemudian cache bakal memberitahukan apabila penulisan berhasil.

Cache bakal memproses penulisan ke database secara asynchronous.



Karena udah ada gambaran tentang caching secara lengkap, berarti sekarang kita coba cari tahu tentang **Caching dengan Redis**.

Belum familiar ya sama Redis? justru abis ini mau kita cari tahu, nih.

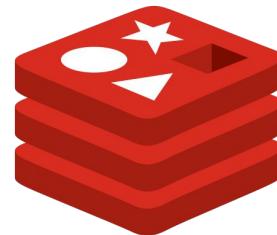


Jadi, Redis tuh gini, gengs...

Redis merupakan teknologi penyimpanan struktur data dalam memori yang berfungsi sebagai database, pengelolaan cache, message broker, dan thread.

Redis singkatan dari **Remote Dictionary Server** yang bersifat open source dan memungkinkan respon cepat dalam mengakses berbagai layanan game, finansial, kesehatan, atau IoT secara real-time.

Redis merupakan database noSQL yang menyimpan data dalam bentuk key dan value.



redis

Berikut kelebihan Redis~

- Nggak ada downtime.
- Performa cepat. lebih cepat dari MongoDB
- Mudah buat digunakan.
- Mampu menyimpan data dalam jumlah yang banyak.
- Scalability yang baik.
- Mudah buat digunakan dan di-setup karena bisa digunakan tanpa database server tambahan.



Kalau ini kekurangan dari Redis~

- Membutuhkan memory RAM.
- Menyimpan data secara sementara aja.



Pada topic ini kita bakal pelajari gimana penggunaan Redis untuk melakukan database caching.

Cache tersebut bakal dilakukan di local memory dari program yang kita bikin.



Buat menambahkan Redis supaya bisa melakukan database caching, kita bisa menambahkan dependency berikut pada pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

Berikut adalah contoh entity yang mau di cache~

```
@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Item implements Serializable {

    @Id
    String id;

    String description;
}
```

Penasaran sama repository-nya?

Berikut adalah repository-nya, ya~



```
public interface ItemRepository extends JpaRepository<Item, String> {  
}
```

Kalau ini adalah bean konfigurasi untuk Redis~

```
● ● ●

@Configuration
public class CacheConfig {

    @Bean
    public RedisCacheManagerBuilderCustomizer redisCacheManagerBuilderCustomizer() {
        return (builder) -> builder
            .withCacheConfiguration("itemCache",
                RedisCacheConfiguration.defaultCacheConfig().entryTtl(Duration.ofMinutes(10)))
            .withCacheConfiguration("customerCache",
                RedisCacheConfiguration.defaultCacheConfig().entryTtl(Duration.ofMinutes(5)));
    }

    @Bean
    public RedisCacheConfiguration cacheConfiguration() {
        return RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofMinutes(60))
            .disableCachingNullValues()
            .serializeValuesWith(SerializationPair.fromSerializer(new
GenericJackson2JsonRedisSerializer())));
    }

}
```

Sedangkan ini adalah implementasi service dari entity yang udah dibikin sebelumnya~

```
@Service
@AllArgsConstructor
public class ItemService {

    private final ItemRepository itemRepository;

    @Cacheable(value = "itemCache")
    public Item getItemForId(String id) {
        return itemRepository.findById(id)
            .orElseThrow(RuntimeException::new);
    }

}
```

Pengumuman, pengumuman □

Seperti biasa, sebelum kita move on ke Quiz. Silakan **lakukan caching menggunakan Redis!**

Latihan ini dilakukan di kelas dan silahkan diskusikan hasil jawaban kamu dengan teman sekelas dan fasilitator.

Selamat mencoba, yaa~



Abis belajar SQL, lanjut lagi NoSQL~

Karena kamu udah mempelajari keduanya,
Sabrina kepo deh. Lebih sulit mana, belajar SQL
atau NoSQL? Jelaskan juga alasannya dong.



Nah, selesai sudah pembahasan kita di Chapter 4 Topic 5 ini.

Selanjutnya, kita bakal bahas tentang Spring Data JPA Part 1.

Penasaran kayak gimana? Cus langsung ke topik selanjutnya~

