Keterangan menggunakan APcu sebagai Performance Tuning

1. **Mengurangi Beban Query Database**

   Caching melalui APCu membantu mengurangi jumlah query langsung ke database dengan menyimpan hasil query yang sering diakses ke dalam memori.

2. **Waktu Akses Lebih Cepat**

   Membaca dari memori (cache) jauh lebih cepat dibandingkan dengan melakukan query ke database yang membutuhkan waktu untuk koneksi, eksekusi query, dan pengambilan hasil. Dengan APCu, data disimpan langsung dalam memori server, membuat proses pengambilan data jauh lebih cepat. Ini penting untuk API yang membutuhkan respon cepat terhadap permintaan data yang berulang.

3. **Mengurangi Latensi**

   Karena APCu menyimpan data dalam memori lokal server, latensi akses data berkurang dibandingkan harus mengambil data dari database yang bisa memiliki latency lebih tinggi, terutama jika database berada di server yang terpisah.

4. **Skalabilitas**

   APCu dapat membantu meningkatkan skalabilitas aplikasi dengan mengurangi beban pada database saat API menerima banyak permintaan. Semakin sedikit query yang dilakukan, semakin sedikit beban yang ditanggung oleh database, memungkinkan sistem menangani lebih banyak permintaan secara bersamaan.

5. **Pengelolaan Cache yang Efisien**

   Dalam kasus ini, cache dihapus (invalidate) secara tepat setiap kali ada perubahan pada data di database, seperti di fungsi createAuthor(), updateAuthor(), deleteAuthor(), createBook(), dll. Dengan begitu, data yang tersimpan di cache selalu relevan dan up-to-date tanpa memberikan hasil lama dari cache.

6. **Simplicity dan Efisiensi untuk Aplikasi Skala Kecil hingga Menengah**

   APCu merupakan solusi caching yang ringan dan sederhana untuk aplikasi dengan skala kecil hingga menengah, yang tidak membutuhkan distribusi cache yang kompleks seperti Redis atau Memcached. Dengan menggunakan APCu, maka dapat menghindari kompleksitas instalasi dan pengelolaan sistem cache eksternal.

Improve jika data pada database sudah sangat banyak.

1. **Pagination untuk Query GET**
   Jika data dalam database sangat besar, memuat semua data dalam satu query bisa menyebabkan beban berat pada server. Sebaiknya gunakan pagination untuk membatasi jumlah data yang ditarik dalam satu permintaan.

2. **Indeks Database**
   Untuk mempercepat pencarian data di tabel besar, pastikan kolom yang sering digunakan dalam kondisi WHERE, JOIN, atau ORDER BY memiliki indeks.

3. **Full-text Search**
   Jika pencarian buku berdasarkan deskripsi, judul, atau bio penulis sering dilakukan, bisa dipertimbangkan untuk menggunakan Full-text Search untuk mempercepat pencarian teks yang besar.

   CREATE FULLTEXT INDEX idx_books_description ON tb_books (description);

4. **Optimasi Cache**
   Meskipun APCu caching sudah diterapkan, teknik caching yang lebih kuat seperti Redis atau Memcached dapat dipertimbangkan untuk memproses dan menyimpan data yang sering diakses, terutama dalam skala besar. Ini lebih efisien jika aplikasi di-deploy dalam lingkungan distributed.

5. **Sharding atau Partisi Database**
   Jika jumlah data terus meningkat, maka dapat mempertimbangkan sharding atau partisi untuk membagi data di beberapa tabel atau server. Misal, buku dapat dipartisi berdasarkan tahun terbit atau penulis.

6. **Asynchronous Processing (Job Queue)**
   Untuk operasi seperti membuat, memperbarui, atau menghapus data dalam jumlah besar, gunakan job queue untuk menangani tugas yang berat secara asinkron. Misalnya, jika ada operasi pembaruan massal, maka dapat menggunakan teknologi seperti RabbitMQ atau Amazon SQS untuk mengelola proses tersebut di luar permintaan utama.

7. **Load Balancer dan Horizontal Scaling**
   Jika traffic ke API sangat besar, pertimbangkan untuk menerapkan load balancer dan horizontal scaling untuk mendistribusikan beban ke beberapa server aplikasi. Pastikan sistem caching seperti Redis di-share antar node untuk konsistensi.