React Memoization - A Simple Guide

Pengertian

Memoization dalam React adalah teknik untuk mengoptimalkan performa aplikasi dengan cara menyimpan hasil perhitungan atau hasil rendering komponen. Ini sangat berguna ketika kita bekerja dengan data atau proses yang mahal dan tidak perlu dihitung ulang setiap kali komponen dirender ulang.

Secara umum, React memiliki dua cara utama untuk melakukan memoization:

- `React.memo()` – digunakan untuk mencegah komponen fungsional merender ulang jika props yang diterimanya tidak berubah.
- `useMemo()` dan `useCallback()` – digunakan untuk menyimpan hasil komputasi atau fungsi agar tidak dihitung ulang jika dependensinya tidak berubah.

Pada kode berikut, kita akan melihat bagaimana `memoization` diterapkan di berbagai tempat menggunakan `memo()`, `useMemo()`, dan `useCallback()` untuk meningkatkan efisiensi render.

Cara Berpikir React

Untuk memahami memoization dalam React, kita perlu memahami cara kerja React dalam merender komponen:

1. **Render ulang komponen**: React merender ulang komponen setiap kali ada perubahan state atau props.
2. **Penyederhanaan render**: Jika suatu komponen menerima props yang sama, React akan merender ulang komponen tersebut lagi secara default. Memoization mencegah render yang tidak perlu ini.
3. **Komponen fungsional dan state**: Komponen fungsional yang mengelola state akan merender ulang setiap kali state atau props-nya berubah.

Dengan memoization, kita dapat memberitahu React untuk **menghindari perhitungan atau render ulang** jika data yang digunakan tidak berubah.

Analogi Sederhana

Bayangkan kamu memiliki sebuah kalkulator yang bisa menghitung angka-angka besar dengan waktu yang lama, seperti menghitung angka faktorial. Ketika kamu menghitung 10!, kalkulator ini akan menghabiskan waktu yang cukup lama. Namun, jika kamu ingin menghitung 10! lagi, kalkulator akan mengingat hasil sebelumnya dan langsung memberikannya tanpa menghitung ulang. Begitu juga dalam React, dengan memoization, kita mengingat hasil perhitungan atau hasil render sebelumnya sehingga kita tidak perlu menghitung atau merender ulang jika data atau props yang digunakan tidak berubah.

Penjelasan Code Tiap Baris

Berikut adalah penjelasan baris per baris dari kode di atas, terutama yang berhubungan dengan memoization.

1. **Import Statement**

```
import { memo, useCallback, useEffect, useMemo, useState } from "react";
```

- `memo`: Fungsi yang digunakan untuk mencegah render ulang komponen fungsional jika props-nya tidak berubah.
- `useCallback`: Hook untuk memoize fungsi agar tidak dibuat ulang setiap kali komponen dirender.
- `useMemo`: Hook untuk memoize nilai yang dihasilkan dari sebuah perhitungan (seperti array atau objek).
- `useEffect`: Hook untuk efek samping (side effect) yang dijalankan setelah render.
- `useState`: Hook untuk menyimpan state dalam komponen.

2. `createRandomPost` Function

```
function createRandomPost() {
  return {
    title: `${faker.hacker.adjective()} ${faker.hacker.noun()}`,
    body: faker.hacker.phrase(),
  };
}
```

- Fungsi ini membuat post acak dengan menggunakan pustaka `faker`. Setiap kali dipanggil, menghasilkan post baru dengan `title` dan `body` yang berbeda.

3. Komponen `App`

```
function App() {
  const [posts, setPosts] = useState(() =>
    Array.from({ length: 30 }, () => createRandomPost())
  );
  const [searchQuery, setSearchQuery] = useState("");
  const [isFakeDark, setIsFakeDark] = useState(false);
```

- `posts`: Menyimpan array 30 post yang dihasilkan secara acak.
- `searchQuery`: Menyimpan kata kunci untuk pencarian.
- `isFakeDark`: Mengelola mode gelap palsu.

4. `searchedPosts` (Penyaringan Post)

```
const searchedPosts =
  searchQuery.length > 0
    ? posts.filter((post) =>
        `${post.title} ${post.body}`
          .toLowerCase()
          .includes(searchQuery.toLowerCase())
```

```
        )
      : posts;
```

- `searchedPosts`: Post yang akan ditampilkan setelah disaring berdasarkan `searchQuery`. Jika `searchQuery` kosong, maka semua post akan ditampilkan.

5. `handleAddPost` dengan `useCallback`

```
const handleAddPost = useCallback(function handleAddPost(post) {
  setPosts((posts) => [post, ...posts]);
}, []);
```

- `useCallback` digunakan untuk menyimpan fungsi `handleAddPost` agar tidak dibuat ulang setiap kali komponen dirender ulang. Fungsi ini hanya akan dibuat ulang jika dependensi (dalam hal ini, array kosong) berubah.

6. `useEffect` untuk Mengubah Mode Gelap

```
useEffect(
  function () {
    document.documentElement.classList.toggle("fake-dark-mode");
  },
  [isFakeDark]
);
```

- `useEffect` digunakan untuk mengubah class HTML ketika `isFakeDark` berubah, yang berfungsi untuk mengganti tampilan mode gelap.

7. `archiveOptions` dengan `useMemo`

```
const archiveOptions = useMemo(() => {
  return {
    show: false,
    title: `Post archive in addition to ${posts.length} main post`,
  };
}, [posts.length]);
```

- `useMemo` digunakan untuk menyimpan objek `archiveOptions` sehingga tidak dihitung ulang setiap kali komponen dirender. Dependensi yang mempengaruhi perhitungan adalah `posts.length`, sehingga hanya dihitung ulang jika jumlah post berubah.

8. `Archive` dengan `memo`

```
const Archive = memo(function Archive({ archiveOptions, onAddPost }) {
  const [posts] = useState(() =>
    Array.from({ length: 10000 }, () => createRandomPost())
  );
  const [showArchive, setShowArchive] = useState(archiveOptions.show);

  return (
    <aside>
      <h2>{archiveOptions.title}</h2>
      <button onClick={() => setShowArchive((s) => !s)}>
        {showArchive ? "Hide archive posts" : "Show archive posts"}
      </button>

      {showArchive && (
        <ul>
          {posts.map((post, i) => (
            <li key={i}>
              <p>
                <strong>{post.title}:</strong> {post.body}
              </p>
              <button onClick={() => onAddPost(post)}>Add as new
post</button>
            </li>
          ))}
        </ul>
      )}
    </aside>
  );
});
```

- `memo()` digunakan untuk membungkus komponen `Archive`. Ini berarti komponen `Archive` hanya akan dirender ulang jika props (`archiveOptions` dan `onAddPost`) berubah.
- `posts`: Sebuah array besar dengan 10.000 post yang dibuat saat pertama kali komponen dirender. Dengan menggunakan `useState`, post ini hanya dihitung sekali, dan tidak dihitung ulang saat komponen dirender ulang.
- `showArchive`: Mengontrol apakah archive akan ditampilkan atau tidak.

9. **Komponen lainnya** (Header, Main, Footer)

Komponen-komponen lainnya adalah bagian-bagian struktural yang tidak berpengaruh langsung terhadap memoization, namun mereka menggunakan state dan props untuk mengelola tampilan dan interaksi pengguna.

Kesimpulan

Memoization dalam React dapat membantu meningkatkan performa aplikasi dengan menghindari perhitungan atau render ulang yang tidak perlu. Di dalam kode ini, kita menggunakan `memo()`, `useMemo()`, dan `useCallback()` untuk memastikan bahwa hanya bagian-bagian yang benar-benar berubah yang akan dihitung ulang atau dirender ulang, sementara sisanya tetap menggunakan hasil sebelumnya.

- `memo()`: Menghindari render ulang komponen jika props tidak berubah.
- `useMemo()`: Menyimpan hasil perhitungan yang mahal agar tidak dihitung ulang.
- `useCallback()`: Menyimpan fungsi agar tidak diciptakan ulang pada setiap render.

Dengan mengoptimalkan aplikasi menggunakan teknik-teknik ini, kita bisa menciptakan aplikasi React yang lebih cepat dan responsif, terutama saat berurusan dengan data atau komponen yang sering direnderer.