System Design

Let's talk with Caching(ক্যাশিং)

Caching কী? (What is Caching?)

Caching হলো একটি Temporary Data Storage। যখনই কোন ডেটা বারবার দরকার হয়, তখন সেটি প্রতিবার Database বা External Source থেকে না এনে, একবার এনে Cache-এ রেখে দেওয়া হয়। পরে যখন একই ডেটা আবার দরকার হয়, তখন সেটা Cache থেকে নিয়ে নেওয়া হয়, ফলে সিস্টেম অনেক দ্রুত কাজ করে।

উদাহরণ:

- আপনি যদি ফেসবুক অ্যাপে কারো প্রোফাইল একবার দেখেন, পরের বার সেটি ক্যাশ থেকে দেখায়, ডেটাবেজে আবার হিট না করে।
- এটি performance বাড়ায় এবং latency কমায়।

CACHING কেন দরকার SYSTEM DESIGN-এ?

1. Performance বাড়ানোর জন্য ব্যবহার করা হয়।
2. Latency কমানোর জন্য ব্যবহার করা হয়।
3. Backend/database-এর উপর লোড কমানোর জন্য ব্যবহার করা হয়।
4. High traffic হ্যান্ডলিং করার জন্য ব্যবহার করা হয়।
5. স্ক্যাল্যাবিলিটি এর জন্য ব্যবহার করা হয়।
6. Cost efficiency (less DB/server resources)

CACHING কিভাবে কাজ করে

- Request → Cache Check → Hit → Serve Data
- Request → Cache Miss → DB Hit → Serve → Cache এ Store করে রাখা

LEARN HUNTER

কয় ধরনের CACHE আছে?

১. Application-level Cache
- Laravel, Django, Express ইত্যাদির মধ্যে থাকে
- উদাহরণ: Laravel Cache::remember()

২. In-Memory Cache
- RAM-based caching: super fast
- উদাহরণ: Redis, Memcached

৩. Database Cache
- Query Result Cache (MySQL Query Cache, ClickHouse Cache)
- Stored Procedure Cache

৪. Browser Cache
- HTML, CSS, JS Static Files ক্যাশে রেখে দেয়

৫. CDN Cache (Edge Caching)
- Cloudflare, Akamai: User-এর সবচেয়ে কাছের server থেকে static content serve করে

৬. Operating System Cache
- OS level page cache, disk cache etc

LEARN HUNTER

CACHE কখন ব্যবহার করা উচিত ?

1. API CALL করা মানে যে DATA আমাদের বারবার লাগে তখন। এটি আমাদের SPEED & DB LOAD কমায়
2. স্ট্যাটিক HTML/CSS/JS এসেটস এগুলা লোড করার জন্য যাতে করে সিস্টেম দ্রুত লোড হয়
3. প্রোডাক্ট ডিটেলস পেজ আছে যেটা বার বার একই ডাটা দেখাচ্ছে তখন যাতে করে বার বার ডাটাবেস কোয়েরি না লাগে বা হিট না করে
4. USER AUTH ইনফো এইটাকে শর্ট টাইম TTL করে রাখতে পারি যদি চাই

CACHE HIT & CACHE MISS

1. CACHE HIT = ক্যাশে DATA পাওয়া গেছে
2. CACHE MISS = ক্যাশে নেই, তাই DB/BACKEND থেকে আনতে হলো
3. EFFICIENT CACHE মানে HIGH HIT RATE

LEARN HUNTER

Cache Invalidation

Cache Invalidation হলো পুরোনো বা ভুল ডেটাকে ক্যাশ থেকে সরিয়ে নতুন ডেটা জায়গা করে দেওয়ার প্রক্রিয়া।
আচ্ছা ধরেন আমি যদি cache-এ একটা পুরোনো ডেটা রেখে দেয় , আর সেই ডেটা যদি backend/database-এ update হয়ে যায়, তখন cache-এর ডেটা আর ঠিক রইলো না।তারমানে ডাটাবেস এ ডাটা আপডেট হলো ঠিকই কিন্তু ভিজিটর ভুল ডাটা দেখলো।
এই ভুল/পুরোনো cache ডেটা মুছে ফেলা বা replace করাকে বলে Cache Invalidation.

উদাহরণ:

ইউজার প্রোফাইল page cache করে রাখা হলো। ইউজার profile update করলো – নাম পরিবর্তন করলো: "Sohel" → "Md. Sohel Rana" , কিন্তু cache-এ এখনো পুরোনো নাম "Sohel" আছে।যতক্ষণ না cache মুছবে (invalidate হবে), ততক্ষণ পুরোনো ডেটা দেখাবে।তাই update-এর পর cache invalidate করতে হয়।

Cache Invalidation Techniques:

1. Manual Invalidation : কোড দিয়ে Cache::forget('key') ব্যবহার করে ডেটা মুছে ফেলা।
2. Time-based Expiry (TTL) : নির্দিষ্ট সময় পরে cache নিজে নিজেই মুছে যায়
3. Write-through Cache : যখনই database update হয়, তখনই cache ও আপডেট হয়
4. Cache Busting : URL বা key change করে নতুন cache তৈরি করা
5. Event-driven : Laravel event/listener বা observer দিয়ে automatic invalidate

LEARN HUNTER

চলুন Caching Strategy গুলো দেখে নেই :

১. Write-through Cache

- DB তে write করে তারপর cache-এ write করে
- Data consistency ভালো

২. Write-around Cache

- সরাসরি DB তে write করে, cache-এ write করে না
- Low-write high-read scenario-তে ভালো

৩. Write-back Cache

- আগে cache-এ write করে, পরে DB তে sync করে
- Speed ভালো কিন্তু data loss হতে পারে

৪. Cache Aside / Lazy Load

- Cache এ নেই → DB hit → Cache এ রেখে দেয়
- Laravel এ remember() method এমনই

Cache Replacement Policies

নাম	কাজ
LRU (Least Recently Used)	পুরাতন যেটা use হয় নাই, সেটাকে replace করে
LFU (Least Frequently Used)	যেটা সবচেয়ে কমবার use হয়েছে সেটা বাদ
FIFO (First In First Out)	সবচেয়ে আগে যেটা ঢুকেছে, সেটি আগে বের হয়

কিছু ক্যাশিং টুলস দেখি :

Tool	Type	Use Case
Redis	In-memory, Key-Value Store	Fast, Persistent cache
Memcached	In-memory	Lightweight, distributed caching
Cloudflare CDN	Edge cache	Static files cache globally
Laravel Cache	App level	File, DB, Redis, etc.
ClickHouse Cache	DB query cache	Fast OLAP queries
Nginx FastCGI Cache	Web layer caching	Full page caching
Varnish	HTTP accelerator	Frontend caching

LEARN HUNTER

Caching-এর সুবিধা

- ✅ PERFORMANCE BOOST
- ✅ LOW LATENCY
- ✅ COST EFFECTIVE
- ✅ LESS DB LOAD
- ✅ BETTER USER EXPERIENCE

Caching-এর অসুবিধা

- ❌ STALE DATA SERVE হতে পারে
- ❌ CACHE INVALIDATION IS HARD
- ❌ MEMORY USAGE বেশি
- ❌ COMPLEX STRATEGY MAINTAIN করতে হয়

Extra Tips

TTL (TIME TO LIVE) সেট করা জরুরি
CACHE PRIORITY বুঝে স্ট্র্যাটেজি ঠিক করা
LOAD BALANCING-এর সাথে CACHE SYNC রাখতে হবে
CACHE MONITORING TOOLS: REDISINSIGHT, PROMETHEUS, GRAFANA

LEARN HUNTER

চলুন আমরা দেখি কখন কোনটা ব্যবহার করা ভালো হবে

PRODUCT DETAILS PAGE / BLOG PAGE (FREQUENTLY READ, RARELY CHANGED)
-> USE: APPLICATION CACHE + IN-MEMORY (REDIS)

CHECKOUT SUMMARY / CART CALCULATION (HIGH READ, REAL-TIME UPDATE)
-> USE: IN-MEMORY CACHE WITH SHORT TT

API RATE LIMITING / SESSION STORE
-> USE: REDIS (BECAUSE OF SPEED + ATOMIC OPERATIONS)

DASHBOARD STATISTICS (HIGH TRAFFIC, DAILY UPDATED)
-> USE: APPLICATION CACHE + SCHEDULED UPDATE (EVERY 5 MIN / HOURLY)

STATIC FILES (HTML, CSS, JS, IMAGES)
-> USE: CDN CACHE (CLOUDFLARE, AWS CLOUDFRONT)

SEARCH SUGGESTIONS / AUTO-COMPLETE RESULTS
-> USE: IN-MEMORY CACHE WITH TTL + FULL TEXT INDEX

MACHINE LEARNING / RECOMMENDATIONS
-> USE: IN-MEMORY CACHE WITH PERIODIC REFRESH

DATABASE QUERY RESULT CACHE
-> USE: DB LAYER CACHING (CLICKHOUSE/MYSQL/REDIS)

LEARN HUNTER

কখন Cache ব্যবহার করা উচিত না ?

Scenario	কেন cache use করবো না
Real-time stock price	প্রতি সেকেন্ডে update হয়
Banking transactions	Data consistency গুরুত্বপূর্ণ
Order placing	Data loss হলে বড় সমস্যা
Live chat messages	Fresh data না হলে UX খারাপ

শেষ কথা: