**Design Category system**

**场景**

* **设计一个 商品分类展示系统（Catalog / Category System），用户通过网页浏览商品。**
* **商品可以属于 多个分类（多对多关系）。**
* **分类可以有 多级层次（父分类、子分类），但前端每次只需要展示 直属子类。**

**用户需求**

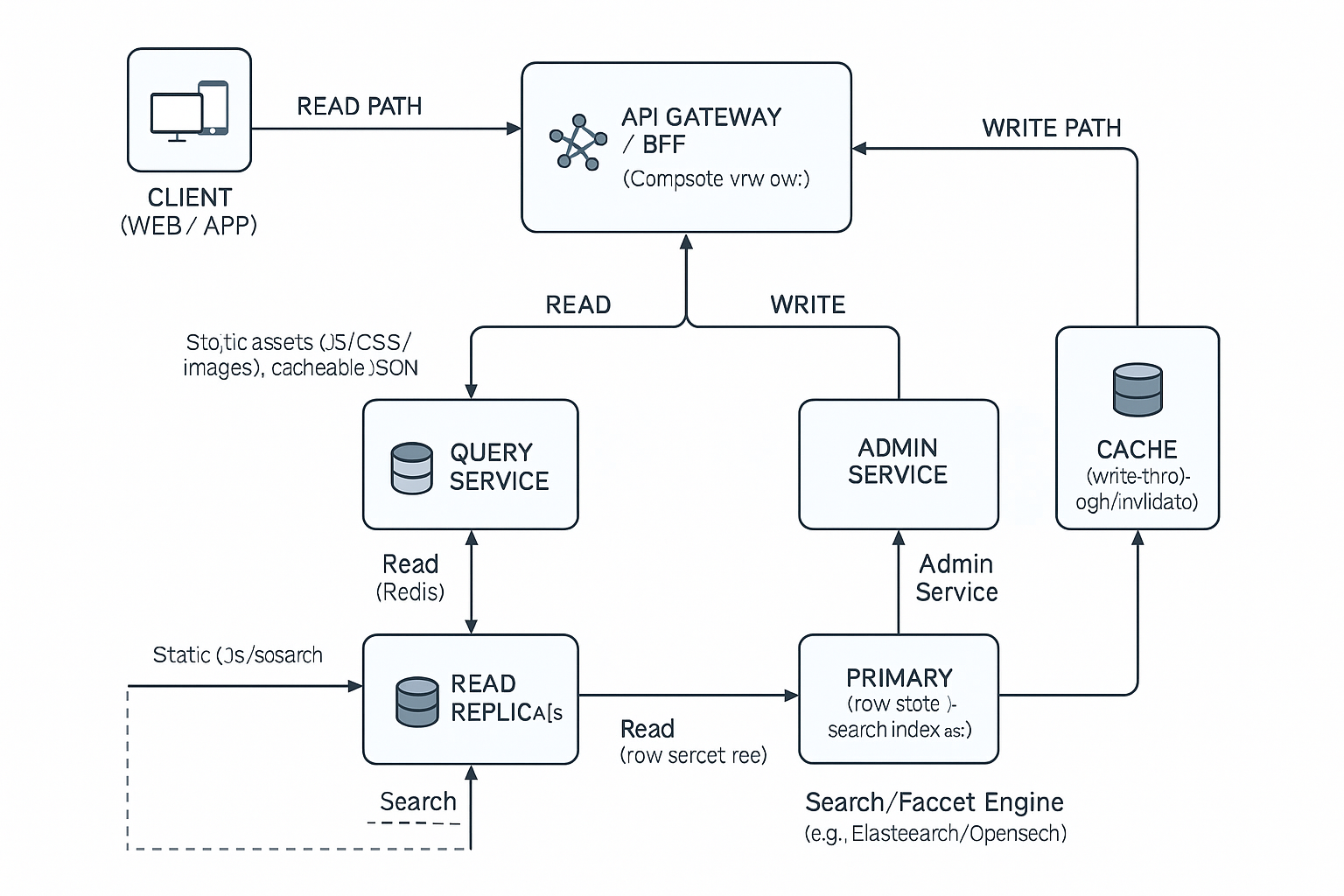
1. **用户打开网页时：**
   * **显示所有顶层分类（例如 Food、Clothes）。**
   * **显示部分商品（热门商品或顶层分类商品）。**
2. **用户点击某个分类：**
   * **显示该分类的 直属子分类。**
   * **显示该分类下的 商品。**
   * **不需要递归显示所有子孙分类（简化查询和缓存）。**
3. **商品可能属于 多个分类：**
   * **一个商品可能在不同分类页面同时出现。**

**面试考察点**

1. **数据库设计**
   * **如何存储分类（Category 表自关联实现树形结构）。**
   * **如何存储商品与分类的多对多关系（Product / ProductCategory 表）。**
2. **API 设计**
   * **如何设计请求与返回格式。**
   * **如何让前端方便地 fetch 分类和商品信息。**
3. **数据访问模式**
   * **如何查询顶层分类。**
   * **如何查询某分类下的直属子分类和商品。**
   * **如何处理商品属于多个分类的情况。**
4. **系统扩展性 / 高可用性（follow-up）**
   * **如何缓存分类和商品信息。**
   * **如何在大量用户访问时扩展系统。**
   * **如何保证系统稳定运行。**

**重点注意**

* **前半部分面试会偏重 API + 数据库 schema。**
* **后半部分可能会问 整体系统架构、Cache、Load Balancer、数据同步策略 等细节。**
* **面试中需要清楚表达：**
  1. **数据在数据库中的存储形式。**
  2. **请求格式和响应格式。**
  3. **前端 fetch 的逻辑。**

****

**1) Public Browse APIs｜公开浏览 API（用户侧）**

面向前端浏览；读多写少；强缓存与边缘缓存友好。  
Read-heavy, edge-cacheable JSON; direct-children only.

**1.1 List top-level categories｜列出顶层分类**

* **GET** /v1/categories?parent\_id=null&locale=en-US&currency=USD
* **Query**
  + parent\_id：null 表示顶层 | null means root
  + locale / currency：用于多语言/币种展示 | i18n & pricing
* **200 Response**

{

"data": [

{ "id": 1, "name": "Food", "slug": "food", "parent\_id": null, "children\_count": 12 },

{ "id": 2, "name": "Clothes", "slug": "clothes", "parent\_id": null, "children\_count": 9 }

],

"meta": { "locale": "en-US", "cache\_ttl\_sec": 600 }

}

* **Caching**：Cache-Control: public, max-age=600, stale-while-revalidate=120  
  CDN/Edge 建议缓存 10 分钟，可 SWR 2 分钟。

**1.2 Get a category node (direct children + products page 1)｜获取分类节点（直属子类 + 商品第一页）**

* **GET** /v1/categories/{id}?limit=24&page=1&locale=en-US&currency=USD&sort=popularity\_desc
* **200 Response**

{

"category": { "id": 10, "name": "Fruits", "slug": "fruits", "parent\_id": 1 },

"subcategories": [

{ "id": 11, "name": "Citrus", "parent\_id": 10 },

{ "id": 12, "name": "Berries", "parent\_id": 10 }

],

"products": [

{

"id": 101, "name": "Apple Gala", "image\_url": "...",

"price": { "currency": "USD", "list": 1.99, "final": 1.59 },

"badges": ["HOT"]

}

],

"paging": { "page": 1, "limit": 24, "total": 187, "next\_page": 2 },

"meta": { "sort": "popularity\_desc", "locale": "en-US" }

}

* **Notes｜说明**
  + 仅返回 **直属子类** 与该分类下 **直接挂载的商品**。
  + 支持 sort：popularity\_desc | price\_asc | price\_desc | newest.

**1.3 List products under category (paged)｜获取分类下商品（分页）**

* **GET** /v1/categories/{id}/products?limit=24&page=2&sort=price\_asc&filters=color:red,size:M
* **200 Response**

{

"data": [ /\* product array as above \*/ ],

"paging": { "page": 2, "limit": 24, "total": 187, "next\_page": 3 },

"applied\_filters": { "color": "red", "size": "M" }

}

* **Facet 注记**：如需前端筛选项，可另设  
  **GET** /v1/categories/{id}/facets 返回可用的 facet buckets。

**1.4 Search products (optional)｜搜索（可选）**

* **GET** /v1/search?q=apple&category\_id=10&limit=20&page=1
* 返回文本/属性匹配与可选 facet 聚合。Backed by search engine。

**1.5 Errors & rate limit｜错误与限流**

* **429** Too Many Requests（前端退避重试）
* **503** Service Unavailable（回退上次缓存/降级无 facet）
* **标准错误结构**：

{ "error": { "code": "RESOURCE\_NOT\_FOUND", "message": "Category not found." } }

**2) Admin APIs（写路径）｜管理端 API（写）**

走强鉴权，支持幂等、审计日志、精确缓存失效。  
Strong authN/Z, idempotency, audit.

* **POST** /v1/admin/categories 创建分类
* **PATCH** /v1/admin/categories/{id} 更新分类
* **DELETE** /v1/admin/categories/{id} 删除（软删更安全，标 is\_active=false）
* **POST** /v1/admin/products 创建商品
* **PUT** /v1/admin/products/{id}/categories 批量绑定分类（幂等）

**Headers**

* Idempotency-Key: <uuid>（服务端去重）
* 成功写入后：发布 MQ 事件 → 触发 **Redis 精准失效** 与 **异步重建索引**。

**3) Database Schema（核心表与索引）｜数据库设计**

以行存（PostgreSQL/MySQL）为真源，搜索引擎承接全文与聚合。  
Row store as SoT; search for text/facets.

**3.1 Category（自关联树）**

CREATE TABLE category (

id BIGINT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

slug VARCHAR(255) NOT NULL UNIQUE,

parent\_id BIGINT NULL,

display\_order INT DEFAULT 0,

is\_active BOOLEAN NOT NULL DEFAULT TRUE,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT fk\_category\_parent

FOREIGN KEY (parent\_id) REFERENCES category(id)

ON UPDATE CASCADE ON DELETE SET NULL

);

-- 索引：按父查子；按名/slug 查找

CREATE INDEX idx\_category\_parent ON category(parent\_id);

CREATE INDEX idx\_category\_active ON category(is\_active);

**说明**：满足“直属子类”高频查询；如要支持全量祖先/后代，可引入 *closure table* 或 *materialized path*（非本题刚需）。

**3.2 Product（商品）**

CREATE TABLE product (

id BIGINT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

slug VARCHAR(255) NOT NULL UNIQUE,

short\_description VARCHAR(512),

long\_description TEXT,

currency CHAR(3) NOT NULL,

list\_price DECIMAL(12,2) NOT NULL,

final\_price DECIMAL(12,2) NOT NULL,

is\_active BOOLEAN NOT NULL DEFAULT TRUE,

popularity\_score BIGINT DEFAULT 0, -- 排序用

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

CREATE INDEX idx\_product\_active ON product(is\_active);

CREATE INDEX idx\_product\_price ON product(final\_price);

CREATE INDEX idx\_product\_popularity ON product(popularity\_score DESC);

价格/人气打分字段用于排序；更复杂的定价可另建 Price 表或通过 BFF 组合。

**3.3 ProductCategory（多对多关联）**

CREATE TABLE product\_category (

product\_id BIGINT NOT NULL,

category\_id BIGINT NOT NULL,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (product\_id, category\_id),

CONSTRAINT fk\_pc\_product FOREIGN KEY (product\_id) REFERENCES product(id)

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT fk\_pc\_category FOREIGN KEY (category\_id) REFERENCES category(id)

ON UPDATE CASCADE ON DELETE CASCADE

);

-- 高频按类取商品、按商品取类的复合索引

CREATE INDEX idx\_pc\_category\_product ON product\_category(category\_id, product\_id);

CREATE INDEX idx\_pc\_product\_category ON product\_category(product\_id, category\_id);

**3.4 ProductAsset（可选：图片/视频）**

CREATE TABLE product\_asset (

id BIGINT PRIMARY KEY,

product\_id BIGINT NOT NULL,

url VARCHAR(1024) NOT NULL,

type ENUM('image','video') NOT NULL,

display\_order INT DEFAULT 0,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT fk\_asset\_product FOREIGN KEY (product\_id) REFERENCES product(id)

ON UPDATE CASCADE ON DELETE CASCADE

);

CREATE INDEX idx\_asset\_product ON product\_asset(product\_id, display\_order);

**4) Query Patterns（与 API 对应的典型查询）｜查询模式**

**4.1 顶层分类｜Top-level categories**

SELECT id, name, slug, parent\_id

FROM category

WHERE parent\_id IS NULL AND is\_active = TRUE

ORDER BY display\_order, id

LIMIT :limit OFFSET :offset;

**4.2 直属子类｜Direct children**

SELECT id, name, slug, parent\_id

FROM category

WHERE parent\_id = :category\_id AND is\_active = TRUE

ORDER BY display\_order, id;

**4.3 分类下商品（仅直属）+ 排序/分页｜Products in a category (direct only)**

SELECT p.id, p.name, p.slug, p.final\_price, p.popularity\_score

FROM product p

JOIN product\_category pc ON pc.product\_id = p.id

WHERE pc.category\_id = :category\_id

AND p.is\_active = TRUE

ORDER BY

CASE WHEN :sort = 'popularity\_desc' THEN p.popularity\_score END DESC,

CASE WHEN :sort = 'price\_asc' THEN p.final\_price END ASC,

CASE WHEN :sort = 'price\_desc' THEN p.final\_price END DESC,

p.id ASC

LIMIT :limit OFFSET :offset;

**4.4 计数（用于分页总数）｜Count for pagination**

SELECT COUNT(\*) AS total

FROM product p

JOIN product\_category pc ON pc.product\_id = p.id

WHERE pc.category\_id = :category\_id AND p.is\_active = TRUE;

**5) Cache Keys & TTL（与实现对齐）｜缓存键与过期**

* cat:{id}:meta → 分类基础信息 | category meta
* cat:{id}:children → 直属子类列表 | direct children
* cat:{id}:products:{page}:{sort}:{filters\_hash}:{locale}:{currency} → 商品页
* landing:hot:{locale}:{currency} → 首页“热门切片”

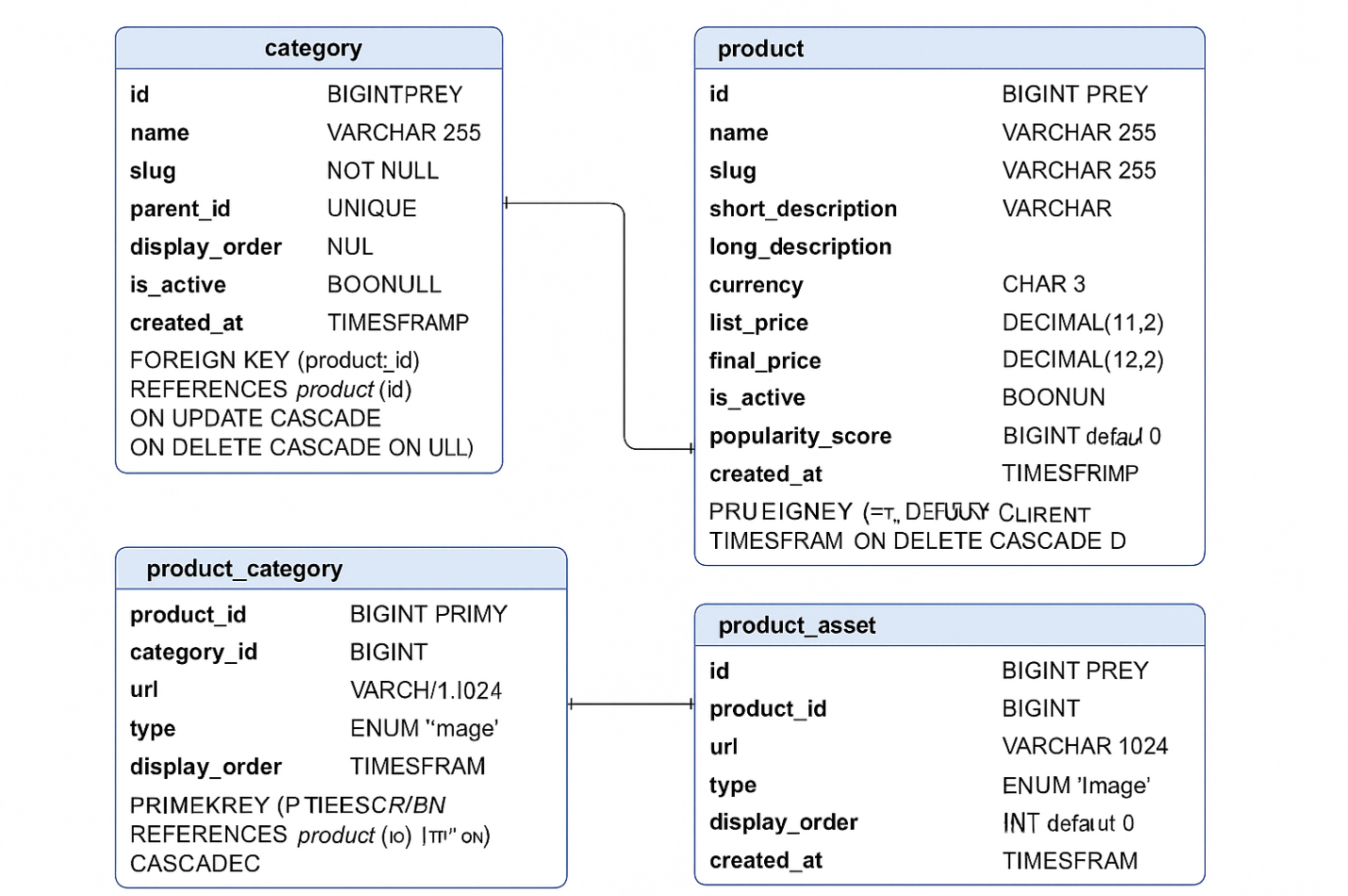
**TTL**：products 5–15 分钟；children/meta 15–30 分钟；landing 10–20 分钟  
**失效**：Admin 写入后，精确删除受影响键；同时发 MQ 事件重建搜索索引。

**6) API 版控与格式｜Versioning & Format**

* **Versioning**：路径前缀 /v1；向后兼容时新增字段置于对象末尾。
* **分页**：page + limit 或 cursor（推荐 cursor：更稳）。
* **排序/筛选**：白名单校验，避免 SQL 注入；面向搜索引擎时在 BFF 翻译为 DSL。
* **国际化**：Accept-Language 或 query locale；BFF 负责价格/货币组合。
* **安全**：公开接口只读；管理接口需 OAuth2/OIDC + RBAC；所有写接口支持 Idempotency-Key。

**7) 一页话术速记（30–60 秒）｜Interview Script Cheat**

“我们的浏览 API 只有三件事：拿顶层分类、拿某分类的**直属**子类、拿该分类**直接**挂载的商品（分页/可排序）。  
数据在行存做真源：category 自关联树、product、product\_category 多对多。  
读路径优先走 CDN/Redis，未命中再到只读副本；写路径经 Admin Service 落主库，同时精确失效缓存并异步重建搜索索引。  
缓存键包含 cat:{id}:children 与 cat:{id}:products:{page}:{sort}，产品列表 5–15 分钟 TTL，满足活动期的读压力。  
这样我们能把首屏命中拉到 90%+，P95 在 200–300ms，浏览可用性 99.9%。”



**1) 数据库设计｜Database Design**

**Q1. 如何存储分类（自关联树）？**

**EN:**  
We model categories with a **self-join** table: category(id, name, slug, parent\_id, is\_active, …).

* parent\_id points to category.id; NULL means root.
* Index on parent\_id for fast **direct-children** lookups.
* Soft-delete via is\_active to avoid breaking foreign keys.
* If we ever need full ancestry/descendants, we can add a **closure table** or **materialized path** later (not required for this prompt, which only needs direct children).

**ZH:**  
分类用**自关联**表：category(id, name, slug, parent\_id, is\_active, …)。

* parent\_id 指向 category.id，NULL 表示顶层。
* 对 parent\_id 建索引，便于**直属子类**查询。
* 用 is\_active 做软删，避免外键连锁影响。
* 若后续需要整族谱（祖先/后代），再扩展 **closure table** 或 **materialized path**（本题只要直属子类，不必现在做）。

**Q2. 如何存储商品与分类的多对多关系？**

**EN:**  
Use a junction table product\_category(product\_id, category\_id, created\_at) with a **composite PK** (product\_id, category\_id).

* FK to product.id and category.id with ON DELETE CASCADE.
* Composite indexes: (category\_id, product\_id) for “products in a category”, (product\_id, category\_id) for “categories of a product”.

**ZH:**  
用中间表 product\_category(product\_id, category\_id, created\_at)，主键是 (product\_id, category\_id)。

* 外键指向 product.id 与 category.id，ON DELETE CASCADE。
* 复合索引：(category\_id, product\_id) 支持“按类取商品”，(product\_id, category\_id) 支持“看商品所在类”。

**2) API 设计｜API Design**

**Q3. 请求与返回格式如何设计？**

**EN:**

* **List root categories**: GET /v1/categories?parent\_id=null&locale=…&currency=…
* **Get a category node** (direct children + products page 1):  
  GET /v1/categories/{id}?limit=24&page=1&sort=popularity\_desc&locale=…  
  Response bundles: category, subcategories, products, paging, meta.
* **List products**: GET /v1/categories/{id}/products?limit=&page=&sort=&filters=
* Edge-cacheable JSON with Cache-Control (max-age + stale-while-revalidate).
* **Error**: structured { "error": { "code", "message" } }.

**ZH:**

* **顶层分类**：GET /v1/categories?parent\_id=null&locale=…&currency=…
* **分类节点**（直属子类 + 商品第一页）：  
  GET /v1/categories/{id}?limit=24&page=1&sort=popularity\_desc&locale=…  
  返回统一打包：category、subcategories、products、paging、meta。
* **分类商品列表**：GET /v1/categories/{id}/products?limit=&page=&sort=&filters=
* JSON 可被边缘缓存：使用 Cache-Control（max-age + SWR）。
* **错误**统一格式：{ "error": { "code", "message" } }。

**Q4. 如何让前端方便地 fetch 分类与商品？**

**EN:**  
Keep **one-click = one API**:

* Landing: call /categories?parent\_id=null (roots).
* Drill-down: call /categories/{id} to get **direct** subcategories + products page 1, then infinite-scroll /categories/{id}/products?page=….
* Predictable paging (page+limit or **cursor**) and a small set of whitelisted sort values.
* Add locale/currency on every call; BFF composes final price & i18n.

**ZH:**  
坚持“**一次点击 = 一次接口**”：

* 首屏：/categories?parent\_id=null（拿根节点）。
* 下钻：/categories/{id}（**直属**子类 + 商品第一页），滚动再调 /categories/{id}/products?page=…。
* 分页可预期（page+limit 或 **cursor**），sort 白名单。
* 每次携带 locale/currency，由 BFF 统一计算价格与多语言。

**3) 数据访问模式｜Data Access Patterns**

**Q5. 如何查询顶层分类？**

**EN:**  
SELECT … FROM category WHERE parent\_id IS NULL AND is\_active = TRUE ORDER BY display\_order, id LIMIT …;

**ZH:**  
SELECT … FROM category WHERE parent\_id IS NULL AND is\_active = TRUE ORDER BY display\_order, id LIMIT …;

**Q6. 如何查询某分类的直属子类和商品？**

**EN:**

* **Children:** SELECT … FROM category WHERE parent\_id = :id AND is\_active = TRUE ORDER BY display\_order, id;
* **Products (direct):** join product\_category + product, filter active, order by popularity/price, and paginate.

**ZH:**

* **子类：** SELECT … FROM category WHERE parent\_id = :id AND is\_active = TRUE ORDER BY display\_order, id;
* **商品（直属）：** product\_category 连接 product，只取 is\_active，按 popularity/price 排序并分页。

**Q7. 商品属于多个分类怎么处理？**

**EN:**  
The junction table naturally supports it. When listing a category, we only join by that **category\_id**, so a product may appear in multiple categories—no duplication issues at storage level; dedupe is only needed if the UI mixes multiple categories in one view.

**ZH:**  
中间表天生支持多对多。按 category\_id 查询时，只会返回该类下的商品；同一商品可出现在多个类页面。只有当前端把多个类混合展示时，才需要在 UI 层去重。

**4) 系统扩展性 / 高可用（Follow-up）｜Scalability & HA**

**Q8. 如何缓存分类与商品？**

**EN:**

* **Redis keys:**  
  cat:{id}:meta, cat:{id}:children, cat:{id}:products:{page}:{sort}:{filters\_hash}:{locale}:{currency};  
  landing:hot:{locale}:{currency}.
* **TTL:** products 5–15 min; children/meta 15–30 min; landing 10–20 min.
* **Invalidation:** precise delete on admin writes (category/product changed); publish MQ event to refresh search index and optionally **warm** hot slices.
* **Edge/CDN:** cache popular nodes with SWR.

**ZH:**

* **缓存键：**  
  cat:{id}:meta、cat:{id}:children、cat:{id}:products:{page}:{sort}:{filters\_hash}:{locale}:{currency}；  
  landing:hot:{locale}:{currency}。
* **TTL：** 商品 5–15 分钟；子类/元信息 15–30 分钟；首页 10–20 分钟。
* **失效：** 管理端写入后**精准删除**相关键；发布 MQ 事件重建搜索索引，并可**预热**热门切片。
* **边缘缓存：** 用 SWR 缓存高访问节点。

**Q9. 如何扩展到大访问量？**

**EN:**

* **Horizontal scale** stateless BFF/Query/Admin behind a load balancer.
* **DB**: primary + multiple **read replicas** (geo-distributed); read traffic routes to nearest replica.
* **Search**: multi-node cluster (sharding + replicas).
* **Cache**: Redis cluster with partitioning and persistence (AOF/RDB).
* **CDN**: global POPs for static and selected JSON.

**ZH:**

* **横向扩展**无状态 BFF/Query/Admin，前置负载均衡。
* **数据库**：主库 + 多只读副本（可多地域），读请求就近路由。
* **搜索**：多节点集群（分片 + 副本）。
* **缓存**：Redis 分片集群并开启持久化。
* **CDN**：全球节点缓存静态与部分 JSON。

**Q10. 如何保证系统稳定运行？**

**EN:**

* **Graceful degradation**: if search is down, serve last cached list and disable facets; if cache is down, reduce page size and read replicas.
* **Rate limiting & circuit breakers** on BFF.
* **Observability**: metrics (cache hit, P50/P95 latency, error rate), tracing (edge→BFF→cache/DB/search), structured logs (category\_id, page, locale).
* **SLA-aware** bulkheads: separate pools for search/DB to avoid cascading failures.
* **Backfills** and **warmers** for hot slices.

**ZH:**

* **降级策略**：搜索异常时关闭筛选并回退缓存；缓存异常时减小分页并走只读副本。
* **限流与熔断**：BFF 层实施。
* **可观测性**：指标（缓存命中、P50/P95、错误率）、全链路追踪、结构化日志（category\_id、页码、语言）。
* **舱壁隔离**：DB/搜索使用独立连接池，避免级联故障。
* **预热与回填**：对热门切片定期预热。

So for this catalog system, my approach is to keep things simple and scalable. The categories are stored in a self-join table, so each category just points to its parent, and that makes it easy to fetch direct children with a single query. Products live in their own table, and because a product can belong to multiple categories, I connect them through a junction table. On the API side, I designed it so that one click on the UI always maps to one API call. For example, the landing page fetches top-level categories and a few featured products, and when the user clicks on a category, the API returns both its direct subcategories and the first page of products. That makes the front-end fetch pattern really predictable and easy to work with. Under the hood, we rely heavily on caching—Redis for category nodes and product lists, and CDN edge caching for popular JSON responses. Reads are routed to replicas, writes go through the admin service into the primary database, and whenever there’s a change, we invalidate the relevant cache and update the search index asynchronously. This way, the system can handle high read traffic during promotions while still keeping data reasonably fresh and consistent for the users.