Step 1-4: Order Dataset Analysis

Dataset: `orders(order_id, customer_id, order_date, amount)`

Candidate Shard Key Evaluation

Shard Key	Pros	Cons	Use Case Fit
order_id	High cardinality; perfectly even distribution if using UUIDs.	Sequential IDs create a **hot shard** on the most recent server. Queries by customer_id must hit all shards (scatter-gather).	Poor for this workload (customer-centric).
customer_id	**Co-locates** all orders for one customer on one shard. Fast single-shard reads.	Risk of **data skew** if one "Power User" has millions of orders.	**Best Fit** for OLTP/Customer portals.
order_date	Excellent for time-range analytics and archiving old data.	Massive **hot shard** problem: 100% of today's writes go to the "Today" shard.	Best for logs/analytics only.

Step 5: Other Dataset Evaluations

1. Application Logs

- **Best Shard Key:** `timestamp` (Range-based).
- **Reasoning:** Logs are rarely updated; we mostly care about "what happened in the last hour." Range sharding allows us to drop or archive an entire "Month" shard instantly.

2. User Profiles

- **Best Shard Key:** `user_id` (Hash-based).
- **Reasoning:** Even distribution is vital here. We want to ensure that as we grow to 100 million users, the load is spread perfectly across all nodes.

3. IoT Sensor Data

- **Best Shard Key:** `device_id` + `timestamp` (Composite).
- **Reasoning:** We usually want to see a specific device's history. Sharding by `device_id` ensures one device's data is together, but adding `timestamp` allows us to sub-partition by time for better performance.

Step 6: Reflection Questions

1. **Why is the choice irreversible?** To change a shard key, you must create a new cluster, define the new key, and move **all** data from the old cluster to the new one while the system is live. This causes massive "rebalancing" traffic and potential downtime.
2. **Analytics vs. OLTP?** OLTP wants to find **one user's** data instantly (Single Shard). Analytics wants to sum **everyone's** data over a time period (Multiple Shards/Range).
3. **Cross-shard queries?** These require a "Coordinator" node to send the query to every shard, wait for all results, and merge them. It is the "slowest link" problem—if one shard is slow, the whole query is slow.

Step 7: E-commerce Product Catalog Design

Product Catalog Sharding Design

Chosen Shard Key
category_id (Hash-based)

Reasoning
Most shoppers browse by category (e.g., "Electronics"). By sharding on `category_id`,
all products in that category live on the same shard, making "Browse" queries extremely fast.

Query Routing
- **By Category:** Routed to a single shard.
- **By Product ID:** Requires a Global Index or a scatter-gather query across shards.
- **Search:** Handled by a separate search engine (like Elasticsearch), not the sharded DB.

Edge Cases
- **Skew:** If "Clothing" has 80% of products, we can use a composite key `category_id + subcategory_id` to break the hot shard into smaller pieces.

Step 8: Comparison Matrix

Strategy	Distribution	Query Routing	Complexity	Best For
Hash-based	**Excellent**	Single shard for Key	Low	OLTP, User Profiles
Range-based	Can be uneven	**Best for ranges**	Medium	Time-series, Logs
Composite	Balanced	Flexible	**High**	IoT, Complex SaaS
Directory	**Perfectly Flexible**	Dynamic	Very High	Multi-tenant apps