































































### **❌ Option 1**

**“Reduce the number of partition keys in the DynamoDB table.”**

This option is **conceptually broken**.

Why?

* You don’t “set” the number of partition keys
* Partition keys are **values**, not configuration knobs
* Fewer partition key values = **MORE hotspot risk**

Reducing partition keys makes performance **worse**, not better.

👉 Instant elimination.

### **❌ Option 2**

**“Use partition keys with low-cardinality attributes.”**

Low cardinality = **few distinct values**

Examples:

* country (India, US, UK)
* status (active/inactive)
* category

Why this is BAD:

* Many requests hit the **same partition**
* One partition gets overloaded
* Others sit idle

This causes:

* Hot partitions
* Throttling
* Wasted provisioned throughput

This is the **#1 DynamoDB anti-pattern**.

### **✅ Option 3 (CORRECT)**

**“Use partition keys with high-cardinality attributes.”**

High cardinality = **many unique values**

Examples:

* userId
* orderId
* sessionId
* UUID

Why this is PERFECT:

* Requests spread across many partitions
* Load is evenly distributed
* Full use of provisioned throughput
* Linear scalability

This directly matches:

“distributing the workload evenly and utilizing the provisioned throughput efficiently”

This is textbook DynamoDB design.

### **❌ Option 4**

**“Avoid using a composite primary key.”**

Composite key = **partition key + sort key**

This option is misleading.

Truth:

* Composite keys are **GOOD**, not bad
* They help model 1-to-many relationships
* They enable efficient queries

Avoiding composite keys does **nothing** to fix workload distribution.

Performance problem ≠ key structure  
 Performance problem = **partition key choice**

So this option is irrelevant → eliminate.







































