



DP-420 Domain 1a: Model Documents in Cosmos DB

Microsoft Learn DP-420 skills outline and Azure Cosmos DB for NoSQL modelling documentation, as of 2026 | Embed vs reference, denormalisation, mixed entity types in one container, item size and ids, system properties, schema versioning, unique keys, TTL, modelling for read and write patterns

DP-420 Data models Embedding Denormalization

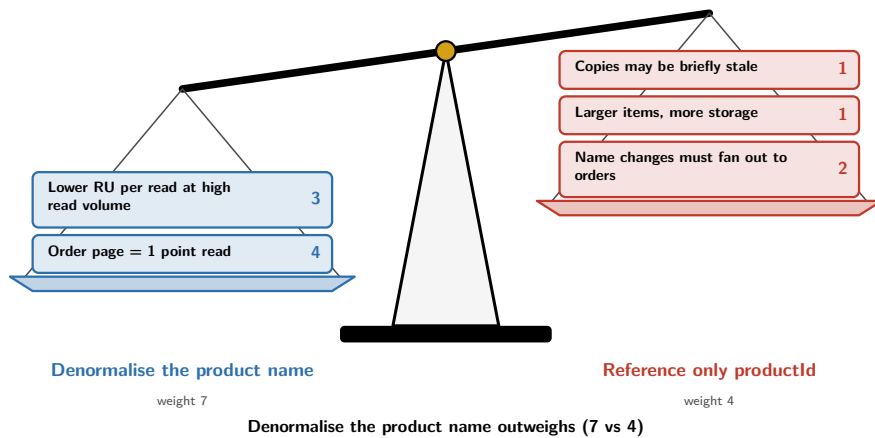
Embed or reference

	Recommended		
	Embed	Reference	Hybrid (reference + denormalised fields)
Relationship	1:1 or 1:few	1:many or many:many	1:many
Read together?	almost always	sometimes	summary often, detail rarely
Growth	bounded	unbounded	bounded summary
Update frequency of child	rare	frequent, independent	summary fields change rarely
Atomic update needed	yes, one item	✗	eventual via change feed
Choose	order lines in order	reviews of a product	product name copied into order line

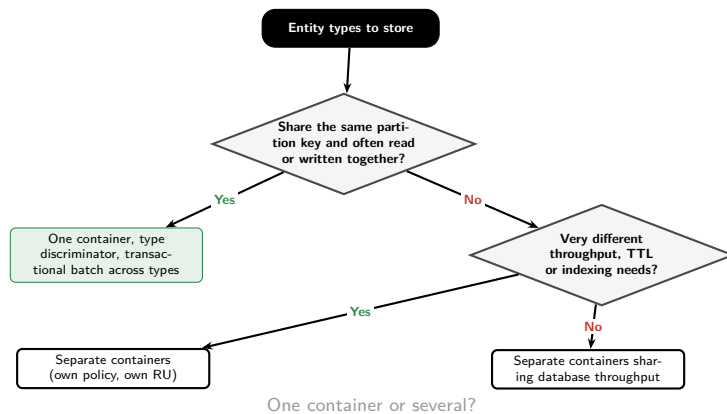
Order document: embedded lines, referenced customer and product, denormalised names

```
{
  "id": "ord-1042",
  "customerId": "c-77",
  "type": "order",
  "schemaVersion": 2,
  "status": "shipped",
  "customer": {
    "name": "Ada Martin",
    "tier": "gold"
  },
  "lines": [
    {
      "productId": "p-9",
      "name": "Trail shoe",
      "qty": 1,
      "price": 120
    },
    {
      "productId": "p-31",
      "name": "Socks",
      "qty": 3,
      "price": 9
    }
  ],
  "total": 147,
  "ttl": -1
}
```

- partition key: all of a customer's orders together
- discriminator: customer, order and cart items share the container
- denormalised summary; refreshed by change feed
- embedded: bounded, read with the order



Container design



Rule or limit	Value	Consequence
Maximum item size	2 MB (NoSQL)	keep items small: RU cost grows with size; split large blobs to Blob Storage
id	string, up to 1,023 bytes, unique per logical partition	avoid / \ ? # characters; id + partition key = point read
Unique key policy	set at creation only, per logical partition	changing it means a new container and data migration
TTL	container defaultTtl (seconds, -1, or off) + item ttl	expired items deleted in background using leftover RUs, no extra charge

System properties	_rid, _self, _etag, _ts, _attachments	_etag for optimistic concurrency, _ts for change ordering
Arrays	bounded only	unbounded arrays → separate items (bucket pattern: one item per device per hour)

- **Model for the access pattern** list the queries and writes with their frequency first; the most frequent read should be a point read or a single-partition query
- **Mixed entity container** customer profile + orders + cart in one container partitioned by customerId: one partition query returns the customer view; transactional batch updates them atomically
- **Bucket pattern** time-series readings grouped into one item per sensor per hour (array of up to N readings) instead of one item per reading: fewer, larger writes
- **Reference data** cache countries, categories and settings in the app (or a small container read at start-up) instead of embedding everywhere
- **Schema evolution** add schemaVersion; readers handle old shapes; migrate with a change feed job or on next write

Worked example

Worked example: embed vs reference at 1,000 order views per second

PROBLEM

The order page is viewed 1,000 times per second. Design A embeds the 5 order lines (item about 3 KB). Design B stores lines as 5 separate items and reads them with a single-partition query. Estimate RU/s (rule-of-thumb costs, measure with the SDK's RequestCharge).

1. Design A per view

⇒ ≈ 2 RU, 1 request

why: id + partition key point read is the cheapest operation

2. Design B per view

⇒ ≈ 1 + 3 = 4 RU, 2 requests

why: queries cost more than point reads and add a round trip

3. Throughput needed

⇒ A: $1,000 \times 2 = 2,000$ RU/s; B: $1,000 \times 4 = 4,000$ RU/s

why: $RU/s = operations/s \times RU \text{ per operation}$

4. Latency

⇒ A: 1 round trip; B: 2 round trips

why: each request is a network call

GIVEN

- v = 1,000 views/s
- A: one point read of about 3 KB ≈ 2 RU (approx.)
- B: point read of the order header ≈ 1 RU + single-partition query returning 5 small items ≈ 3 RU (approx.)

ANSWER

Embedding halves the RU/s (≈ 2,000 vs 4,000) and the round trips for this read-heavy page.

Check: If lines were updated independently and often, or unbounded, the write cost of rewriting a large item would flip the decision.

YOU MAY HEAR

"Normalise like in SQL Server and join at query time"

BUSTED✓ **ACTUALLY**

Cosmos DB has no cross-item joins

JOIN in the query language is a self-join over arrays inside one item; model for single-item or single-partition reads.

YOU MAY HEAR

"One container per entity type is the rule"

BUSTED✓ **ACTUALLY**

Mixing types under one partition key is a recommended pattern

It enables single-partition queries and transactional batch across related items.

YOU MAY HEAR

"Denormalised copies go stale forever"

BUSTED✓ **ACTUALLY**

Change feed keeps them in sync

A change feed processor on the source container updates the copies within seconds.

YOU MAY HEAR

"Big items are fine because storage is cheap"

BUSTED✓ **ACTUALLY**

RU cost grows with item size on reads and writes

And the 2 MB limit is hard; split or reference large parts.

🚩 Recap and self-test

Tick a box only when you could explain the point to someone else

**Start from access patterns**

most frequent read = point read

**Embed bounded, read-together data**

reference unbounded or independent data

**Denormalise hot fields, sync by change feed**

hybrid model

**Mix entity types with a discriminator**

same partition key, transactional batch

**Mind the limits**

2 MB items, unique keys at creation, TTL in seconds

Q1 When should you embed related data?

A 1:1 or 1:few, bounded, read together, rarely updated independently.

Q2 When should you reference instead?

A Unbounded or many-to-many relationships, data updated often and independently, or large child data.

Q3 How do you keep denormalised copies current?

A A change feed processor (or Functions trigger) that updates the copies when the source changes.

Q4 Which property lets several entity types share a container?

A A type discriminator such as type, filtered in queries.

Q5 Can you add a unique key policy to an existing container?

A No: only at creation; migrate to a new container.

Q6 What does defaultTtl = -1 mean?

A TTL is enabled, but items never expire unless they carry their own ttl.

Q7 Which system property supports optimistic concurrency?

A _etag (send it with IfMatchEtag on replace or patch).

Q8 What is the bucket pattern?

A Grouping many small time-series events into one item per entity per period to reduce writes and item count.

Q9 What does JOIN do in a Cosmos DB NoSQL query?

A A self-join with arrays inside the same item, not a join between items.