Module 08: Implement a data modeling and partitioning strategy for Azure Cosmos DB SQL API

Implement a non-relational data model

What's the difference between NoSQL and relational databases

Relational database:

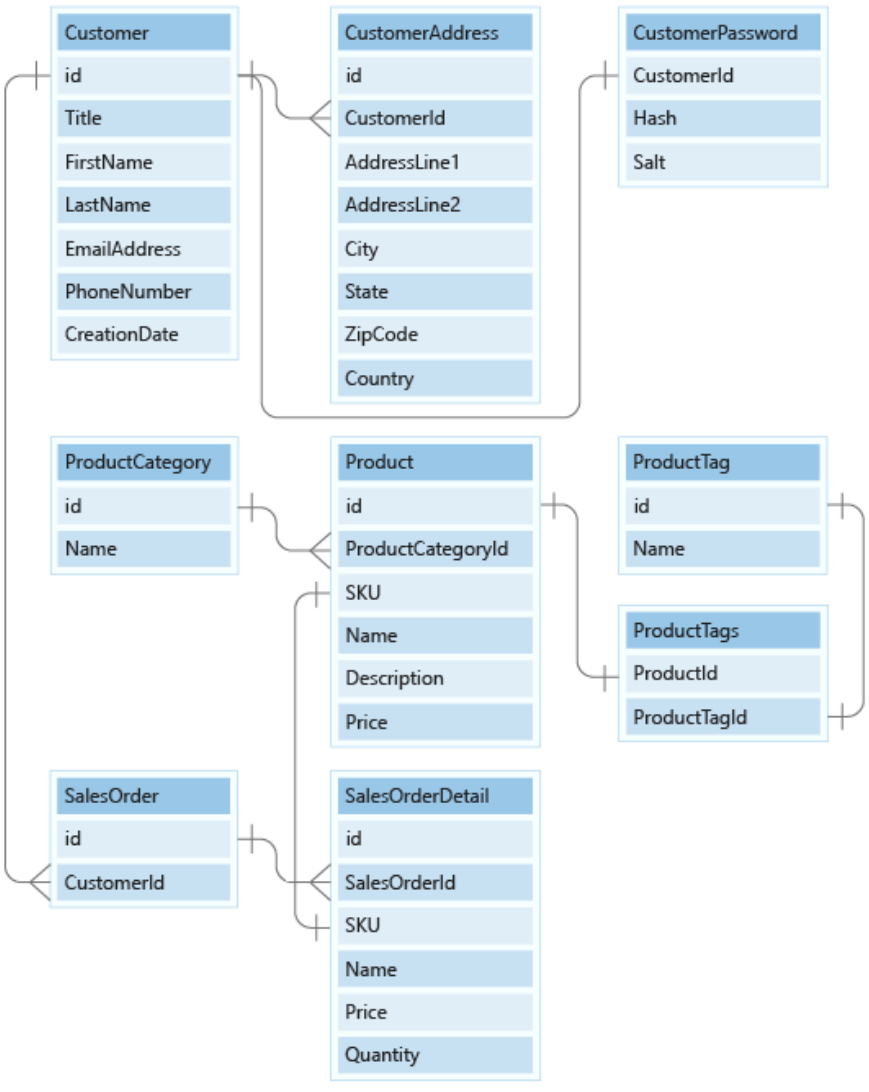
- Vertical scaling only
- Fixed schema
- Relational data

NoSQL database:

- Horizontal scaling
- Flexible schema
- Non-relational data

Scenario

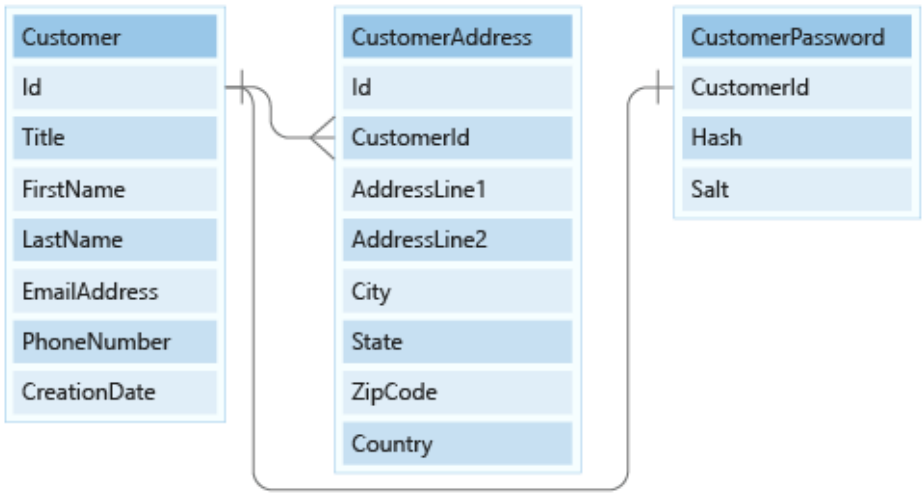
The following entity-relationship (ER) diagram details the nine entities used in the next modules.



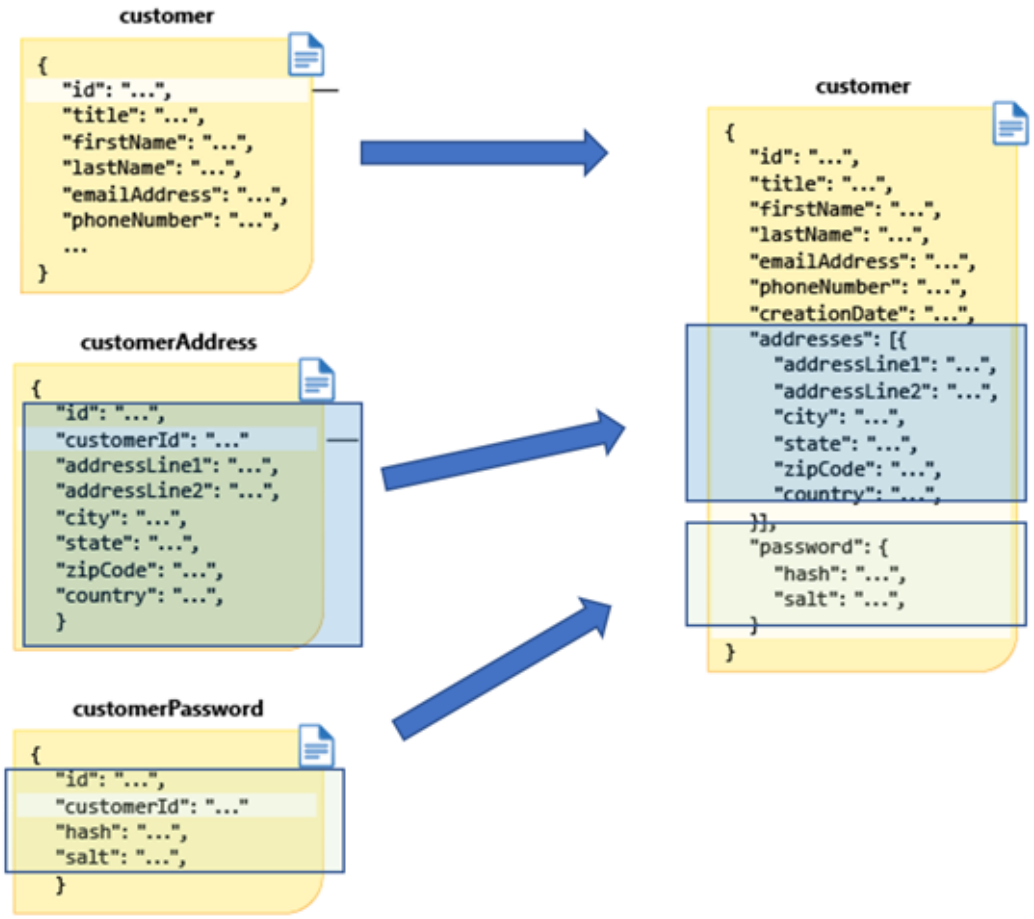
Identify access patterns for your app

When you're designing a data model for a NoSQL database, the objective is to ensure that operations on data are done in the fewest requests.

Identify access patterns for customer entities



Model customer entities



When to embed or reference data

There are rules for when you should embed data in a document instead of referencing it in a different row.

When should you **embed** data?

- Read or updated together

- 1:1 relationship
- 1:Few relationship

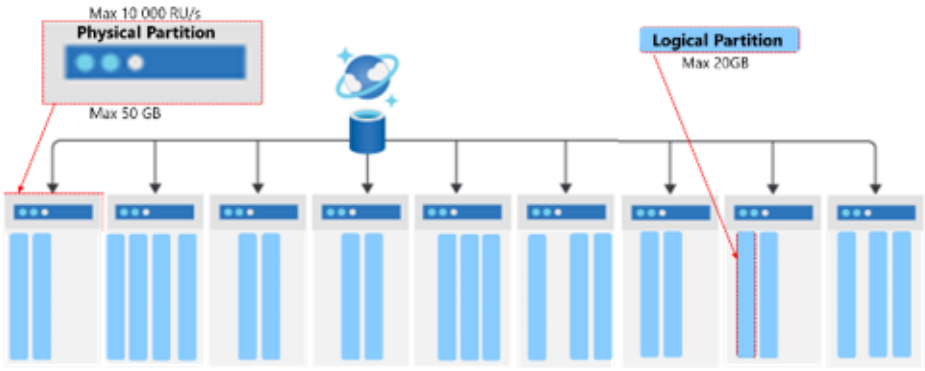
When should you **reference** data?

- Read or updated independently
- 1:Many relationship
- Many:Many relationship

Choose a partition key

The partition key is a required document property that ensures documents with the same partition key value are routed to and stored within a specific physical partition.

Physical and Logical partitions in Azure Cosmos DB



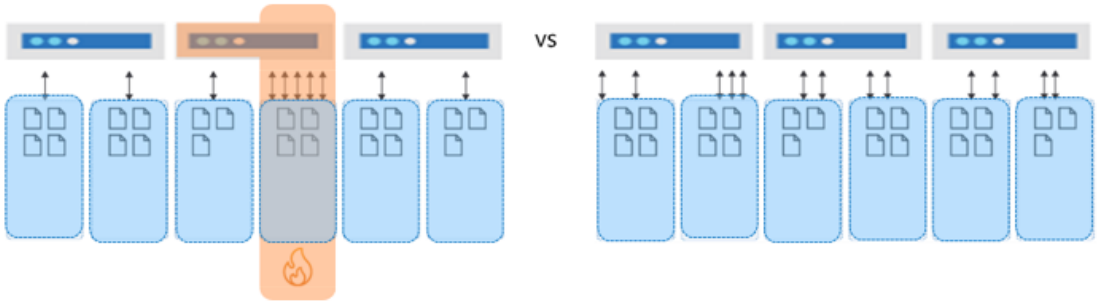
Partitions Key

Avoid hot partitions

When data is not partitioned correctly, it can result in hot partitions. Hot partitions prevent your application workload from scaling.

Storage hot partitions

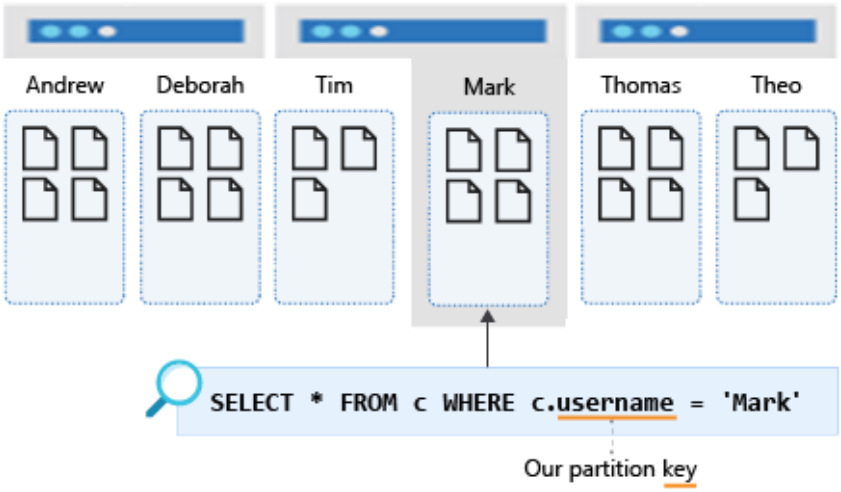
Throughput hot partitions



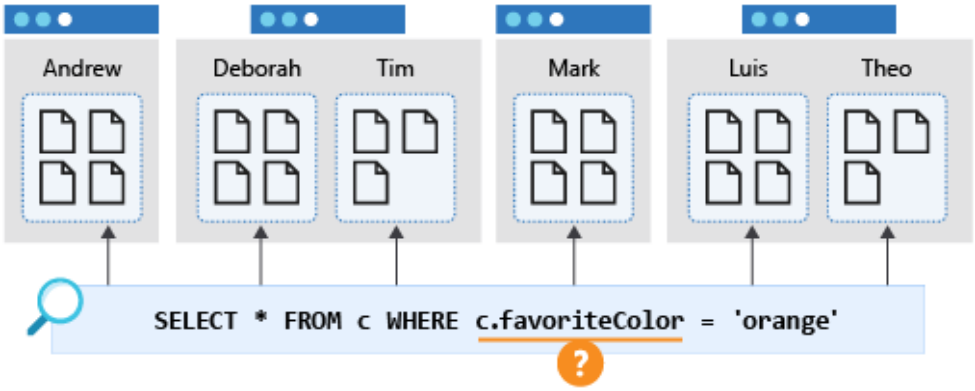
Consider reads versus writes

When you're choosing a partition key, you also need to consider whether the data is read heavy or write heavy.

Single Partition Query

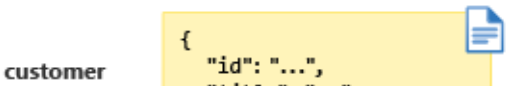


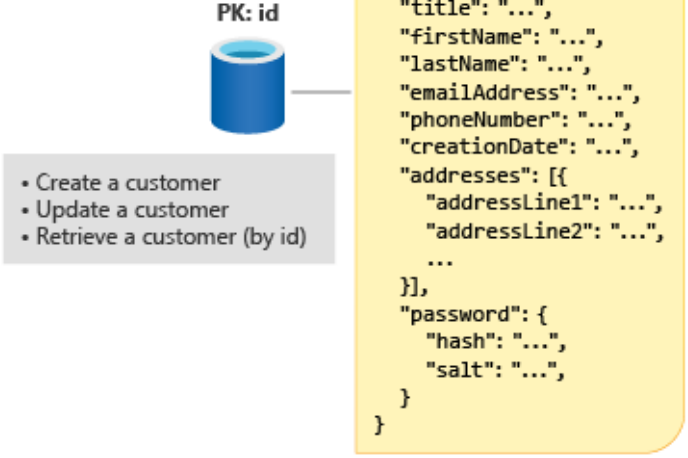
Cross Partition Query



Choose a partition key for customers

Now that you understand partitioning in Azure Cosmos DB, we can decide on a partition key for our customer data.

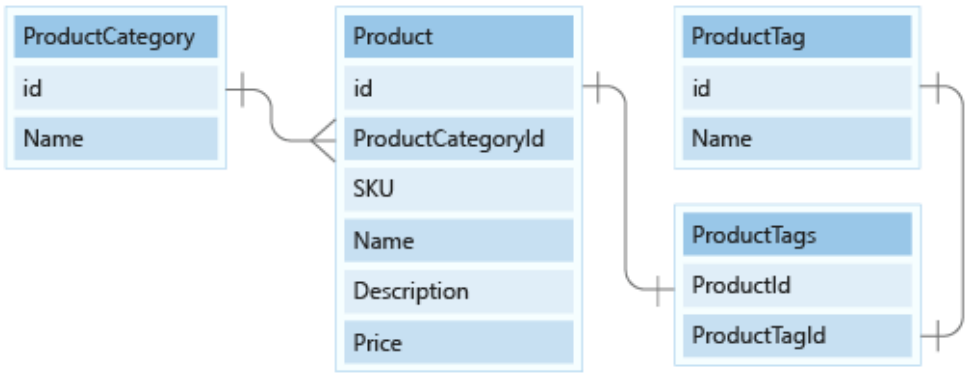




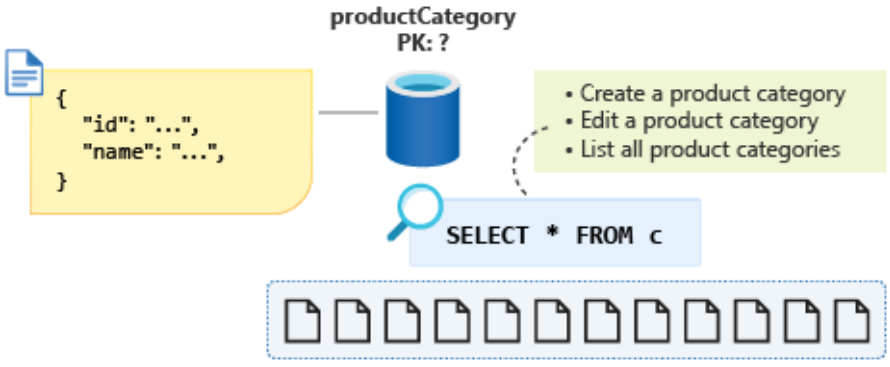
Model small lookup entities

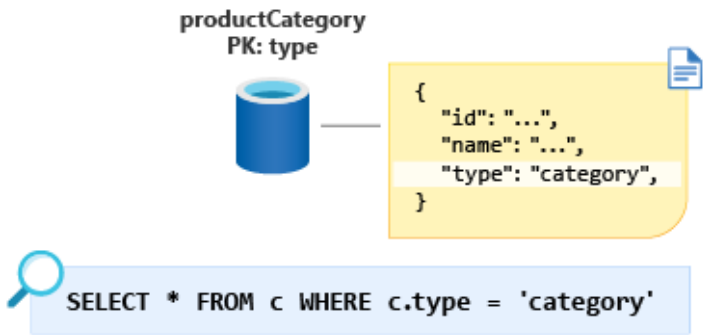
Our data model includes two small reference data entities, ProductCategory and ProductTag.

Product relational model



Model product categories





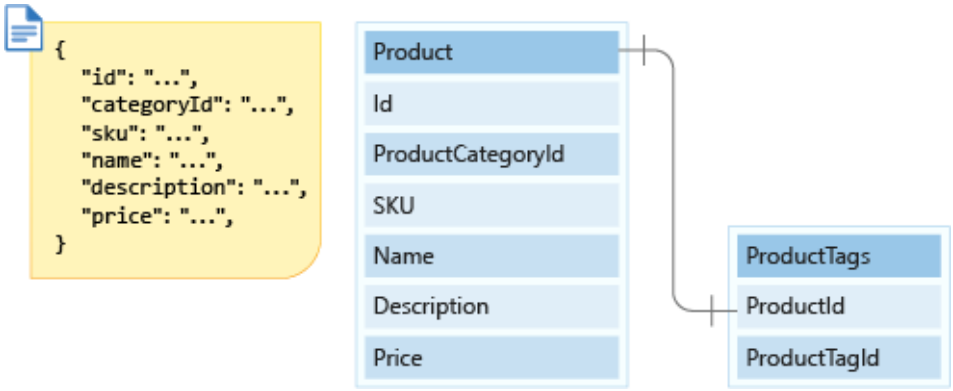
Model product tags

Design a data partitioning strategy

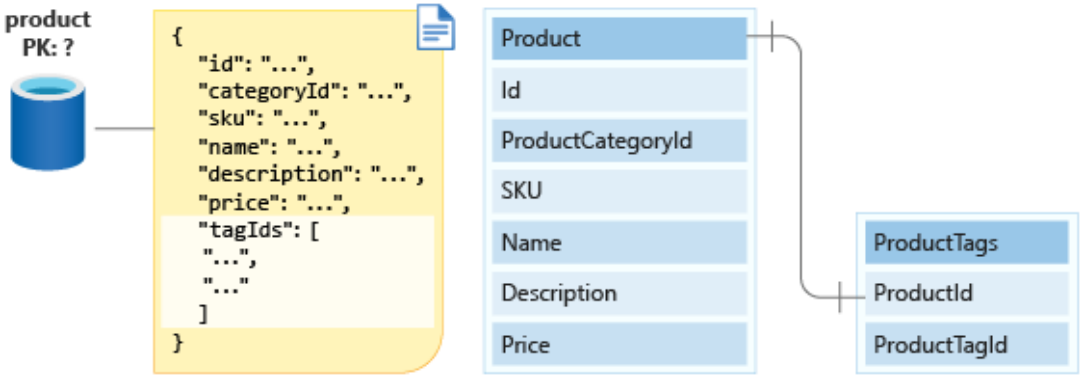
Denormalize data in your model

Let's model a product table from your relational database into a NoSQL database.

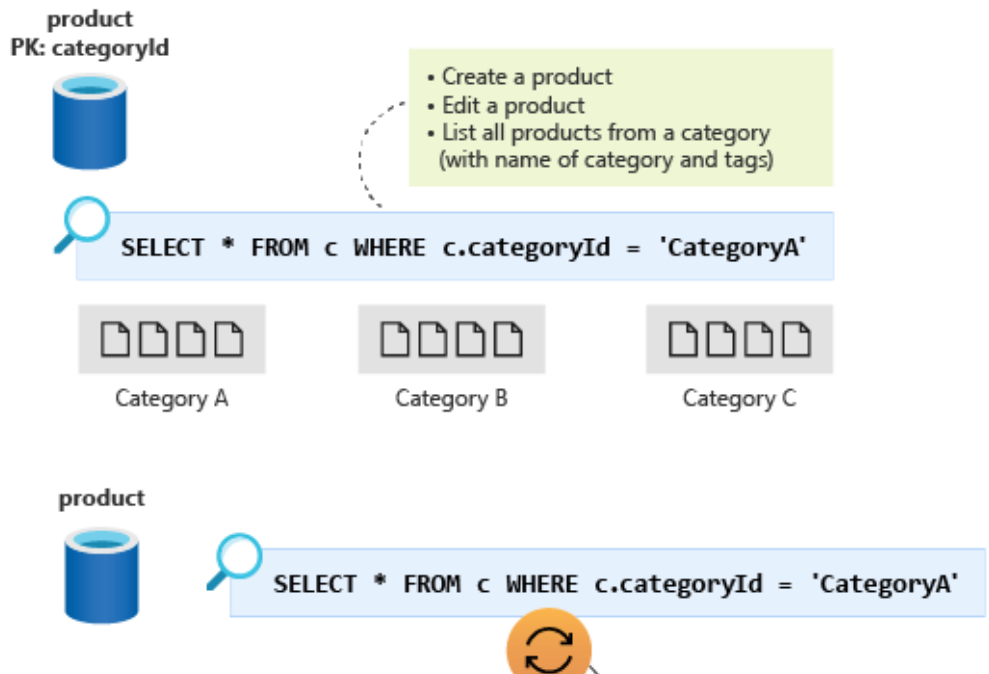
Product/Product Tags relational model

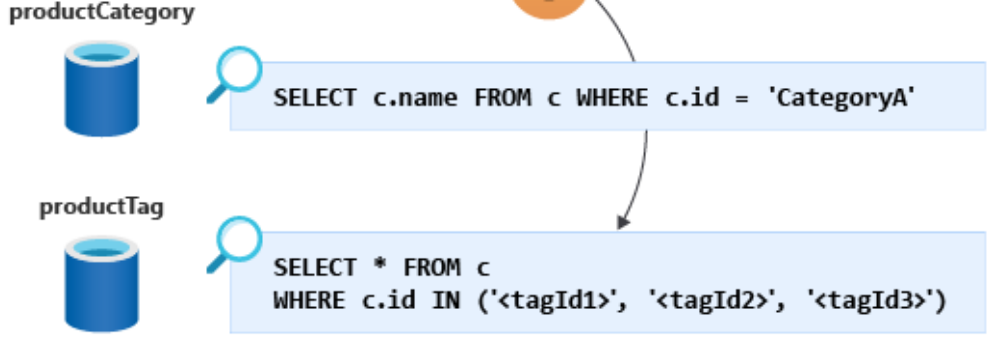


Model the product entities

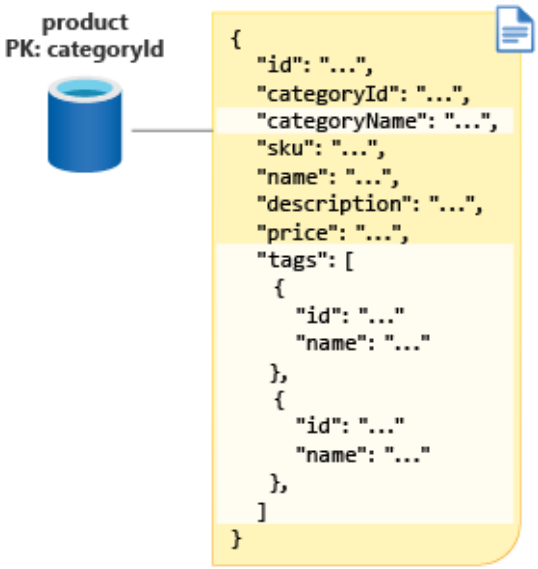


Select a partition key



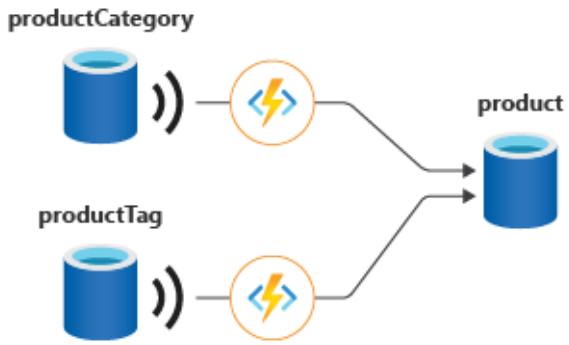


Denormalize product entities



Manage referential integrity by using change feed

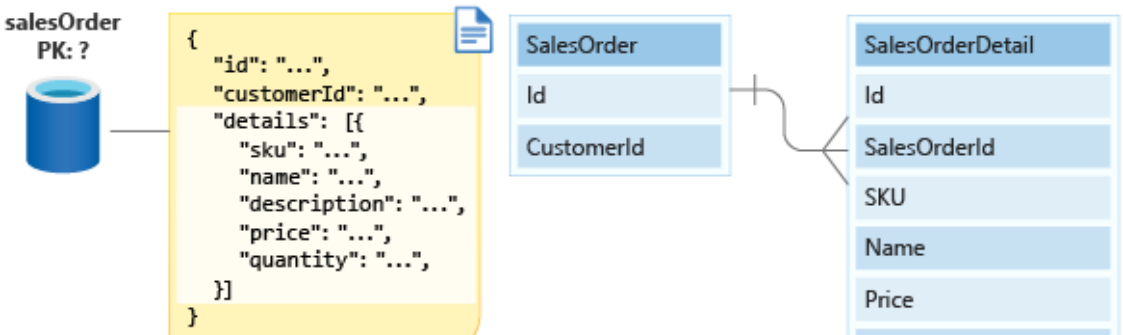
When denormalizing data like shown for the Product, ProductCategory and ProductTag tables, referential integrity must be maintained when changes occur on the categories or tags.

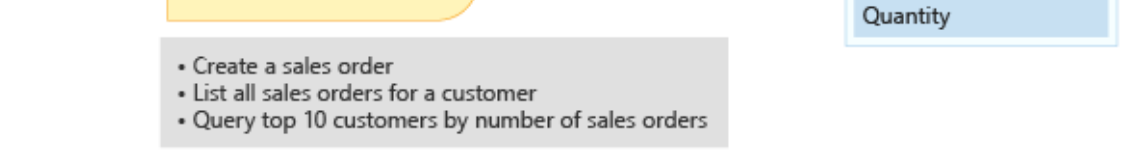


Combine multiple entities in the same container

Review your operations and then decide whether to embed or reference your related data.

Model sales order entities





- Create a sales order
- List all sales orders for a customer
- Query top 10 customers by number of sales orders

Identify optimization opportunities