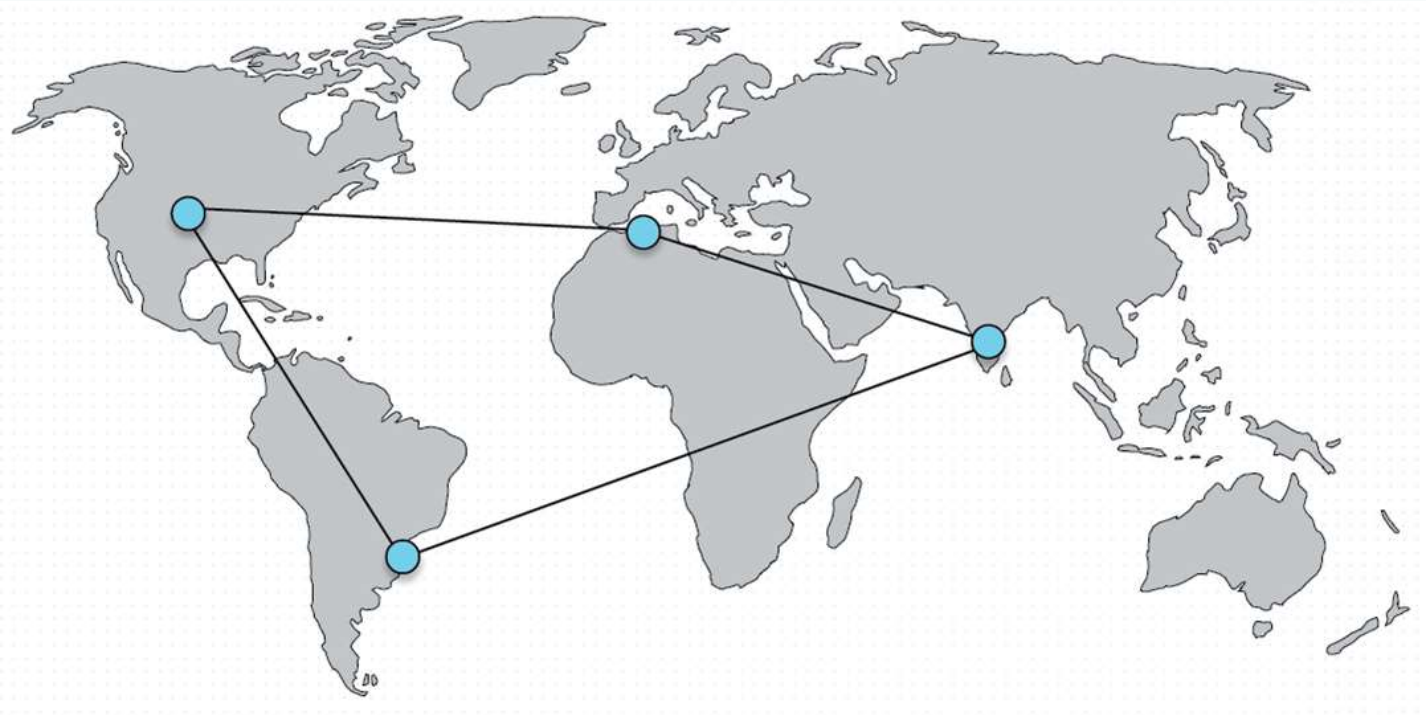
Azure Cosmos DB Concepts

Global Distribution

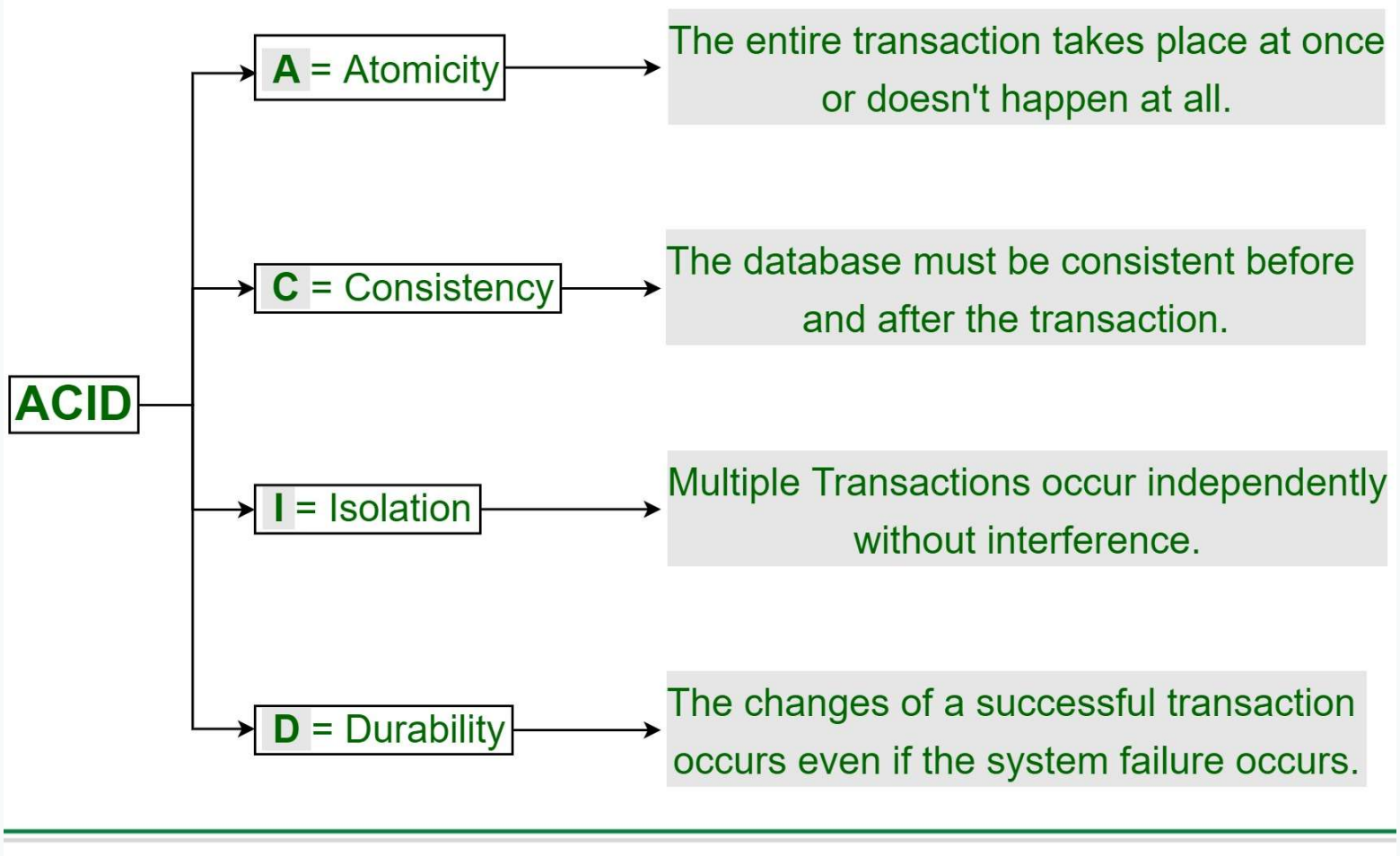
- Microsoft Azure is available globally in over 30 regions
- Global distribution is a comparable concept to what replication is for relational databases
- Don’t need complex configurations



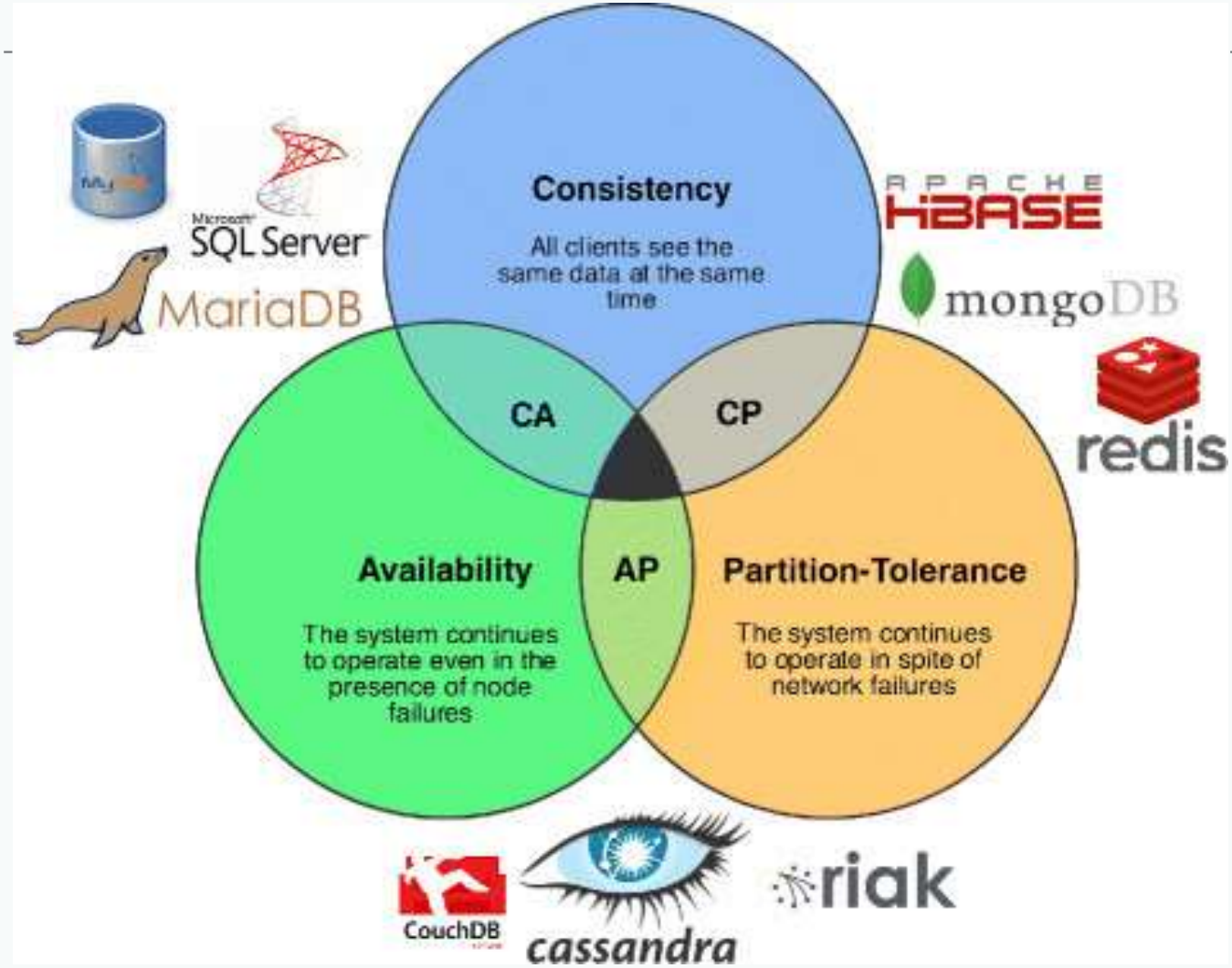
Write and Read Regions

- When the database was first created, it was based on only one region
- This default configuration defines the first (and only) region where a database accepts read and write operations.
- When you distribute the database to more regions, the new regions automatically become read regions

ACID Properties

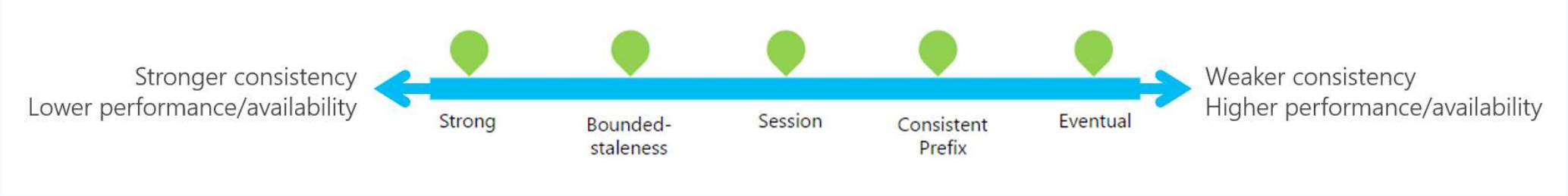


CAP Theoram

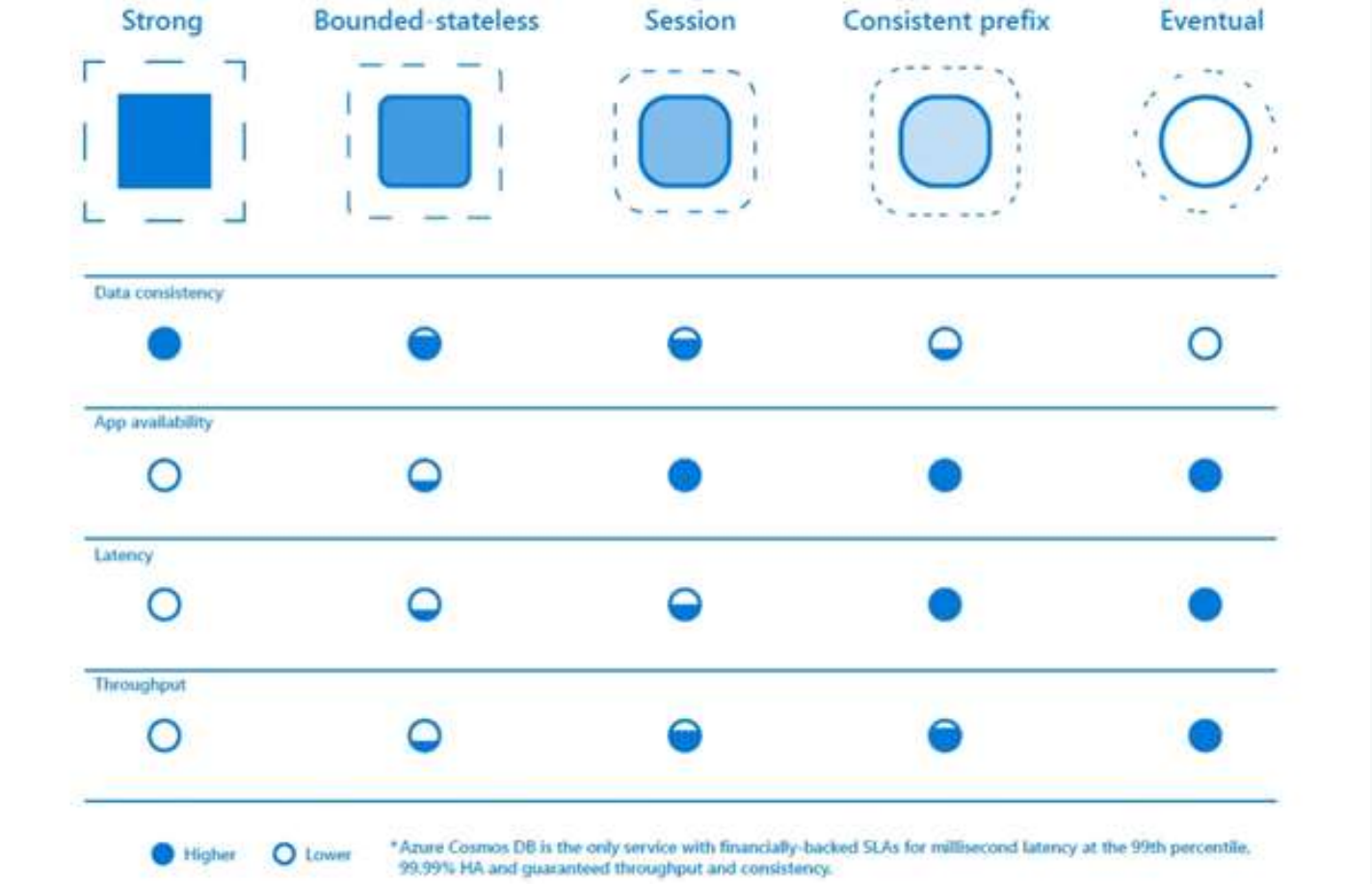

Consistency
All clients see the same data at the same time
Availability
The system continues to operate even in the presence of node failures
Partition-Tolerance
The system continues to operate in spite of network failures
CA
CP
AP
SQL Server
MariaDB
APACHE HBASE
mongoDB
redis
CouchDB
cassandra
riak

Consistency Models

- Consistency defines the rules under which distributed data is available to users.
- Azure Cosmos DB implements five different consistency models



Consistency Models

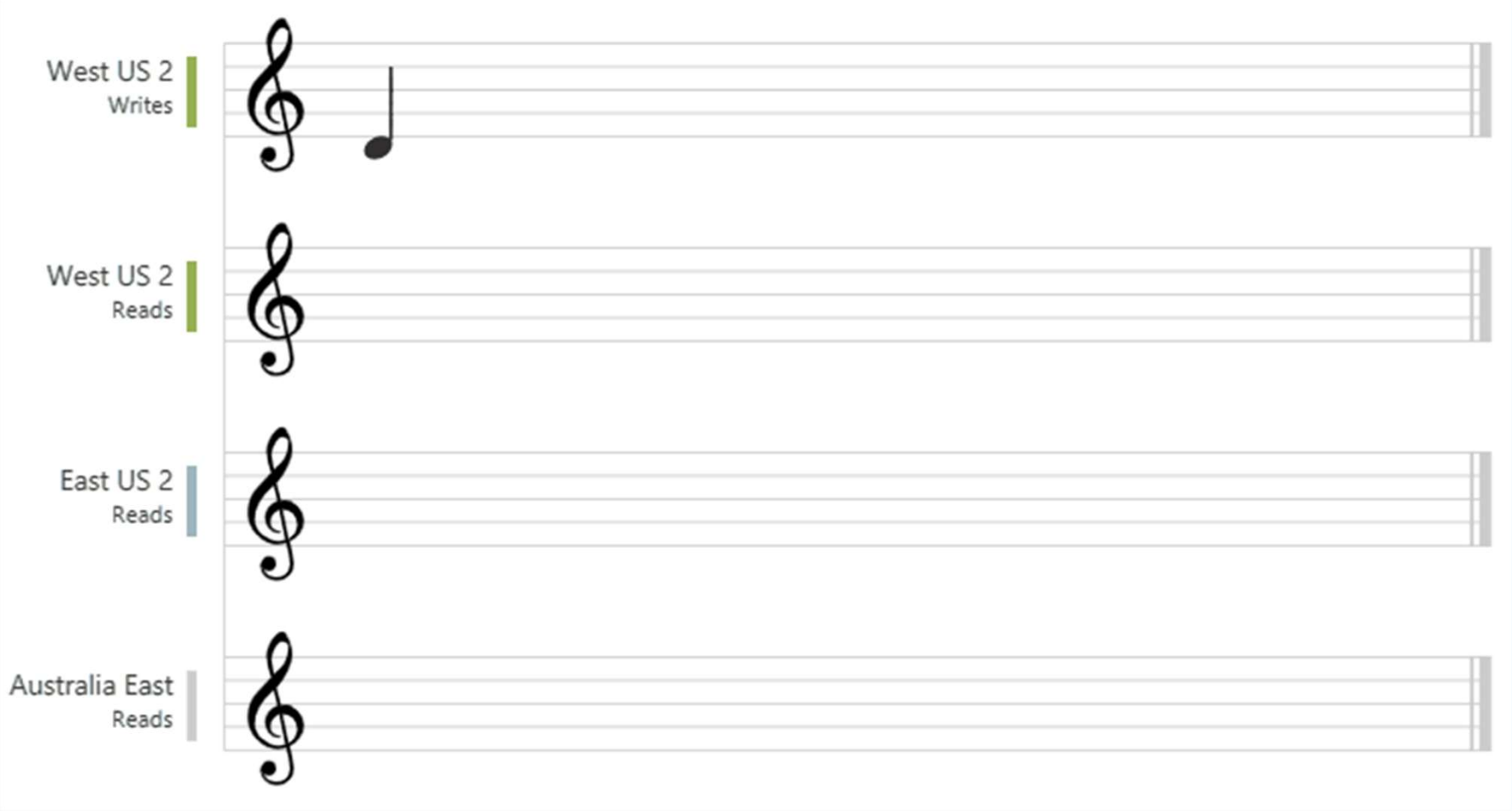


Scope of Consistency

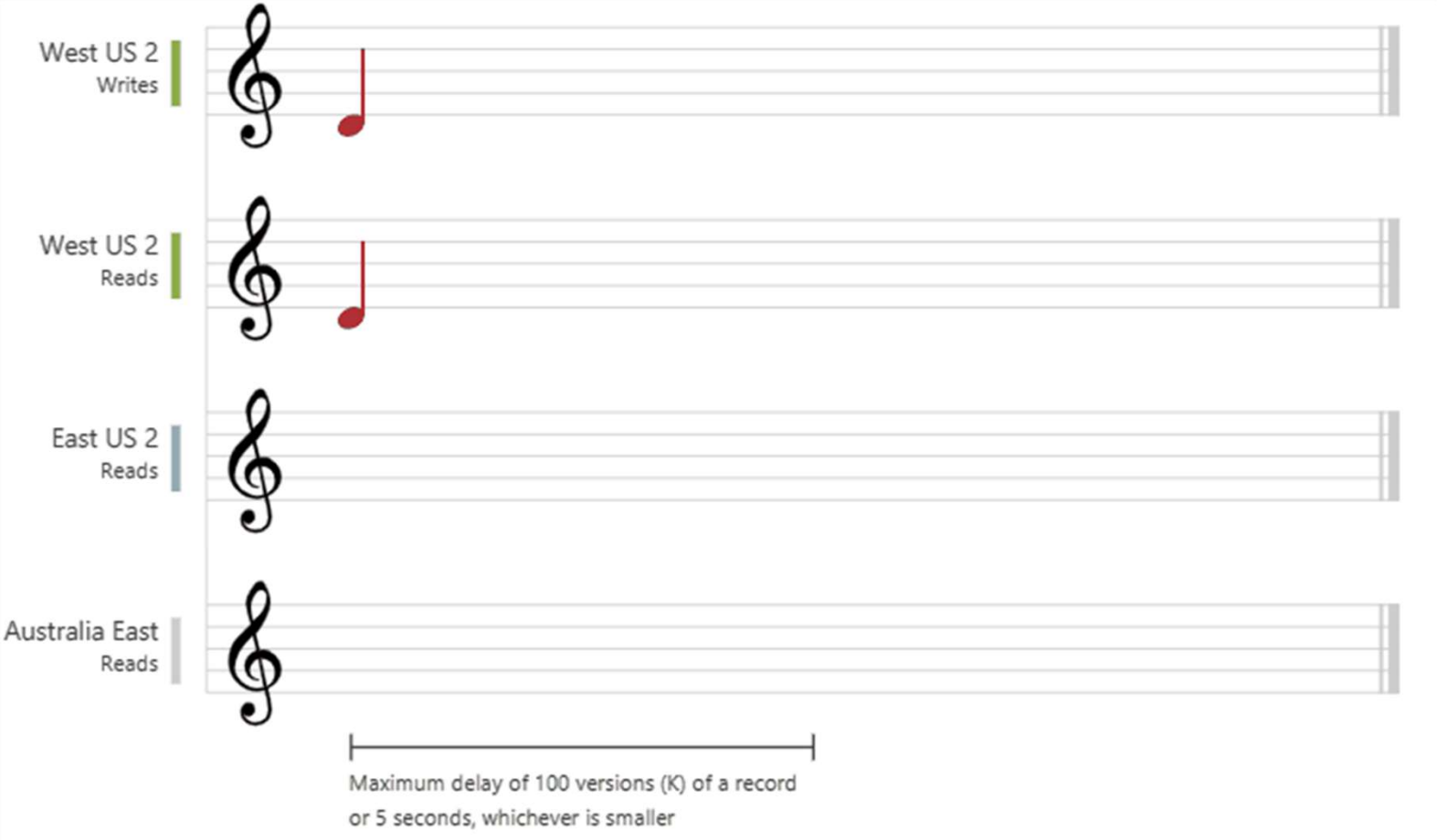
- The granularity of consistency is scoped to a single user request.
- A write request may correspond to an insert, replace, update, or delete Transaction
- As with writes, a read/query transaction is also scoped to a single user request.

Strong Consistency Model

Eventual Consistency Model

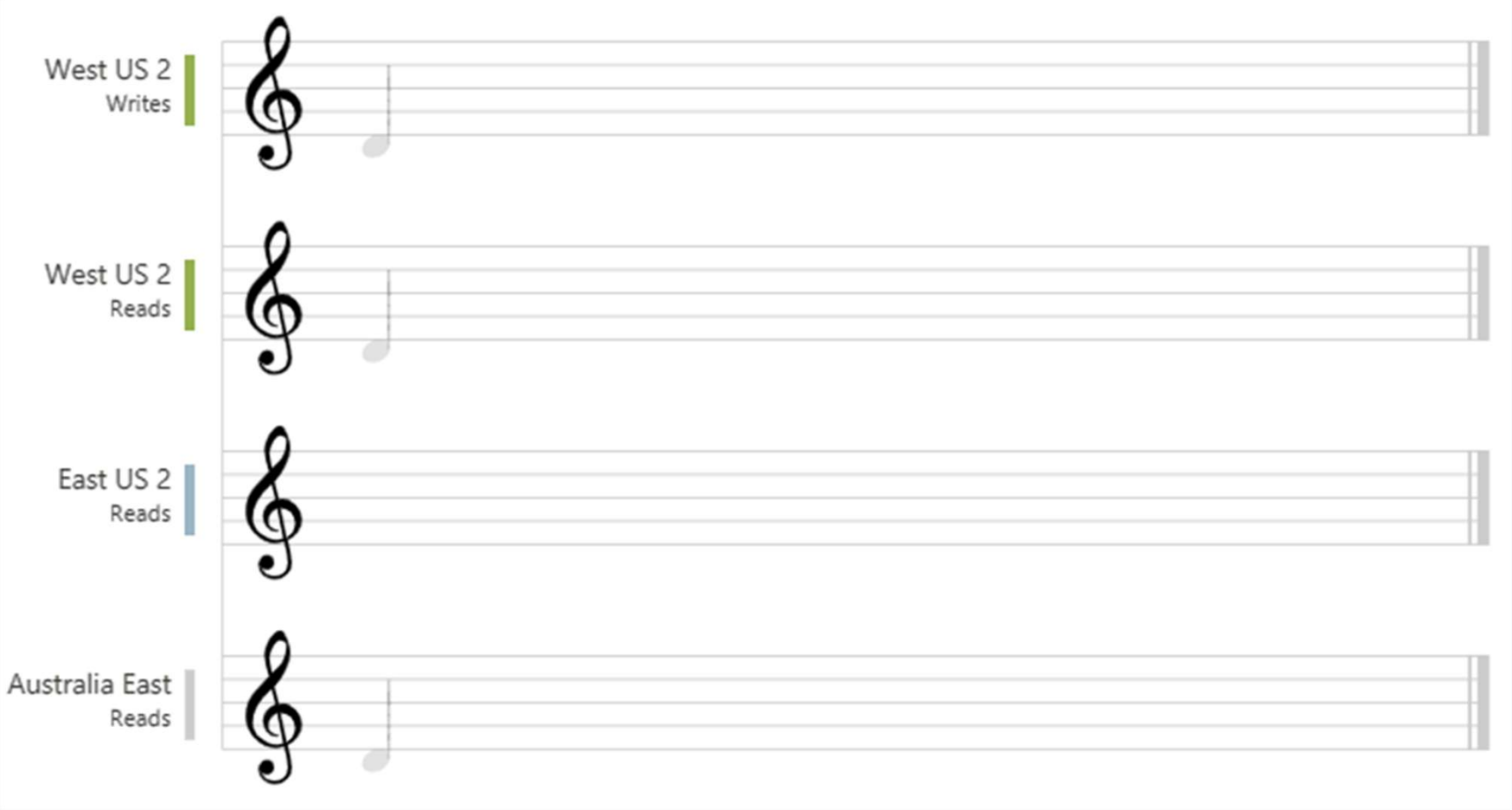


Bounded Staleness Consistency Model



Session Consistency Model

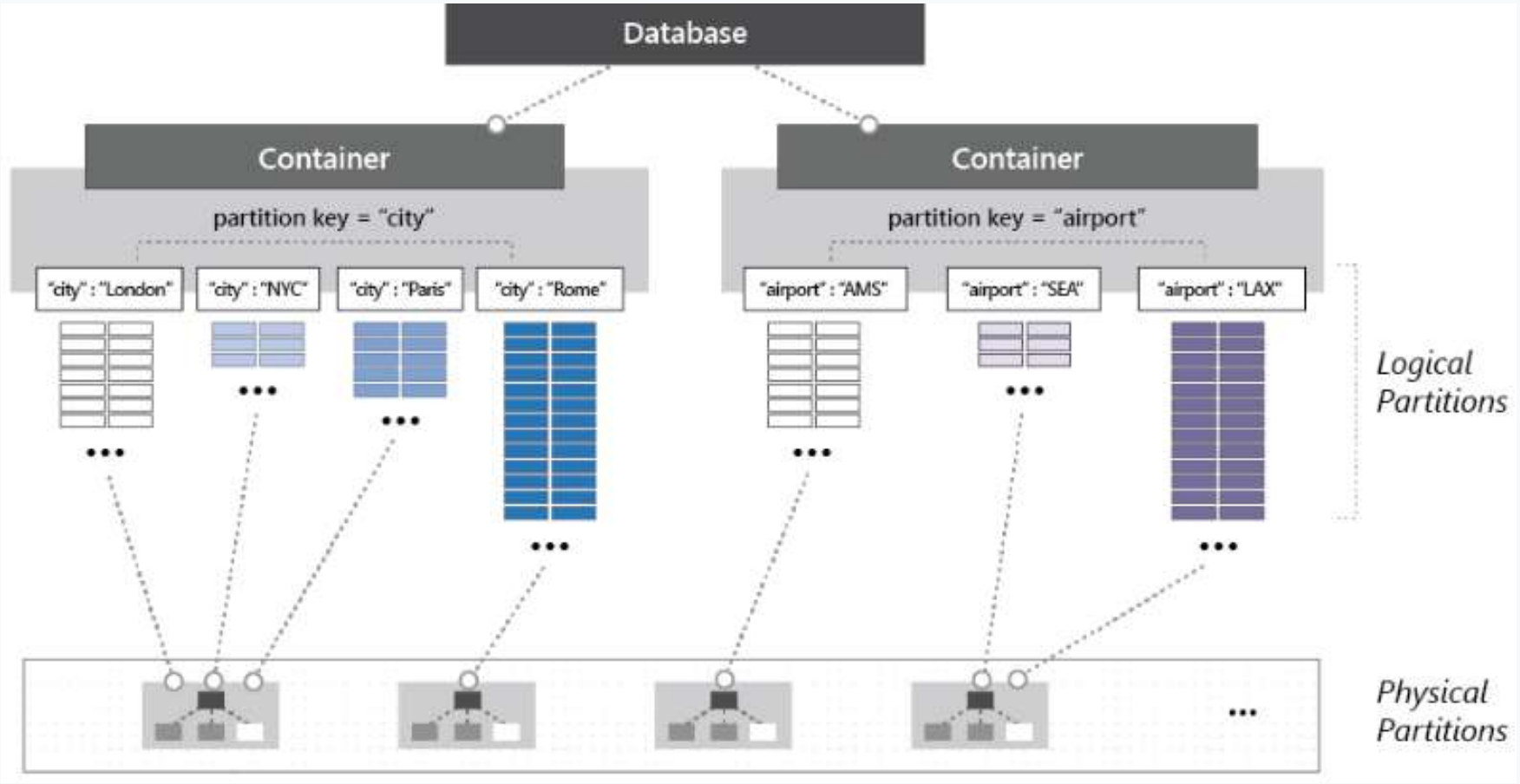
Consistent Prefix Consistency Model



Consistency for Queries

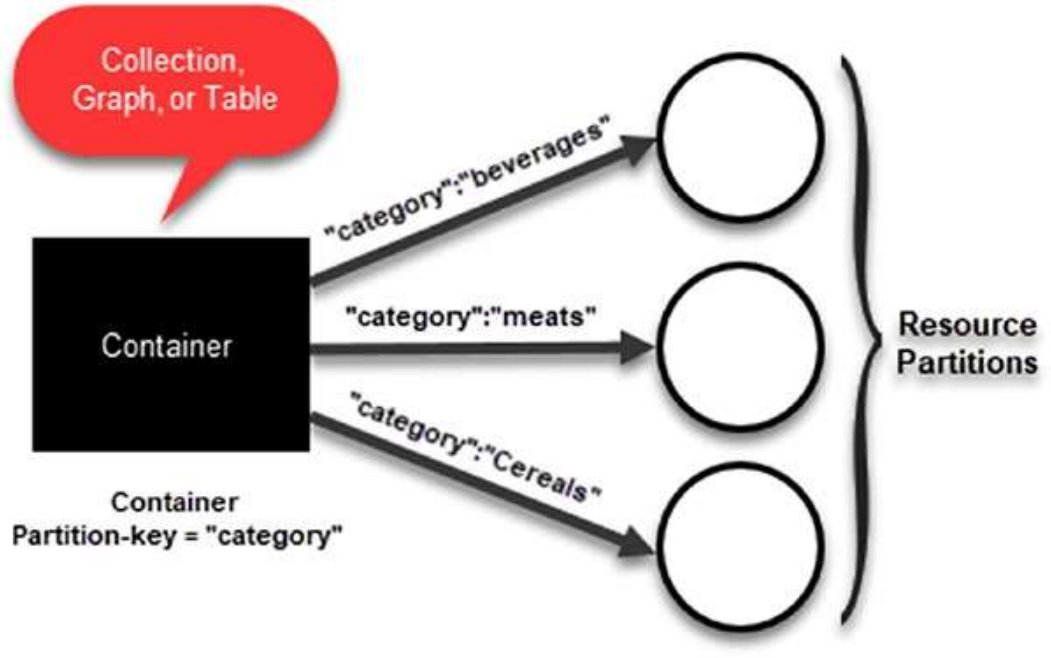
- By default, any user-defined resource would have the same consistency level for queries as was defined for reads
- This is possible because indexes are updated synchronously
- You can also change the index update strategy to be lazy
  - This will boost the performance of writes
- When changing to lazy, regardless of the read consistency level, queries will have a consistency level of eventual

Understanding Partitioning



What Are Containers?

- Containers are logical resources that group together one or more physical partitions.
- Don’t have any restrictions in terms of amount of storage or throughput

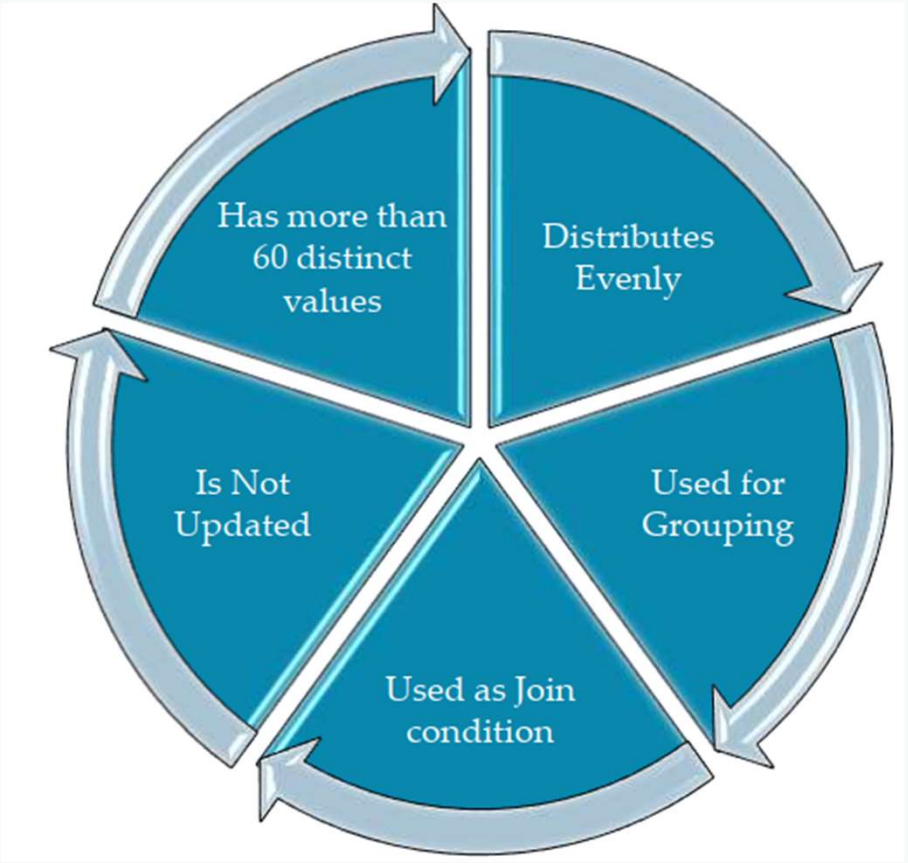


How Does Partitioning Work?

- Need to define a partition key and a row key for each item in your container.
- These key combinations uniquely identify the item
- The partition key determines the logical partition for your data
- Choosing the right partition key is crucial

Designing for Partitioning

- There are two key considerations when choosing the partition key



Understanding Throughput

- Azure Cosmos DB has predictable performance
- This is achieved by provisioning a specific amount of request units (RU) per second
- This amount is what is called throughput.
- This means that you will be billed by how much reserved throughput you have as opposed to how much you actually use

Specifying Request Unit Capacity

- When defining a new collection, you need to configure the specific number of request units per second you want reserved for the container.
- The reserved throughput can be between 400 and 10,000 request units per second.
- Based on this number, Azure Cosmos DB allocates physical partitions to host the collection and it will manage the data across partitions as it grows.

Estimating Throughput

- Refer:
  - https://azure.microsoft.com/en-gb/products/cosmos-db/

Implementing Security

- Encryption at Rest
- Firewall Support
- Securing Access to Data

Supported APIs

- Azure Cosmos DB supports several APIs for resource and data management
- Several software development kits (SDKs)
- At its core is the REST API, which provides a foundation for all actions
- There are also other APIs such as
  - DocumentDB
  - Mongo DB
  - Apache Cassandra
  - Table
  - Graph

Azure Cosmos DB REST API

- The REST API interacts with Azure Cosmos DB using the HTTP protocol.

Resources	Base URI
Database	{base}/dbs/{db}
User	{base}/dbs/{db}/users/{user}
Permission	{base}/dbs/{db}/users/{user}/permissions/{perm}
Collection	{base}/dbs/{db}/colls/{coll}
Stored Procedure	{base}/dbs/{db}/colls/{coll}/sprocs/{sproc}
Trigger	{base}/dbs/{db}/colls/{coll}/triggers/{trigger}
UDF	{base}/dbs/{db}/colls/{coll}/udfs/{udf}
Document	{base}/dbs/{db}/colls/{coll}/docs/{doc}
Attachment	{base}/dbs/{db}/colls/{coll}/docs/{doc}/attachments/{attch}
Offer	{base}/offers/{offer}

DocumentDB API

- The DocumentDB API is built on top of the REST API and is implemented in several languages and platforms including .NET, Java, NodeJS, JavaScript, and Python via their respective SDKs.

  - var dbUrl = "https://productcatalog.documents.azure.com/dbs";
  - var authKey = "the primary or secondary key for the account";
  - client = new DocumentClient(new Uri(dbUrl),authKey);
  - await client.CreateDatabaseAsync(new Database { Id = "Products" });

MongoDB API

- With the MongoDB API, you can leverage your knowledge of MongoDB
- In most cases, an existing MongoDB application would work without any code changes.

```
var host = "host string shown in the Azure portal";
var dbName = "ProductCatalog";
var username = "jose";
var password = "p@sswOrd";
MongoClientSettings settings = new MongoClientSettings();
settings.Server = new MongoServerAddress(host, 10255);
settings.UseSsl = true;
settings.SslSettings = new SslSettings();
settings.SslSettings.EnabledSslProtocols = SslProtocols.Tls12;

MongoIdentity identity =
                    new MongoInternalIdentity(dbName, userName);
MongoIdentityEvidence evidence = new PasswordEvidence(password);

settings.Credentials = new List<MongoCredential>()
    {
        new MongoCredential("SCRAM-SHA-1", identity, evidence)
    };

MongoClient client = new MongoClient(settings);
```

Graph API

- Graphs are very useful to understand a wide range of datasets in different fields such as science and business

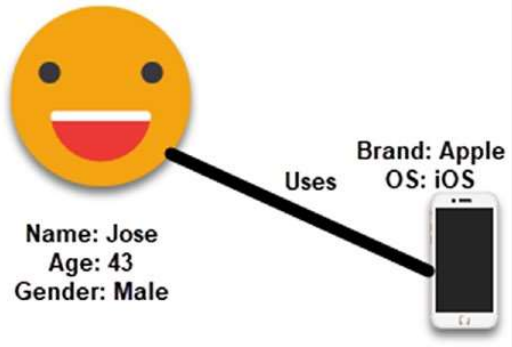


Table API

- An Azure Cosmos DB account implementing the Table API provides the same functionality as Azure Table storage
- With the benefits of scalability and throughput from Cosmos DB.

```
CloudStorageAccount storageAccount =
                    CloudStorageAccount.Parse(connectionString);
CloudTableClient tableClient =
                        storageAccount.CreateCloudTableClient();
CloudTable table = tableClient.GetTableReference("products");
table.CreateIfNotExists();
ProductEntity item = new ProductEntity()
                            {
                                 PartitionKey =
                                        Guid.NewGuid().ToString(),
                                 RowKey = Guid.NewGuid().ToString(),
                                 Name = $"Oranges",
                                 Origin = "Florida"
                            };
TableOperation insertOperation = TableOperation.Insert(item);
table.Execute(insertOperation);
```

Thank You