

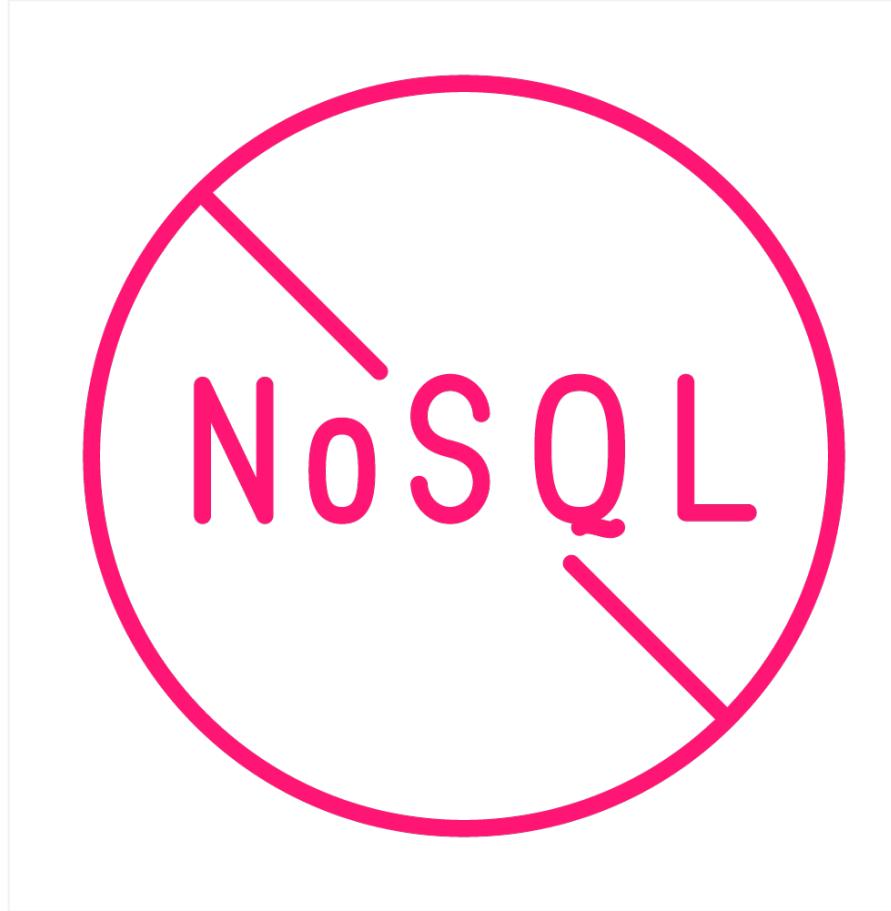
Working with NoSQL



Filip Ekberg

Principal Consultant & CEO

@fekberg | fekberg.com



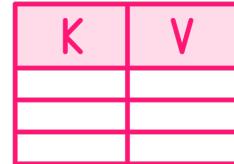
Non-relational Database

Commonly referred to as NoSQL

A group of many different data stores



NoSQL Databases



Key-value store or Key-value cache



Document store or Document database



Object storage

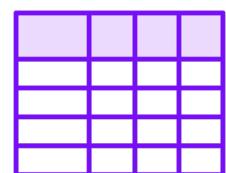
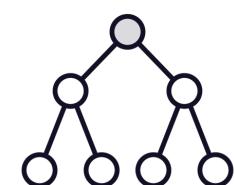


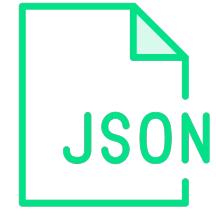
Table storage



Graph database



NoSQL Databases in This Module



Document database
JSON serialized data

K	V

Key-value store
Persistent dictionary

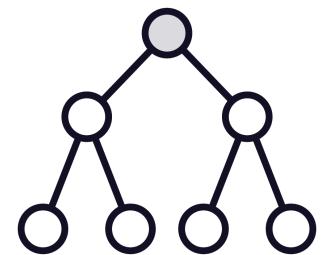
Table store
Tabular data, often terabytes of it



NoSQL or Relational Database



A relational database engine enforces relationships



Relational data is not verified by the NoSQL database engine

A … B … C
↓
A … B … C

**Data in NoSQL is accessed through keys
Can be used to define a relationship but this will NOT be enforced**



Document Database

Store documents

JSON, XML, or other supported
formats

All data in one place

Useful to store entities with all its
related data in one place



Example: Document in a Document Database

```
{  
  "InvoiceId": "IN123",  
  "CustomerId": "1",  
  "Due": "2025-01-01",  
  "Items": [],  
  "Total": 1999.50  
}
```



Invoice

```
public class Invoice
{
    public string InvoiceId { get; set; }
    public Guid CustomerId { get; set; }
    public DateOnly Due { get; set; }
    public IEnumerable<Item> Items { get; set; }
    public decimal Total { get; set; }
}
```



Access a Document

Partition Key

A good partitioning leads to better scaling

Row Key

The key for the row in the specified partition



**NoSQL databases are often
very fast and scalable**



Key Value Store

Dictionary

JSON, binary, simple or complex types

Good for caching!

It's used like a hosted, scalable dictionary to cache data



Redis is a common key value store



Feature Toggle

```
// Ask the Key Value Store for a Boolean value  
if(FeatureService.IsEnabled("Order_Page_2.0"))  
{  
    ...  
}
```



Table Store

Tabular data

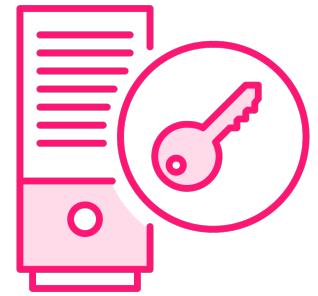
Looks like any column from a traditional database, without relationships

Good for LOTS of data

Not uncommon to be used for terabytes of data

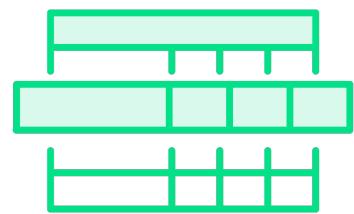


Document Database Content



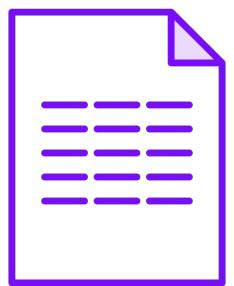
Partition Key

Use to determine in which partition to store the data.
A good partition key makes scaling easier and better.



Row Key

Commonly referred to as id and is used to index the data in a partition. Has to be unique!



Data

The data serialized to a supported format. Commonly stored as JSON.



Distributed Cache (Key Value Store)

```
public string GetDetailsFor(string itemId)
{
    var details = database.StringGet(itemId);

    if (details.HasValue)
    {
        return details;
    }
    var loadedDetails = ...;

    database.StringSet(itemId, loadedDetails);

    return loadedDetails;
}
```



Always access data through its key!

Not meant to be used to iterate through all cached items.



**Introduce a (distributed)
cache to store data that is
frequently accessed**



Redis is an in-memory cache

If the instance/server restart all data is lost.



Using a Table Store

```
TableServiceClient tableServiceClient = new("CONNECTION STRING");  
  
TableClient tableClient =  
    tableServiceClient.GetTableClient("SupportRequests");  
  
tableClient.CreateIfNotExists();  
  
tableClient.AddEntity(...);
```



Access a Row in the Table

Partition Key

A good partitioning leads to better scaling

Row Key

The key for the row in the specified partition

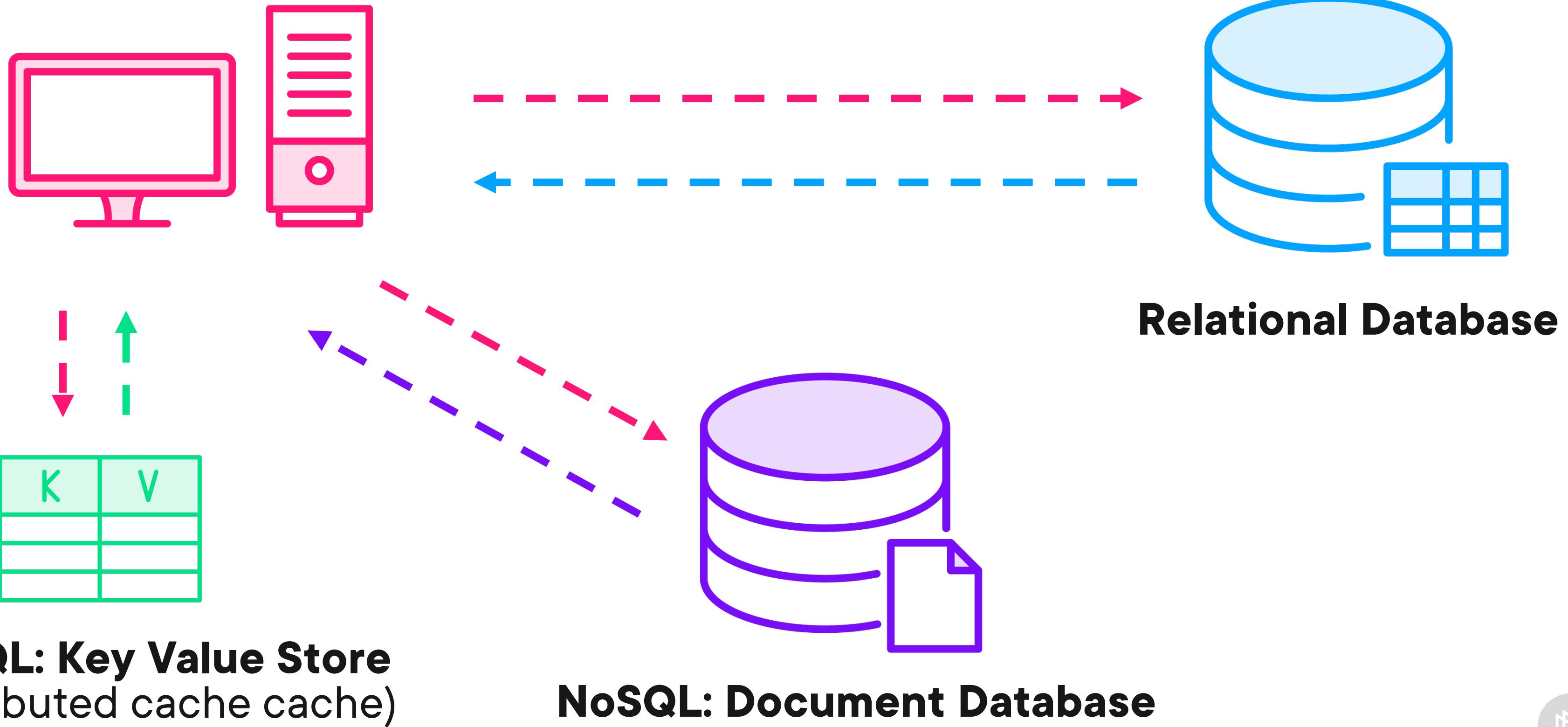


**Choose a partition key that
evenly distributes your data**

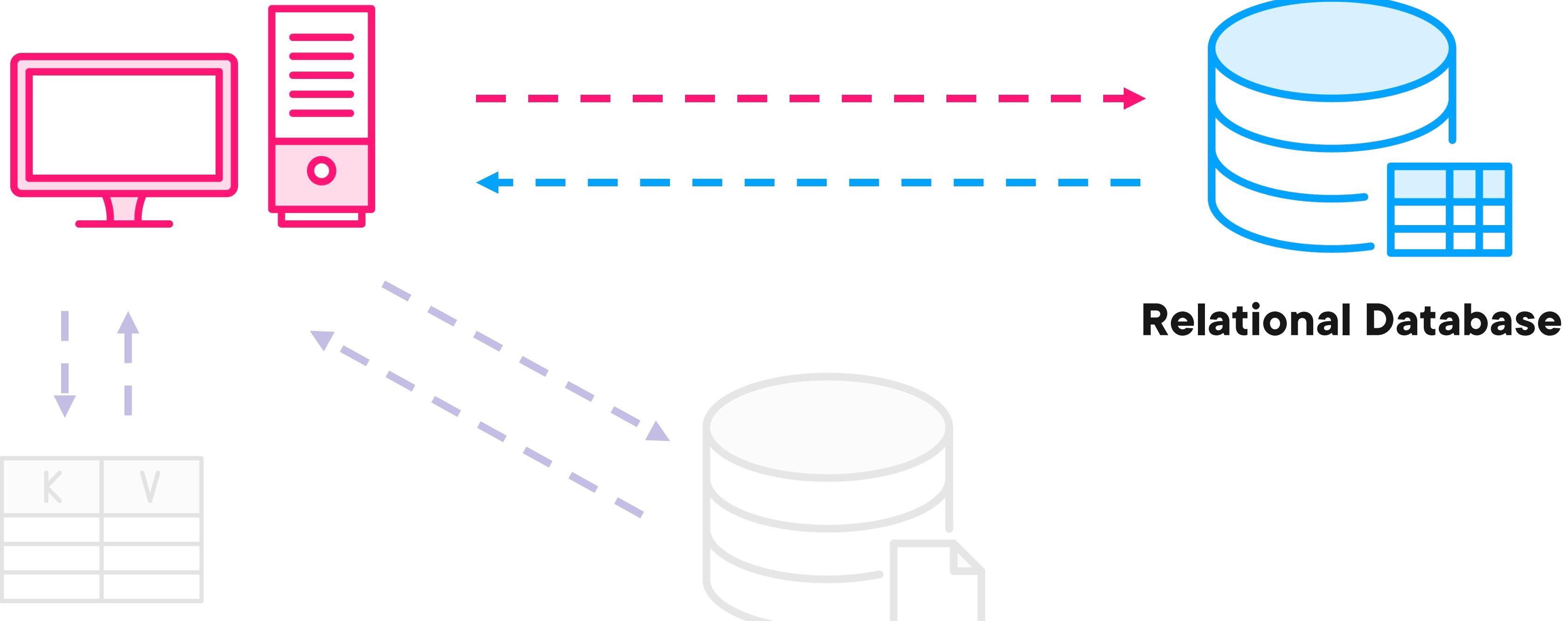
Warehouse identifier?
Customer Identifier?



Using More Than One Database



Using More Than One Database

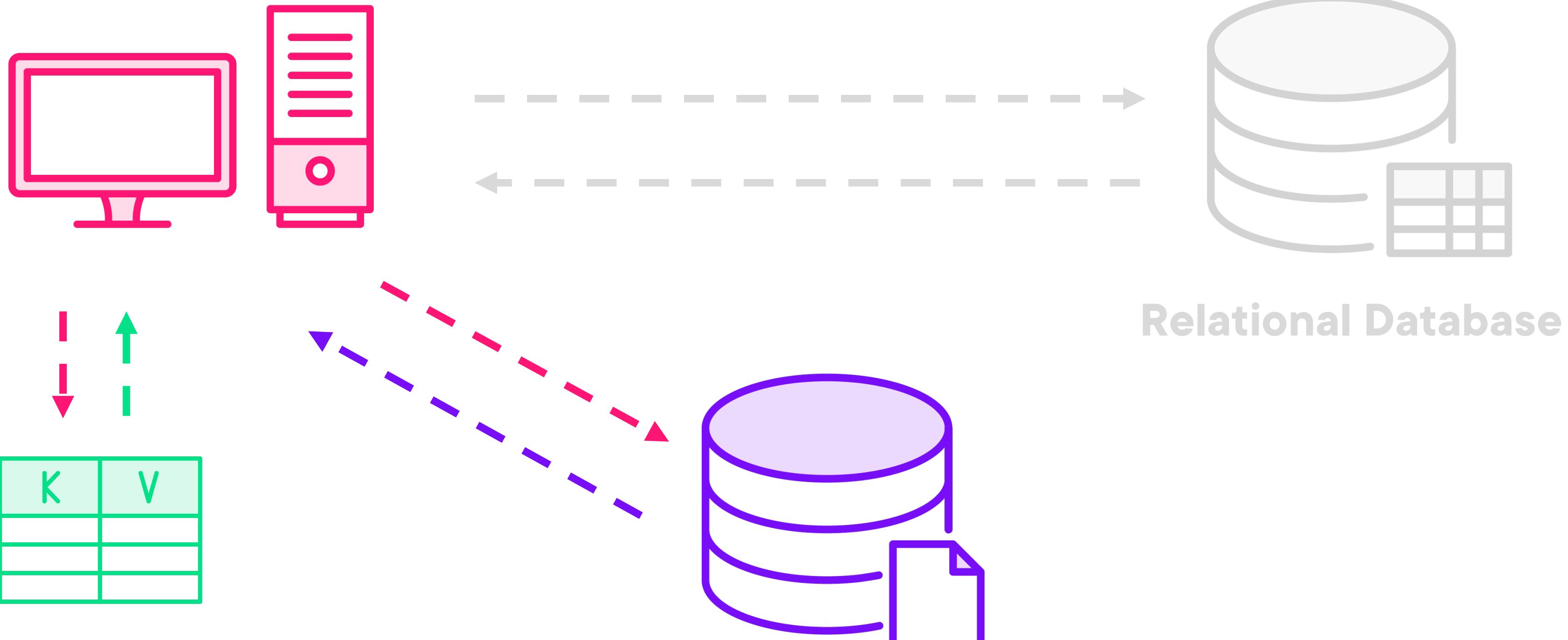


NoSQL: Key Value Store
(distributed cache cache)

NoSQL: Document Database



Using More Than One Database



NoSQL: Key Value Store
(distributed cache cache)

NoSQL: Document Database



Cloud Hosted NoSQL Databases

Scalable

High performance



Know your pricing!

Always read the documentation to avoid unnecessary cost



Create a Document

```
using var client = new CosmosClient(  
    accountEndpoint: "ADD THE ENDPOINT HERE",  
    authKeyOrResourceToken: "ADD THE TOKEN HERE"  
);  
  
Database database =  
    await client.CreateDatabaseIfNotExistsAsync("WarehouseManagementSystem");  
  
Container container = await database.CreateContainerIfNotExistsAsync(  
    id: "invoices",  
    partitionKeyPath: "/customerId", throughput: 400);  
  
await container.UpsertItemAsync(invoice);
```



Create a Document

```
using var client = new CosmosClient(  
    accountEndpoint: "ADD THE ENDPOINT HERE",  
    authKeyOrResourceToken: "ADD THE TOKEN HERE"  
);
```

```
Database database =  
    await client.CreateDatabaseIfNotExistsAsync("WarehouseManagementSystem");
```

```
Container container = await database.CreateContainerIfNotExistsAsync(  
    id: "invoices",  
    partitionKeyPath: "/customerId", throughput: 400);
```

```
await container.UpsertItemAsync(invoice);
```



Automatically serializes data to json



There's a lot more to explore!



Redis

In-memory key value store

Distributed cache



Using Redis

```
static ConnectionMultiplexer connection =  
    ConnectionMultiplexer.Connect("CONNECTION STRING");  
  
IDatabase database = connection.GetDatabase();  
  
var cachedItem = database.StringGet(itemId);  
  
if(cachedItem.HasValue)  
{  
    return cachedItem;  
}  
  
database.StringSet(itemId, "the data");
```



Using Redis

```
static ConnectionMultiplexer connection =  
    ConnectionMultiplexer.Connect("CONNECTION STRING");  
  
IDatabase database = connection.GetDatabase();  
  
var cachedItem = database.StringGet(itemId);  
  
if(cachedItem.HasValue)  
{  
    return cachedItem;  
}  
  
database.StringSet(itemId, "the data");
```



Add caching in hot paths to reduce load times!



Automatic Expiration of Cached Items

```
database.StringSet(itemId,  
    "the data",  
    expiry: TimeSpan.FromHours(1)  
);
```



You can reference other data!

It will not be enforced by the database.



Table Store

Built for extreme amounts of data

Very fast



Using a Table Store

```
TableServiceClient tableServiceClient = new("CONNECTION STRING");  
  
TableClient tableClient =  
    tableServiceClient.GetTableClient("SupportRequests");
```



Using a Table Store

```
TableServiceClient tableServiceClient = new("CONNECTION STRING");  
  
TableClient tableClient =  
    tableServiceClient.GetTableClient("SupportRequests");  
  
tableClient.CreateIfNotExists();
```



Using a Table Store

```
TableServiceClient tableServiceClient = new("CONNECTION STRING");  
  
TableClient tableClient =  
    tableServiceClient.GetTableClient("SupportRequests");  
  
tableClient.CreateIfNotExists();  
  
tableClient.AddEntity(...);  
  
var requests = tableClient.Query<SupportRequest>(  
    r => r.PartitionKey == customerId.ToString()  
);
```



**You now have the tools to
work with data access in
your applications!**

