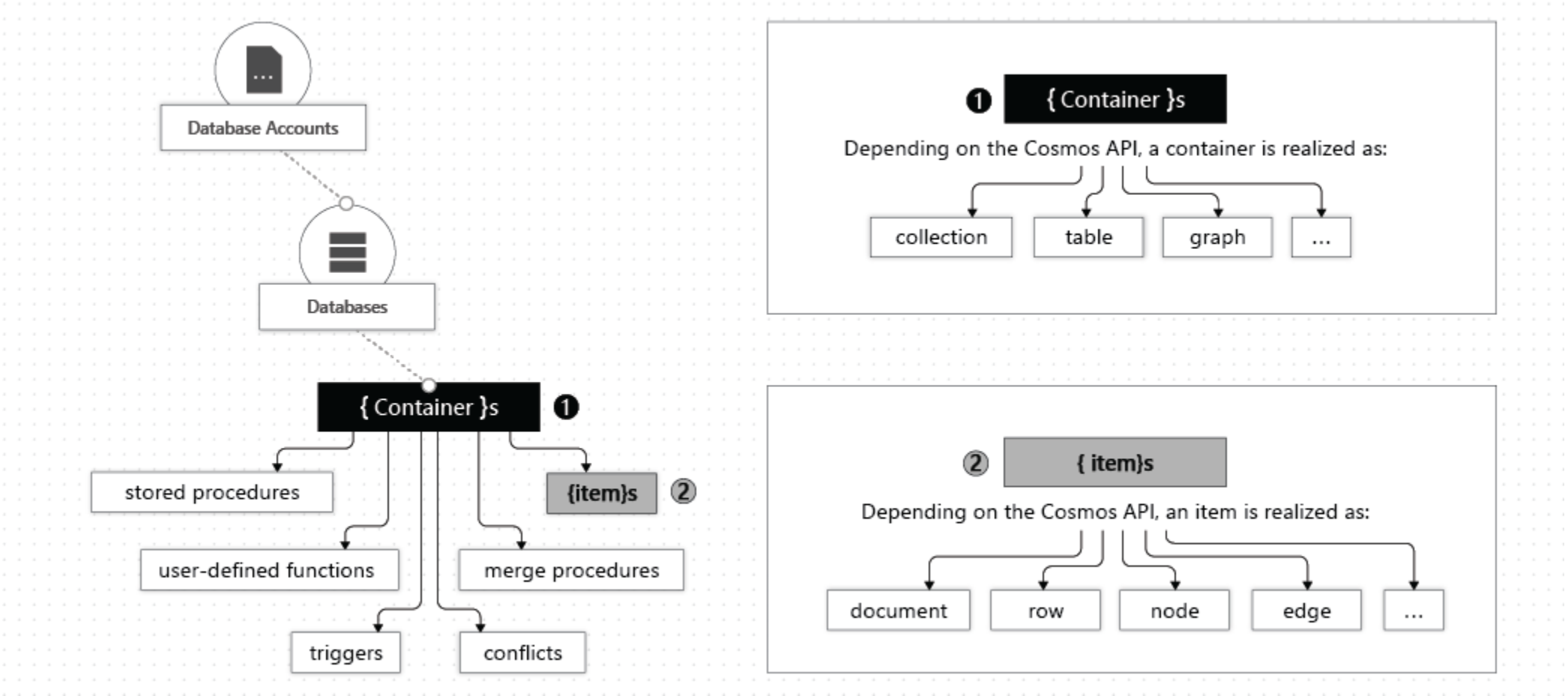
Azure Cosmos DB

Jiří Resler

What is it?

- NoSQL database
- Developed by Microsoft
- Runs in Azure
- PaaS
- Multi-model: document, graph, key-value, column-family
- Offers multiple database APIs

Data model



Source: https://docs.microsoft.com/en-us/azure/cosmos-db/media/account-databases-containers-items/cosmos-entities.png

Database APIs

- Core (SQL) API
- Table API
- API for MongoDB
- Cassandra API
- Gremlin API

Core (SQL) API

- Native for Cosmos DB
- JSON documents
- Container = Collection
- Item = Document (JSON)

```
private async Task CreateDatabase()
{
    await this.cosmosClient.CreateDatabaseIfNotExistsAsync(databaseId);
}
private async Task CreateContainer()
{
    await this.database.CreateContainerIfNotExistsAsync(containerId, "/LastName");
}
```

Core (SQL) API - insert

```
private async Task AddItemsToContainer()
{
    // Create a family object for the Andersen family
    Family andersenFamily = new Family
    {
        Id = "Andersen.1",
        LastName = "Andersen",
        Parents = new Parent[]
        {
    new Parent { FirstName = "Thomas" },
    new Parent { FirstName = "Mary Kay" }
        },
        Children = new Child[]
        {
    new Child
    {
        FirstName = "Henriette Thaulow",
        Gender = "female",
        Grade = 5,
    };
```

```
await this.container.CreateItemAsync<Family>(andersenFamily, new PartitionKey(andersenFamily.LastName));
```

Core (SQL) API – reading data

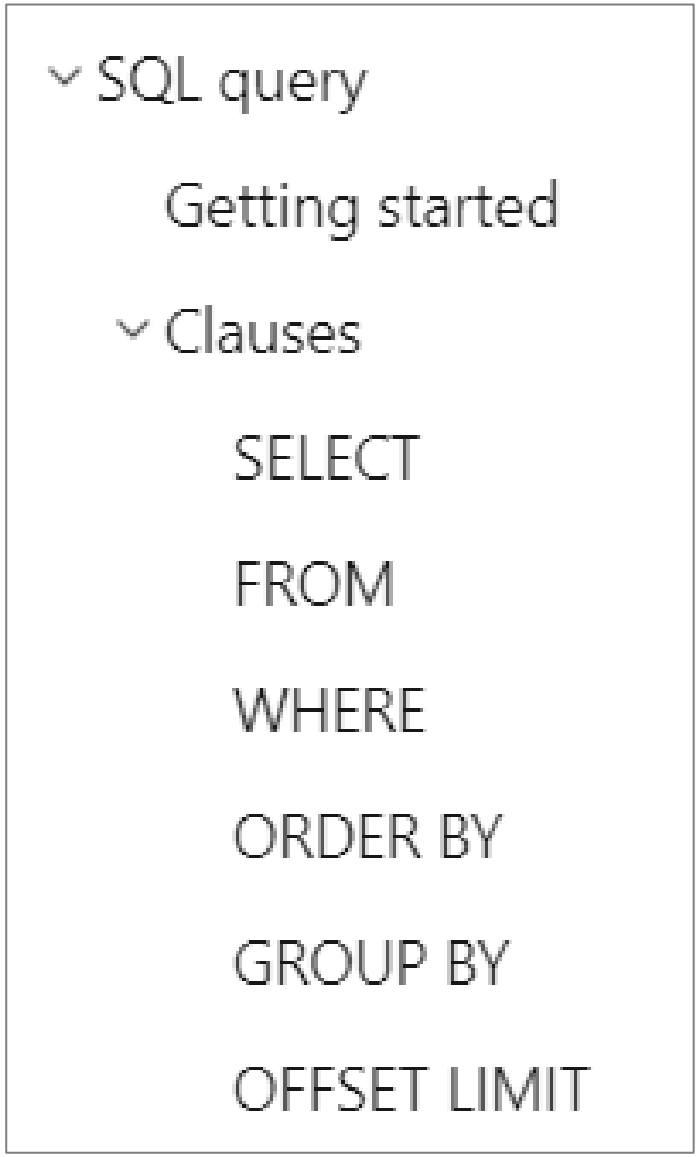
```
private async Task PointRead()
{
    Family family = await this.container.ReadItemAsync<Family>("Andersen.1", new PartitionKey("Andersen"));
    Console.WriteLine(family);
}
```

Core (SQL) API – reading data

```
private async Task QueryItems()
{
    var sqlQueryText = "SELECT * FROM c WHERE c.LastName = 'Andersen'";

    QueryDefinition queryDefinition = new QueryDefinition(sqlQueryText);
    using FeedIterator<Family> queryResultSetIterator = this.container.GetItemQueryIterator<Family>(queryDefinition);

    List<Family> families = new List<Family>();

    while (queryResultSetIterator.HasMoreResults)
    {
        FeedResponse<Family> currentResultSet = await queryResultSetIterator.ReadNextAsync();
        foreach (Family family in currentResultSet)
        {
            families.Add(family);
        }
    }
}
```

Core (SQL) API

- Subqueries
- Joins
- Aggregate functions



Core (SQL) API

- Array functions
- Date and Time functions
- Mathematical functions
- Spatial functions
- String functions
- Type checking functions

Table API

- Key/Value
- Container = Table
- Item = Row
- Key/Value => Partition key & Row key/Row value

```
TableOperation getOperation = TableOperation.Retrieve<HeroEntity>(partitionKey, rowKey);
TableResult result = await table.ExecuteAsync(getOperation);
HeroEntity heroEntity = result.Result as HeroEntity;
```

API for MongoDB

- JSON documents
- Container = Collection
- Item = Document
- .NET, Python, Java, Node.js, Golang


```
public Task<Product> GetByIdAsync(string id)
{
    return _products.Find(p => p.id == id).FirstOrDefaultAsync();
}
```

API for MongoDB

Command	Supported
$arrayElemAt	Yes
$arrayToObject	Yes
$concatArrays	Yes
$filter	Yes
$indexOfArray	Yes

Cassandra API

- Column-family
- Container = Table (column family)
- Item = Row
- .NET, Python, Java, Node.js, Golang, CQLSH

```
await mapper.InsertAsync<User>(new User(1, "LyubovK", "Dubai"));
```

Cassandra API

```
// Query to get all user's information
foreach (User user in await mapper.FetchAsync<User>("Select * from user"))
{
    Console.WriteLine(user);
}

// Query to get a single user's information
mapper.FirstOrDefault<User>("Select * from user where user_id = ?", 3);
```

Gremlin API

- Graph
- Container = Graph
- Item = Vertex/Edge
- .NET, Python, Java, Node.js, PHP, Gremlin console

Gremlin API

```
// Insert the "Thomas" vertex into the graph
:> g.addV('person').property('id', 'thomas.1').property('firstName', 'Thomas').property
('lastName', 'Andersen').property('age', 44)

// Insert a "knows" edge between Thomas and Robin
:> g.V('thomas.1').addE('knows').to(g.V('robin.1'))

// Return all "person" vertices in descending order of their first names
:> g.V().hasLabel('person').order().by('firstName', decr)

// "What operating systems do friends of Thomas use?"
:> g.V('thomas.1').out('knows').out('uses').out('runsos').group().by('name').by(count())
```

Pros and cons of Azure Cosmos DB

- Pros
  - PaaS – We do not have to take care of hardware infrastructure
  - Automatic scaling
  - Several data consistency models to choose from
  - High availability backed by SLAs
  - Low latency
  - Support for APIs
- Cons
  - Proprietary except for open source APIs
  - Paid, but free to some extent with the free pricing tier

Thank you for your attention.