100 XP

Identify Azure Cosmos DB APIs

6 minutes

Azure Cosmos DB is Microsoft's fully managed and serverless distributed database for applications of any size or scale, with support for both relational and non-relational workloads. Developers can build and migrate applications fast using their preferred open source database engines, including PostgreSQL, MongoDB, and Apache Cassandra. When you provision a new Cosmos DB instance, you select the database engine that you want to use. The choice of engine depends on many factors including the type of data to be stored, the need to support existing applications, and the skills of the developers who will work with the data store.

Azure Cosmos DB for NoSQL

Azure Cosmos DB for NoSQL is Microsoft's native non-relational service for working with the document data model. It manages data in JSON document format, and despite being a NoSQL data storage solution, uses SQL syntax to work with the data.

A SQL query for an Azure Cosmos DB database containing customer data might look similar to this:

SQL

```sql
SELECT *
FROM customers c
WHERE c.id = "joe@litware.com"
```

The result of this query consists of one or more JSON documents, as shown here:

JSON

```json
{
    "id": "joe@litware.com",
    "name": "Joe Jones",
    "address": {
         "street": "1 Main St.",
         "city": "Seattle"
```

```
    }
}
```

Azure Cosmos DB for MongoDB

MongoDB is a popular open source database in which data is stored in Binary JSON (BSON) format. Azure Cosmos DB for MongoDB enables developers to use MongoDB client libraries and code to work with data in Azure Cosmos DB.

MongoDB Query Language (MQL) uses a compact, object-oriented syntax in which developers use *objects* to call *methods*. For example, the following query uses the **find** method to query the **products** collection in the **db** object:

JavaScript

```
db.products.find({id: 123})
```

The results of this query consist of JSON documents, similar to this:

JSON

```
{
   "id": 123,
   "name": "Hammer",
   "price": 2.99
}
```

Azure Cosmos DB for PostgreSQL

Azure Cosmos DB for PostgreSQL is a native PostgreSQL, globally distributed relational database that automatically shards data to help you build highly scalable apps. You can start building apps on a single node server group, the same way you would with PostgreSQL anywhere else. As your app's scalability and performance requirements grow, you can seamlessly scale to multiple nodes by transparently distributing your tables. PostgreSQL is a relational database management system (RDBMS) in which you define relational tables of data, for example you might define a table of products like this:

ProductID	ProductName	Price
123	Hammer	2.99
162	Screwdriver	3.49

You could then query this table to retrieve the name and price of a specific product using SQL like this:

SQL

```
SELECT ProductName, Price
FROM Products
WHERE ProductID = 123;
```

The results of this query would contain a row for product 123, like this:

ProductName	Price
Hammer	2.99

Azure Cosmos DB for Table

Azure Cosmos DB for Table is used to work with data in key-value tables, similar to Azure Table Storage. It offers greater scalability and performance than Azure Table Storage. For example, you might define a table named Customers like this:

PartitionKey	RowKey	Name	Email
1	123	Joe Jones	joe@litware.com
1	124	Samir Nadoy	samir@northwind.com

You can then use the Table API through one of the language-specific SDKs to make calls to your service endpoint to retrieve data from the table. For example, the following request returns the row containing the record for *Samir Nadoy* in the table above:

text

```
https://endpoint/Customers(PartitionKey='1',RowKey='124')
```

Azure Cosmos DB for Apache Cassandra

Azure Cosmos DB for Apache Cassandra is compatible with Apache Cassandra, which is a popular open source database that uses a column-family storage structure. Column families are tables, similar to those in a relational database, with the exception that it's not mandatory for every row to have the same columns.

For example, you might create an **Employees** table like this:

ID	Name	Manager
1	Sue Smith	
2	Ben Chan	Sue Smith

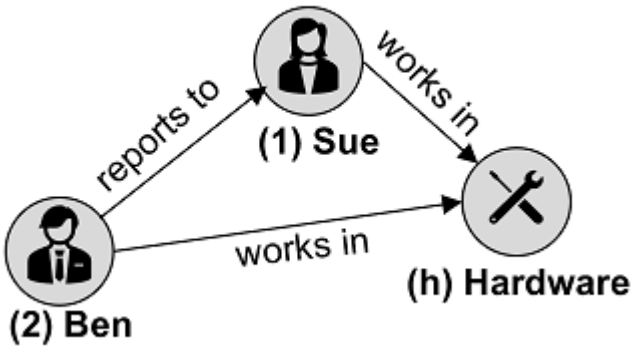
Cassandra supports a syntax based on SQL, so a client application could retrieve the record for *Ben Chan* like this:

SQL

```
SELECT * FROM Employees WHERE ID = 2
```

Azure Cosmos DB for Apache Gremlin

Azure Cosmos DB for Apache Gremlin is used with data in a graph structure; in which entities are defined as vertices that form nodes in connected graph. Nodes are connected by edges that represent relationships, like this:



The example in the image shows two kinds of vertex (employee and department) and edges that connect them (employee "Ben" reports to employee "Sue", and both employees work in the "Hardware" department).

Gremlin syntax includes functions to operate on vertices and edges, enabling you to insert, update, delete, and query data in the graph. For example, you could use the following code to add a new employee named *Alice* that reports to the employee with ID **1** (*Sue*)

```
g.addV('employee').property('id', '3').property('firstName', 'Alice')
g.V('3').addE('reports to').to(g.V('1'))
```

The following query returns all of the *employee* vertices, in order of ID.

```
g.V().hasLabel('employee').order().by('id')
```

Next unit: Exercise: Explore Azure Cosmos DB

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆