**Identify Azure Cosmos DB APIs**

Completed100 XP

* 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:

SQLCopy

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:

JSONCopy

{

"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:

JavaScriptCopy

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

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

JSONCopy

{

"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:

Expand table

| **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:

SQLCopy

SELECT ProductName, Price

FROM Products

WHERE ProductID = 123;

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

Expand table

| **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:

Expand table

| **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:

textCopy

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:

Expand table

| **ID** | **Name** | **Manager** |
| --- | --- | --- |
| 1 | Sue Smith |  |
| 2 | Ben Chan | Sue Smith |

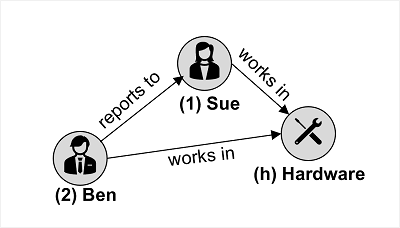
Cassandra supports a syntax based on SQL, so a client application could retrieve the record for *Ben Chan* like this:

SQLCopy

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*)

Copy

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.

Copy

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

**Azure Cosmos DB: Cơ sở dữ liệu đám mây phân tán, đa mô hình**

**Azure Cosmos DB** là dịch vụ cơ sở dữ liệu đám mây được quản lý hoàn toàn của Microsoft, hỗ trợ lưu trữ và truy vấn dữ liệu hiệu suất cao cho các ứng dụng web và di động ở mọi quy mô. Cosmos DB cung cấp tính linh hoạt cao, cho phép bạn chọn lựa giữa cơ sở dữ liệu quan hệ (relational) hoặc phi quan hệ (NoSQL) tùy theo nhu cầu của ứng dụng. Bên cạnh đó, Cosmos DB còn tương thích với nhiều công cụ mã nguồn mở phổ biến như PostgreSQL, MongoDB và Apache Cassandra, giúp các nhà phát triển dễ dàng xây dựng và di chuyển ứng dụng lên nền tảng đám mây Azure.

**Các loại Azure Cosmos DB:**

* **Azure Cosmos DB cho NoSQL:** Đây là dịch vụ lưu trữ dữ liệu phi quan hệ riêng của Microsoft, sử dụng định dạng tài liệu JSON để quản lý dữ liệu. Mặc dù là NoSQL nhưng Cosmos DB vẫn cho phép truy vấn dữ liệu bằng cú pháp SQL quen thuộc.

Ví dụ: Truy vấn dữ liệu khách hàng trong Cosmos DB với cú pháp SQL:

SQL

SELECT \*

FROM customers c

WHERE c.id = "joe@litware.com"

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

Kết quả trả về là một hoặc nhiều tài liệu JSON, ví dụ:

JSON

{

"id": "joe@litware.com",

"name": "Joe Jones",

"address": {

"street": "1 Main St.",

"city": "Seattle"

}

}

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

* **Azure Cosmos DB cho MongoDB:** Là dịch vụ tương thích với MongoDB, một công cụ lưu trữ dữ liệu NoSQL mã nguồn mở phổ biến, sử dụng định dạng Binary JSON (BSON) để lưu trữ dữ liệu. Cosmos DB for MongoDB cho phép các nhà phát triển tận dụng các thư viện và mã client của MongoDB để làm việc với dữ liệu trên nền tảng đám mây Azure.

Ví dụ: Truy vấn dữ liệu sản phẩm trong bộ sưu tập "products" của đối tượng "db" bằng cú pháp truy vấn MongoDB (MQL):

JavaScript

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

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

Kết quả trả về là các tài liệu JSON, ví dụ:

JSON

{

"id": 123,

"name": "Hammer",

"price": 2.99

}

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

* **Azure Cosmos DB cho PostgreSQL:** Là dịch vụ cơ sở dữ liệu quan hệ phân tán toàn cầu tương thích với PostgreSQL. Cosmos DB for PostgreSQL tự động phân vùng dữ liệu để giúp bạn xây dựng các ứng dụng có khả năng mở rộng cao. Bạn có thể bắt đầu xây dựng ứng dụng trên một nhóm server đơn, sau đó dễ dàng mở rộng sang nhiều node hơn khi nhu cầu về tính năng và hiệu suất của ứng dụng tăng lên. PostgreSQL là hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) cho phép bạn định nghĩa các bảng dữ liệu quan hệ.

Ví dụ: Bảng "Products" lưu trữ thông tin sản phẩm:

|  |  |  |
| --- | --- | --- |
| ProductID | ProductName | Price |
| 123 | Hammer | 2.99 |
| 162 | Screwdriver | 3.49 |

drive\_spreadsheetExport to Sheets

Truy vấn bảng "Products" để lấy tên và giá của sản phẩm có ID 123 bằng cú pháp SQL:

SQL

SELECT ProductName, Price

FROM Products

WHERE ProductID = 123;

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

Kết quả trả về là một dòng dữ liệu cho sản phẩm 123:

|  |  |
| --- | --- |
| ProductName | Price |
| Hammer | 2.99 |

drive\_spreadsheetExport to Sheets

* **Azure Cosmos DB cho Table:** Là dịch vụ lưu trữ dữ liệu dạng bảng key-value, tương tự như Azure Table Storage nhưng có khả năng mở rộng và hiệu suất cao hơn.

Ví dụ: Bảng "Customers" lưu trữ thông tin khách hàng:

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

drive\_spreadsheetExport to Sheets

Sử dụng Table API để truy xuất dữ liệu từ bảng "Customers". Ví dụ, yêu cầu GET sau sẽ trả về dòng dữ liệu của khách hàng "Samir Nadoy":

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

* **Azure Cosmos DB cho Apache Cassandra:** Là dịch vụ tương thích với Apache Cassandra, một công cụ lưu trữ dữ liệu NoSQL mã nguồn mở phổ biến, sử dụng cấu trúc lưu trữ theo "family cột" (column-family). "Family
* **Azure Cosmos DB cho Apache Gremlin:** Là dịch vụ lưu trữ dữ liệu dạng đồ thị (graph), trong đó các thực thể được định nghĩa là các đỉnh (vertex) tạo thành các nút trong đồ thị được kết nối. Các nút được kết nối bởi các cạnh (edge) thể hiện mối quan hệ.

Ví dụ: Đồ thị thể hiện mối quan hệ giữa nhân viên và phòng ban:

Đồ thị thể hiện mối quan hệ giữa nhân viên và phòng ban: [đã xoá URL không hợp lệ]

Hình ảnh minh họa hai loại đỉnh (nhân viên và phòng ban) và các cạnh kết nối chúng (nhân viên "Ben" báo cáo với nhân viên "Sue", và cả hai nhân viên đều làm việc trong phòng ban "Hardware").

Cú pháp Gremlin bao gồm các hàm để thao tác trên các đỉnh và cạnh, cho phép bạn chèn, cập nhật, xóa và truy vấn dữ liệu trong đồ thị. Ví dụ, bạn có thể sử dụng mã sau để thêm một nhân viên mới tên "Alice" báo cáo với nhân viên có ID 1 (Sue):

g.addV('employee').property('id', '3').property('firstName', 'Alice')

g.V('3').addE('reports to').to(g.V('1'))

Truy vấn sau trả về tất cả các đỉnh "nhân viên" theo thứ tự ID:

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