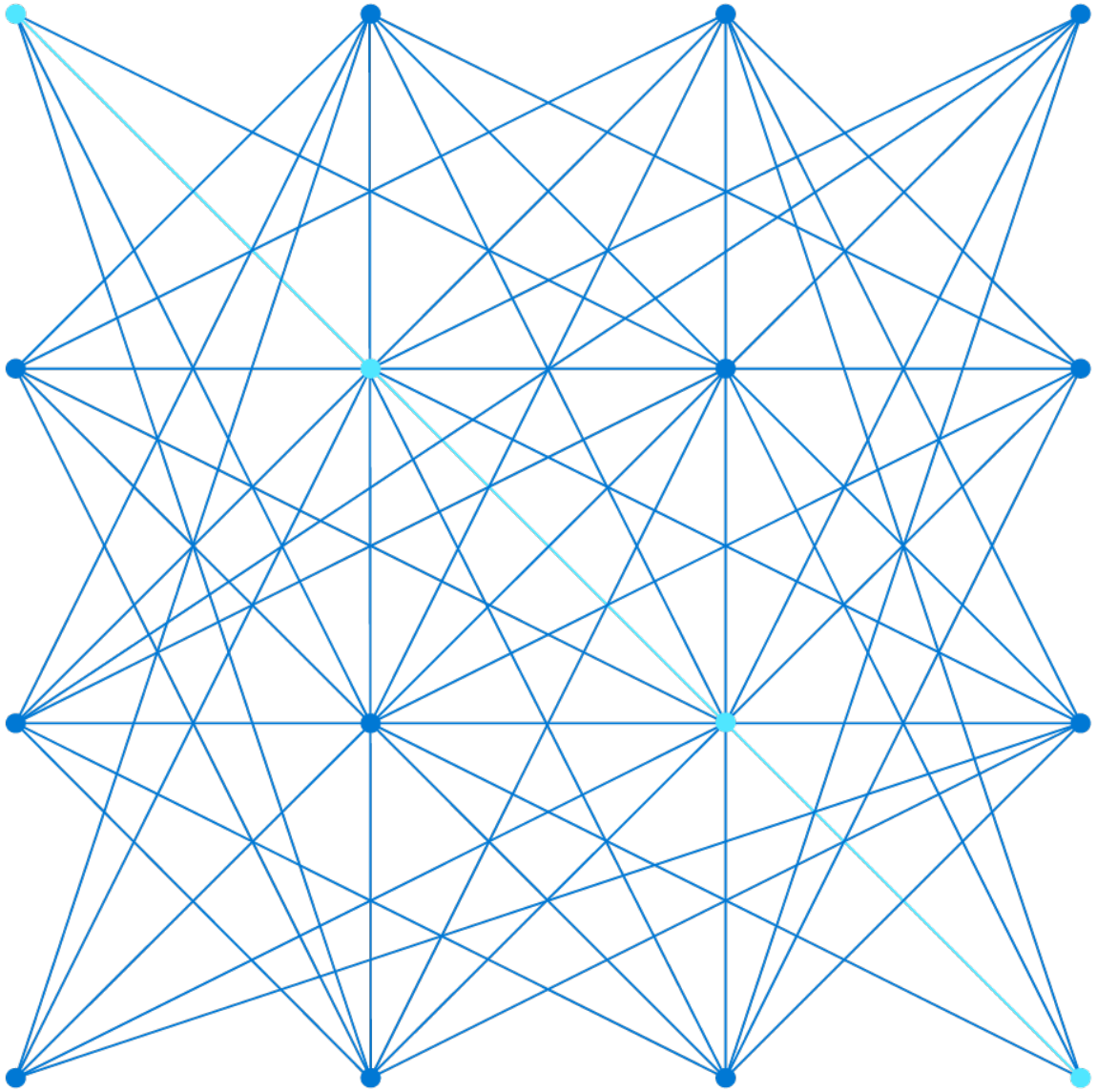
Microsoft Azure

Azure Cosmos DB for NoSQL에서 쿼리 실행



안건

Azure Cosmos DB for NoSQL 쿼리

Azure Cosmos DB for NoSQL을 사용하여 복잡한 쿼리 작성

Azure Cosmos DB for NoSQL 쿼리

SQL을 사용하여 쿼리 만들기

Azure Cosmos DB for NoSQL(SQL)은 구문을 사용하여 반정형 데이터에 대한 쿼리를 수행합니다.

Azure Cosmos DB SQL API SQL 쿼리의 몇 가지 예입니다.

```
SELECT * FROM products
```

```
SELECT products.id,
    products.name,
    products.price,
    products.categoryName FROM products
```

```
SELECT p.name,
    p.price
FROM p
```

```
SELECT ReferenceContainerHoweverYouLike.name,
 ReferenceContainerHoweverYouLike.price
FROM ReferenceContainerHoweverYouLike
```

```
SELECT p.name,
    p.categoryName,
    p.price
FROM products p
WHERE p.price >= 50
    AND p.price <= 100
```

쿼리 결과 추정

Azure Cosmos DB for NoSQL은 JSON 결과를 조작하기 위해 SQL을 확장합니다.

AS

```
SELECT p.categoryName AS category
FROM products p
```

DISTINCT

```
SELECT DISTINCT p.categoryName
FROM products p
```

```
public class CategoryReader {
    public string CategoryName { get; set; }
}
// Developers read this as List<CategoryReader>
```

값

```
SELECT VALUE p.categoryName
FROM products p
```

```
// Developers read this as List<string>
```

Child properties

```
// Suppose we have the following ProductAdvertisement class definition.
public class ProductAdvertisement {
    public string Name { get; set; }
    public string Category { get; set; }
    public class ScannerData { public decimal Price { get; set; } }
}
```

```
// scannerData expression added to return the expected class format.
SELECT p.name,
    p.categoryName AS category,
    { "price": p.price } AS scannerData
FROM products p
WHERE p.price >= 50 AND p.price <= 100
```

```
// The query will return this JSON.
{
    "name": "LL Bottom Bracket",
    "category": "Components, Bottom Brackets",
    "scannerData": { "price": 53.99 }
}
```

쿼리에서 형식 검사 구현

NoSQL은 스키마가 없으므로 형식 검사가 종종 쿼리에 포함됩니다.

IS_DEFINED

```
// Let's assume this is a document on the Product container.
{
    "id": "6374995F-9A78-43CD-AE0D-5F6041078140",
    "categoryid": "3E4CEACD-D007-46EB-82D7-31F6141752B2",
    "sku": "FR-R38R-60",
    "name": "LL Road Frame - Red, 60",
    "price": 337.22
}
```

```
-- Note how in the previous document there are no tags properties
SELECT IS_DEFINED(p.tags) AS tags_exist
FROM products p
```

```
// This query returns.
[ { "tags_exist": false } ]
```

다른 형식 검사 함수

- IS_BOOLEAN
- IS_OBJECT

IS_ARRAY, IS_NULL, IS_STRING, IS_NUMBER

```
// Let's assume this is a document on the Product container.
{
    "id": "6374995F-9A78-43CD-AE0D-5F6041078140",
    "categoryid": "3E4CEACD-D007-46EB-82D7-31F6141752B2",
    "sku": "FR-R38R-60",
    "name": "LL Road Frame - Red, 60",
    "price": 337.22, "tags": "fun, sporty, rad" }
```

```
-- IS_ARRAY
SELECT IS_ARRAY(p.tags) AS tags_is_array
FROM products p

-- IS_NULL
SELECT IS_NULL(p.tags) AS tags_is_null
FROM products p

-- IS_NUMBER
SELECT p.id, p.price,(p.price *1.25) AS priceWithTax
FROM products p
WHERE IS_NUMBER(p.price)

-- IS_STRING, returns false
SELECT p.id, p.price
FROM products p
WHERE IS_STRING(p.price)
```

기본 제공 함수 사용

Azure Cosmos DB for NoSQL용 SQL이 쿼리의 일반적인 작업에 대한 기본 제공 함수와 함께 제공됩니다.

다음은 이러한 함수의 몇 가지 예입니다.

CONCAT

```
SELECTVALUE CONCAT(p.name,' | ', p.categoryName)
FROM products p
```

LOWER

```
SELECTVALUE LOWER(p.sku)
FROM products p
```

RTRIM, LTRIM, LEFT, RIGHT

```
SELECT VALUE LTRIM(RTRIM(p.sku))
FROM products p

SELECT VALUE LEFT(p.fullname, 10)
FROM Customers c
```

GETCURRENTDATETIME

```
SELECT *
FROM products p
WHERE p.retirementDate >= GetCurrentDateTime()
```

CONTAINS

```
SELECT p.CompanyName
FROM companies p
WHERE CONTAINS(p.CompanyName, "contoso", true)
```

ARRAY_CONTAINS

```
SELECT p.ProductName
FROM products p
WHERE ARRAY_CONTAINS(p.tags, "wheel", true)
```

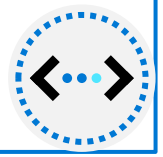
SDK에서 쿼리 실행

SDK의 *Microsoft.Azure.Cosmos.Container* 클래스에는 쿼리를 조작하기 위한 기본 제공 클래스가 있습니다.

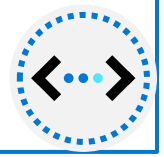
```
// Suppose we want to run the query SELECT * FROM products p.

// Let's define a class for our products.
public class Product
{
    public string id { get; set; }
    public string name { get; set; }
    public string price { get; set; }
}

// Let's create our query definition.
QueryDefinition query = new ("SELECT * FROM products p");

// Let's finally execute our query. We will run it inside a foreach loop in case we get multiple documents back

await foreach (Product product in container.GetItemQueryIterator<Product>(query))
{
    Console.WriteLine($"[{product.id}]\t{product.name,35}\t{product.price,15:C}");
}
```

랩 - Azure Cosmos DB for NoSQL SDK를 사용하여 쿼리 실행

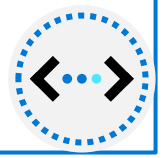
개발 환경 준비



Azure Cosmos DB for NoSQL 계정 만들기

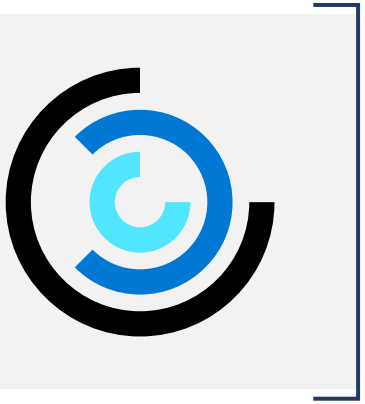


데이터를 사용하여 Azure Cosmos DB for NoSQL 계정 시드



SDK를 사용하여 SQL 쿼리의 결과 반복

Azure Cosmos DB for NoSQL을 사용하여 복잡한 쿼리 작성



SDK를 사용하여 트랜잭션 일괄 처리 만들기

관계형 데이터베이스의 JOIN과 달리 Azure Cosmos DB for NoSQL 범위의 JOIN은 단일 항목에만 해당합니다. JOIN은 단일 항목의 서로 다른 섹션 간에 교차곱을 만듭니다.

```
// Suppose this is a document in our product container.
{
    "id": "80D3630F-B661-4FD6-A296-CD03BB7A4A0C",
    "categoryId": "629A8F3C-CFB0-4347-8DCC-505A4789876B",
    "categoryName": "Clothing, Vests",
    "sku": "VE-C304-L",
    "name": "Classic Vest, L",
    "description": "A worn brown classic",
    "price": 32.4,
    "tags": [
              {
                "id": "2CE9DADE-DCAC-436C-9D69-B7C886A01B77",
                "name": "apparel", "class": "group"
              },
              {
                "id": "CA170AAD-A5F6-42FF-B115-146FADD87298",
                "name": "worn", "class": "trade-in"
              },
              {
                "id": "CA170AAD-A5F6-42FF-B115-146FADD87298",
                "name": "no-damaged", "class": "trade-in"
              }
            ]
}
```

```
-- Return the tag name for all tags embedded in the document
SELECT p.id, p.name, t.name AS tag
FROM products p
JOIN t IN p.tags
```

```
// JSON returned by this query for the document on the left.
[
    {
        "id": "80D3630F-B661-4FD6-A296-CD03BB7A4A0C",
        "name": "Classic Vest, L",
        "tag": "apparel"
    },
    {
         "id": "80D3630F-B661-4FD6-A296-CD03BB7A4A0C",
         "name": "Classic Vest, L",
         "tag": "worn"
    },
    {
         "id": "80D3630F-B661-4FD6-A296-CD03BB7A4A0C",
         "name": "Classic Vest, L",
         "tag": "no-damaged"
     }
]
```

상관 관계가 있는 하위 쿼리 구현

교차곱 집합에 포함할 배열 항목의 수를 필터링하는 하위 쿼리를 작성하여 JOIN 식을 추가로 최적화할 수 있습니다.

```
// Suppose this is a document in our product container.
{
    "id": "80D3630F-B661-4FD6-A296-CD03BB7A4A0C",
    "categoryId": "629A8F3C-CFB0-4347-8DCC-505A4789876B",
    "categoryName": "Clothing, Vests",
    "sku": "VE-C304-L",
    "name": "Classic Vest, L",
    "description": "A worn brown classic",
    "price": 32.4,
    "tags": [
              {
                "id": "2CE9DADE-DCAC-436C-9D69-B7C886A01B77",
                "name": "apparel", "class": "group"
              },
              {
                "id": "CA170AAD-A5F6-42FF-B115-146FADD87298",
                "name": "worn", "class": "trade-in"
              },
              {
                "id": "CA170AAD-A5F6-42FF-B115-146FADD87298",
                "name": "no-damaged", "class": "trade-in"
              }
            ]
}
```

```
-- Return the tag name for trade-in tags embedded in the document
SELECT p.id,
    p.name,
    t.name AS tag
FROM products p
JOIN (SELECT VALUE t
      FROM t IN p.tags
      WHERE t.class = 'trade-in') AS t
```

```
// JSON returned by this query for the document on the left.
[
    {
        "id": "80D3630F-B661-4FD6-A296-CD03BB7A4A0C",
        "name": "Classic Vest, L",
        "tag": "worn"
    },
    {
        "id": "80D3630F-B661-4FD6-A296-CD03BB7A4A0C",
        "name": "Classic Vest, L",
        "tag": "no-damaged"
    }
]
```

쿼리에서 변수 구현

QueryDefinition 클래스를 사용하여 쿼리 매개 변수를 추가할 수 있습니다.

```
// Suppose we want to run the query below from the SDK, but we would like to send the condition values as parameters
// SELECT p.name, t.name AS tag FROM products p JOIN t IN p.tags WHERE p.price > 500 AND p.price <= 1000.

// Let's define a class for our products.
public class Product
{
    public string id { get; set; }
    public string name { get; set; }
    public string price { get; set; }
}

// Let's create our query definition.
string sql = "SELECT p.name, t.name AS tag FROM products p JOIN t IN p.tags WHERE p.price >= @lower AND p.price <= @upper"

QueryDefinition query = new (sql)
    .WithParameter("@lower", 500)
    .WithParameter("@upper", 1000);

// Let's finally execute our query. We will run it inside a foreach loop in case we get multiple documents back

await foreach (Product product in container.GetItemQueryIterator<Product>(query))
{
    Console.WriteLine($"[{product.id}]\t{product.name,35}\t{product.price,15:C}");
}
```

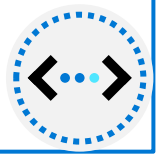
쿼리 결과 페이지 매김

Microsoft. Azure. Cosmos. Container 클래스는 여러 결과 페이지를 반복하기 위해 비동기 스트림을 지원합니다.

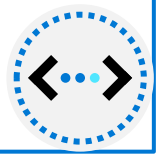
```
// Suppose we want to run the query below from the SDK, but we would like to return 100 documents at a time
// SELECT * FROM products p WHERE p.price > 500.

// Let's define a class for our products.
public class Product
{
    public string id { get; set; }
    public string name { get; set; }
    public string price { get; set; }
}

// Let's create our query definition.
string sql = "SELECT * FROM products WHERE p.price > 500";

QueryDefinition query = new (sql)
QueryRequestOptions options = new() { MaxItemCount = 100 };

// Let's finally execute our query. We will run it inside a foreach and a while loop in case we get multiple documents back
FeedIterator<Product> iterator = container.GetItemQueryIterator<Product>(query, requestOptions: options);

while(iterator.HasMoreResults)
{
    foreach(Product product in await iterator.ReadNextAsync())
    { // Handle individual items }
}
```

랩 - Azure Cosmos DB for NoSQL SDK를 사용하여 교차곱 쿼리 결과에 페이지를 매김

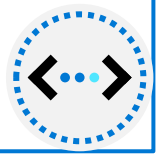
개발 환경 준비



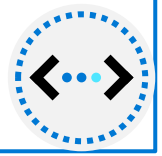
Azure Cosmos DB for NoSQL 계정 만들기



데이터를 사용하여 Azure Cosmos DB for NoSQL 계정 시드



SDK를 사용하여 SQL 쿼리의 작은 결과 집합을 통해 페이지 매김



검토

쿼리 결과를 각 항목에 대한 특정 필드 값의 배열로 평면화하는 데 사용되는 SQL 키워드는 무엇인가요?

- ❑ FROM.
- ❑ DISTINCT.
- ☑ VALUE.

제품이라는 컨테이너가 있습니다.쿼리에서 *FROM* 키워드 뒤에 데이터 원본의 이름은 무엇을 따라 지정해야 하나요?

- ❑ p.
- ❑ product.
- ☑ 위 항목 모두.

Microsoft.Azure.Cosmos.Container 클래스에서 SQL 쿼리를 문자열 매개 변수로 사용하고, 쿼리 결과를 deserialize된 C# 개체로 반복하는 데 사용할 수 있는 반복기를 반환하는 메서드는 무엇인가요?

- ❑ GetItemQueryStreamIterator<>
- ☑ GetItemQueryIterator<>
- ❑ GetItemLinqQueryable<>