

한국 마이크로소프트

# Microsoft Technical Trainer

Enterprise Skills Initiative

**Microsoft Azure**

## **Azure Table Storage**

이 문서는 Microsoft Technical Trainer팀에서 ESI 교육 참석자분들에게 제공해 드리는 문서입니다.

## 요약

이 내용들은 표시된 날짜에 Microsoft에서 검토된 내용을 바탕으로 하고 있습니다. 따라서, 표기된 날짜 이후에 시장의 요구사항에 따라 달라질 수 있습니다. 이 문서는 고객에 대한 표기된 날짜 이후에 변화가 없다는 것을 보증하지 않습니다.

이 문서는 정보 제공을 목적으로 하며 어떠한 보증을 하지는 않습니다.

저작권에 관련된 법률을 준수하는 것은 고객의 역할이며, 이 문서를 마이크로소프트의 사전 동의 없이 어떤 형태(전자 문서, 물리적인 형태 막론하고) 어떠한 목적으로 재 생산, 저장 및 다시 전달하는 것은 허용되지 않습니다.

마이크로소프트는 이 문서에 들어있는 특허권, 상표, 저작권, 지적 재산권을 가집니다. 문서를 통해 명시적으로 허가된 경우가 아니면, 어떠한 경우에도 특허권, 상표, 저작권 및 지적 재산권은 다른 사용자에게 허여되지 않습니다.

© 2022 Microsoft Corporation All right reserved.

Microsoft®는 미합중국 및 여러 나라에 등록된 상표입니다.

이 문서에 기재된 실제 회사 이름 및 제품 이름은 각 소유자의 상표일 수 있습니다.

## 문서 작성 연혁

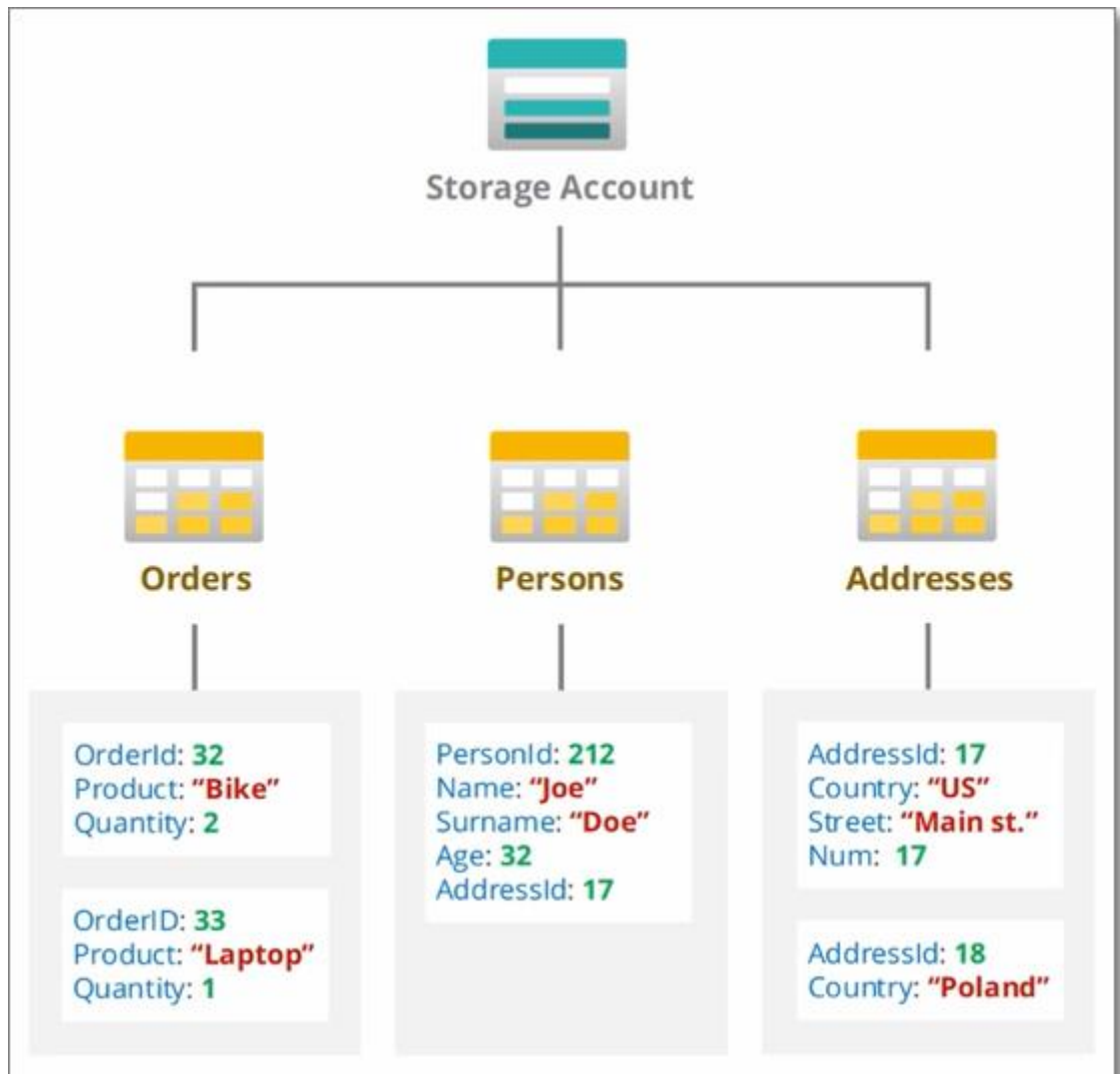
날짜	버전	작성자	변경 내용
2022.02.22	0.3.0	우진환	TASK 01 작성
2022.02.23	0.9.0	우진환	TASK 02 ~ TASK 04 작성
2022.02.27	1.0.0	우진환	TASK 05 작성
2022.04.03	1.1.0	우진환	Azure 포털 UI 변경 적용

## 목차

TABLE STORAGE 사용 시기.....	6
COSMOS DB TABLE API와 AZURE TABLE STORAGE 비교 .....	7
TASK 01. AZURE TABLE STORAGE 만들기 .....	8
TASK 02. TABLE 데이터 관리 및 쿼리.....	9
TASK 03. LOGIC APPS으로 TABLE 스토리지 작업 .....	13
TASK 04. POWERSHELL로 TABLE 스토리지 작업 .....	19
TASK 05. .NET으로 TABLE 스토리지 작업 .....	23
TASK 06. 리소스 정리.....	33

Azure Table Storage는 스토리지 계정의 하위 서비스이며 Azure에서 제공되는 NoSQL 데이터베이스입니다. 즉 스키마가 없기 때문에 키/값 속성만 설정하면 되며 매우 많은 양의 데이터를 처리할 수 있습니다.

Table Storage는 스토리지 계정의 일부이기 때문에 스토리지 계정이 제공하는 geo-replication, 높은 확장성, 저렴한 가격을 그대로 제공합니다.



Azure Table Storage의 용량, 확장성 및 성능 목표는 다음과 같습니다.

리소스	대상
Azure Storage 계정에서 테이블의 수	스토리지 계정의 용량에 의해서만 제한됨
테이블에 있는 파티션 수	스토리지 계정의 용량에 의해서만 제한됨
파티션의 엔터티 수	스토리지 계정의 용량에 의해서만 제한됨
단일 테이블의 최대 크기	500TiB
모든 속성 값을 비롯한 단일 엔터티의 최대 크기	1MiB
테이블 엔터티의 최대 속성 수	255 개(세 가지 시스템 속성 PartitionKey, RowKey 및 Timestamp 포함)
엔터티에 있는 개별 속성의 최대 전체 크기	속성 유형에 따라 다름

PartitionKey	최대 1KiB의 문자열
RowKey	최대 1KiB의 문자열
엔터티 그룹 트랜잭션의 크기	<ul style="list-style-type: none"> <li>한 개 트랜잭션에는 최대 100 개의 엔터티가 포함될 수 있고, 페이로드 크기는 4MiB 미만이어야 함</li> <li>엔터티 그룹 트랜잭션은 엔터티 업데이트를 한 번만 포함할 수 있음</li> </ul>
테이블별로 저장된 최대 액세스 정책 수	5
스토리지 계정당 최대 요청 속도	초당 20,000 개 트랜잭션(1KiB 엔터티 크기로 가정)
단일 테이블 파티션의 목표 처리량(1KiB 엔터티)	초당 최대 2,000 개 엔터티

- Table은 스토리지 노드간 부하 분산을 지원하기 위해 분할(partitioned)됩니다. 테이블의 엔터티는 파티션으로 구성됩니다. 파티션은 동일한 파티션 값을 가지고 있는 연속적인 엔터티의 범위입니다. 파티션 키는 주어진 테이블 내의 파티션에 대한 고유한 식별자로 **PartitionKey** 속성으로 지정됩니다. 파티션 키는 엔터티의 기본 키(primary key)의 첫 번째 부분을 형성합니다. 따라서 모든 **insert**, **update**, **delete** 작업에 **PartitionKey** 속성을 포함해야 합니다.
- 기본 키(primary key)를 구성하는 두 번째 부분은 **RowKey** 속성에 의해 지정되는 행 키(row key)입니다. **row key**는 지정된 파티션 내의 엔터티에 대한 고유한 식별자입니다. **PartitionKey**와 **RowKey**를 함께 사용하여 테이블 내의 모든 엔터티를 고유하게 식별할 수 있습니다. 따라서 모든 **insert**, **update**, **delete** 작업에 **RowKey** 속성을 포함해야 합니다.
- Timestamp** 속성은 **DateTime** 값이며 엔터티가 마지막으로 수정된 시간을 기록하기 위해 서버 측에서 유지 관리됩니다. 테이블 서비스는 낙관적 동시성(optimistic concurrency)을 제공하기 위해 내부적으로 **Timestamp** 속성을 사용합니다. 엔터티의 **Timestamp** 속성의 값은 엔터티가 수정될 때마다 증가합니다. 이 속성은 **insert**나 **update**에서 설정하면 안 되며 값은 무시됩니다.

## Table Storage 사용 시기

Table Storage를 사용하는 시나리오가 많지만 다음과 같은 경우 Table Storage의 사용을 고려할 수 있습니다.

- TB 단위의 구조화된 데이터 저장. Table Storage는 스키마가 없지만 열과 열의 유형을 지정할 수 있기 때문에 구조를 포함할 수 있습니다.
- 데이터를 비정규화(denormalization)할 수 있는 경우
- 복잡한 **join**이나 스키마가 필요하지 않지 않는 경우
- 클러스터된 인덱스(clustered index)로 빠른 쿼리가 필요할 때, **PartitionKey**로 빠른 쿼리를 수행할 수 있습니다.
- OData 쿼리 지원 및 JSON serializable 데이터 지원

따라서 다음과 같은 일반적인 시나리오에서 Table Storage를 사용할 수 있습니다.

- serverless: serverless 애플리케이션은 매우 빠르게 확장되고 축소되기 때문에 Table Storage가 이를 적절히 처리할 수 있습니다.

- 웹 애플리케이션: 웹 애플리케이션의 데이터나 메타데이터를 저장하는데 Table Storage를 사용할 수 있습니다.
- 단순 로깅: Azure에 로깅을 위해 더 적합한 서비스가 있지만 매우 간단한 프로세스 로깅이나 비즈니스 로직 로깅에 Table Storage를 사용할 수 있습니다.
- 메타데이터 및 구성 저장소: 가격이 저렴하고 확장성이 뛰어나기 때문에 메타데이터 및 구성을 저장하는데 적합합니다.

## Cosmos DB Table API와 Azure Table Storage 비교

Cosmos DB Table API와 Azure Table Storage는 동일한 테이블 데이터 모델을 공유하고 SDK를 통해 동일한 `create`, `delete`, `update`, `query` 작업을 노출합니다. Cosmos DB Table API와 Table Storage는 대기 시간, 처리량 및 글로벌 분산의 관점에서 크게 비교해볼 수 있습니다.

기능	Azure Table Storage	Azure Cosmos DB 테이블 API
대기 시간	빠르지만 대기 시간에 대한 상한이 없음	읽기/쓰기에 대한 1 자리 밀리초 대기 시간이 지원(전 세계의 모든 규모에 대해 99 번째 백분위수에서 <10ms 미만의 읽기 및 <15ms 미만의 쓰기 대기 시간 지원)
처리량	가변 처리량 모델임. 테이블의 확장 제한은 20,000 개 operation/s	<ul style="list-style-type: none"> <li>▪ SLA를 통해 지원하는 테이블당 예약된 전용 처리량으로 확장성이 뛰어남</li> <li>▪ 계정에는 처리량에 대한 상한이 없으며, 테이블당 &gt;1,000 만 개 이상 operation/s 을 지원(프로비저닝 처리량 모드).</li> </ul>
글로벌 분산	자동 및 수동 계정 failover를 지원하는고가용성을 위해 읽기 가능한 보조 읽기 지역이 하나 지원되는 단일 지역	<ul style="list-style-type: none"> <li>▪ 1 ~ 30 개가 넘는 지역까지 터키 전역 배포를 수행할 수 있음</li> <li>▪ 전 세계 어디에서나 자동 및 수동 장애 조치를 지원</li> </ul>
인덱싱	PartitionKey 및 RowKey에 대한 기본 인덱스만 제공. 보조 인덱스가 없음	모든 속성에 대해 자동 및 전체 인덱싱을 수행할 수 있으며, 인덱스를 관리할 필요가 없음
쿼리	쿼리 실행 시 기본 키에 대한 인덱스를 사용하고 그렇지 않은 경우 스캔	쿼리는 빠른 쿼리 시간을 위해 속성에 대해 자동 인덱싱을 활용할 수 있음
일관성	주 지역 내에서 강력하게 유지되며, 최종적으로는 보조 지역 내에서 유지됨	애플리케이션 요구 사항에 따라 가용성, 대기 시간, 처리량 및 일관성을 절충할 수 있는 잘 정의된 5 가지 일관성 수준이 적용
가격 책정	소비 기반	소비 기반과 프로비저닝된 용량 모드에서 모두 사용할 수 있음
SLA	99.99%	모든 단일 지역 계정 및 모든 다중 지역 계정에는 99.99% 가용성 SLA와 완료된 일관성이 제공되고, 일반 공급 시에는 모든 다중 지역 데이터베이스 계정에 업계 최고 수준의 포괄적인 SLA와 99.999% 읽기 가용성이 제공



## TASK 01. Azure Table Storage 만들기

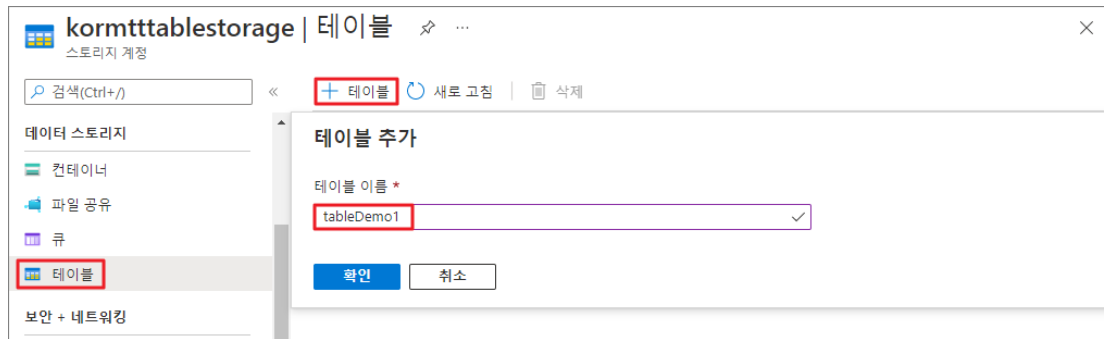
1. Azure 포털에서 [리소스 만들기]를 클릭한 후 "스토리지 계정"을 검색하고 클릭합니다. [스토리지 계정] 블레이드에서 [만들기]를 클릭합니다.



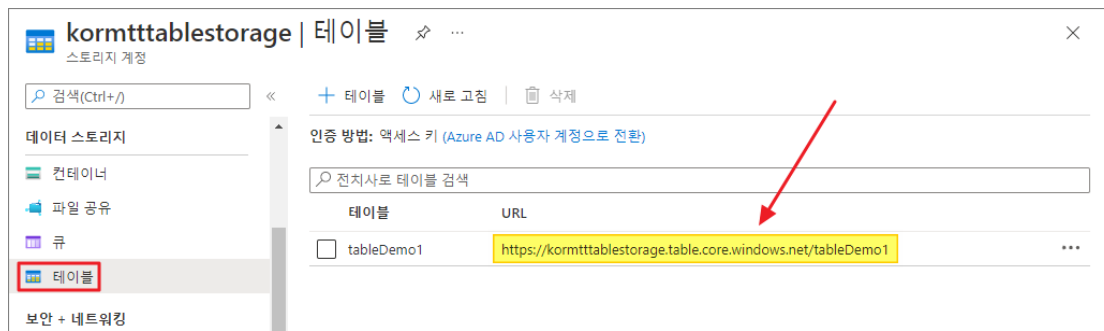
2. [저장소 계정 만들기] 블레이드의 [기본] 탭에서 아래와 같이 구성한 후 [검토 + 만들기]를 클릭합니다. [검토 + 만들기] 탭에서 [만들기]를 클릭합니다.
  - [프로젝트 정보 - 리소스 그룹]: "새로 만들기"를 클릭한 후 "04\_tableStorageRg"를 입력합니다.
  - [인스턴스 정보 - 스토리지 계정 이름]: 중복되지 않는 고유한 이름을 입력합니다.
  - [인스턴스 정보 - 지역]: (Asia Pacific) Korea Central
  - [인스턴스 정보 - 성능]: 표준
  - [인스턴스 정보 - 중복]: LRS(로컬 중복 스토리지)

## TASK 02. Table 데이터 관리 및 쿼리

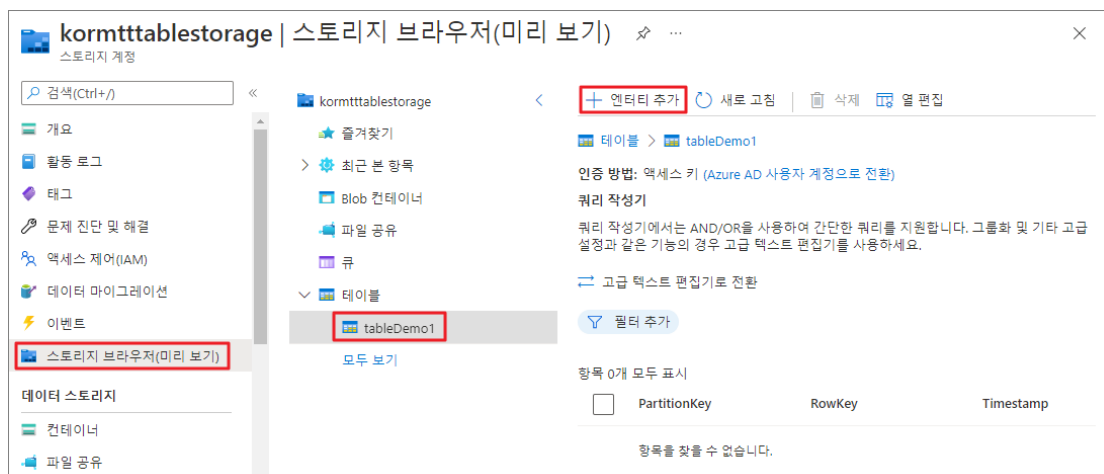
1. 새로 만든 [스토리지 계정] 블레이드로 이동합니다. 새 테이블을 만들기 위해 [데이터 스토리지 - 테이블]로 이동한 후 [테이블]을 클릭합니다. [테이블 추가]에서 "tableDemo"을 입력한 후 [확인]을 클릭합니다.



2. 새 테이블이 생성되면 스토리지 계정의 테이블에 대한 URL이 생성되는 것을 확인할 수 있습니다. 즉 이 URL은 공용 URL이기 때문에 특수 문자를 사용할 수 없습니다. 새로 만든 테이블은 Azure 포털의 기본 블레이드에서 직접 열어 확인할 수 없습니다.



3. 테이블의 데이터를 관리하기 위해서는 로컬에 Storage Explorer를 설치하거나 Azure 포털의 스토리지 브라우저를 사용해야 합니다. [스토리지 계정]의 [스토리지 브라우저]를 클릭합니다. 아래와 같이 새로 만든 테이블과 테이블에서 수행할 수 있는 작업이 메뉴에 표시되는 것을 확인할 수 있습니다. 새 엔터티를 추가하기 위해 [엔터티 추가]를 클릭합니다.



4. [엔터티 추가]에서 PartitionKey와 RowKey가 표시되는 것을 확인할 수 있습니다. Timestamp는 자동으로 추가되기 때문에 따로 표시되지 않습니다. PartitionKey와 RowKey는 어떤 값으로 지정해도

됩니다. 예를 들어 **PartitionKey**는 성 **RowKey**는 이름으로 사용할 수 있습니다. 또한 Table Storage는 schema-less NoSQL이므로 고유한 속성을 추가할 수 있습니다. 다음과 같이 설정한 후 [삽입]을 클릭합니다.

- PartitionKey: Simpson
- RowKey: Homer
- [속성 추가]를 클릭한 후 속성 이름은 **Age**, 형식은 **Int32**, 값은 **45**를 입력합니다.

속성 이름	형식	값
PartitionKey	String	Simpson
RowKey	String	Homer
Age	Int32	45

속성 추가

- 아래와 같이 추가된 엔터티를 확인할 수 있습니다. 이 엔터티는 **PartitionKey**, **RowKey**, **Timestamp**, **Age**가 추가된 것을 확인할 수 있습니다. 다시 [엔터티 추가]를 클릭합니다.

항목 1개 모두 표시	PartitionKey	RowKey	Timestamp	Age
<input type="checkbox"/>	Simpson	Homer	2022-04-03T01:54:33.08...	45

- [엔터티 추가]에서 앞서 추가했던 **Age** 속성이 표시되는 것을 확인할 수 있습니다. 하지만 이 속성은 불필요한 경우 삭제할 수 있으며 새 속성을 추가할 수도 있습니다. 다음과 같이 입력한 후 [삽입]을 클릭합니다.

- PartitionKey: Simpson
- RowKey: Marge
- Age: 42
- [속성 추가]를 클릭하고 속성 이름은 **Country**, 형식은 **String**, 값은 **US**를 입력합니다.

Z

### 엔터티 추가

속성 이름	형식	값
PartitionKey	String	Simpson
RowKey	String	Marge
Age	Int32	42
Country	String	US

속성 추가

7. 추가된 엔터티를 확인하면 두 번째 추가한 엔터티의 경우 **Country**에 값이 있지만 첫 번째로 추가한 엔터티에는 이 열의 값이 비어 있는 것을 확인할 수 있습니다. 즉 테이블은 스키마를 정의할 수 없기 때문에 각 엔터티에 항상 동일한 열이 있다고 보장할 수 없습니다.

- 클러스터된 키(clustered key)와 같이 결합된 키를 만들 수도 있습니다. 예를 들어 성, 이름, 국가를 결합하여 이 값을 **PartitionKey**로 사용할 수도 있습니다. 그런 다음 특정 값이 아닌 전체 내용을 포함하고 있는 이 키로 쿼리를 실행할 수도 있습니다.
- PartitionKey**와 **RowKey**는 인덱싱되며 이를 제외한 다른 모든 속성에 대해서는 전체 스캔(full scan)이 실행됩니다. 따라서 **PartitionKey**와 **RowKey**를 잘 디자인하는 것이 중요합니다.

kormttablestorage < + 엔터티 추가 새로 고침 삭제 열 편집

즐거찾기 > 최근 본 항목 Blob 컨테이너 파일 공유 큐 > 테이블 > tableDemo1 모두 보기

테이블 > tableDemo1  
인증 방법: 액세스 키 (Azure AD 사용자 계정으로 전환)  
쿼리 작성기  
쿼리 작성기에서는 AND/OR을 사용하여 간단한 쿼리를 지원합니다. 그룹화 및 기타 고급 설정과 같은 기능의 경우 고급 텍스트 편집기를 사용하세요.  
고급 텍스트 편집기로 전환

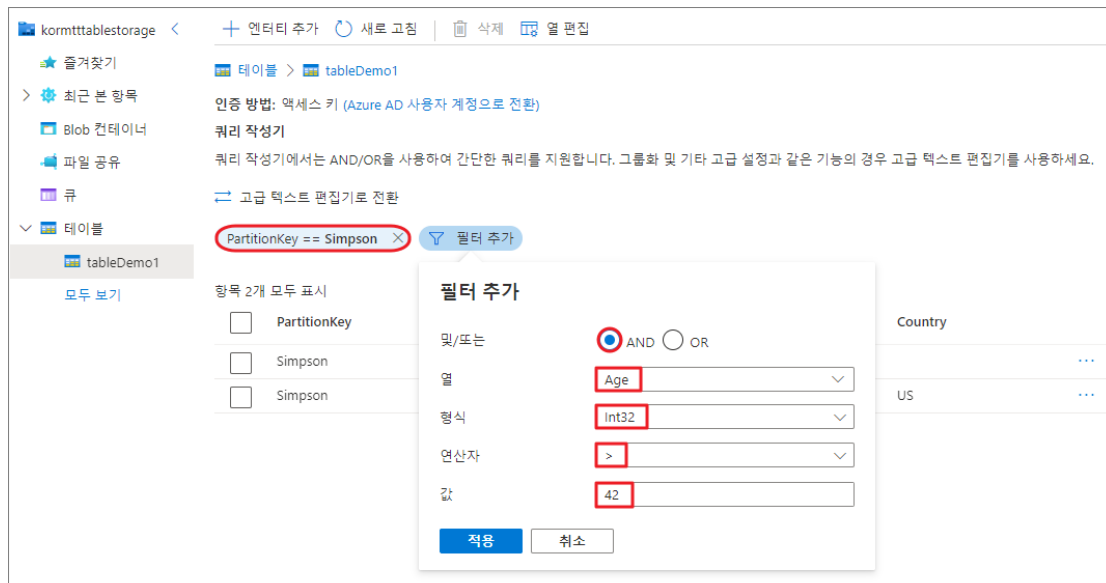
필터 추가

항목 2개 모두 표시

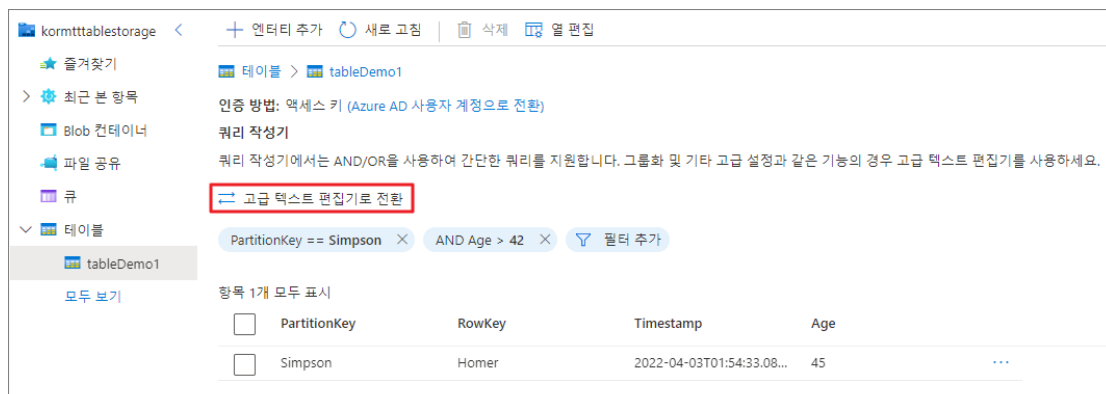
	PartitionKey	RowKey	Timestamp	Age	Country	
<input type="checkbox"/>	Simpson	Homer	2022-04-03T01:54:33.08...	45		...
<input type="checkbox"/>	Simpson	Marge	2022-04-03T01:55:24.64...	42	US	...

8. [스토리지 브라우저]의 테이블 화면에서 [필터 추가]를 클릭한 후 다음과 같은 필터를 추가하고 [적용]을 클릭합니다.

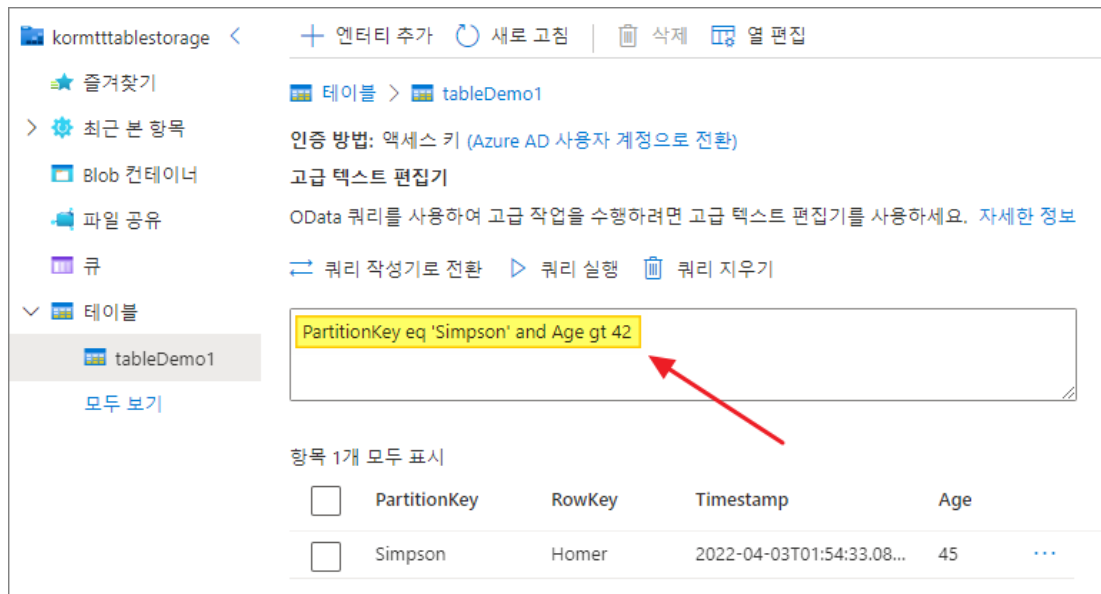
및/또는	열	형식	연산자	값
	PartitionKey	String	==	Simpson
AND	Age	Int32	>	42



9. 다음과 같이 필터에 의해 조건을 만족하는 엔터티만 표시되는 것을 확인할 수 있습니다. 이러한 쿼리에서 새 절을 추가하거나 제거할 수 있으며 속성 형식에 따라 여러 연산자를 선택하여 쿼리를 수행할 수 있습니다. 즉 **PartitionKey**와 **RowKey**가 쿼리하려는 데이터에 대해 잘 디자인되어 있는 경우 데이터를 매우 효과적으로 쿼리할 수 있습니다. [스토리지 브라우저]의 테이블 선택 화면에서 [고급 텍스트 편집기로 전환]을 클릭합니다. [텍스트 편집기로 전환] 창이 표시되면 [전환]을 클릭합니다.

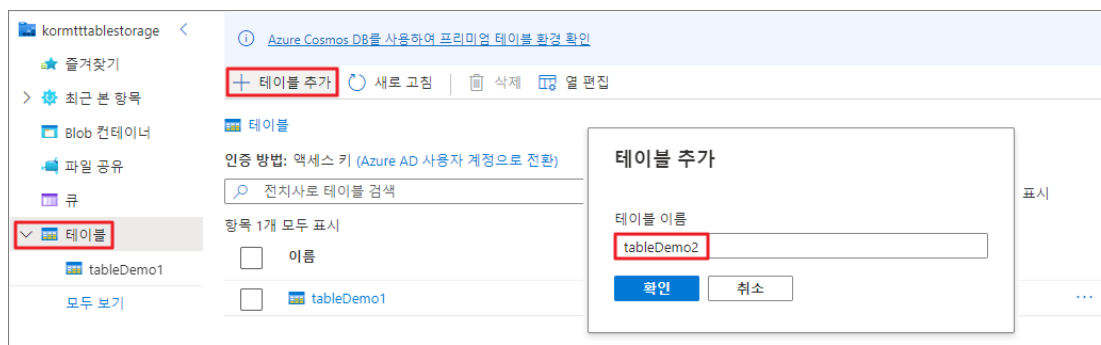


10. 필터 추가를 통해 구성했던 쿼리가 다음과 같이 쿼리 언어로 표시되는 것을 확인할 수 있습니다.

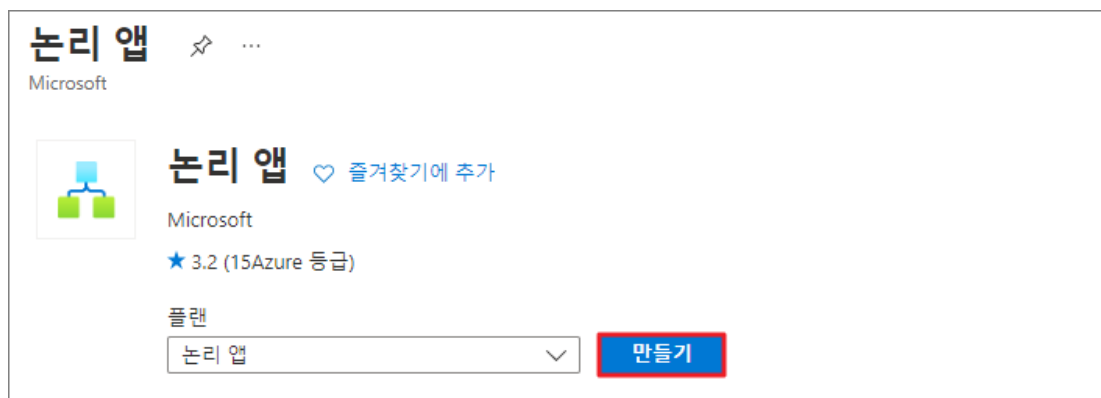


### TASK 03. Logic Apps으로 Table 스토리지 작업

1. [스토리지 계정] 블레이드의 [스토리지 브라우저]에서 좌측 메뉴의 [테이블]을 선택합니다. 메뉴에서 [테이블 추가]를 클릭합니다. [테이블 추가]에서 "tableDemo2"를 입력하고 [확인]을 클릭합니다.



2. Azure 포털에서 [리소스 만들기]를 클릭한 후 "논리 앱"을 검색하고 클릭합니다. [논리 앱] 블레이드에서 [만들기]를 클릭합니다.



3. [논리 앱 만들기] 블레이드의 [기본] 탭에서 아래와 같이 구성한 후 [검토 + 만들기]를 클릭합니다. [검토 +

만들기] 탭에서 [만들기]를 클릭합니다.

- [프로젝트 세부 정보 - 리소스 그룹]: 04\_tableStorageRg
- [인스턴스 정보 - 논리 앱 이름]: tableLogicApp
- [인스턴스 정보 - 지역]: Korea Central
- [인스턴스 정보 - Log analytics 사용 설정]: 아니요
- [플랜 - 플랜 유형]: 소비

**논리 앱 만들기** ...

기본   태그   검토 + 만들기

손쉬운 리소스 관리, 배포 및 공유를 위해 논리적 단위로 워크플로를 그룹화할 수 있는 논리 앱을 만듭니다. 워크플로를 사용하면 업무상 중요한 앱 및 서비스를 Azure Logic Apps에 연결하고 코드를 한 줄도 작성하지 않고 워크플로를 자동화할 수 있습니다.

**프로젝트 세부 정보**

배포된 리소스와 비용을 관리할 구독을 선택합니다. 폴더 같은 리소스 그룹을 사용하여 모든 리소스를 정리 및 관리합니다.

구독 \* ① Azure Pass - 스폰서십 ▼

리소스 그룹 \* ① 04\_tableStorageRg ▼  
[새로 만들기](#)

**인스턴스 정보**

논리 앱 이름 \* tableLogicApp ▼

지역 \* Korea Central ▼

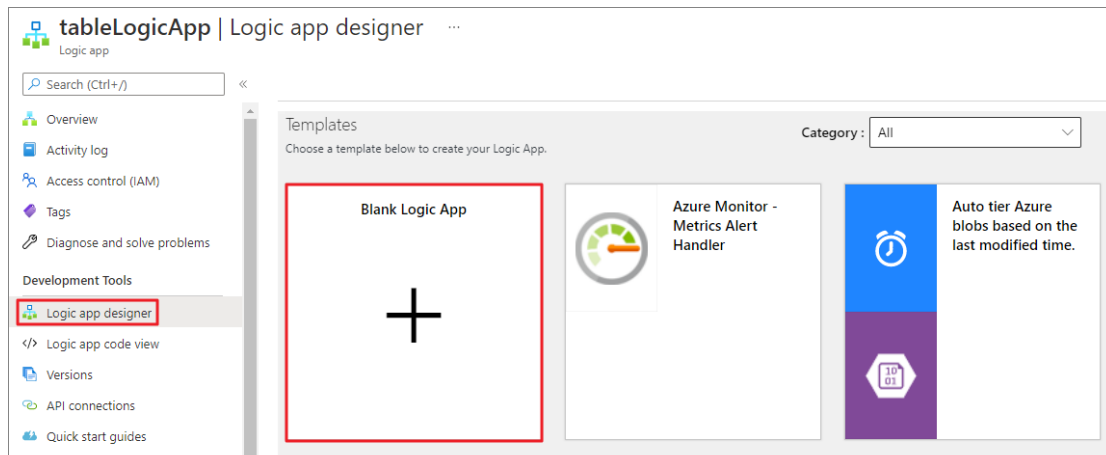
Log analytics 사용 설정 \* ☐ 예 ☒ 아니요

**플랜**

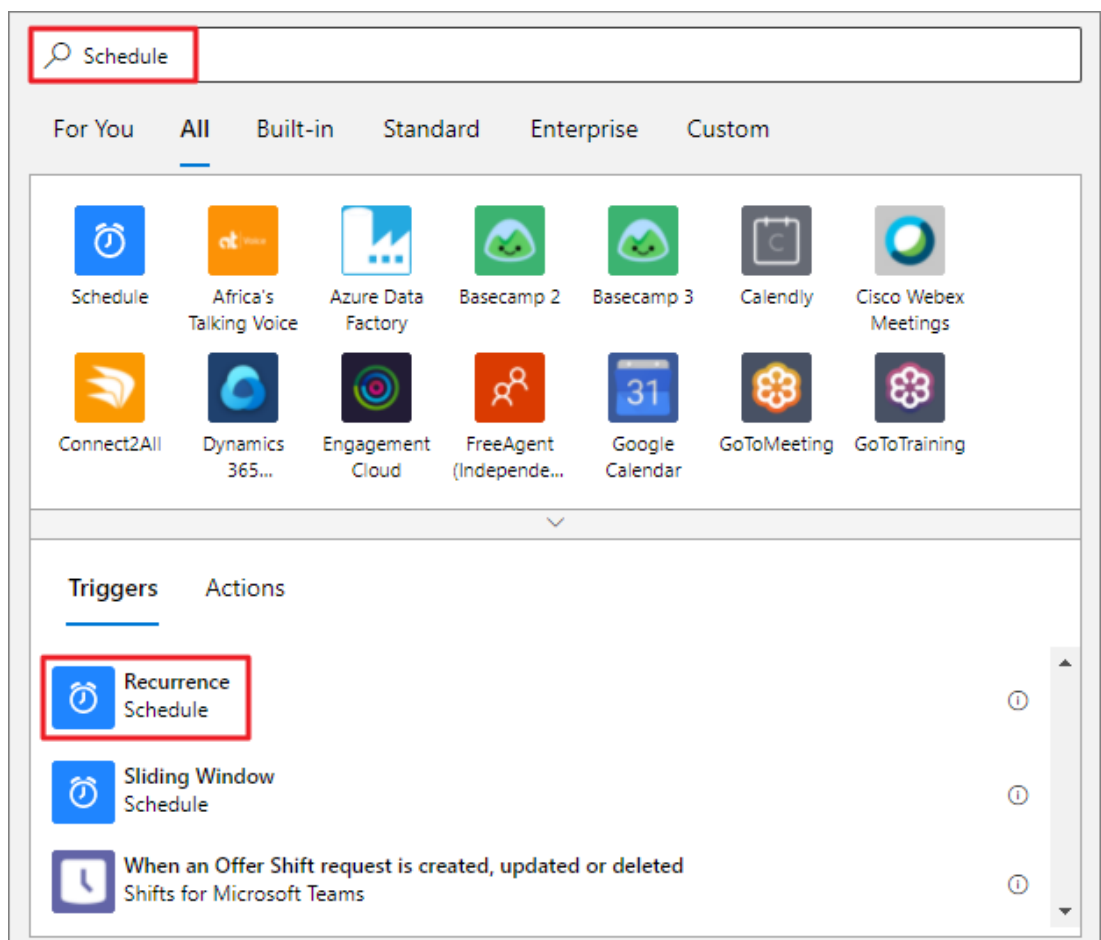
선택한 플랜 유형에 따라 앱 확장 방식, 사용하도록 설정된 기능 및 가격 책정 방식이 결정됩니다.

플랜 유형 \* ☒ 소비: 입문 수준에 가장 적합합니다. 워크플로가 실행되는 만큼만 비용을 지불하세요.  
☐ 기준: 이벤트 기반 확장 및 네트워킹 격리를 통해 엔터프라이즈 수준의 서버리스 애플리케이션에 가장 적합합니다.  
 ⓘ 클래식 소비 경험 만들기를 찾고 계십니까? [여기를 클릭](#)

4. 새로 만든 `tableLogicApp` Logic app] 블레이드의 [Development Tools - Logic app designer]로 이동한 후 [Blank Logic App] 타일을 클릭합니다.

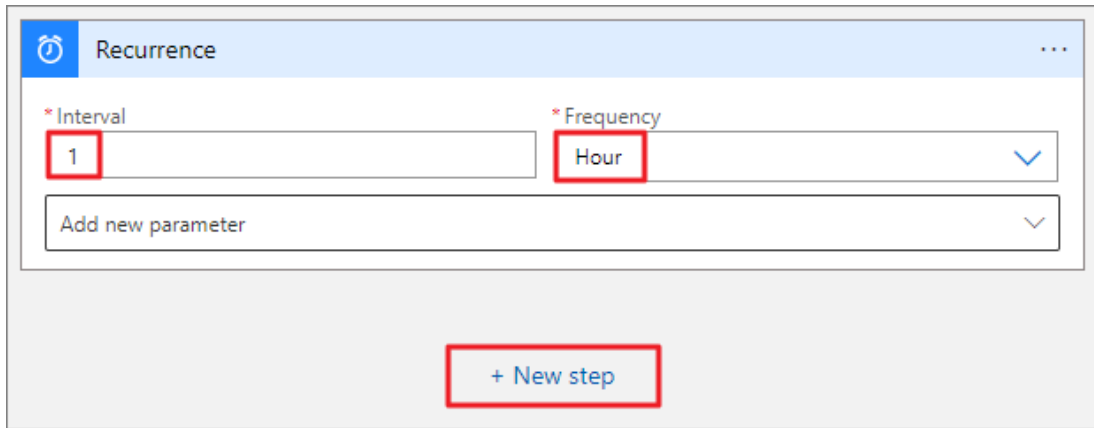


5. Logic App 디자이너의 검색창에 "Schedule"을 검색한 후 [Triggers] 탭에서 "Recurrence"를 클릭합니다.

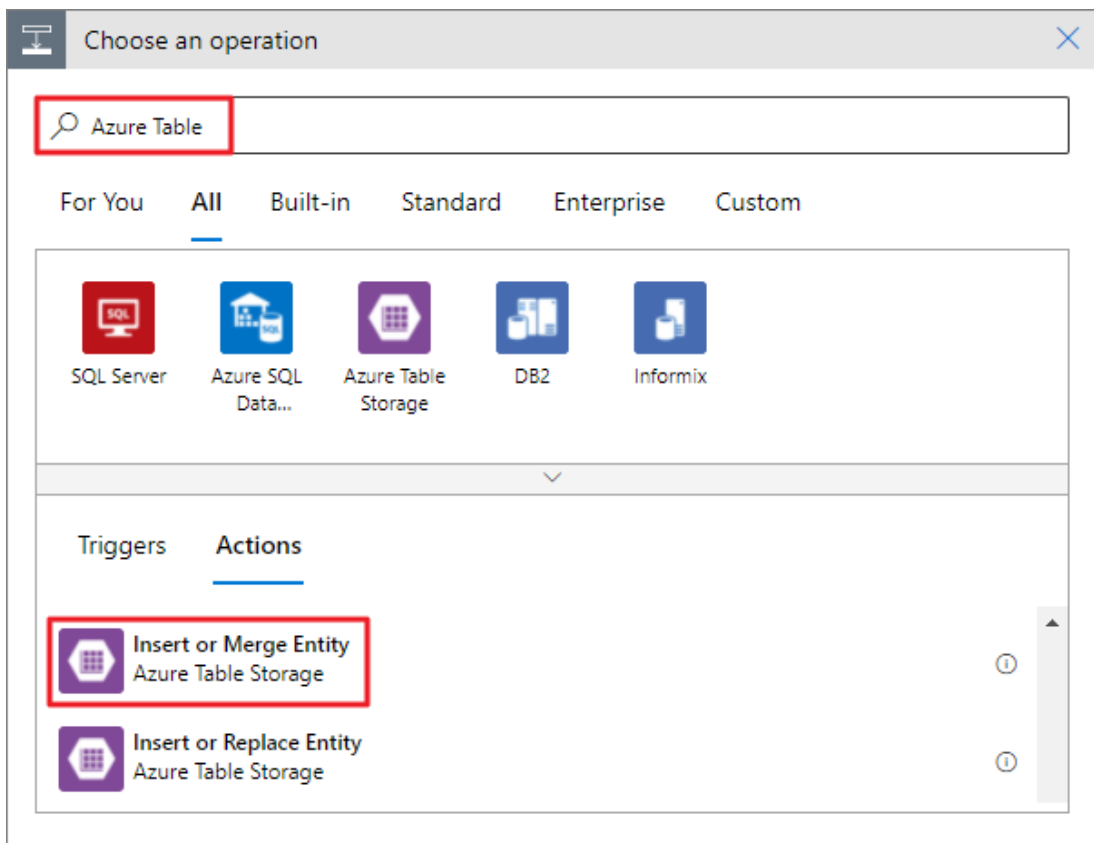


6. [Recurrence] 블록에서 1 시간 간격을 설정하고 [New step]을 클릭합니다.





7. [Choose an operation] 블록의 검색창에서 "Azure Table"을 검색합니다. [Actions] 탭에서 "Insert or Merge Entity"를 클릭합니다. 이 외에도 테이블 만들기, 삭제 등 Table Storage와 관련된 여러 작업을 수행할 수 있습니다.



8. [Insert or Merge Entity] 블록에서 아래와 같이 구성한 후 [Create]를 클릭합니다.
- Connection name: tableConnection
  - Storage Account: 앞서 만들었던 Table Storage를 포함하고 있는 스토리지 계정을 선택합니다.

**Insert or Merge Entity**

\* Connection name:

\* Storage Account

Name	Resource Group	Location
00labrgguestdiag	00_labRg	koreacentral
<b>kormtttablestorage</b>	04_tableStorageRg	koreacentral

**Create**

[Manually enter connection information](#)

9. [Insert or Merge Entity] 블록에서 아래와 같이 구성합니다. Logic App 디자이너 메뉴에서 [Save]를 클릭한 후 [Run Trigger - Run]을 클릭합니다.

- Table: tableDemo2
- Partition Key: User
- Row Key: 1
- Entity: {"Name": "Bart", "Surname": "Simpson"}

Save Discard Run Trigger Designer Code view Parameters Templates Connectors Help Info

Recurrence

**Insert or Merge Entity**

\* Table:

\* Partition Key:

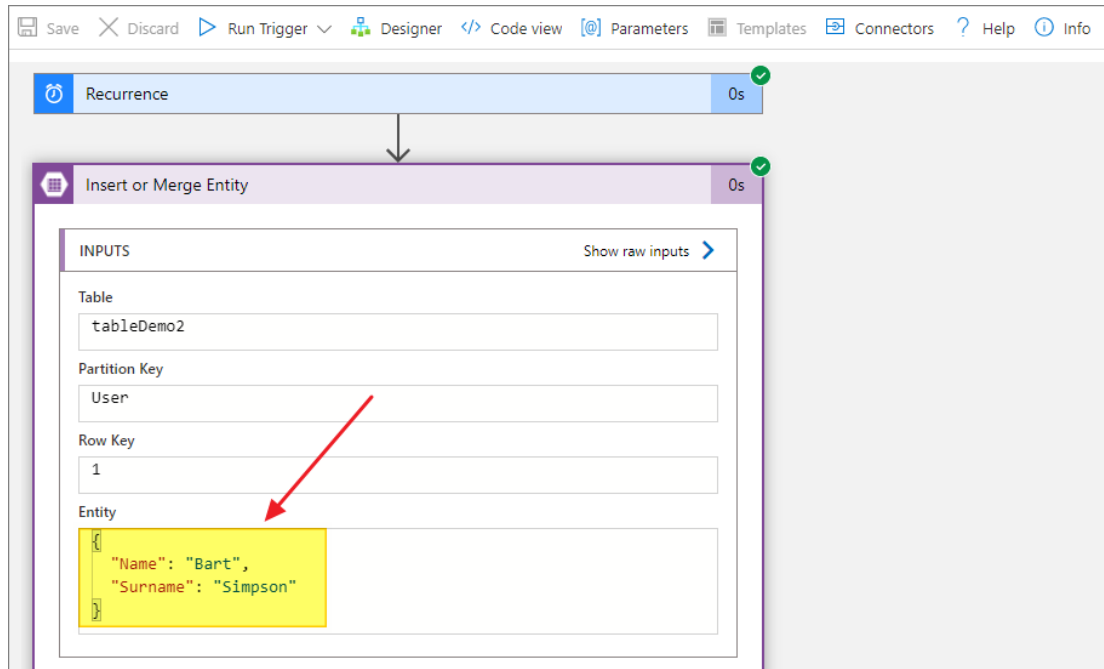
\* Row Key:

\* Entity:

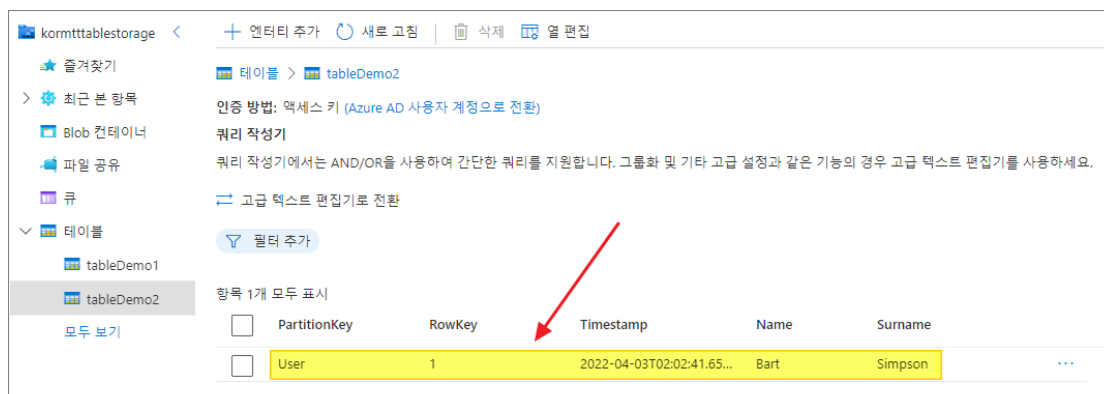
Add new parameter

Connected to tableConnection. [Change connection.](#)

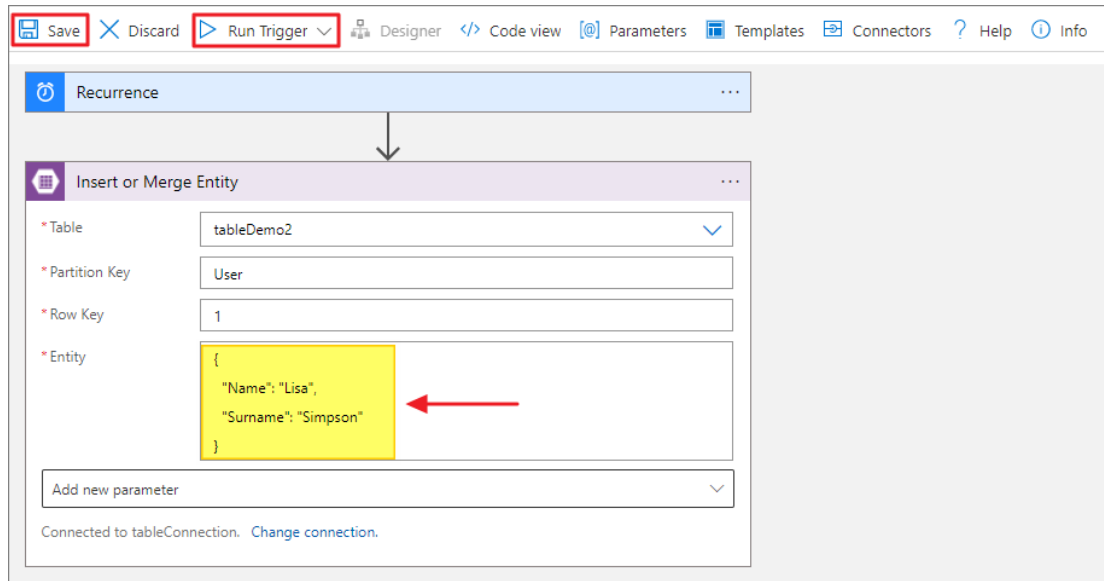
10. Logic App 디자이너에서 실행이 완료된 것을 확인합니다.



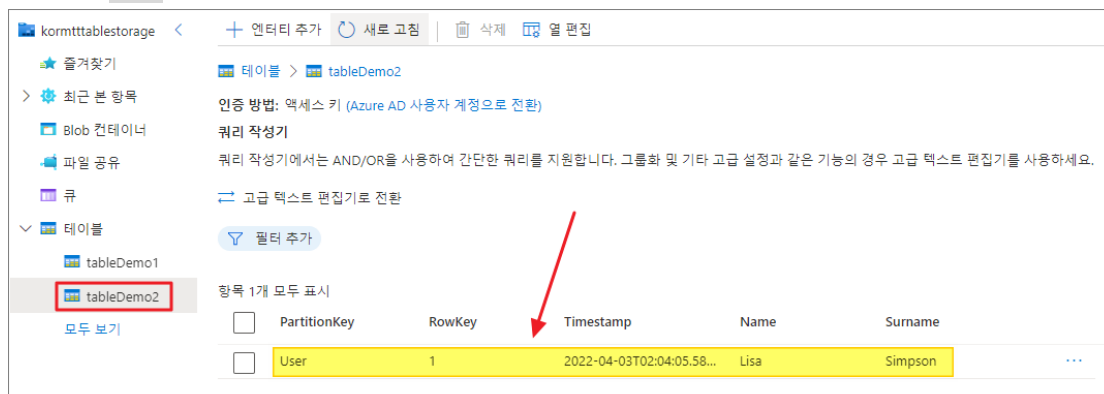
11. [스토리지 계정] 블레이드의 [스토리지 브라우저]로 이동한 후 [테이블 - tableDemo2]를 클릭합니다. 아래와 같이 Logic App을 통해 새 엔터티가 추가된 것을 확인합니다.



12. Table Storage에 삽입한 엔터티는 간단한 JSON 파일로 되어 있기 때문에 이 파일을 다시 업데이트하여 기존 엔터티를 수정할 수 있습니다. [tableLogicApp Logic app] 블레이드의 [Development Tools - Logic app designer]로 이동합니다. [Insert or Merge Entity] 블록을 확장한 후 JSON 파일의 **Name** 값을 **Lisa**로 변경합니다. Logic App 디자이너 메뉴에서 [Save]를 클릭한 후 [Run Trigger - Run]을 클릭합니다.

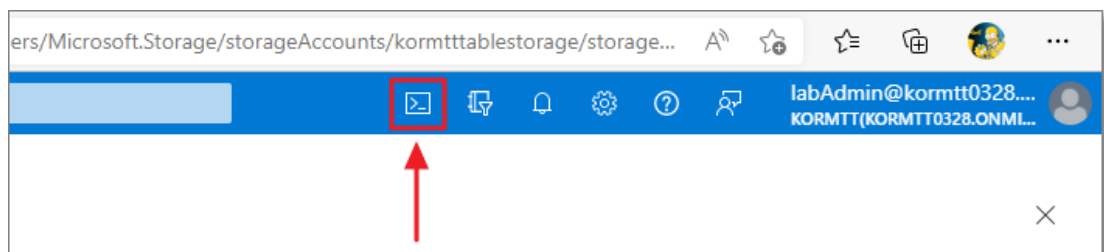


13. [스토리지 계정] 블레이드의 [스토리지 브라우저]로 이동한 후 [테이블 - tableDemo2]를 클릭합니다. 기존 엔터티의 **Name** 속성 값이 변경된 것을 확인합니다.

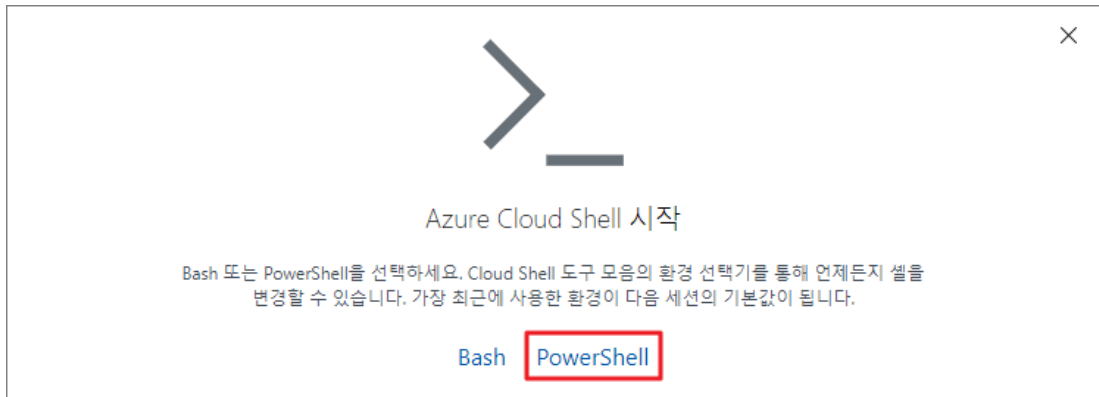


#### TASK 04. PowerShell로 Table 스토리지 작업

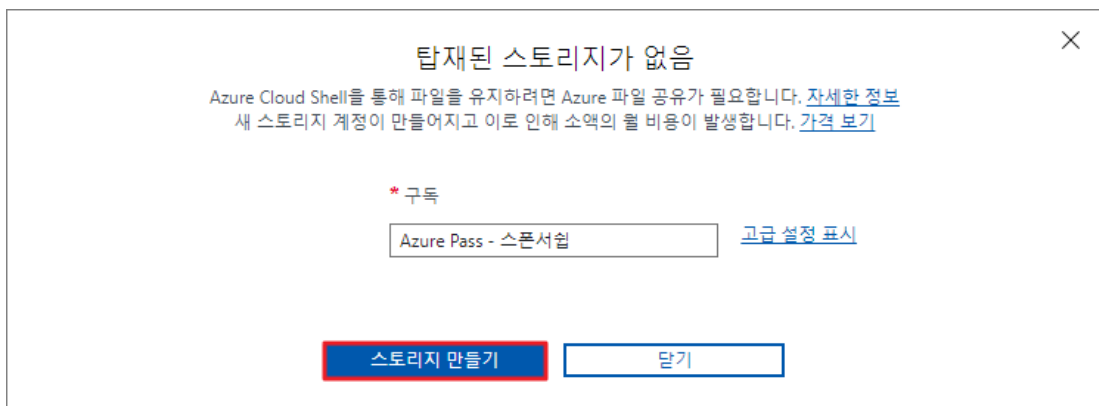
1. Azure 포털의 우측 상단에서 [Cloud Shell] 아이콘을 클릭합니다.



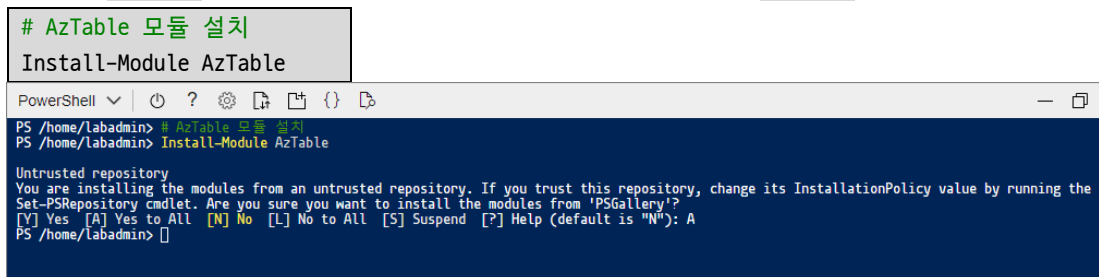
2. [Azure Cloud Shell 시작] 화면에서 [PowerShell]을 클릭합니다.



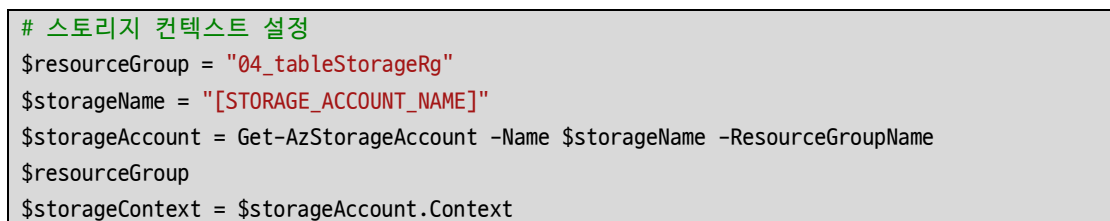
3. [탑재된 스토리지가 없음]에서 기본값을 유지하고 [스토리지 만들기]를 클릭합니다.



4. [Cloud Shell]에서 PowerShell을 실행합니다. PowerShell에서 Table Storage 관련 작업을 실행하기 위해서는 AzTable 모듈을 먼저 설치해야 합니다. 다음 명령을 실행하여 AzTable 모듈을 설치합니다.



5. 먼저 스토리지 계정에서 작업을 하기 위해 스토리지 계정의 컨텍스트를 가져와야 합니다. 다음 명령을 실행하여 스토리지 계정 컨텍스트를 변수로 설정합니다.



```
PowerShell
PS /home/labadmin> # 스토리지 컨텍스트 설정
PS /home/labadmin> $resourceGroup = "04_tableStorageRg"
PS /home/labadmin> $storageName = "kormtttablestorage"
PS /home/labadmin> $storageAccount = Get-AzStorageAccount -Name $storageName -ResourceGroupName $resourceGroup
PS /home/labadmin> $storageContext = $storageAccount.Context
PS /home/labadmin> 
```

6. 다음 명령을 실행하여 PowerShell을 통해 새 테이블을 생성합니다.

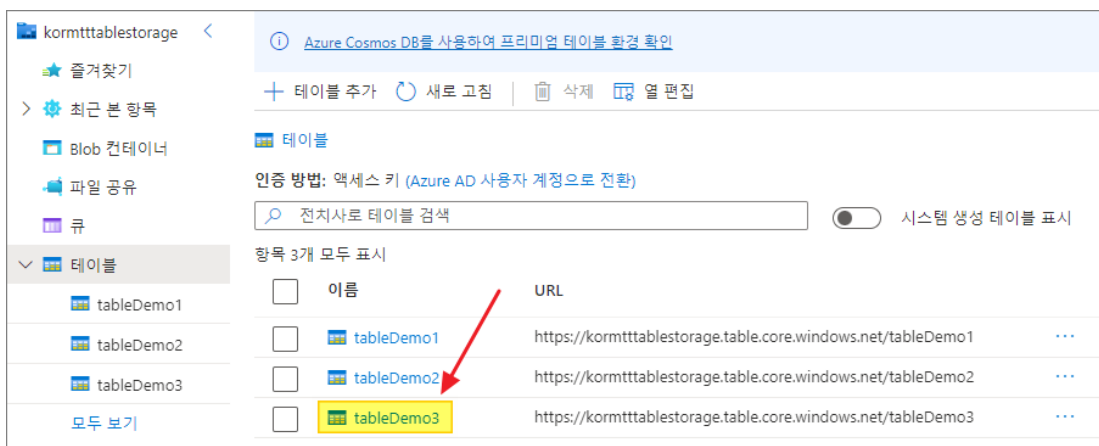
```
# 새 테이블 만들기
$tableName = "tableDemo3"
New-AzStorageTable -Name $tableName -Context $storageContext
```

```
PowerShell
PS /home/labadmin> # 새 테이블 만들기
PS /home/labadmin> $tableName = "tableDemo3"
PS /home/labadmin> New-AzStorageTable -Name $tableName -Context $storageContext

Table End Point: https://kormtttablestorage.table.core.windows.net/

Name      Uri
-----
tableDemo3 https://kormtttablestorage.table.core.windows.net/tableDemo3
PS /home/labadmin> 
```

7. Azure 포털의 [스토리지 계정] 블레이드로 이동한 후 [스토리지 브라우저]를 클릭합니다. [테이블]에서 아래와 같이 PowerShell로 만든 테이블이 표시되는 것을 확인합니다.



8. [Cloud Shell]에서 다음 명령을 실행하여 \$cloudTable 이름의 개체를 만듭니다.

```
# 새 테이블 개체 만들기
$cloudTable = (Get-AzStorageTable -Name $tableName -Context $storageContext).CloudTable
```

```
PowerShell
PS /home/labadmin> # 새 테이블 개체 만들기
PS /home/labadmin> $cloudTable = (Get-AzStorageTable -Name $tableName -Context $storageContext).CloudTable
PS /home/labadmin> 
```

9. [Cloud Shell]에서 다음 명령을 실행하여 PartitionKey를 2개 만들고 4개의 엔터티를 추가합니다.

```
# 테이블에 엔터티 추가
$partitionKey1 = "partition1"
$partitionKey2 = "partition2"

Add-AzTableRow -table $cloudTable -partitionKey $partitionKey1 `
```

```
-rowKey ("CA") -property @{"username"="Chris";"userid"=1} | Out-Null
Add-AzTableRow -table $cloudTable -partitionKey $partitionKey2 `
  -rowKey ("NM") -property @{"username"="Jessie";"userid"=2} | Out-Null
Add-AzTableRow -table $cloudTable -partitionKey $partitionKey1 `
  -rowKey ("WA") -property @{"username"="Christine";"userid"=3} | Out-Null
Add-AzTableRow -table $cloudTable -partitionKey $partitionKey2 `
  -rowKey ("TX") -property @{"username"="Steven";"userid"=4} | Out-Null
```

```
PowerShell
PS /home/labadmin> # 테이블에 엔터티 추가
PS /home/labadmin> $partitionKey1 = "partition1"
PS /home/labadmin> $partitionKey2 = "partition2"
PS /home/labadmin>
PS /home/labadmin> Add-AzTableRow -table $cloudTable -partitionKey $partitionKey1 `
>> -rowKey ("CA") -property @{"username"="Chris";"userid"=1} | Out-Null
PS /home/labadmin> Add-AzTableRow -table $cloudTable -partitionKey $partitionKey2 `
>> -rowKey ("NM") -property @{"username"="Jessie";"userid"=2} | Out-Null
PS /home/labadmin> Add-AzTableRow -table $cloudTable -partitionKey $partitionKey1 `
>> -rowKey ("WA") -property @{"username"="Christine";"userid"=3} | Out-Null
PS /home/labadmin> Add-AzTableRow -table $cloudTable -partitionKey $partitionKey2 `
>> -rowKey ("TX") -property @{"username"="Steven";"userid"=4} | Out-Null
PS /home/labadmin>
```

10. [스토리지 브라우저]로 이동한 후 PowerShell을 통해 추가한 엔터티가 표시되는지 확인합니다.

kornttablestorage < + 엔터티 추가 새로 고침 삭제 열 편집

즐거찾기

최근 본 항목

Blob 컨테이너

파일 공유

큐

테이블

tableDemo1

tableDemo2

tableDemo3

모두 보기

테이블 > tableDemo3

인증 방법: 액세스 키 (Azure AD 사용자 계정으로 전환)

쿼리 작성기

쿼리 작성기에서는 AND/OR를 사용하여 간단한 쿼리를 지원합니다. 그룹화 및 기타 고급 설정과 같은 기능의 경우 고급 텍스트 편집기를 사용하세요.

고급 텍스트 편집기로 전환

필터 추가

항목 4개 모두 표시

<input type="checkbox"/>	PartitionKey	RowKey	Timestamp	userid	username	
<input type="checkbox"/>	partition1	CA	2022-04-03T02:11:32.71...	1	Chris	...
<input type="checkbox"/>	partition1	WA	2022-04-03T02:11:33.16...	3	Christine	...
<input type="checkbox"/>	partition2	NM	2022-04-03T02:11:32.94...	2	Jessie	...
<input type="checkbox"/>	partition2	TX	2022-04-03T02:11:33.39...	4	Steven	...

11. 위의 내용은 PowerShell에서도 확인할 수 있습니다. [Cloud Shell]에서 다음 명령을 실행하여 테이블의 엔터티를 확인합니다.

```
# PowerShell로 테이블 쿼리
Get-AzTableRow -table $cloudTable | FT

PowerShell
PS /home/labadmin> # PowerShell로 테이블 쿼리
PS /home/labadmin> Get-AzTableRow -table $cloudTable | FT

userid username PartitionKey RowKey TableTimestamp Etag
-----
1 Chris partition1 CA 4/3/2022 2:11:32 AM +00:00 W/"datetime'2022-04-03T02:3A11:3A32.71069592'"
3 Christine partition1 WA 4/3/2022 2:11:33 AM +00:00 W/"datetime'2022-04-03T02:3A11:3A33.16643362'"
2 Jessie partition2 NM 4/3/2022 2:11:32 AM +00:00 W/"datetime'2022-04-03T02:3A11:3A32.94056372'"
4 Steven partition2 TX 4/3/2022 2:11:33 AM +00:00 W/"datetime'2022-04-03T02:3A11:3A33.39330142'"

PS /home/labadmin>
```

12. [Cloud Shell]에서 다음 쿼리를 실행하여 특정 PartitionKey에 대한 행만 반환합니다.

```
# 특정 PartitionKey로 쿼리
Get-AzTableRow -table $cloudTable -PartitionKey $partitionKey1 | FT
```

```
PowerShell
PS /home/labadmin> # 특정 PartitionKey로 쿼리
PS /home/labadmin> Get-AzTableRow -table $cloudTable -PartitionKey $partitionKey1 | FT

userid username PartitionKey RowKey TableTimestamp Etag
-----
1 Chris partition1 CA 4/3/2022 2:11:32 AM +00:00 W/"datetime'2022-04-03T02:3A11:3A32.7106959Z'"
3 Christine partition1 WA 4/3/2022 2:11:33 AM +00:00 W/"datetime'2022-04-03T02:3A11:3A33.1664336Z'"

PS /home/labadmin>
```

13. [Cloud Shell]에서 다음 명령을 실행하여 특정 속성의 값을 기준으로 쿼리를 실행합니다.

```
# 특정 속성의 값을 기준으로 쿼리
Get-AzTableRow -Table $cloudTable -ColumnName "username" `
-Value "Chris" -Operator Equal

PowerShell
PS /home/labadmin> # 특정 속성의 값을 기준으로 쿼리
PS /home/labadmin> Get-AzTableRow -Table $cloudTable -ColumnName "username" `
>> -Value "Chris" -Operator Equal

userid      : 1
username    : Chris
PartitionKey : partition1
RowKey      : CA
TableTimestamp : 4/3/2022 2:11:32 AM +00:00
Etag        : W/"datetime'2022-04-03T02:3A11:3A32.7106959Z'"

PS /home/labadmin>
```

14. [Cloud Shell]에서 다음 명령을 실행하여 사용자 지정 필터로 쿼리를 수행할 수 있습니다.

```
# 사용자 지정 필터로 쿼리
Get-AzTableRow -Table $cloudTable `
-CustomFilter "(userid eq 1)"

PowerShell
PS /home/labadmin> # 사용자 지정 필터로 쿼리
PS /home/labadmin> Get-AzTableRow -Table $cloudTable `
>> -CustomFilter "(userid eq 1)"

userid      : 1
username    : Chris
PartitionKey : partition1
RowKey      : CA
TableTimestamp : 4/3/2022 2:11:32 AM +00:00
Etag        : W/"datetime'2022-04-03T02:3A11:3A32.7106959Z'"

PS /home/labadmin>
```

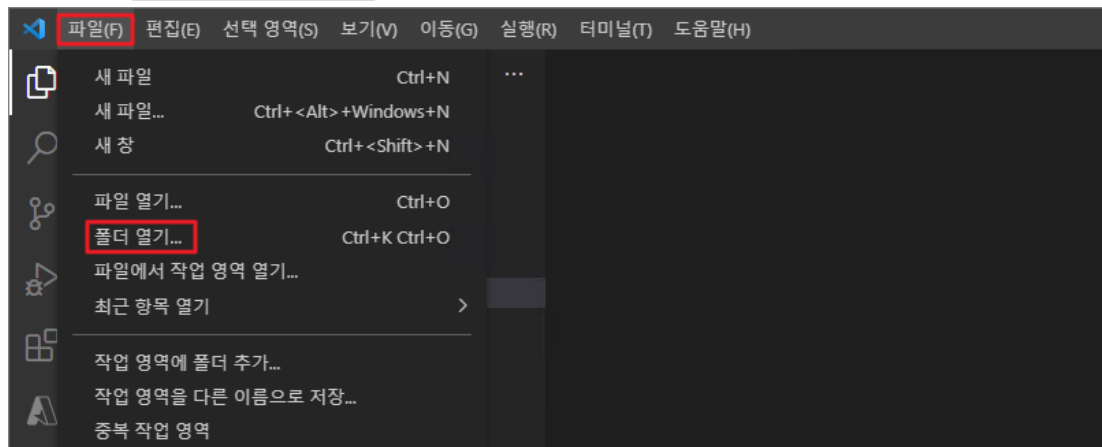
## TASK 05. .NET으로 Table 스토리지 작업

1. [스토리지 계정] 블레이드의 [스토리지 브라우저]에서 [테이블]을 선택합니다. 메뉴에서 [테이블 추가]를 클릭한 후 테이블 이름에 "tableDemo4"를 입력한 후 [확인]을 클릭합니다.

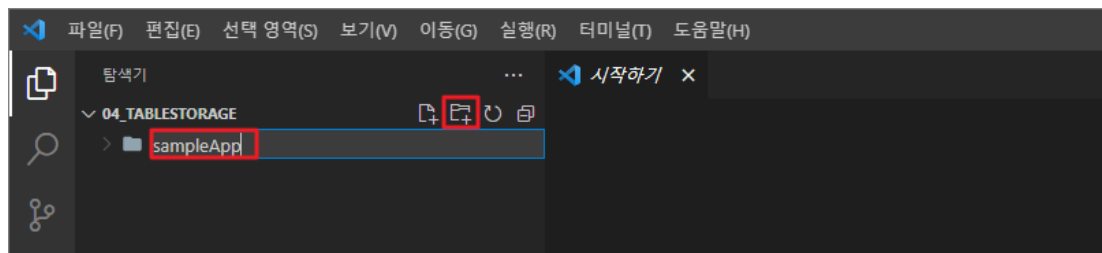




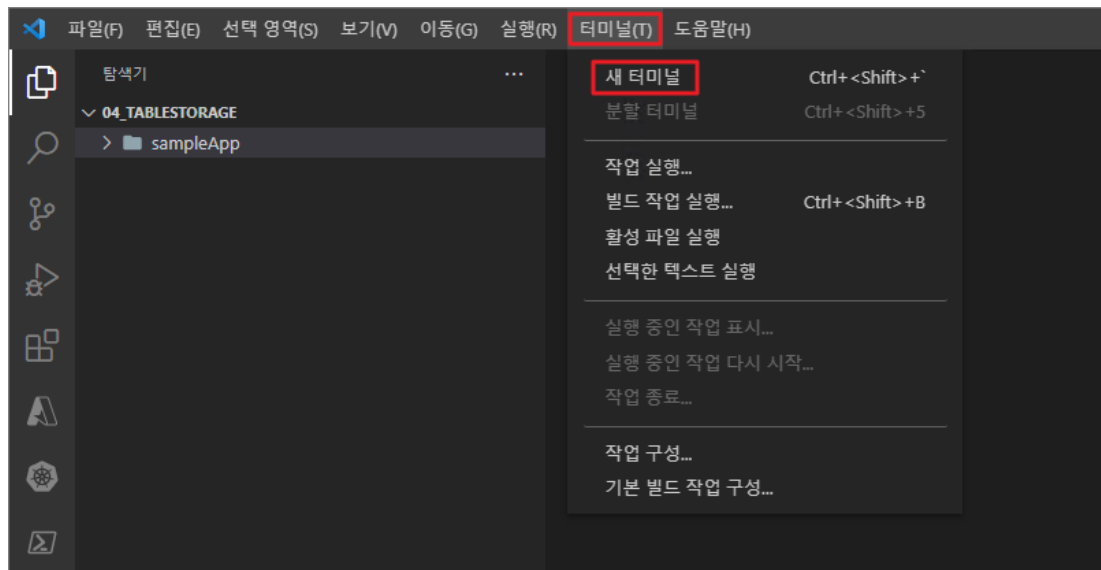
2. labVM 실험 가상 머신에 로그인합니다. Visual Studio Code를 실행한 후 메뉴에서 [파일 - 폴더 열기]를 클릭합니다. C:\04\_tableStorage 폴더를 만들고 이 폴더를 선택합니다.



3. Visual Studio Code의 [탐색기] 창에서 [새 폴더] 아이콘을 클릭한 후 sampleApp 이름의 폴더를 만듭니다.

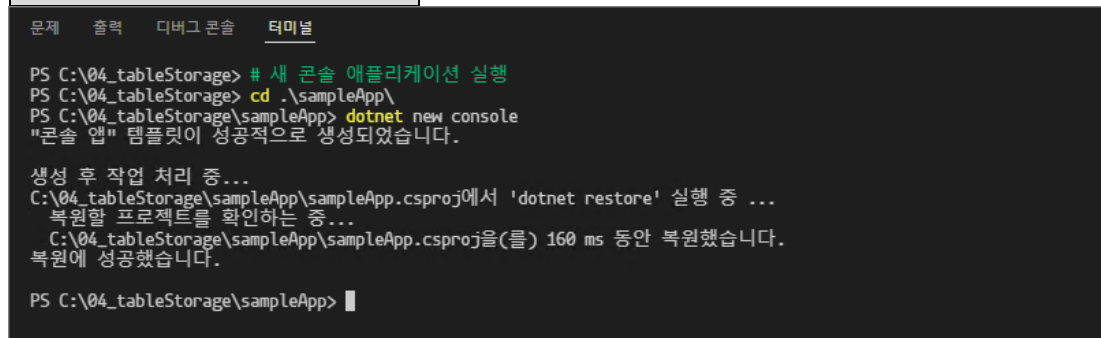


4. Visual Studio Code의 메뉴에서 [터미널 - 새 터미널]을 클릭합니다.



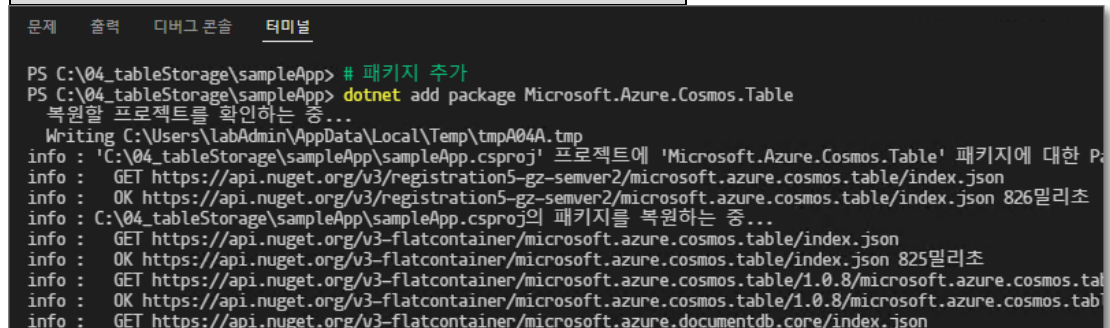
5. [터미널] 창에서 다음 명령을 실행하여 새로 만든 `C:\04_tableStorage\sampleApp` 폴더로 이동하고 새 콘솔 애플리케이션을 초기화합니다.

```
# 새 콘솔 애플리케이션 실행
cd .\sampleApp\
dotnet new console
```



6. [터미널]에서 다음 명령을 실행하여 애플리케이션에 필요한 패키지를 설치합니다.

```
# 패키지 추가
dotnet add package Microsoft.Azure.Cosmos.Table
```



7. [탐색기]에서 `C:\04_tableStorage\sampleApp\Program.cs` 파일을 열고 다음과 같은 기본 코드를 작성합니다. Table Storage 작업에 필요한 `using` 문을 추가합니다.

```
using System;
using System.Threading.Tasks;
```

```
using Microsoft.Azure.Cosmos.Table;

namespace demo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Table Storage Sample!");
        }
    }
}
```

파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) 도움말(H)

탐색기

04\_TABLESTORAGE

- > .vscode
- > sampleApp
  - > bin
  - > obj
  - C# Program.cs
  - sampleApp.csproj
  - command.ps1

C# Program.cs X

sampleApp > C# Program.cs > {} demo

```
1 using System;
2 using System.Threading.Tasks;
3 using Microsoft.Azure.Cosmos.Table;
4
5 namespace demo
6 {
7     0 references
8     class Program
9     {
10         0 references
11         static void Main(string[] args)
12         {
13             Console.WriteLine("Table Storage Sample!");
14         }
15     }
16 }
```

8. Table Storage에 연결하기 위한 변수를 아래와 같이 추가합니다.

```
var storageConnectionString = "";
var tableName = "";
```

C# Program.cs 2 X

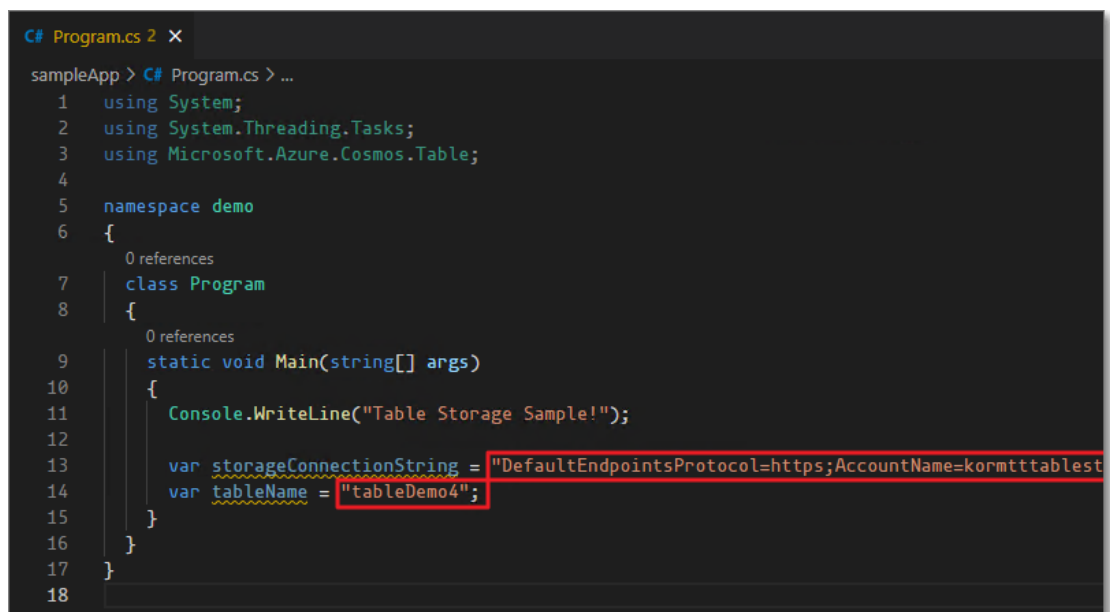
sampleApp > C# Program.cs > ...

```
1 using System;
2 using System.Threading.Tasks;
3 using Microsoft.Azure.Cosmos.Table;
4
5 namespace demo
6 {
7     0 references
8     class Program
9     {
10         0 references
11         static void Main(string[] args)
12         {
13             Console.WriteLine("Table Storage Sample!");
14             var storageConnectionString = "";
15             var tableName = "";
16         }
17     }
18 }
```

9. [스토리지 계정] 블레이드의 [보안 + 네트워킹 - 액세스 키]로 이동한 후 [키 표시]를 클릭합니다. **key1**의 연결 문자열 값을 클립보드에 복사합니다.



10. **storageConnectionString**에 클립보드에 복사한 스토리지 연결 문자열 값을 붙여 넣고 **tableName**에 새로 만들 테이블 이름인 "tableDemo4"를 입력합니다.



11. 다음 코드를 추가하여 스토리지 연결 개체를 만듭니다. 이를 위해 **CloudStorageAccount** 클래스를 사용합니다. 코드에서 연결 문자열을 구문분석(parsing)하는 스토리지 계정 변수를 지정하여 연결 위치를 알 수 있게 합니다.

```
CloudStorageAccount storageAccount;
storageAccount = CloudStorageAccount.Parse(storageConnectionString);
```

```

C# Program.cs 1 X
sampleApp > C# Program.cs > ...
1  using System;
2  using System.Threading.Tasks;
3  using Microsoft.Azure.Cosmos.Table;
4
5  namespace demo
6  {
7      0 references
8      class Program
9      {
10         0 references
11         static void Main(string[] args)
12         {
13             Console.WriteLine("Table Storage Sample!");
14             var storageConnectionString = "DefaultEndpointsProtocol=https;AccountName=kormtttablest";
15             var tableName = "tableDemo4";
16             CloudStorageAccount storageAccount;
17             storageAccount = CloudStorageAccount.Parse(storageConnectionString);
18         }
19     }
20 }
21

```

12. 테이블에 대한 클라우드 참조를 만들기 위해 다음과 같은 코드를 추가합니다.

```

CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new
TableClientConfiguration());
CloudTable table = tableClient.GetTableReference(tableName);

```

```

C# Program.cs X
sampleApp > C# Program.cs > ...
1  using System;
2  using System.Threading.Tasks;
3  using Microsoft.Azure.Cosmos.Table;
4
5  namespace demo
6  {
7      0 references
8      class Program
9      {
10         0 references
11         static void Main(string[] args)
12         {
13             Console.WriteLine("Table Storage Sample!");
14             var storageConnectionString = "DefaultEndpointsProtocol=https;AccountName=kormtttablestorage;AccountKey=pC4Yu4Ctp";
15             var tableName = "tableDemo4";
16             CloudStorageAccount storageAccount;
17             storageAccount = CloudStorageAccount.Parse(storageConnectionString);
18             CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new TableClientConfiguration());
19             CloudTable table = tableClient.GetTableReference(tableName);
20         }
21     }
22 }
23
24

```

13. 이제 테이블 참조를 추가하였기 때문에 데이터를 추가할 수 있습니다. 다음과 같은 코드를 추가합니다.

- ① 새 클래스를 만들고 `TableEntity` 클래스를 상속하여 `PartitionKey`와 `RowKey`를 가져오도록 합니다.
- ② `CustomerEntity`에서 `PartitionKey`와 `RowKey`를 `lastName`과 `firstName`으로 재정의합니다.
- ③ 추가 속성으로 `Email`과 `PhoneNumber`를 추가합니다.

```

public class CustomerEntity : TableEntity

```

```
{
    public CustomerEntity() {}
    public CustomerEntity(string lastName, string firstName)
    {
        PartitionKey = lastName;
        RowKey = firstName;
    }
    public string? Email { get; set; }
    public string? PhoneNumber { get; set; }
}
```

```
C# Program.cs X
sampleApp > C# Program.cs > ...
19     CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new TableClientConfiguration());
20     CloudTable table = tableClient.GetTableReference(tableName);
21 }
22 }
23
24 public class CustomerEntity : TableEntity { 1
25 {
26     public CustomerEntity() {}
27     public CustomerEntity(string lastName, string firstName) { 2
28     {
29         PartitionKey = lastName;
30         RowKey = firstName;
31     }
32     public string? Email { get; set; } 3
33     public string? PhoneNumber { get; set; }
34 }
35 }
36 }
```

14. 이제 테이블에 행을 삽입할 수 있는 메서드를 추가해야 합니다. 다음과 같은 코드를 추가합니다.

- ① `MergeUser` 이름의 새 메서드를 만듭니다. 비동기식 프로그래밍을 사용해야하기 때문에 앞서 `using System.Threading.Tasks`를 추가하였습니다.
- ① 메서드에서 `CloudTable`과 앞서 만들었던 `CustomerEntity`를 매개 변수로 가져옵니다.
- ② `TableOperation`을 만들고 `CustomerEntity`에 대한 `InsertOrMerge` 작업을 만듭니다.
- ③ 작업을 비동기로 실행한 후 결과를 얻습니다.

```
public static async Task MergeUser(CloudTable table, CustomerEntity customer) {
    TableOperation insertOrMergeOperation = TableOperation.InsertOrMerge(customer);

    // 작업 실행
    TableResult result = await table.ExecuteAsync(insertOrMergeOperation);
    CustomerEntity? insertedCustomer = result.Result as CustomerEntity;

    Console.WriteLine("New user is added.");
}
```

```

C# Program.cs x
sampleApp > C# Program.cs > ...
19 CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new TableClientConfiguration());
20 CloudTable table = tableClient.GetTableReference(tableName);
21 }
22
0 references
23 public static async Task MergeUser(CloudTable table, CustomerEntity customer) {
24     TableOperation insertOrMergeOperation = TableOperation.InsertOrMerge(customer);
25
26     // 작업 실행
27     TableResult result = await table.ExecuteAsync(insertOrMergeOperation);
28     CustomerEntity? insertedCustomer = result.Result as CustomerEntity;
29
30     Console.WriteLine("New user is added.");
31 }
32 }
33
3 references
34 public class CustomerEntity : TableEntity
35 {
    0 references
  
```

15. 이제 새 고객 엔터티를 만드는 내용을 추가할 수 있습니다. 다음과 같은 코드를 추가하여 메서드를 사용하여 해당 엔터티를 추가합니다.

```

CustomerEntity customer = new CustomerEntity("Maggie", "Simpson")
{
    Email = "MarggieS@contoso.com",
    PhoneNumber = "1-555-4353"
};
  
```

```

MergeUser(table, customer).Wait();
  
```

```

C# Program.cs x
sampleApp > C# Program.cs > {} demo
19 CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new TableClientConfiguration());
20 CloudTable table = tableClient.GetTableReference(tableName);
21
22 CustomerEntity customer = new CustomerEntity("Maggie", "Simpson")
23 {
24     Email = "MarggieS@contoso.com",
25     PhoneNumber = "1-555-4353"
26 };
27
28 MergeUser(table, customer).Wait();
29 }
30
1 reference
31 public static async Task MergeUser(CloudTable table, CustomerEntity customer) {
32     TableOperation insertOrMergeOperation = TableOperation.InsertOrMerge(customer);
33
34     // 작업 실행
35     TableResult result = await table.ExecuteAsync(insertOrMergeOperation);
36     CustomerEntity? insertedCustomer = result.Result as CustomerEntity;
37
38     Console.WriteLine("New user is added.");
39 }
40 }
41
42
  
```

16. Visual Studio Code의 [터미널]에서 다음 명령을 실행하여 애플리케이션을 빌드하고 실행합니다.

```

# 애플리케이션 빌드
dotnet build

# 애플리케이션 실행
dotnet run
  
```

```

문제  출력  디버그 콘솔  터미널

PS C:\04_tableStorage\sampleApp> # 애플리케이션 빌드
PS C:\04_tableStorage\sampleApp> dotnet build
.NET용 Microsoft (R) Build Engine 버전 17.1.0+ae57d105c
Copyright (C) Microsoft Corporation. All rights reserved.

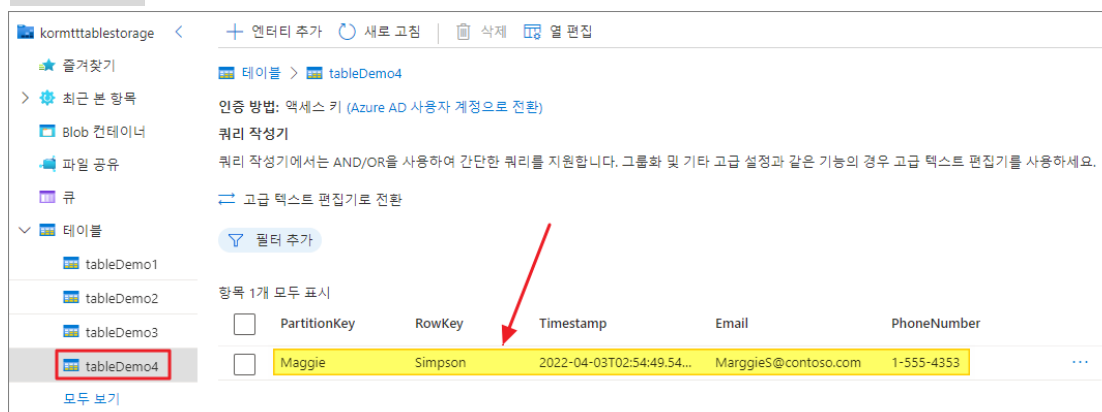
복원할 프로젝트를 확인하는 중...
복원할 모든 프로젝트가 최신 상태입니다.
sampleApp -> C:\04_tableStorage\sampleApp\bin\Debug\net6.0\sampleApp.dll

빌드했습니다.
경고 0개
오류 0개

경과 시간: 00:00:16.53
PS C:\04_tableStorage\sampleApp> # 애플리케이션 실행
PS C:\04_tableStorage\sampleApp> dotnet run
Table Storage Sample!
New user is added.
PS C:\04_tableStorage\sampleApp>

```

17. Azure 포털의 [스토리지 계정] 블레이드로 이동한 후 [스토리지 브라우저]를 클릭합니다. [테이블 - tableDemo4]로 이동한 후 아래와 같이 엔터티가 추가된 것을 확인합니다.



항목 1개 모두 표시	PartitionKey	RowKey	Timestamp	Email	PhoneNumber
<input type="checkbox"/>		Maggie	2022-04-03T02:54:49.54...	MaggieS@contoso.com	1-555-4353

18. 기존에 입력했던 엔터티의 내용을 변경할 수 있습니다. Visual Studio Code로 전환한 후 앞서 추가했던 엔터티의 Email과 PhoneNumber 값을 변경합니다.

```

C# Program.cs x
sampleApp > C# Program.cs > {} demo
19 CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new TableClientConfiguration());
20 CloudTable table = tableClient.GetTableReference(tableName);
21
22 CustomerEntity customer = new CustomerEntity("Maggie", "Simpson")
23 {
24     Email = "MaggieSimpson@contoso.com",
25     PhoneNumber = "82-2-555-8950"
26 };
27
28
29 MergeUser(table, customer).Wait();
30 }
31
1 reference

```

19. 데이터 쿼리를 위해 다음과 같은 코드를 추가합니다.

- ① QueryUser라는 새 메서드를 만들고 쿼리에 사용할 firstName, lastName을 매개 변수로 지정합니다.
- ② 쿼리를 위해 TableOperation을 정의하고 CustomerEntity를 검색한 후 JSON 테이블 데이터에서 개체로 자동 매핑을 수행합니다.
- ③ 쿼리 결과를 저장합니다.



- ④ 엔터티가 있으면 지정한 형식으로 값을 출력하도록 구성합니다.

```
public static async Task QueryUser(CloudTable table, string firstName, string lastName) {
    TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>(firstName,
lastName);

    TableResult result = await table.ExecuteAsync(retrieveOperation);
    CustomerEntity? customer = result.Result as CustomerEntity;

    if (customer != null)
    {
        Console.WriteLine("Fetched {0}\t{1}\t{2}\t{3}",
            customer.PartitionKey, customer.RowKey, customer.Email, customer.PhoneNumber);
    }
}
```

The screenshot shows the Visual Studio Code editor with the `Program.cs` file open. The `QueryUser` method is highlighted with a red box. Four red circles with numbers 1 through 4 are placed over specific parts of the code: 1 is over the `QueryUser` method signature, 2 is over the `TableOperation.Retrieve` call, 3 is over the `TableResult` and `CustomerEntity` assignment, and 4 is over the `if` block that prints the customer information.

20. 이제 엔터티를 쿼릴 수 있으므로 다음과 같은 코드를 추가하고 검색할 사용자 값을 입력합니다.

```
QueryUser(table, "Maggie", "Simpson").Wait();
```

The screenshot shows the Visual Studio Code editor with the `Program.cs` file open. The `main` method is highlighted with a red box. It shows the creation of a `CustomerEntity` object named `customer` with the name "Maggie" and the last name "Simpson". Then, the `MergeUser` method is called, followed by the `QueryUser` method, which is highlighted with a red box. The `QueryUser` method is called with the `table` and the name "Maggie".

21. Visual Studio Code에서 다음 명령을 실행하여 애플리케이션을 다시 빌드하고 실행합니다. 실행 결과에서 업데이트된 엔터티 정보가 표시되는 것을 확인합니다.

```
# 애플리케이션 빌드
dotnet build
```

```
# 애플리케이션 실행
dotnet run
```

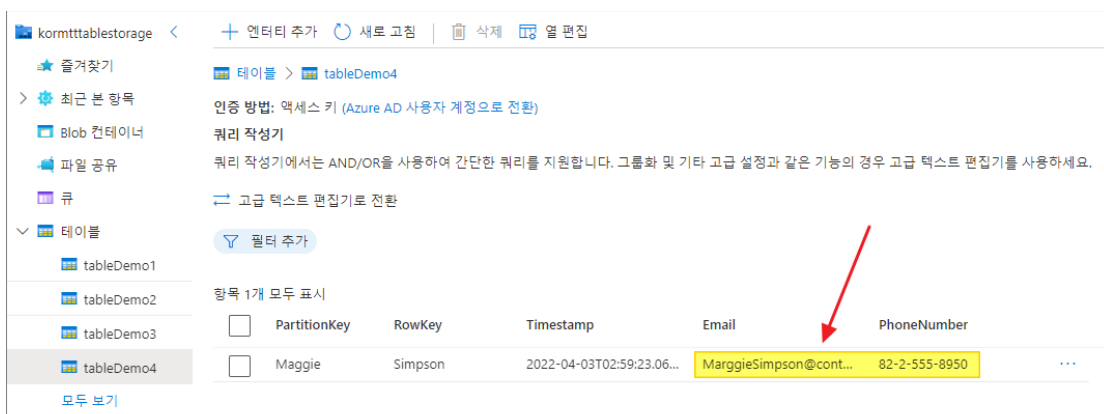
```
PS C:\04_tableStorage\sampleApp> # 애플리케이션 빌드
PS C:\04_tableStorage\sampleApp> dotnet build
.NET용 Microsoft (R) Build Engine 버전 17.1.0+ae57d105c
Copyright (C) Microsoft Corporation. All rights reserved.

복원할 프로젝트를 확인하는 중...
복원할 모든 프로젝트가 최신 상태입니다.
sampleApp -> C:\04_tableStorage\sampleApp\bin\Debug\net6.0\sampleApp.dll

빌드했습니다.
경고 0개
오류 0개

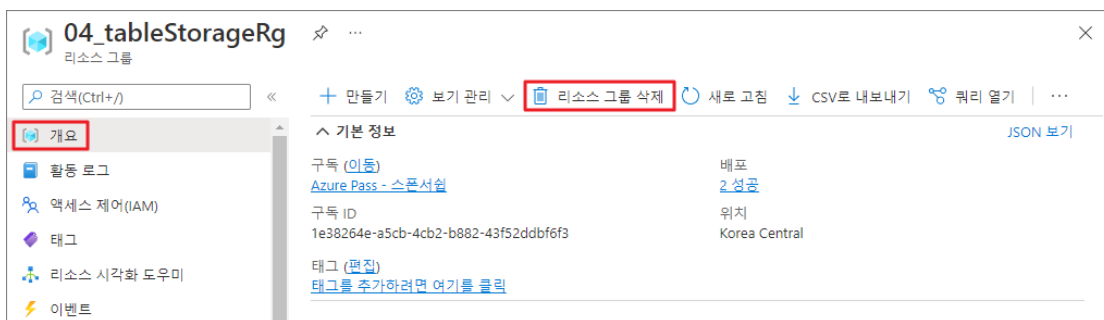
결과 시간: 00:00:02.85
PS C:\04_tableStorage\sampleApp> # 애플리케이션 실행
PS C:\04_tableStorage\sampleApp> dotnet run
Table Storage Sample!
New user is added.
Fetched      Maggie Simpson MarggieSimpson@contoso.com      82-2-555-8950
PS C:\04_tableStorage\sampleApp> █
```

22. Azure 포털로 전환한 후 [스토리지 계정] 블레이드의 [스토리지 브라우저]에서 아래와 같이 업데이트된 내용을 확인할 수 있습니다.



## TASK 06. 리소스 정리

1. Azure 포털에서 [04\_tableStorageRg 리소스 그룹] 블레이드로 이동한 후 메뉴에서 [리소스 그룹 삭제]를 클릭합니다.



2. 리소스 그룹 삭제 확인 창에서 리소스 그룹 이름을 입력한 후 [삭제]를 클릭합니다.

