**실습 3 (Python) - Amazon DynamoDB를 사용한 솔루션 개발**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. 본 내용은 Amazon Web Services, Inc.의 사전 서면 허가 없이 전체 또는 일부를 복제하거나 재배포할 수 없습니다. 상업적인 복제, 임대 또는 판매는 금지됩니다. 모든 상표는 해당 소유자의 자산입니다.

참고: 개인 정보, 개인을 식별할 수 있는 정보 또는 기밀 정보는 실습 환경에 입력하지 마십시오. 입력한 정보가 다른 사용자에게 공개될 수도 있습니다.

수정 사항이나 피드백 또는 기타 질문이 있으십니까? [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training)에서 문의해 주십시오.

**소요 시간**

이 실습은 완료하는 데 약 **60분**이 소요됩니다.

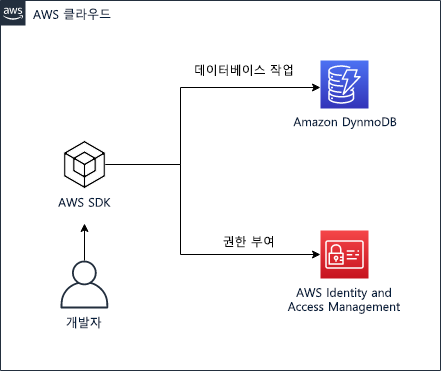
**선행 조건**

이 실습에는 다음이 필요합니다.

* Wi-Fi가 연결된 Microsoft Windows 또는 macOS 노트북 컴퓨터.
* 인터넷 브라우저(예: Chrome, Firefox 또는 IE9 이상).
* **참고:** 이전 버전의 Internet Explorer는 지원되지 않습니다.
* **참고:** 이 지침은 iPad 또는 태블릿 디바이스를 사용해 실습 콘솔에서 확인할 수 있습니다.
* **추가 정보:** 추가 실습 환경 관련 세부 정보는 [부록](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#appendix)을 참조하십시오.

**개요**

이 실습에서는 실습 애플리케이션에 대한 노트를 저장할 Amazon DynamoDB 테이블을 생성하는 프로그램을 개발합니다. 이후 테이블에서 프로그래밍 방식으로 CRUD 작업을 수행합니다.



이 실습에 대한 코드를 개발하기 위해 [Python용 AWS SDK(Boto3) 설명서](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html)를 사용합니다. 시간을 단축하기 위해 기본 코딩 구성 요소가 AWS Cloud9 IDE에 제공됩니다. 실습 동안 TODO 섹션 대신 코드를 완료하는 도전 과제가 주어집니다. 도전 과제 완료를 위한 추가 지원이 필요한 경우 인라인 솔루션 단서 또는 솔루션 폴더에 있는 완료된 프로그램 파일을 참조할 수 있습니다.

**목표**

이 실습을 완료하면 다음을 할 수 있게 됩니다.

* 프로그램의 하위 수준, 문서 및 상위 수준 API를 사용하여 프로그래밍 방식으로 DynamoDB와 상호 작용합니다.
* 파티션 키, 정렬 키 및 적합한 프로비저닝된 처리량을 포함한 Waiter를 사용하여 테이블을 생성합니다.
* 파일에서 JSON 객체를 읽고 테이블을 로드합니다.
* 키 속성, 필터, 표현식 및 페이지 배열을 사용하여 테이블에서 항목을 검색합니다.
* 새 속성을 추가하고 조건부로 데이터를 변경하여 항목을 업데이트합니다.
* PartiQL을 사용하여 DynamoDB 데이터에 액세스합니다.

**실습 시작**

1. 실습을 시작하려면 페이지 상단에서 **실습 시작**을 선택합니다.

 계속 진행하기 전에 프로비저닝된 AWS 서비스가 준비될 때까지 기다려야 합니다.

1. 실습을 열려면 **콘솔 열기**를 선택합니다.

그러면 새 웹 브라우저 탭에서 자동으로 AWS Management Console에 로그인됩니다.

**별다른 지시가 없는 한 리전을 변경하지 마십시오.**

**일반적인 로그인 오류**

**오류: 우선 로그 아웃 필요**

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**You must first log out before logging into a different AWS account**라는 메시지가 표시된다면 다음을 수행합니다.

* **click here**의 링크를 선택합니다.
* **Amazon Web Services Sign In** 웹 브라우저 탭을 닫고 초기 실습 페이지로 돌아갑니다.
* **콘솔 열기**를 다시 선택합니다.

**오류: 실습 시작 선택 시 아무 반응이 없음**

경우에 따라 일부 팝업 또는 스크립트 차단 웹 브라우저 확장 프로그램 때문에 **실습 시작** 버튼이 제대로 작동하지 않을 수 있습니다. 실습을 시작하는 데 문제가 있는 경우 다음을 수행합니다.

* 팝업 또는 스크립트 차단 프로그램의 허용 목록에 실습 도메인 이름을 추가하거나 차단 프로그램을 끕니다.
* 페이지를 새로 고친 후 다시 시도하십시오.

**과제 1: Amazon DynamoDB 테이블 생성**

이 과제에서는 AWS Cloud9 IDE를 사용하여 createTable.py 프로그램을 완료합니다. 수강생의 과제는 이름이

Notes

이고, 파티션 키가 **UserId**이고, 정렬 키가 **NoteId**이며, 프로비저닝된 처리량이 5개의 읽기 및 쓰기 용량 단위인 DynamoDB 테이블을 생성하는 것입니다. 필요한 경우 Waiter를 사용하여 테이블의 가용성을 확보해야 합니다.

**main** 함수에 의해 다음 순서에 따라 **labRepo/createTable.py** 파일이 처리됩니다.

* config.ini 파일을 읽고 모든 정적 값에 대한 설정을 사용합니다.
* **createTable**은 실습 AWS 계정에 Amazon DynamoDB 테이블을 생성합니다.
* **waitForTableCreation**은 테이블이 준비될 때까지 스크립트를 일시 중지합니다.
* **getTableInfo**는 확인을 위해 계정에서 생성된 테이블의 구성을 가져옵니다.

**고려 사항:** 이 실습은 숙련된 개발자와 신입 개발자 모두를 위해 고안되었습니다.

* 도전을 즐기는 숙련된 개발자를 위해 각 과제에 앞서 과제 완료에 도움이 되는 충분한 정보가 담긴 **개략적 지침**이 제공됩니다.
* 업데이트를 완료하면 코드를 테스트하여 작동하는지 확인하고, 필요시 문제를 해결한 후 다음 과제를 진행하십시오.
* 신입 개발자는 실습의 각 단계를 자세히 설명하는 **상세 지침**을 참고할 수 있습니다.

**개략적 지침**

**참고:** Cloud9 환경이 **ready**인지 확인하세요. **labRepo**라는 폴더와 **Lab-Is-Ready**라는 폴더가 있는 구조를 확인하여 이를 확인할 수 있습니다. 표시되지 않으면 최종 구성이 완료될 때까지 잠시 기다리십시오.

**참고:** 다음 메시지가 표시되면 **Accept** 을 선택하세요.

* **.c9/project.settings have been changed on disk**
  + 프로젝트 설정 파일(.c9/project.settings)이 IDE 외부에서 업데이트되었습니다. 직접 변경하지 않은 경우 수락하기 전에 파일을 검토하는 것을 권장합니다. 새 설정을 수락하시겠습니까?
* **labRepo/createTable.py**를 완료하여 새 Amazon DynamoDB 테이블을 생성합니다.
* **TODO 1** 섹션을 완료하고 Amazon DynamoDB 서비스 클라이언트를 생성합니다.
* **TODO 2**를 완료하고 DynamoDB 테이블을 생성합니다.
* 계속하기 전에 테이블이 존재하기 전까지 스크립트가 대기하도록 **TODO 3**를 완료합니다.
* 스크립트를 실행하고 오류를 해결합니다.

**상세 지침**

**과제 1.1: AWS Cloud9 환경 구성 및 이 실습에 대한 구성 설정 검사**

1. **AWS Management Console**에서 **AWS 검색 창**을 사용하여

Cloud9

을 검색한 다음 결과 목록에서 서비스를 선택합니다.

1. **Environments** 페이지의 **Lab3** 환경 목록 옆에 있는 **Open**을 선택합니다.

**참고:** Cloud9 환경이 **ready**인지 확인하세요. **labRepo**라는 폴더와 **Lab-Is-Ready**라는 폴더가 있는 구조를 확인하여 이를 확인할 수 있습니다. 표시되지 않으면 최종 구성이 완료될 때까지 잠시 기다리십시오.

**참고:** 다음 메시지가 표시되면 **Accept** 을 선택하세요.

* **.c9/project.settings have been changed on disk**
  + 프로젝트 설정 파일(.c9/project.settings)이 IDE 외부에서 업데이트되었습니다. 직접 변경하지 않은 경우 수락하기 전에 파일을 검토하는 것을 권장합니다. 새 설정을 수락하시겠습니까?

1. **labRepo/config.ini** 파일을 엽니다.

이 구성 파일에는 이 실습의 모든 스크립트에 대한 모든 정적 값이 있습니다. 테이블 이름, 키 이름 및 용량 레벨 파라미터를 검토합니다. 설정을 변경할 필요는 없습니다. 이 실습에서 사용되는 다른 스크립트를 개발할 때 참조할 수 있도록 스크립트를 열어두십시오.

**과제 1.2: Amazon DynamoDB 서비스 클라이언트 생성**

이 실습의 스크립트는 이전 실습과 동일한 구조로 구성되어 있습니다. 모든 종속성을 가져온 상태이고, 필요한 모든 함수가 정의되어 있고, 서비스 클라이언트가 생성되고, **main** 함수가 호출됩니다.

1. **labRepo/createTable.py** 파일을 엽니다.

이 파일은 테이블 생성에 사용됩니다. 스크립트가 성공적으로 함수를 수행하도록 코드를 완료해야 합니다. 파일에서 함수가 정의된 이후 첫 번째 TODO 섹션이 있습니다.

1. **복사/붙여넣기:** 올바른 옵션을 선택하고 삽입하여 **TODO 1**을 위한 서비스 클라이언트를 생성합니다.

* 보기 A

client = boto3.client('dynamodb')

* 보기 B

client = boto3.client('rds')

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 1 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-1-solution) 섹션에서 찾을 수 있습니다.

클라이언트가 생성된 이후 main 함수를 호출하고 이를 사용할 서비스 클라이언트에 전달합니다. 처리되지 않은 클라이언트 또는 파라미터 검증 오류가 main 함수 내에서 처리되지 않은 경우 이 레벨에서 탐지되고 출력에 표시됩니다.

**과제 1.3: 코드를 개발하여 테이블 생성**

이 실습의 코드를 쉽게 따라할 수 있도록 모든 로직이 **main** 함수에서 호출되어 있습니다. 모든 코드 도전 과제와 결과 형식 지정에 필요할 수 있지만 학습 목표에 속하지 않는 코드는 별도의 함수로 분류됩니다.

main 함수에서 생성될 테이블에 대한 정보는 **config.ini** 파일에서 읽습니다.

첫 번째 코드 도전 과제에서는 Amazon DynamoDB 테이블을 생성해야 합니다. **createTable** 함수가 DynamoDB 클라이언트와 정보가 포함된 객체로 전달되어 테이블을 생성합니다. 전체 config 객체가 전달된 이후 tableDefinition 변수로서 **config.ini**의 모든 것에 액세스할 수 있습니다.

1. **복사/붙여넣기:** **TODO 2**: 올바른 코드 조각을 선택하고 삽입하여 **createTable.py**를 완료합니다.

* 보기 A

response = ddbClient.create\_table(tableDefinition)

* 보기 B

response = ddbClient.create\_table(

AttributeDefinitions=[

{

'AttributeName': tableDefinition["partitionKey"],

'AttributeType': 'S',

},

{

'AttributeName': tableDefinition["sortKey"],

'AttributeType': 'N',

},

],

KeySchema=[

{

'AttributeName': tableDefinition["partitionKey"],

'KeyType': 'HASH',

},

{

'AttributeName': tableDefinition["sortKey"],

'KeyType': 'RANGE',

},

],

ProvisionedThroughput={

'ReadCapacityUnits': int(tableDefinition["readCapacity"]),

'WriteCapacityUnits': int(tableDefinition["writeCapacity"]),

},

TableName=tableDefinition["tableName"]

)

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 2 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-2-solution) 섹션에서 찾을 수 있습니다.

**과제 1.4: 테이블이 준비되었을 경우에만 이어서 코드를 실행하도록 하는 코드 개발**

DynamoDB 테이블을 생성하는 것은 비동기식 작업이며 테이블을 생성 중이라는 응답이 반환됩니다. 테이블이 생성될 때까지 코드가 대기하도록 **TODO 3**를 완료합니다.

1. **복사/붙여넣기:** 올바른 코드 조각을 선택하고 삽입하여 **TODO 3**를 완료합니다.

* 보기 A

waiter = ddbClient.get\_waiter('table\_exists')

waiter.wait(TableName=tableName)

* 보기 B

waiter = ddbClient.get\_waiter('table\_not\_exists')

waiter.wait(TableName=tableName)

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 3 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-3-solution) 섹션에서 찾을 수 있습니다.

1. **createTable.py**를 저장하고 닫습니다.

두 코드 모두 도전 과제를 완료하고 나서 스크립트를 실행할 수 있습니다.

1. **AWS Cloud9 터미널**에서 다음 명령을 실행하여 디렉터리를 **~/environment** 디렉터리로 변경합니다.

cd ~/environment

1. **명령:** 아래 명령을 사용하여 **createTable.py**를 실행합니다.

python labRepo/createTable.py

**예상 출력:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Creating an Amazon DynamoDB table "Notes"

with a Partition key: "UserId"

and sort key: "NoteId".

Table Status: CREATING

Waiting for the table to be available...

Table is now available.

Table Status: ACTIVE

스크립트는 create\_table call에서 응답을 반환하고, 대기하고, describe\_table call의 정보를 표시합니다.

 축하합니다! Amazon DynamoDB에서 테이블을 성공적으로 생성했습니다.

**과제 2: 테이블에 데이터 로드**

테이블을 성공적으로 생성했으므로 실습 애플리케이션 데이터를 가져올 수 있습니다. 이 과제에서는 **loadData.py** 파일을 완료하고 이를 사용하여 **notes.json** 파일에서 노트를 가져옵니다. **notes.json** 파일은 이전 실습에서 생성한 것과 동일한 파일이며, 이미 **labRepo** 폴더에 다운로드되었습니다.

이 스크립트는 낮은 수준 클라이언트 대신 DynamoDB 테이블 리소스를 사용하고 다음 순서로 처리됩니다.

* config.ini 파일을 읽고 모든 정적 값에 대한 설정을 사용합니다.
* 소스 JSON 파일에서 JSON 노트 목록을 생성합니다.
* 지난 과제에서 생성한 테이블에 대한 테이블 리소스를 생성합니다.
* 노트 목록에서 반복하여 각 노트를 **putNote** 함수가 있는 테이블에 추가합니다.

**개략적 지침**

* **labRepo/loadData.py**를 완료하고 새 테이블로 노트를 가져옵니다.
* **TODO 4** 섹션을 완료하고 함수로 전달된 리소스와 노트를 사용하여 테이블에 항목을 추가합니다.
* 스크립트를 실행하고 오류를 해결합니다.

**상세 지침**

**과제 2.1: 코드를 개발하여 테이블에 데이터를 로드하고 코드 실행**

1. **labRepo/loadData.py**를 엽니다.

이 스크립트는 **notes.json** 파일에서 노트를 가져오고 DynamoDB 테이블 리소스를 사용하여 각 노트를 지난 과제에서 생성한 DynamoDB 테이블에 추가합니다.

1. **복사/붙여넣기:** **TODO 4**: 다음 보기 중 하나를 사용하여 putNote 함수를 완료하고 DynamoDB 테이블에 각 노트를 추가합니다.

* 보기 A

table.put\_item(

Item={

'UserId': note["UserId"],

'Note': note["Note"]

}

)

* 보기 B

table.put\_item(

Item={

'UserId': note["UserId"],

'NoteId': int(note["NoteId"]),

'Note': note["Note"]

}

)

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 4 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-4-solution) 섹션에서 찾을 수 있습니다.

1. **loadData.py** 파일을 저장하고 닫습니다.
2. **명령:** **~/environment** 디렉터리에서 **loadData.py** 스크립트를 실행하고, 아래 명령을 사용하여 테이블에 레코드를 생성합니다.

python labRepo/loadData.py

**예상 출력:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Loading "Notes" table with data from file "notes.json"

loading note {'UserId': 'testuser', 'NoteId': '001', 'Note': 'hello'}

loading note {'UserId': 'testuser', 'NoteId': '002', 'Note': 'this is my first note'}

loading note {'UserId': 'newbie', 'NoteId': '001', 'Note': 'Free swag code: 1234'}

loading note {'UserId': 'newbie', 'NoteId': '002', 'Note': 'I love DynamoDB'}

loading note {'UserId': 'student', 'NoteId': '001', 'Note': 'DynamoDB is NoSQL'}

loading note {'UserId': 'student', 'NoteId': '002', 'Note': 'A DynamoDB table is schemaless'}

loading note {'UserId': 'student', 'NoteId': '003', 'Note': 'PartiQL is a SQL compatible language for DynamoDB'}

loading note {'UserId': 'student', 'NoteId': '005', 'Note': 'Maximum size of an item is \_\_\_\_ KB ?'}

loading note {'UserId': 'student', 'NoteId': '004', 'Note': 'I love DyDB'}

Finished loading notes from the JSON file.

 축하합니다! 테이블로 항목을 로드하는 데 성공했습니다.

**과제 3: 파티션 키 및 프로젝션을 사용하여 데이터 쿼리**

데이터를 저장하는 것만큼 DynamoDB 테이블에서 데이터를 가져오는 것도 중요합니다. 이 과제에서는 **queryData.py** 파일을 완료하여 파티션 키만을 사용해 테이블에서 데이터를 가져옵니다. 또한 프로젝션 표현식을 사용하여 지정한 값만 표시합니다.

**main** 함수에 의해 다음 순서에 따라 **labRepo/queryData.py** 파일이 처리됩니다.

* config.ini 파일을 읽고 모든 정적 값에 대한 설정을 사용합니다.
* **queryNotesByPartitionKey**는 DynamoDB 쿼리를 사용하여 프로젝션 표현식을 사용하여 파티션 키가 알려진 특정 항목을 반환합니다.

**개략적 지침**

* **labRepo/queryData.py**를 완료하여 특정 UserId에 대한 노트를 찾습니다.
* **TODO 5**를 완료하고 테이블을 쿼리합니다.
* 스크립트를 실행하고 오류를 해결합니다.

**상세 지침**

**과제 3.1: 코드를 개발하여 테이블에서 특정 노트 반환**

이 과제에서는 DynamoDB 테이블을 쿼리하고 파티션 키를 사용하여 특정 항목을 가져옵니다. 또한 Projection Expression을 사용하여 반환된 항목에서 NoteId 및 Note만 표시합니다. **printNotes** 함수가 사용되어 화면에 인쇄될 때 DynamoDB JSON을 더욱 읽기 쉬운 형식으로 변환합니다.

1. **labRepo/queryData.py** 파일을 엽니다.
2. **복사/붙여넣기:** **TODO 5**: 올바른 보기를 사용하여 함수로 전달된 파라미터를 사용해 코드를 완료하여 테이블을 쿼리합니다.

* 보기 A

response = ddbClient.query(

TableName=tableName,

KeyConditionExpression='UserId = :userId',

ExpressionAttributeValues={

':userId': {"S": qUserId}

},

ProjectionExpression="NoteId, Note"

)

* 보기 B

response = ddbClient.query(

TableName=tableName,

KeyConditionExpression='UserId = :userId',

ExpressionAttributeValues={

':userId': qUserId

},

ProjectionExpression="NoteId, Note"

)

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 5 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-5-solution) 섹션에서 찾을 수 있습니다.

**참고:** **An attribute defined in json.encoder line 158 hides this method** 오류는 무시해도 좋습니다.

1. **명령:** **~/environment** 디렉터리에서 **queryData.py** 스크립트를 실행하고, 아래 명령을 사용하여 테이블에서 항목을 표시합니다.

python labRepo/queryData.py

**예상 출력:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*

Querying for notes that belong to user student...

{"Note": "DynamoDB is NoSQL", "NoteId": "1"}

{"Note": "A DynamoDB table is schemaless", "NoteId": "2"}

{"Note": "PartiQL is a SQL compatible language for DynamoDB", "NoteId": "3"}

{"Note": "I love DyDB", "NoteId": "4"}

{"Note": "Maximum size of an item is \_\_\_\_ KB ?", "NoteId": "5"}

 축하합니다! 이제 테이블에서 항목을 검색하는 방법을 알게 되었습니다.

**과제 4: Paginator를 사용하여 테이블 스캔**

이 과제에서는 Notes 테이블에서 모든 데이터를 읽습니다. Scan 작업을 사용하여 전체 테이블을 읽습니다.

Scan 작업은 테이블 또는 보조 인덱스의 모든 항목에 액세스하여 하나 이상의 항목 및 항목 속성을 반환합니다. DynamoDB가 반환하는 항목 수를 줄이려는 경우 FilterExpression 작업을 제공할 수 있습니다. DynamoDB는 Scan 작업의 결과를 페이지로 배열합니다. 페이지 배열을 사용하면 Scan 결과는 크기가 1MB(또는 미만)인 데이터 '페이지’들로 나뉩니다. 애플리케이션은 결과의 첫 번째 페이지를 처리한 다음, 필요한 페이지를 연속으로 처리할 수 있습니다. 또한 단일 페이지에서 검색할 최대 항목 수를 제한할 수 있습니다.

**main** 함수에 의해 다음 순서에 따라 **labRepo/paginateData.py** 파일이 처리됩니다.

* config.ini 파일을 읽고 모든 정적 값에 대한 설정을 사용합니다.
* **queryAllNotesPaginator**는 Paginator를 사용하여 스캔을 수행하고 반환된 모든 페이지를 표시합니다.

**개략적 지침**

* **labRepo/paginateData.py**를 완료하여 특정 UserId에 대한 노트를 찾습니다.
* **TODO 6**을 완료하고 Paginator를 사용하여 스캔합니다.
* 스크립트를 실행하고 오류를 해결합니다.

**상세 지침**

**과제 4.1: 모든 노트를 반환하고, 페이지에 표시는 코드개발 및 코드 실행**

이 파일은 Paginator를 사용하여 결과를 처리할 수 있는 여러 페이지에 분할합니다. 이 과제는 쿼리 또는 스캔을 사용하여 완료할 수 있습니다. 예시에서는 더 많은 레코드가 반환되도록 스캔을 사용합니다.

1. **labRepo/paginateData.py** 파일을 엽니다.
2. **복사/붙여넣기:** **TODO 6**: Paginator를 생성하고 printNotes 함수에 각 페이지의 항목을 반환하는 올바른 보기를 선택합니다.

* 보기 A

paginator = ddbClient.get\_paginator('scan')

page\_iterator = paginator.paginate(

TableName=tableName,

PaginationConfig={

'PageSize': pageSize

},

ProjectionProcessor=printNotes

)

* 보기 B

paginator = ddbClient.get\_paginator('scan')

page\_iterator = paginator.paginate(

TableName=tableName,

PaginationConfig={

'PageSize': pageSize

})

pageNumber = 0

for page in page\_iterator:

if page["Count"] > 0:

pageNumber += 1

print("Starting page " + str(pageNumber))

printNotes(page['Items'])

print("End of page " + str(pageNumber) + "\n")

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 6 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-6-solution) 섹션에서 찾을 수 있습니다.

1. **명령:** **~/environment** 디렉터리의 **paginateData.py** 스크립트를 실행하고, 다음 명령을 사용하여 테이블의 항목을 표시합니다.

python labRepo/paginateData.py

**예상 출력:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Scanning with pagination...

Starting page 1

{"Note": "hello", "UserId": "testuser", "NoteId": "1"}

{"Note": "this is my first note", "UserId": "testuser", "NoteId": "2"}

{"Note": "DynamoDB is NoSQL", "UserId": "student", "NoteId": "1"}

End of page 1

Starting page 2

{"Note": "A DynamoDB table is schemaless", "UserId": "student", "NoteId": "2"}

{"Note": "PartiQL is a SQL compatible language for DynamoDB", "UserId": "student", "NoteId": "3"}

{"Note": "I love DyDB", "UserId": "student", "NoteId": "4"}

End of page 2

Starting page 3

{"Note": "Maximum size of an item is \_\_\_\_ KB ?", "UserId": "student", "NoteId": "5"}

{"Note": "Free swag code: 1234", "UserId": "newbie", "NoteId": "1"}

{"Note": "I love DynamoDB", "UserId": "newbie", "NoteId": "2"}

End of page 3

 축하합니다! 이제 테이블에서 항목을 검색하는 방법을 알게 되었습니다.

**과제 5: 테이블의 항목 업데이트**

이 과제에서는 2개의 업데이트 작업을 수행합니다. 첫 번째 작업은 *Is\_Incomplete*라는 새 속성을 추가하여 기존 항목을 업데이트하고 *Yes* 값을 할당하는 것입니다. 두 번째 작업은 특정 조건이 충족되는 경우, 즉 *Is\_Incomplete = Yes*인 경우에만 동일한 항목을 업데이트하는 것입니다. 이를 통해 항목의 원치 않는 변경을 막을 수 있습니다.

**main** 함수에 의해 다음 순서에 따라 **labRepo/updateItem.py** 파일이 처리됩니다.

* config.ini 파일을 읽고 모든 정적 값에 대한 설정을 사용합니다.
* **updateNewAttribute**는 특정 노트에 관한 새 속성을 생성하여 이를 incomplete으로 입력합니다.
* **updateExistingAttributeConditionally**는 해당 항목을 업데이트하되 플래그가 incomplete으로 표시된 경우에만 업데이트합니다.

**개략적 지침**

* **labRepo/updateItem.py**를 완료하고 두 가지 방식으로 항목을 업데이트합니다.
* **TODO 7**을 완료하고 기존 항목에 속성을 추가합니다.
* **TODO 8**을 완료하고 추가한 속성을 기준으로 조건부로 항목을 업데이트합니다.
* 스크립트를 실행하고 오류를 해결합니다.

**상세 지침**

**과제 5.1: DynamoDB 테이블에서 항목을 업데이트하는 코드 개발**

항목 업데이트는 속성 값 중 일부를 변경해야 하는 경우에 한하여 항목 교체에 비해 더욱 효율적입니다.

1. **labRepo/updateItem.py** 파일을 엽니다.
2. **복사/붙여넣기:** **TODO 7**에서 올바른 보기를 선택하고 파일에 삽입하여 **Is\_Incomplete** 속성을 항목에 추가합니다.

* 보기 A

response = ddbClient.put\_item(

TableName=tableName,

Item={

'UserId': {'S': qUserId},

'NoteId': {'N': str(qNoteId)},

'Is\_Incomplete': {'S': 'Yes'}

},

ReturnValues='ALL\_NEW'

)

* 보기 B

response = ddbClient.update\_item(

TableName=tableName,

Key={

'UserId': {'S': qUserId},

'NoteId': {'N': str(qNoteId)}

},

ReturnValues='ALL\_NEW',

UpdateExpression='SET Is\_Incomplete = :incomplete',

ExpressionAttributeValues={

':incomplete': {'S': 'Yes'}

}

)

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 7 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-7-solution) 섹션에서 찾을 수 있습니다.

**과제 5.2: DynamoDB 테이블에서 조건부로 항목을 업데이트하는 코드 개발 및 코드 실행**

항목에 대한 무언가가 True일 때에만 항목을 업데이트하려는 경우가 있습니다. 예를 들어 사용자가 많은 애플리케이션의 경우 사용 중일 때 레코드를 잠그고자 할 수 있습니다. 레코드가 잠긴 경우 이 상태에서 레코드가 변경될 수 있는지 여부를 애플리케이션에서 알 수 있습니다.

1. **labRepo/updateItem.py**를 계속 편집합니다.
2. **복사/붙여넣기:** **TODO 8**에서 올바른 보기를 선택하여 항목을 조건부로 업데이트합니다. 이 TODO에서 두 가지 보기 모두에서 코드가 올바르고, 정확하지 않은 노트를 사용하여 항목을 업데이트하는 옵션을 선택합니다.

**참고:** **config.ini** 파일에서 **notePrefix** 변수의 값을 확인할 수 있습니다.

* 보기 A

notePrefix += ' 400 KB'

* 보기 B

notePrefix += ' 100 KB'

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 8 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-8-solution) 섹션에서 찾을 수 있습니다.

1. **updateItem.py**를 저장합니다.
2. **명령:** **~/environment** 디렉터리에서 **updateItem.py** 스크립트를 실행하고, 아래 명령을 사용하여 테이블에 레코드를 생성합니다.

python labRepo/updateItem.py

**예상 출력:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Updating the note flag for remediation...

{'Note': {'S': 'Maximum size of an item is \_\_\_\_ KB ?'}, 'UserId': {'S': 'student'}, 'NoteId': {'N': '5'}, 'Is\_Incomplete': {'S': 'Yes'}}

Remediating the marked note...

{'Note': {'S': 'The maximum item size in DynamoDB is 400 KB'}, 'UserId': {'S': 'student'}, 'NoteId': {'N': '5'}, 'Is\_Incomplete': {'S': 'No'}}

 축하합니다! 이제 테이블에서 특정 속성을 업데이트할 수 있습니다.

**과제 5.3: 조건이 True가 아닌 항목 업데이트**

지금까지는 스크립트에서 Is\_Incomplete 플래그를 ‘YES’ 로 설정하고 이 플래그의 값을 확인하는 동안 항목을 업데이트했습니다. 이 과제에서는 Is\_Incomplete 플래그를 먼저 설정하지 않고 항목을 업데이트 하였을때 어떻게 되는지 확인해 보겠습니다.

1. 다음 코드를 표시하는 **updateItem.py** 파일의 라인 16을 삭제합니다. (참고로 이 함수 호출은 특정 함목을 찾아 IS\_Incomplet = ‘YES’ 로 선언하는 함수 입니다. )

print(updateNewAttribute(ddbClient, tableName, qUserId, qNoteId))

1. **updateItem.py**를 저장하고 닫습니다.
2. **명령:** **~/environment** 디렉터리에서 **updateItem.py** 스크립트를 실행하고 다음 명령을 사용하여 플래그를 먼저 설정하지 않고 항목을 업데이트합니다.

python labRepo/updateItem.py

**예상 출력:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Updating the note flag for remediation...

Remediating the marked note...

Sorry, your update is invalid mate!

 축하합니다! 이제 테이블에서 특정 속성을 업데이트할 수 있습니다. 또한 조건부로 항목을 업데이트할 수 있음을 확인했습니다.

**선택 과제 6: DynamoDB용 PartiQL 사용**

이 실습에서는 데이터베이스 작업에 대해 기존 API 메서드를 사용했습니다. 한 가지 중요한 점은 기존 SQL 기반 데이터베이스 엔진과 완전히 다르다는 점입니다. Amazon DynamoDB는 또한 SQL 호환 쿼리 언어인 **PartiQL**을 지원하여 테이블에 데이터를 선택, 삽입, 업데이트 및 삭제합니다.

이 선택 과제에서 **PartiQL**을 사용하여 과제 3에서 먼저 검색한 것과 동일한 노트를 쿼리합니다.

**main** 함수에 의해 다음 순서에 따라 **labRepo/partiQL.py** 파일이 처리됩니다.

* config.ini 파일을 읽고 모든 정적 값에 대한 설정을 사용합니다.
* **querySpecificNote**는 partiQL을 사용하여 특정 노트에 대한 DynamoDB를 쿼리합니다.

**개략적 지침**

* **labRepo/partiQL.py**를 완료하고 테이블에서 항목을 검색합니다.
* **TODO 9**를 완료하고 PartiQL을 사용하여 테이블을 쿼리합니다.
* 스크립트를 실행하고 오류를 해결합니다.

**상세 지침**

**과제 6.1: PartiQL을 사용하여 DynamoDB를 쿼리하는 코드를 개발하고 코드 실행**

이 과제에서는 이 실습의 최종 TODO 섹션을 완료합니다.

1. **labRepo/partiQL.py** 파일을 엽니다.
2. **복사/붙여넣기:** **TODO 9**에서 올바른 코드 조각을 선택하고 **querySpecificNote** 함수에 삽입합니다.

* 보기 A

response = ddbClient.query(

KeyConditionExpression="SELECT \* FROM " + tableName + " WHERE UserId = ? AND NoteId = ?",

ExpressionAttributeValues=[

{"S": qUserId},

{"N": str(qNoteId)}

]

)

* 보기 B

response = ddbClient.execute\_statement(

Statement="SELECT \* FROM " + tableName + " WHERE UserId = ? AND NoteId = ?",

Parameters=[

{"S": qUserId},

{"N": str(qNoteId)}

]

)

**해답:** 이 단계의 해답은 이 지침 하단의 [TODO 9 해답](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-9-solution) 섹션에서 찾을 수 있습니다.

**참고:** **An attribute defined in json.encoder line 158 hides this method** 오류는 무시해도 좋습니다.

1. **명령:** **~/environment** 디렉터리에서 **partiQL.py** 스크립트를 실행하고, 다음 명령을 사용하여 테이블에 레코드를 생성합니다.

python labRepo/partiQL.py

**예상 출력:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*

Querying for note 5 that belongs to user student...

{"Note": "The maximum item size in DynamoDB is 400 KB", "UserId": "student", "NoteId": "5", "Is\_Incomplete": "No"}

 축하합니다! 이제 PartiQL을 사용하여 DynamoDB 테이블과 상호 작용하는 방법을 알게 되었습니다.

**요약**

**축하합니다! 실습의 Python 버전을 완료했습니다!** 이제 다음을 수행할 수 있습니다.

* 프로그램의 하위 수준, 문서 및 상위 수준 API를 사용하여 프로그래밍 방식으로 DynamoDB와 상호 작용합니다.
* 파티션 키, 정렬 키 및 적합한 프로비저닝된 처리량을 포함한 Waiter를 사용하여 테이블을 생성합니다.
* 파일에서 JSON 객체를 읽고 테이블을 로드합니다.
* 키 속성, 필터, 표현식 및 페이지 배열을 사용하여 테이블에서 항목을 검색합니다.
* 새 속성을 추가하고 조건부로 데이터를 변경하여 항목을 업데이트합니다.
* PartiQL을 사용하여 DynamoDB 데이터에 액세스합니다.

**실습 종료**

다음 단계에 따라 콘솔을 닫고 실습을 종료합니다.

1. **AWS Management Console**로 돌아갑니다.
2. 페이지 오른쪽 상단에서 **AWSLabsUser**를 선택하고 **Sign out**을 선택합니다.
3. **실습 종료**를 선택한 다음 실습을 종료할 것임을 확인합니다.

**추가 자료**

* [Amazon DynamoDB 개발자 안내서](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html)
* [Python용 AWS SDK(Boto3) - DynamoDB Service](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html)

AWS Training and Certification에 대한 자세한 내용은 [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/)을 참조하십시오.

*여러분의 피드백을 환영합니다.* 피드백, 제안 사항 또는 수정 요청 사항을 제공하려면 [*AWS Training and Certification 문의 양식*](https://support.aws.amazon.com/#/contacts/aws-training)에 세부 정보를 입력해 주시기 바랍니다.

**코드 도전 과제 해답**

**TODO 1 해답**

* **보기 A가 올바른 코드 조각입니다**.

client = boto3.client('dynamodb')

* 보기 B는 올바르지 않습니다. DynamoDB가 관계형 데이터베이스가 아니며 고유한 클라이언트가 있기 때문입니다.

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-1)

**TODO 2 해답**

* 보기 A는 올바르지 않습니다. tableDefinition 변수에 전달된 사전이 DynamoDB 테이블을 생성하기에 올바른 형식이 아니기 때문입니다.
* **보기 B가 올바른 코드 조각입니다**.

response = ddbClient.create\_table(

AttributeDefinitions=[

{

'AttributeName': tableDefinition["partitionKey"],

'AttributeType': 'S',

},

{

'AttributeName': tableDefinition["sortKey"],

'AttributeType': 'N',

},

],

KeySchema=[

{

'AttributeName': tableDefinition["partitionKey"],

'KeyType': 'HASH',

},

{

'AttributeName': tableDefinition["sortKey"],

'KeyType': 'RANGE',

},

],

ProvisionedThroughput={

'ReadCapacityUnits': int(tableDefinition["readCapacity"]),

'WriteCapacityUnits': int(tableDefinition["writeCapacity"]),

},

TableName=tableDefinition["tableName"]

)

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-2)

**TODO 3 해답**

* **보기 A가 올바른 코드 조각입니다**. *table\_exists* Waiter를 사용하여 테이블이 사용 가능할 때까지 일시 중지하고자 합니다.

waiter = ddbClient.get\_waiter('table\_exists')

waiter.wait(TableName=tableName)

* 보기 B는 올바르지 않습니다. 파라미터가 올바르게 설정되지 않았기 때문입니다.

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-3)

**TODO 4 해답**

* *NoteId* 속성이 포함되어 있지 않으므로 보기 A는 올바르지 않습니다. 복합 파티션 키의 경우 파티션 키 및 정렬 키 모두에 대한 값을 제공해야 합니다.
* **보기 B가 올바른 코드 조각입니다**.

table.put\_item(

Item={

'UserId': note["UserId"],

'NoteId': int(note["NoteId"]),

'Note': note["Note"]

}

)

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-4)

**TODO 5 해답**

* **보기 A가 올바른 코드 조각입니다**.

response = ddbClient.query(

TableName=tableName,

KeyConditionExpression='UserId = :userId',

ExpressionAttributeValues={

':userId': {"S": qUserId}

},

ProjectionExpression="NoteId, Note"

)

* ExpressionsAttributeValues 형식이 올바르게 지정되지 않았으므로 보기 B는 올바르지 않습니다. 값에서 값의 데이터 유형을 지정해야 합니다.

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-5)

**TODO 6 해답**

* 보기 A는 올바르지 않습니다. 이를 처리하기 위해 페이지 배열 호출의 결과를 반복해야 하기 때문입니다.
* **보기 B가 올바른 코드 조각입니다.**

paginator = ddbClient.get\_paginator('scan')

page\_iterator = paginator.paginate(

TableName=tableName,

PaginationConfig={

'PageSize': pageSize

})

pageNumber = 0

for page in page\_iterator:

if page["Count"] > 0:

pageNumber += 1

print("Starting page " + str(pageNumber))

printNotes(page['Items'])

print("End of page " + str(pageNumber) + "\n")

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-6)

**TODO 7 해답**

* 유효한 구문이 있지만 전체 노트를 전달된 항목으로 교체하므로 보기 A는 올바르지 않습니다. 이 경우 실제 Note 속성이 제거됩니다.
* **보기 B가 올바른 코드 조각입니다**.

response = ddbClient.update\_item(

TableName=tableName,

Key={

'UserId': {'S': qUserId},

'NoteId': {'N': str(qNoteId)}

},

ReturnValues='ALL\_NEW',

UpdateExpression='SET Is\_Incomplete = :incomplete',

ExpressionAttributeValues={

':incomplete': {'S': 'Yes'}

}

)

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-7)

**TODO 8 해답**

* **보기 A가 올바른 코드 조각입니다**. DynamoDB의 최대 항목 크기는 400KB입니다.

notePrefix += ' 400 KB'

* 보기 B는 올바르지 않습니다. 100KB는 DynamoDB에서 최대 항목 크기가 아니기 때문입니다.

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-8)

**TODO 9 해답**

* 파라미터가 올바르게 설정되지 않았으므로 보기 A는 올바르지 않습니다.
* **보기 B가 올바른 코드 조각입니다.** PartiQL을 사용하려면 execute\_statement 메서드를 사용해야 합니다.

response = ddbClient.execute\_statement(

Statement="SELECT \* FROM " + tableName + " WHERE UserId = ? AND NoteId = ?",

Parameters=[

{"S": qUserId},

{"N": str(qNoteId)}

]

)

[지침으로 돌아가기](https://classrooms.labs.aws.training/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A006961644361%3Ablueprintversion%2FILT-TF-200-DODEVA-4%2Flab-3-python%3A4.5.10-cb60e669/ko-KR/e8919513-39ae-48d9-af22-3a606f897383::a2xyUiReAFJzLdS3VtTyoX#todo-9)

**부록**

**이 실습에서 사용하지 않는 AWS 서비스**

이 실습에서 사용하지 않는 AWS 서비스는 실습 환경에서 비활성화됩니다. 또한 이 실습에 사용되는 서비스의 기능은 실습에 필요한 작업으로 제한됩니다. 다른 서비스에 액세스하거나 실습 안내서에서 제공하는 것 외의 작업을 수행하는 경우 오류가 발생할 수 있습니다.

**아이콘 설명**