Perform cross-document transactional operations

Introduction

- Create a transactional batch and review results
- Implement optimistic concurrency control for an operation

Create a transactional batch with the SDK

- Product saddle = new("0120", "Worn Saddle", "accessories-used");
- Product handlebar = new("012A", "Rusty Handlebar", "accessories-used");

- public record Product(string id, string name, string categoryId);

- PartitionKey partitionKey = new ("accessories-used");
- TransactionalBatch batch = container.CreateTransactionalBatch(partitionKey)
- .CreateItem<Product>(saddle)
- .CreateItem<Product>(handlebar);
- using TransactionalBatchResponse response = await batch.ExecuteAsync();

Create a transactional batch with the SDK

- The transactional batch supports operations with the same logical partition key.
- Operations with different logical partition keys will fail. In the example below, the transactional batch will fail with a bad request due to having a different logical partition key.

```
Product saddle = new("0120", "Worn Saddle", "accessories-used");
Product handlebar = new("012C", "Pristine Handlebar", "accessories-new");
PartitionKey partitionKey = new ("accessories-used");
TransactionalBatch batch = container.CreateTransactionalBatch(partitionKey)
    .CreateItem<Product>(saddle)
    .CreateItem<Product>(handlebar);
```

Create a transactional batch with the SDK

- Transactional batch also supports a wide variety of operations using the fluent syntax including, but not limited to:

Method	Description
`CreateItemStream()`	Create item from existing stream
`DeleteItem()`	Delete an item
`ReadItem()`	Read an item
`ReplaceItem()` & `ReplaceItemStream()`	Update an existing item or stream
`UpsertItem()` & `UpsertItemStream()`	Create or update an existing item or stream based on the item's unique identifier

Review batch operation results with the SDK

- The TransactionalBatchResponse class contains multiple members to interrogate the results of the batch operation.
  - response.StatusCode
  - batchResponse.IsSuccessStatusCode
- The GetOperationResultAtIndex<> generic method returns the individual deserialized item at the index you specify.
  - TransactionalBatchOperationResult<Product> result = response.GetOperationResultAtIndex<Product>(0);
  - Product firstProductResult = result.Resource;
  - TransactionalBatchOperationResult<Product> result = response.GetOperationResultAtIndex<Product>(1);
  - Product secondProductResult = result.Resource;

Exercise

- Batch multiple point operations together with the Azure Cosmos DB for NoSQL SDK

Implement optimistic concurrency control

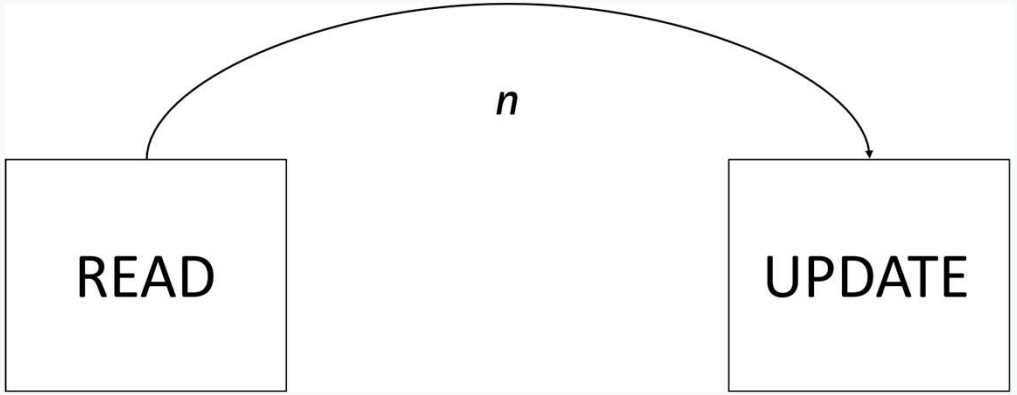
- Using the SDK to read an item and then update the same item in a subsequent operation carries some inherent risk.
- Another operation could potentially come in from a separate client and change the underlying document before the first client's update operation is finalized
- This conflict could create a "lost update" situation.

Implement optimistic concurrency control

- Here is a typical C# code example with a separate read and update operation.
  - string categoryId = "9603ca6c-9e28-4a02-9194-51cdb7fea816";
  - PartitionKey partitionKey = new (categoryId);
  - Product product = await container.ReadItemAsync<Product>("01AC0", partitionKey);
  - product.price = 50d;
  - await container.UpsertItemAsync<Product>(product, partitionKey);

Implement optimistic concurrency control

- Since read and write in this example are distinct operations, there is a latency between these operations. This latency is represented in this diagram as n.



Implement optimistic concurrency control

- This issue can be resolved by implementing optimistic concurrency control.
- Each item has an ETag value. This value is updated when the item is updated.
- To prevent lost updates, you can use the if-match rule to see if the ETag still matches the current ETag header of the item server-side as part of your update request.
  - string categoryId = "9603ca6c-9e28-4a02-9194-51cdb7fea816";
  - PartitionKey partitionKey = new (categoryId);

  - ItemResponse<Product> response = await container.ReadItemAsync<Product>("01AC0", partitionKey);
  - Product product = response.Resource;
  - string eTag = response.ETag;

  - product.price = 50d;

  - ItemRequestOptions options = new ItemRequestOptions { IfMatchEtag = eTag };
  - await container.UpsertItemAsync<Product>(product, partitionKey, requestOptions: options);

Thank You