Technical posts

Catering for Azure Cosmos DB Optimistic Concurrency

Optimistic concurrency control (OCC) is a form of concurrency control that allows multiple database transactions to occur without conflicting with each other. This blog will explore how to implement OCC for Azure Cosmos DB, when making data transactions from a C# code.

Andrew Camilleri
16 September 2021

We shall start this blog post by understanding what causes a database concurrency issue, the difference between how Transactional Databases and NoSQL databases handle such issues, and an example on how to implement optimistic concurrency in an Azure Cosmos DB when submitting documents from code.

Database concurrency issues occur when multiple users/processes try to access and write a record in the database, resulting in multiple versions of the data having conflicting values. Typically, to update a record the process would involve: reading the record, updating the record in memory, and writing it back to the database. If, for any reason, multiple processes have read the same record, modified it in memory and try to write it back, the record that was modified might not be the latest and thus cause a versioning issue with the data (Figure 1).

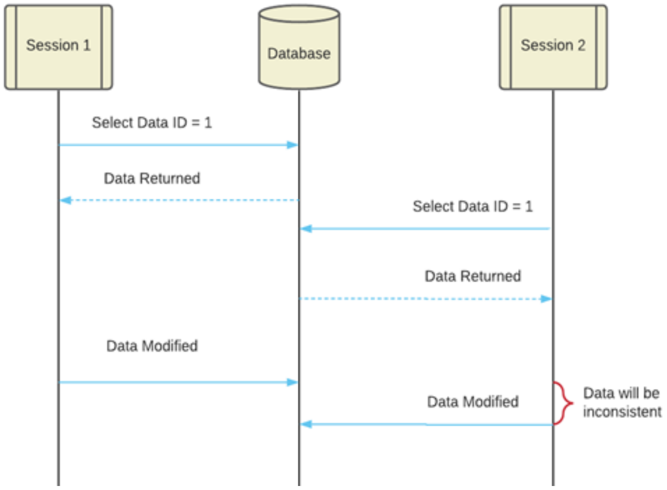
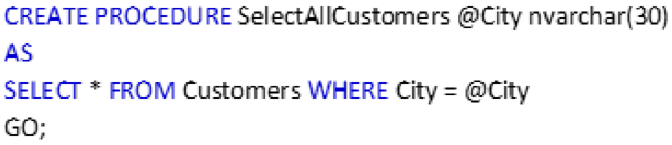


Figure 1: Database Concurrency Issue

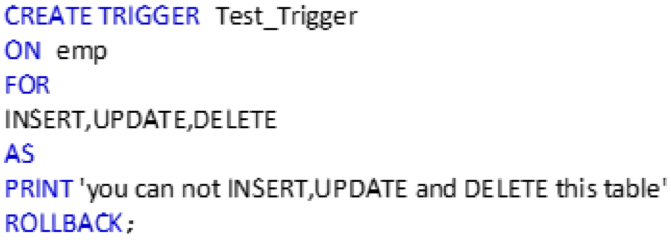
Traditional relational databases (such as Microsoft SQL) handle OCC with the use of transactions. One can implement a transaction either through the use of applicable programming languages such as Python, C#, Java, etc., or by implementing the code as a

Transactional Programming Language (T-SQL), which is executed by the database through the use of a stored-procedures (Figure 2) and/or triggers (Figure 3).



```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

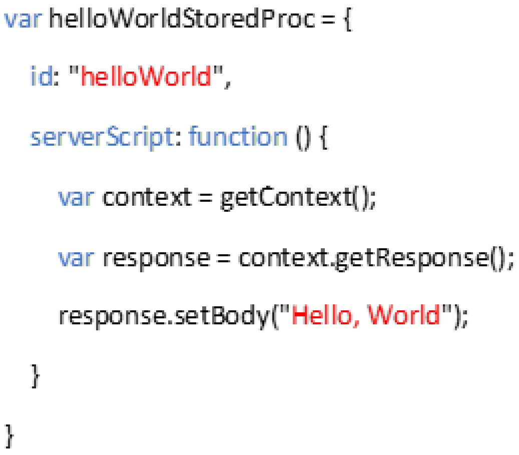
Figure 2: Sample Stored Procedure



```
CREATE TRIGGER  Test_Trigger
ON  emp
FOR
INSERT,UPDATE,DELETE
AS
PRINT 'you can not INSERT,UPDATE and DELETE this table'
ROLLBACK;
```

Figure 3: Sample Database Trigger

In contrast, NoSQL databases implement their own version of Triggers and Stored Procedures. In the case of Azure Cosmos DB, Stored Procedures and Triggers are written using the Cosmos DB JavaScript Server-side SDK.



```
var helloWorldStoredProc = {
   id: "helloWorld",
   serverScript: function () {
      var context = getContext();
      var response = context.getResponse();
      response.setBody("Hello, World");
   }
}
```

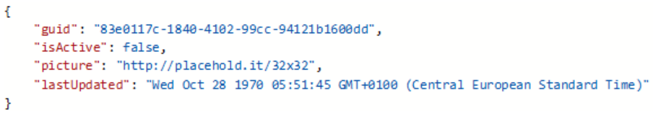
Figure 4: Cosmos DB Stored Procedure

On the other hand, to handle OCC from code, Cosmos DB implements a series of options that upon saving of the document check the _etag value in the document, relaying back an error if the current _etag and the _etag found in the Database are not the same. This mechanism allows the developer to then handle any concurrency issues accordingly.

The following is an example of how to edit a document in a Cosmos DB. Assume we are storing a simple document that represents an image Thumbnail.

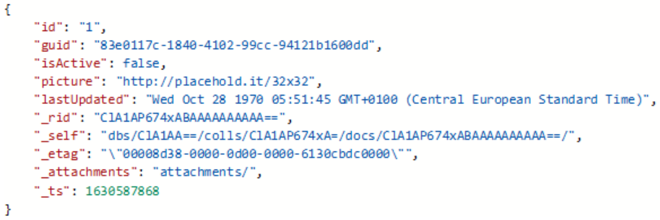
Data

Firstly, let’s look at the data in the database. The JSON data used in this instance shall be a thumbnail submission as indicated in Figure 5. Once the document is saved in Cosmos DB, the platform shall add additional properties to our JSON, as indicated in Figure 6



```
{
    "guid": "83e0117c-1840-4102-99cc-94121b1600dd",
    "isActive": false,
    "picture": "http://placehold.it/32x32",
    "lastUpdated": "Wed Oct 28 1970 05:51:45 GMT+0100 (Central European Standard Time)"
}
```

Figure 5: JSON Data



```
{
    "id": "1",
    "guid": "83e0117c-1840-4102-99cc-94121b1600dd",
    "isActive": false,
    "picture": "http://placehold.it/32x32",
    "lastUpdated": "Wed Oct 28 1970 05:51:45 GMT+0100 (Central European Standard Time)",
    "_rid": "C1A1AP674xABAAAAAAAAAA==",
    "_self": "dbs/C1A1AA==/colls/C1A1AP674xA=/docs/C1A1AP674xABAAAAAAAAAA==/",
    "_etag": "\"00008d38-0000-0d00-0000-6130cbdc0000\"",
    "_attachments": "attachments/",
    "_ts": 1630587868
}
```

Figure 6 Cosmos DB Document

One should notice that in the document saved in Cosmos DB, we now have the _etag property, which Cosmos DB will use to track any changes done to the data.

Code

Secondly, we'll look at the code to insert/update our document from an Azure Function. In this sample, we are initializing a Cosmos Client and reading the body of the HTTP Request. We then get the document from the database and update the last updated property to the current date and time. Finally, we upload or insert the record accordingly.

```
using (dbClient = new CosmosClient(Cosmo
{
    var database = await dbClient.Create
    var container = await database.Datab

    string requestBody;
    using (StreamReader streamReader = r
    {
        requestBody = await streamReader
    }
    var data = JsonConvert.DeserializeOb

    var dbData = await container.Contair
    dbData.Resource.lastUpdated = DateTi

    await container.Container.UpsertIten
}
```

HTTPTrigger_No_OCC.cs hosted with ❤ by GitHub view raw

The above is fine if we are sure that the API with the same data can't be triggered more than once at the same time. However, in a distributed environment this can't be ensured. To implement OCC, we need to add: `ItemRequestOptions` to our `UpsertItemAsync` method.

```
using (dbClient = new CosmosClient(Cosmc
{
    var database = await dbClient.Create
    var container = await database.Datab

    string requestBody;
    using (StreamReader streamReader = r
    {
        requestBody = await streamReader
    }
    var data = JsonConvert.DeserializeOb

    var dbData = await container.Contair
    dbData.Resource.lastUpdated = DateTi

    var requestOptions = new ItemRequest
```

```

        await container.Container.UpsertIte
    }
```

HTTPTrigger_OCC.cs hosted with ❤ by GitHub view raw

If there is a case where the data is not consistent, the CosmosException will have an HTTP status code of 412 PreconditionFailed. Once this is given, it is up to the developer to handle that record. In some instances one would retry the update, but in other instances one could just throw back the error to the invoker and leave the invoker to handle it.

Conclusion

In reality, Concurrency Control may not be the first thing to come to mind when updating the same record from multiple locations. However, it becomes an issue if not handled. Whilst NoSql databases are not traditionally used for transactional queries, if needs be one can see that implementing OCC does not have too much of an effect on the architecture of the solution, as all the mechanisms are already in place within the Cosmos DB API.

Subscribe to our RSS feed

Related articles

Building Landing Zones with CARML

A landing zone provides a foundation for deploying workloads in Azure while adhering to organizational policies and standards. These days, many companies are investing in building up Landing Zones in order to provide standardization, consistency, security, governance, compliancy and more to their future workloads. Let's take a look at some…

Avoiding Concurrency Issues with Agents in Functional C#

Ah, concurrency! That pesky little devil that's always lurking around the corner, waiting to trip us up when we least expect it. But fear not, dear reader, for we have a secret weapon in our functional programming arsenal: agents!

Exploring Azure Defender for APIs

Microsoft Defender for Cloud has announced a new component and it's worth looking into. The new solution is called Azure Defender for APIs and brings security insights, ML-based detections, and unauthenticated assessment in the APIs that are exposed via Azure API Management. Codit and Microsoft worked together to shape the…

Language

English

Privacy Policy
Terms & Conditions