INTRODUCTION TO CLOUD DATABASE : WINDOWS AZURE TABLE

Prepared By:

Oky Purwantiningsih

Poly Kinya Muriithi

Table of Contents

1. PREFACE

The'cloud' is one of the most ambiguous terms in the IT sector. The only agreeable aspect is that you need the internet and without it the cloud concept does not exist. Some of the many definitions include: "The definition of the term 'cloud' can be referred to as various services where information and files are kept on servers connected to the internet". Also, "anything that can be offered as a service for which you don't need to bother about how it's implemented and maintained"

One of the major reasons for using the cloud is the optimization on cost due to the offload of the purchase of hardware and software on the vendors. Nowadays the trend of moving towards cloud computing has raise the need to have database runs in the same architecture. Companies that already have their application runs on the cloud would want their application to be more integrated. In which case they will also need their data to be on the cloud. Databases that run on a cloud computing platform are called Cloud Database. There are two deployment models to run database on the cloud [1]:

- Virtual machine: user purchase virtual machine instances for a period of time and run a database on these machines. Oracle uses this method for Oracle Database 11g Enterprise Edition which runs on Amazon EC2 cloud computing platform.
- Database as a service: cloud computing platform offers database as a service. Database service provider takes the responsibility of installing and managing the database. In this case, users do not need to think about licenses, server requirement, maintenance, etc. They only need to pay based on their usage.For example, Amazon Web Services provides three cloud database services: SimpleDB, MySQL, and DynamoDB.

There are two types of cloud database, NoSQL based and SQL based. In this document, we will discuss one of NoSQL based cloud database offered as a storage service on Windows Azure: Windows Azure Table. We will discuss in details about its storage structure, how we can create, insert and delete data and how we can use it within our application. An example of its implementation in C# will be given in part 6. In part 7 we will compare Windows Azure Table with other SQL-based and NoSQL-based cloud database. We will also discuss in what scenario Azure Table will be the right database to use.

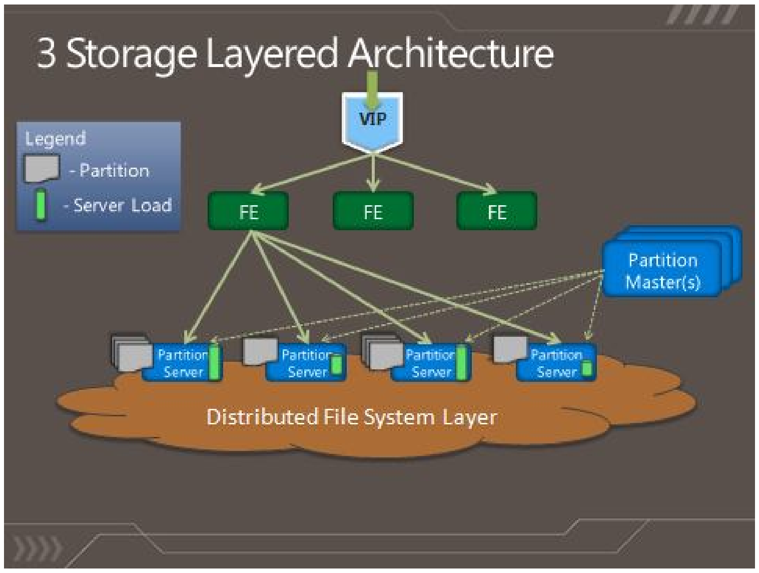
2. WHAT IS WINDOWS AZURE AND WINDOWS AZURE STORAGE

Windows Azure is a collection of virtual Microsoft operating systems that can run your web applications and services in the cloud. It's a cloud computing platform and infrastructure released by Microsoft in 2010. It supports building, deploying and managing applications and services through a global network of datacenters managed by Microsoft. It offers both platform as a service and infrastructure as a service. With Windows Azure we can build applications in any language, tool, or framework. We can also integrate non cloud application with cloud application in Windows Azure.

One of the services offered by Windows Azure is Windows Azure Storage. Since it is stored on the cloud, applications can easily access and store data from anywhere in the world via authenticated HTTP or HTTPS call. Windows Azure provides three types of storage services:

- Queue: a service for storing large numbers of messages. Queue storage is usually used to create a backlog of work to process asynchronously or passing messages from Windows Azure Web role to Windows Azure Worker role.
- Blob (*Binary Large Objects*): a service for storing large amounts of unstructured data. It is commonly used for streaming video and audio, storing files for distributed access, performing secure backup and disaster recovery.
- Table: a service for storing large amounts of structured data. As mentioned before, Table storage is built with NoSQL approach which is efficient to store structured and non-relational data. It is commonly used for storing data that doesn't require complex joins, foreign keys, or stored procedure.

Windows Azure Storage Architecture



As shown in the picture above, there are 3 layers representing Windows Azure Storage Architecture[6]
:

1. Front End (FE) Layer: takes the incoming requests, authenticates and authorizes the requests, and then routes them to a partition server in the Partition Layer
2. Partition Layer: manages the partitioning of all data objects in the system. Each object has partition key which defines what partition the object belongs to. Each partition is served by one Partition Server. This layer also provides automatic load balancing of partitions across the server.
3. Distributed and Replicated File System (DFS) layer: is where the data actually stored and in charge of distributing and replicating the data across many servers to keep it durable (geo-replication). All DFS server can be accessed from any partition server.

3. WINDOWS AZURE STORAGE ACCOUNT

Storage account enables one to manage the azure storage services. These accounts are different from hosted services because Azure Storage is in a sense one big data store managed by Microsoft. It is possible to have multiple accounts therefore various applications can be stored in separate accounts.

To access the storage account you need a key, and each account has a unique key, thus ensuring only applicants with the key can access the account. In case of security risk or compromise of the account thee key can be regenerated.

To store files and data in the Blob, Table, and Queue services in Windows Azure, you must create a storage account in the geographic region where you want to store the data. A storage account can contain up to 100 TB of blob, table, and queue data. You can create up to five storage accounts for each Windows Azure subscription.

Windows Azure provides storage as a cloud storage service as part of the Windows Azure Platform, scalable, cost effective cloud storage.

The following table describes the scalability targets for storage accounts, based on when they were created and the level of redundancy chosen:

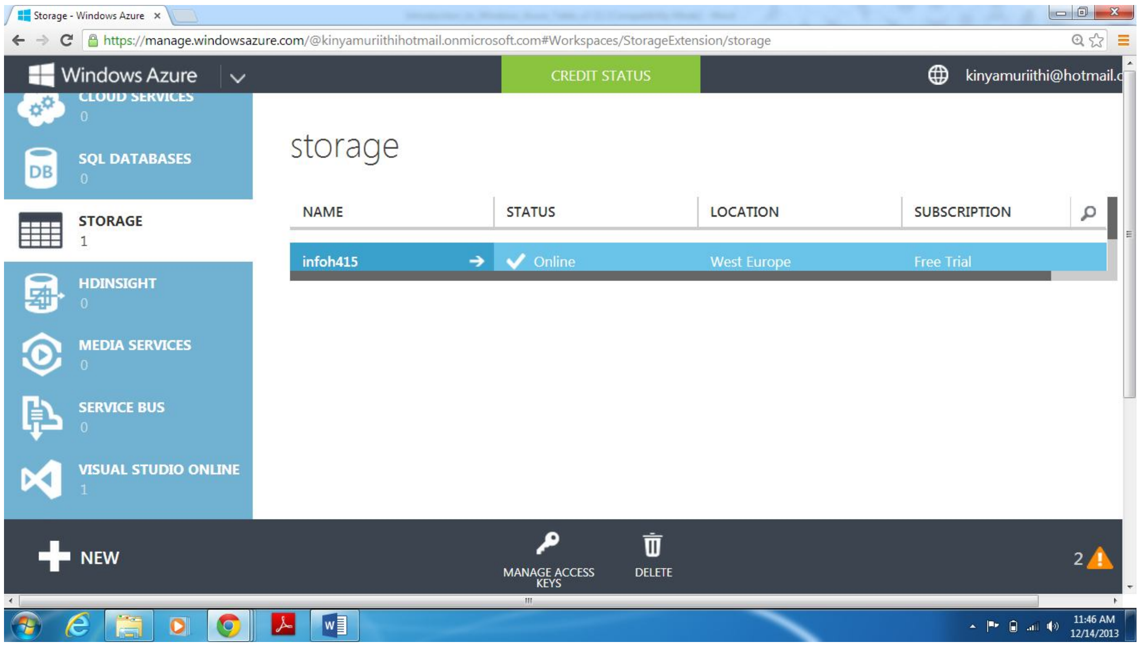
Storage Account Creation Date	**Total Account Capacity**	**Total Transaction Rate (assuming 1KB object size)**	**Total Bandwidth for a Geo-Redundant Storage Account**	**Total Bandwidth for a Locally Redundant Storage Account**
On or before June 7, 2012	100 TB	Up to 5,000 transactions per second	*Ingress+Egress: Up to 3 gigabits per second	*Ingress+Egress: Up to 3 gigabits per second
June 8, 2012 or later	200 TB	Up to 20,000 transactions per second	*Ingress: Up to 5 gigabits per second *Egress: Up to 10 gigabits per second	*Ingress: Up to 10 gigabits per second *Egress: Up to 15 gigabits per second

* Ingress refers to all data (requests) being sent to a storage account.
* Egress refers to all data (responses) being received from a storage account.

How to Create a Storage Account

Creating a storage account is easy, and it involves a series of steps including:

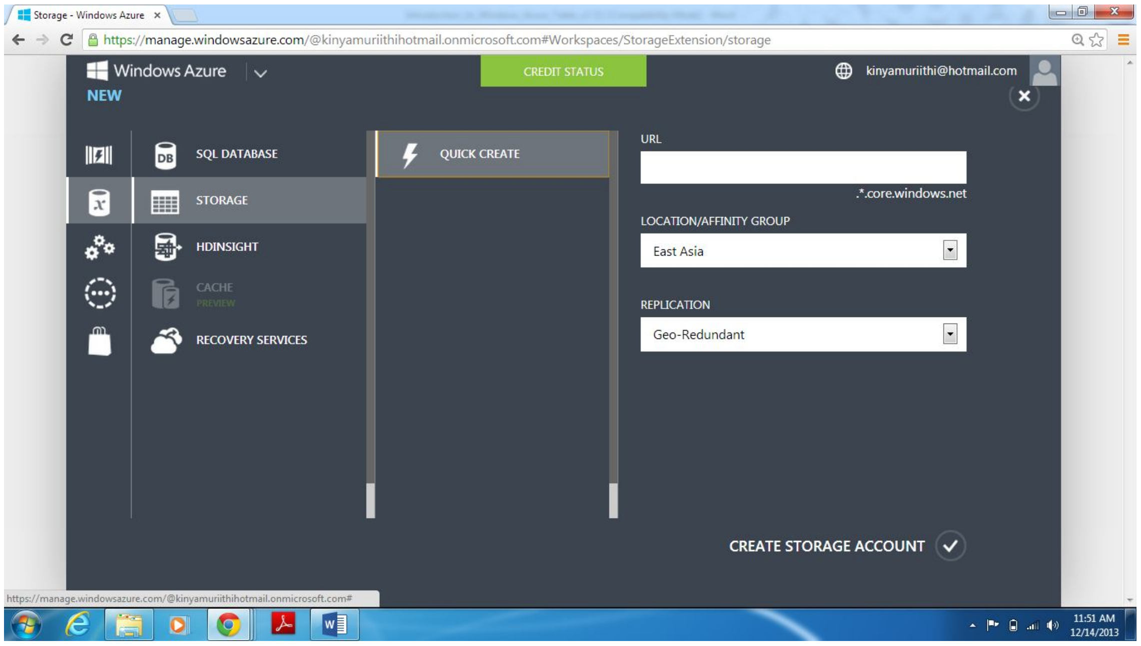
1. Firstly, subscribe Windows Azure by creating a Windows Azure Account.
2. Log in to the Windows Azure platform management portal account. This is where you manage the Azure subscription.
3. On the open window click on storage in the menu. Click on the **NEW** command in the app bar.



storage

NAME	STATUS	LOCATION	SUBSCRIPTION
infoh415	Online	West Europe	Free Trial

4. On the new window click on QUICK CREATE



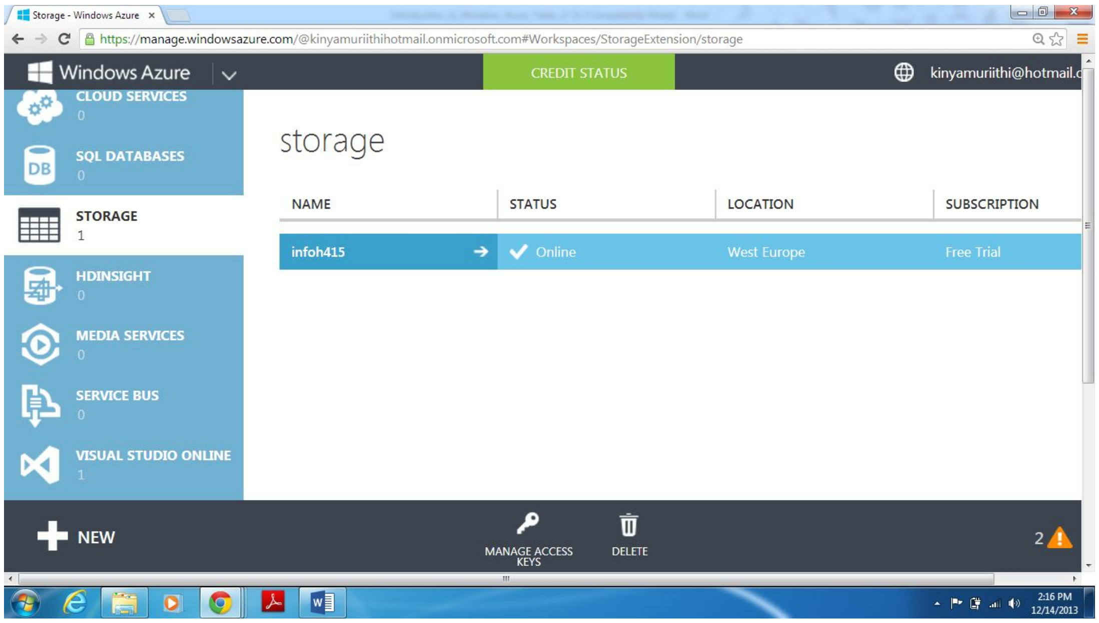
URL: .core.windows.net

LOCATION/AFFINITY GROUP: East Asia

REPLICATION: Geo-Redundant

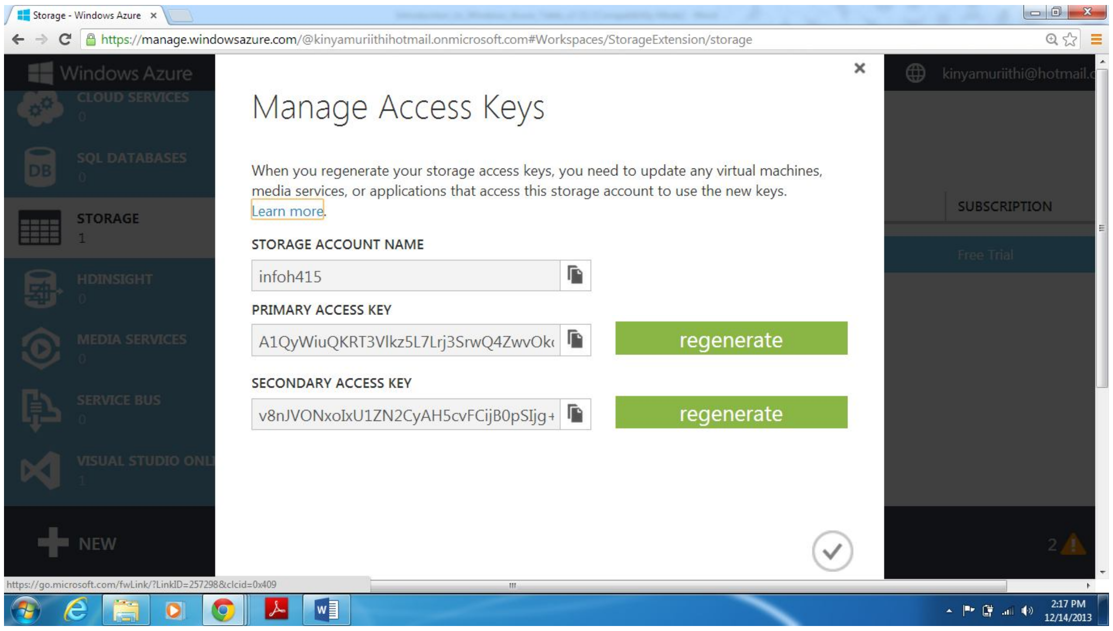
5. Choose a name and fill it in the URL, to use in the URI for the storage account.
6. Choose a location/Region in which to locate the storage. If you will be using storage from your Windows Azure application, select the same region where you will deploy your application. Always choose the same location for your storage account and your hosted services, which you can also do in the Developer Portal. This allows the computation to have high bandwidth and low latency to storage, and the bandwidth is free between computation and storage in the same location.

7. In Replication, select the level of replication that you desire for your storage account. By default, replication is set to Geo-Redundant. With geo-redundant replication, your storage account and all data in it is copied to a secondary location in the event of a major disaster in the primary location. Windows Azure assigns a secondary location in the same region, which cannot be changed. In case of a failure, the secondary location becomes the primary location for the storage account, and your data is replicated to a new secondary location.
8. Click on the CREATE STORAGE ACCOUNT command in the app bar. The account is created and ready for use.



NAME	STATUS	LOCATION	SUBSCRIPTION
infoh415	✓ Online	West Europe	Free Trial

9. Once the account is created integration with an application is easy all you need to get is the name of the storage account and the access keys and everything is ready to go. To get the name and access key click on the MANAGE ACCESS KEYS command on the app bar at the bottom of the screen.



Manage Access Keys

When you regenerate your storage access keys, you need to update any virtual machines, media services, or applications that access this storage account to use the new keys. Learn more.

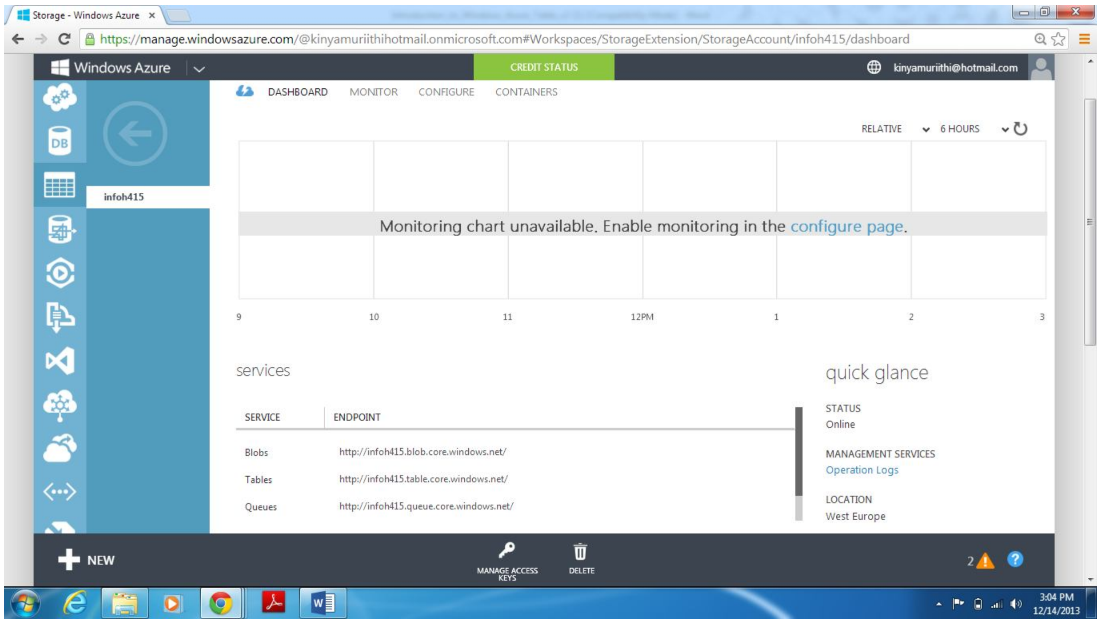
STORAGE ACCOUNT NAME
infoh415

PRIMARY ACCESS KEY
A1QyWiuQKRT3Vlkz5L7Lrj3SrwQ4ZwvOk
regenerate

SECONDARY ACCESS KEY
v8nJVONxoIxU1ZN2CyAH5cvFCijB0pSIjg+
regenerate

Once the account is created you can use some features like:

DASHBOARD: Allow you to choose up to six metrics to plot on the metrics chart from nine available metrics. For each service (Blob, Table, and Queue), the Availability, Success Percentage, and Total Requests metrics are available. The metrics available on the dashboard are the same for minimal or verbose monitoring.


Storage - Windows Azure
https://manage.windowsazure.com/@kinyamuriithihotmail.onmicrosoft.com#Workspaces/StorageExtension/StorageAccount/infoh415/dashboard
Windows Azure
CREDIT STATUS
kinyamuriithi@hotmail.com
infoh415
DASHBOARD
MONITOR
CONFIGURE
CONTAINERS
RELATIVE
6 HOURS
9
10
11
12PM
1
2
3
NEW
MANAGE ACCESS KEYS
DELETE

Monitoring chart unavailable. Enable monitoring in the configure page.

services

SERVICE	ENDPOINT
Blobs	http://infoh415.blob.core.windows.net/
Tables	http://infoh415.table.core.windows.net/
Queues	http://infoh415.queue.core.windows.net/

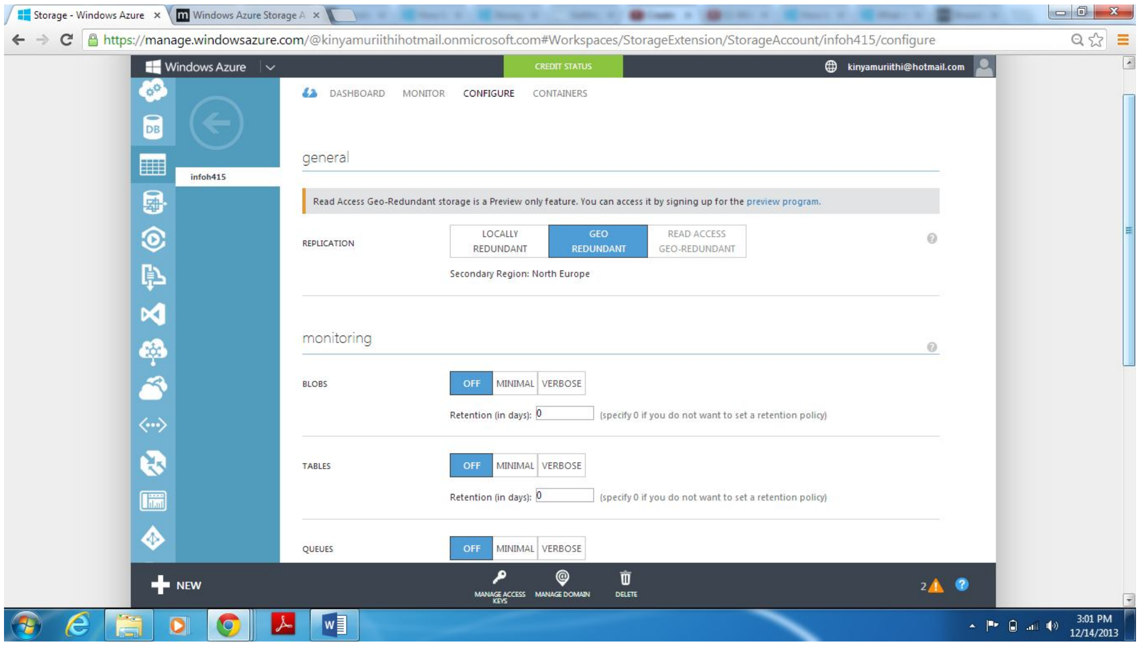
quick glance

STATUS
Online

MANAGEMENT SERVICES
Operation Logs

LOCATION
West Europe

CONFIGURE: To configure and monitor diagnostic logging for debugging and performance tuning, with the flexibility to control granularity of the logs.



general

Read Access Geo-Redundant storage is a Preview only feature. You can access it by signing up for the preview program.

REPLICATION: LOCALLY REDUNDANT | [✓ GEO REDUNDANT] | READ ACCESS GEO-REDUNDANT

Secondary Region: North Europe

monitoring

BLOBS: [✓ OFF] | MINIMAL | VERBOSE

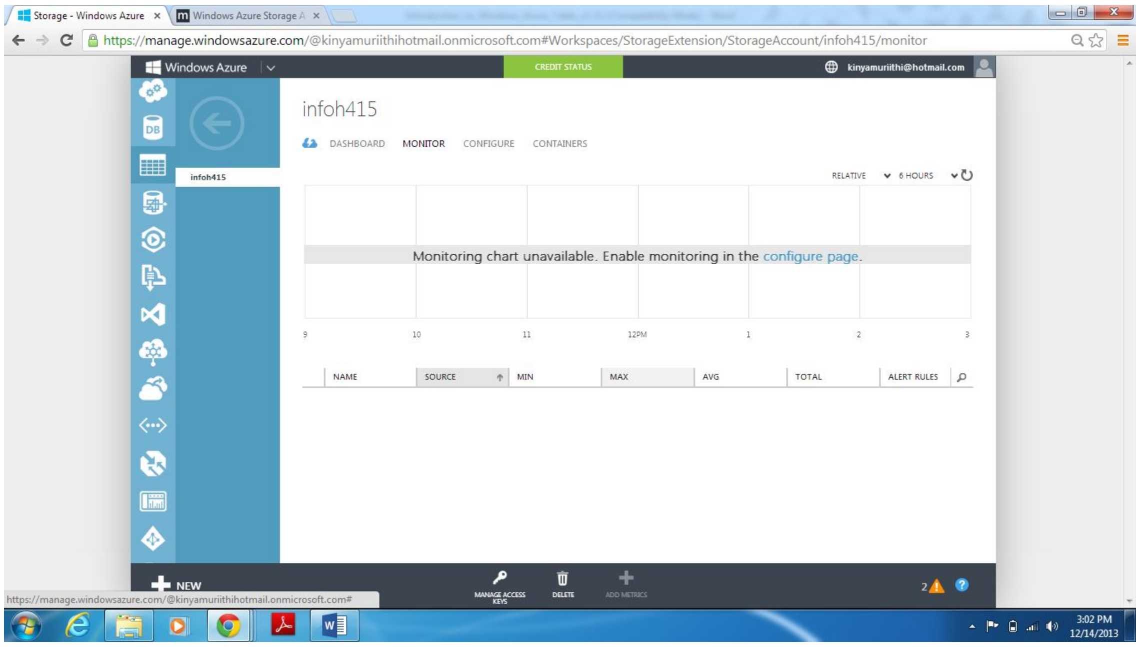
Retention (in days): 0 (specify 0 if you do not want to set a retention policy)

TABLES: [✓ OFF] | MINIMAL | VERBOSE

Retention (in days): 0 (specify 0 if you do not want to set a retention policy)

QUEUES: [✓ OFF] | MINIMAL | VERBOSE

Once the configuration is complete you complete you can use the MONITORING to pick and choose exactly which matrix you want to monitor or track.

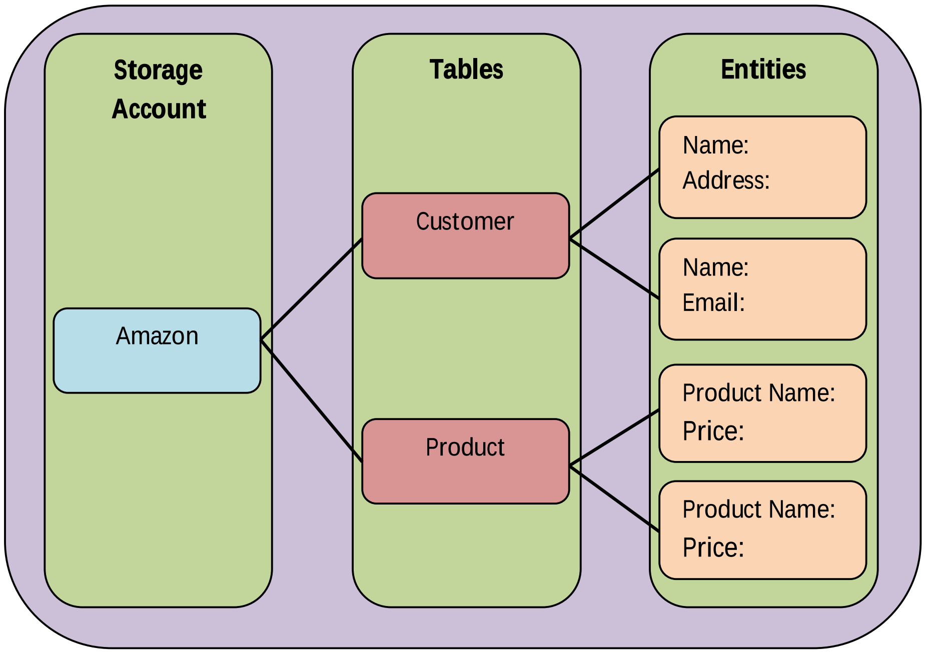


Monitoring chart unavailable. Enable monitoring in the configure page.

4. WINDOWS AZURE TABLE OVERVIEW

The term 'table' used in Azure table is a bit misleading in a sense it makes people think of it as 'table' in traditional database. To have a better understanding on Windows Azure Table, the following are some of terms related to Azure Table[2]:

- Table: A table is a container or collection of entities. A table has no fixed schema, therefore two different entities in a table can have different properties which means a single table can contain entities that have different sets of properties.
- Entity: An entity is a set of properties, similar to a database row. An entity can be up to 1MB in size.
- Properties: A property is a name-value pair. These properties are typed per entity, which means that you can have the same property name in another entity , but with a different data type. Each entity can include up to 252 properties to store data. Each entity also has 3 system properties that specify a partition key, a row key, and a timestamp. Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations. An entity's row key is its unique identifier within a partition. Azure Table properties can have the following data type: byte, bool, DateTime, double, Guid, Int32 or Int, Int64 or long, String.



Every entity has 3 fixed system properties:

- PartitionKey: is the key which define where the system will distribute the entities over different storage nodes, ie partition the entity belongs. Partition Key is a mandatory property and is of String type. We have to define the partition key for each entity.

- RowKey: is a unique id of the entity within the same partition. Two entities can have the same Row Key as long as it has different partition key. Partition Key combined with Row Key is like composite key in traditional database and it identifies an entity in a table. Similar to Partition Key, Row Key is also a mandatory property and is of String type. We have to define the Row Key for each entity.
- Timestamp: is a property which is automatically filled in by the system every time we insert an entity. It is a read only value maintained by the system for versioning, used to maintain optimistic concurrency. By Default, every Update and Delete request send an ETag using If-Match condition. The operation will fail if the Timestamp property value differs from the one in If-Match header.

Accessing Azure Table

Azure Table Storage uses a REST APA consisting of table operations – to manage tables in the store and entity operations – to manipulate data in the table. This means that it can access and manipulate data using HTTP requests. The URL determines what data structure you work with while the HTTP verb determines the operations you perform. Due to the use of this API any platform that supports the HTTP protocol and understand XML can talk to Azure Table Storage

Indexing in Azure Table

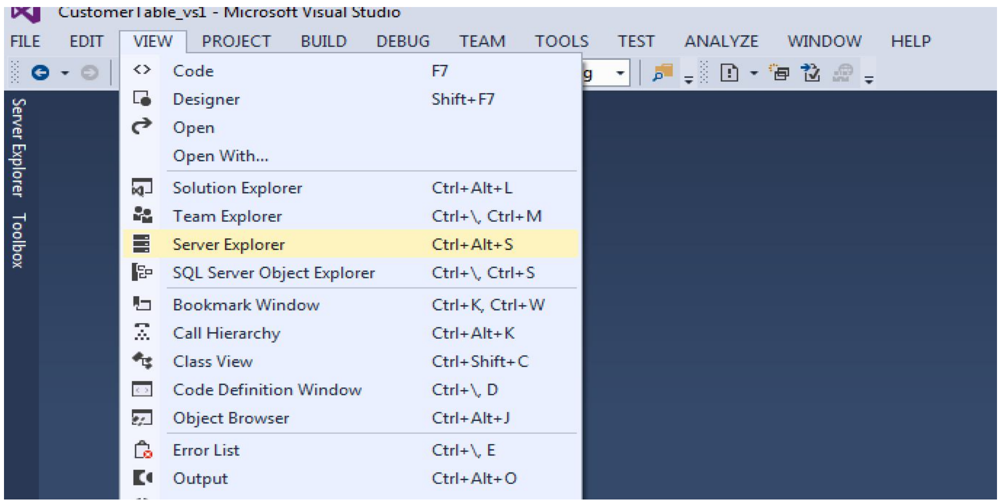
Windows Azure Table provides one clustered index based on Partition Key and Row Key. Query results are always sorted by these properties in ascending order. We cannot define our own index based on the other properties. Since Partition Key and Row Key is the only index available, both properties also define how fast a query can be executed. Query on the same Partition Key will be faster than query on different Partition Key. When defining what data we will use as Partition Key and Row Key, we have to consider what kind of data will be most often requested. Key selection is the first most important aspects to consider when deciding to use table storage.

5. WINDOWS AZURE TABLE EXPLORER

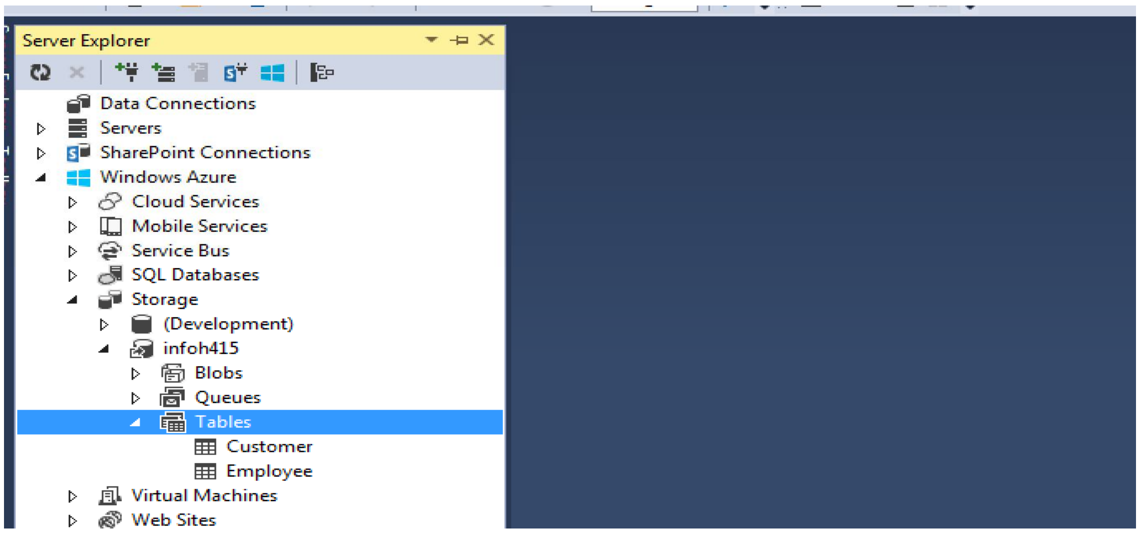
Access to Azure Table Storage is accomplished either via REST API or Storage Client Library provided with the Windows Azure SDK. Using the REST API allows client applications to communicate and use data from Table Storage without having detailed and specific knowledge of an Azure API, but it is more complex and difficult to work with. The Storage Client Library (which leverages LINQ to Objects) provides a layer of convenience but requires the application reference the Storage Client Library APIs,to do this, both of which hide the underlying HTTP requests. REST and the Storage Client Library incur a learning curve that is typically not there when using SQL Azure. .NET or otherwise, is also free to make these requests directly. O data can also be used to query this tables. In this chapter we will look at querying using Visual Studio (ADO.NET Data Services).

You can view blob, queue, and table data from your storage accounts for Windows Azure using the Windows Azure tool for Microsoft visual studio. By using the Windows Azure Storage node in **Server Explorer**, you can display data from your local storage emulator account and also from storage accounts that you've created for Windows Azure.

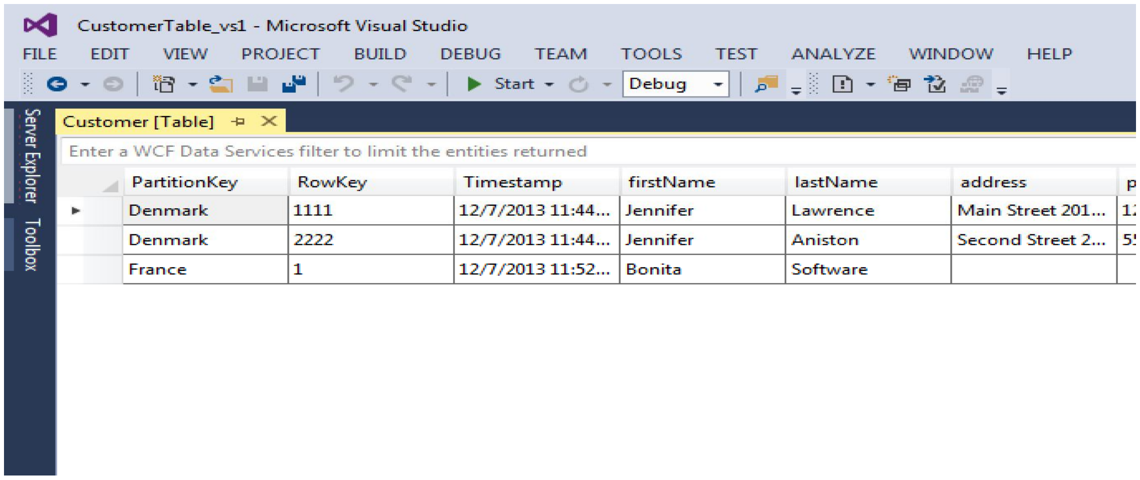
Server Explorer in Visual Studio is displayed by, choosing **View**, then **Server Explorer**from the menu bar.


CustomerTable_vs1 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Server Explorer Toolbox
Code F7
Designer Shift+F7
Open
Open With...
Solution Explorer Ctrl+Alt+L
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
SQL Server Object Explorer Ctrl+\, Ctrl+S
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O

Any Windows Azure storage accounts to which you've connected appear below the **Windows Azure Storage** node. If your storage account doesn't appear, you can add it.



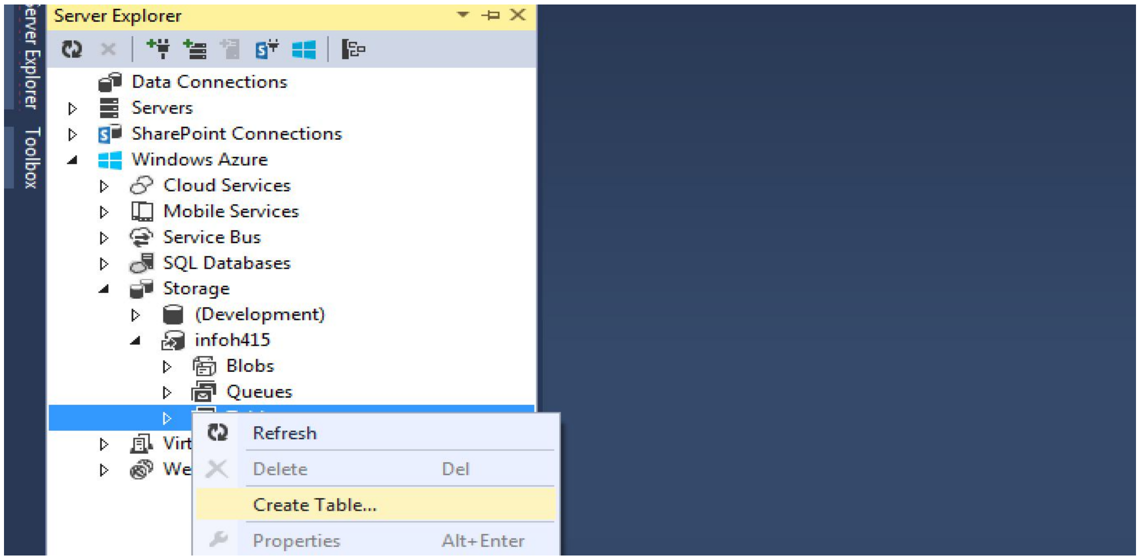
Expand the **Tables** node to see a list of tables for the storage account. To display the data in a table, open the shortcut menu for a table and then choose **View Table**. The table is organized by entities (shown in rows) and properties (shown in columns).


CustomerTable_vs1 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Start
Debug
Server Explorer
Toolbox
Customer [Table]
Enter a WCF Data Services filter to limit the entities returned

PartitionKey	RowKey	Timestamp	firstName	lastName	address	p
Denmark	1111	12/7/2013 11:44...	Jennifer	Lawrence	Main Street 201...	1:
Denmark	2222	12/7/2013 11:44...	Jennifer	Aniston	Second Street 2...	5!
France	1	12/7/2013 11:52...	Bonita	Software		

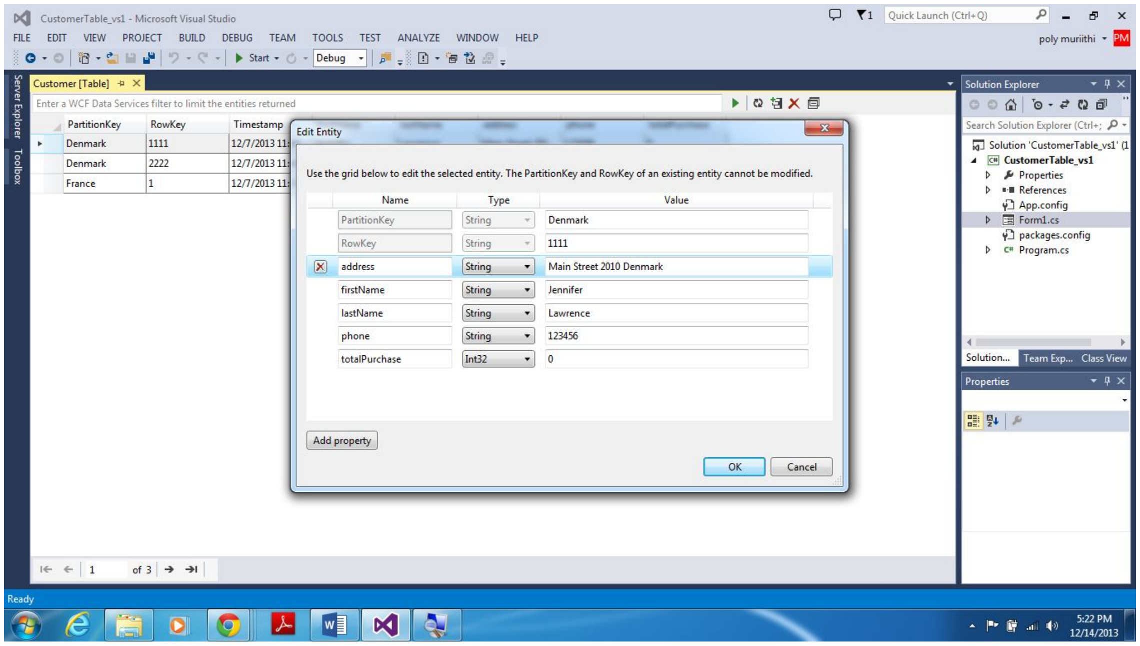
5.1. CREATE TABLE

You can create tables by using Server Explorer. To create a table, open the shortcut menu for the Tables node, and then choose **CREATE TABLE**.



5.2. EDIT ENTITY

The server explorer allows you to edit a table manually without the use of any code. You can edit table data by opening the shortcut menu for an entity (a single row) or a property (a single cell) and then choosing **Edit.**


CustomerTable_vs1 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Start Debug
Customer [Table]
Edit Entity
Add property
OK
Cancel
Solution Explorer
Properties
Ready

Enter a WCF Data Services filter to limit the entities returned

PartitionKey	RowKey	Timestamp
Denmark	1111	12/7/2013 11:
Denmark	2222	12/7/2013 11:
France	1	12/7/2013 11:

Use the grid below to edit the selected entity. The PartitionKey and RowKey of an existing entity cannot be modified.

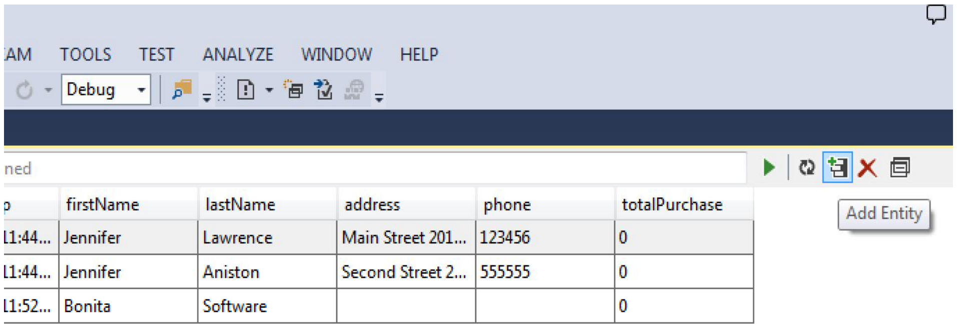
Name	Type	Value
PartitionKey	String	Denmark
RowKey	String	1111
address	String	Main Street 2010 Denmark
firstName	String	Jennifer
lastName	String	Lawrence
phone	String	123456
totalPurchase	Int32	0

Entities in a single table aren't required to have the same set of properties (columns).The following restrictions on viewing and editing table data need to be observed:

- Binary data (type byte []), can store it in a table but it is not possible to view or edit it.
- **PartitionKey** or **RowKey** values cannot be edited, as this operation is notsupport by table storage in Windows Azure.
- **Timestamp** is not an eligible name for a property since, Windows Azure Storage services use a property with that name.
- DateTime value, must follow a format that's appropriate to the region and language settings of your computer

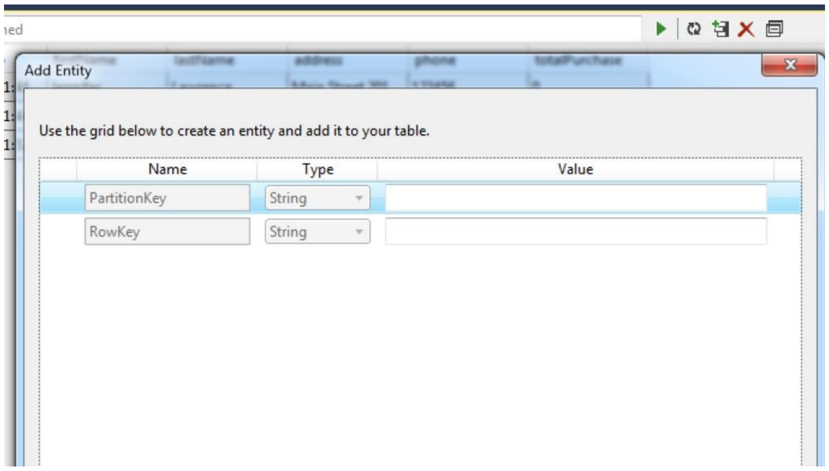
5.3. ADD ENTITY

Entities (rows) can also be added to the table by choosing the **Add Entity** button.



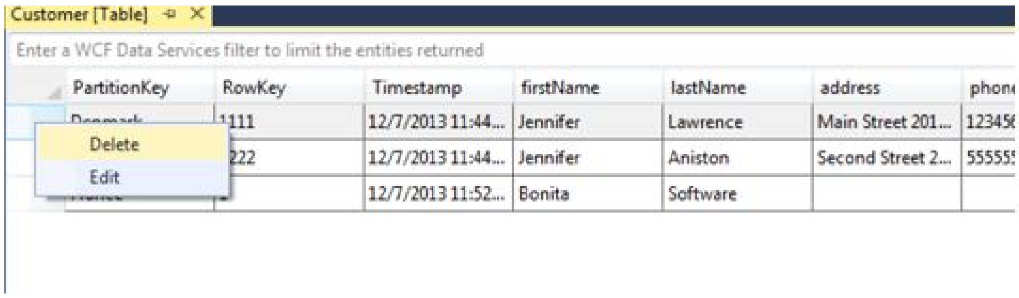
p	firstName	lastName	address	phone	totalPurchase
11:44...	Jennifer	Lawrence	Main Street 201...	123456	0
11:44...	Jennifer	Aniston	Second Street 2...	555555	0
11:52...	Bonita	Software			0

In the **Add Entity** dialog box, enter the values of the **PartitionKey** and **RowKey** properties. Entering of these values must be done very carefully because once the values are entered and the dialog box closed no changes can be made.


Add Entity
Use the grid below to create an entity and add it to your table.
Name
Type
Value
PartitionKey
String
RowKey
String

5.4. DELETE ENTITY

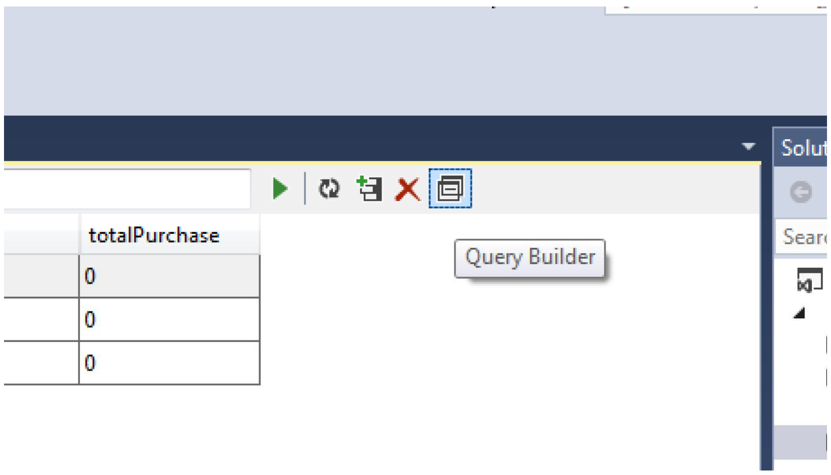
To delete a table permanently, open its shortcut menu, and then choose **Delete**.


Customer [Table]
Enter a WCF Data Services filter to limit the entities returned
Delete
Edit

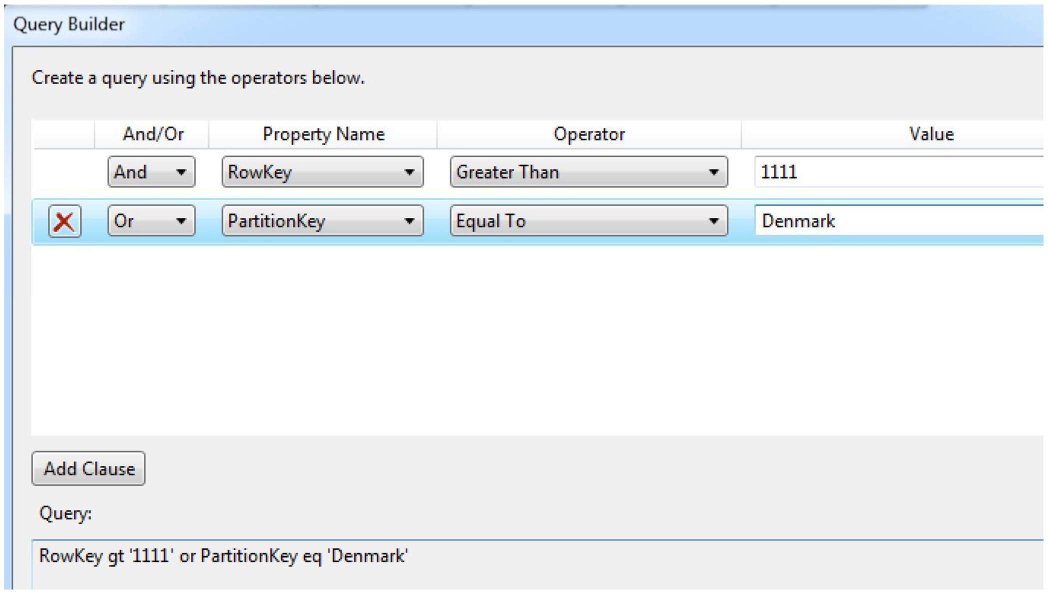
PartitionKey	RowKey	Timestamp	firstName	lastName	address	phon
	1111	12/7/2013 11:44...	Jennifer	Lawrence	Main Street 201...	12345
	222	12/7/2013 11:44...	Jennifer	Aniston	Second Street 2...	55555
		12/7/2013 11:52...	Bonita	Software		

5.5. FILTER ENTITY / QUERY

You can customize the set of entities that are shown from a table if you use the query builder. To open the query builder, open a table for viewing, and then choose the rightmost button on the table view's toolbar.



The **Query Builder** dialog box opens.



And/Or	Property Name	Operator	Value
And	RowKey	Greater Than	1111
Or	PartitionKey	Equal To	Denmark

RowKey gt '1111' or PartitionKey eq 'Denmark'

Once the query build andthe dialog box is closed, the resulting text form of the query appears in a text box as a WCF Data Services filter. To run the query, choose the green triangle icon.

5.5.1. Comparison Operators

The following logical operators are supported for all property types:

Logical operator	Description	Example filter string
eq	Equal	City eq 'Denmark'
gt	Greater than	Price gt 20
ge	Greater than or equal to	Price ge 10
lt	Less than	Price lt 20
le	Less than or equal	Price le 100
ne	Not equal	City ne 'Paris'
and	And	Price le 200 and Price gt 3.5
or	Or	Price le 3.5 or Price gt 200
not	Not	not is Available

When constructing a filter string, the following rules are important:

- It is not possible to compare a property to a dynamic value; one side of the expression must be a constant. Therefore the logical operators are used to compare a property to a value
- All parts of the filter string are case-sensitive.
- The constant value must be of the same data type as the property in order for the filter to return valid results.

5.5.2. *Filtering on String Properties*

In general, when you filter on string properties, enclose the string constant in single quotation marks. Or enclose each filter expression in parentheses

```
(PartitionKey eq 'Denmark' ) and (RowKey eq '1')
```

However Table service does not support wildcard queries, and they are not supported in the Table Designer either. However, you can perform prefix matching by using comparison operators on the desired prefix.

```
LastName ge 'A' and LastName lt 'B'
```

5.5.3. *Filtering on Numeric Properties*

To filter on an integer or floating-point number, specify the number without quotation marks, just use it in SQL numeric filtering.
This example returns all entities with a distance property whose value is greater than 100:

```
Distance gt 100
```

5.5.4. *Filtering on Boolean Properties*

To filter on a Boolean value, specify **true** or **false** without quotation marks. Returns all entities where the IsActive property is set to **true:**

```
IsExpensive eq true
```

You can also write this filter expression without the logical operator. The Table service will also return all entities where IsExpensive is **true**:

```
IsExpensive
```

To return all entities where IsActive is **false**, you can use the **not** operator:

```
Not IsExpensive
```

5.5.5. *Filtering on DateTime Properties*

To filter on a DateTime value, specify the **datetime** keyword, followed by the date/time constant in single quotation marks. The date/time constant must be in combined UTC format.

```
Denmark eq datetime'2013-12-07T00:00:00Z'
```

6. WINDOWS AZURE TABLE IMPLEMENTATION IN C#

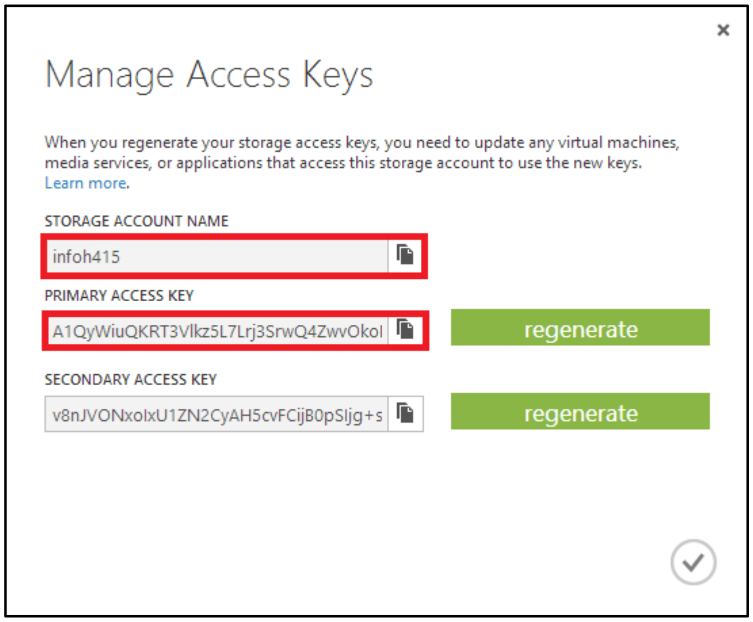
We can access Windows Azure Table through application written in .NET, Node.js, PHP, Java, Phyton or Ruby. In This document we will give examples on how to create, manipulate, and delete Windows Azure Table Storage Service in C#.

6.1. Create a storage connection

The connection to Windows Azure Table can be maintained through the .NET configuration system (web.config or app.config). Here we store the account name and account key of our Windows Azure Storage account in <*appsetting*>element as the following:

```
<configuration>
<startup>
<supportedRuntimeversion="v4.0"sku=".NETFramework,Version=v4.5" />
</startup>
<appSettings>
<addkey="AccountName"value="infoh415"/>
<addkey="AccountKey" value="A1QyWiuQKRT3VlSrsKvnFaGGG15dA=="/>
</appSettings>
</configuration>
```

Account name is the name of our Windows Azure Storage account and account key is the access key provided in the portal.


Manage Access Keys

When you regenerate your storage access keys, you need to update any virtual machines, media services, or applications that access this storage account to use the new keys. Learn more.

STORAGE ACCOUNT NAME

infoh415

PRIMARY ACCESS KEY

A1QyWiuQKRT3Vlkz5L7Lrj3SrwQ4ZwvOkol

SECONDARY ACCESS KEY

v8nJVONxolxU1ZN2CyAH5cvFCijB0pSljg+s

6.2. Programmatically Access Table Storage

Obtaining the assembly

To obtain Windows Azure Storage assembly, right click on **Solution Explorer** and choose **Manage NuGet Packages**. Search online for "WindowsAzure.storage" and click **Install**. This will install Microsoft.WindowsAzure.Storage.dll assembly in our project. We will also need to add reference to System.Configuration.dll in order to be able to retrieve the connection information we stored in web.config or app.config.

Namespaces Declaration

The following code are namespace declaration needed to access Windows Azure Storage:

```
usingMicrosoft.WindowsAzure.Storage;
usingMicrosoft.WindowsAzure.Storage.Auth;
usingMicrosoft.WindowsAzure.Storage.Table;
usingSystem.Configuration
```

Retrieving the Connection

To retrieve the connection within our application, we can use the following code:

```
CloudStorageAccountstorageAccount;
StorageCredentialscreds;

stringaccountName = ConfigurationManager.AppSettings.Get("AccountName");
stringaccountKey = ConfigurationManager.AppSettings.Get("AccountKey");

creds = newStorageCredentials(accountName, accountKey);
storageAccount = newCloudStorageAccount(creds, true);
```

All C# codes in this documents is built with Windows Form Application project and uses the above code to retrieve windows azure storage connection stored in app.config.

6.3. Create a table

The CloudTableClient object is used to get reference object for tables and entities. The following codes create a table named Customer if the table is not exists in our storage account.

```
CloudTableClienttableClient = storageAccount.CreateCloudTableClient();

CloudTable table = tableClient.GetTableReference("Customer");
table.CreateIfNotExists();
```

6.4. Add an entity to a table

To add an entity into a table, we need a class which defines the properties of our entity. In the following example, we create a class called CustomerEntity which derived from TableEntity class. We define country as Partition Key and customerNo as Row Key. We also define other attributes. These attributes must be a public property of a supported type that exposes both get and set. The class must also expose a parameter-less constructor.

```
publicclassCustomerEntity : TableEntity
    {   publicCustomerEntity(string country, stringcustomerNo)
        {
this.PartitionKey = country;
this.RowKey = customerNo;
        }

publicCustomerEntity() { }
publicstringfirstName { get; set; }
publicstringlastName { get; set; }
publicstring address { get; set; }
publicstring phone { get; set; }
publicinttotalPurchase { get; set; }
    }
```

Table operations are performed with CloudTable object and TableOperation. In the following code, *table* variable refers to Customer table storagein which we want to insert the entity and *insert* variable is the operation we want to execute.

```
CloudTable table = tableClient.GetTableReference("Customer");
CustomerEntityemp = newCustomerEntity("Belgium","1111");

emp.firstName = "Bilbo";
emp.lastName = "Baggins";
emp.address = "Shire";
emp.phone = "555-555-555";
emp.totalPurchase = Convert.ToInt32(("100");

TableOperation insert = TableOperation.Insert(emp);
table.Execute(insert);
```

6.5. Insert a batch of entities

We can insert a batch of entities in a single operation with the following constraint:

1. Update, Delete, and Insert can be done in the same single batch operation.
2. Maximum number of entities in one batch operation is 100 entities.
3. All the entities in one batch operation must have the same partition key.
4. We can perform query in batch operation, but it must be the only operation in the batch.

The following code inserts two entities in one batch operation. For batch operation, we use *TableBatchOperation* object instead of the regular *TableOperation*.

```
// Create the CloudTable object that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

//Create the batch operation.
TableBatchOperationbatchOperation = newTableBatchOperation();

//Create a customer entity
CustomerEntity emp1 = newCustomerEntity("Denmark","1111");
emp1.firstName = "Jennifer";
emp1.lastName = "Lawrence";
emp1.phone = "123456";
emp1.address = "Main Street 2010 Denmark";

//create another customer entity
CustomerEntity emp2 = newCustomerEntity("Denmark", "2222");
emp2.firstName = "Jennifer";
emp2.lastName = "Aniston";
emp2.phone = "555555";
emp2.address = "Second Street 2111 France";

// Add both customer entities to the batch insert operation.
batchOperation.Insert(emp1);
batchOperation.Insert(emp2);

// Execute the batch operation.
table.ExecuteBatch(batchOperation);
```

6.6. Retrieve all entities in a partition

To query entities in a table, we use *TableQuery* object. The following codes retrieve all entities which Partition Key is equal to “Denmark” and show the result in a data grid view component.

```
// Create the CloudTable object that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

// Construct the query operation for all customer entities where
PartitionKey="Denmark".
TableQuery<CustomerEntity> query =
newTableQuery<CustomerEntity>().Where(TableQuery.GenerateFilterCondition("P
artitionKey", QueryComparisons.Equal, "Denmark"));

// Show the query result in data grid view
IEnumerable<CustomerEntity> list = table.ExecuteQuery(query);
grdEntities.DataSource = list.ToList();
```

6.7. Retrieve a range of entities in a partition

We can also query for entities with a specific range of property. The following codes retrieve entities which have Partition Key equal to “France” and have purchase more than 20 items and show the result in a data grid view component.

```
// Create the CloudTable object that represents the "Customer" table
CloudTable table = tableClient.GetTableReference("Customer");

// Construct the query
TableQuery<CustomerEntity>rangeQuery =
newTableQuery<CustomerEntity>().Where(
TableQuery.CombineFilters(
TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal,
"France"),TableOperators.And,
TableQuery.GenerateFilterConditionForInt("totalPurchase",
QueryComparisons.GreaterThan, 20)));

// Show the query result in data grid view
IEnumerable<CustomerEntity> list = table.ExecuteQuery(rangeQuery);
grdEntities.DataSource = list.ToList();
```

6.8. Retrieve a single entity

To query for a single entity, we have to define a specific value for Partition Key and Row Key. The following codes retrieve one entity of customer who live in Denmark and have customer number 1111.

```
// Create the CloudTable object that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

// Create a query to retrieve specific record
TableQuery<CustomerEntity> query = newTableQuery<CustomerEntity>().Where(
TableQuery.CombineFilters(
TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal,
"Denmark"),TableOperators.And,
TableQuery.GenerateFilterCondition("RowKey", QueryComparisons.Equal,
"1111")));

// Show the query result in data grid view
IEnumerable<CustomerEntity> list = table.ExecuteQuery(query);
grdEntities.DataSource = list.ToList();
```

6.9. Replace an entity

Replace an entity is basically an update operation. The difference is that this operation replaces the entity in the server. In the case when the entity in the server has changed since it was retrieved then the operation will fail. This mechanism prevents an application to make changes between retrieval and update by other component or application. To override this behavior, we can use insert or replace operation which will be explained in the next section.

The following code replace customer who live in Denmark and has Customer Number 2222 and set his phone number and total purchase information.

```
// Create the CloudTable object that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

// Create a retrieve operation that takes a customer entity.
TableOperation retrieve = TableOperation.Retrieve<CustomerEntity>("Denmark",
"2222");
TableResult Result = table.Execute(retrieve);
CustomerEntityent = (CustomerEntity)Result.Result;

if (ent != null)
{       // Change the phone number.
        ent.phone = "555-555-2222";
        ent.totalPurchase = 99;

        // Create the InsertOrReplaceTableOperation
        TableOperationreplaceOp = TableOperation.Replace(ent);

        // Execute the operation.
        table.Execute(replaceOp);
        MessageBox.Show("Entity updated.");
}
else
        MessageBox.Show("Entity could not be retrieved.");
```

6.10. Insert or Replace an entity

This operation replaces an entity if it exists in the server and insert if it doesn't. This operation is useful when we're not sure whether the entity exists in the server or whether the current values stored is still relevant. Any updates that occur between retrieval and replace operation will be overwritten. The following code will insert a new entity in the table if customer 3333 who live in Belgium is not exist and will replace the entity if it already exist in the server.

```
// Create the CloudTable object that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

CustomerEntityemp = newCustomerEntity("Belgium", "3333");

emp.firstName = "Gandalf";
emp.lastName = "The Grey";
emp.address = "Rivendell 1080";
emp.phone = "534-343-3434";
emp.totalPurchase = Convert.ToInt32("600");

TableOperation insert = TableOperation.InsertOrReplace(emp);
table.Execute(insert);
```

6.11. Merge an entity

Merge operation is similar with update operation. If the entity already exists in server, this operation will update the properties. For example, if there exists a customer who live in France (Partition Key) with customer number 1111 (Row Key) whose name is Samwise Gamgee and no other information is stored, the merge operation of customer entity with the same partition key and row key and has phone number information will result in updating the phone number of the already exist customer in the server. The first name and last name information will remain unchanged. This behavior is different with Replace operation in which it will replace the whole entity with the new entity. If we use Replace operation, we will lose the first name and last name information.

```
// Create the CloudTable object that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

// Create a retrieve operation that takes a customer entity.
TableOperation retrieve = TableOperation.Retrieve<CustomerEntity>("France",
"1111");
TableResult Result = table.Execute(retrieve);
CustomerEntityent = (CustomerEntity)Result.Result;

if (ent != null)
{       // Change the phone number.
        ent.phone = "99-999-9999";

        // Create the InsertOrReplaceTableOperation
        TableOperationmergeOp = TableOperation.Merge(ent);

        // Execute the operation.
        table.Execute(mergeOp);
        MessageBox.Show("Entity Merge.");
}
else
        MessageBox.Show("Entity could not be retrieved.");
```

6.12. Insert or Merge an entity

Similar to Insert or Replace operation, this operation will insert a new entity if it does not exist in the server and will update the entity if it already does. The following code will insert customer who live in Belgium with customer number 1234. If the same customer already exists, it will update the entity.

```
// Create the CloudTable object that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

CustomerEntityemp = newCustomerEntity("Belgium", "1234");

emp.firstName = "Harry";
emp.lastName = "Potter";
emp.address = "Hogwart 43";
emp.phone = "666-666-666";
emp.totalPurchase = Convert.ToInt32("234");

TableOperation insert = TableOperation.InsertOrMerge(emp);
table.Execute(insert);
MessageBox.Show("Entity Inserted or Merged");
```

6.13. Query a subset of entity properties

We can retrieve just few properties from an entity instead of all properties. This is called projection and can improve query performance since it reduces bandwidth, especially for large entities. The following code retrieves only the first name property of each entity and show the result in a data grid view.

```
// Create the CloudTable object that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

// Construct the query, select first name
TableQuery<DynamicTableEntity>projectionQuery =
newTableQuery<DynamicTableEntity>().Select(newstring[] { "firstName" });

// Define an entity resolver to work with the entity after retrieval.
EntityResolver<string> resolver = (pk, rk, ts, props, etag)
=>props["firstName"].StringValue;

// Show the query result in data grid view
IEnumerable<string>listName = table.ExecuteQuery(projectionQuery, resolver,
null, null);
grdEntities.DataSource = listName.Select(x =>new{ Name = x }).ToList();
```

6.14. Delete an Entity

To delete an entity, we need to retrieve it first and then perform a delete operation on it. The following code will delete a customer who lives in Denmark and has customer number 1111.

```
CloudTable table = tableClient.GetTableReference("Customer");

// Create a retrieve operation that takes a customer entity.
TableOperation retrieve =
TableOperation.Retrieve<CustomerEntity>("Denmark", "1111");
TableResult Result = table.Execute(retrieve);
CustomerEntityent = (CustomerEntity)Result.Result;

if (ent != null)
{       TableOperation delete = TableOperation.Delete(ent);
        table.Execute(delete);
        MessageBox.Show("Customer Deleted!");
}
```

6.15. Delete a table

The following code will delete table Customer in the storage account if the table exists.

```
//Create the CloudTable that represents the "Customer" table.
CloudTable table = tableClient.GetTableReference("Customer");

// Delete the table it if exists.
table.DeleteIfExists();
MessageBox.Show("Table Deleted");
```

7. WINDOWS AZURE TABLE VS OTHER DATABASE

This section compares Windows Azure Table with other database. The following table compares the noSQL based Windows Azure Table with SQL Server Database:

Comparison Criteria	Windows Azure Table Storage	SQL Server
Data relationships	**No.** Windows Azure Table Storage does not provide a way to represent relationships between data.	**Yes.** Similar to SQL Server, relationship between tables are maintained with foreign key.
Server-side processing	**No.** It does not support joins, foreign keys, stored procedures, triggers, or any processing on the storage engine side.	**Yes.** It supports stored procedures, views, multiple indices, joins, and aggregation.
Transaction support	**Limited.** Supports up to 100 operations in one transaction for entities which is in the same table and the same partition.	**Yes.** Supports typical ACID transactions within the same database.
Geo-replication	**Yes.** By default, a table is replicated to other subregions.	**No.** Since SQL Server is not cloud based, it doesn't support geo-replication.
Table schema	**Relaxed.** Each entity (row) can have different properties.	**Managed.** Fixed schema for the entire table once defined but can be altered at any time.
Data types	**Simple.** Supports only 8 data types: byte, bool, DateTime, double, Guid, Int32, Int64, String	**Complex.** Supports more complex data types such as xml in addition to other data types

The table below compares Windows Azure Table Storage with Amazon DynamoDB, a NoSQL based cloud database hosted by Amazon[7].

Comparison Criteria	Windows Azure Table Storage	Amazon DynamoDB
Consumption based pricing	Yes	Yes
Only pay for the data stored	Yes	Yes
Pay for transactions	Yes	No
Indexing support	Partial	Partial
Ability to fetch list of tables	Yes	Yes
Maximum number of tables returnedper	1000	All

call to the service		
Maximum number of tables	N/A	256
Maximum size of a table	N/A	N/A
Ability to have user defined Primary Key for a table	No	Yes
Returns continuation token in case more tables are available	Yes	Yes
Ability to delete tables	Yes	Yes
Ability to get metadata about a table like number of entities/items, size of table etc.	No	Yes
Ability to create an entity/item in a table	Yes	Yes
Maximum number of attributes per entity/item	256	N/A
Maximum number of custom attributes per entity/item	253	N/A
Attribute data type	One of eight data types (Binary, Boolean, DateTime, Decimal, Int32, Int64, Guid, and String)	String, Number, String/Number Sets (Arrays)
Maximum size of an entity/item	1 MB	64 KB
Ability to update an entity/item in a table	Yes	Yes
Supports conditional updates	Yes	Yes
Ability to delete an entity/item from a table	Yes	Yes
Ability to get all attributes of an entity/item	Yes	Yes
Ability to get selected attributes of an entity/item	Yes	Yes
Ability to perform batch operations on multiple entities/items	Yes	Yes
Maximum number of entities/items returned per call	1000	N/A
Default number of entities/items returned per call	1000	N/A
Maximum size of response payload	N/A	1 MB
System can time out queries	Yes	Yes
Maximum execution time after which a query will be timed out by the system	5 Seconds	5 Seconds

8. CONCLUSION

Eventhough the concept of Windows Azure Table looks simple and more to the point compared to traditional database, querying and manipulating data from an application requires more complicated code. Evenmore, there are some restriction such as limited data type option or inavailability of user defined index which limited the type of application can be built using Azure Table. Therefore, as a solution architect or developer, we can consider using Windows Azure Table Storage when[5]:

- The application requires storing large data volumes (in terabytes) and we want to minimize the cost.
- The application stores and retrieves large data sets and does not have complex relationships between tables that requires joins, secondary indexes, or complex server-side operation such as triggers or stored procedure.
- The application requires flexible data schema to store non-uniform objects.
- From business perspective, the application requires disaster recovery capabilities across geographical locations.
- The application need to store more than 150 GB of data without implementing sharding or partitioning logic.
- The application need to achieve high level scaling without having to manually shard the dataset.
- The application will serve pages on high traffic websites.

One of the scenario where Windows Azure Table will be suitable is when we want to create a consumer application that needs to store customer profile information for each user. We assume that when the application gets to be very popular, the data it has collected will be very large but the operation we're going to do will only include storing and retrieving it in simple ways without any complicated query or operation.

9. REFERENCES

1. Cloud Database.(2013).Retrieved 2013-12-06, from http://en.wikipedia.org/wiki/Cloud_database
2. How to use the Table Storage Service.(2012).Retrieved 2013-12-06, from http://www.windowsazure.com/en-us/develop/net/how-to-guides/table-services/
3. Windows Azure Storage Client Library 2.0 Tables Deep Dive.(2012).Retrieved 2013-12-07, from http://blogs.msdn.com/b/windowsazurestorage/archive/2012/11/06/windows-azure-storage-client-library-2-0-tables-deep-dive.aspx
4. Windows Azure Tables: Introducing Upsert and Query Projection.(2011).Retrieved 2013-12-07, from http://blogs.msdn.com/b/windowsazurestorage/archive/2011/09/15/windows-azure-tables-introducing-upsert-and-query-projection.aspx
5. Windows Azure Table Storage and Windows Azure SQL Database – Compared and Contrasted.(2013).Retrieved 2013-12-08, from http://msdn.microsoft.com/en-us/library/windowsazure/jj553018.aspx

6. Windows Azure Storage Architecture Overview.(2010).Retrieved 2013-12-08, from http://blogs.msdn.com/b/windowsazurestorage/archive/2010/12/30/windows-azure-storage-architecture-overview.aspx
7. Comparing Windows Azure Table Storage and Amazon DynamoDB – a Summary.(2012).Retrieved 2013-12-08, from http://architects.dzone.com/articles/comparing-windows-azure-table
8. Constructing Filter String for the Table Designer.(2013).Retrieved 2013-12-10, from http://msdn.microsoft.com/en-us/library/windowsazure/ff683669.aspx
9. Windows Azure Storage Abstractions and their Scalability Targets.(2010). Retrieved 2013-12-06, from http://blogs.msdn.com/b/windowsazurestorage/archive/2010/05/10/windows-azure-storage-abstractions-and-their-scalability-targets.aspx
10. Rizzo, T.(2012). *Programming Microsoft's clouds Azure and office 365*. Indianapolis: Wiley Publishing, Inc.
11. Klein, S., Roggero, H.(2012). *Pro SQL Database for Windows Azure, 2nd Edition*. New York City: Appress.