

**Synchronize Distributed Cache
With Database
Through Polling**

September 09, 2009

Introduction

NCache is a distributed caching solution that helps speed up your applications while reducing expensive trips to the database. This in turn adds up scalability because a large number of clients are moved from the database to the cache, resulting in lesser load on your database and eliminating the need to upgrade your hardware.

When data is added in a cache, you actually place a copy of the original data from the database in the cache. When a particular data in your database is updated, its copy in the cache remains unaffected and hence becomes stale. However, this situation occurs only when there are multiple applications updating the same data in the database but not all of them are taking the responsibility of updating the data in the cache.

In such a situation when some applications are updating the data in the database but not updating the cache then you need a way to synchronize your cache with a database so that any modification to the data in the database also affect the data in the cache. This can be achieved either by enabling event notifications or by polling the database to look for updates. However, if your application is using one of the databases such as SQL Server 2000, an older version of Oracle or other OLEDB compliant database that do not support event notifications, the only method to synchronize with it is by the use of polling.

Further in this article, we will study how NCache synchronizes the cache with a database using polling.

Using DbDependency in Code

Here is an example code you need to implement in your application in order to synchronize your database with the cache through polling.

```
String conString = "Provider=OraOLEDB.Oracle;User Id=SYSTEM;" +
"Password=xe;Data Source=xe;OLEDB.NET=true;";

OleDbConnection con = new OleDbConnection(conString);
con.Open();

String sqlCmd = "SELECT ProductID, ProductName FROM dbo.Products WHERE ProductID <
12";

OleDbCommand cmd = new OleDbCommand(sqlCmd, con);
cmd.ExecuteReader();

OleDbDataReader myReader = cmd.ExecuteReader();

List<Products> lstProducts = new List<Products>();
while (myReader.Read())
{
    Products cProducts = new Products();
    cProducts.ProductID = myReader.GetInt32(0);
    cProducts.ProductName = myReader.GetString(1);
    cProducts.SupplierID = myReader.GetInt32(2);
    cProducts.UnitPrice = myReader.GetDecimal(4);

    CacheItem item = new CacheItem(cProducts);
```

```

DBCacheDependency dependency =
DBDependencyFactory.CreateOleDbCacheDependency(conString, "");

_cache.Insert(cProducts.ProductID + ":dbo.Products", cProducts,
dependency, Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
CacheItemPriority.Default);

lstProducts.Add(cProducts);
};

myReader.Close();
con.Close();

```

In the above code, a single row from the database is loaded and converted into an object. Later, CreateOleDbCacheDependency is used to establish a connection with the database and register the rows for which NCache should look updates for. Note that when you create OleDb dependency using CreateOleDbCacheDependency, you have to supply it with the same SQL statement that you used earlier in the code to fetch the particular row.

Once the above code has been implemented and executed, the NCache will keep on going to the database after every specified clean interval and fetch any updated rows and then replace them with the corresponding expired rows in the cache.

Configuring Database

We have already seen in the above code how NCache implements synchronization with the database. However, the procedure for enabling polling based synchronization needs some more steps, which we will see below.

1. Create a table 'ncache_db_sync' in your Oracle database and add four fields: (1) cache_key VARCHAR2, (2) cache_id VARCHAR2, (3) modified NUMBER and (4) work_in_progress NUMBER.

Here is the SQL for creating the table:

```

Create table ncache_db_sync
(
  cache_key varchar2(256) not null enable,
  cache_id varchar2(256) not null enable,
  modified number(2,1) default 0 not null enable,
  work_in_progress number(2,1) default 0 not null enable,
  primary key (cache_key, cache_id) enable
);

```

2. Create UPDATE and DELETE triggers for every table on which notification is required. The triggers will be used to set the 'modified' field of the corresponding row in the ncache_db_sync table to 1.

Here is the SQL to create the trigger:

```

Create Trigger MyTrigger
After
Update [or delete] on dbo.Products
Referencing OLD AS oldRow
For each row

```

```
Begin
Update ncache_db_sync
Set modified = 1
Where cache_key = (Cast(Select oldRow.ProductID FROM deleted OLD) As VarChar)
+ ':dbo.Products'
End Trigger;
```

Note: cache_key must be the same key that is used to add the corresponding record in the cache.

The way the triggers and the 'ncache_db_sync' table works is that whenever a value in the database table gets changed or deleted, the appropriate trigger automatically gets called. The trigger will change the value of the 'modified' field of the 'ncache_db_sync' table to 1, indicating the value of the primary key has been modified.

The NCache performs clean up of expired and unnecessary items in the cache after every specified period. This period is called as clean interval. On every clean interval, the NCache also looks for the value of the 'modified' field in the 'ncache_db_sync' table. If the value is found to be '1', the NCache then removes the appropriate expired keys from the cache and fetches all the updated rows from the database and places in the cache.

Notes:

1. This is a row based dependency. The cache key is associated with the table's primary key value. It means that an item expires only if the row with that primary key gets changed.
2. The cache_key value must be the same that was used to add the corresponding record in the cache.

Configuring Clean Interval

Clean interval is the periodic interval after which the expired items in the cache are removed. The clean interval can be changed from the Policies tab of the NCache Manager, as shown in the Figure 1 below.

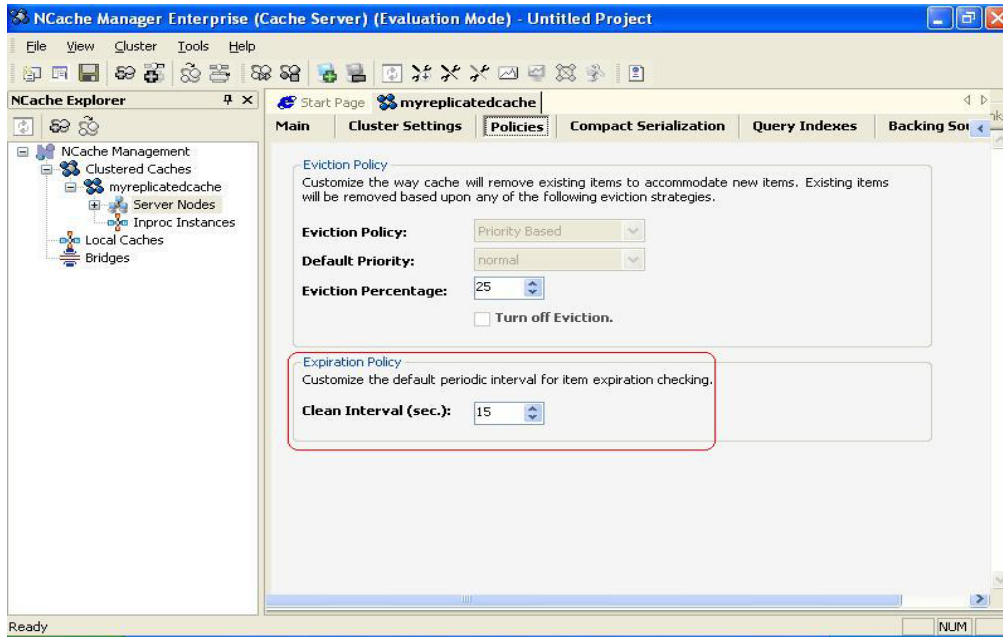


Figure 1: Changing the Clean Interval period from the Policies Tab.

Testing Synchronization

After implementing the code in your app and configuring the database to enable notifications, we may follow these steps to test the synchronization:

1. Get a particular value from the cache and make a note of it.
2. Change the value of a particular field in the database table.
3. After time interval passes, get the same data from the cache again to see the updated value. If the value is different from the previous one, this will mean that your cache is successfully synchronized with the database.

Conclusion

We have seen how NCache allows you to easily synchronize your Oracle database with your distributed cache. By doing this, whenever your database gets any changes, these would automatically affect your data in the cache, giving you confidence that your cache will never contain any invalidated data.