NHibernate를 이용한 계층형 자료구조 조회 방법

리얼웹	배성혁	2011-04-03 오후 1:49

계층형 자료구조라는 것은 DB 상에서는 Self-Reference를 가지는 구조를 말하고, Class에서는 Parent, Children 속성을 가져서, 자신과 같은 수형의 부모, 자식들을 가질 수 있는 class 를 말합니다.
우리가 자주 사용하는 "부서(Department)" 는 최상위 부서부터 하위부서로 트리구조로 이루어져 있죠? 이런 것을 말합니다. 그럼 Tree 구조로 형상화 할 수 있는 것은? 메뉴, BPA의 프로세스 트리 (Hierarchy), PMS의 WBS, BOM (Bill Of Material) 등이 있습니다. 엄청 다양한 정보들이 트리 구조 (계층형 자료 구조)를 가지고 있습니다.

근데 이런 계층형 자료구조의 경우 부모, 자식은 직접적으로 연결되어 있어 찾기 쉬운데, 특정 노드의 최상위 조상이라던가, 특정 노드의 모든 자손들 또는 특정 노드의 최하위 자손들 (Leaf Node) 를 찾아라 한다면?

기존 MS SQL 2000까지는 커서를 사용하여 작업했습니다. (예전 BPA DB를 보면 온통 커서입니다)
MS SQL 2005부터는 CTE (Common Table Expression - 이건 SQL 99의 표준이랍니다) 를 사용하여 재귀호출이 가능해졌습니다. 이를 통해, 조상이나 자손을 찾는 작업이 상당히 편해졌습니다.
이번에 BPA 팀에서 사용하던 기존 커서 방식을 CTE 재귀호출 방식으로 모두 바꾸었습니다.

Oracle은 9i 부터 재귀호출을 독자적으로 지원하고 있었고, CTE 방식도 지원합니다. 제가 보기엔 Oracle 독자적인 방식의 문법이 더 직관적이고, 간단해보입니다.

자 그런데, 우리가 Oracle과 MS SQL만 사용하는 것도 아니고, Nhibernate를 이용하니 DB가 아닌 Class 영역에서 계층형 자료구조를 처리하는 로직을 가지고 있어야 하는 것은 당연합니다.
이를 위해 RCL.Data.Domain.ITreeNodeEntity<Tid>를 위한 확장 메소드 중에는 GetAncestors(), GetDescendents() 라는 확장 메소드를 제공합니다. 물론 자료 구조 이론에 입각하여, Breadth First Search 또는 Depth First Search 같은 알고리즘을 이용하지요^^

```
198         /// <summary>
199         /// 지정한 TreeNode Entity와 모든 조상을 가져온다.
200         /// </summary>
201         /// <typeparam name="T">TreeNode 엔티티의 수형</typeparam>
202         /// <param name="current">기준이 되는 ITreeNodeEntity{T}</param>
203         /// <returns>지정한 TreeNode의 모든 조상 노드들 (자신은 제외)</returns>
204         public static IEnumerable<T> GetAncestors<T>(this T current) where T : class, ITreeNodeEntity<T>
205         {
206             if(IsDebugEnabled)
207                 log.Debug(@"지정된 노드의 모든 조상 노드를 조회한다... current=" + current);
208 
209             if(current == null)
210                 yield break;
211 
212             var parent = current;
213 
214             while(parent != null)
215             {
216                 yield return parent;
217                 parent = parent.Parent;
218             }
219         }
```

```
221         /// <summary>
222         /// 지정된 TreeNode Entity와 모든 자손들을 가져온다.
223         /// </summary>
224         /// <typeparam name="T">TreeNode 엔티티의 수형</typeparam>
225         /// <param name="current">기준이 되는 ITreeNodeEntity{T}</param>
226         /// <returns>지정한 TreeNode의 모든 자손 노드들 (자신은 제외)</returns>
227         public static IEnumerable<T> GetDescendents<T>(this T current) where T : ITreeNodeEntity<T>
228         {
229             current.ShouldNotBeDefault<T>("current");
230 
231             if(IsDebugEnabled)
232                 log.Debug(@"지정된 노드의 모든 자손 노드를 깊이 우선 탐색으로 조회한다... current=" + current);
233 
234             return current.GraphDepthFirstScan(node => node.Children).Where(child => child != null);
235         }
```

```
286         /// <summary>
287         /// 트리의 모든 끝 노드(자식이 없는 노드)를 구합니다.
288         /// </summary>
289         /// <typeparam name="T"></typeparam>
290         /// <returns></returns>
291         public static IList<T> GetLeafs<T>() where T : class, ITreeNodeEntity<T>
292         {
293             return Repository<T>.Session.Query<T>().Where(node => node.Children.Any() == false).ToList();
294         }
295 
296         /// <summary>
297         /// 트리의 모든 끝 노드(자식이 없는 노드)의 갯수를 구합니다.
298         /// </summary>
299         /// <typeparam name="T"></typeparam>
300         /// <returns></returns>
301         public static int GetLeafCount<T>() where T : class, ITreeNodeEntity<T>
302         {
303             return Repository<T>.Session.Query<T>().Where(node => node.Children.Any() == false).Count();
304         }
```

자 그럼 Nhibernate를 사용하면서, 재귀호출을 지원하지 않는 DB는 Class 영역에서 자료구조 알고리즘으로 처리한다고 하고, MS SQL Server나 Oracle의 경우에는 CTE 를 지원하므로, 그 기능을 활용하는 것이 성능 향상에 도움이 될 것입니다.
간단한 부서 트리 같은 것으로 성능 비교를 할 게 아니라, BOM 같은 것으로 비교해 봐야 CTE 성능이 더 좋다는 것을 느낄겁니다.

예전에도 각 DB별로 특정 Database 객체를 만드는 작업에 대해 설명했습니다만, 우선 MS SQL Server 의 경우에 대해 CTE를 사용하여 부서정보를 조상, 자손을 구하는 Procedure를 만들도록 하는 매핑 정보를 정의합니다.

```
<database-object>
    <create>
        create proc [GetDepartmentAndAncestors]
        @DepartmentId int
        as

            WITH DeptHierarchy (DepartmentId, ParentId)
            AS (
                SELECT DepartmentId, ParentId
                FROM dbo.Department
                WHERE DepartmentId = @DepartmentId

                UNION ALL

                SELECT D.DepartmentId, D.ParentId
                FROM dbo.Department D
                INNER JOIN DeptHierarchy DH ON (D.DepartmentId = DH.ParentId)
            )

            SELECT D.* FROM Department D inner join DeptHierarchy DH on (D.DepartmentId = DH.DepartmentId);

        RETURN
    </create>
    <drop>
        drop proc [GetDepartmentAndAncestors]
    </drop>
    <dialect-scope name="NHibernate.Dialect.MsSql2005Dialect" />
    <dialect-scope name="NHibernate.Dialect.MsSql2008Dialect" />
</database-object>

<database-object>
    <create>
        create proc [GetDepartmentAndDescendents]
        @DepartmentId int
        as

            WITH DeptHierarchy (DepartmentId, ParentId)
            AS (
                SELECT DepartmentId, ParentId
                FROM dbo.Department
                WHERE DepartmentId = @DepartmentId

                UNION ALL

                SELECT D.DepartmentId, D.ParentId
                FROM dbo.Department D
                INNER JOIN DeptHierarchy DH ON (D.ParentId = DH.DepartmentId)
            )

            SELECT D.* FROM Department D inner join DeptHierarchy DH on (D.DepartmentId = DH.DepartmentId);

        RETURN
    </create>
    <drop>
        drop proc [GetDepartmentAndDescendents]
    </drop>
    <dialect-scope name="NHibernate.Dialect.MsSql2005Dialect" />
    <dialect-scope name="NHibernate.Dialect.MsSql2008Dialect" />
</database-object>
```

뭐 내용은 많지만 결국 CTE를 활용하여, 조상 부서, 자손부서를 구하는 Procedure 입니다.
이 Procedure 들을 활용하려면 Named Query 로 정의해 놓으면 간단하게 활용할 수 있습니다.

```
<sql-query name="GetDepartmentAndAncestors.MsSql" cacheable="true" comment="자신과 조상 부서들을 모두 조회합니다.">
    <return class="Department" />
    <![CDATA[
        EXEC GetDepartmentAndAncestors :DepartmentId
    ]]>
</sql-query>

<sql-query name="GetDepartmentAndDescendents.MsSql" cacheable="true" comment="자신과 하위의 모든 자손 부서들을 조회합니다.">
    <return class="Department" />
    <![CDATA[
        EXEC GetDepartmentAndDescendents :DepartmentId
    ]]>
</sql-query>
```

즉 Procedure 를 수행하고, Result Set으로 Department 로 반환하라고 한 것입니다.

그럼 Oracle의 경우를 볼까요? Oracle은 START WITH XXXX CONNECT BY PRIOR 라는 Oracle 전용 문법을 사용했습니다.^^

```
92  <sql-query name="GetDepartmentAndAncestors.Oracle" cacheable="true" comment="자신과 조상 부서들을 모두 조회합니다.">
93      <return class="Department" />
94      <![CDATA[
95          SELECT *
96            FROM Department
97           START WITH DepartmentId = :DepartmentId
98           CONNECT BY PRIOR ParentId = DepartmentId
99      ]]>
100 </sql-query>
101
102 <sql-query name="GetDepartmentAndDescendents.Oracle" cacheable="true" comment="자신과 조상 부서들을 모두 조회합니다.">
103     <return class="Department" />
104     <![CDATA[
105         SELECT *
106           FROM Department
107          START WITH DepartmentId = :DepartmentId
108          CONNECT BY PRIOR DepartmentId = ParentId
109     ]]>
110 </sql-query>
```

Oracle이 훨씬 간단하죠? 재귀호출에 대해서는 정말 그렇더군요...

자 그럼 이놈들을 어떻게 사용하는지 봅시다. Nhibernate의 NamedQuery 를 사용하면 되는데, 코드를 보면...

```
79          [Test]
80          public void GetAncestors_By_HierarchyQuery()
81          {
82              IQuery query = null;
83
84              if(UnitOfWork.CurrentSessionFactory.IsMsSqlServer2005OrHigher())
85                  query = UnitOfWork.CurrentSession.GetNamedQuery("GetDepartmentAndAncestors" + ".MsSql");
86
87              else if(UnitOfWork.CurrentSessionFactory.IsOracle())
88                  query = UnitOfWork.CurrentSession.GetNamedQuery("GetDepartmentAndAncestors" + ".Oracle");
89
90              if(query != null)
91              {
92                  var depts = Repository<Department>.GetPage(new NHOrder<Department>(d => d.NodePosition.Level));
93
94                  foreach(var dept in depts)
95                  {
96                      query.SetParameter("DepartmentId", dept.Id);
97                      var ancestors = query.List<Department>();
98
99                      Assert.Greater(ancestors.Count, 0);
100
101                     ancestors.Contains(dept);
102
103                     if(dept.Parent != null)
104                         ancestors.Any(a => dept.Parent == a);
105                 }
106             }
107         }
```

```
109         [Test]
110         public void GetDescendents_By_HierarchyQuery()
111         {
112             IQuery query = null;
113 
114             if(UnitOfWork.CurrentSessionFactory.IsMsSql2005OrHigher())
115                 query = UnitOfWork.CurrentSession.GetNamedQuery("GetDepartmentAndDescendents" + ".MsSql");
116 
117             else if(UnitOfWork.CurrentSessionFactory.IsOracle())
118                 query = UnitOfWork.CurrentSession.GetNamedQuery("GetDepartmentAndDescendents" + ".Oracle");
119 
120             if(query != null)
121             {
122                 var depts = Repository<Department>.GetPage(new NHOrder<Department>(d => d.NodePosition.Level));
123 
124                 foreach(var dept in depts)
125                 {
126                     query.SetParameter("DepartmentId", dept.Id);
127                     var descendents = query.List<Department>();
128 
129                     Assert.Greater(descendents.Count, 0);
130 
131                     descendents.Contains(dept);
132                     descendents.Any(d => d.Parent == dept);
133                 }
134             }
135         }
```

보시다시피 GetNamedQuery로 Iquery를 얻어서 작업하기만 하면 됩니다. 물론 다른 DB인 경우에는 전용 NamedQuery가 없을 테니 위의 ITreeNodeEntity<T>의 확장 메소드를 이용하는 수 밖에 없습니다.

계층형 자료구조에 대한 질의를 제공해주는 DB라면, 위와 같이 전용 SQL 문을 만들어서 사용하는 편이 성능상 잇점이 많습니다. 거기다가 우리가 만드는 제품의 거의 SQL Server나 Oracle 이 될 확률이 99%이니 이 두가지 DB에 대해서는 전용 Query를 이용하여 대응하는 것이 좋을 것입니다.