عنوان: **استفاده از Full text search توسط Entity Framework**
نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۴/۲۸ ۲۳:۱۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Full Text Search

پیشنیاز مطلب:
پشتیبانی از Full Text Search در SQL Server

Full Text Search یا به اختصار FTS یکی از قابلیت‌های SQL Server جهت جستجوی پیشرفته در متون میباشد. این قابلیت تا کنون در EF 6.1.1 ایجاد نشده است.
در ادامه پیاده سازی از FTS در EF را مشاهده مینمایید.
جهت ایجاد قابلیت FTS از متد Contains در Linq استفاده شده است.
ابتدا متدهای الحاقی جهت اعمال دستورات FREETEXT و CONTAINS اضافه میشود.سپس دستورات تولیدی EF را قبل از اجرا بر روی بانک اطلاعاتی توسط امکان Command Interception به دستورات FTS تغییر میدهیم.
همانطور که میدانید دستور Contains در Linq توسط EF به دستور LIKE تبدیل میشود. به جهت اینکه ممکن است بخواهیم از دستور LIKE نیز استفاده کنیم یک پیشوند به مقادیری که میخواهیم به دستورات FTS تبدیل شوند اضافه مینماییم.
جهت استفاده از Fts در EF کلاس‌های زیر را ایجاد نمایید.

کلاس FullTextPrefixes :

```
/// <summary>
    ///
    /// </summary>
    public static class FullTextPrefixes
    {
        /// <summary>
        ///
        /// </summary>
        public const string ContainsPrefix = "-CONTAINS-";

        /// <summary>
        ///
        /// </summary>
        public const string FreetextPrefix = "-FREETEXT-";

        /// <summary>
        ///
        /// </summary>
        /// <param name="searchTerm"></param>
        /// <returns></returns>
        public static string Contains(string searchTerm)
        {
            return string.Format("({0}{1})", ContainsPrefix, searchTerm);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="searchTerm"></param>
        /// <returns></returns>
        public static string Freetext(string searchTerm)
        {
            return string.Format("({0}{1})", FreetextPrefix, searchTerm);
        }

    }
```

کلاس جاری جهت علامت گذاری دستورات FTS میباشد و توسط متدهای الحاقی و کلاس FtsInterceptor استفاده میگردد.

کلاس FullTextSearchExtensions :

```
public static class FullTextSearchExtensions
    {
```

```
        /// <summary>
        ///
        /// </summary>
        /// <typeparam name="TEntity"></typeparam>
        /// <param name="source"></param>
        /// <param name="expression"></param>
        /// <param name="searchTerm"></param>
        /// <returns></returns>
        public static IQueryable<TEntity> FreeTextSearch<TEntity>(this IQueryable<TEntity> source,
Expression<Func<TEntity, object>> expression, string searchTerm) where TEntity : class
        {
            return FreeTextSearchImp(source, expression, FullTextPrefixes.Freetext(searchTerm));
        }

        /// <summary>
        ///
        /// </summary>
        /// <typeparam name="TEntity"></typeparam>
        /// <param name="source"></param>
        /// <param name="expression"></param>
        /// <param name="searchTerm"></param>
        /// <returns></returns>
        public static IQueryable<TEntity> ContainsSearch<TEntity>(this IQueryable<TEntity> source,
Expression<Func<TEntity, object>> expression, string searchTerm) where TEntity : class
        {
            return FreeTextSearchImp(source, expression, FullTextPrefixes.Contains(searchTerm));
        }

        /// <summary>
        ///
        /// </summary>
        /// <typeparam name="TEntity"></typeparam>
        /// <param name="source"></param>
        /// <param name="expression"></param>
        /// <param name="searchTerm"></param>
        /// <returns></returns>
        private static IQueryable<TEntity> FreeTextSearchImp<TEntity>(this IQueryable<TEntity> source,
Expression<Func<TEntity, object>> expression, string searchTerm)
        {
            if (String.IsNullOrEmpty(searchTerm))
            {
                return source;
            }

            // The below represents the following lamda:
            // source.Where(x => x.[property].Contains(searchTerm))

            //Create expression to represent x.[property].Contains(searchTerm)
            //var searchTermExpression = Expression.Constant(searchTerm);
            var searchTermExpression = Expression.Property(Expression.Constant(new { Value = searchTerm
}), "Value");
            var checkContainsExpression = Expression.Call(expression.Body,
typeof(string).GetMethod("Contains"), searchTermExpression);

            //Join not null and contains expressions

            var methodCallExpression = Expression.Call(typeof(Queryable),
                                                       "Where",
                                                       new[] { source.ElementType },
                                                       source.Expression,
                                                       Expression.Lambda<Func<TEntity,
bool>>(checkContainsExpression, expression.Parameters));

            return source.Provider.CreateQuery<TEntity>(methodCallExpression);
        }
```

در این کلاس متدهای الحاقی جهت اعمال قابلیت Fts ایجاد شده است.

متد FreeTextSearch جهت استفاده از دستور FREETEXT استفاده میشود.در این متد پیشوند -FREETEXT- جهت علامت گذاری این دستور به ابتدای مقدار جستجو اضافه میشود. این متد دارای دو پارامتر میباشد ، اولی ستونی که میخواهیم بر روی آن جستجوی FTS انجام دهیم و دومی عبارت جستجو.

متد ContainsSearch جهت استفاده از دستور CONTAINS استفاده میشود.در این متد پیشوند -CONTAINS- جهت علامت گذاری این دستور به ابتدای مقدار جستجو اضافه میشود. این متد دارای دو پارامتر میباشد ، اولی ستونی که میخواهیم بر روی آن جستجوی FTS انجام دهیم و دومی عبارت جستجو.

متد FreeTextSearchImp جهت ایجاد دستور Contains در Linq میباشد.

کلاس FtsInterceptor :

```
public class FtsInterceptor : IDbCommandInterceptor
    {

        /// <summary>
        ///
        /// </summary>
        /// <param name="command"></param>
        /// <param name="interceptionContext"></param>
        public void NonQueryExecuting(DbCommand command, DbCommandInterceptionContext<int>
interceptionContext)
        {
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="command"></param>
        /// <param name="interceptionContext"></param>
        public void NonQueryExecuted(DbCommand command, DbCommandInterceptionContext<int>
interceptionContext)
        {
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="command"></param>
        /// <param name="interceptionContext"></param>
        public void ReaderExecuting(DbCommand command, DbCommandInterceptionContext<DbDataReader>
interceptionContext)
        {
            RewriteFullTextQuery(command);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="command"></param>
        /// <param name="interceptionContext"></param>
        public void ReaderExecuted(DbCommand command, DbCommandInterceptionContext<DbDataReader>
interceptionContext)
        {
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="command"></param>
        /// <param name="interceptionContext"></param>
        public void ScalarExecuting(DbCommand command, DbCommandInterceptionContext<object>
interceptionContext)
        {
            RewriteFullTextQuery(command);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="command"></param>
        /// <param name="interceptionContext"></param>
        public void ScalarExecuted(DbCommand command, DbCommandInterceptionContext<object>
interceptionContext)
        {
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="cmd"></param>
        public static void RewriteFullTextQuery(DbCommand cmd)
        {
            var text = cmd.CommandText;
            for (var i = 0; i < cmd.Parameters.Count; i++)
            {
                var parameter = cmd.Parameters[i];
```

```
                if (
                    !parameter.DbType.In(DbType.String, DbType.AnsiString, DbType.StringFixedLength,
                        DbType.AnsiStringFixedLength)) continue;
                if (parameter.Value == DBNull.Value)
                    continue;
                var value = (string)parameter.Value;
                if (value.IndexOf(FullTextPrefixes.ContainsPrefix, StringComparison.Ordinal) >= 0)
                {
                    parameter.Size = 4096;
                    parameter.DbType = DbType.AnsiStringFixedLength;
                    value = value.Replace(FullTextPrefixes.ContainsPrefix, ""); // remove prefix we
added n linq query
                    value = value.Substring(1, value.Length - 2); // remove %% escaping by linq
translator from string.Contains to sql LIKE
                    parameter.Value = value;
                    cmd.CommandText = Regex.Replace(text,
                        string.Format(
                            @"\[(\w*)\].\[(\w*)\]\s*LIKE\s*@{0}\s?(?:ESCAPE N?'~')",
parameter.ParameterName),
                        string.Format(@"CONTAINS([$1].[$2], @{0})", parameter.ParameterName));
                    if (text == cmd.CommandText)
                        throw new Exception("FTS was not replaced on: " + text);
                    text = cmd.CommandText;
                }
                else if (value.IndexOf(FullTextPrefixes.FreetextPrefix, StringComparison.Ordinal) >= 0)
                {
                    parameter.Size = 4096;
                    parameter.DbType = DbType.AnsiStringFixedLength;
                    value = value.Replace(FullTextPrefixes.FreetextPrefix, ""); // remove prefix we
added n linq query
                    value = value.Substring(1, value.Length - 2); // remove %% escaping by linq
translator from string.Contains to sql LIKE
                    parameter.Value = value;
                    cmd.CommandText = Regex.Replace(text,
                        string.Format(
                            @"\[(\w*)\].\[(\w*)\]\s*LIKE\s*@{0}\s?(?:ESCAPE N?'~')",
parameter.ParameterName),
                        string.Format(@"FREETEXT([$1].[$2], @{0})", parameter.ParameterName));
                    if (text == cmd.CommandText)
                        throw new Exception("FTS was not replaced on: " + text);
                    text = cmd.CommandText;
                }
            }
        }

    }
```

در این کلاس دستوراتی را که توسط متدهای الحاقی جهت امکان Fts علامت گذاری شده بودند را یافته و دستور LIKE را با دستورات CONTAINSو FREETEXT جایگزین میکنیم.

در ادامه برنامه ای جهت استفاده از این امکان به همراه دستورات تولیدی آنرا مشاهده مینمایید.

```
public class Note
    {
        /// <summary>
        ///
        /// </summary>
        public int Id { get; set; }

        /// <summary>
        ///
        /// </summary>
        public string NoteText { get; set; }
    }

  public class FtsSampleContext : DbContext
    {
        /// <summary>
        ///
        /// </summary>
        public DbSet<Note> Notes { get; set; }

        /// <summary>
        ///
        /// </summary>
        /// <param name="modelBuilder"></param>
```

```
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new NoteMap());
        }
    }

/// <summary>
    ///
    /// </summary>
    class Program
    {
        /// <summary>
        ///
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args)
        {
            DbInterception.Add(new FtsInterceptor());
            const string searchTerm = "john";
            using (var db = new FtsSampleContext())
            {
                var result1 = db.Notes.FreeTextSearch(a => a.NoteText, searchTerm).ToList();
                //SQL Server Profiler result ===>>>
                //exec sp_executesql N'SELECT
                //    [Extent1].[Id] AS [Id],
                //    [Extent1].[NoteText] AS [NoteText]
                //    FROM [dbo].[Notes] AS [Extent1]
                //    WHERE FREETEXT([Extent1].[NoteText], @p__linq__0)',N'@p__linq__0
                //char(4096)',@p__linq__0='(john)'
                var result2 = db.Notes.ContainsSearch(a => a.NoteText, searchTerm).ToList();
                //SQL Server Profiler result ===>>>
                //exec sp_executesql N'SELECT
                //    [Extent1].[Id] AS [Id],
                //    [Extent1].[NoteText] AS [NoteText]
                //    FROM [dbo].[Notes] AS [Extent1]
                //    WHERE CONTAINS([Extent1].[NoteText], @p__linq__0)',N'@p__linq__0
                //char(4096)',@p__linq__0='(john)'
            }
            Console.ReadKey();
        }
    }
```

ابتدا کلاس FtsInterceptor را به EF معرفی مینماییم. سپس از دو متد الحاقی مذکور استفاده مینماییم. خروجی هر دو متد توسط SQL Server Profiler در زیر هر متد مشاهده مینمایید.

تمامی کدها و مثال مربوطه در آدرس https://effts.codeplex.com قرار گرفته است.

منبع:

Full Text Search in Entity Framework 6

نظرات خوانندگان

نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۵/۰۹ ۱۶:۵۶

بسته نوگت پروژه جاری:

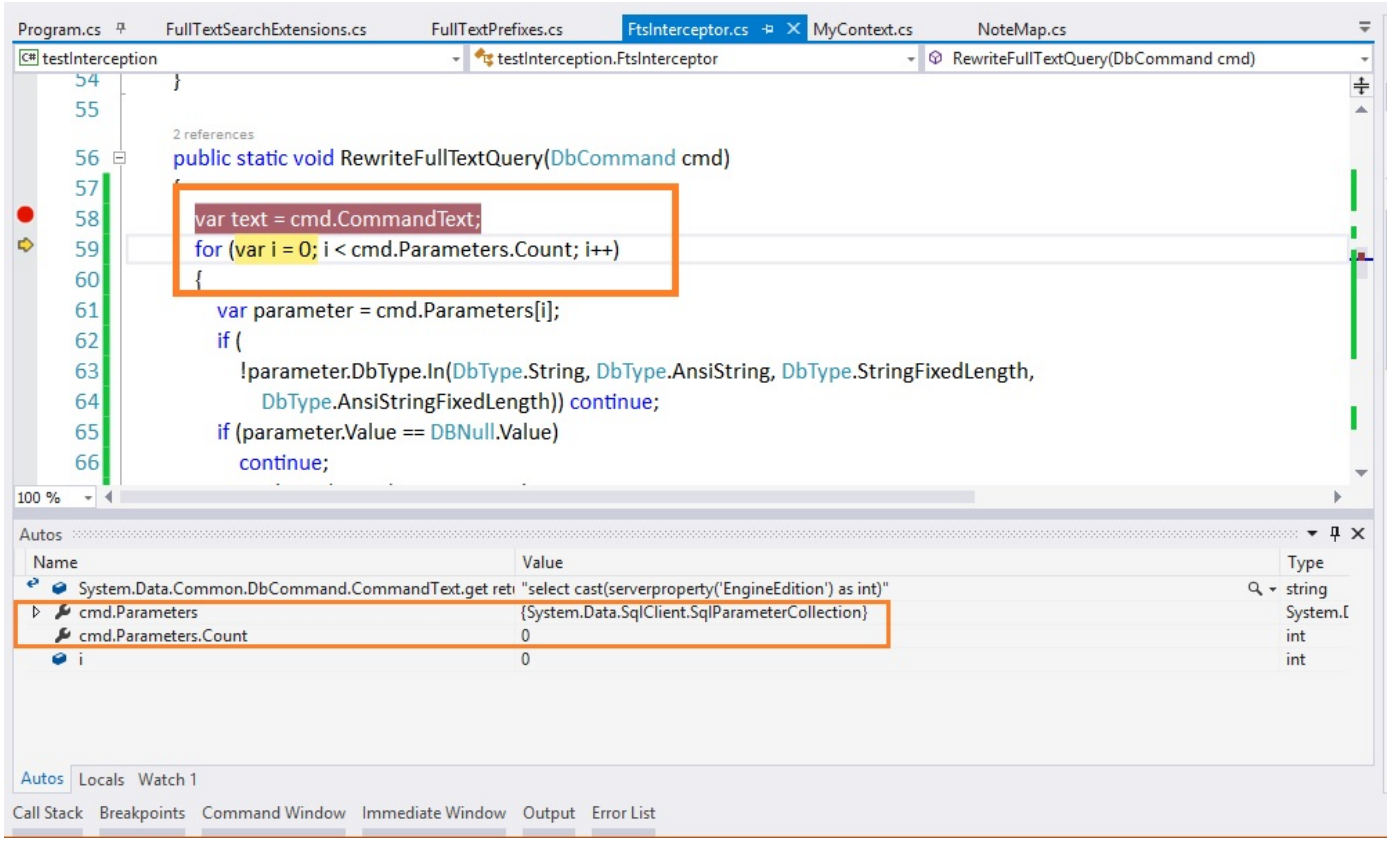
```
Install-Package EfFts
```

نویسنده: امیرحسین ابراهیمیان
تاریخ: ۱۳۹۴/۰۴/۱۴ ۱۱:۱۸

وقتی برنامه را اجرا کردم خطای

An error occurred while executing the command definition. See the inner exception for details

را می‌داد. وقتی تریس کردم دیدم command ای ارسال نمیشه. ممکنه بگید اشکال اش کجاست؟



نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۴/۱۴ ۱۱:۵۳

شاید بهتر باشه کوئری و دستوراتی را هم که نوشتید برای دیباگ ارسال کنید. یکی از مراحل رسیدگی به مشکل، امکان تولید

مجدد آن هست .

نویسنده: امیرحسین ابراهیمیان
تاریخ: ۱۳۹۴/۰۴/۱۴ ۱۷:۸

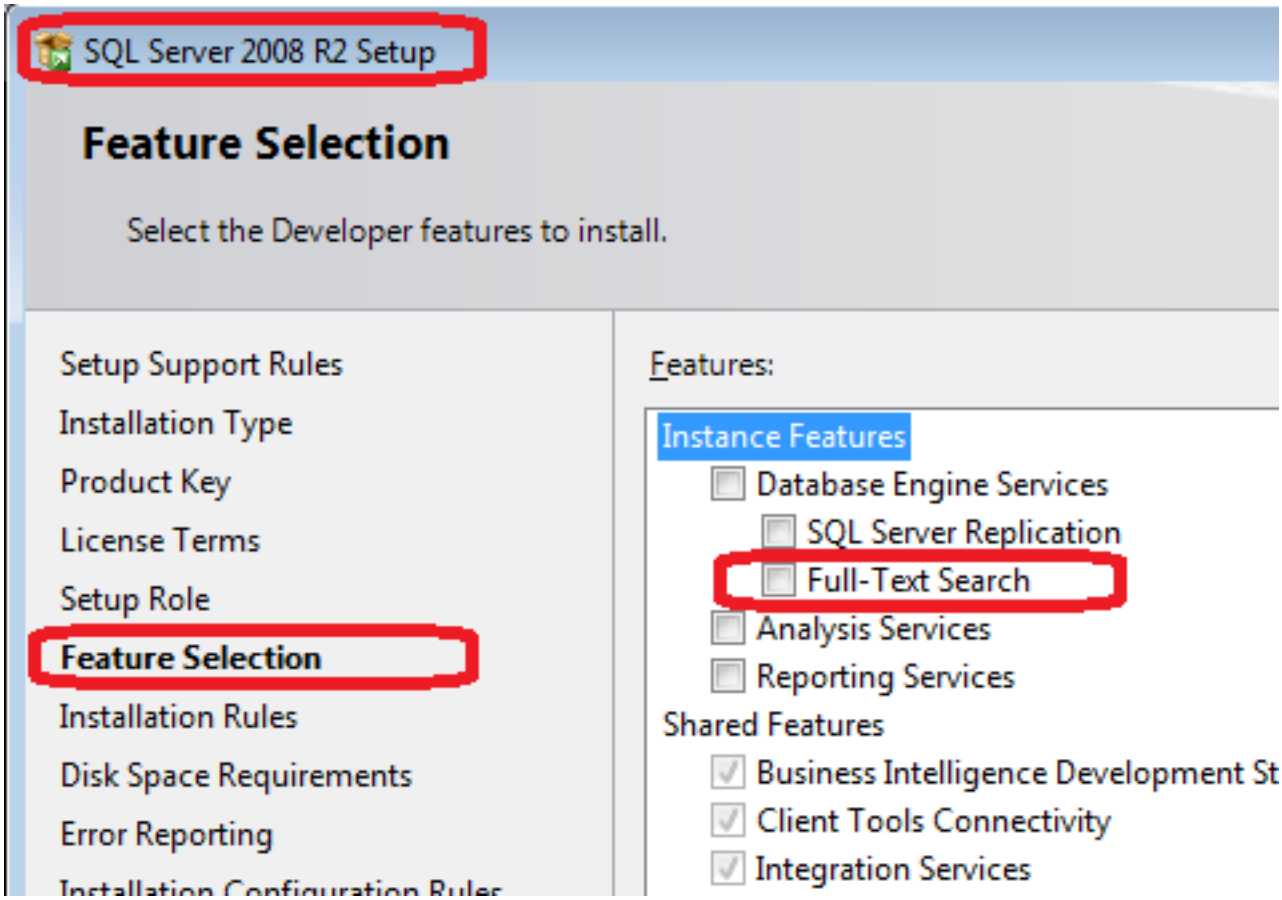
کدها را مطابق مطلب بالا زدم. چون fulltext را در چند تا سایت محدود دیدم که کد زده بودند. به خاطر همین اصلا منطق برنامه اش را متوجه نشدم و کدها را کپی کردم. بعد توی تریس کردن این مشکل اومد. چون هیچ چیزی نمی‌دونستم از دوستان خواهش کردم که اگر به مشکل من برخورد کردند لطفا جواب من را بدهند.

نویسنده: محسن موسوی
تاریخ: ۱۳۹۴/۰۴/۱۶ ۹:۲۸

لطفا کدهای نمونه را ارسال نمایید.

نویسنده: جواد حاجیان نژاد
تاریخ: ۱۳۹۴/۰۸/۰۸ ۱۴:۳۹

چند نکته بسیار مهم درباره قابلت Full Text Search که دوستان باید مد نظر داشته باشند عبارتست :
1- ابتدا باید در هنگام نصب این قابلیت را در SQL Server فعال کرده باشید



2- برای اینکه بتوان بر روی ستون‌های مورد نظر Full text Serach زد باید indexهای لازم و همچنین کاتولوگ‌های لازم را تعریف نمود.

```
ایجاد کاتالوگ//
 use AdventureWorks
```

```
create fulltext catalog FullTextCatalog as default

select *
from sys.fulltext_catalogs

//تعریف ایندکس بر روی ستون  مورد نظر
create fulltext index on Production.ProductDescription(Description)
key index PK_ProductDescription_ProductDescriptionID
```

3- توجه داشته باشید برای حجم داده‌های کم قابلیت Full text Search بسیار کند و زمان برتر از جست و جوهای پایه نظیر استفاده از Like می‌باشد و زمانی که حجم داده‌ها زیاد می‌باشد باید از قابلیت Full text Search استفاده شود

4- خروجی جست وجوی Full Text Search و جست و جوی معمولی برای داده‌های زیاد یکسان نمی‌باشد ، کافی است در یک دیتا بیس با حجم بالای داده‌ها جست و جو را با هر دو روش انجام دهید ، آن وقت خواهید دید که جست و جوی سنتی (نظیر استفاده از دستور LIKE) بسییار دقیق‌تر می‌باشد و تمامی اطلاعات خواسته شده را درست بر خواهد گرداند، امام با استفاده از Full Text Search این اطلاعات کامل نمی‌باشد! کافی است خودتان امتحان کنید

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۸/۰۸ ۱۴:۴۸

برای گرفتن خروجی مناسب از FTS نیاز هست یک سری نکات ویژه را رعایت کرد؛ اطلاعات بیشتر در دوره‌ی پشتیبانی از Full Text Search در SQL Server مطرح شده‌اند.