عنوان: **معرفی Lex.Db**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱:۰
آدرس: www.dotnettips.info
گروه‌ها: Silverlight, WinRT, Lex.Db, Embedded Database

Lex.Db یک بانک اطلاعاتی درون پروسه‌ای (مدفون شده یا embedded) بسیار سریع نوشته شده با سی‌شارپ است. این بانک اطلاعاتی کم حجم، سورس باز بوده و مجوز استفاده از آن LGPL است. به این معنا که استفاده از اسمبلی‌های آن در هر نوع پروژه‌ای آزاد است.
نکته مهم آن سازگاری با برنامه‌های دات نت 4 به بعد، همچنین برنامه‌های ویندوز 8، سیلورلایت 5، ویندوز فون 8 و همچنین اندروید (از طریق Mono) است. به علاوه چون با دات نت تهیه شده است، دیگر نیازی نیست دو نگارش 32 بیتی و 64 بیتی آن توزیع شوند و به این ترتیب مشکلات توزیع بانک‌های اطلاعاتی native مانند SQLite را ندارد (و مطابق ادعای نویسنده آلمانی آن، از SQLite سریعتر است).
API این بانک اطلاعاتی، هر دو نوع متدهای synchronous و asynchronous را شامل می‌شود؛ به همین جهت با برنامه‌های ویندوز 8 و سیلورلایت نیز سازگاری دارد.
Lex.Db از برنامه‌های چندریسمانی و همچنین استفاده از یک بانک اطلاعاتی آن توسط چندین پروسه همزمان نیز پشتیبانی می‌کند.
در ادامه مروری خواهیم داشت بر نحوه استفاده از آن در حالت طراحی رابطه‌ای؛ از این جهت که فعلا به ظاهر این بانک اطلاعاتی روابط را پشتیبانی نمی‌کند، اما در عمل پیاده سازی آن مشکل نیست.

دریافت Lex.Db

برای دریافت Lex.Db، دستور ذیل را در خط فرمان پاورشل نیوگت وارد نمائید:

```
PM> Install-Package Lex.Db
```

بسته به نوع پروژه شما (دات نت یا WinRT یا ...)، اسمبلی متناسبی به پروژه اضافه خواهد شد.

مدل‌های برنامه

```
public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }

    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string City { get; set; }
    }

    public class Order
    {
        public int Id { get; set; }
        public int? CustomerFK { get; set; }
        public int[] ProductsFK { get; set; }
    }
```

مدل‌های برنامه آزمایشی مطلب جاری را در اینجا ملاحظه می‌کنید. برای طراحی روابط یک به صفر یا یک و همچنین یک به چند، تنها کافی است کلیدهای اصلی یا آرایه‌ای از کلیدهای اصلی مرتبط را در اینجا ذخیره کنیم، که نمونه‌ای از آن‌را در کلاس Order ملاحظه می‌کنید.

آغاز بانک اطلاعاتی

```
public static class Database
    {
        public static DbInstance Instance { get; private set; }

        public static DbTable<Product> Products { get; private set; }
        public static DbTable<Order> Orders { get; private set; }
        public static DbTable<Customer> Customers { get; private set; }

        /// <summary>
        /// سازنده استاتیکی که در طول عمر برنامه فقط یکبار اجرا می‌شود
        /// </summary>
        static Database()
        {
            createDb();
            getTables();
        }

        private static void getTables()
        {
            Products = Instance.Table<Product>();
            Customers = Instance.Table<Customer>();
            Orders = Instance.Table<Order>();
        }

        private static void createDb()
        {
            Instance = new DbInstance(Path.Combine(Environment.CurrentDirectory, "LexDbTests"));

            Instance.Map<Product>()
                    .WithIndex("NameIdx", x => x.Name)
                    .Automap(i => i.Id, true);

            Instance.Map<Order>()
                    .Automap(i => i.Id, true);

            Instance.Map<Customer>()
                    .WithIndex("NameIdx", x => x.Name)
                    .WithIndex("CityIdx", x => x.City)
                    .Automap(i => i.Id, true);

            Instance.Initialize();
        }
    }
```

کلاس دیتابیس و سازنده آن، استاتیک تعریف شده‌اند؛ تا در طول عمر برنامه تنها یکبار وهله سازی شوند. new DbInstance یک وهله جدید از بانک اطلاعاتی را آغاز می‌کند. سازنده آن، مسیر پوشه‌ای که فایل‌های این بانک اطلاعاتی در آن ذخیره خواهند شد را دریافت می‌کند. Lex.Db به ازای هر کلاس مدلی که به آن معرفی شود، دو فایل data و index را ایجاد می‌کند.

سپس توسط وهله‌ای از بانک اطلاعاتی که ایجاد کردیم، کار معرفی خواص مدل‌های برنامه توسط متد Map و Automap انجام می‌شود. متد Automap خاصیت primary key کلاس را دریافت کرده و همچنین پارامتر دوم آن مشخص می‌کند که آیا این کلید اصلی به صورت خودکار ایجاد شود یا خیر. به علاوه در همینجا می‌توان روی فیلدهای مختلف، ایندکس نیز ایجاد کرد. متد WithIndex یک نام دلخواه را دریافت کرده و سپس خاصیتی را که باید بر روی آن ایندکس ایجاد شود، دریافت می‌کند.

در نهایت متد Initialize باید فراخوانی گردد. البته اگر برنامه شما WinRT است، این متد *Async* Initialize خواهد بود.

جداول نیز بر اساس مدل‌های برنامه از طریق متد Instance.Table در دسترس قرار گرفته‌اند.

افزودن اطلاعات به بانک اطلاعاتی

```
private static void addData()
        {
            var customer1 = new Customer { Name = "customer1", City = "City1" };
            var customer2 = new Customer { Name = "customer2", City = "City2" };
            Database.Instance.Save(customer1, customer2); // automatic Id assignment after Save

            var product1 = new Product { Name = "product1" };
            var product2 = new Product { Name = "product2" };
            Database.Instance.Save(product1, product2); // automatic Id assignment after Save

            var order1 = new Order { CustomerFK = customer1.Id, ProductsFK = new[] { product1.Id } };
            var order2 = new Order { CustomerFK = customer2.Id, ProductsFK = new[] { product1.Id,
product2.Id } };
            Database.Instance.Save(order1, order2); // automatic Id assignment after Save
```

```
        }
```

اکنون که کار آغاز بانک اطلاعاتی صورت گرفت، برای افزودن اطلاعات از متد Database.Instance.Save می‌توان استفاده کرد (در برنامه‌های WinRT از متد *Async* Save استفاده کنید).
در اینجا نیازی به ذکر Id نمونه‌های ساخته شده نیست؛ از این جهت که در حین عملیات Save، به صورت خودکار انتساب خواهند یافت.
همچنین نحوه مقدار دهی کلیدهای خارجی نیز با استفاده از همین کلیدهای اصلی آماده شده است.

واکشی تمام اطلاعات

```
private static void loadAll()
        {
            var orders = Database.Orders.LoadAll();
            foreach (var order in orders)
            {
                // نحوه دریافت اطلاعات مشتری بر اساس کلید خارجی ثبت شده
                var orderCustomer = Database.Customers.LoadByKey(order.CustomerFK.Value);
                Console.WriteLine("Order Id: {0}, Customer: {1} ({2}) {3}", order.Id,
orderCustomer.Name, orderCustomer.Id, orderCustomer.City);

                // نحوه بازیابی لیستی از اشیاء مرتبط از طریق آرایه‌ای از کلیدهای خارجی ثبت شده
                var orderProducts = Database.Products.LoadByKeys(order.ProductsFK);
                foreach (var product in orderProducts)
                {
                    Console.WriteLine("  Product Id: {0}, Name: {1}", product.Id, product.Name);
                }
            }
        }
```

بانک اطلاعاتی آغاز شد؛ تعدادی رکورد نیز در آن ثبت گردید. اکنون برای بازیابی اطلاعات می‌توان از متدهای در دسترس جداول کلاس Database استفاده کرد. برای مثال متد LoadAll تمام رکوردهای یک جدول را واکشی می‌کند (در برنامه‌های WinRT این متد LoadAll *Async* خواهد بود).
سپس با استفاده از متدهای LoadByKey و LoadByKeys، به سادگی می‌توان اشیاء مرتبط با هر سفارش را نیز واکشی کرد.

استفاده از ایندکس‌ها برای کوئری گرفتن

```
private static void queryingByAnIndex()
        {
            var name = "customer1";
            var customersList = Database.Customers
                                        .IndexQueryByKey("NameIdx", name)
                                        .ToList();
            foreach (var person in customersList)
            {
                Console.WriteLine(person.Name);
            }
        }
```

در ابتدای بحث، توسط متد WithIndex، تعدادی ایندکس را نیز تعریف کردیم. اکنون توسط این ایندکس‌ها و متد IndexQueryByKey، می‌توان کوئری‌هایی بسیار سریع را تهیه کرد.

```
// Using Take and Skip
            var list1 = Database.Orders.Query<int>() // primary idx
                                       .Take(1).Skip(2).ToList();

            // Querying Between Ranges
            var list2 = Database.Customers
                                .IndexQuery<string>("NameIdx")
                                .GreaterThan("a", orEqual: true).LessThan("d").ToList();
```

همچنین در اینجا متدهایی مانند Take و Skip و یا جستجو در یک بازه توسط متدهای GreaterThan و LessThan نیز پشتیبانی می‌شوند.

حذف رکوردها

```
private static void deletingRecords()
        {
            Database.Customers.DeleteByKey(key: 1);

            var customers = Database.Customers.LoadByKeys(new[] { 1, 2 });
            Database.Customers.Delete(customers);
        }
```

برای حذف رکوردها از متدهای DeleteByKey و یا Delete می‌توان استفاده کرد. متد Delete می‌تواند آرایه‌ای از اشیاء را نیز قبول کند.
و اگر خواستید کل بانک اطلاعاتی را خالی کنید، متد Database.Instance.Purge اینکار را انجام خواهد داد.

کدهای کامل این مثال را از اینجا نیز می‌توانید دریافت کنید:
Program-LexDb.cs

نظرات خوانندگان

نویسنده: ناصر طاهری
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱۳:۲

ممنون از آموزشتون. آیا متدی مثل include هم داره ؟ یا باید مثل قسمت واکشی اطلاعات در همین آموزش عمل کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۹ ۱۳:۱۵

lazy loading و eager loading ندارد. مثل همین مثال باید عمل کرد. هرجایی که نیاز است، خودتان باید اطلاعات را واکشی کنید. cascade delete هم ندارد. اگر لازم هست، خودتان دستی لیست Idها را بدهید تا حذف کند.

نویسنده: علیرضا
تاریخ: ۱۳۹۲/۰۸/۲۹ ۲۰:۵۹

و غیر از سرعت و مسئله 64/32 بیتی چه مزیتی نسبت به SQLite داره؟

نویسنده: خوزستان
تاریخ: ۱۳۹۲/۰۸/۲۹ ۲۲:۵۷

این بانک اطلاعاتی چون بصورت embeded هست سرعت لود اطلاعاتش نسبت با بانکهای دیگر بالاست ؟

نویسنده: سوین
تاریخ: ۱۳۹۲/۰۹/۰۲ ۰:۴۴

سلام
آیا از TransactionScope به صورت کامل پشتیبانی می‌کنه ؟ SqlLite به طور کامل پشتیبانی نمی‌کرد .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱:۶

- Lex.Db در حقیقت یک بانک اطلاعاتی NoSQL است. مثال رابطه‌ای رو که من در اینجا نوشتم، فقط یک شبیه سازی روابط است.
- به صورت توکار با استفاده از قفل گذاری توسط کلاس ReaderWriterLockSlim آن، خواندن‌ها و نوشتن‌های همزمان توسط چندین ترد را مدیریت می‌کند. یعنی نیازی نیست کار اضافه‌تری از این لحاظ توسط استفاده کننده انجام شود. (SQLite برای این مساله نیاز به پیاده سازی اضافی دارد و نمی‌شود با آن در حالت معمول از طریق چندین ترد همزمان کار کرد)
- از الگوریتم RedBlackTree برای ایندکس گذاری و جستجو استفاده می‌کند.

نویسنده: مهدی
تاریخ: ۱۳۹۲/۰۹/۲۶ ۲۳:۲۶

با سلام؛ آیا از lex.db در حالت چند کاربره می‌توان استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۶ ۲۳:۵۷

Lex.DB supports concurrent database access