# EF知识

1. Code First在第一次查询、更新或者删除时创建，或者使用context.Database.CreateIfNotExists()显式创建。Id或者类名+Id确认主键。
2. DbContext跟踪所有保存在DbSet中的对象的状态，这些状态可以是 Deleted Added Modified Unchanged
3. Add 会把关联的表的数据插入到数据库 Modify不会。When you set the state to modified, Entity Framework does not propagate this change to the entire object graph.因此需要手工设置其状态。
4. 当foreach迭代查询结果时，实际上是用的DataReader.
5. DbContext.SaveChanges() 运行在一个事务中，所有等待的提交要么全部执行，要么全部不执行或者回滚。SaveChanges保存在上下文中修改的数据到数据库中，但是没有关闭底部的连接。
6. Entity Framework 使用延迟执行，查询命令只有在遍历或者ToList等时才真正到数据库查询。
7. 数据库初始化策略有四种：A: Database.SetInitializer<XXXXXContext>(new DropCreateDatabaseAlways<XXXXXContext>()) 每次运行时创建新的数据库 B:  Database.SetInitializer<XXXXXContext>(new CreateDatabaseIfNotExists<XXXXXContext>()); 只有第一次运行时创建数据库，这是默认的方式 C:  Database.SetInitializer<XXXXXContext>(new DropCreateDatabaseIfModelChanges<XXXXXContext>()); 修改模型后创建新的数据库 D:   Database.SetInitializer<XXXContext>(null); 从不创建数据库
8. Sql Server和.Net数据类型之间的映射。 Seed方法用来初始化数据，但手工调用了CreateIfNotExists则Seed不会被调用。String默认对应nvarchar(max)。Entity Framework decides if a column is nullable based on the property type by convention.

|  |  |
| --- | --- |
| **SQL Server Database type** | **.NET Framework type** |
| Bigint | Int64 |
| binary, varbinary | Byte[] |
| Bit | Boolean |
| date, datetime, datetime2, smalldatetime | DateTime |
| Datetimeoffset | DateTimeOffset |
| decimal, money, smallmoney, numeric | Decimal |
| float | Double |
| int | Int32 |
| nchar, nvarchar,, char, varchar | String |
| real | Single |
| rowversion, timestamp | Byte[] |
| smallint | Int16 |
| time | TimeSpan |
| tinyint | Byte |
| uniqueidentifier | Guid |
|  |  |

1. EF默认是Unicode的
2. 三种关系： One-To-Many One-To-One Many-To-Many

One-To-Many 在One的一方加入 Virtual ICollection<Many> Property，Virtual是用来Lazy Loading的，Lazy Loading指的是真正访问数据时才去数据库访问加载数据。ICollection在构造函数是使用HashSet来初始化，避免null异常（注：EF默认反射为HashSet）。在Many中可以加也可以不加OneId外键标志和virtual One对象。可选的在OneMap中加入HasMany映射。We can always configure relationships from either the dependent or principal side of the relationship.

One To Many 人有多个电话

modelBuilder.Entity<Person>().HasMany(p=>p.Phones).WithRequired().HasForeignKey(m => m.PersonId) .WillCascadeOnDelete(False);

*Many-To-Many通过Junction连接表实现，只需在其中一个Map中添加*

HasMany(p => p.Companies).WithMany(c => c.Persons).Map(m =>

{

m.MapLeftKey("PersonId");

m.MapRightKey("CompanyId");

m.ToTable(“Person\_Compmay”);

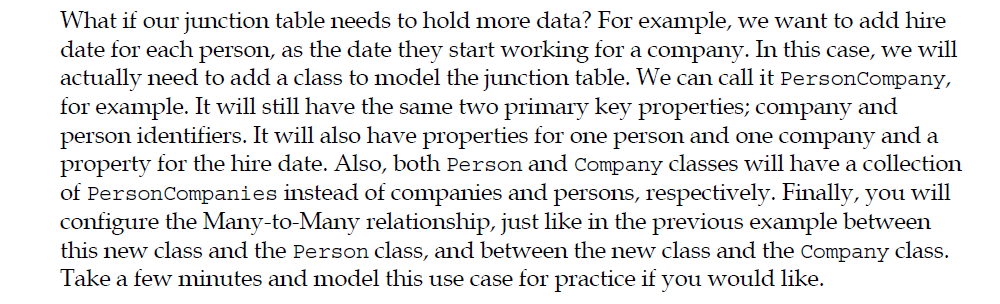
});

系统自动生成连接表。连接表可以添加额外的字段。

One-To-One One A其中一个用另一个B的主键做主键和外键 Virtual A A;virtual B B;

HasKey(s=>s.PersonId).HasRequired(s => s.Person).WithOptional(p => p.Student);

删除默认是不级联的，不级联的删除会删除父项，将子项外键置为Null.



1. String.Contains()转换为SQL的LIKE.
2. First FirstOrDefault Single SinglrOrDefault Any All StartsWith EndsWith Count Sum Min Max Average ToUpper ToLower Contains Skip Take Distinct Union Intersect Except
3. EF默认是Lazy Loading. 可以通过 context.Configuration.LazyLoadingEnable = false;来激活Eager Loading或者使用Include方法. If you are adding entities that contain child entities, the Added state is propagated to all the objects in the graph.
4. Ignore(p => p.FullName);可以忽略一些Property.
5. EF支持枚举。
6. 使用Map实现Entity Split
7. Projection使用匿名类或者定义新的类。
8. DbFunctions可以使用的SQL函数
9. JOIN指的是Inner Join

from person in context.People

join personType in context.PersonTypes

on person.PersonTypeId equals personType.PersonTypeId

select new{}；

1. Distinct Union Intersect Except
2. SqlQuery ExecuteSqlCommand

Concurrency是多个用户在同一时间修改同一行记录时，产生冲突。冲突有：丢失更新和脏读。悲观锁：**假定会发生并发冲突，**屏蔽一切可能违反数据完整性的操作。 乐观锁：**：假设不会发生并发冲突，**只在提交操作时检查是否违反数据完整性。

**悲观并发**：比如有两个用户A,B，同时登录系统修改一个文档，如果A先进入修改，则系统会把该文档锁住，B就没办法打开了，只有等A修改完，完全退出的时候B才能进入修改。

**乐观并发**：同上面的例子，A,B两个用户同时登录，如果A先进入修改紧跟着B也进入了。A修改文档的同时B也在修改。如果在A保存之后B再保存他的修改，此时系统检测到数据库中文档记录与B刚进入时不一致，B保存时会抛出异常，修改失败。

1. EF内建支持乐观锁。在SQL Server中添加一列byte[] RowVersion. Entity Framework不支持悲观并发，只支持乐观并发

Property(p=>p.RowVersion).IsFixedLength().HasMaxLength(8).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Computed).IsRowVersion();

1. 共享锁（S锁），又称读锁，若事务T对数据对象A加上S锁，则事务T可以读A但不能修改A，其他事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。这保证了其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改。排它锁（X锁）又称写锁。若事务T对数据对象A加上X锁，事务T可以读A也可以修改A，其他事务不能再对A加任何锁，直到T释放A上的锁。这保证了其他事务在T释放A上的锁之前不能再读取和修改A。