1. چرا از LINQ استفاده می کنیم؟
   1. یک کد برای هر دیتابیس. مهم نیست چه دیتابیسی داریم استفاده می کنیم، کدی که با LINQ می نویسیم روی همه دیتابیس ها جواب میده.
   2. strongly typed. یعنی مثل raw query نیست که خود کوئری مورد نظر که برای دیتابیسمون میخوایم بنویسیم رو ببریم داخل یه string بلکه با کد می نویسیمیش در نتیجه trace ، تست و دیباگ کردنش راحتتره.
   3. 2 نوع syntax داره؛ query که مشابه sql هست و مدل دیگه که بهش linq method یا fluent میگن. این دومی از extension method ها استفاده میکنه. توی اجرا این دوتا تفاوتی ندارن و یه خروجی رو میدن.
   4. خوندن کوئری در حالت LINQ ساده تر و قابل فهم تره؛ توی LINQ منظور بهتر منتقل میشه و راحتتر میشه فهمید که هر کوئری هدفش چیه.
2. طبق قسمت بالا ما دوتا مدل برای نوشتن LINQ داریم: Query و Fluent. از این به بعد با همین اسم به هر کدوم مدل ها اطلاق می کنیم.
3. برای قسمت Where داخل query، میتونیم علاوه بر نوشتن یه expression که حاصل یک یا چندتا شرط هست، یه متد که خروجیش bool هست رو استفاده کنیم (ورودی متد میتونه یه آبجکت از نوع کلاس داخل لیست ما باشه). به طور مشابه، fluent هم این قابلیت رو داره. حتی چیزی هست به اسم method group که کافیه ما فقط اسم متد مورد نظر رو داخل Where() بنویسیم؛ این حالت نیازی به lambda function نداره چون خود csharp میفهمه که باید یه انتیتی از نوع مورد نظر رو به عنوان ورودی به متد بده.
4. در هنگام OrderBy، اگه از type های از قبل تعریف شده مثل عدد ها، string، DateTime و... استفاده کنیم، به طور پیش فرض از IComparable تعریف شده خود csharp استفاده می کنه اما برای انواع دیگه باید یه کلاس که IComparable رو implement می کنه رو برای اون انتیتی تعریف کنیم.  
   همچنین در Query، با یه ویرگول راحت میشه بر اساس چندتا فیلد OrderBy رو انجام داد اما برای fluent اینطور نیست. اول باید OrderBy رو برای فیلد اول مورد نظر مرتب سازی بنویسیم و بعدش از ThenBy استفاده کنیم و همینطور تا جایی که میخوایم فیلد مرتب سازی داشته باشیم، اضافه کنیم.
5. deffered execution چیست؟
   1. این یه مفهوم به نظر ساده اما در عین حال پیچیده اس. کلیت اش اینه که یه کوئری، تا زمانی که واقعا نیاز نباشه اجرا نمیشه. بحث پیچیدگیش اما تو اینه که در شرایط مختلف این نحوه اجرای کوئری چطوریه. مثلا یه مدل اجرا اینه که وقتی داریم روی کوئری، loop میزنیم، بیاد و در حین loop زدن کوئری رو اجرا کنه؛ فرض کنیم یه دیتابیس از فیلم ها داریم و میخوایم فیلم هایی رو بیاریم که تام کروز توشون بازی کرده. کوئری رو می نویسیم و میذاریم تو یه متغیر. حالا اگه روی این متغیر که از نوع IEnumerable هست بیایم یه foreach بزنیم، میبینیم که کوئری داره روی دونه دونه فیلم هایی که داخل دیتابیس هستن اجرا میشه و وقتی به یه فیلم از تام کروز برسه، میره داخل بدنه foreach و بدنه رو اجرا می کنه. بعدش دوباره ادامه کوئری اجرا میشه و میره تا به فیلم بعدی تام کروز برسه و دوباره تا رسید، بدنه foreach رو اجرا کنه و همینطور الی آخر.
   2. همین دیتابیس رو اگه بعد از نوشتن کوئری، یه ToList() روش بزنیم و اینو داخل متغیر ذخیره کنیم، بلافاصله کوئری به طور کامل اجرا میشه و دیتا میاد داخل مموری. این ToList چه بعد از خود کوئری نوشته بشه و چه بعد از اینکه کوئری رو داخل یه متغیر ریختیم و حالا روی اون متغیر ToList بزنیم، فرقی نداره و میاد کوئری رو کامل اجرا میکنه و میریزه داخل یه متغیر.
   3. این داستان ToList برای First هم صادقه اما با یه تفاوت اونم اینه که به محض رسیدن به اولین رکورد که شرط مورد نظر رو ارضا کنه اجرای کوئری متوقف میشه.
   4. یکی از موارد کاربرد این ویژگی deferred execution اینه که میتونیم چندتا کوئری رو ترکیب کنیم. مثلا یه کوئری داخل متغیر a هست و یه کوئری دیگه داریم که داره روی کوئری a کوئری میزنه. اینجا اینطوری میشه که اگه بخوایم کوئری دوم رو روش مثلا foreach بزنیم، تا زمان رسیدن به foreach هیچ کدوم از بخش های کوئری اول یا دوم اجرا نمیشن و تازه سر foreach این کوئری ها شروع به اجرا شدن می کنن.
6. ما میتونیم شروطی که داخل where می نویسیم رو داخل متد هایی مثل First یا Last هم بنویسیم و اینطوری کوئری کوتاه تری داریم اما بسته به شرایط میتونه استفاده از این ویژگی تغییراتی ایجاد کنه؛ از بحث پرفورمنس تا نحوه اجرا و بیزنس پشت کوئری. پس اینطوری نیست که همیشه صرفا برای راحتی از این مدل استفاده کنیم.
7. متد Chunk چه کار می کند؟
   1. این متد لیست مورد نظر را به بخش های با تعداد درخواست شده (پارامتر int ورودی) تقسیم می کند. در واقع خروجی آن یک IEnumerable<Entity[]> است که هر دیتا در این IEnumerable یک آرایه به طول پارامتر ورودی می باشد. اگر تعداد کل دیتا بر این پارامتر بخش پذیر نباشد، همه بخش ها به غیر از بخش آخر دارای طول مورد نظر بوده و فقط بخش آخر به تعداد باقی مانده دیتا دارد.
   2. اگر بعد از Chunk، از Select استفاده کنیم، میتونیم علاوه بر خود Chunk ها، یه index هم کنارشون داشته باشیم. مثلا این:  
      list.Chunk(4).Select((chunk, index) => new {chunk, Number = index+1}))
8. متد های Skip و Takeچی هستن و تفاوتشون چیه؟
   1. این دوتا مثل Include و Exclude هستن و هر کدوم 3 مدل دارن: عادی شون که پسوند دیگه ای نداره، Last شون که از آخر کار مورد نظر رو لحاظ می کنه و While شون که هر دیتایی که تو شرط مورد نظر صدق کنه رو روش کار مورد نظرشون رو انجام میدن.  
      حالا خود کار اینا چیه؟ متد های Take میان Include رو انجام میدن؛ یعنی میگن چه دیتایی برداشته بشه. متدهای Skip اما کار exclude رو انجام میدن و برعکسن؛ یعنی میگن چه دیتایی برداشته نشه.
   2. این متد ها به صورت ترکیبی هم میتونن استفاده بشن. مثلا اگه بخوایم 4 دیتای اول کوئری رو نداشته باشیم و بعد از اون، 7 تا دیتای بعدی (دیتای 5 الی 11 از حاصل کوئری اصلی) رو برداریم، میتونیم اول یه Skip(4) و در ادامه اش یه Take(7) داشته باشیم.
9. متد های All و Any چی هستن و تفاوتشون چیه؟
   1. هر دو یه شرطی رو درباره تمام عناصر موجود در یه لیست بررسی می کنن منتهی Any به دنبال اینه که حداقل یه عنصر پیدا بشه که این شرط دربارش صدق کنه؛ اولین عنصری که ببینه این شرط دربارش صدق می کنه، بلافاصله از حلقه بررسی میپره بیرون و true میده و در غیر این صورت false. از اونور All میخواد ببینه که آیا تمام عناصر داخل لیست، این شرطه دربارشون صدق می کنه یا نه؛ اولین عنصری که این شرط دربارش صدق نکنه، بلافاصله از حلقه بررسی میپره بیرون و false میده و در غیر اینصورت true میده.
10. متد Concat؟
    1. یه جورایی از اسمش پیداس؛ قراره عناصر داخل یه IEnumerable ای رو به یه IEnumerable دیگه اضافه کنه. مشخصا عناصر داخل هر دو IEnumerable باید از یه type باشن.
11. متد های Union, Intersect, Except برای چه کاری هستن و تفاوتشون در چیه؟
    1. فرض کنیم 2 تا لیست یا IEnumerable داریم به اسم A و B. اینا رو میشه مثل مجموعه های داخل ریاضیات در نظر گرفت. در نتیجه میتونیم مفاهیم اشتراک و اجتماع و تفاضل و... رو براشون داشته باشیم. هر کدوم از این متد ها یکی از این مفاهیم رو پوشش میدن که در ادامه هر کدوم رو توضیح میدیم:
       1. Union: این مثل اجتماع دو مجموعه اس؛ یعنی هرچی عنصر داخل مجموعه A و B هست رو توی یه مجموعه جدا بذاره و به عنوان خروجی بهمون بده. نکته ای که داره اینه که مثل قواعد مجموعه ها در ریاضیات که میگه از یه عنصر فقط یه نمونه داخل مجموعه میتونیم داشته باشیم، این متد هم به صورت پیش فرض اگه ببینه یه عنصری داخل هر دو مجموعه وجود داره، فقط یه نمونه از اون رو داخل مجموعه خروجی میذاره.
       2. Intersect: این مثل اشتراک دو مجموعه اس؛ یعنی فقط عناصری که داخل هر دو مجموعه وجود دارن رو به عنوان خروجی توی یه مجموعه جدا میده.
       3. Except: این مثل تفاضل دو مجموعه اس. اگه روی مجموعه A متد Except رو با ورودی B بنویسیم، میاد و هر عنصری که مشترکا داخل هر دو مجموعه A و B هست رو از A حذف میکنه و مجموعه حاصل رو به عنوان خروجی میده.
    2. هر سه این متد ها یه نسخه By هم دارن (UnionBy, IntersectBy, ExceptBy). تفاوت این نسخه دوم در اینه که کار هر متد رو بر اساس یه پراپرتی یا key خاصی انجام بده. یه مثال میزنیم: فرض کنیم مجموعه A از type مثلا مشتری (customer) باشه که یه سری پراپرتی مثل اسم و شماره تلفن داره. از طرفی ما یه لیست دیگه داریم (مثلا C) که شامل اسم مشتری هایی هست که میخوایم اطلاعاتشونو در بیاریم. اینجا میایم از IntersectBy استفاده می کنیم و لیست اسم ها (که string هستن) رو به عنوان ورودی میدیم و در ادامه یه func هم میدیم که پراپرتی Name از کلاس customer رو داره بر میگردونه. اتفاقی که میفته اینه که از مجموعه A، فقط مشتری هایی که داخل لیست C هستن رو به ما میده.   
       مزیتی که این نسخه By داره اینه که لازم نیست دو مجموعه مورد نظر، از یه type باشن.
    3. تمام این متد ها از IEqualityComparer هم پشتیبانی می کنن.
12. متد SelectMany؟
    1. ما میتونیم type ای داشته باشیم که داخل خودش، پراپرتی (ها)ای داشته باشه که خودش یه لیست از یه type دیگه باشه. مثلا اگه یه لیست از فیلمها داشته باشیم، هر فیلم خودش یه لیستی از بازیگرها میتونه داشته باشه. حالا اگه ما لیستی از فیلمها داشته باشیم و بخوایم به بازیگرهای هر فیلم هم دسترسی داشته باشیم (روی اون لیست های بازیگرها کوئری بزنیم)، از متد SelectMany استفاده می کنیم.
    2. این متد ورودی اولش مشخص میکنه که کدوم پراپرتی درونی هست که لیسته و میخوایم بریم سراغ اون. ورودی بعدی، یه func هست که دوتا ورودی داره: اولی میشه شئ ای از type لیست اصلی (فیلم) و دومی میشه شئ ای از type لیست درونی (بازیگر). بعدش داخل بدنه func مینویسیم که چه خروجی ای میخوایم بهمون داده بشه.
13. انواع join ها در LINQ و کاربرد هر کدام؟
    1. اصل join که همون برگردوندن عناصر بین دو لیست بر اساس رابطه بین شون هست. حالا دو مدل join داریم:
    2. Join: این عین Inner join توی sql هست؛ یعنی اشتراک بین دو لیست رو برمیگردونه. اینم یه extension method ئه و روی یه لیست اعمال میشه که برای نوشتنش، اول باید روی لیست اول اعمالش کنیم بعدش باید 4 تا ورودی بدیم:
       1. اسم لیست دوم که قراره با این لیست join بخوره.
       2. یه پراپرتی از لیست اول که رابط با لیست دومه
       3. یه پراپرتی از لیست دوم که رابط با لیست اوله.
       4. یه func که ورودیش میشه شئ از لیست اول و شئ از لیست دوم و خودش در نهایت باید خروجی مد نظر ما رو داشته باشه.

این وسط اگه لازم باشه که join بر پایه بیش از 2 پراپرتی انجام بشه، کافیه که ورودی های دوم و سوم رو از نوع یه کلاس یا یه anonymous type بدیم که ترتیب تعریف کردن پراپرتی های داخل اونا، مشخص میکنه که کدوم پراپرتی از لیست اول باید با کدوم پراپرتی از لیست دوم، نظیر بشه.

* 1. GroupJoin: این میاد که معادل Left join توی sql باشه؛ یعنی از لیست اول همه رو بیاره و اگه از لیست دوم چیزی بود که بشه با چیزی از لیست اول همخونی داشته باشه، اینا رو به هم میچسبونه و اگه نبود، فقط همون عنصر به تنهایی بیاد.  
     اینم مثل join 4 تا ورودی می گیره که معادل همونا رو اینجا هم داریم. چیزی که فرق میکنه، بحث نحوه خالی یا پر کردن عناصر از لیست دوم در صورتی که به عنصری از لیست اول بتونن متصل بشن یا نشن، هست. برای این موضوع، کافیه که یه SelectMany استفاده کنیم که ورودی اولش، میشه همون عنصر دوم (در واقع پراپرتی از نوع عنصر دوم) اما در ادامه اون، یه .DefaultIfEmpty() هم میاد. اینطوری اون پر یا خالی بودن عنصر دوم تعیین میشه.

1. Splitted query ها چیستند؟
2. Entity framework vs nhibernate vs dapper؟