Comparison of database management possibilities of C # programming language as educational technology tools

M. A. Moldován

Verint Systems, Irinyi József street 4-20, 1117 Budapest, Hungary, Akos.Moldovan@verint.com

Abstract

This article examines different database management technologies which are available in point of programming education based on C# and using the results of exploratory research, findings related to effectiveness of programming, including database management education are presented. Standard ADO.NET, LinqToSql, Entity Framework and NHibernate technologies and their properties and opportunities are examined. The efficiency of data processing of this technologies is also investigated. The examined technologies could be important in the field of education such as usage and development, moreover, Microsoft developer environment can be learnable wherein many new possibilities of C# may be discovered.

Keywords: *educational technologies, database; C#*

C# programozási nyelv adta adatbáziskezelési lehetőségek, mint oktatástechnológiai eszközök összehasonlítása

Moldován Á. M.

Verint Systems, Irinyi József utca 4-20, 1117 Budapest, Magyarország, Akos.Moldovan@verint.com

Absztrakt

A cikkben vizsgálatra kerül, hogy a programozás oktatása tekintetében a C# programozási nyelven milyen adatbáziskezeléssel kapcsolatos lehetőségek állnak rendelkezésre, illetve az elvégzett vizsgálat alapján, feltáró kutatás eredményeként, a programozás, azon belül is az adatbáziskezelés oktatásának hatékonyságával összefüggő következtetések kerülnek megfogalmazásra. Áttekintésre kerülnek a Standard ADO.NET, LinqtoSQL-t, Entity Framework és a NHibernate technológiák, jellemzőik és lehetőségeik. Az egyes adatbázisok adatfeldolgozási hatékonysága is tesztelésre kerül, ezzel összehasonlítva az egyes technológiákat. A vizsgált technológiák az oktatás szempontjából is kiemelt jelentőséggel bírhatnak a felhasználás, fejlesztés területén, ráadásul megismerhetővé válhat a Microsoft fejlesztőkörnyezete is, amelyben a tanulók számára sok új lehetőség is megtanulható C# tekintetében is.

Kulcsszavak: oktatás technológia; adatbázisok; C#;

1. Bevezető

A C# programozási nyelv adta adatbáziskezelési lehetőségek oktatása során több, egymástól eltérő technika oktatására is lehetőség nyílik, amely hozzájárulhat a hatékonyabb fejlesztéshez, a gyorsabb megértéshez, a könnyebb átláthatósághoz, így oktatás szempontjából is könnyebb értehetőséget, bemutatást és tesztelhetőséget biztosíthat. Különböző kutatások

más-más oktatástechnológiai eszközzel és módszertani lehetőségekkel vizsgálják a hatékonyabb tudásátadást. (Katona, 2016), (Ujbanyi et al, 2016), (Katona et al, 2015)

Manapság, ha egy fejlesztő olyan feladatot kap, amelyben szükséges használni valamilyen adatbázis kezelési technológiát, akkor nem feltétlen egyértelmű a választás, mivel nem minden esetben lehet tudni az adott technológiákról, hogy éppen mennyi erőforrást használnak, vagy mennyire működnek megbízhatóan, de az adott technológia sebessége is lényeges faktor lehet a technológia kiválasztásánál. A cikkben négy egymástól teljes mértékben eltérő technológiák kerülnek összehasonlításra, majd a vizsgálat alapján javaslat kerül megfogalmazásra, hogy melyik technológiát milyen területen érdemes alkalmazni, valamint kiemelve, hogy az adott technológiáknak milyen gyengeségei vannak, valamint, hogy milyen erősségeik vannak. Fontos továbbá, hogy a vizsgálandó programok ugyanabból az adatbázisból dolgozzanak, így a későbbi teszt eredményeit összehasonlítva pontosabb képet kaphattam a különféle technológiák sebességéről, valamint erőforrás igényéről.

A már említett adatbázis kezelési technológiák közül az ADO.NET Entity Framework-öt, a Linq to SQL-t, a Standard ADO.NET-et, valamint az NHibernate-t választottam, adatbázisnak pedig egy SQL szervert használtam, így minden technológia egy adatbázisból tud dolgozni.

A mérési eredmények kinyeréséhez eleinte szerettem volna valamilyen külső programot használni, de ezt a későbbiekben elvetettem, és inkább beépítettem ezt a funkciót a már kész programokba.

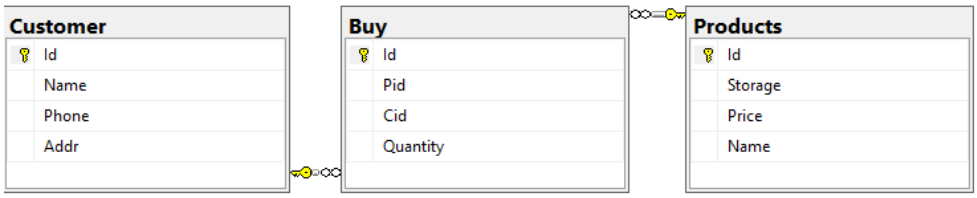
Fejlesztőkörnyezetként a Microsoft Visual Studio 2017 került alkalmazásra, amivel a Microsoft által fejlesztett SQL server-t is könnyedén össze lehet hangolni. Az SQL szerver konfigurálásához és menedzseléséhez a Microsoft SQL Server 2016-ot választottam. Troelsen, A., & Japikse, P. (2017)

2. Tervezés

2.1. Adatbázis

Első lépésként meg kellett tervezni a vizsgálat céljából kidolgozandó rendszer felépítését. Mivel adatbázis nélkül nem tudtam elkezdeni a programok fejlesztését, így először is ki kellett gondolnom, hogy milyen adatstruktúrával szeretném tesztelni a későbbiekben a technológiákat. Arra jutottam, hogy a legjobb az lenne a teszt szempontjából, ha valamilyen módon megpróbálnám szimulálni a valóságot, és mondjuk modellezném egy bolt működését.

Az adatbázis felépítéséhez három táblát használtam (1. ábra), mivel ezek összekapcsolásával minden lehetőséget lefedhettem, és ennek segítségével minden vásárló (Customer) szükséges adatát tudtam tárolni, valamint a vásárolni kívánt termékek (Product) minden az alkalmazás szempontjából szükséges információja is tárolva van, és tökéletesen elkülönül. A vásárlások adatait egy külön táblában tároltam, mivel így ez könnyebben kezelhető a fejlesztés során.



1. ábra: Az Adatbázis felépítése

2.2. C# programok

A program előzetes megtervezésénél mindenképp fontosnak tartottam, hogy a kezelő felület egyszerű legyen, valamint az is fontos volt, hogy ellenőrizni lehessen az adatbázist, így kellett valamilyen módszer, hogy meg tudjam jeleníteni az adatbázis összes elemét. Fontos volt továbbá az is, hogy véletlenül se tudjanak összeakadni a programok, ezért készült négy egymástól teljesen független alkalmazás, így nem kellett arra odafigyelni, hogy a fejlesztés, valamint a tesztelés alatt ne akadjanak össze a használt módszerek, valamint, hogy ne befolyásolják a mért eredményeket.

2.2.1. ADO.NET program

A tervezés tekintetében ezt volt a legegyszerűbb, mivel ehhez a módszerhez nem volt szükséges semmilyen plugin vagy kiegészítő csomag telepítése, így előre meghatározható volt, hogy pontosan, milyen módon telepíthető az alkalmazás, és tudható, hogy azt hogy kell majd kivitelezni. Patrick, T. (2010)

2.2.2. LINQ to SQL program

Ezt a technológiát egy laikus külső szemlélő könnyen összekeverheti az alap LINQ (Language Integrated Query)-vel. A különbség a kettő között, hogy a LINQ to SQL egy olyan eszköz, aminek segítségével objektumokként kezelhetők az adatbázisból lekért adatok, ezzel megkönnyítve az adatok lekérdezését, vagy módosítását. Calvert, C., & Kulkarni, D. (2009)

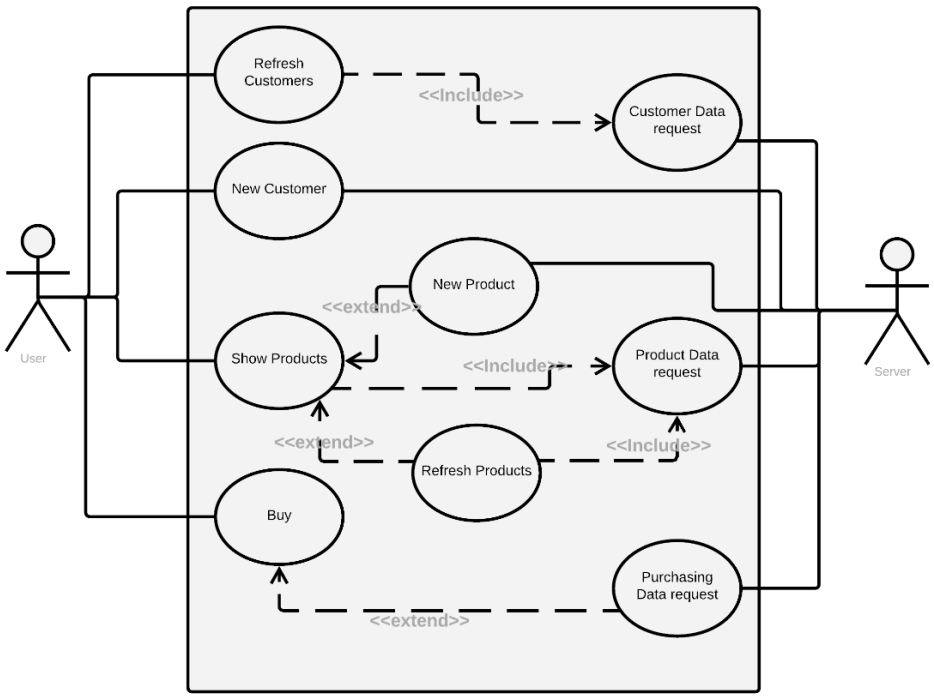
2.2.3. Entity Framework program

Mivel korábban már sokat dolgoztam Entity Framework-el így tudtam, hogy nagyjából mikkel kell számolnom a kész programot, illetve, ebben az esetben is automatikusan generált egy plusz .edmx Model file-t a Visual Studio, aminek segítségével egyszerűbbé vált az adatbázis elérése, valamint a kezelése. Lerman, J., & Miller, R. (2011)

2.2.4. NHibernate program

Ez a technológia nem biztosít előre generált osztályokat, így ezeket fejlesztésnél kell kézzel felépíteni, esetemben ez azt eredményezte, hogy minden használni kívánt táblához készíteni kellett egy osztályt, ami pontosan lemodellezte a tábla felépítését, valamint egy osztályt, ami biztosította az NHibernate számára az elérést ezekhez az osztályokhoz. Ezen felül szükség volt egy osztályra, amely biztosítja a kapcsolatot az adatbázis és az alkalmazás között. Kuaté, P. H., Harris, T., Bauer, C., & King, G. (2009)

A programok fejlesztésének megkezdése előtt tervezésre kerültek a programok funkciói (2. ábra). A funkciókat mindenképp úgy kellett megtervezni, hogy majd azok később mérhetők legyenek.



2. ábra: A programok Use-Case diagramja

3. Implementálás, integrálás

Ahhoz, hogy lehetőség legyen arra, hogy a programok csatlakozzanak, valamint, hogy interakcióba lépjenek az adatbázissal minden technológiánál más módszert kellett alkalmazni. Minél egyszerűbbnek bizonyult a fejlesztés annál bonyolultabb lett a végleges program.

A programok felépítésénél minden esetben a Windows Forms platform került felhasználásra a felhasználói felület tekintetében, így szétbonthatóvá váltak a felületi kezelések, valamint a kód is jobban olvashatóvá vált. Elkészült egy külön form a Vásárlók kezelésére, valamint ugyanígy, egy külön form a termékek kezelésére. Minden új termék, és új vásárló felvételéhez megvalósításra került egy-egy külön felület, így ezek is elkülönülnek a program többi részétől, valamint a vásárlás szimulációjához is külön felület került elkészítésre. Az alkalmazás alapját képző form-ok nem végeznek semmilyen adatbázis kezelő műveletet, ezeket a műveleteket minden program esetében az SQLHandler osztályban definiált metódusok végzik, amelyek technológiánként eltérnek egymástól.

3.1. ADO.NET program

Mivel az ADO.NET minden eszközt biztosít egy adatbázis eléréséhez, így nem volt szükséges semmi plusz kiegészítőt telepíteni, vagy hozzáadni a projekthez.

Az adatbázis elérése ebben az esetben az SQLConnection osztály használatával történik, melynek használatához szükség van a Connection String karakterláncra, amely tartalmazza az SQL szerver pontos helyét, valamint, hogy a szerverről melyik adatbázist akarjuk elérni, illetve alapvető adatokat az adatbázisról. Patrick, T. (2010)

Ezután egy egyszerű SQL query segítségével adatok kérhetők le, vagy vihetők fel a szerverre. Az alkalmazás minden esetben úgy került megtervezésre, hogy az adatbázis elérése, valamint az adatok kezelése egy külön osztályba kerüljön, ezt minden esetben SQLHandler nevet vette fel.

3.2. LINQ to SQL program

Ahhoz, hogy használható legyen az ORM eszköz szükséges volt telepíteni a Visual Studio egy külön kiegészítő csomagját, a LINQ to SQL tools-t, miután ez megtörtént már hozzá adható váltak a projekthez szükséges osztályok, amit a Visual Studio generált, miután pontosan megadásra került, hogy az adatbázisból milyen táblákat használjon. Calvert, C., & Kulkarni, D. (2009)

Az adatbázis eléréséhez szükséges osztályt LINQSQLAdapterDataContext nevett vette fel, ebben az osztályban történt az adatbázis lemodellezése, valamint a hozzátartozó mapping objektumok generálása. Ezen kívül ebben a programban is szükséges volt egy osztályban összegyűjteni az adatbázist kezelő metódusokat, ez ebben az esetben is SQLHandler volt.

3.3. Entity Framework program

Az alkalmazáshoz a Database first megközelítés került kiválasztásra, mivel a korábban elkészített programokhoz már el kellett készíteni az adatbázist, így ez a megközelítés volt a legkézenfekvőbb. Lerman, J., & Miller, R. (2011)

A keretrendszer által generált MyShopDbEntities osztály segítségével elérhetővé vált az adatbázis, viszont az adatbázissal történő interakció itt is az SQLHandler külön osztályba került összegyűjtésre, így egyszerűsítve a kódot.

Abban az esetben, ha új elemet szükséges felvenni az adatbázisba minden esetben using utasítás blokk került behívásra, mivel ez csökkentette a memória terheltségét, és így hasonló működés volt elérhető, mint a többi alkalmazásnál, így pontosabb képet kapva arról, hogy az adott technológia mögötte működése mennyire veszi igénybe a rendszert. Ha csak adatok kerültek lekérdezésre, nem voltak szükségesek az utasítás blokkok használata.

3.4. NHibernate program

Meglepően sok aspektusában eltér ez a technológia a cikk elkészítéséhez használt többi lehetőségtől, mint például, hogy ezt az eszközt NuGet-en keresztül lehet telepíteni, valamint, hogy a használatához létre kellett hozni az osztályokat, amelyekkel a program interakcióba tud lépni, valamint azokat az osztályokat, amelyeket az NHibernate mapping objektumai használnak. Ezen kívül szükséges volt még egy SessionFactory osztály létrehozása, amely biztosította a kapcsolatot az adatbázissal, Hasonlóképp a többi megközelítéshez itt is szükséges a connection string, hogy csatlakozni tudjon az adatbázishoz, viszont a FluentNHibernate segítségével konfigurálni kellett a session-t, ahol meg kellett adni az SQL szerver verzióját, valamint hozzá kellett adni a program assembly-jéhez a Fluent mapping objektumokat. Kuaté, P. H., Harris, T., Bauer, C., & King, G. (2009)

4. Tesztelés

A programok tesztelése több szempontból is megközelítésre került, így a különböző technológiák sebességét, valamint erőforrás igényességét is meg lehetett vizsgálni. Ahhoz,

hogy pontos kép születhessen mind sebességben mind erőforrás igényességben el kellett érni, hogy mérhető különbség legyen a programok között, ez két módon valósult meg, a vizsgált adatok számának növelésével, valamint a program rendelkezésére álló hardver gyengítésével. Mivel kézenfekvőbb volt a vizsgált adatok számának növelése, így a kezdetben is generált adatokkal feltöltött adatbázisba további adatok kerültek.

Fontos a tesztelés kapcsán specifikálni, hogy milyen hardware alapon történtek a mérések.

- A méréshez használt processzor: Intel Core I5 7600 3.5 Ghz
- A méréshez használt alaplap: MSI B250M
- A méréshez használt memória: 16 Gb DDR4 2400Mhz
- A méréshez használt HDD:1tb WD Blue 1TB 7200rpm

Ahhoz, hogy a mérést meg tudja ismételni a felhasználó fontos tudni, hogy a programokat újra kell konfigurálni, mivel az SQL szerver a mérés pillanatában lokálisan volt létrehozva.

A tesztek végrehajtásához először is szükség van a mögöttes adatbázisra, amely ebben az esetben egy SQL szerver biztosít. Ezután Visual Studio-n belül fel kell építeni a megfelelő adatstruktúrát, majd a mellékelt sql query segítségével fel kell tölteni azt. A generált connection string-et ezután csak ki kell másolni, majd minden program esetében az alkalmazás konfigurációs fájljában meg kell adni.

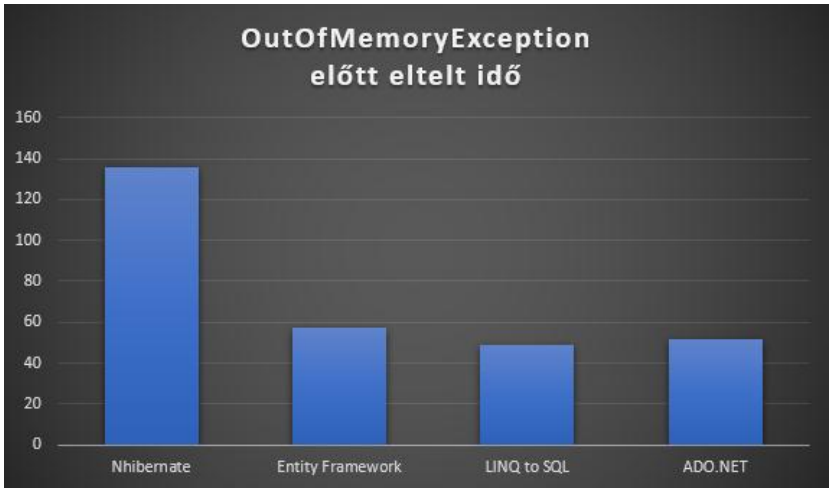
Miután összehangolásra kerültek a programok az adatbázissal, és minden program eléri azt, el lehet kezdeni a tesztet. A fejlesztés időszaka alatt először 12000 elemmel került feltöltésre a tábla, a Termékek tábla pedig 20000 elemmel. Ez a teszteléshez kevésnek bizonyult, így további 30, és 50 millió elem került generálásra a két táblába. A vásárlás táblába nem kerültek elemek, mivel abból a táblából nem lettek lekérdezve adatok, ott csak az insert-ek kerültek vizsgálatra amire nincs kihatással az, hogy fel van e töltve a táblázat.

A technológiák közötti különbséget több szempontból is vizsgálva lett, így először is talán a legfontosabb, hogy mennyire is megbízható az adott módszer, ezután összehasonlításra került a technológiák sebessége, végül az adott technológiák erőforrás igényessége került górcső alá.

4.1. Megbízhatóság

Mivel nem mindegy, hogy melyik lehetőséget választjuk az itt tesztelt 4 lehetőség közül, így fontos tudni, hogy melyik technológia mennyire megbízható. Mivel egyik technológiával sem volt elindítható az alkalmazás, a vizsgált adat mennyiség 1, illetve 2 millió sorban került meghatározásra a Customer, és a Products táblában. Ebben az esetben mind a négy

technológia sikeresen be tudta tölteni a kezdőképernyőt. A további tesztek során azonban az Entity Framework, és a LINQ to SQL nem bizonyult megbízhatónak, adat lekérdezéskor az esetek 60%-ban OutOfMemoryexception hibával megállt a program futása. Ezzel szemben az ADO.NET, valamint az NHibernate hiba nélkül, bár az elvárttól sokkal lassabban működött. Továbbá amikor 30, illetve 50 millió adattal kerültek tesztelésre az alkalmazások minden esetben összesen 3,5 Gb memóriát foglalt le magának minden alkalmazás, viszont ezt különböző idő alatt érték el, ez arra enged következtetni, hogy amelyik a leglassabban érte el ezt az értéket annak a leghatékonyabb a memória kezelése.



3. ábra: Exception előtt eltelt idő

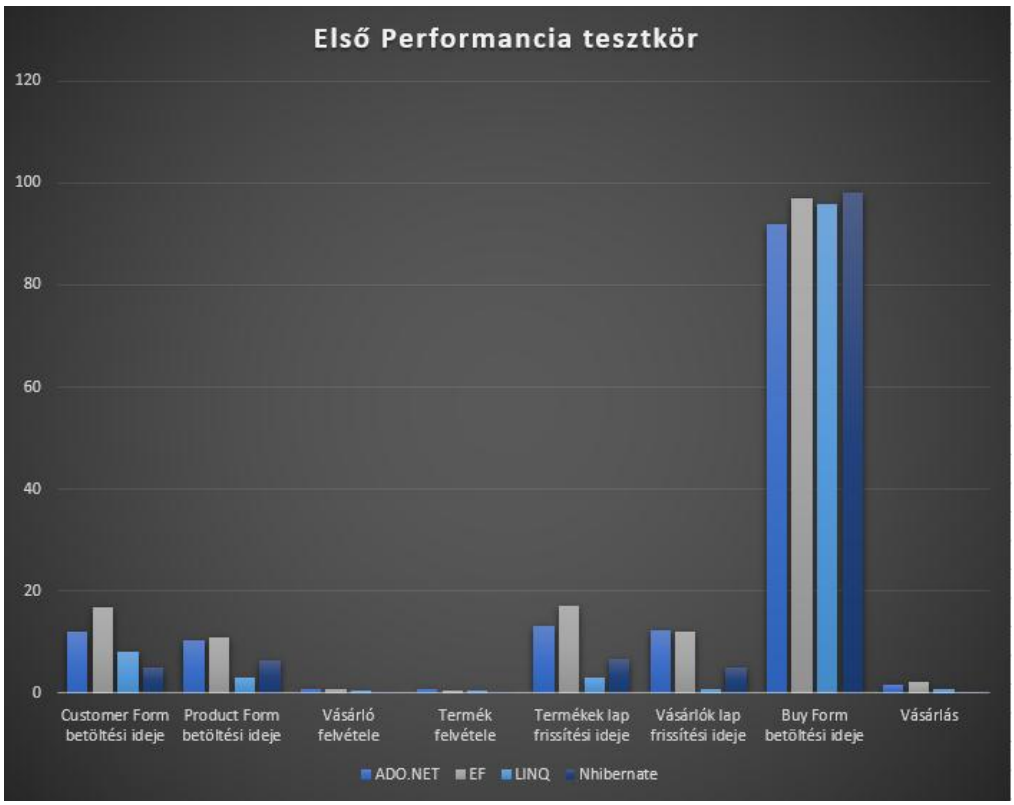
Ez alapján elmondható, hogy az Entity Framework, valamint a LINQ to SQL a nagy adatmennyiséget nem képes megbízhatóan kezelni, ezzel szemben viszont az NHibernate, valamint az ADO.NET megbízható.

4.2. Performance teszt

Hogy egy fejlesztő ki tudja választani a technológiát, amellyel majd fel akarja építeni az alkalmazást fontos szempont a sebesség, így a cikkben vizsgált 4 lehetőség között felállításra került egy sorrend, amely ad egy pontosabb képet arról, hogy adott felhasználási környezetben mely módszert érdemes alkalmazni.

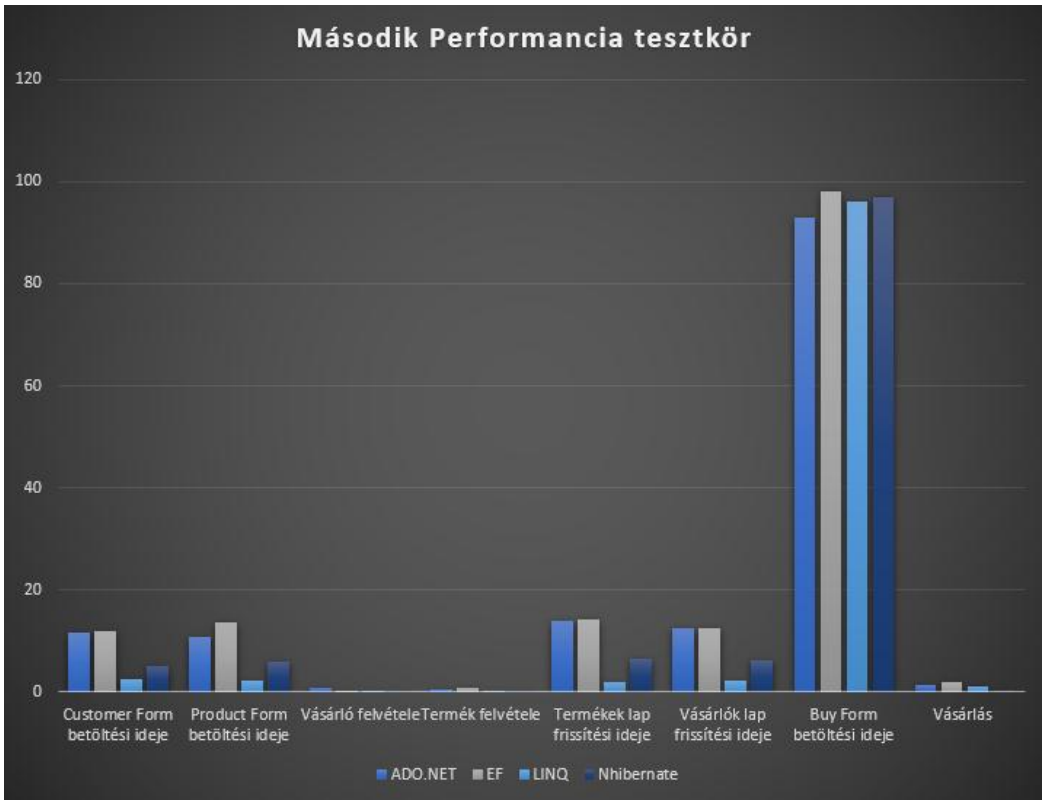
A programok közötti sebesség béli különbségeket a Stopwatch osztály segítségével került megmérésre, dokumentálásra. Minden esetben amikor meghívódik egy újabb Form előtte elindul a Stopwatch, majd a load esemény után megáll, és átadja a mért értéket egy cimkének, így minden esetben elkerülhető az emberi hiba. Hogy elkezdhető legyen a tesztelés először szükséges volt egy lista összeállítása a tesztelni kívánt funkciókról, ez tartalmazta a program összes funkcióját, ami lefedte az UPDATE, a SELECT, valamint az INSERT parancsokat.

A szoftver tesztelés alapelveit követve, két teszt kör került futtatásra, amelyek elég markáns eredményeket produkáltak.



4. ábra: Az első performancia tesztkör eredmények

Az első tesztkör adott egy átfogó képet, amely alapján már egy előzetes sorrend volt felállítható a technológiák sebessége szerint. Már ennél a tesztkörnél látható volt, hogy bármelyik technológiát is vizsgáljuk az új elem felvétele egyik esetben sem terhelte meg a rendszert, minden esetben zökkenő mentesen azonnal feltöltésre kerültek az új elemek.



4. ábra: A második performancia tesztkör eredmények

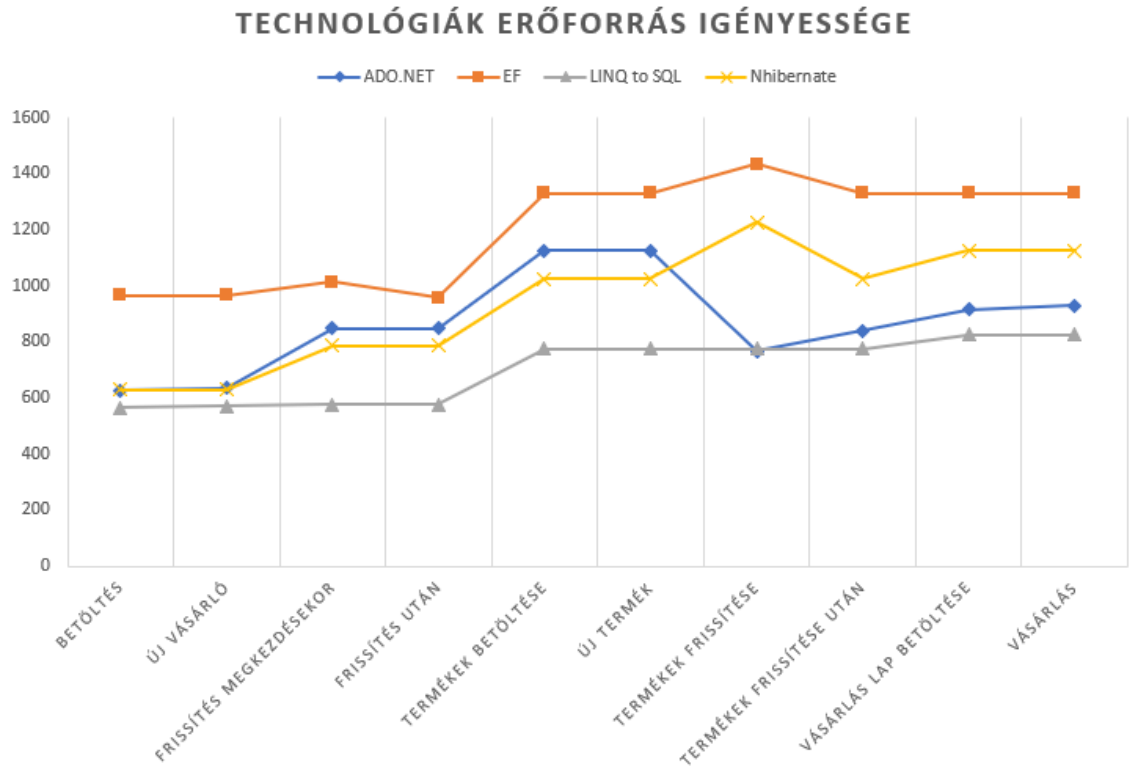
A második tesztkör hasonló eredményeket produkált, mint az első, így a teszt eredményeket összesítve elmondható, hogy sebesség szerint sorba állítva a következő a sorrend a leggyorsabbtól a leglassabbig.

- LINQ to SQL
- NHibernate
- ADO.NET
- Entity Framework

Az eredményeket elemezve észrevehető, hogy a Buy Form betöltési ideje vette igénybe a legtöbb időt, ez betudható annak hogy amikor meghívódik a Buy Form a form load eseményében egyből feltöltődik mindkét lista a két táblázat elemeivel, ami gyakorlatilag két SELECT utasítás, mivel látható a mért eredményeken hogy ez minden esetben sok időbe telik, így érthető hogy minden esetben több mint másfél perc alatt nyílt meg a vásárlásokat menedzselő ablak.

Azon felül, hogy milyen sebességgel képes az adott technológia elvégezni az adatbázissal való interakciót, fontos még az, hogy mennyi erőforrást igényel. Mivel napjainkban már ritkán lehet találkozni olyan eszközzel, amely nem lenne képes tárolni a memóriájában akár 1, vagy több millió adatot, mégis fontos lehet egy erőforrás igényességbéli sorrend felállítása.

A sorrend felállításához a Visual Studio beépített memória, valamint processzor kihasználtság mérő eszköze került felhasználásra, amely futás közben mutatja, hogy éppen mennyi erőforrást használ az alkalmazás. Minden alkalmazás esetében még egyszer futtatásra került teljes tesztkör, így pontos kép alakult ki a hardware kihasználtságáról. Az első vizsgálati szempont mindenképp a memória kezelés volt, az alábbi ábrán látszik megabyte-ban, hogy melyik módszer mennyi memóriát használ futás közben.



5. ábra: Erőforrás igényesség diagram

Ahogy az ábrán is jól látható a technológiák memória kezelés szempontjából is markánsan eltérnek, így egy erőforrásbéli különbségek alapján összeállított sorrendet is fel lehet állítani közöttük.

- LINQ to SQL
- ADO.NET
- NHibernate
- Entity Framework

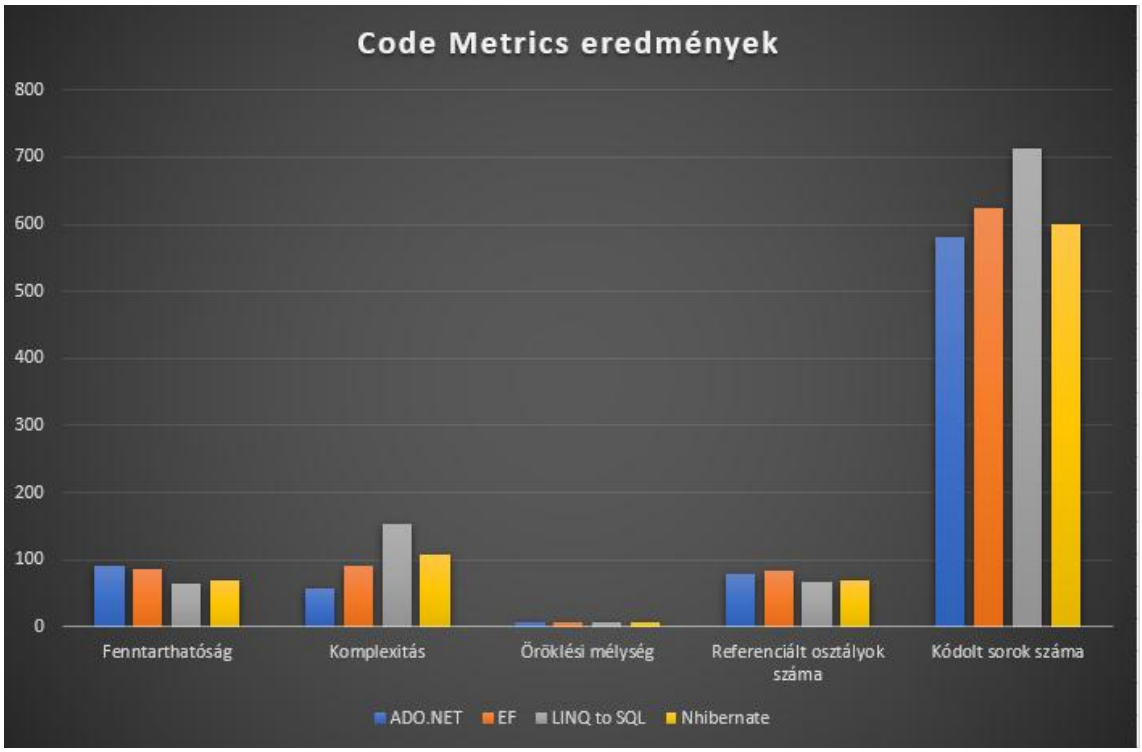
Fontos szempont volt a memória kezelésen túl, hogy az adott technológia mennyire terheli a processzort, a legpontosabb képet a Visual Studio beépített analízis eszköze adta.

Összességében mindegyik technológia effektív, és egyik sem használja feleslegesen a CPU-t, így nem terhelve azt, viszont egy sorrend ebben az esetben is felállítható, ami majd szükséges lesz a végső konklúzió levonásához.

- LINQ to SQL
- NHibernate
- ADO.NET
- Entity Framework

A Visual Studio számos lehetőséget kínál a már kész alkalmazás analizálására, melyek közül a vizsgálat szempontjából hasznos volt a Code Metrics funkció, amely több szempontból is értékeli az alkalmazást, úgy, mint Fenntarthatóság, Kód komplexitása, öröklési mélység, referenciált osztályok száma, kódolt sorok száma.

Ezt az eszközt alkalmazva minden elkészült alkalmazást értékeltettem a Visual Studio-val, és az alábbi eredményekre jutottam.



6. ábra: Code Metrics eredmények

Ahogy az látható az eredményeken minél kevesebb kódot generál valamilyen eszköz annál könnyebben fenntartható az alkalmazás, viszont az eddigi tesztek alapján elmondható, hogy a fejlesztés ideje ezzel exponenciálisan nő, viszont így vissza lehet szorítani a kód komplexitását, valamint a kódolt sorok számát.

Érdemes még megfigyelni, hogy az itt látható eredmények alapján a LINQ to SQL teljesített a legrosszabbul mind fenntarthatóság szempontjából, mind komplexitás szempontjából.

5. Javaslatok a technológiákkal kapcsolatban

A különféle tesztek futtatása alatt viszont formálódott minden technológiához egy pontosabb javaslat, hogy az adott technológiával készült program mégis milyen felhasználási területen képes a legjobban teljesíteni.

5.1. ADO.NET

Mivel a tesztelés időszaka alatt egyértelműen kiderült ez a technológia biztosítja a legkevésbé bonyolult kódot, valamint egyszerűen fenntartható, ebből következik, hogy valószínűleg ezzel a technológiával lehetséges a leggyorsabban elkészíteni egy adatbázis alapú programot. Ezt a technológiát abban az esetben kellene alkalmazni, ha rövid időn belül kell megbízható programot készíteni egy adatbázis egyszerű kezelésére, mivel egyszerűen fenntartható, így az alap adatbázis kezelést könnyűszerrel lehetséges bővíteni, illetve mivel nem komplex a kód, így a tovább fejlesztést megkönnyíti az is, hogy könnyen olvasható. Nagyobb projektekhez nem javasolt, mivel ahogy az látható több kódsorból áll egy-egy SQL parancsot eljuttatni az adatbázishoz. Ez bonyolultabb esetekben megnehezítheti a fejlesztők dolgát. A fejlesztés szempontjából azt sem szabad figyelmen kívül hagyni, hogy meglehetősen erőforrás igényes, így egy nagyobb projekt esetében, ha egyszerre több táblát is betöltve kell tartania az alkalmazásnak, vagy esetleg egyszerre több adatbázissal kell kapcsolatot létesítenie, nagyon sok memóriát használhat el, illetve a processzort is folyamatosan terheli.

5.2. Entity Framework

A tesztelés egyértelműen kimutatta, hogy az Entity Framework a vizsgált lehetőségek közül a legerőforrás igényesebb, illetve a leglassabb. A Code Metrics eredmények egyértelműen mutatják, hogy viszonylag sok kódsor szükséges ahhoz, hogy egy adatbázist alapszinten tudjon kezelni a keretrendszer, viszont tekintve a lekérdezéseket, illetve az adatok bevitelét végző eljárásokat látható, hogy azok egyszerűbbek, illetve rövidebbek, így egyértelmű, hogy a háttérben történő adatbázis modellezés tesz hozzá a kód komplexitásához, illetve valószínűleg az is lassítja le a performancia tesztelésnél látható mértékben az alkalmazást. Mivel nagyon erőforrás igényes, illetve meglehetősen lassú, viszont a fejlesztő által készített kód könnyen átlátható egyértelműen olyan felhasználási területre javasolt, ahol nem

szükséges azonnal megkapni a kívánt értéket, illetve ahol az alkalmazás felhasználói tudják biztosítani a megfelelő erőforrást a program számára.

5.3. LinqToSql

A tesztelési eredmények alátámasztják, hogy minden esetben ez a módszer biztosítja a leggyorsabb adat elérést, illetve adatküldést az adatbázis irányába, attól függetlenül, hogy az ORM szintaktikáját tekintve meglehetősen hasonlít az Entity Framework szintaktikájához. Érdekes, hogy sebességét tekintve, és erőforrás igényét tekintve is gyorsabb és jobb, mint a többi technológia. Akkor javasolt az alkalmazása, ha a fejlesztést egy nagyobb csapat végzi, illetve, ha a fejlesztésre sok idő áll rendelkezésre. Egyszerű felépíteni az eszközzel a háttérbeli adatbázis modellt, viszont a későbbi bővítés nehézkes, és bonyolult lehet, ezt jól mutatja a Code Metrics által mutatott komplexitást mutató szám is, illetve jól látható, hogy a lekérdezéseket, illetve az adatátadást végző kód részletek meglehetősen rövidek, így egyértelmű, hogy a háttérben generált adatbázis modellek foglalnak nagyon sok helyet, ezzel bonyolítva a kódot, és növelve a későbbi lehetséges hibák javításának idejét.

5.4. NHibernate

A performancia teszt egyértelműen kimutatta, hogy az NHibernate egy meglehetősen fejlett eszköz, viszont sok esetben bonyolult lehet az ezzel az eszközzel készült program fenntartása. Erőforrás igényességét tekintve közel képes arra a sebességre, mint a LINQtoSQL, valamint erőforrás igényességében sem sokkal megterhelőbb a hardver számára. Fenntarthatóságát nézve nehezebb lehet hosszútávon üzemeltetni mivel nehezebben átlátható a kód, valamint a fejlesztés is elhúzódhat abban az esetben, ha bonyolult az adatszerkezet az adatbázisban, viszont az, hogy az adatbázist kézzel kell a fejlesztőnek lemodelleznie ad egy olyan mélységű átlátást a kódra, amit se a LINQtoSQL, se az Entity Framework nem volt képes. Ez a későbbi hibakeresést, vagy az új funkciók implementálását is megkönnyítheti, de ha változik az adatstruktúra is könnyedén meg lehet változtatni az adatbázist modelljét. Mivel a fejlesztés lassabban megy, viszont a hibák keresése valószínűleg gyorsabb egy saját kézzel írt kódban, illetve a sebességét, valamint az erőforrás igényességét is tekintve olyan felhasználási környezet javasolt, ahol a fejlesztők hosszútávon fognak foglalkozni az alkalmazással, illetve fontos sok adatból gyorsan visszakapni a kért értéket. Mivel könnyebben olvasható a mögöttes modell, illetve a mögöttes kód így mindenképp javasolt a kód titkosítása, illetve a szerver irányába közölt csomagok valamilyen módszerrel való kódolása.

6. Összefoglalás

A cikkben bemutatásra került, hogy a programozás oktatása tekintetében a C# programozási nyelven milyen adatbáziskezeléssel kapcsolatos lehetőségek állnak rendelkezésre, illetve az elvégzett vizsgálat alapján, feltáró kutatás eredményeként, a programozás, azon belül is az adatbáziskezelés oktatásának hatékonyságával összefüggő következtetések kerültek megfogalmazásra. Áttekintésre kerültek a Standard ADO.NET, LinqtoSQL-t, Entity Framework és a NHibernate technológiák, jellemzőik és lehetőségeik.

Az egyes technológiák teszteléséhez létrehozásra került egy tesztkörnyezet, majd mind a négy esetben azok az adatbázis szerverrel való kommunikációja is implementálásra került, így biztosítva az egységes tesztek elvégzését. A tesztkörnyezet stabilitásának vizsgálata után a tesztek sikeresen lefutottak, és dokumentálásra kerültek, majd végül minden technológia esetében javaslat született velük kapcsolatban, így teljes képet mutatva az adott technológiák képességeiről, illetve hiányosságairól. A négy vizsgált technológia az oktatás szempontjából is kiemelt jelentőséggel bírhat a felhasználás, fejlesztés, illetve mögöttes tartalom szempontjából is. Megismerhetővé válhat a Microsoft SQL Server környezete is, ráadásul sok új lehetőséget is felfedezhetnek C# tekintetében is.

A vizsgált technológiák az oktatás szempontjából is kiemelt jelentőséggel bírhatnak a felhasználás, fejlesztés területén, ráadásul megismerhetővé válhat a Microsoft fejlesztőkörnyezete is, amelyben a tanulók számára sok új lehetőség is megtanulható C# tekintetében is.

Irodalomjegyzék

Patrick, T. (2010). *Microsoft ADO. NET 4 Step by Step*. Pearson Education.

Lerman, J., & Miller, R. (2011). *Programming Entity Framework: Code First: Creating and Configuring Data Models from Your Classes*. " O'Reilly Media, Inc.".

Calvert, C., & Kulkarni, D. (2009). *Essential Linq*. Addison-Wesley Professional.

Troelsen, A., & Japikse, P. (2017). *Pro C# 7: With. net and. net Core*. Apress.

Kuaté, P. H., Harris, T., Bauer, C., & King, G. (2009). *NHibernate in action* (Vol. 2). Manning.

Katona, J., Kovari, A. (2016). A Brain–Computer Interface Project Applied in Computer Engineering. *IEEE Transactions on Education*, *59*(4), 319-326.

Ujbanyi, T. et al (2016). Eye-tracking analysis of computer networks exam question besides different skilled groups. In *2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)* (pp. 000277-000282). IEEE.

Katona, J. et al (2015). Investigation of the Correspondence between Problems Solving Based on Cognitive Psychology Tests and Programming Course Results. *iJET*, *10*(3), 62-65.