SQL:

-Adatmanipuláló nyelv.
-Adatbázissal való kommunikációra használják.
-Célja az adatok tárolása, lekérdezése, módosítása relációs adatbázisokban.
-Egyszerű, angol kifejezéseket használ.

XML:

-Célja az adatok szerializálása, azaz hordozása, tárolása és rekonstruálása.
-Általában két vagy több különböző rendszer közötti adatátvitelre használják.
-Hierarchikus (fa) felépítésű.
-Egyszerűen olvasható gép és ember számára is.

Relációs és XML adatmodell:

1. XML dokumentum elemei közötti kapcsolat a dokumentum felépítési hierarchiáján keresztül értelmezhező.
   Relációs modellben az adatok összefüggése a táblák közötti kapcsolatokkal határozható meg.
2. XML dokumentum önleíró, formátumó. A tartalmat befoglaló címkék elmondják, hogy milyen típusú adatról van szó. Egy dokumentumban több típusú adat is lehet.
   Relációs modellben az adatok típusát az oszlopdefiníció határozza meg. Egy oszlopban lévő összes adatnak egyező típusúnak kell lennie.
3. XML adatmodell jobban kezeli a ’dizájn változtatásokat’, rugalmasabb felépítésű, mint egy relációs adatmodell.
4. XML adatmodell szerializálása / értelmezése költségesebb feladat, mint egy relációs adatmodellé.
5. XML dokumentum adatainak módosítása csak a teljes dokumentum cseréjével lehetséges. Ha egy nagyobb dokumentum kisebb részeit nagy gyakorisággal módosítjuk, akkor ésszerűbb lehet relációs adatmodellt alkalmazni, ha viszont kisebb dokumentumot frissítünk az XML adatmodell szintén hatékony lehet.
6. Komplexebb felépítésű adat XML dokumentumban könnyebben leírható (egymásba ágyazódás) , mint egy relációs adatbázisban (sok tábla).

XML dokumentum tárolására alkalmas adatbázisok:

1. IBM DB2
2. Microsoft SQL Server
3. Oracle Database
4. PostgreSQL

Natív XML adatbázisok:

1. BaseX
2. eXist
3. MarkLogic Server
4. Qizx
5. Sedna

XML dokumentumokat, adatokat tárolnak. Nem SQL lekérdező parancsokkal működnek, hanem XPath utasításokkal.

Előny:

-Nagy mennyiségű XML típusú adat tárolására, lekérdezésére sokkal hatékonyabbak, mint relációs adatbázisok.

-Beállításukhoz nincs szükség táblák és egyéb bonyolultabb struktúrák létrehozásához.

Hátrány:

-Kevésbé elterjedtek, a relációs adatbázisok sok éve használtak, jól bevált adatbázisok.

-XPath nehezen tanulható, bonyolult szintaktikájú, míg az SQL olvasható, könnyebben értelmezhető, átlátható.

Az adatokat valahogy tárolni kell, vagy fájlként, vagy adatbázisba. Tegyük fel, hogy két eszközön használjuk ugyanazt az alkalmazást, három tárolási módot fogok ebben az esetben jellemezni.

Mivel személyes használatról beszélünk, feltételezhetjük, hogy az adat mennyisége nem hatalmas méretű, ezért a tárterületből való kifogyás lehetőségét minden esetben elvethetjük.

Lokális fájl

Az alkalmazás adatai a számítógépen tárolódnak. Szöveges formátumú adat tárolása esetén a leggyakrabban használt fájl formátumok a CSV, JSON, XML és YAML.

Előny:

-Gyors és egyszerű hozzáférés az adatokhoz.

-Könnyedén implementálható megoldás, évtizedek óta alkalmazott módszer az adatok lokális fájlként való tárolása.

-Könnyedén átmásolható fizikai adathordozóra, így potenciálisan növelve a védelmet.

Hátrány:

-Két eszköz esetén adat duplikáció lehetséges.

-Tartalma csak teljes fájl olvasása esetén érhető el.

Lokális adatbázis

Hasonló előnyei vannak, mint a helyi fájlnak.

-Egyszerű hozzáférés az adatokhoz, teljes adat kontrollálás.

-Nincs szükség internetkapcsolatra.

-Gyorsabb, mint egy távoli adatbázis.

-Adatelérés esetén csak a szükséges információval dolgozik.

Hátrány:

-Nehéz az adatmegosztás külső résztvevővel (másik számítógép lokális adatbázisa).

-Ha az eszköz olyan állapotba kerül, hogy a fájlokhoz nem lehet hozzáférni, akkor elveszlik minden adat.

Távoli (felhőalapú / cloud) adatbázis

-Bárhonnan elérhető, internet elérés szükséges. Negatívuma ugyanebből származik, ha nincs internet, nem lehet elérni.

-Adatok nem helyileg a számítógépen tárolódnak, emiatt lassabb lehet az elérésük. Figyelembe kell venni, hogy más szerverek / alkalmazások is használhatják ugyanazt a hálózatot, ami szintén befolyásolhatja az adatok elérését.

-Teknikai hibák ellenére (számítógép elromlik) az információ biztonságba marad a távoli adatbázison.

---Saját ötlet---

Szóba került egy probléma többgépes alkalmazás készítése esetében, mégpedig az adatok szinkronizálása különböző eszközökön.

Az adatszinkronizálás egy olyan folyamat, ami alatt a különböző eszközökön eltárolt információkat megpróbáljuk össze egyeztetni, összhangba hozni.

Szükséges megoldani ezt a problémát alkalmazások készítésekor, hiszen nem szeretnénk ugyanazt az információt bevinni a rendszerünkbe, amit már egy másik eszközön egyszer megtettünk. Legtöbb esetben szeretnénk azonos adatokat elérni minden készüléken.

Programozási szempontból ez adatbázisok és/vagy fájlok szinkronizálását jelenti.

A felsorolt tárolási módszerekből következtetve három esetet fogok jellemezni.

Tegyük fel, hogy két eszköz esetén az alábbi módon tárolódnak az adatok:

Lokális fájl – lokális fájl, lokális adatbázis – lokális adatbázis, távoli adatbázis.

Lokális fájl – lokális fájl

Első probléma két eszközön a fájlok megosztása.

Megoszthatók fizikai adathordozók használatával.

Ha ugyanazon a hálózaton találhatóak az eszközök, a megosztása egyszerűen megoldható.

Ha nem, szükség van valamilyen szolgáltatásra, ami mindkét eszköz esetében használható. Internetelérés esetében felhőalapú tárhely jó megoldás lehet (pl.: Dropbox, Google Drive).

Ahhoz, hogy a fájlok tartalmát szinkronizáljuk, vagy egy külső alkalmazást veszünk segítségül, vagy saját alkalmazást készítünk külön erre a feladatra.

További problémák lehetnek:

-Ha az egyik fájlba benne van bizonyos adat, a másikba pedig nem, akkor az törlésre került, vagy újonnan létrehozott információ? Valamilyen megkülönböztetés szükséges.

-Két fájlban ugyanazok az adatok szerepelnek? Itt is valamilyen megkülönböztetés szükséges. Jó megoldás lehet az utolsó módosítási dátum hozzáadása minden fontosabb információhoz.

-Hogyha a két fájl különböző, nem azonos módon lett tárolva (pl.: egyik XML, másik JSON), valamilyen közös értelmezési megoldás szükséges. Ez ugyanazon alkalmazás használata esetében nem jellemző, inkább cégeknél, ha egy régebbi alkalmazás és egy újabb adatait kell egyeztetni.

Lokális adatbázis – lokális adatbázis

Néhány adatbázis-kezelő rendszer esetén létezik beépített funkció azonos típusú adatbázisok szinkronizálására.

Manuálisan ez egy nehezebb feladat és végrehajtását manuálisan tudom elképzelni, hasonló módon a két fájl szinkronizálásához.

Ha a fájlhoz hasonló adatmegosztási problémák sikeresen megoldhatók, a szinkronizációs program / script működését így tudnám elképzelni:

-Az adatbázisokba minden adat rendelkezik egyedi azonosítóval, illetve új adat bevitelekor vagy módosításakor egy tulajdonság mindig frissül. Például bevitelkor és szerkesztéskor lehet az adott időpontot megadni. Törléskor lehet egy boolean mezőbe eltárolni az adott elem állapotát, egy törölt elem például true lenne.

-Az adatokat XML / JSON vagy más adathordozó állománnyá konvertáljuk.

-Egyik fél megosztja az adatait ezen állomány segítségével a másik féllel.

-A befogadó fél szintén átkonvertálja az adatait az adott formátumra, majd egy összehasonlítás történik. Egyező azonosító esetén az adott elem tulajdonságai az utolsó módosításnak megfelelően változnak. Ha valami törlésre került, akkor itt is true legyen a törlési állapota.

-Ha ezek a lépések megtörténtek, a befogadó adatbázison elvégezhetőek a változtatások a dokumentum alapján.

Távoli adatbázis

Elegendő egy távoli adatbázis létrehozása kettő vagy több eszköz esetén.

Megkönnyíti az adatszinkronizálást, hiszen mindegyik kliens(?) ugyanazzal az adatbázissal dolgozik. Ebből következik, hogy az adatok mindig szinkronizálva lesznek. Ha egy cella értékét megváltoztatja X felhasználó, akkor Y felhasználó is látni fogja azt a módosítást.

Ebben az esetben inkább az adattitkosítási problémák a jelentősek.

---Saját ötlet vége---

Adatbázis titkosítás

(11.link)

Egy olyan folyamatot nevezhetünk adatbázis titkosításnak, ami egy algoritmus segítségével az adatbázisban tárolt adatokat titkosított szöveggé (cipher text) alakítja, ami értelmezhetetlen a megfelelő kulcs ismerete nélkül.

Célja, hogy megvédje az adatainkat a potenciális fenyegetések ellen. Hogyha egy hacker valahogy sikeresen feltöri az adatbázist, akkor számára értéktelen, értelmezhetetlen szöveggel fog találkozni.

Többféle titkosítási technika is létezik:

Transparent Data Encryption (TDE) (Átlátható adat titkosítás):

(github linkek)

-Teljes adatbázist, úgynevez 'nyugvó adatokat' (data-at-rest) titkosít. Olyan adatot nevezünk így, amit éppen semmilyen alkalmazás nem használ, szerkeszt és nem visznek keresztül a hálózaton. Például egy szöveges dokumentum, amit még nem nyitottak meg. Ilyen adatokat általában fizikaim adathordozókon tárolnak. Lopással kapcsolatos kérdéseket vet fel.

-A módszer biztosítja, hogyha még el is lopják a fizikai adathordozót, akkor sem férnek hozzá a tolvajok a rajta lévő adatokhoz.

-Mivel az összes adatot titkosítja, ezért nem szükséges speciális módon rendezni az adatokat.

-Adatbázis biztonsági mentéseit is titkosítja.

-Adatok titkosítási a tároláskor történik, visszafejtésük pedig rendszer memóriába való hívásakor történik.

-Szimmetrikus kulcsot használ a kódoláshoz.

Column Level Encryption (Oszlop szintű titkosítás)

(github linkek)

-Relációs adatmodell esetén egy adatbázis táblákból, oszlopokból, sorokból és cellákból vagy mezőkből áll. Ahogy a nevéből is következik, ez a módszer egy ilyen adatbázis egy sorát titkoítja.

-Független oszlopokat lehet titkosítani. Az oszlop összes adatát titkosítja kivétel nélkül.

-Előnye, hogy könnyedén megkülönböztethető az érzékeny és a nem érzékeny adat, illetve külön kulcs használható minden oszlop titkosításához, így növelve a biztonságot. Sokkal rugalmasabb, mint a teljes adatbázist titkosító TDE.

-Hátrány az előnyeiből fakad. Több oszlop több kulccsal való titkosítása az adatbázis teljesítményének csökkenéséhez vezethet. Lassabban lehet keresni és indexelni is.

Field Level Encryption (Cella szintű titkosítás)

(github linkek)

-Relációs adatmodell esetén használható.

-Kiválasztható, hogy pontosan melyik mezőt szeretnénk titkosítani.

-Előnyei és hátrányai megegyeznek az előbb ismertetett Column Level Encryption-nel.

-Nincs minden esetben szükség a mezők dekódolására, lehetőség van egyenlőség vizsgálatra.

További titkosítási formák, amik már nem pontosan az adatbázis titkosításhoz sorolhatók.

Filesystem Encryption

(8.link)

-Fájlrendszer titkosítás, szokás még fájl / mappa titkosításnak, FBE-nek (file-based encryption) is nevezni.

-Célja adott fájl tartalmának titkosítása

-Előnye, hogy minden fájlt külön kulccsal lehet titkosítani, így növelve a biztonságot.

-A kriptográfiai kulcs addig van a memóriában, amíg az adott fájl meg van nyitva.

-Aki fizikailag hozzáfér a tároló számítógéphez, az láthatja, hogy milyen nevű fájlok találhatóak a rendszeren, holott a tartalmukat nem tudja megnézni, amíg nem ismeri a kulcsot.

-Olyan adatokat is képes titkosítani, amelyek nem részei az adatbázis rendszernek.

-Csökkenti a teljesítményt és operációs rendszer hozzáférést is kíván a használatához.

-Teljesítményproblémák miatt nem igazán alkalmazzák, de ennek elenére kis felhasználószámú rendszerek esetében ajánlják.

Full Disk Encryption

(7.link)

-Teljes merevlemez titkosítás

-Az egész merevlemez tartalma titkosításra kerül.

-Általában ugyanazt a kulcsot használja az egész meghajtó titkosításához, ezért futásidőben az összes adat visszafejthető. Néhány módszer több kulcsot is használ különböző ’fejezetek’(?) titkosításához.

-Nagy hátránya, ha a támadó futásidőben fér hozzá a számítógéphez, minden fájl elérhető számára.

Application Level Encryption

-Kódolás és dekódolás adatátvitel és tárolás előtt történik.

-Maga az alkalmazás végzi a titkosítási folyamatot.

-Az adat csak a megfelelő alklamazáson keresztül érhető el. Egy hacker-nek szüksége van az adatbázis és az adatokat használó alkalmazásra is az adatok visszafejtéséhez.

-Negatívuma lehet, ha ezt a titkosítási módszert szeretné alkalmazni egy cég, akkor maguknak kell implementálniuk, ami egy nem informatikai cég esetében nehézkes lehet.

-Másik hátránya, a kulcsok kezelésének az összetettsége is megnövekedhet, ha több különböző alkalmazásnak kell egy adatbázishoz hozzáférnie, írnia, olvasnia.

Hashing

-Érzékeny adatok tárolására használják, mint a jelszavak.

-Egyediek és ismételhetőek, ami azt jelenti, hogy egy szót ugyanazzal a hash algoritmussal transzformálva ugyan azt a titkosított szöveget fogja eredményezni.

-Nagyon nehéz két olyan különböző szót találni, amelyek ugyanazon hash algoritmussal transzformálva ugyanazt a titkosított szöveget eredményezné.

-Egy hash algoritmussal transzformált szöveget visszaalakítani egyszerű, értelmes szöveggé (plain text) már nem lehet, éppen ezért használják jelszavak titkosítására.

-Nem visszaalakíthatósága miatt egyezőség vizsgálatát lehet megnézni két ugyanazon hash algoritmussal transzformált , kódolt szöveg között. Ha két hash egyezik, akkor ugyan az volt a bemeneti szöveg, ha nem egyeznek , akkor különbözőek voltak.

-Úgynevezett salt és pepper –rel szokás a hash-eket ellátni.

-Salt: A titkosítandó szövegrészhez extra szöveg csatolása bonyolultság növeléséhez. Regisztrációkor lehet például az email címet és jelszót kombinálni, majd a kombinált szövegen elvégezni a hash algoritmust. Ilyen adatokat salted hash-nek nevezünk.

-Pepper: Salted hash adathoz , tehát már titkosított adathoz fűz további értékeket. Egy adatbázis esetébe ez általában egy értéket jelent, azaz minden hash-elt adathoz ugyanaz a pepper érték lesz hozzáadva.

Minden titkosítási módszer egy titkosítási algoritmussal történik. Egy titkosítási algoritmus lehet szimmetrikus vagy aszimmetrikus. Ez a két fogalom a kulcs és a titkosított szöveg közötti kapcsolatot írja le.

(13.link)

Szimmetrikus titkosítás:

-Az adatok titkosítása az adatbázisba történő mentéskor és visszafejtéskor történik.

-Adat titkosítására és dekódolására egy privát kulcsot használ.

-Az adatok megosztásához a címzettnek rendelkeznie kell a visszafejtési kulcs másolatával. Ez a titkosítás legegyszerűbb, legrégebbi és legismertebb fajtája.

-A hátránya a kulcs megosztásából ered, ha a visszafejtési kulcs kiderül, az adatszivárgáshoz (data leak) vezethet.

-Előnye a sebesség.

Aszimmetrikus titkosítás:

-Egy privát és egy publikus kulcsot használ.

- A publikus kulcs lehetővé teszi, hogy bárki titkosítsa az adatokat.

-A privát kulcs szükséges az adatok dekódolásához. Ez minden felhasználó esetében különböző.

-Biztonságosabbnak vélhető, mivel a privát kulcsok nem kerülnek megosztásra, de ugyanakkor nagyobb teljesítményigényű is.

Kulcskezelés / Key-management

Ez a fogalom a kulcs előállítását, cseréjét, tárolását és használatát jelenti.

Kulcsok nem megfelelő tárolása és kezelése adatszivárgást eredményezhet. Ha a kulcskezelő rendszer valamilyen oknál fogva elveszti vagy törli a kulcsokat, akkor a titkosított adatok is elvesznek.

Komplexitás szempontjából, minél több alkalmazás adata kerül titkosításra, úgy nő a tárolandó és kezelendő kulcsok száma is.

Kulcsok tárolása a leggyakoribb, hogy egy titkosítási alkalmazás kezeli a kulcsokat és a hozzáférés jelszóval megadásával lehetséges.

Kimondottan a kulcskezelésre számos úgynevezett kulcskezelő rendszert (KMS) létrehoztak. Egy ilyen rendszer magába foglalja a kulcsok generálását, cseréjét, tárolását.

Szimmetrikus vagy aszimmetrikus titkosítási algoritmusnak olyan matematikai folyamatot nevezünk, amikor egy szimpla szöveget (plain text) az adott módszer kódolt szöveggé transzformál (cipher text).

Pár oldallal ez előtt bemutatott módszerek ilyen algoritmusokat használnak adat titkosításához.

Az, hogy egy algoritmus szimmetrikus vagy aszimmetrikus itt is a kulcs és a kódolt szöveg közötti kapcsolatot jelenti.

Szimmetrikus titkosítási algoritmusok közül gyakoriak:

Advanced Encryption Standard (AES)

-Bevált, megbízható titkosítási módszer. Komplex, több fázisból álló matematikai számításokat hajt végre.

-Úgynevezett block cipher, azaz blokk titkosítás. Blokkok mérete 128 bit.

-3 fajtája van, AES-128 – 128 bit nagyságú titkosítási kulcsot alklamaz.
AES-182 – 192 bit nagyságú titkosítási kulcsot alkalmaz.
AES-256 – 256 bit nagyságú titkosítási kulcsot alkalmaz.

-Hátránya, hogy szoftveresen nehezen implementálható, illetve komplexitása és nagysága miatt teljesítményi ideje is nagy.

Data Encryption Standard (DES)

-Block cipher, egy blokk 64 bit nagyságú.

-64 bit nagyságú sima szöveg titkosítás után 64bit nagyságú titkosított szöveget eredményez.

-16 sorozatot véget el matematikai számításokból, mindegyikhez külön titkosítási kulcsot használ.

-Kulcsok mérete 56 bit.

-Hátránya lehet, hogy a titkosítás és visszafejtés ugyanazzal az algoritmussal és kulcsokkal történik.

Triple DES

-Működése megegyezik a DES működésével.

-3x16 sorozatot végez, így biztonságosabbnak mondható.

-Hátránya lehet, hogy a titkosítás és visszafejtés ugyanazzal az algoritmussal és kulcsokkal történik.

Blowfish

-Block cipher, 64 bit méretű blokkok.

-Kulcsok mérete 32 bittől 448 bit-ig terjed, változó méretűek, így lehetőséget nyújt személyes és ipari felhasználásra is.

-Sokkal gyorsabb mint a DES és Triple DES.

Aszimmetrikus titkosítási algoritmusok közül gyakoriak:

RSA

-Azon az elven alapul, hogy a nagy számok szorzása könnyű, de a nagy számok tényezőkre bontása nehéz.

-A publikus kulcs két számból áll, ami közül az egyik két nagy prímszám szorzata.

-A privát kulcs egyik száma ugyanennek a két prímnek a szorzata.

Digital Signature Algorithm (DSA)

-Digitális aláírások titkosítására használt szabvány. Digitális üzenet hitelesítésére is alkalmas.

-Moduláris exponenciáláson (modular exponentiation) alapul, a diszkrét logaritmus (discrete logarithm) problémájával együtt.

-Privát kulcsot egy üzenet digitális aláírásának generálására használják. Ellenőrizni az aláíró publikus kulcsával lehetséges.

Elliptic Curve Cryptography (ECC)

(15.link)

-RSA-val vetélkedő aszimmetrikus titkosítási módszer.

-Kulcs generálása matematikai elliptikus görbék segítségével történik.

-„Véges mező feletti sík görbe, amely az egyenletet kielégítő pontokból áll: $y^2=x^3 + ax + b$.”

-Alapvető kulcs méret 256 bit, de görbétől függően változhat.