1. Ismertesse az üzleti intelligencia rendszerekben használt alapfogalmakat (adat, információ, jellemző KPI, esemény). Milyen típusú adatforrásokat tudunk megkülönböztetni? Mik a fő eltérések az OLTP és a BI rendszerek között?

Alapfogalmak:

- adat: nyers tény (például augusztus 13-án 35 fok volt)
- információ: újdonság, amit az adatokból ki tudunk következtetni (augusztus 13-án volt egész nyáron a legmelegebb)
- jellemző (KPI - key performance indicator): mérhető, amivel leírjuk az eseményeinket (pl. az eladott termékek darabszáma)
- esemény: az adat egysége, mérhető jellemzőkkel bír amelyek általában additívak, és értelmezhető rájuk valamiféle aggregáció. Tartozhatnak hozzá egyéb attribútumok (mikor, hol, ki stb.).

Adatforrás típusok:

- strukturált: meghatározott struktúra és tartalom (excel táblázat, database)
- félig strukturált: valamilyen szintű struktúra, de a tartalom azon belül kötetlen (XML fájl, log fájl)
- struktúrálatlan: képek, videók, szabad szövegek

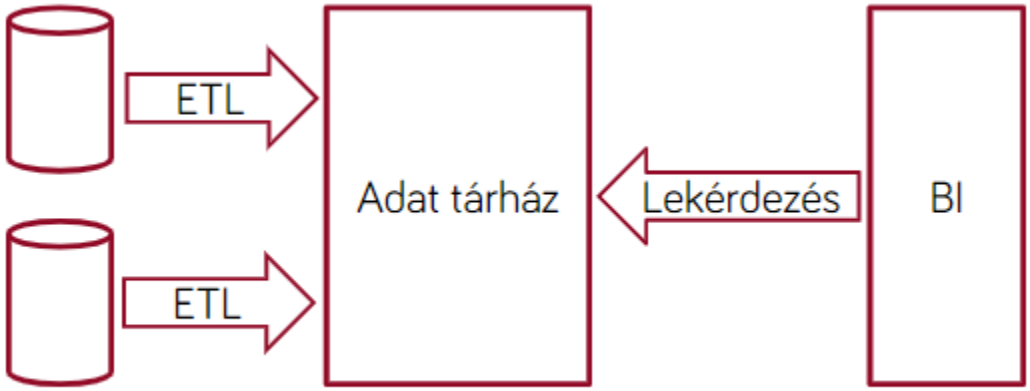
ERP (Enterprise Resource Planning) és BI (Business Intelligence) rendszerek közötti különbségek:

- stratégia kialakítása vs működés könnyítés: BI eszközök a stratégia döntéshozást segítik, az OLTP rendszerek a mindennapos működésben vesznek részt. A BI segítségével analizálhatók a különböző vállalati paraméterek. Az ERP a működéshez szükséges adatokat tartalmazza, ezek elérését segíti.
- OLAP vs OLTP: a BI rendszer az OLAP (Online Analytical Processing) rendszerre épít, amely az adatok elemzését és vizualizálását segíti. Ehhez új adatot hozzáadni nehéz, feltöltése lassú, de utána a sok adatot érintő lekérdezések is gyorsak. Historikus adatokat is tartalmaz.
  Az ERP az OLTP (Online Transaction Processing) rendszerre épít, aminek az alapja a tranzakció, fontos tulajdonsága hogy nagy sebességgel képes tranzakciókat rögzíteni, de sok adatot érintő lekérdezések lassabbak. Aktuális információkat tartalmaz, kevés historikus adatot tárol.

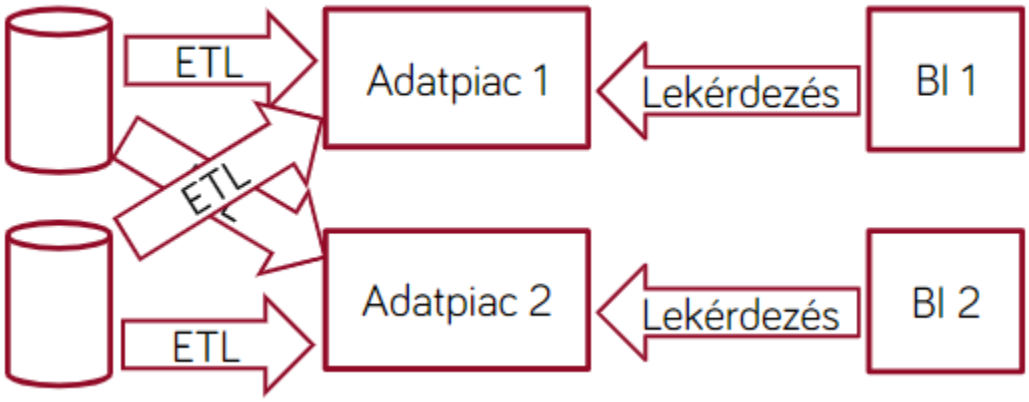
2. Ismertesse az üzleti intelligencia rendszerek általános felépítését, valamint a Kimball és Immon féle architektúrákat. Ismertesse az építőkövekt funkcióit, és mutasson be konkrét szoftvereket vagy technológiákat amelyek az adott funkciót megvalósítják.

- könnyű hozzáférés: a felhasználók számára az adat hozzáférhető és érthető
- konzisztens: sok helyről származó adatok egységesítése, inkonzisztencia szűrés, normalizálás
- megbízható
- rugalmas: új kérdések, új adatok esetén könnyű legyen bővíteni
- teljesítmény
- biztonság

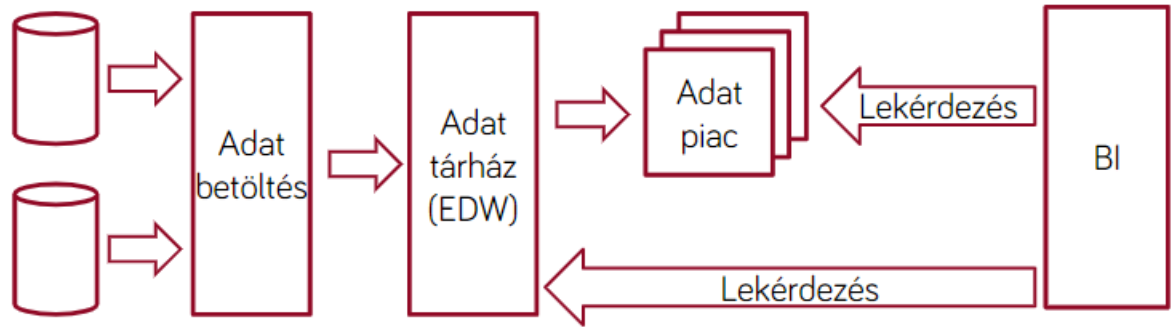
Kimball (és általános) architektúra:



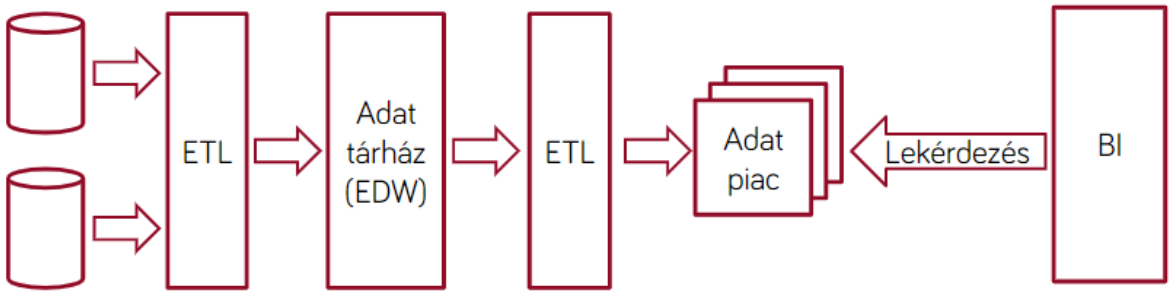
Független adatpiacok:



Immon architektúra:



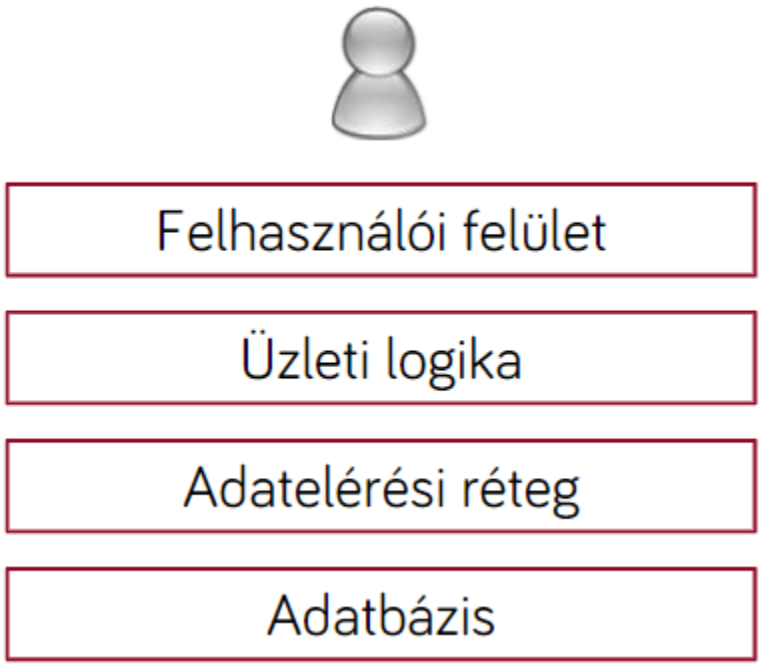
Hibrid Kimball-Immon architektúra:



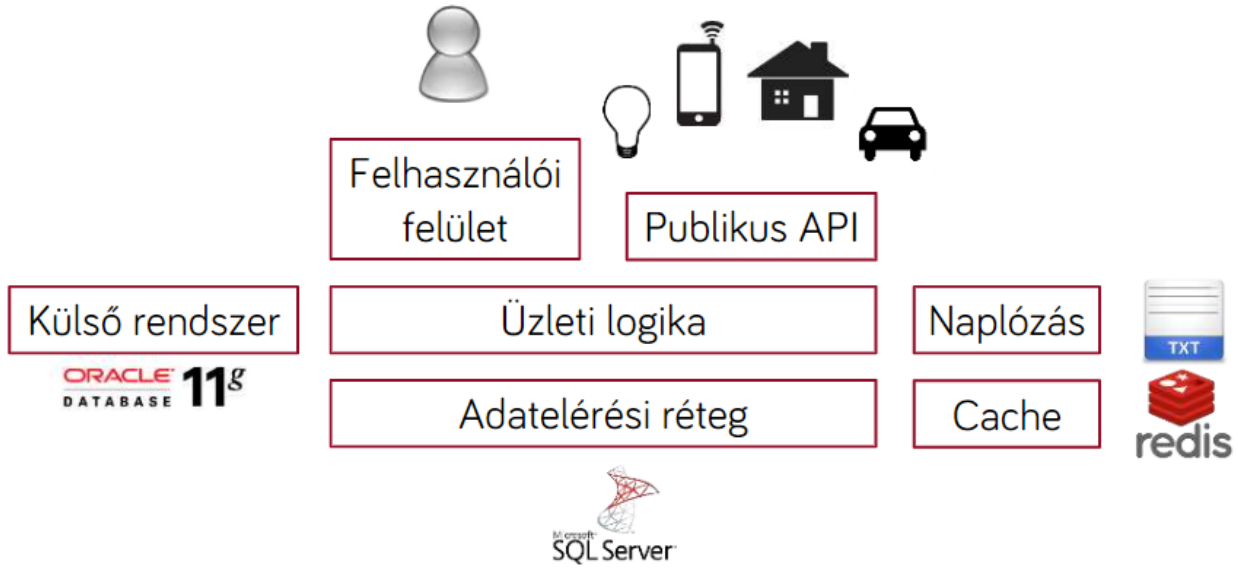
- Adatbázisok (databases): pl. termék katalógus, raktárkészlet stb.
- **Adattárház (data warehouse):**
  - normalizált (3NF jellegű) modell. [3NF azt jelenti, hogy az egyes attribútumok csak a kulcstól függenek] Kombinálja az adatokat, tisztítva, normalizálva, egy központi helyen teszi elérhetővé.
  - Microsoft SQL Server (SS)
- **Adatpiac (data mart):**
  - nem teljes adattartalom, felhasználási területre szabva, esetleg előre feldolgozva. Felhasználási célnak megfelelő tárolási modell, akár előre aggregálva.
  - Microsoft SSAS (Analysis Services): online adatelemzés (OLAP, Tabular - memóriában tárolt), adatbányászat (döntési fa, neuárlis háló, osztályozás, klaszterezés).
- **BI rendszer:**
  - adatokhoz való könnyű hozzáférés, intuitív kezelőfelület. Ezt kell változtatni ha az igények változnak.
  - Microsoft SSRS (Reporting Services): táblázatok, diagrammok, exportálás, integrálható saját alkalmazásba.
  - Microsoft SSAS:
- **ETL (extract, transform, load)**:
  - adatkonzisztencia vizsgálata, ha a forrásadat változik, ezt kell változtatni.
  - Microsoft SSIS (Integration Services): adatok importálása, exportálása

3. Ismertesse, milyen kihívásokkal szembesülnek a modern alkalmazások az adattárolás és feldolgozás terén, amire a tradicionális relációs adatbázisok nem minden esetben nyújtanak megoldást. Mutassa be, milyen módszerekkel kezelhetőek ezek kihívások. Térjen ki a 'near data processing'-re és a dimenzionális adatmodellre is.

'Klasszikus' többrétegű alkalmazások:



Modern alkalmazások:



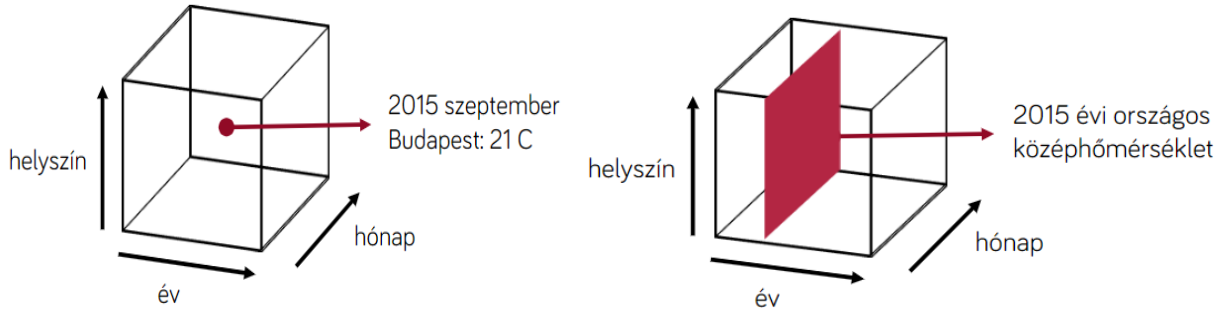
Kihívások:

- heterogén tárolás (több rendszerben tárolt adat)

- változatos formátumok (relációs, szöveges, gráf stb.)
- nagy adatmennyiség (BigData, Folyamatosan generálódó új adat)

Megoldások:
- új adattárolási megoldások (memória adatbázisok, NoSQL adatbázisok)
- ‘Near data processing’:
    - feldolgozás idejét az adat hozzáférés ideje határozza meg
    - nagy adathalmazok utaztatása költséges
    - adat helyett vigyük a feldolgozó logikát az adathoz
- masszívan párhuzamos feldolgozás

Dimenzionális adatmodell:
Többdimenziós kocka, melynek dimenziói a mért tartományok. A kocka egy pontja a dimenziók adott pontjában mért érték. Célja a gyors lekérdezések, aggregáció támogatása, és hogy az adat könnyen értelmezhető legyen.



Megvalósítása: OLAP (Online Analytical Processing)
- többdimenziós adatbázis
- nem relációs
- hatékonyabb mint relációs adatbázis, ehhez a rendszer előre kiszámol összesített értékeket

Műveletek:
- Drill down: aggregált értékek felbontása komponensekre. A Drill down ellentéte. (pl. éves középhőmérséklet felbontása napi középhőmérsékletre).
- Roll up: komponensek aggregálása.
- Pivot: más tengely szerinti nézet. (éves középhőmérséklet évenként → éves középhőmérséklet városonként)
- Dice: egy részkocka kiválasztása (február márciusban, 2014-2015, Esztergom, Budapest hőmérséklete)
- Slice: egy szeletet kiválasztunk (2014 év hőmérséklet információi)

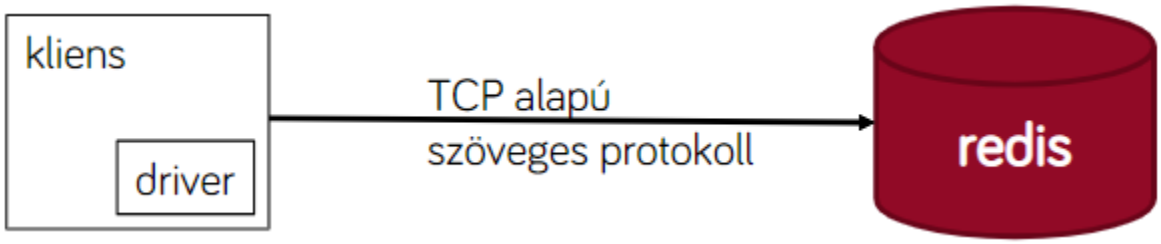
4. Mutassa be a memória adatbázisok működését, ismertesse, miben különböznek a tradicionális adatbázisoktól. Ismertessen egy választott memória adatbázist. Mutassa be az oszlop alapú- és a sor alapú tárolás módot, ismertesse előnyeiket és hátrányaikat.

Memória adatbázisok:

- adatok elsődleges példánya a fizikai memóriában (hagyományosan a diszken szokott lenni)
- gyorsabb adatelérés, tranzakció feldolgozás
- alkalmazás logika változik: disc IO alrendszert ki lehet hagyni, de a tartósság biztosítására egyéb rendszereket kell építeni (biztonsági mentések stb.).
- de induláskor fel kell tölteni az adatbázist
- fontos probléma a tartósság: áramkimaradás esetén biztosítani kell hogy ne vesszen el adat. (naplózás, elemes RAM)

Redis:

- struktúra:
  - data structure server: you can directly access and manipulate your data through direct commands
  - kulcs-érték tár:
- perzisztencia:
  - memóriában tárolt (gyors), de snapshotok a lemezre (konfigurálható)
- architektúra:



- adattípusok:
  - változatos (pl. string, bitmap, hashes - python dict, lists, sets)
  - atomi adatmódosítások
  - TimeToLive: beállítható egy elévülés az adatoknak.
- replikáció (ugyanaz több fizikai gépen):
  - aszinkron módon a háttérben
  - master-slave architektúra
  - nincsen konzisztencia garancia
- tranzakció támogatva van, de nem atomi
- biztonság: trusted environment modell (nincsenek jogok, authorizáció), 1 clear text jelszó az authentikáció

Oszlop- és sor-alapú tárolási mód:

1	A	Q
2	B	W
3	C	Z

- sor: tipikusan OLTP rendszerekben. Sor szintű hozzáféréshez praktikus.

1	A	Q	2	B	W	3	C	Z

- oszlop: OLAP feldolgozáshoz. Oszloponkénti hozzáféréshez praktikus (pl. aggregáció oszlopok mentén).

1	2	3	A	B	C	Q	W	Z

5. Ismertesse a NoSQL adatbázisok típusait. Mutassa be általánosságban az egyes fajtákat és ismertessen egy-egy konkrét szoftvert. Ismertesse a CAP tételt.

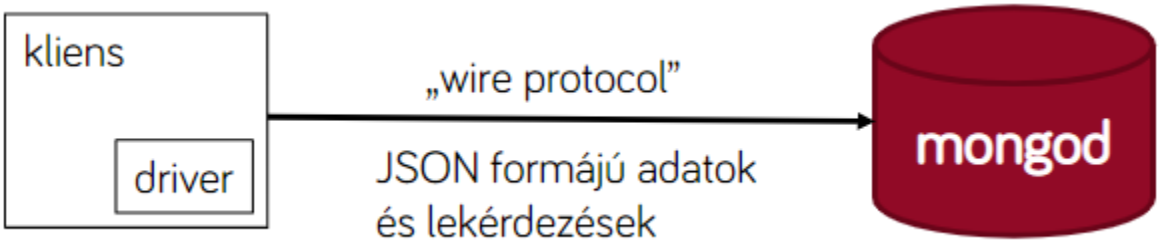
NoSQL adatbázisok típusai:

- Dokumentum (MongoDB)
- Kulcs-érték (Redis)
- Oszlop alapú (Cassandra)
- Gráf (neo4j)

Dokumentum:

Documents are the main concept in document databases. The database stores and retrieves documents, which can be XML, JSON, BSON, and so on. These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values. The documents stored are similar to each other but do not have to be exactly the same. Document databases store documents in the value part of the key-value store; think about document databases as key-value stores where the value is examinable. (lényegében olyan kulcs-érték adatbázis ahol van sejtésünk hogy mi a value).

MongoDB:



logikai felépítés: klaszter → szerver → adatbázis → gyűjtemény → dokumentum

- dokumentum: elemi tárolás egység, JSON fájl, kulcs-érték párokat tartalmaz.
  - érték a legtöbb megszokott típus lehet
  - kulcs: egyedi string
- gyűjtemény:
  - relációs adatbázisban a tábla hasonlít hozzá
  - nincs sémája, csak ‘hasonlóak’ a dokumentumok

Egyéb tulajdonságok:

- atomi módosítás dokumentum szinten
- komplex lekérdező nyelv (tartomány, kulcs, szöveges, aggregált keresés)
- Indexeket tudunk definiálni (gyűjtemény szinten).
- Sharding: adatbázis feldarabolása, alkalmazás számára transzparens. Horizontális skálázás és hibatűrés javítása.

Kulcs-érték:

Key-value stores are the simplest NoSQL data stores to use from an API perspective. The client can either get the value for the key, put a value for a key, or delete a key from the data store.

The value is a blob that the data store just stores, without caring or knowing what's inside; it's the responsibility of the application to understand what was stored. Since key-value stores always use primary-key access, they generally have great performance and can be easily scaled.

Oszlop alapú:

Column-family databases store data in column families as rows that have many columns associated with a row key. Column families are groups of related data that is often accessed together. For a Customer, we would often access their Profile information at the same time, but not their Orders.

Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns. The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows.

When a column consists of a map of columns, then we have a super column. A super column consists of a name and a value which is a map of columns. Think of a super column as a container of columns.

Cassandra:

- Elosztott adatbázis, minden csomópont egyenrangú - így lineárisan skálázható.
- Támogatja a replikációt.
- Konsziztencia: ‘eventually consistent’. Írás után régi értékek még léteznek a replikált csomópontokon.
- Gyűjtemény oszlopok (super columns)
- API: CQL (Cassandra Query Language)
    - SQL szerű szöveges parancsok
    - join, beágyazott lekérdezések nem támogatottak

Gráf alapú:

Entitások és köztük lévő kapcsolatok. Speciális célokra, speciális lekérdező nyelv kell hozzá.

CAP tétel:

Elosztott rendszer a három alapvető képesség közül legfeljebb kettőt tud megvalósítani.

- Consistency: minden csomópont azonos állapotot lát
- Availability: minden kérésre érkezik válasz.
- Partition-tolerance: hálózati és egyéb hibák esetén is működőképes marad a rendszer. (The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.)

Skálázás miatt P nem elhagyható, C és A közötti egyensúlyt kell megtalálni.

6. Mutass be, miről szól az adatbányászat. Milyen adatbányászati problémákat ismer? Milyen tipikus feladatokat tudunk megoldani adatbányászati eszközökkel üzleti intelligencia rendszerekben?

Nemtriviális, rejtett összefüggések és információk feltárása nagy adathalmazokban, és ezek felhasználásával ismeretlen adatok becslése.

Vektor-tér modell: egy megfigyeléshez egy vektor tartozik, aminek a dimenzióját a feature-rök adják. Lehetnek idősorok, gráf reprezentáció is. Supervised esetben az egyes pontokhoz label is tartozik.

Tanítóhalmaz, teszthalmaz, validációs halmaz.

Adatok lehetnek: címkézettek, címkézetlenek. Folytonosak, diszkrétek.

Adatbányászat lépései:

- adattisztítás (hiányzó adatok, outlierek, zajszűrés)
- mintavételezés (ha túl sok az adat)
- adattranszformáció (dimenziócsökkentést, normalizálás, standardizálás)
- algoritmus futtatása
- kiértékelés

Felügyelt, felügyelet nélküli tanítás.

Osztályozás, regresszió, klaszterzés.

Tipikus algoritmusok:

- neural networks
- k-means
- linear, logistic regression
- random forest
- decision tree
- support vector machine

Kiértékelés:

- osztályozás: precision, recall, F-score
- regresszió: átlagos négyzetes hiba, abszolút hiba
- klaszterezés: nehéz

Minták, trendek feltárása.

7. Ismertesse, milyen célokra használunk statisztikai szoftvereket. Milyen funkciókat biztosítanak az ilyen programok? Milyen feladatokat lát el egy 'data scientist'?

Statisztikai és numerikus számításokat végzünk velük. Saját script nyelvvel rendelkeznek. speciális adattípusokkal (vektor, mátrix), és alapvető nyelvi elemekkel a matematikai műveletekhez. Támogatják az adatvizualizációt.

Funkciók:

- Speciális adattípusok (vektor, mátrix)
- Alapvető nyelvi elemek matematikai műveletekhez.
- Adatvizualizáció
- Gyakori funkciókra beépített megoldások (CSV fájl beolvasása)
- Nagyobb numerikus pontosság

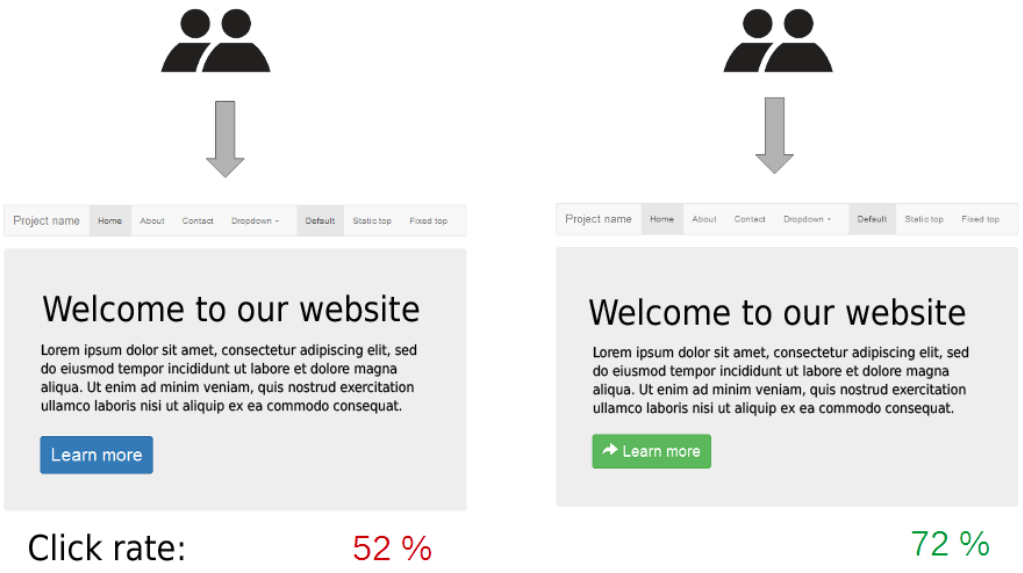
R nyelv:

- script nyelv
- nem OO
- interpretált kód
- gyengén típusos

Data Scientist:

feladatai:

- adatfeldolgozás, vizualizáció
- A/B tesztelés



- trendek, összefüggések felismerése verifikálása
- adatelemzés
- matematikai, statisztikai, adatbányászat, szöveg- és kép feldolgozás

8. Ismertesse a “big data 3V”-jét. Mutasson példákat, ahol nagy adat keletkezik, és ahol annak feldolgozása kihívás. Ismertesse az általános big data stacket.

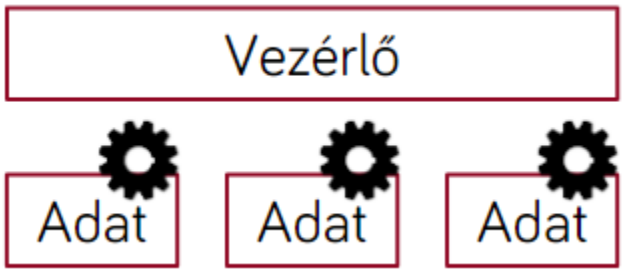
3V:

- volume: nagy adatmennyiség
- velocity: gyorsan
- variety: nagy változatosság.

Példák ahol sok adat keletkezik:

- digitális lábnyom: internetes aktivitás (history), beszélgetések (facebook, twitter, skype), fotók, videók, IoT, szenzorok (pl. GPS, gyorsulás szenzor)
- Web2: személyes felhasználói élmény (termék ajánlás, reklámok), felhasználói szokások, spam elemzés stb.
- Orvosi diagnosztika: viselhető eszközök, hosszú távú mérések, fertőzések terjedésének vizsgálata
- Internet of Things

Általános big data stack:

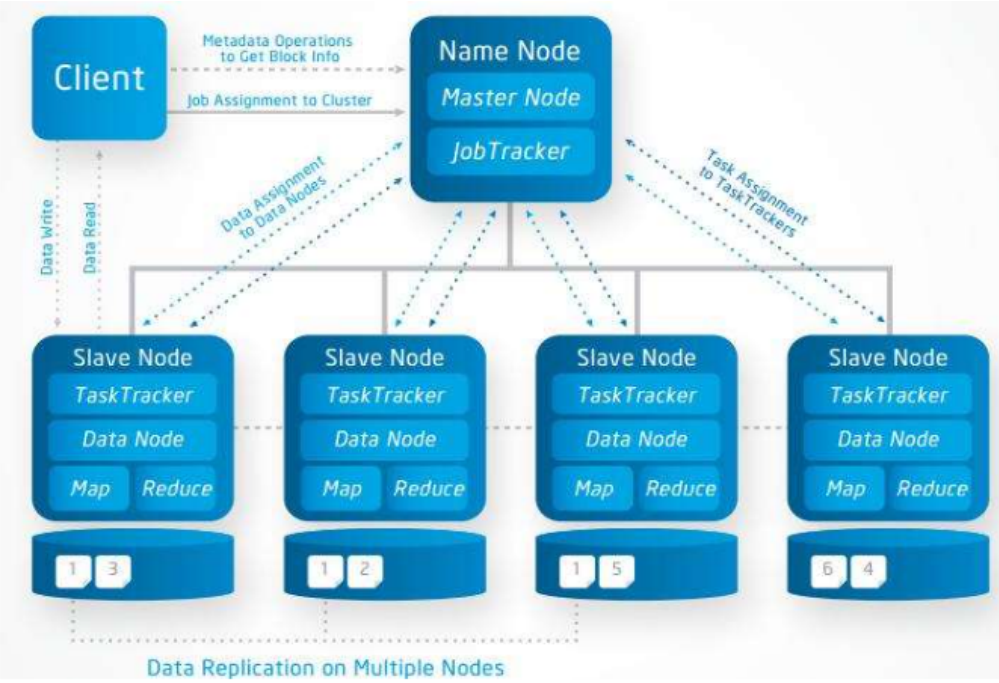


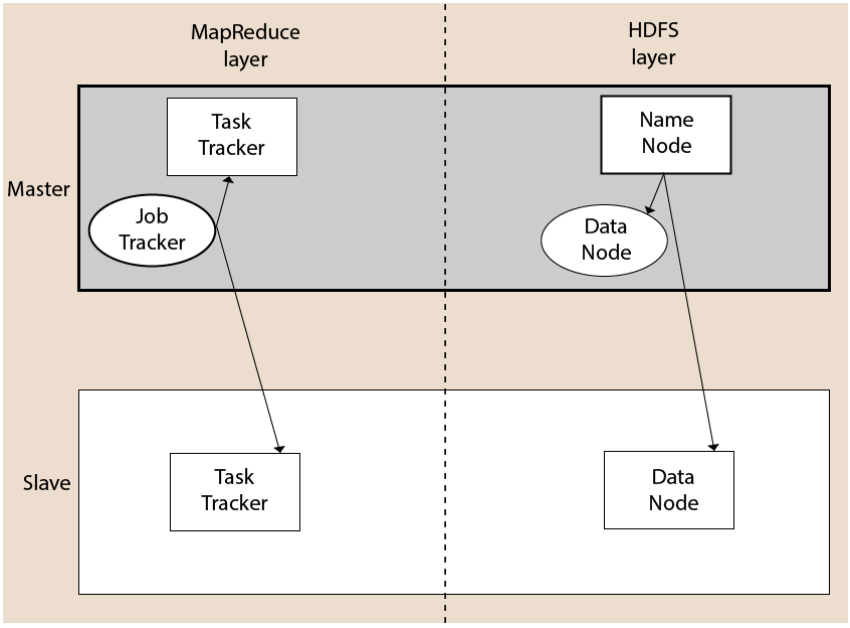
- Megjelenítés:
  - rendszer egésze feletti menedzsment
  - feladatok beadás, státusz ellenőrzés
  - eredményekhez hozzáférés, vizualizáció
- Feldolgozó infrastruktúra:
  - elosztott rendszer feletti algoritmusok
  - a platform szolgáltatásaira épül
  - beadott feladat végrehajtásáért felelős (időzíti, futtatja, elosztja a platform gépei között, szervezi a feladat szétbontását, a részeredmények összefésülését)
- Adat:
  - nem funkcionális réteg
  - platform réteg szolgáltatja a feldolgozó rétegnek
  - near data processing
  - egyazon rendszer sok független adat, feladat feldolgozására képes
- Platform infrastruktúra:
  - számítógép klaszter egy egységként, kezeli a klaszterbe tartozó csomópontokat

    - redundáns, megbízható, meghibásodásra nem érzékeny
- Adat tárolás:
    - Nagy mennyiségű adat
    - műveletek: írás - új adat hozzáadása (ritkán, de nagy mennyiségben), adat olvasás (elejétől a végéig)
    - diszken tárolt, redundánsan, elosztottan
- Fizikai infrastruktúra:
    - szempontok: teljesítmény, rendelkezésre állás, skálázhatóság, költség
    - relatív olcsó és megbízható hardverek, klaszterekbe szervezve.

9. Mutassa be a Hadoop-ot. Térjen ki az architektúrájára, skálázhatóságára, az adattárolási megoldására, a feldolgozási modelljére. Ismertessen olyan Hadoop komponenseket, amelyek az alap rendszerre épülnek kiegészítve azok működését.

architektúra:





Job Tracker:

- The role of Job Tracker is to accept the MapReduce jobs from client and process the data by using NameNode.

- In response, NameNode provides metadata to Job Tracker.

Task Tracker:

- It works as a slave node for Job Tracker.
- It receives task and code from Job Tracker and applies that code on the file. This process can also be called as a Mapper.

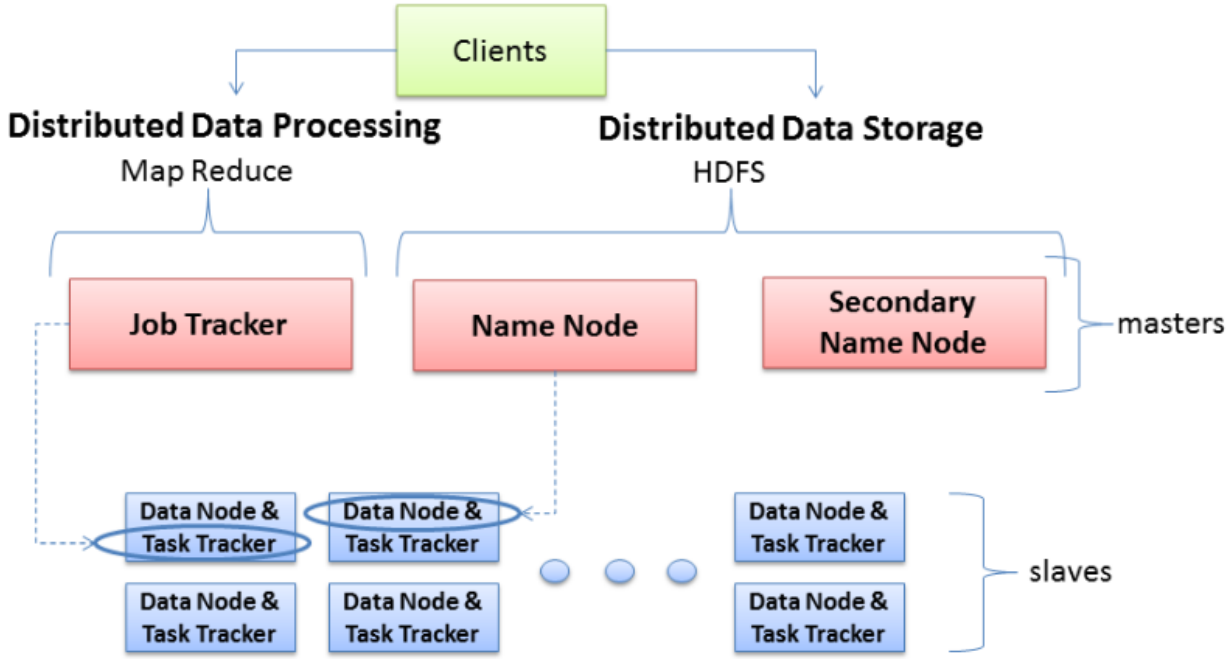
skálázhatóság:

ha új slave-node-okat adunk a clusterhez a teljesítmény is növekedni fog

adattárolási megoldás:

HDFS (Hadoop Distributed File System):

- hibatűrő, blokk alapú, elosztott fájlrendszer
- blokkok replikálása (default replikációs faktor 3)
- nagyméretű fájlok tárolása
- átlagos PC-ken is futhat, így költséghatékony



Name Node:

- általában 1/klaszter
- metaadatok tárolása:
  - Namespace Image (NSI) - lenyomat a fájlokról
  - edig log: legutóbbi NSI-hez képest történt változások, időnként hozzáfűzi
- Data Node-ok heartbeat-jeinek figyelése

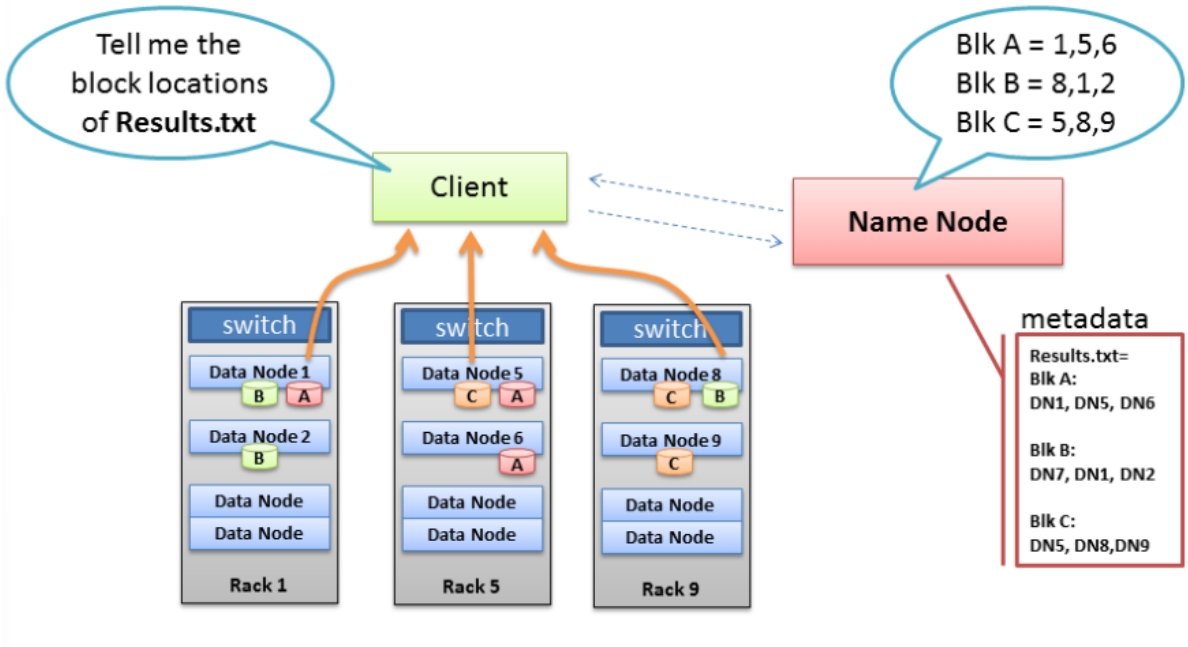
Secondary Name Node (SNN):

- feladata NN segítése az NSI karbantartásában, NSI backup, edit log NSI-hez fűzése

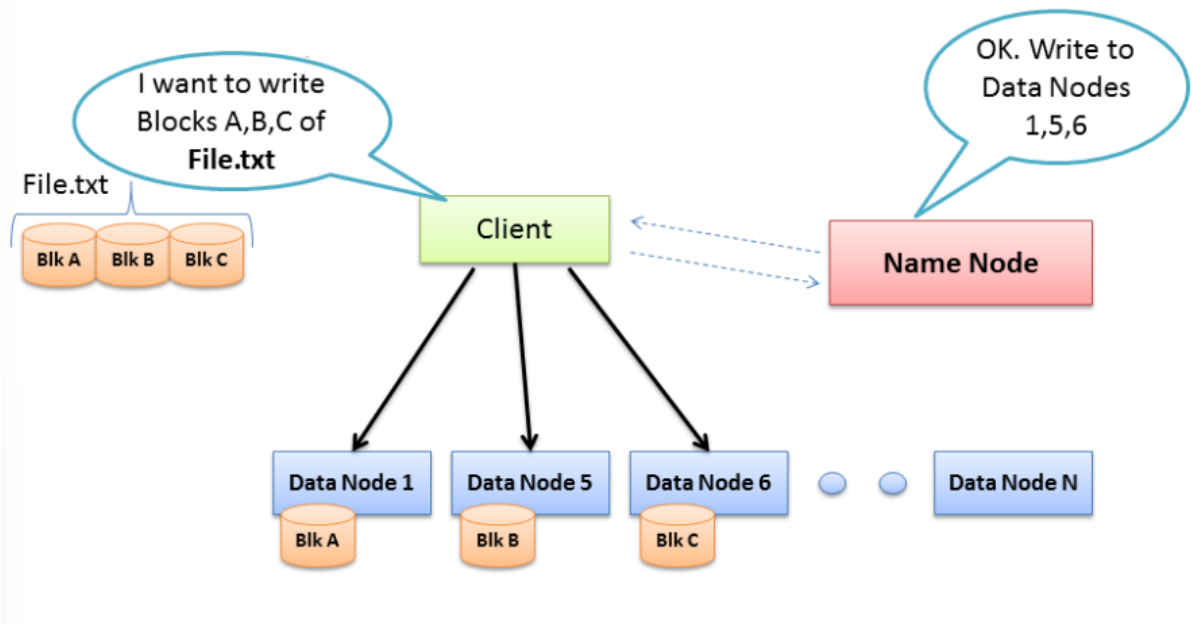
Data Node:

- fájlok blokkjainak tárolása
- report NN felé: Heartbeat, miket tárolok

HDFS olvasás:



HDFS írás:



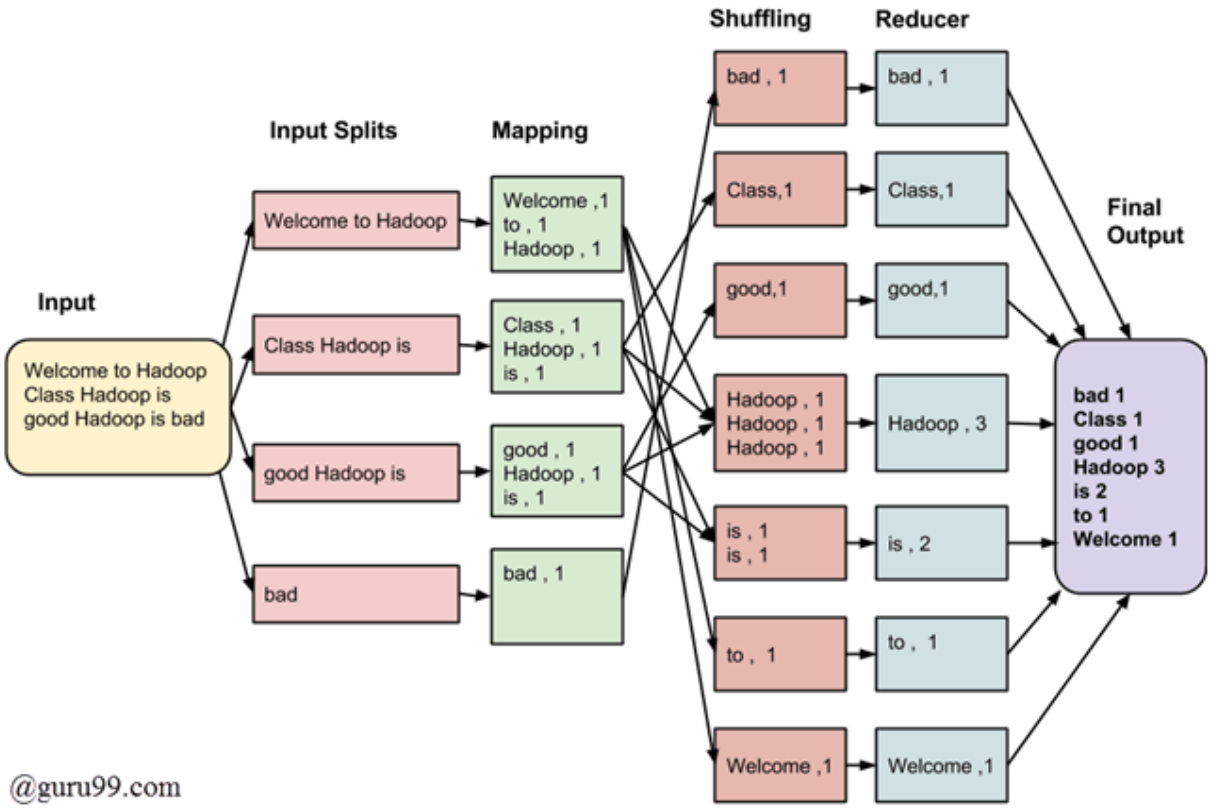
feldolgozási modell:

YARN (Yet Another Resource Negotiator):

- erőforrás kezelés
- az alkalmazásoknak erőforrás rendelés, életciklus kezelés

MapReduce:

- kulcs-érték párok
- 3 lépés:
    - Map: a worke node-ok végrehatják a saját adathalmazukon a map() fv-t
    - Shuffle: adatok csoportosítása kulcs szerint és küldés a reduce-rekhez
    - Reduce: reduce függvény végrehajtása kulcsonként párhuzamosan



Hive

Adattárház megoldás, nagy adathalmazok kezelése. A Hadoop összekötése a SQL világgal.

- táblák létrehozása - séma kényszerítése fájlokra olvasási időben
- lekérdezések, majdnem SQL kompatibilis (HiveQL) → a scriptek MapReduce-okká fordulnak

Spark

Általános végrehajtó motor elosztott adatfeldolgozáshoz. Java, Scala, Python támogatás.

RDD - Resilient Distributed Dataset: elosztott párhuzamosan feldolgozható adathalmaz, bármilyen objektumokat tartalmazhat. Immutable, így a transformation-kor új RDD jön létre.

Műveletek:

- transformation: lazy execution - only when action called, produce RDD (pl. map, filter)
- action: egy értéket kiszámít az RDD elemein ezzel tér vissza, olyan művelet ami nem RDD-t generál. (pl. count - hány eleme van az RDD-nek, collect - visszaadja az RDD összes elemét).

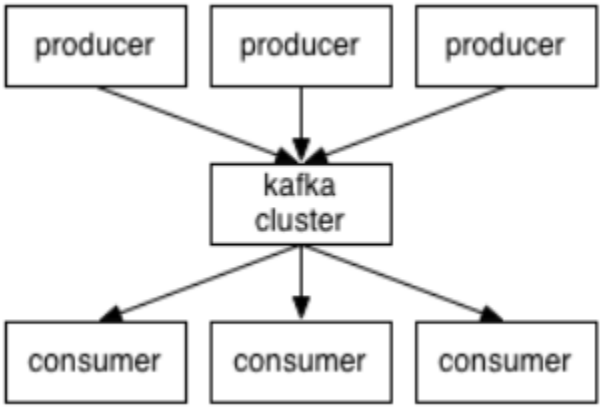
Oozie:

workflow készítő és ütemező eszköz, ideális ETL előállításához. DAG-ként definiálhatunk folyamatokat. Időközönként vagy új adat érkezésekor futhatnak.

Kafka:

Elosztott publis-subscribe üzenetküldő rendszer.

- gyors: egy kafka bróker több ezer klienst tud kiszolgálni
- megbízható: üzeneteket lemezre menti és replikálja
- skálázható: az adatfolyamokat szükség esetén elosztja a node-ok között



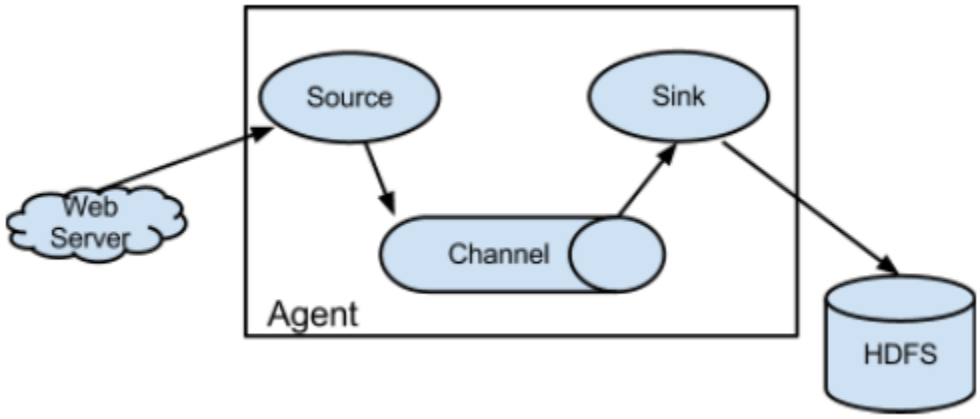
- Producer: adatokat küld a Kafka-nak, általa megjelölt topic-ba
- Topic: az üzenetek topic-okra vannak osztva, topic-ba lehet üzenetet küldeni és fel lehet rájuk iratkozni
- Consumer: feliratkozik topicokra, és kiolvassa az üzeneteket.
- Broker: kafka instance

Use case-ek:

- Üzenetsor
- Log gyűjtés

Flume

Elosztott szolgáltatás nagymennyiségű adat gyűjtésére. Megbízható.



- Source: ide érkezik az adat, (HTTP, Kafka)
- Channel: tárolja, míg a sink-en keresztül ki nem kerül a rendszerből (Memory, JDBC, File)
- Sink: HDFS, Hive
- Event: a rendszerben levő adat alapegysége, header + adat
- Agent: adatfolyamok kezeléséért felel
- interceptorok: minden event-re lefutnak (transzformáció, filterezés, adminisztráció)

Sqoop

- Adatok mozgatása különböző források között.
- MapReduce alapú
- párhuzamosan dolgozik több Data Node-on
- SQL adatbázisbeli adat betöltése HDFS-re.

10. Ismertesse a tény-dimenzió modellezés alapelveit. Adjon példát tény és dimenzió táblára. Milyen előnyei vannak egy csillag sémának? Mit nevezünk degenerált dimenziónak? Mit nevezünk aggregációnak, adjon példát.

Tény táblák

A tény táblák (fact tables) a dimenzionális modellezés központi elemei: ezek azok a táblák, ahol az adott üzleti folyamat számszerű mértékei szerepelnek. Például egy üzletlánc napi eladásait reprezentáló táblában ez a mérték lehet az eladott mennyiség, vagy az értük kapott pénzösszeg. Minden nap, bármelyik boltban bármelyik termék értékesítésre kerül, készül egy bejegyzés is. A dimenziók ezen listája határozza meg a tény tábla finomságát, felbontását.

Egy adattárház szempontjából a leghasznosabb mértékek számszerűek és összeadhatóak, mivel igen ritka az az eset, amikor egyetlen sorra kíváncsi a felhasználó a tény táblából. Éppen ellenkezőleg, általában a sorok százezreinek az aggregált értékére kíváncsi (az elmúlt hónapban eladott termékek mennyisége, bevétel, stb.). A fenti példában az eladott mennyiség és a pénzösszeg is összeadható bármelyik dimenzió mentén.

Nem minden tényadat összeadható, léteznek részlegesen összeadható (semi additive) mértékek, amelyeket csak bizonyos dimenziók mentén lehet összeadni, és nem összeadható mérték ek is. Például egy raktárkészlet aktuális állapotát vagy számlák aktuális egyenlegét reprezentáló tény táblák tipikusan ilyen részlegesen összeadható adatokat tartalmaznak, ugyanis értelmes összeadni a számlaegyenlegeket például ügyfelek szerint, de értelmetlen az idő szerint. Ilyen esetekben a legcélszerűbb megközelítés az átlagolás: az adott periódusra szóló átlag - egyenleg, vagy átlagos raktárkészlet.

Dimenziók

A dimenziók a tény táblák kísérői. Ezek a táblák tartalmazzák a szöveges leírásait az adott üzleti folyamatnak. Egy jól megtervezett dimenzionális modellben egy dimenzió táblának lehető legmagasabb számú oszlopa vagy másnéven attribútuma van, ugyanis ezek az attribútumok játszanak a lekérdezéseknél, elemzéseknél csoportosító, megszorító, vagy magyarázó szerepeket. Emiatt létfontosságú, hogy minél több jól definiált, értelmes dimenzió attribútum legyen, mert ezek határozzák meg az adattárház használhatóságát. A dimenziók jelentik az interfészt az adattárház és a felhasználó között.

Míg a tény adatok főleg számszerűek és folytonos értékkészletűek, addig a dimenzió attribútumok általában szövegesek, és diszkrétek. A dimenziók sokszor hierarchikus kapcsolatot reprezentálnak. Például egy termék egy adott márkához tartozik, amiket kategóriákba sorolunk, és így tovább. A termék dimenzió táblában minden sorban (minden termékre) eltároljuk az adott termék márkáját és a kategória szöveges jellemzését is. Ez épp ellentétes egy normalizált adatbázissal, ugyanis rengeteg redundáns információt tartalmaz. A dimenzió táblák tipikusan denormalizáltak (kivéve snowflake séma esetében), a performancia és az egyszerűség, könnyen érthetőség érdekében feláldozzák a szükséges tárhely mennyiségét.

Általánosan:

- Tény és dimenzió táblák, a lekérdezések legtöbb esetben JOIN műveletek a tény és dimenzió táblák között.
- Dimenziós táblákat tipikusan nem JOIN-olunk.
- Egyszerű kulcsok a tény és dimenzió táblákban.

- A dimenziótáblák tipikusan kisebbek a ténytáblák hoz képest.
- A dimenzióknak tipikusan van surrogate kulcsuk. (adatbázis kezelő által generált unique kulcs):
    - A - tény táblák méretének csökkentése (surrogat kulcs rövidebb általában mint a natural kulcsok - tény táblában sok sor van így ott ez különösen számít)
    - A - független a forrás rendszer kulcs változásától
    - A - leírhatja az entitások időbeli változását
    - D - új kulcs oszlop, extra tárolandó információ
    - D - a generálást meg kell valósítani
- Előnyei:
    - lekérdezések könnyen optimalizálhatók
    - egyszerű a modell bővítése, nincs szükség az adatbázis átrendezésére ha új dimenzió kerül a rendszerbe
    - egyszerűbb struktúra (nincsenek bonyolult kapcsolatok a táblák között) - laikusok által is könnyen kezelhető
    - nem szükséges a meglévő lekérdezéseket módosítani új dimenzió esetén

A modellezés lépései:

- üzleti folyamat azonosítása
- tény adatok azonosítása:
    - események melyek a működést leírják
    - esemény-tény típus: csak egy időbélyeget tartalmaz
    - állapot-tény típus: általában két időbélyeget tartalmaz (új tény létrehozásakor egy előző befejeződik)
- tény adatok granularitásának kiválasztása:
    - túl részletes - túl sok adat, nagy tárhely követelmény, lassabb lekérdezése
    - nem elé részletes - nehéz pontos analízist végezni
- dimenzió adatok azonosítása:
    - reportok részletességét növelik
    - tipikusan szöveges értékek, számok, idő, hely stb.

Aggregációk:

- előre kiszámított speciális lekérdezés, ahol a tény tábla adatai különböző feltételek mentén aggregálásra kerülnek (pl. dátum és idő alapján - össz bevétel egy hónapban)
- összesíti a tény adatokat valamely dimenzió alapján
- performancia növelés - gyorsítja a lekérdezést és az elemzéseket

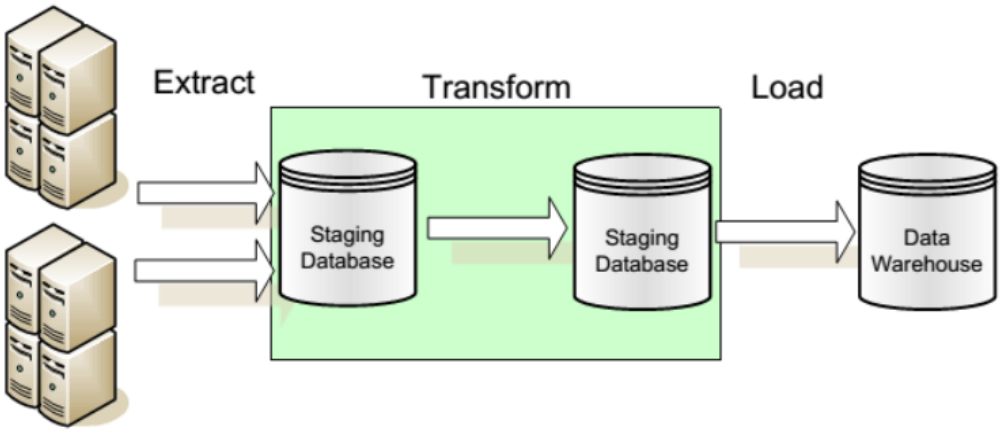
11. Mit nevezünk ETL-nek? Mi a különbség az ETL és ELT között? Sorolja fel a tipikus ETL transzformációs feladatokat! Milyen ETL eszközöket ismer? Hasonlítson össze két választott eszközt előnyök és hátrányok összevetésével. Mit nevezünk diszkretizálásnak, binarizálásnak? Milyen adat tisztítási lépéseket ismer?

Az ETL, ELT folyamatok végzik a különböző típusú és interfészű forrásokból, és különböző formátumokban érkező adatok fogadását, formázását és megfelelő helyre való eljuttatását.

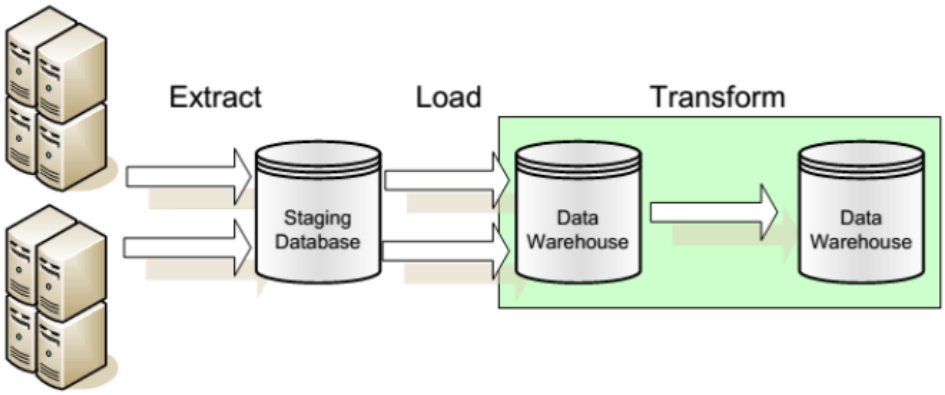
ETL - Extract, Load, Transform

ETL vs ELT:

- ETL: adat kinyerése külső forrás rendszerekből, feldolgozás és átalakítás, majd adat áttöltése a cél rendszerbe. A transzformációt tipikusan az ETL engine végzi.



- ELT: adat kinyerése a forrás rendszerekből, betöltés a staging táblákba, majd átalakítás és tárolás véglegesen a DWH-ba
    - tipikusan nagy mennyiségű adat esetén használjuk és akkor ha az adatbázis motor jól támogatja a transzformációt és a nagy adat mozgatását, átalakítását.
    - a forrás rendszer igénye minimális, hátránya, hogy az adatbázis oldalra nagyobb feladat hárul



Alapfogalmak:

- Forrásrendszer (adatbázis, alkalmazás, file stb.)
- Mappelés: forrás és cél adatok közti viszony

- Metaadat:
    - azon adatok melyek leírják az adatstruktúrákat, objektumokat, üzleti szabályzatokat és folyamatokat. (pl. adatbázis séma, érték megkötések)
    - folyamat során készített működési adatok
- Staging area: ahol az adat fogadásra és feldolgozásra kerül, mielőtt a DWH-ba kerülne.
- Transzformáció: minden adatmanipuláció
    - Adattiszítás: feloldja a forrás rendszerekből érkező adatokban lévő inkonzisztenciákat, és anomáliákat.
    - Data Integration: Data integration is the process of bringing data from disparate sources together to provide users with a unified view.
    - Tipikus feladatok:
        - aggregálás
        - szűrés
        - join
        - rendezés
- Transport: transzformált adat forrás és cél közti mozgatása.
- Cél rendszer (adatbázis, alkalmazás, file stb.)

ETL eszközök:

- Microsoft SSIS (SQL Server Integration Service):
    - grafikus szerkesztő felület
    - fő elemek:
        - vezérlő elem: egyedi feladatok, amik a folyamat során elő- és utófeldolgozást végeznek, vagy metaadatot gyűjtenek.
        - adat folyam elem: egy ETL feladat
    - funkció kiegészítők:
        - esemény kezelők (pl. küld egy emailt bizonyos esemény bekövetkeztekor)
        - konfiguráció és paraméterek: kulcs-érték párok, befolyásolják a package futását.
        - Logkezelés
        - változók: a futás során változók hozhatók létre, ezek később felhasználhatók scriptekben és konfigurációkban.
- Pentaho Data Integration:
    - Kettle: más név a PDI-re
    - Spoon eszköz: grafikus felület (PDI client)
    - Nyílt forráskódú
    - SDK támogatás, testreszabható
- Összevetés:
    - platform: PDI java alapú, így bármilyen platformon futtatható, az SSIS a Visual Studiohoz kötődik, így csak Windowson fejleszthető
    - mindkettő egy sor adatforrást támogat: adatbázisok, flat-file, XML
    - bővíthetőség: SSIS .NET-ben bővíthető (új komponensek, taskok), ehhez használható a VisualStudio. A PDI Java-ban bővíthető de a fejlesztői környezet sokkal kevesebb funkciót szolgáltat.

Diszkretizálás és binarizálás:

Folytonos attribútumok kategórikussá alakítása a diszkretizálás. Ha bináris attribútumokká alakítunk akkor az a binarizálás.

Normalizáció:

Adat normalizációs szint kialakítása: csökkenő redundancia, kisebb táblák, egyszerűbb hatékonyabb áttekintés. A konzisztencia könnyebb fentartása.

Tanulónév	Általános	Középiskola
Tóth Aladár	Napsugár Általános Iskola Gyöngyvirág u. 4.	Zengő Gimnázium, Zerge u. 13. Dobogókő Szakközépiskola, Sziklás u. 44.
Kis Virág	Csillagvár Általános Iskola Kocka u. 14.	Zengő Gimnázium, Zerge u. 13. Baradla Gimnázium, Köves tér 3. Kékes Gimnázium, Havas út 51.
Alföldi Noémi	Csillagvár Általános Iskola Kocka u. 14.	Zengő Gimnázium, Zerge u. 13. Dobogókő Szakközépiskola, Sziklás u. 44.
Végső Albert	Tóparti Általános Iskola Strand út 23.	Baradla Gimnázium, Köves tér 3. Kékes Gimnázium, Havas út 51.
Tóth Aladár	Napsugár Általános Iskola Gyöngyvirág u. 4.	Zengő Gimnázium, Zerge u. 13. Dobogókő Szakközépiskola, Sziklás u. 44.

- 1NF: minden oszlop neve egyedi, nincsenek összetett attribútumok.

Tanulónév	Általános	ÁltCím	Középiskola	KözépCím
Alföldi Noémi	Csillagvár Általános Iskola	Kocka u. 14.	Zengő Gimnázium	Zerge u. 13.
Alföldi Noémi	Csillagvár Általános Iskola	Kocka u. 14.	Dobogókő Szakközépiskola	Sziklás u. 44.
Kis Virág	Csillagvár Általános Iskola	Kocka u. 14.	Zengő Gimnázium	Zerge u. 13.
Kis Virág	Csillagvár Általános Iskola	Kocka u. 14.	Baradla Gimnázium	Köves tér 3.
Kis Virág	Csillagvár Általános Iskola	Kocka u. 14.	Kékes Gimnázium	Havas út 51.
Tóth Aladár	Napsugár Általános Iskola	Gyöngyvirág u. 4.	Zengő Gimnázium	Zerge u. 13.
Tóth Aladár	Napsugár Általános Iskola	Gyöngyvirág u. 4.	Dobogókő Szakközépiskola	Sziklás u. 44.
Végső Albert	Tóparti Általános Iskola	Strand út 23.	Baradla Gimnázium	Köves tér 3.
Végső Albert	Tóparti Általános Iskola	Strand út 23.	Kékes Gimnázium	Havas út 51.

- 2NF: minden másodlagos attribútum (nemkulcs mező) teljes függőségben álljon minden kulcstól.

TOVÁBBTANULÁS

TanulóAZ	KözépAZ
1	1
1	2
2	3
2	4
2	1
3	2
3	1
4	3
4	4

TANULÓK

TanulóAZ	Tanulónév	Általános	ÁltCím
1	Alföldi Noémi	Csillagvár Általános Iskola	Kocka u. 14.
2	Kis Virág	Csillagvár Általános Iskola	Kocka u. 14.
3	Tóth Aladár	Napsugár Általános Iskola	Gyöngyvirág u. 4.
4	Végső Albert	Tóparti Általános Iskola	Strand út 23.

KÖZÉPISKOLÁK

KözépAZ	Középiskola	KözépCím
1	Zengő Gimnázium	Zerge u. 13.
2	Dobogókő Szakközépiskola	Sziklás u. 44.
3	Kékes Gimnázium	Havas út 51.
4	Baradla Gimnázium	Köves tér 3.

- 3NF: egyetlen másodlagos attribútum sem függ tranzitívan egyetlen kulcstól sem.

TOVÁBBTANULÁS

TanulóAZ	KözépAZ
1	1
1	2
2	3
2	4
2	1
3	2
3	1
4	3
4	4

TANULÓK

TanulóAZ	Tanulónév	ÁltAZ
1	Alföldi Noémi	1
2	Kis Virág	1
3	Tóth Aladár	2
4	Végső Albert	3

KÖZÉPISKOLÁK

KözépAZ	Középiskola	KözépCím
1	Zengő Gimnázium	Zerge u. 13.
2	Dobogókő Szakközépiskola	Sziklás u. 44.
3	Kékes Gimnázium	Havas út 51.
4	Baradla Gimnázium	Köves tér 3.

ÁLTALÁNOS ISKOLÁK

ÁltAZ	Általános	ÁltCím
1	Csillagvár Általános Iskola	Kocka u. 14.
2	Napsugár Általános Iskola	Gyöngyvirág u. 4.
3	Tóparti Általános Iskola	Strand út 23.

Adattisztítás:

- Adatban szereplő hibák felmérése - hibás adatok felülvizsgálata, javítása, szűrése.
- Outlier szűrés
- Hiányzó értékek kezelése
- Adatfile szerkezeti épségének ellenőrzése

12. Milyen ETL folyamat tesztelési módszereket ismer? Ismertesse az egyes tesztek előfeltételeit és a teljesülés kritériumait.

ETL teszt fázisok:

- integrációs teszt
- renszer teszt
- regressziós teszt
- performancia teszt

Integrációs teszt:

- célja a mappeléshez szükséges előkövetelmények ellenőrzése
- igazolnia kell hogy megfelelő számú sor került át a forrás rendszerből a cél rendszerbe
- hibakezelés, mapping változók validálhatók
- előkövetelmények:
    - hozzáférés a hálózati könyvtárakhoz
    - implementáció megléte sikeres unit tesztek
    - adat elérhető a teszt környezetben

Rendszer teszt:

- egységbe teszteli a rendszer működését
- end-to-end performancia vizsgálat, inicializálás és inkrementális terhelési statisztikák
- hibakezelés verifikálása
- előkövetelmények:
    - fejlesztés lezárása
    - minden integrációs teszt sikeresen lefutott
    - sikeres migrálás a teszt környezetből a validációs környezetbe
    - adatok és rendszer konfiguráció elérhetősége

Regressziós teszt:

- azután fut le miután egy jelentett issue javításra került - megvizsgálja hogy a javítás helyes volt, nem került-e újabb hiba a rendszerbe
- akkor is ha egy Change Request került megvalósításra
- előkövetelmények:
    - fejlesztés lezárása
    - integrációs tesztek sikeres lefuttatása

Performancia teszt:

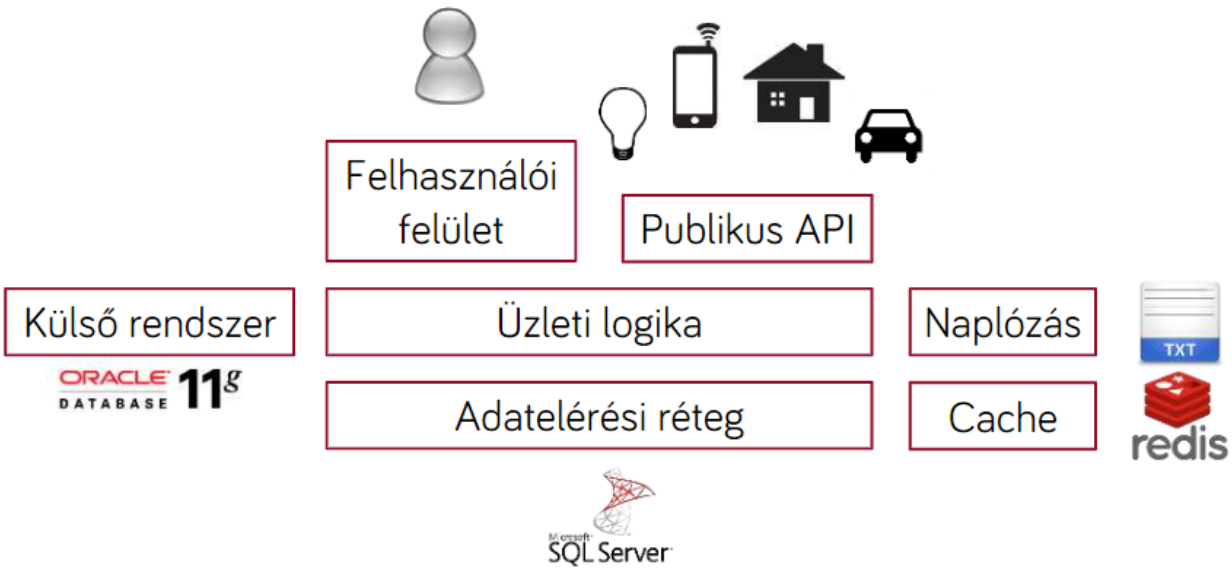
- rendszer teljesítményének vizsgálata meghatározott terhelési szint mellett
- szűk keresztmetszetek feltárása, biztosítja a rendszer elvárt sebességét
- típusai: Load, Stress, Volume stb.

13. Mikre kell ügyelni több adatforrás összekapcsolása esetén? Milyen esetben alkalmazna több adatbázis típust egy rendszerben? Milyen előnyei lehetnek memória, NoSQL és relációs megoldások együttes használatának?

Több adatforrás összekapcsolása:

- ugyanaz az adat több forrásból: például több nap megrendelései különböző fájlokban - ezeket konkatenálhatjuk, esetleg idő szerint rendezve
- összefüggő adatnak különböző nézetei: egy fájlban a megrendelések egy másik adatbázisban a vevők adatati. Kezelni kell az összerendeléseket, konfliktusokat.
    - szükség lehet a különböző adatok összevetésére
- adatforrások eltérő sebességének kezelése
- milyen stratégiával fogadjuk az adatokat a forrásokból?
    - round robin
    - prioritás sor
    - párhuzamos feldolgozás

Több adatbázis típus egy rendszerben:



Akkor van értelme együtt használni őket, ha mindkettő előnyeit szeretnéd élvezni. Pl. a komplex kapcsolatokkal rendelkező adatstruktúrát lehet relációs adatbázisban tárolni, az eseményeket (pl. logokat) pedig NoSQL adatbázisban. Olyanra is láttam példát, hogy a kettő között szinkronizációt alkalmaztak, hogy mindkét helyen ugyan azok az adatok legyenek. A reláció adatbázis felelt azért, hogy az üzleti követelmények teljesüljenek (pl. az objektumok kapcsolataira), a NoSQL pedig a gyors kereshetőséget biztosította pl.: több mezőben egyszerre kereséssel, vagy szinonima kereséssel.

Az egyik fő különbség, hogy számíthatsz arra, szinte bármi, ami támogatja az SQL-t, támogatja az olyan adatokat, mint a triggerek az adatbázisban, azaz szabályokat tervezhetsz az adatbázisba, amelyek célja, hogy biztosítsák az adatok belső következetességét . Például

beállíthatja a dolgokat, így az adatbázis azt állítja, hogy egy személynek kell címmel rendelkeznie. Ha ezt teszi, bármikor hozzáad egy személyt, alapvetően arra kényszeríti Önt, hogy társítsa a személyt valamilyen címmel. Lehet, hogy új címet adhat meg, vagy esetleg meglévő címmel társíthatja őket, de így vagy úgy, a személynek rendelkeznie kell egy címmel. Hasonlóképpen, ha töröl egy címet, arra kényszeríti Önt, hogy távolítsa el az összes olyan személyt, aki jelenleg az adott címen van, vagy hozzárendel minden más címet. Ugyanezt tehetjük más kapcsolatok esetében is, például azt, hogy minden embernek anyja van, minden irodának telefonszámot kell tartalmaznia stb.

Ne feledje, hogy ez a fajta dolog garantáltan atomi módon történik, így ha valaki más az adatbázist nézi, amikor hozzáadja a személyt, akkor egyáltalán nem látják a személyt, vagy pedig látják a személyt címmel (vagy az anyával, stb.)

A NoSQL adatbázisok többsége nem tesz kísérletet az ilyen típusú végrehajtás biztosítására a megfelelő adatbázisban. Az Ön adataihoz szükséges kapcsolatok érvényesítése az adatbázis használatát végző kódban van rajtad múlik. A legtöbb esetben csak részben helyes adatokat láthatunk, így még akkor is, ha van olyan családfa, ahol minden embernek szülőkkel kell társulnia, lehet, hogy az Ön által megadott korlátozások valójában nem lesznek érvényesítve. Vannak, akik azt akarják, hogy akarják. Mások garantálják, hogy csak átmenetileg történik, bár pontosan mennyi ideig lehet / lehet tartani a kérdést.

14. Mit nevezünk csalás detektálásnak? Mi a csalás fogalma? Milyen csalás felderítési eszközöket ismer? Adjon példát egy csalás detektálásra.

Csalás

Valamilyen szoftveres rendszer kiskapuinak kihasználása kártékony célból. Pl. banki tranzakció, szolgáltatások ingyenes használata. A technika fejlődésével egyre több formája jelent meg, aminek a gyors szoftver fejlődés is kedvez. A biztonság vs. gyors fejlődés trade-offot kell vizsgálni.

Nagyon nagy számúak (12x) a hagyományos csalásokhoz képest. Tipikus csalási környezetek: mobil hálózatok, biztosítási követelések, adó visszakövetelések, bankkártya műveletek.

Csalás detektálás:

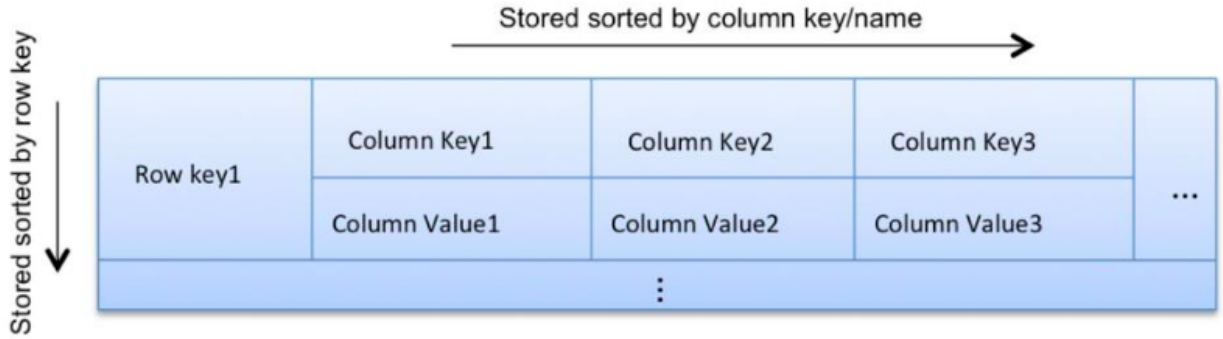
- Adat elemzéssel:
    - fel lehet tárni a csalásra utaló összefüggéseket
    - de komplex domain megismerést és összetett fejlesztést igényel
    - gyakran ismétlődő események - fellelhetők hasonló mintájú de más tartalmú előfordulásban is
    - rendelkezésre álló adatbázisokban
    - kiugró vagy csalásra jellemző értékek, mintázatok keresése
- statisztikai és gépi tanulási módszerek:
    - adatfeldolgozás, validáció, hibás/hiányzó adatok javítása
    - statisztikák számítása/karbantartása: átlagok, mennyiségek, teljesítmény metrikák, valószínűségi eloszlások
    - modellek és eloszlások számítása (klaszterezés, osztályozás):
        - a kapott adat összevetése korábbi adatokkal, idősor elemzés
        - predikció (hipotézis)
    - felhasználó profilok elemzése

Példák

Gépi bankkártya műveletek - ha nincsen randomizálva akkor egyenletes időnként történnek, ami miatt lehet látni hogy ez egy gépi felhasználó nem emberi. Például botok.

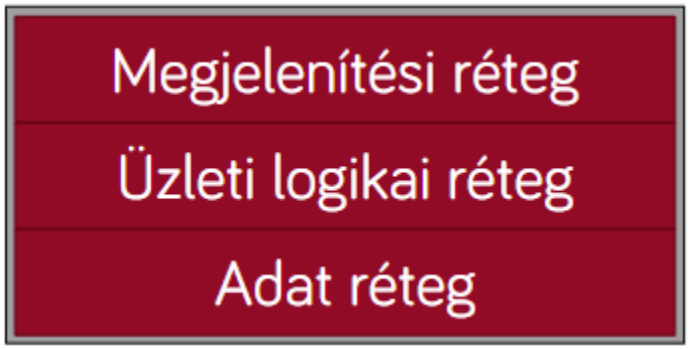
SPAM detektálás Cassandra Fraud Detection segítségével:

- Cassandra Fraud Detection
  - nagy adathalmazok hatékony kezelés (oszlop orientált NoSQL-től ez elvárható)
  - közel valós idejű adat elemzés
  - adat minták tárolása
- tárolási struktúra:
  - row key : a Social Media tartalom hash-e (szövegből képzett bináris érték)
  - Column Key: kommentelő egyedi azonosítója
  - Column Value: post azonosítója és időbélyege
  - ha n oszlop van egy sorban, akkor adott postot n-szer posztolták
  - idősor elemzés is végrehajtható (időbélyegek) és feltárhatók a post burst-ök, vagy intervallumok



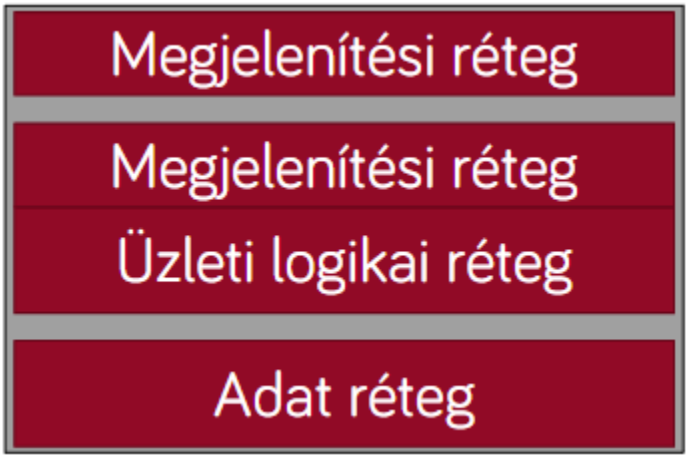
15. Milyen adat vizualizációs lehetőségeket ismer? Vesse össze a vastag, vékony kliens technikákat a reporting eszközökkel. Milyen reporting eszközöket ismer, ezeknek mi az előnye, hátránya?

Architektúrák:
Háromrétegű architektúra



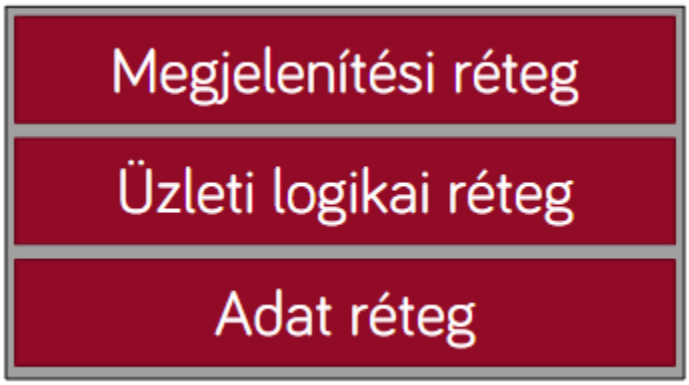
- szoros kapcsolat a rétegek között (gyakran nem is egyértelműen elválaszthatók) - közös kódbázis és technológiák
    - előnye: teljesítmény, capability (bármi megvalósítható ilyen formán)
    - hátránya: rugalmatlan, platform specifikus vagy túl általános

Web (szerver oldali renderelés):



- jobban elkülönülő rétegek:
    - adat réteg szinte mindig külön adatbázisban
    - megjelenítés webes technológiákkal történik a kliens böngészőjében
    - megjelenítés két része
        - HTML a szerver generálja, kitöltve a dinamikus részeket
        - kliens oldalon megjelenítés, illetve a logikák kezelése (pl. auto-complete) általában JavaScript
- sok kész keretrendszer és technológia

Web (kliens oldali renderelés):



- teljesen elkülönülő rétegek:
  - API-n keresztül kommunikálnak
  - technológiailag teljesen eltérhetnek
- megjelenítési rétegnek (kliens oldalon) sokkal több feladata lesz, logika kerül ide is
  - HTML itt kerül kitöltésre
  - validáció
  - megjelenítés - pagelés, térkép kezelés, grafikonok

Architektúra rétegei:

Adat réteg:
Általánosan:
- adat tárolás és hozzáférés a rendszer felől
- legyen egyszerű (lehetőleg ne SQL)
- biztosítson konzisztencia garanciákat, rendelkezésre állást stb.
- sok fajta adatforrás lehet: RDB, NoSQL, file

BI esetén:
- ‘vastag’ réteg: sok adatforrás és transzformáció, nagy mennyiségű adat
- kettéválik performancia szempontból:
  - ütemezett jobok esetén lehet lassabb
  - online felületek kiszolgálásánál gyorsnak kell lennie

Üzleti logikai réteg:
Általánosan:
- Itt van implementálva az üzleti logika (keretrendszerek használata esetén ezt kell megírni)
- lehet egy komponensben vagy elosztottan - kliens szerver minta

BI esetén:
- vékony kliens: a backend elkülönül a klienstől
  - biztosítja az API-t
  - szabályozza a hozzáférést

    - struktúrálja az adatokat: DTO
- vastag kliens: a kliens közvetlen DB csatlakozással rendelkezik
    - biztonsági rés (a DB kifelé elérhető)
    - nehéz frissíteni
- DTO:
    - adat egy nézete, nincsenek benne felesleges adatok
    - hozzáférés szabályozása
    - ld. ábra: fent objektum, lent a belőle létrehozott DTO

Megjelenítési réteg:
Általánosan:

- felhasználó számára érthető formában
- adat megjelenítés, manipuláció
- kommunikáció az üzleti logikai réteggel:
    - ‘programon belül’: függvény hívások
    - API-n: HTTP/REST, WebServices

BI esetén:

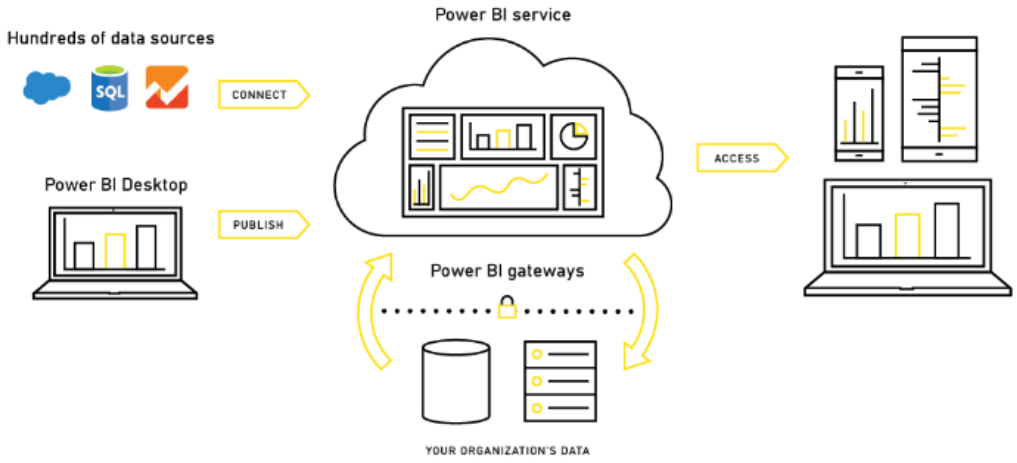
- elfedi a kommunikációt a backend API-val
- grafikus megjelenés
- a kommunikáció és megjelenés közötti adatkötés nagyon fontos - kevés feldolgozás legyen itt. Az API változtatása kell ha ez nem teljesül.
- webes felület esetén szükséges hogy a frontend is tartalmazzon logikát:
    - ha sokat kell várni egy kérésre (timeout)
    - HTTP 403 (Forbidden) vagy egyéb autentikációs probléma

Adat megjelenítési elvek:

- segíti hogy a felhasználó hatékony legyen:
    - az adat lényeges és felhasználható - rábírja a felhasználót hogy a lényegre fokuszáljon
    - elkerüli az adatok torzítását
    - különböző adatok összehasonlítását elősegíti - nagy adathalmazok megjelenítése
    - szaknyelv: aki ismeri annak nő a kifejező erő. Aki nem ismeri annak is szolgáltasson információt a vizualizáció.
- performancia:
    - adatkötés módja megfelelő (kis várakozási idő)
    - adatszinkronizáció és kliensoldali események összehangolása - pagination (adatokat nem egyszerre jelenítjük meg)
- átláthatóság - egyszerű, letisztult
- megfelelő adatvizualizációs elem kiválasztása:
    - diagram
    - grafikon
    - táblázat

Reporting eszközök:

- Pentaho Reporting:
  - thick client - standalone running
  - adat transzformálás információvá
  - felügyeleti és végfelhasználói felhasználás
  - kimeneti formátumok: PDF, Excel, HTML, Text, XML, CSV
  - böngésző alapú megjelenítés
  - pozitívumok:
    - big data integration
    - beágyazható más alkalmazásokba (SDK)
    - import data from any sources and different databases
    - elérhető macre is
  - negatívumok:
    - lack of solid user community
    - lack of good documentation
    - resource hungry
- PowerBI (vékony kliens):
  - felhő alapú kiszolgálás (SaaS): Microsoft Azure, REST API (thin client)
  - desktop, mobil és webes megjelenítés
  - machine learning eszközök
  - pozitívumok:
    - integration with Microsoft solutions
    - dynamic reports
    - quick filtering
  - negatívumok:
    - no MacOS version of Desktop app
    - Premium version is expensive



16. Mik a JavaFX technológia előnyei a korábbi Swing/AWT-hez képest? Mire szolgál a FXML? Vesse össze a JavaFX és a web alapú alkalmazásokat előnyök és hátrányok szempontjából.

JavaFX:

- gazdag felhasználói felület és RIA (Rich Internet Application) támogatás
- konzisztens működés több platformon
- Java alapú API grafikai és média támogatásra (pl. grafikonok támogatása)
- előnyei Swing/AWT-hez képest:
    - advanced look and feel
    - fast UI development with screen builder
    - új elemekkel bővítik
    - MVC (Model - View - Controller) támogatása

FXML:

- JavaFX UI leírására, XML alapú
- együtt működik Java kódokkal,
- támogatja mind a dinamikus mind a statikus felületeket
- előnyei:
    - kettéválasztja a UI leírást a funkcionalitástól, nem kell újrafordítani a kódot ha változik a FXML

Electron

- webes technológiákkal desktop app: HTML+CSS+JavaScript+browser
- rendszer funkciók API-n keresztül
- web és desktop előnyeinek ötvözése:
    - web: több platformon is egységes megjelenítés, közös nyelven
    - desktop: natív élmény, teljesítmény
- hátrányai:
    - lassú
    - nagy

Webes technológiák:

- reszponzivitás:
    - képernyőfüggetlen megjelenítés (képarány, felbontás, képátló független)
    - webes környezetbe különösen fontos a nagyon különböző készülékek miatt
    - képernyő dimenziók szerinti layout
- adaptív:
    - a kliens típusától függő felület impelementációk (desktop, tablet, mobil)
    - adott kliensre optimalizálva

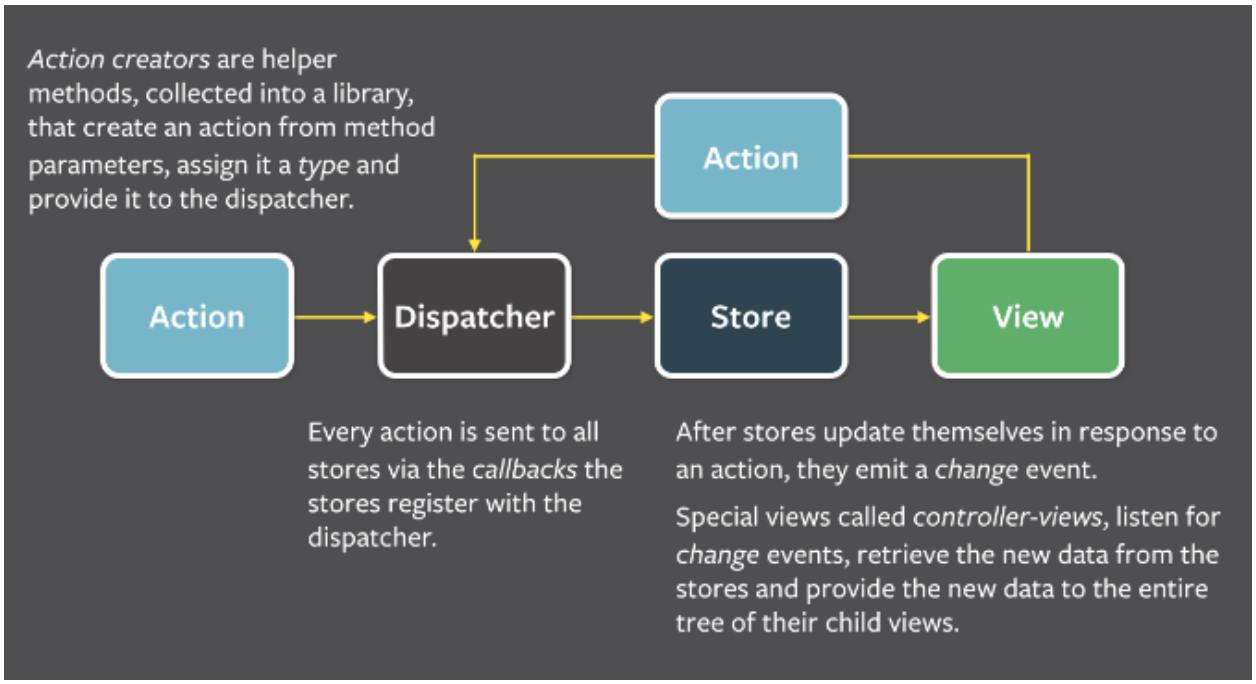
AngularJS:

- egyik legelterjedtebb JS alapú webes keretrendszer

- Single Page Application: A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages. The goal is faster transitions that make the website feel more like a native app.
- kétirányú adat binding a HTML elemekre rakott attribútumok segítségével
- open source és ingyenes
- reszponzív

React:

- JS Library felhasználó felületek készítéséhez
- VDOM: virtual DOM
    - DOM (Document Object Model): The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.
    - a virtuális DOM a DOM egy leképeződése, amivel lehet műveleteket végezni szabadon, hiszen csak a végeredmény lesz ténylegesen visszaírva a DOM-ba.
- JSX: JS szintaxis kiterjesztése, írhatunk HTML-t egyből kódba (jobban integrálódik a JS a többi webkomponenssel)
- Flux - Architektúrális minta



Polymer

- webes komponens készítő könyvtár
- dinamikusan fejlődik
- számos elem előre el van készítve

Angular és Polymer:

- két rendszer használható közösen is
- adatkötések használata:
    - dart package segítségével kétirányú adatkötés megoldható

JavaFX vs webtechnológiák:

- desktop környezet lassabban változik (stabilitás)
- gyorsabb, könnyebben hozzáfér az adatokhoz
- a webes alkalmazás mindennel is kompatibilis lesz - több platformon egységes megjelenítés

Ami kimaradt:

- Cloud hosting
- Docker
- Hadoop vége (Nifi stb.)
- 5. előadás második fele:
    - API leírók
    - Monigoring, Alerts
    - További adat/ esemény feldolgozó megoldások
- 6. előadás vége diagram könyvtárak.
- Elasticsearch