Nem-relációs adatbáziskezelés (No-SQL, BigData)

Gajdos Sándor

2024. máj. 6.

Motiváció

- Hogyan lehet adatokat minél nagyobb hatékonysággal kezelni?
- Mit jelent az „adatkezelés”?
- Meddig lehet a rendszer funkcionalitását egyszerűsíteni/csökkenteni?
- Mitől lesz ez adatbáziskezelés?

Történelmi előzmények

- Hierarchikus adatbáziskezelés (IBM, 1960-)
- Hálós adatbáziskezelés (~1970-85)
- Ami mindkettőben volt:
  - Imperatív lekérdezések
    - A programozó dolgozik, nem a számítógép
    - Deklaratív lekérdezések hiánya, nem kell lekérdezés optimalizálás
  - Kapcsolatok megvalósítása direkt linkekkel
    - V.ö: relációsnál adatok értékegyezése alapján – keresés
- Következmény: eleve lényegesen gyorsabbak lehetnek.

A legnagyobb weboldalak adatbázisai

Weboldal	Főbb adatbázismotor	Adatbázis típusa
Google	Google BigTable	wide-column-based NoSQL
Facebook	Cassandra, Hadoop/HIVE	wide-column-based NoSQL
Youtube	Memcached	Key-Value
Microsoft Live, Bing	Azure	RDBMS
Yahoo!	Hadoop, PNUTS	wide-column-based NoSQL
Twitter	FlockDB, Cassandra, Hadoop/Hbase	Graph, wide-column-based NoSQL
Wikipedia	Memcached, Flatfile, MySQL	Key-Value, Flat file, RDBMS
BBC	CouchDB	Document-oriented
Tiktok	PostgreSQL, Cassandra	RDBMS, wide-column-based NoSQL
Amazon	Amazon DynamoDB	wide-column-based NoSQL and document-oriented

Mi van a relációson túl?

- Hierarchikus DB
- Multidimenziós DB
- Document store
- Gráf DB
- Key/value store
- Object DB
- ...

Összefoglaló nevük: NoSQL adatbáziskezelők

NoSQL

- Relációs rendszerek gyengéi
  - Sok dokumentum indexelése
  - Nagyforgalmú weboldalak kiszolgálása
  - Adatstream-ek szolgáltatása
- Általános jellemzők
  - Gyenge konzisztenciagaranciák
  - Elosztott architektúra

Ez mind NoSQL…

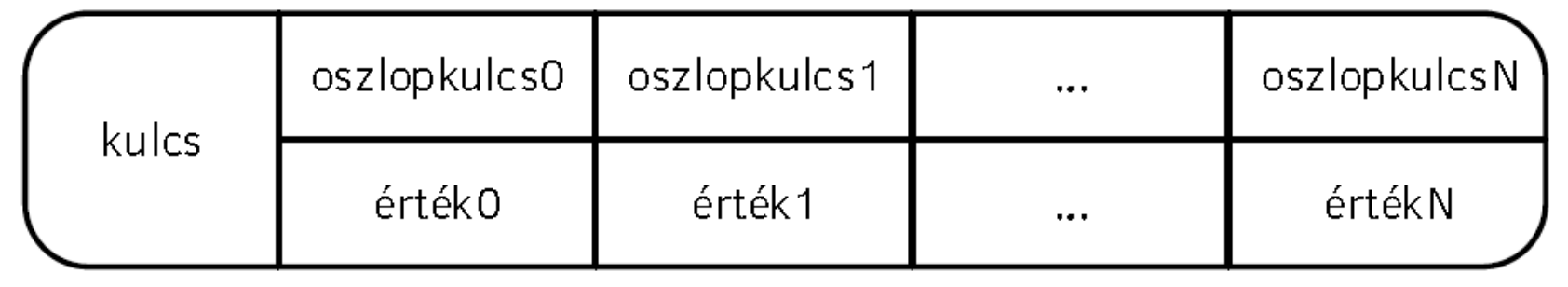
- „Klasszikus”:
  - Kulcs-érték tárak (key-value store)
  - Oszlopcsaládok (wide-column-based)
  - Dokumentumtárak
  - Gráf adatbázisok
- Tágabb értelemben:
  - Hierarchikus adatbázisok
  - (Natív) multidimenziós adatbázisok

Key/value adatbázisok

- A legegyszerűbb NoSQL DB
- Kulcs+adat (akár blob, a tartalma közömbös)
- Az alkalmazás felelőssége az adat értelmezése
- Nincs hagyományos séma
- Hozzáférés gyakran csak a kulcsértéken kereszül, igen jól optimalizálható
- Természetesen az adat is indexálható
- A tárolás lehet diszken, memóriában, rendezve vagy anélkül...
- Berkeley DB, Memcachedb, Redis, BigTable, SimpleDB, Tokyo Cabinet, Tuple space, NMDB,...

Oszlopcsaládok (wide-column-based)

- Nem összetévesztendő az oszlopalapú (relációs!) adatbáziskezelőkkel (SAP Hana, SybaseIQ, Vertica,…)
- Egyfajta kombinációja a kulcs-érték táraknak és az oszloporientált tárolásnak
- Amazon DynamoDB, Google BigTable, Apache Cassandra, …

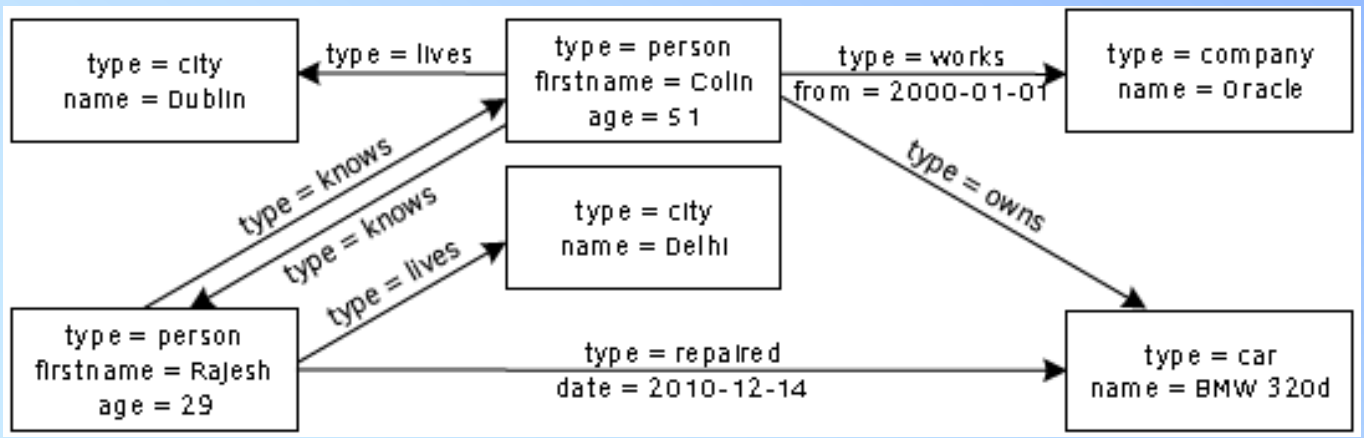


Dokumentumtárak

- Dokumentum itt: szemistrukturált adatok, tip. JSON v. XML formában
- Tkp. OODB
- Minden rekord egy dokumentum, aminek tetszőleges számú, nevű és méretű mezője lehet (attribútum név-érték párok)
- Nincs üres mező, a mezők többszörös adatelemeket is tartalmazhatnak
- Egyszerű használat, könnyű programozhatóság
- Jellegzetes: tartalomkezelő, ajánlórendszerekben
- CouchDB, MongoDB, RavenDB,…

Gráf adatbázisok

- Az adattárolást gráfstruktúrák (csomópontok, élek + attribútumok (kulcs+érték formában)) valósítják meg
- A gráf lehet tetszőleges vagy spec. (ld. hálós adatbázisok)
- OO alkalmazások esetén (is) hatékonyabb
- Gyakran változó séma esetén
- Neo4j, AllegroGraph, Core Data, FlockDB



Hierarchikus adatbázisok

- Az adatok faszerű struktúrákban
- Rekordorientált szemlélet
- 1:N kapcsolatok natív leképezése
- Imperatív lekérdezhetőség
- Windows registry (Microsoft) IMS (IBM),

Multidimenziós adatbázisok

- N-dimenziós tömbök
- Cellák: tényadatok, melyeket a dimenziók koordinátáival címezhetünk
- Segédstruktúrák (indexek)
- Nem tévesztendő össze a relációs alapú dimenzióssal
- Microsoft Analysis Services, Oracle (Hyperion) Essbase, SAP BW,...

Mit lehet feladni a sebességért?

- Elvileg mindent, ami egy klasszikus ACID DBMS-t jellemez: atomicitás, konzisztencia, izoláció, tartósság
  - Humán analógia ☺
- A relációs modell előnyeit is:
  - Kényelem
  - Változatos lekérdezések hatékonyan

Mindegyik jelentősen megkönnyíti (hiánya pedig megnehezíti…) az alkalmazás-fejlesztők munkáját is.

+ horizontális skálázás („scale out”): több számítógép (tip. shared nothing) bevonása -> “elosztottság”

A “sörfőző sapka-elmélete”

- Elosztott rendszerek alapvető problémája: hogyan viselkedik a teljes rendszer, ha egy része („partíció”) kiesik
- Eric Brewer’s CAP theory (2000):
  - **Consistency (itt!):** olyan a működés, mintha a műveletek egyetlen csomóponton egy pillanat alatt futnának le („atomi” konzisztencia). Bármely csomópontból lekérdezve az elosztott rendszert ugyanazt az eredményt kapjuk.
  - **Availability (rendelkezésreállás):** minden művelet a tervezett eredménnyel, véges idő alatt fejeződik be (nem pl. hibaüzenettel).
  - **Partition tolerance:** Semmilyen részleges hiba nem okozhatja, hogy a rendszer helytelen választ ad

közül egyszerre csak kettő teljesülhet elosztott környezetben.

Az elméletből formális bizonyítás után tétel lett (2002)

- Következmény: horizontálisan skálázott elosztott rendszerek ha toleránsak a partíciók hibáira, akkor vagy konzisztensen működnek (CP), vagy a rendelkezésreállásuk biztosított (AP).

Értelmezés

- **Példa1:** Ha sem a C-t, sem az A-t nem akarjuk feladni (CA): minimalizáljuk a partícióhiba valószínűségét (pl. megbízható LAN kapcsolat, tipikus)
- **Példa2:**
  - Két-node-os adatbázis klaszter, 2PC, mindkét node szükséges egy tranzakcióhoz
  - Konzisztencia OK, partition tolerance OK (CP), de hogyan érinti a rendelkezésreállást?
  - Rendelkezésreállás:
    - A szükséges komponensek rendelkezésreállásának szorzata
    - A használható, de nem használt komponensek nem csökkentik a rendelkezésreállást
  - Ha mindkét node-nak 99% a rendelkezésreállása, akkor a tranzakcióé már csak 98%. (10 node esetén már csak 90%)

Igen nagy tranzakciós terhelések

- Új gondolkodásmód kell az erőforrások kezeléséről
- ACID problémás, ha a terhelést/feladatot több node között osztjuk meg (ld. DDBMS algoritmusok)
- Műveletek szétcsatolása javítja a rendelkezésreállást és skálázhatóságot a konzisztencia rovására

Egy alternatíva: BASE konzisztencia

- Klasszikus konzisztencia helyett lazítás
- BASE=Basically Available, Soft state, Eventually consistent (2008)
- ACID: pessszimista, BASE: optimista, elfogadja a DB „fokozatos” konzisztenciáját a tranzakció végén
- Következmény:
  - Kezelhető
  - Magas szintű horizontális skálázhatóság
  - Rendelkezésreállás: részleges hibák megtűrése

Példa:


```
Begin transaction
  Insert into transaction(xid, seller_id, buyer_id, amount);
  Update user set amt_sold=amt_sold+$amount where id=$seller_id;
  Update user set amt_bought=amount_bought+$amount where id=$buyer_id;
End transaction
```

```
Begin transaction
  Insert into transaction(id, seller_id, buyer_id, amount);
End transaction
Begin transaction
  Update user set amt_sold=amt_sold+$amount where id=$seller_id;
  Update user set amt_bought=amount_bought+$amount
        where id=$buyer_id;
End transaction
```

- Sajátos séma tranzakciókhoz
- ACID-stílusú megoldás
- Konzisztencia lazítása

Irodalom:

- Pritchett: BASE: An Acid Alternative, ACM Queue, May/June 2008.
- Brewer’s keynote speech: http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- Brewer, E., "CAP twelve years later: How the "rules" have changed", Computer , vol.45, no.2, Feb. 2012, pp.23-29.
- Gajdos: Adatbázisok, C. függelék: NoSQL adatbázis-kezelők, 2022.