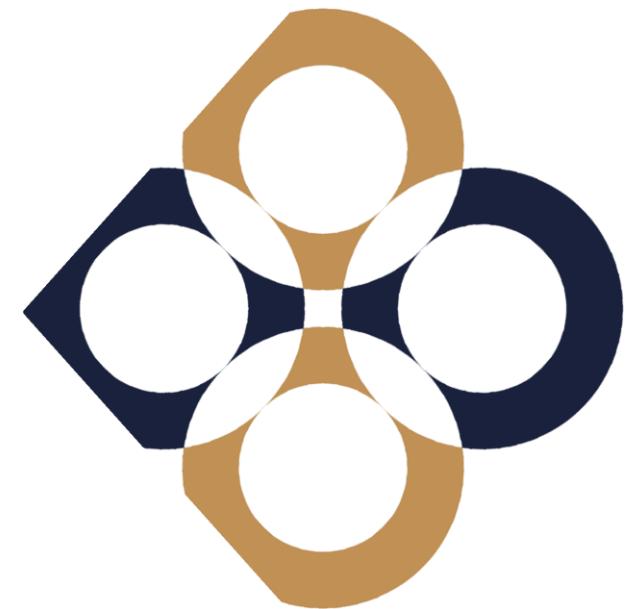


Adatbázisok előadás 09

Gráf adatbázisok



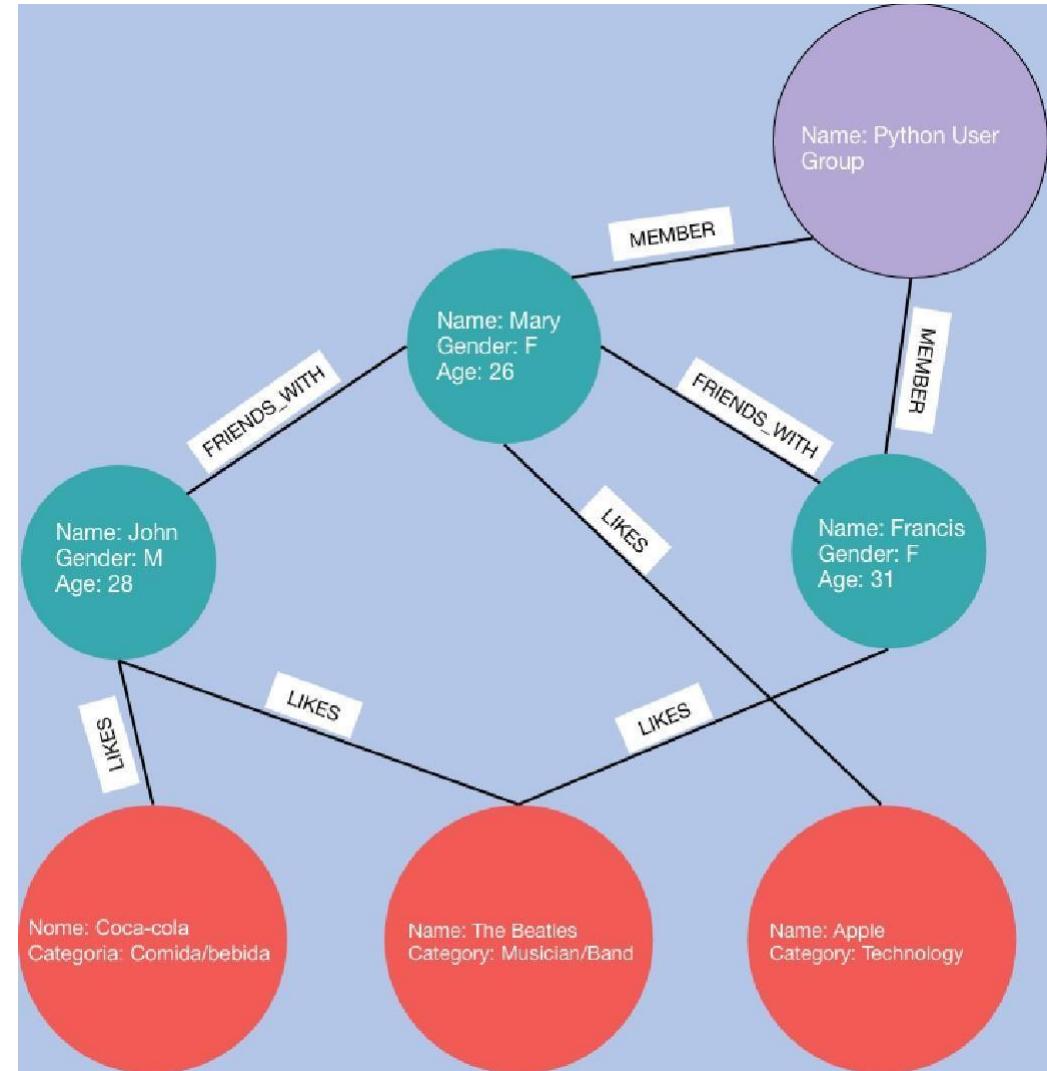
Miről lesz szó?

- Gráf adatbázisok jellemzői
- Neo4J adatbázis
 - Jellemzők
 - Logikai és fizikai tárolás
 - Adatbázis objektumok, tranzakciók
 - Kényszerek
 - Jogosultságok
 - Indexek
 - Elérés Python-ból

Gráf adatbázisok

Olyan adatbázisok, amelyek az adatok tárolására és megjelenítésére gráf struktúrát alkalmaznak

- A gráf csúcsaiban vannak az adatok
 - Az adatok sémája nem rögzített
- A gráf élei jelentik a kapcsolatokat
 - Az élek irányítottak
 - Az éleknek adott név a kapcsolatra jellemző



Gráf adatbázisok – előnyök és hátrányok

Előnyök

- Flexibilis séma
- Logikus, jól érthető lekérdezések
- Gyors adatelérés
- Nagymértékben összefüggő adatok kezelése
- Sokféle feladathoz megfelelők

Hátrányok

- OLTP rendszerekhez nem a legjobbak
- A nagy adatmennyiséget érintő lekérdezések nem optimálisak
- Sok esetben egy szerveren tárolódnak

Gráf adatbázisok – mire használják őket?

Tudásbázisok

Csalás felderítés

Termék ajánló rendszer

Social media

Törzsadatok kezelése

Hálózati infrastruktúra monitorozás

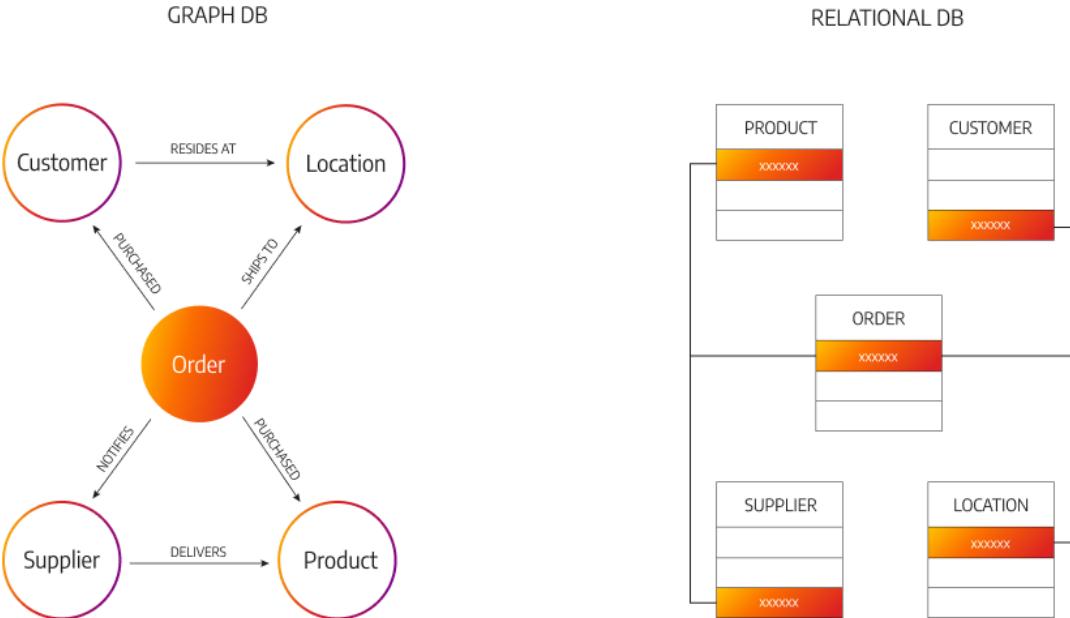
Gráf adatbázisok vs. Relációs adatbázisok

Gráf adatbázisok	Relációs adatbázisok
Egy csúcs vagy egy kapcsolat	Sor
Tulajdonsgáok	Oszlopok
Csúcsok halmaza	Tábla
Gráf (csúcsok +élek + tulajdonságok)	Adatbázis
Nincs séma	Fix séma
A kapcsolatok direkt módon definiáltak	A kapcsolatok idegen kulcsokkal valósulnak meg
A kapcsolódó adatok megjelenítése minták segítségével	A kapcsolódó adatok megjelenítése JOIN-okkal

Gráf adatbázis vs. Relációs adatbázis példa

Példa csúcsokra:

```
(:Customer {name: "Alice", customerId: 1})
(:Location {city: "Budapest", locationId: 101})
(:Order {orderId: 5001, date: "2025-04-28"})
(:Supplier {name: "SupplierCo", supplierId: 200})
(:Product {name: "Laptop", productId: 3001})
```



Példa élekre:

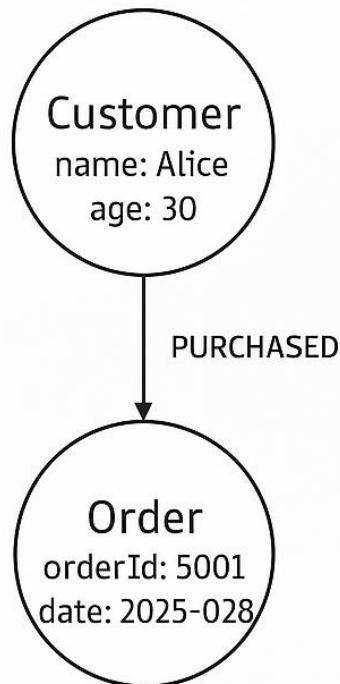
```
(:Customer {name: "Alice", customerId: 1})-[:RESIDES_AT]->(:Location {city: "Budapest", locationId: 101})
(:Customer {name: "Alice", customerId: 1})-[:PURCHASED]->(:Order {orderId: 5001, date: "2025-04-28"})
(:Order {orderId: 5001, date: "2025-04-28"})-[:SHIPS_TO]->(:Location {city: "Budapest", locationId: 101})
(:Order {orderId: 5001, date: "2025-04-28"})-[:PURCHASED]->(:Product {name: "Laptop", productId: 3001})
(:Supplier {name: "SupplierCo", supplierId: 200})-[:NOTIFIES]->(:Order {orderId: 5001, date: "2025-04-28"})
(:Supplier {name: "SupplierCo", supplierId: 200})-[:DELIVERS]->(:Product {name: "Laptop", productId: 3001})
```

Gráf adatbázisok vs. Dokumentum adatbázisok

Gráf adatbázisok	Dokumentum adatbázisok
Csúcs	Dokumentum
Címkék	Gyűjtemény
Kapcsolatok a modellben	Kapcsolatok beágyazással vagy „idegen” kulcsokkal
A kapcsolódó adatok megjelenítése minták segítségével	A kapcsolódó adatok megjelenítése beágyazással vagy join-okkal

Gráf adatbázis vs. Dokumentum adatbázis példa

GRAPH DB



DOCUMENT DB

```
{  
  "name": "Alice",  
  "age": 30,  
  "city": "Budapest"  
}  
"orders": [  
  {  
    "orderId": 5001,  
    "date": "2025-04-28"  
  }  
]
```

Gráf adatbázisok - Példák



Neo4j – kik használják?

Helping solve the world's most challenging problems
across every industry



Financial Services



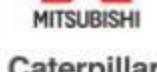
Telco



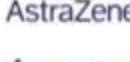
Retail



Manufacturing



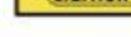
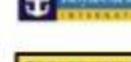
Life Sciences



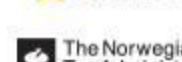
Technology



Logistics



Government



Gráf adatbázisok – Neo4j

- A legismertebb gráf adatbázis, Java-alapú
- A csúcsok ~ entitások, objektumok
 - Lehetnek tulajdonságaik (kulcs-érték párok)
 - Lehetnek címkéik
- A kapcsolatok
 - Elnevezhetők
 - Lehetnek tulajdonságaik
 - Indexelhetők
- Flexibilis séma
- Magas rendelkezésre állás (elosztott rendszerben)
- Beépített gráf algoritmusok



The #1 Database for Connected Data

Neo4j Community vs Enterprise

	COMMUNITY	ENTERPRISE
Cluster support	✗	✓
Automatic replication	✗	✓
Role-based access control	✗	✓
Fine-grained security	✗	✓
High availability (HA)	✗	✓
Online backup	✗	✓
Advanced monitoring	✗	✓
Neo4j Fabric	✗	✓
Hot backups	✗	✓
APOC	✓	✓

Egyedeket (entitások) reprezentálnak a gráfban

- A tulajdonságok kulcs-érték párok
- Címkékkel kategorizálhatók
- Önmagukban is létezhetnek, de kapcsolatokban is részt vehetnek
- Egyedi azonosítójuk van, automatikusan generálódik (ID)
- Pl: egy személy adatait tartalmazó csúcs
(`:Személy {Név: "Kiss Béla", Életkor: 25}`)
- Pl: egy csúcs több címkével:
(`:Személy :Alkalmazott {Név: "Kiss Béla", Életkor: 25}`)

Kapcsolatokat reprezentálnak a gráfban

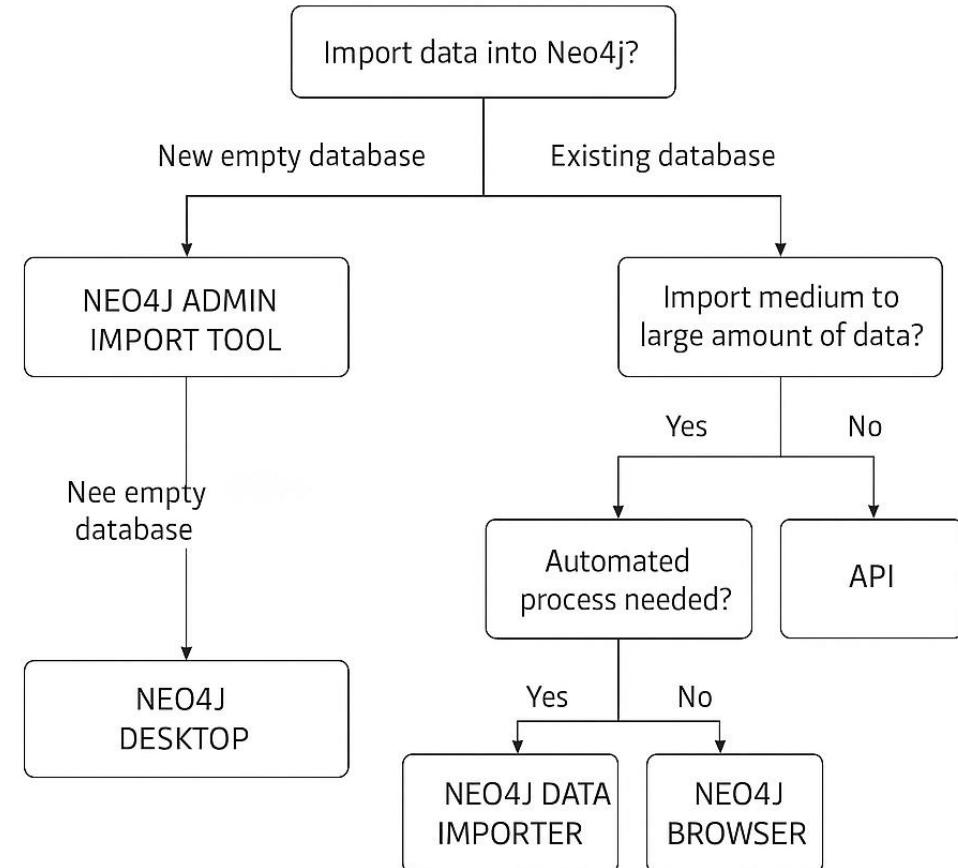
- Mindig irányítottak
- Pontosan egy címkéjük van, pl: :PURCHASED
- Lehetnek tulajdonságaik is,
pl: [:PURCHASED {date: "2025-04-28", amount: 150}]
- Mindig két csúcsot kötnek össze
- Indexelhetők
- Van egyedi azonosítójuk (ID)
- Pl: egy él tulajdonságokkal

(:Customer {name: "Alice"})-[:PURCHASED {date: "2025-04-28", amount: 150}]->(:Order {orderId: 5001})

Neo4j – Adatok bevitelé

- Cypher CRUD parancsokkal (ld. Gyakorlat)
- Cypher LOAD CSV parancssal
- Importálással (Neo4j Admin Import Tool)
- ETL eszközzel (Neo4j ETL Tool)
- API segítségével
- Neo4j Data Importer (webes eszköz)

DATA IMPORT INTO NEO4J



Neo4j – Normalizálás kérdése

A relációs adatmodellhez hasonló normalizálás itt nem szükséges

- A redundancia általában nem probléma
- A kapcsolatok az elsődlegesek
- Optimalizálni inkább a lekérdezési mintákat kell
- Anomáliák rendszerint helytelen modellezés vagy CRUD műveletek következményei

Neo4j – Példa anomáliákra

- Új rendelést szeretnénk rögzíteni, de még nem létezik a hozzá tartozó ügyfél (bővítés anomália)

pl: CREATE (o:Order {orderId: 5001, date: "2025-04-28"})
CREATE (c)-[:PURCHASED]->(o)

- Az ügyfél neve több helyen is szerepel (módosítási anomália)

pl: CREATE (c:Customer {name: "Alice"})
CREATE (o:Order {orderId: 5001, customerName: "Alice"})

- Törlünk egy olyan ügyfelet, akinek vannak rendelései (törölési anomália)

pl: MATCH (c:Customer {name: "Alice"})
DELETE c

Neo4j vs Adattárház/BI

A Neo4j nem alkalmas adattárház platformnak

- Kiegészítő szerepe lehet viszont a BI esetén
- A Neo4j kapcsolódhat Tableau, Power BI stb rendszerekhez (BI Connector)
- Van kiegészítő könyvtár (APOC) az aggregációkhoz
- Alternatíva lehet pl. sok join-t igénylő lekérdezések esetén

A Neo4j tárlómotorja a Neo4j Native Graph Store

- Saját bináris formátum neostore
- Fix méretű rekordok, mutatók neostore.id
- Írás előtti naplázás (WAL – Write Ahead Logging) neostore.nodestore.db
neostore.relationshipstore.db
neostore.propertystore.db
neostore.propertystore.db.strings
neostore.propertystore.db.arrays
neostore.labeltokenstore.db
neostore.relationshiptypestore.db
transaction logs (neostore.transaction.db.*)
- Page Cache

A Neo4j Sharding rendszere

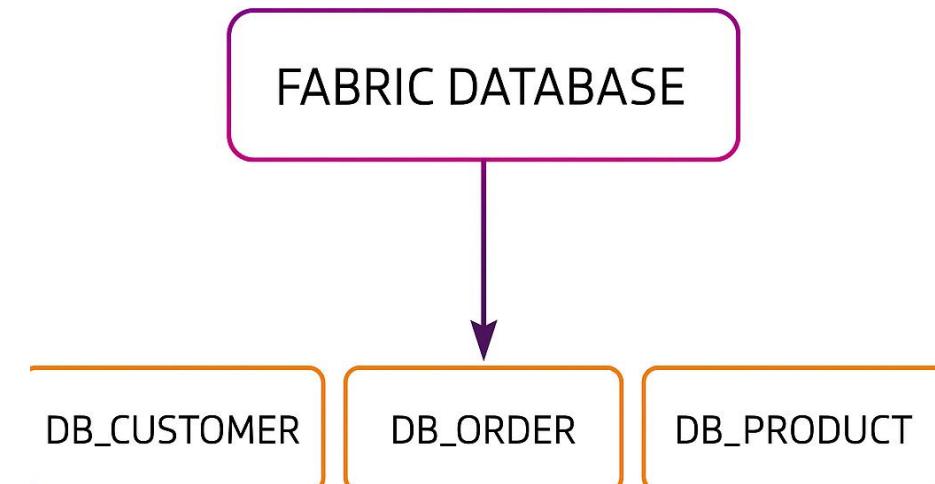
A Neo4j-ben nincsen automatikus, fizikai sharding

- Csak logikai sharding létezik
- Eszköz: Neo4j Fabric
 - Elosztott lekérdezési réteg
 - Képes több adatbázis egyidejű kezelésére
 - Az adat helyét lekérdezéskor kell meghatározni
 - Nem tárol adatot, csak továbbít
 - minden shard egy Neo4j adatbázis

A Neo4j Sharding - Példa

A lekérdezésekben kell megadni, hogy melyik shard-ban van az adat

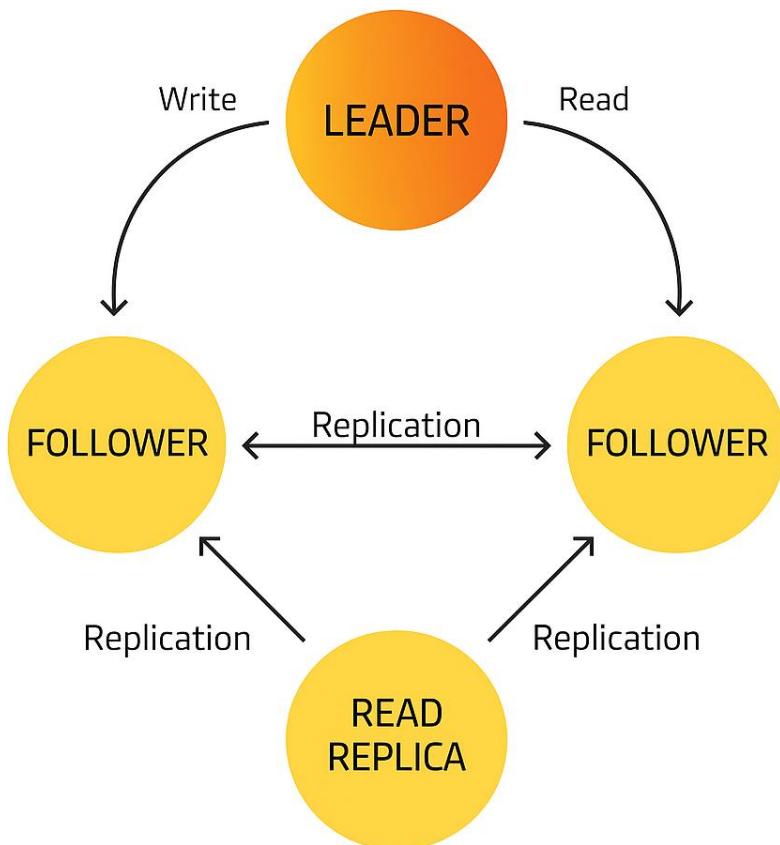
```
USE fabric.db
CALL {
    USE db_customer
    MATCH (c:Customer {name: "Alice"}) RETURN c
}
CALL {
    USE db_order
    MATCH (o:Order {customerId: c.id}) RETURN o
}
RETURN c, o
```



A Neo4j replikációs rendszere

A replikáció csak az Enterprise változatban érhető el.

- Technológia: Causal Cluster
- Automatikus működés
- RAFT algoritmus (konszenzus alapú)
- Szerepkörök:
Leader, Follower, Read Replica



A RAFT algoritmus

RAFT = Reliable, Replicated, Redundant, Fault-Tolerant

Lépés

1. Leader választás

Mi történik?

A cluster tagjai szavazással választanak egy Leader-t.

2. Napló (Log) szinkronizáció

A Leader minden tranzakciót először saját naplójába ír, majd elküldi a Follower-eknek.

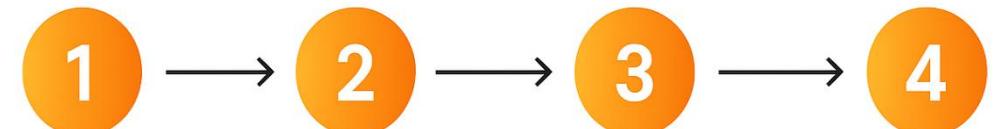
3. Konszenzus

A tranzakció akkor tekinthető véglegesnek (committed), ha a többség (quorum) jóváhagyja.

4. Hibatűrés

Ha a Leader meghibásodik, a Follower-ek közül automatikusan újat választanak.

RAFT
CONSISTENT ALGORITHM



LEADER
ELECTION → LOG
REPLICATION → LOG
COMMIT

CONSENSUS

Follower vs Read Replica

Tulajdonság

Részt vesz a RAFT szavazásban?

Tud olvasni?

Tud írni?

Adatkonzisztencia követése

Feladata

Failover esetén lehet Leader?

Tipikus helye

Jellemző használat

Follower

- Igen (részt vesz a konszenzusban)
- Igen
- Nem, csak a Leader tud írni
- Teljesen szinkronban kell lennie

Biztonsági mentés, failover tartalék

- Igen (Followerből lehet új Leader)

Clusteren belül

Magas rendelkezésre állás biztosítása

Read Replica

- Nem (csak szinkronizál, nem szavaz)

- Igen
- Nem

Szinkronizál, de **késhet**

Csak olvasási terhelés kiszolgálása

- Nem (Read Replica nem lehet Leader)

Lokálisan vagy földrajzilag máshol is

Olvasási skálázás (nagy forgalom, geo-elosztás)

Neo4j - Konzisztencia

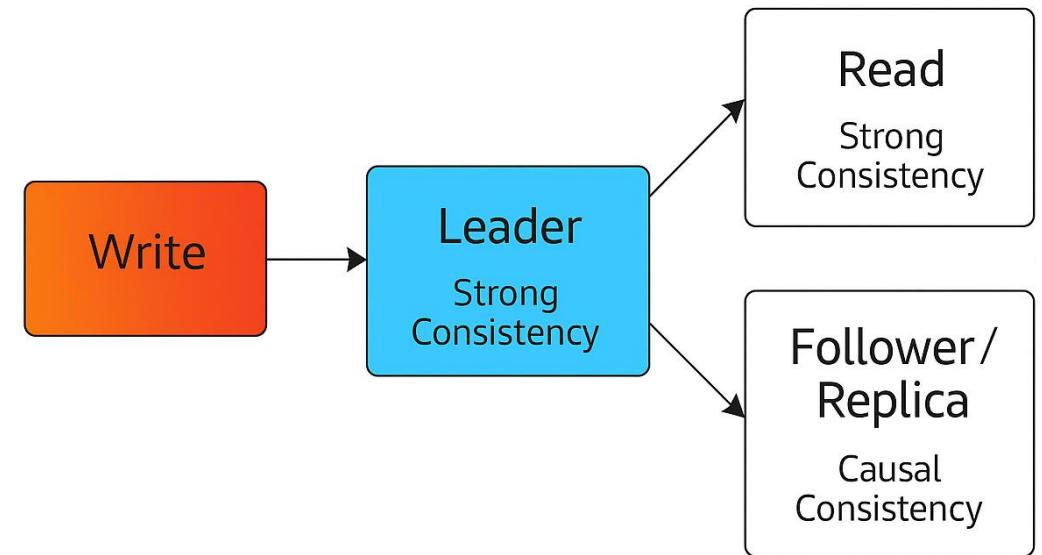
Önálló példány esetén

- Szigorú konzisztencia (ACID)

Elosztott rendszerben

- Írásnál szigorú konzisztencia
- Olvasásánál szerepkörtől függően lehet szigorú vagy kazuális (ok-okozati)

Neo4j Consistency



Neo4J- Nézetek

Hagyományos értelemben vett nézetek nincsenek.

Lekérdezési minták

- Ideiglenes nézatként viselkednek
- A WITH utasítással generálhatók
- Az APOC library-vel újrafelhasználhatók
- A Fabric segítségével kompozit nézetek is létrehozhatók

Példa:

```
MATCH (p:Person)
WITH p.name AS personName,
     size((p)--()) AS degree
WHERE degree > 5
RETURN personName, degree
```

Neo4J – tárolt eljárások és függvények

A Neo4j támogatja a tárolt eljárásokat és függvényeket.

Tárolt eljárások

Tulajdonság	Leírás
Fogalom	Előre definiált műveletcsomagok, amelyeket Cypher-ből hívhat sz meg (CALL)
Alkalmazás	Bonyolult logika végrehajtására, amit Cypher-rel nehéz lenne megoldani
Létrehozás	Java-ban (vagy kompatibilis nyelven) fejlesztve, majd pluginként telepítve
Példa	CALL db.labels() (beépített eljárás, listázza a címkeket)

Neo4J – tárolt eljárások és függvények

A Neo4j támogatja a tárolt eljárásokat és függvényeket.

Függvények

Tulajdonság	Leírás
Fogalom	Egy kifejezés, amely egyetlen értéket ad vissza
Alkalmazás	Értékek transzformálására, számításokra lekérdezés közben
Létrehozás	Szintén Java-ban írva, telepítve pluginként
Példa:	<code>RETURN toUpper('hello') → visszaadja: 'HELLO'</code>

Neo4j tárolt eljárások és függvények gyűjteménye

Példák

Kategória	Parancs	Rövid leírás
Gráf metaadatok	CALL apoc.meta.graph()	Az aktuális gráf szerkezetének felfedezése (címkék, kapcsolatok)
Random adatok	RETURN apoc.text.random(10)	Véletlen szöveg generálása
Adatkonvertálás	RETURN apoc.convert.toJson({key: 'value'})	Objektum JSON formátumba konvertálása
Szövegfeldogozás	RETURN apoc.text.join(['a','b','c'], '-')	Tömb elemeinek összefűzése kötőjellel
Útvonalkeresés	CALL apoc.algo.dijkstra(startNode, endNode, 'KNOWS', 'weight')	Dijkstra algoritmussal legrövidebb útvonal keresése
Exportálás	CALL apoc.export.csv.all("all.csv", {})	Teljes gráf exportálása CSV fájlba
Adatbetöltés	CALL apoc.load.json('file:///data.json')	JSON fájl beolvasása és betöltése gráfba
Időkezelés	RETURN apoc.date.format(timestamp(), 'ms', 'yyyy-MM-dd')	Időbélyeg formázása
Gráf módosítás	CALL apoc.refactor.mergeNodes([n1, n2])	Két csúcs összeolvasztása egyetlen csúccsal

Neo4j – triggerek

A Neo4j-ben nincsenek klasszikus triggerek

Eseménykezelés

- Csak Enterprise változatban
- APOC triggerekkel
- A memóriában tárolódnak
- Hibánál nincs automatikus Rollback

Példa:

```
CALL apoc.trigger.add(  
  'addCreatedAt',  
  "UNWIND $createdNodes AS n  
  WHERE n:Person  
  SET n.createdAt = timestamp()",  
  {phase:'after'}  
)
```

Tranzakciók a Neo4j-ben

A Neo4j biztosítja az ACID tranzakciókezelést

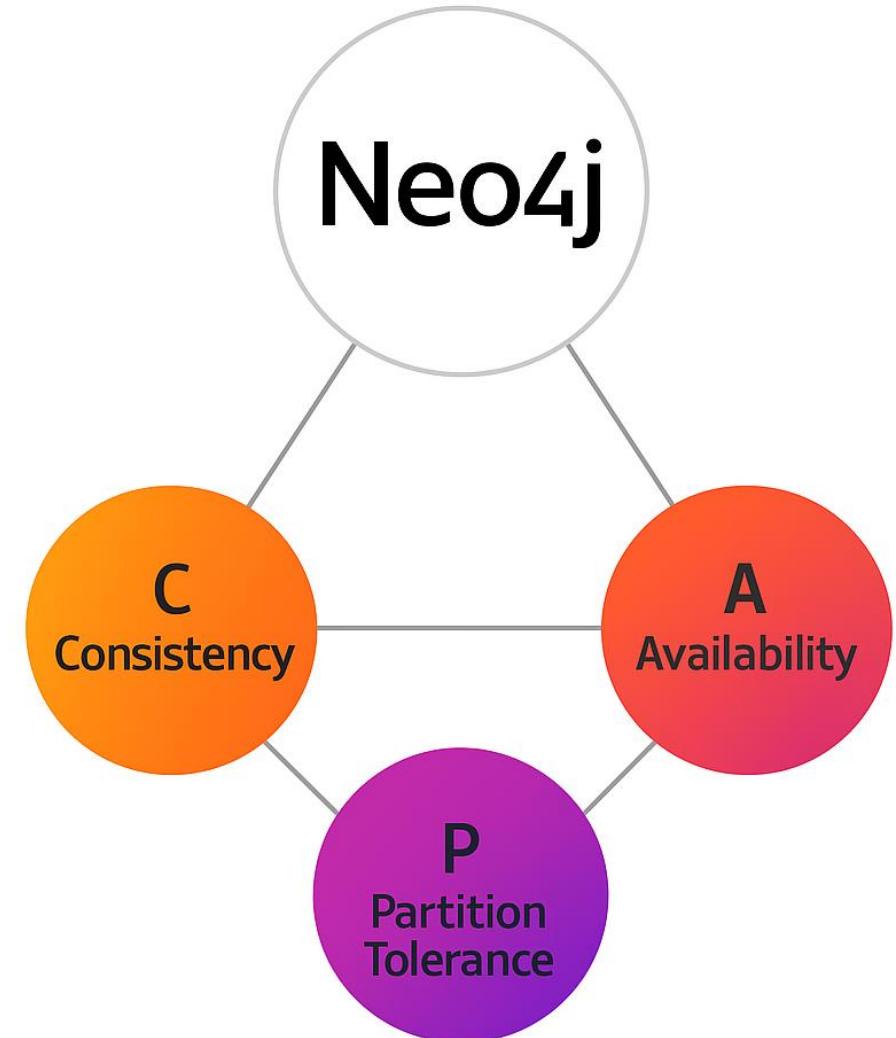
Jellemzők

- minden Cypher parancs tranzakción belül fut
- zárolások az ütközések elkerülésére
- csak snapshot jellegű izoláció, phantom read lehetséges
- Holtpont esetén automatikus feloldás
- A tranzakciók naplózva vannak (WAL)
- Cluster estén ACID + Casual consistency

Neo4j vs CAP Theorem

A Neo4j a CP rendszerek közé tartozik

- Consistency + Partition Tolerance biztosított
- Availability esetén néha kompromisszum pl. Leader újraválasztásnál
- Adat nem veszik el



Lépések

- ❑ Indítás: A kliens indít egy tranzakciót
- ❑ Lockolás: A rendszer zárakat helyez el az érintett csúcsokra, élekre.
- ❑ Módosítás: A Cypher parancsok végrehajtják a módosításokat a memóriában.
- ❑ Naplázás: A módosítások bekerülnek az írási naplóba (Write-Ahead Log, WAL).
- ❑ Commit vagy Rollback

Példa

```
BEGIN
MATCH (a:Person {name: "Alice"})
SET a.age = 31
COMMIT
```

NEO4J INDEX TYPES

B-TREE INDEX

For property value lookup

FULLTEXT INDEX

For text search

POINT INDEX

For spatial lookup

RANGE INDEX

For range query

COMPOSITE INDEX

For index on multiple properties

CONSTRAINT INDEX

For uniqueness or existence constraint

Neo4j – Indexek létrehozása

CREATE INDEX – index létrehozása

SHOW INDEXES [VERBOSE] – indexek listázása

DROP INDEX – index törlése

- A VERBOSE segítségével opcionálisan részletesebb lista jeleníthető meg
- A PROFILE utasítással megjeleníthető a végrehajtási terv
- Az EXPLAIN utasítás hasonlóan működik, de magát az utasítást nem hajtja végre, csak a végrehajtási tervet jeleníti meg

Neo4j – Indexek - példák

```
profile  
match (p:Person)  
return p
```

A lekérdezés és végrehajtási terv

```
CREATE INDEX i_name  
IF NOT EXISTS FOR (n:Person)  
ON (n.name)
```

A személyeket indexeli név
alapján, ha még nincs index

```
CREATE INDEX i_filmek  
FOR (m:Movie)  
ON (m.title, m.released)
```

Összetett index létrehozása

```
DROP INDEX i_filmek
```

Törli az adott indexet

Kényszerek a Neo4j-ben

A Neo4j többféle kényszert is támogat az adatintegráció megőrzéséhez.

Kényszertípus

UNIQUE Constraint

EXISTS Constraint

NODE KEY Constraint

RELATIONSHIP PROPERTY EXISTENCE Constraint

Mire szolgál?

Egyedi értéket biztosít egy property-re

Kötelezővé tesz egy property meglétét csúcsokon vagy éleken

Több property együttes egyediségét biztosítja csúcsokra

Kötelezővé tesz egy property meglétét éleken

Cypher példa

```
CREATE CONSTRAINT FOR (p:Person)  
REQUIRE p.email IS UNIQUE
```

```
CREATE CONSTRAINT FOR (p:Person)  
REQUIRE p.name IS NOT NULL
```

```
CREATE CONSTRAINT FOR (p:Person)  
REQUIRE (p.firstName, p.lastName) IS  
NODE KEY
```

```
CREATE CONSTRAINT FOR ()-  
[r:FRIENDS_WITH]-()  
REQUIRE r.since  
IS NOT NULL
```

Jogosultságok a Neo4j-ben

Szerepkör-alapú jogosultságkezelés (RBAC – Role-Based Access Control)

Jogosultsági szintek

- Adatbázis
- Címke
- Tulajdonság
- Kapcsolat

Példa

```
CREATE USER alice SET PASSWORD 'secret'
```

```
GRANT ROLE reader TO alice
```

```
GRANT READ {Person} ON GRAPH neo4j TO reader
```

Szerepkörök

- Admin
- Reader
- Editor
- Publisher
- Architect

Fontos:

- Csak az kap jogot, akinek adunk
- A Community verzióban a jogkezelés korlátozott

Neo4j Elérés Python-ból

```
!pip install neo4j
from neo4j import GraphDatabase

# Csatlakozás adatai
uri = "bolt://localhost:7687" # vagy "neo4j+s://<url>" ha SSL kell
username = "neo4j"
password = "your_password"
# Driver példány létrehozása
driver = GraphDatabase.driver(uri, auth=(username, password))
# Példa lekérdezés futtatására
def print_person_names(tx):
    result = tx.run("MATCH (p:Person) RETURN p.name AS name")
    for record in result:
        print(record["name"])
with driver.session() as session:
    session.read_transaction(print_person_names)
driver.close()
```



**Köszönöm
a figyelmet!**