Régi anyag:
https://docs.google.com/presentation/d/1_nmA1F4ag_O-qlJAE4TfVuyfLgSruZLW/edit?usp=sharing&ouid=110539736176923279178&rtpof=true&sd=true

Adatbázisok története és fejlődése

Relációs adatbázisoktól a NoSQL-en át a vektor adatbázisokig

1. Bevezetés

Az adatbázisok az informatikai rendszerek alapvető elemei, amelyek lehetővé teszik az adatok hatékony tárolását, visszakeresését, rendezését és módosítását. A modern világban hatalmas mennyiségű adat keletkezik másodpercenként, és ezek strukturált tárolása elengedhetetlen a döntéstámogatáshoz, automatizáláshoz és gépi tanuláshoz.

Miért van szükség adatbázisokra?

- Az adatok szervezett tárolása lehetővé teszi azok gyors visszakeresését és elemzését.
- Több felhasználó egyidejűleg dolgozhat ugyanazon adatokkal (konkurens hozzáférés).
- Biztosítani lehet az **adatintegritást** és a **hozzáférés-ellenőrzést**.
- Az adatok rendszeres **mentése és visszaállítása** egyszerűbbé válik.
- Automatizált folyamatokhoz (pl. webalkalmazások, gépi tanulás) szükséges egy megbízható háttértároló.

Példa: Egy webshopban több ezer termék, rendelés és felhasználói adat van. Ezek kereshető tárolása adatbázis nélkül gyakorlatilag lehetetlen lenne.

Adat vs. információ

- **Adat**: nyers tények, mért vagy rögzített értékek, amelyek még nem feltétlenül bírnak jelentéssel.
  - Példa: `42`, `2025-05-07`, `„Piros”`
- **Információ**: értelmezett adat, amely kontextusba helyezve új tudást jelent.
  - Példa: „42 éves a felhasználó”, vagy „A rendelés dátuma: 2025-05-07”

Az adat tehát az alap, amiből – megfelelő feldolgozással – információt nyerhetünk. Az adatbázis célja az adatok strukturált tárolása annak érdekében, hogy információvá alakíthassuk őket.

Strukturált, félstrukturált és strukturálatlan adatok

- **Strukturált adat**:
  - Táblázatos formában rendezett, előre meghatározott sémával rendelkezik.
  - Példa: SQL adatbázisok, Excel-táblák, CRM rendszerek.
  - Jellemző: gyors kereshetőség, jól lekérdezhető.

- **Félstrukturált adat**:
- Nincs fix sémája, de tartalmaz címkéket vagy metaadatokat.
- Példa: XML, JSON, YAML fájlok, logok.
- Rugalmasabb, de nehezebb indexelni és lekérdezni.

- **Strukturálatlan adat**:
  - Nincs formázása, nem egyértelmű a tartalma.
  - Példa: szövegfájlok, képek, videók, e-mailek, hangfelvételek.
  - Ezeket általában keresőmotorokkal vagy mesterséges intelligenciával lehet feldolgozni (pl. szöveg- vagy képelemzés).

Az adatbázisok fejlődése szorosan összefügg azzal, hogyan és milyen típusú adatokat akarunk hatékonyan kezelni.

2. Relációs adatbázisok (RDBMS)

A relációs adatbázisok az egyik legelterjedtebb adatbázis-típusok közé tartoznak. Az adatok táblákban (relációkban) tárolódnak, ahol minden tábla sorokból (rekordokból) és oszlopokból (mezőkből) áll. A relációs modell megbízható és jól definiált struktúrát biztosít az adatok kezeléséhez.

Történet

- A relációs modellt **Edgar F. Codd** matematikus javasolta 1970-ben.
- A modell a halmazelméletre és a predikátumlogikára épül.
- A ’70-es, ’80-as években megjelentek az első RDBMS rendszerek: IBM System R, Ingres, Oracle.
- A **SQL (Structured Query Language)** nyelv szabványosítása nagyban hozzájárult az elterjedéséhez.

Jellemzők

- **Adatszerkezet**: táblázatos (sorok és oszlopok).
- **Szigorú séma**: minden táblának előre meghatározott szerkezete van (mezőtípusok, kulcsok stb.).
- **Kapcsolatok**: táblák között idegen kulcsokon keresztül hozunk létre kapcsolatokat.
- **ACID tulajdonságok**:
  - *Atomicity* – a tranzakciók oszthatatlanok.
  - *Consistency* – az adatbázis érvényességi szabályai nem sérülnek.
  - *Isolation* – párhuzamos tranzakciók nem befolyásolják egymást.
  - *Durability* – a végrehajtott tranzakciók tartósan megmaradnak.

Előnyök

- **Adatintegritás**: kulcsok, megszorítások és szabályok segítségével biztosítható.
- **Lekérdezhetőség**: a SQL nyelv rendkívül erős és kifejező eszköztárat ad.
- **Stabilitás és érettség**: hosszú távon kipróbált, optimalizált rendszerek.
- **Többfelhasználós működés**: jogosultságkezelés, tranzakciókezelés.

Hátrányok

- **Skálázhatóság**: nagy adatmennyiség és sok párhuzamos felhasználó esetén nehezen osztható szét több gépre (horizontális skálázás).
- **Rugalmatlanság**: séma módosítása bonyolult lehet, különösen ha az adatstruktúra gyakran változik.
- **Strukturálatlan adatok kezelése**: szöveg, kép, videó típusú adatok kezelésére nem ideális.

Népszerű relációs adatbázis-kezelő rendszerek (RDBMS-ek)

- **MySQL** – nyílt forráskódú, népszerű webalkalmazások körében (pl. WordPress).
- **PostgreSQL** – fejlett funkciók, szabványos SQL támogatás, objektum-orientált kiegészítések.
- **Oracle Database** – robusztus vállalati megoldás, fejlett biztonság és replikációs lehetőségek.
- **Microsoft SQL Server** – integráció Microsoft-technológiákkal, könnyű használat.

Példa: Egyszerű SQL tábla és lekérdezés

Tábla: `diakok`

id	nev	szuletesi_ev
1	Kovács Anna	2003
2	Nagy Péter	2002

SQL lekérdezés:

```
SELECT nev FROM diakok WHERE szuletesi_ev < 2003;
```

Ez a lekérdezés visszaadja azoknak a diákoknak a nevét, akik 2003 előtt születtek.

3. NoSQL adatbázisok

A NoSQL adatbázisok a relációs modellek alternatívájaként jelentek meg, amikor az internetes alkalmazások, a Big Data és a valós idejű feldolgozások új követelményeket támasztottak az adatkezeléssel szemben. A „NoSQL” nem feltétlenül jelenti azt, hogy nem használható benne SQL, hanem inkább azt, hogy nem relációs modell alapján működik.

Miért jelentek meg?

- Az adatok mennyisége gyorsan növekedett (pl. közösségi média, szenzoradatok).
- Az adatszerkezet egyre rugalmasabb lett – gyakran változó mezők, új attribútumok.
- Szükség volt **horizontális skálázásra** – sok szerver párhuzamos futtatása.
- Alkalmazásoknak nagy sebességre és alacsony késleltetésre volt szüksége.

Jellemzők

- **Sémanélküli modell**: nincs előre rögzített mezőszerkezet.
- **BASE modell**:
  - *Basically Available* – mindig elérhető adatbázis
  - *Soft state* – az adat idővel változhat
  - *Eventually consistent* – idővel elérhető a konzisztencia (nem garantált azonnal)
- **Horizontálisan skálázható**: könnyen bővíthető több gépre.
- **Nagy teljesítmény**: ideális valós idejű alkalmazásokhoz.

NoSQL típusok és példák

- **Dokumentum-orientált adatbázisok**
  - Minden rekord egy önálló dokumentum (pl. JSON formátumban).
  - Rugalmas szerkezet, ideális REST API-khoz.
  - Példa: **MongoDB**, CouchDB
- **Kulcs-érték tárolók**
  - Egyszerű szerkezet: kulcs → érték (mint egy szótár).
  - Nagyon gyors, kiváló gyorsítótárazásra.
  - Példa: **Redis**, Amazon DynamoDB
- **Oszlop-orientált adatbázisok**
  - Az adatok nem sorokban, hanem oszlopokban tárolódnak.
  - Hatékony analitikai feldolgozás nagy adattáblákon.
  - Példa: **Apache Cassandra**, HBase
- **Gráf adatbázisok**
  - Adatok és azok kapcsolatai gráfként ábrázolva (csomópontok és élek).
  - Nagyon hasznos hálózati elemzésekhez, pl. közösségi hálók, útvonalkeresés.
  - Példa: **Neo4j**, ArangoDB

Előnyök

- Nagy teljesítmény, alacsony válaszidő.
- Könnyen skálázható horizontálisan.
- Rugalmas adatszerkezet – ideális gyorsan változó alkalmazásokhoz.
- Néhány esetben beépített replikáció, sharding, elosztott feldolgozás.

Hátrányok

- Nincs szabványos nyelv (mint az SQL).
- Az adatintegritás és konzisztencia nem mindig garantált.
- Alkalmazásfejlesztéskor a logika egy része az adatbázis helyett a kódban jelenik meg.
- Komplex lekérdezések megvalósítása nehézkes lehet.

Példa: MongoDB dokumentum

Egy MongoDB kollekcióban tárolt dokumentum példája:

```
{
  "_id": "u123",
  "nev": "Kiss János",
  "eletkor": 29,
  "cim": {
    "varos": "Budapest",
    "iranyitoszam": "1111"
  }
}
```

Egy lekérdezés: azokat a felhasználókat kérdezzük le, akik életkora nagyobb mint 25:

```
db.felhasznalok.find({ "eletkor": { "$gt": 25 } })
```

4. NewSQL adatbázisok

A NewSQL adatbázisok a relációs adatbázisok továbbfejlesztett változatai, amelyek célja, hogy megtartsák a klasszikus RDBMS rendszerek előnyeit (pl. SQL, ACID), ugyanakkor képesek legyenek a modern alkalmazások által megkövetelt **horizontális skálázásra** és **magas teljesítményre**, mint a NoSQL rendszerek.

Miért jöttek létre?

- A vállalatok továbbra is igénylik a **strukturált, relációs adatsémát** és az **ACID** garanciákat.
- Ugyanakkor szükség van a **modern skálázhatóságra**, amit a NoSQL rendszerek nyújtanak.
- A cél: **„legyen meg a torta és együk is meg”** – relációs modell + felhőbarát architektúra.

Jellemzők

- **Relációs modell**: táblák, kulcsok, SQL nyelv
- **ACID tranzakciók**: konzisztencia és megbízhatóság
- **Elosztott architektúra**: több szerveren futó adatbázis
- **Skálázhatóság**: automatikus sharding, load balancing
- **Magas rendelkezésre állás**: beépített replikáció, hibatűrés

Előnyök

- Kombinálja a relációs adatbázisok előnyeit a NoSQL skálázhatóságával
- Alkalmas nagy forgalmú, üzletkritikus alkalmazásokhoz
- Használható ismerős SQL nyelvvel, nem igényel új tanulási görbét

Hátrányok

- Bonyolultabb architektúra és üzemeltetés, mint egy egyszerű SQL adatbázisnál
- Még viszonylag fiatal technológia – kisebb ökoszisztéma, kevesebb szakember
- Néhány megoldás zárt forráskódú, kereskedelmi licenc alatt áll

Példák

- **Google Spanner**
  - A Google saját globálisan elosztott relációs adatbázisa.
  - Világszintű szinkronizációval garantálja a konzisztenciát.
- **CockroachDB**
- Nyílt forráskódú, elosztott relációs adatbázis.
- Automatikus skálázás és replikáció.
- PostgreSQL-kompatibilis SQL interfész.
- **VoltDB**
  - Főként valós idejű analitikára és tranzakciókra specializált.
  - Nagy sebesség, memóriabeli működés.

Példa: CockroachDB SQL lekérdezés

```
CREATE TABLE felhasznalok (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  nev TEXT NOT NULL,
  regisztralt TIMESTAMP DEFAULT now()
);

SELECT * FROM felhasznalok WHERE nev LIKE 'K%';
```

A fenti példa egy CockroachDB tábla létrehozását és egyszerű lekérdezését mutatja be, PostgreSQL-kompatibilis SQL nyelven.

5. Vektor adatbázisok

A vektor adatbázisok új generációs adatbázis-rendszerek, amelyek a mesterséges intelligencia és a gépi tanulás elterjedésével váltak népszerűvé. Céljuk nem strukturált adatok – például szövegek, képek, hangok – jelentés szerinti keresése, amelyet az ún. *embedding* (beágyazott vektorreprezentáció) segítségével valósítanak meg.

Miért van rájuk szükség?

- A hagyományos adatbázisok nem alkalmasak jelentésalapú keresésre.
- Az LLM-ek, képfeldolgozók, hangfelismerők vektorokat állítanak elő, amelyek geometriailag

összehasonlíthatók.
- Példa: „melyik másik dokumentum tartalmilag hasonlít ehhez a kérdéshez?”

Hogyan működik?

- Az objektumokat (pl. szöveg, kép) **embedding**-gé alakítjuk egy neurális háló segítségével.
- Az így kapott vektorokat egy vektoradatbázis tárolja.
- Keresés során szintén vektort képezünk, majd megkeressük a **legközelebbi** (használatos: k-NN – *k Nearest Neighbors*) vektorokat.
- A hasonlóságot általában **koszinusz-hasonlóság**, **euklideszi távolság** vagy más metrika alapján számítják.

Jellemzők

- Nincs klasszikus SQL – speciális API vagy Python/REST kliensek használatosak.
- Beépített **indexelési módszerek** a gyors kereséshez (pl. HNSW, IVF, PQ).
- Gyors és hatékony **nagy dimenziójú adatok** keresése.

Előnyök

- Jelentésalapú keresés – szöveg vagy kép jelentésének megértése
- Ideális LLM-alapú rendszerekhez, mint pl. kérdés-válasz vagy dokumentumkeresés
- Kiváló teljesítmény nagy adathalmazokon is

Hátrányok

- Nem hagyományos adatmodell – nem tárol klasszikus táblákat, kapcsolatokkal
- A „miért kaptuk ezt a találatot?” kérdésre nehezebb választ adni (black-box jelleg)
- A pontosság függ az embedding minőségétől és a választott metrikától

Népszerű vektor adatbázisok

- **FAISS** (Facebook AI Similarity Search) – nagy teljesítményű, Python-könyvtárként is elérhető
- **Milvus** – nyílt forráskódú, skálázható, GPU-t is támogató vektor DB
- **Weaviate** – REST API + GraphQL támogatás, integrált gépi tanulási pipeline
- **Pinecone** – felhőalapú, egyszerűen használható szolgáltatás
- **Chroma** – egyszerű integráció LangChain/RAG rendszerekhez

Példa: vektoros keresés szövegek között

1. Szöveg → embedding: `“Milyen magas a Mount Everest?”` → `[0.12, -0.45, ..., 0.33]` 2. A vektoradatbázisban tárolt több ezer dokumentum embeddingje közül kiválasztjuk a legközelebbit:

```
query_vector = model.encode("Milyen magas a Mount Everest?")
results = vector_db.search(query_vector, top_k=5)
```

3. Eredmény: olyan dokumentumokat kapunk, amelyek tartalmilag legközelebb állnak a kérdéshez – még akkor is, ha szó szerint nem egyeznek meg.

6. Tendenciák és jövőkép

- Multimodális adatbázisok (szöveg + kép + struktúrált adatok)
- Hybrid keresés: SQL + vektoros keresés együtt
- AI-native rendszerek: LLM + adatbázis integráció (pl. LangChain)

7. Összefoglaló táblázat

Típus	Terméknevek	Előnyök	Hátrányok
Relációs (RDBMS)	MySQL, PostgreSQL, Oracle	Stabil, jól definiált séma, SQL támogatás, ACID garancia	Nehezen skálázható, rugalmatlan struktúra
NoSQL	MongoDB, Redis, Cassandra, Neo4j	Rugalmas séma, jól skálázható, gyors lekérdezések	Gyenge konzisztencia, nincs egységes lekérdezőnyelv
NewSQL	CockroachDB, Google Spanner, VoltDB	ACID + skálázhatóság, SQL-kompatibilitás	Bonyolultabb architektúra, kisebb ökoszisztéma
Vektor	FAISS, Milvus, Weaviate, Pinecone, Chroma	Jelentésalapú keresés, AI integráció, gyors k-NN keresés	Nem klasszikus lekérdezések, nehezebb magyarázhatóság

8. Példa

Az alábbi példák bemutatják, hogyan tér el az adatok kezelése a különböző adatbázistípusokban. A példák egyszerűek, mégis jól szemléltetik az eltérő adatmodelleket és lekérdezési módokat.

1. Relációs adatbázis: SQL példa (PostgreSQL)

Tábla létrehozása:

```
CREATE TABLE diakok (
  id SERIAL PRIMARY KEY,
  nev TEXT NOT NULL,
  szuletesi_ev INTEGER
);
```

Adatok beszúrása:

```
INSERT INTO diakok (nev, szuletesi_ev)
VALUES ('Kiss Anna', 2002), ('Nagy Péter', 2001);
```

Lekérdezés: ki született 2001 előtt?

```
SELECT nev FROM diakok WHERE szuletesi_ev < 2001;
```

2. NoSQL példa: MongoDB dokumentum és lekérdezés

Dokumentum beillesztése:

```
db.diakok.insertOne({
  nev: "Kiss Anna",
  szuletesi_ev: 2002
});
```

Lekérdezés: azok a diákok, akik 2001 előtt születtek:

```
db.diakok.find({ szuletesi_ev: { $lt: 2001 } });
```

3. Vektor adatbázis példa: FAISS keresés (Python)

Tegyük fel, hogy szövegeket átalakítunk vektorokká egy nyelvi modell (pl. `sentence-transformers`) segítségével, majd ezekben keresünk.

Embedding előállítás (példa):

```
from sentence_transformers import SentenceTransformer
import faiss
import numpy as np

model = SentenceTransformer('all-MiniLM-L6-v2')
dokumentumok = ["A kutya ugat.", "A macska nyávog.", "Az autó gyors."]

vektorok = model.encode(dokumentumok)
index = faiss.IndexFlatL2(vektorok.shape[1])
index.add(np.array(vektorok))
```

Keresés hasonló szöveg alapján:

```
kerdes = "Melyik állat nyávog?"
kerdes_vektor = model.encode([kerdes])
_, eredmenyek = index.search(np.array(kerdes_vektor), k=1)

print("Legjobban illeszkedő dokumentum:", dokumentumok[eredmenyek[0][0]])
```

Eredmény: