Curs 3 : **Modele de baze de date distribuite și mobile: Relaționale, NoSQL și hibride**

3.1 Modele relaționale

Modelul relațional este cel mai cunoscut și utilizat model de date în ultimii zeci de ani. **Baza de date relațională** organizează informația sub formă de tabele (relații) cu tupluri (rânduri) și atribute (coloane). Fiecare tabel are o cheie primară (identificator unic pe rând) și poate avea chei externe (foreign keys) ce leagă între ele tabelele, exprimând relații între entități. Modelul relațional este susținut de un **SGBD relațional (RDBMS)**, care oferă un limbaj standard de interogare (SQL – *Structured Query Language*) pentru definirea, manipularea și interogarea datelor. Studenții sunt deja familiarizați cu concepte ca integritatea referențială, normalizarea datelor și operațiile relaționale (JOIN, proiectări, selecții etc.) din cursurile anterioare, așa că în acest modul vom accentua aspectele relevante pentru distribuirea datelor și comparația cu noile modele.

Caracteristici cheie ale modelului relațional:

- **Schema rigidă și structurată:** Datele urmează o schemă predefinită (tabele și tipuri fixe de coloane). Această structurare facilitează consistența și calitatea datelor, dar face mai dificilă schimbarea modelului pe parcurs (orice modificare necesită migrarea sau modificarea schemei în SGBD).
- **ACID și tranzacționalitate:** SGBD-urile relaționale implementează proprietățile ACID (Atomicitate, Consistență, Izolare, Durabilitate) pentru tranzacții. Aceasta înseamnă că operațiile de citire/scriere pot fi grupate în tranzacții fiabile: fie toate operațiile din tranzacție au efect (commit), fie niciuna (rollback). Modelul relațional excelează la menținerea **consistenței puternice** a datelor într-un mediu centralizat, asigurând că toate interogările văd cea mai recentă actualizare.
- **Limbaj declarativ standard (SQL):** SQL permite interogări complexe (îmbinări de tabele, agregări, subinterogări) optimizate de motorul SGBD. Aceasta face bazele relaționale potrivite pentru rapoarte complexe și **analize ad-hoc** pe date structurate.
- **Scalare verticală tradițională:** Istoric, bazele de date relaționale au fost concepute să ruleze pe un singur server puternic, mărind capacitatea prin adăugarea de resurse hardware (scalare verticală). Deși multe SGBDR moderne suportă replicare și partiționare, în mod implicit ele favorizează consistența și disponibilitatea, nefiind tolerante la partiționări de rețea (conform teoremei CAP)[1]. Cu alte cuvinte, un SGBD relațional tipic oferă consistență și disponibilitate ridicată pe un singur nod, dar nu garantează funcționarea continuă dacă nodul devine izolat sau cade (lipsind toleranța la partiționare).

Baze de date relaționale în medii distribuite: În ultimii ani, necesitatea de a gestiona volume mari de date și trafic intens a forțat extinderea modelului relațional dincolo de un singur server. Două tehnici principale permit utilizarea SGBD relaționale la scară mai mare: **replicarea** și **fragmentarea (sharding)**.

- ✓ *Replicarea datelor:* SGBD-urile pot menține copii ale bazei de date pe servere secundare. De obicei există un nod principal (primary/master) care acceptă operațiile de scriere, replicând modificările către noduri secundare (replicații) folosite în principal pentru operații de citire sau ca backup pentru failover. Această abordare îmbunătățește disponibilitatea și permite distribuirea sarcinii la citire pe mai multe noduri, însă scrierile tot prin nodul principal trec, deci nu se obține neapărat o scalare lină pentru operațiile de

update[2][3]. Un exemplu popular este **MySQL** cu replicare master-slave (sau master-master în unele configurații), folosit în numeroase aplicații web.

- ✓ *Fragmentarea (sharding):* Baza de date este împărțită pe orizontală în mai multe fragmente distribuite pe noduri diferite. Fiecare nod stochează o parte din date (de exemplu, unele tabele sau partiții după intervale de chei). Sharding-ul crește capacitatea totală de stocare și throughput-ul, dar introduce complexitate majoră – aplicația sau un orchestrator trebuie să direcționeze interogările către fragmentul corect, iar operațiile care implică mai multe fragmente devin complicate (ex. join-uri distribuite, tranzacții peste noduri multiple)[4]. Un exemplu de fragmentare este distribuirea utilizatorilor pe mai multe servere după regiune. Sharding-ul a fost folosit de companii mari (ex. Facebook a fragmentat baza de date MySQL pentru utilizatori la nivel global), însă necesită efort semnificativ de administrare.

Chiar dacă se pot construi baze de date relaționale distribuite prin replicare și fragmentare, menținerea **consistenței** tranzacționale peste noduri multiple implică protocoale costisitoare (precum two-phase commit) și poate degrada performanța. În prezența unei defecțiuni de rețea între noduri, un sistem relațional configurat să sincronizeze strict toate replicile ar prefera să blocheze operațiile (pentru a menține consistența), afectând disponibilitatea serviciului[3]. De aceea, modelul relațional tradițional este adesea considerat mai puțin potrivit pentru scenarii de **scalare orizontală masivă** sau **sisteme distribuite geografic**, dacă se dorește menținerea proprietăților ACID strice.

Exemple și evoluții moderne (relațional):

Marile sisteme de gestionare a bazelor de date relaționale includ Oracle, IBM DB2, Microsoft SQL Server, PostgreSQL, MySQL/MariaDB etc., care domină în continuare piața enterprise pentru date structurate. Pentru a răspunde cerințelor moderne, aceste sisteme au evoluat: de exemplu, **Oracle** și **DB2** oferă opțiuni avansate de clusterizare și replicare, iar **PostgreSQL** și **MySQL** au introdus suport pentru stocarea JSON, indexare pe date semi-structurate și extensii NoSQL, despre care vom discuta la secțiunea hibridă. Au apărut, de asemenea, sisteme **NewSQL**, adică noi motoare de baze de date care păstrează modelul relațional și interfața SQL, dar sunt construite pentru a scala orizontal pe clustere distribuite și pentru a oferi performanțe similare NoSQL fără a sacrifica consistența ACID[5]. Exemple notabile de NewSQL includ **Google Spanner** (bază de date relațională global distribuită care folosește ceasuri fizice *TrueTime* pentru consistență globală), **CockroachDB** și **VoltDB**. Aceste sisteme demonstrează că modelul relațional poate fi adaptat pentru era distribuției și a volumelor mari de date, deși cu costul unei complexități de implementare sporite.

În contextul bazelor de date mobile, modelul relațional își găsește locul de exemplu prin **SQLite** – un SGBD relațional ușor, încorporat, folosit frecvent pe dispozitive mobile pentru stocare locală (aplicații Android, iOS etc.). SQLite păstrează simplitatea modelului relațional pe device, permițând aplicațiilor mobile să opereze offline cu o mică bază de date locală și apoi să sincronizeze datele cu un server central (de obicei prin servicii web). Astfel, cunoștințele de model relațional rămân relevante chiar și în scenarii mobile, deși adesea în combinație cu servicii cloud și cu modele NoSQL.

3.2 Modele NoSQL (documente, grafuri etc.)

Termenul **NoSQL** („Not Only SQL” sau „non-SQL”) se referă la o nouă generație de sisteme de baze de date care **nu urmează modelul relațional tradițional**, permițând stocarea și gestionarea datelor în formate mai flexibile. Aceste sisteme au apărut în anii 2000 ca răspuns la provocările pe care aplicațiile web moderne și sistemele la scară foarte mare le-au pus bazelor de date relaționale. Giganți precum Amazon, Google, Facebook, LinkedIn au fost pionieri în dezvoltarea unor soluții alternative, pentru a gestiona volume uriașe de informații, trafic de milioane de operații pe secundă și tipuri de date diverse (semi-structurate sau nestrcturate) pe care RDBMS-urile întâmpinau dificultăți în a le acomoda[6]. Modelul NoSQL nu este unic, ci cuprinde o varietate de modele de date (documente, grafuri, cheie-valoare, coloane largi etc.), fiecare optimizat pentru anumite cazuri de utilizare.

Trăsături generale ale bazelor de date NoSQL:

- **Fără schemă fixă (schema-free):** Majoritatea SGBD-urilor NoSQL nu impun o schemă statică a datelor. În loc de tabele cu coloane fixe, se permit structuri flexibile – de exemplu, documentele JSON pot avea câmpuri diferite de la un document la altul, iar într-un magazin cheie-valoare orice valoare arbitrară poate fi stocată sub o anumită cheie. Această flexibilitate sporește viteza de dezvoltare (modelul de date se poate modifica oricând, nu este necesară migrarea întregii baze) și permite stocarea de **date semi-structurate sau nestandardizate** (ex: obiecte complexe, fișiere JSON/XML, date binare etc.)[7][8].
- **Scalare orizontală și distribuție inerentă:** SGBD-urile NoSQL sunt concepute de la bun început să ruleze într-o **arhitectură distribuită**, pe clustere de servere ieftine (commodity hardware). Datele sunt **replicate** și **partiționate** automat pe mai multe noduri pentru a asigura toleranța la defecțiuni și extinderea capacității prin adăugarea de noduri noi[9]. De exemplu, într-o bază NoSQL, dacă un nod se defectează, copiile de pe alte noduri asigură disponibilitatea, iar dacă volumul de date crește, se pot adăuga noduri suplimentare pe care sistemul reechilibrează fragmentele. Acest model a permis companiilor web să atingă niveluri extreme de scalabilitate și disponibilitate, adesea sacrificând *consistența puternică* în favoarea *consistenței eventuale* – conform teoremei CAP, majoritatea bazelor NoSQL aleg să garanteze Disponibilitatea și Toleranța la Partiții, permițând ca replicile să fie nu întotdeauna perfect sincronizate tot timpul (consistența devine **eventuală**, adică nodurile converg către același conținut într-un interval de timp)[10].
- **Performanță și simplitate în anumite sarcini:** NoSQL optimizează operațiile simple de citire/scriere pe structuri de date specifice (de ex., citirea unei valori după cheie) și evită costul operațiilor complexe tipice SQL (nu există join-uri costisitoare la nivel de SGBD; dacă e nevoie de combinații de date, de multe ori aplicația le gestionează). De asemenea, prin denormalizarea datelor (stocarea redundanta a informației legate, în loc de a o normaliza în tabele separate) se pot obține timpi de răspuns foarte mici pentru interogările frecvente. Acest lucru vine însă cu **dezavantajul** potențial al inconsistenței datelor denormalizate și cu efort mutat la nivel de logică a aplicației pentru menținerea integrității, ceea ce necesită proiectare atentă.

Există **patru categorii principale de modele NoSQL**, pe care le vom detalia mai jos, cu exemple și cazuri de utilizare:

1) **Baze de date cheie-valoare (Key-Value Stores):** Sunt cele mai simple tipuri de NoSQL – stochează o colecție de perechi *cheie–valoare*, similar unui dicționar sau unei hărți asociative. Cheia este un identificator unic, iar valoarea poate fi orice tip de date (nestructurate binar sau structurate într-un format nativ aplicației). Aceste sisteme excelează în **viteză și scalare orizontală**, deoarece operațiile de bază sunt foarte simple (get/put pe cheie) și pot fi repartizate facil pe noduri diferite pe baza hash-ului cheii. *Cazuri de utilizare:* cache distribuit (ex. **Redis** este adesea folosit pentru cache de sesiune sau date temporare), stocarea de preferințe ale utilizatorilor, coșuri de cumpărături (Amazon a descris în celebrul articol Dynamo o astfel de bază key-value pentru coșul de cumpărături online). **Exemple de SGBD cheie-valoare: Redis** (stocare în memorie, foarte rapidă, cu persistență opțională), **Memcached** (cache in-memory volatil), **Riak** (sistem distribuit tolerant la partiții, inspirat de Dynamo) etc. Acestea oferă adesea consistență eventuală și un API simplu (get/put/delete), fără limbaj de interogare declarativ.

2) **Baze de date orientate pe documente:** Stochează informația sub formă de **documente** auto-conținute, de obicei folosind un format precum JSON sau BSON (o variantă binară de JSON). Un document reprezintă un agregat de date (echivalentul unui obiect complex sau a unui „rând" cu structură internă ierarhică). Documentele sunt grupate în **colecții** (analog cu tabelele din SQL, dar fără o schemă fixă impusă – fiecare document din colecție poate avea propriul set de câmpuri). Acest model este foarte flexibil: dezvoltatorii pot adăuga sau omite atribute fără operațiuni de alterare a schemei și pot imbrica (embeda) sub-documente sau liste direct într-un document pentru a reprezenta relații 1-la-multe fără a apela la tabele separate. *Cazuri de utilizare:* aplicații web ce manipulează obiecte JSON (ex. profiluri de utilizator, articole de blog, mesaje în format JSON), sisteme de gestionare a conținutului, cataloage de produse cu atribute variabile. **Exemple populare: MongoDB** – probabil cea mai cunoscută bază de date NoSQL document, open-source, foarte folosită pentru aplicații web moderne datorită ușurinței în dezvoltare și a scalabilității sale[11]. MongoDB permite atât stocarea de documente neuniforme, cât și indexarea pe câmpurile din interiorul documentelor, suportă replicare și sharding integrat, oferind un limbaj de interogare proprie (bazat pe JSON) pentru filtre și agregări. Alte exemple: **CouchDB** și **Couchbase** (stochează documente JSON, cu replicare master-master și sincronizare - CouchDB este folosit de ex. la aplicații cu replicare ocazional deconectată, inclusiv în mediu mobil), **Amazon DynamoDB** (serviciu cloud NoSQL care intern folosește model cheie-document și oferă performanță la scară Amazon). Document stores au avantajul că datele relativ complexe pot fi obținute dintr-o singură citire (dacă sunt toate într-un document, evitând join-uri), însă interogările care implică corelarea documentelor diferite pot fi dificile sau ineficiente.

3) **Baze de date coloană largă (Wide-Column Stores):** Acest model, inspirat de sistemul **Bigtable** al Google, organizează datele pe coloane familii de coloane în loc de rânduri tradiționale. Practic, datele sunt stocate într-o structură matricială, unde un rând poate avea un număr variabil de coloane grupate pe familii. Cheia primară include adesea un identificator de rând și o coloană (sau un *timp* pentru seriile de timp). Modelul de coloană largă este foarte potrivit pentru **seturi masive de date și acces de tip scalabil** – de exemplu stocarea de log-uri, date de telemetrie, date de serie de timp, unde fiecare intrare poate avea atribute

suplimentare. *Cazuri de utilizare:* aplicații de analitică la scară mare, rețele de socializare (de ex. Facebook Messenger a folosit Cassandra pentru stocarea mesajelor), stocarea indexurilor pentru motoare de căutare, sisteme de recomandare care înregistrează acțiunile utilizatorilor. **Exemple cunoscute: Apache Cassandra** – o bază de date open-source derivată din Bigtable și Dynamo, creată inițial la Facebook, care este complet distribuită (fără master central), asigurând toleranță ridicată la defecțiuni și scriere/lectură pe mai multe centre de date. Cassandra și sisteme similare (ex: **Apache HBase** – care rulează peste Hadoop HDFS) asigură consistență configurabilă (prin mecanisme de *quorum*) și pot gestiona volume de date de ordinul petabyților cu mii de noduri. **Google Bigtable** (serviciu intern Google, disponibil și ca Google Cloud Bigtable) și **Amazon DynamoDB** (menționat și la documente, dar oferă și model tip tabel dinamic) intră tot în această categorie. Pentru dezvoltatori, modelele coloană largă sunt mai greu de folosit direct (nu au SQL complet, dar deseori oferă API sau limbaje specifici, ex. CQL – Cassandra Query Language – asemănător SQL pentru Cassandra).

4) **Baze de date graf:** Concepute special pentru a stoca și interoga date puternic relaționale, al căror **model natural este un graf** (noduri conectate prin muchii). În loc de tabele sau documente, un SGBD de graf gestionează **noduri** (care reprezintă entități, fiecare având proprietăți sub formă de atribute) și **muchii** (*edge-uri*, care reprezintă relațiile directe dintre noduri, și pot avea și ele proprietăți). Avantajul acestor sisteme este că permit **traversări rapide ale relațiilor** de orice adâncime și complexitate – lucru greu de obținut prin join-uri recursive în SQL. *Cazuri de utilizare:* rețele sociale (conexiuni între persoane, like-uri, follow etc.), detecția de fraude (conexiuni între tranzacții, conturi, IP-uri), motoare de recomandare și cunoaștere (grafuri de cunoștințe, relații între termeni), gestionarea ierarhiilor sau a rețelelor (rutare pe hărți, rețele telecom). **Exemple celebre: Neo4j** – cel mai popular SGBD de graf, folosit pentru grafuri sociale și de cunoștințe, oferă limbajul de interogare Cypher foarte expresiv pentru pattern matching în graf. **Apache JanusGraph** (stochează grafuri mari pe infrastructura distribuită existente, suportând diverse backends precum Cassandra sau HBase pentru stocare), **Amazon Neptune** (serviciu cloud de graf de la Amazon), **OrientDB** (un SGBD multimodel care suportă și graf, vom aminti la hibride) etc. Bazele de graf excelează în interogări de tip "găsește relații n-arce între X și Y", dar nu sunt optimizate pentru scanning full sau procesare analitică la scară, de aceea adesea se folosesc în combinație cu alte sisteme pentru aspecte diferite ale datelor.

Avantaje și limitări NoSQL:

Beneficiile NoSQL decurg în principal din flexibilitate și scalare. Permițând modele de date adaptate cazului (ex: JSON pentru date ierarhice, graf pentru rețele complexe), aceste sisteme pot oferi **performanță masivă** când sunt proiectate corect – de exemplu, latențe de milisecunde chiar și sub încărcare intensă, și capacitatea de a gestiona volume înainte de neconceput cu RDBMS (Facebook, Amazon, Google rulează baze NoSQL care *înmagazinează petabyți de date și servesc milioane de operații pe secundă*). Fiind **distribuite**, ele asigură toleranță la căderi: dacă un nod sau un întreg centru de date e inaccesibil, arhitectura multi-nod permite sistemului să rămână funcțional (poate cu unele degrade de consistență temporară). În plus, lipsa rigidității de schemă înseamnă că echipele pot itera rapid pe dezvoltarea aplicațiilor fără barierele impuse de migrarea schemelor relaționale.

Pe de altă parte, aceste beneficii vin cu compromisuri.

- **Consistența eventuală** înseamnă că aplicația trebuie să tolereze faptul că imediat după o scriere, o citire de pe alt nod s-ar putea să nu vadă actualizarea, ceea ce complică dezvoltarea (trebuie gestionate versiuni, reîncercări, reconcilieri).
- **Lipsa suportului tranzacțional** general (unele NoSQL oferă tranzacții, dar limitat la o singură cheie sau document) cere ca integritatea datelor să fie asigurată în codul aplicației sau prin strategii compensatorii (pattern-uri ca *Saga*, event sourcing etc., care depășesc scopul acestui curs).
- **Funcționalitatea de interogare** e adesea mai limitată – de exemplu, în loc de join-uri flexibile, NoSQL cere deseori să duplicăm datele relevante într-un singur document sau denormalizat, pentru a le putea prelua dintr-o lovitură.

Administrarea clusterelor mari de NoSQL necesită cunoștințe specializate, iar ecosistemul de instrumente de monitorizare, backup, securitate este mai tânăr comparativ cu cel matur din lumea SQL.

Exemple din industrie (NoSQL):

- ✓ **Amazon & DynamoDB:** Amazon a dezvoltat la mijlocul anilor 2000 sistemul **Dynamo** – un magazin cheie-valoare distribuit, pentru a asigura disponibilitate maximă și scalare orizontală la nivelul serviciilor lor de cumpărături (ex. coșul de cumpărături global). Ulterior principiile Dynamo (hashing distribuit, replicare cu consistență eventuală, reechilibrare automată) au stat la baza multor proiecte NoSQL open-source (Riak, Cassandra). Amazon oferă astăzi **DynamoDB** ca serviciu complet gestionat, folosit pe scară largă de clienți enterprise pentru aplicații cu latență foarte mică și cerințe elastice de throughput.
- ✓ **Facebook & Cassandra:** Facebook a creat inițial **Cassandra** pentru a gestiona mesageria instantanee și *inbox*-ul utilizatorilor, necesitând scriere intensă și replicare pe mai multe centre de date. Cassandra a devenit open-source și este acum folosită de companii precum Netflix, Twitter, Reddit pentru stocarea de evenimente, log-uri, metrici, mesagerie sau feed-uri. De exemplu, **Netflix** rulează clustere Cassandra pentru a păstra statusul de streaming și vizualizările utilizatorilor la nivel global, profitând de replicarea multi-regiune și de arhitectura fără punct unic de eșec.
- ✓ **MongoDB în aplicații web:** MongoDB a fost adoptat rapid de startup-uri și echipe de dezvoltare web datorită flexibilității sale. De exemplu, platforma de anunțuri **Craigslist** a folosit MongoDB pentru arhivarea a milioane de postări (documente semi-structurate variind ca format). Companii din fintech, jocuri online, IoT folosesc MongoDB pentru stocarea datelor generate de utilizatori sau echipamente în formate JSON fluide. MongoDB Inc. oferă și serviciul cloud Atlas, facilitând utilizarea Mongo în aplicații mobile (de exemplu, sincronizarea datelor de pe dispozitive mobile cu o bază Mongo în cloud, folosind Realm).
- ✓ **Sisteme de graf în rețele sociale și recomandări: LinkedIn** a dezvoltat intern un motor de graf pentru a calcula legături de gradul al doilea și al treilea între profesioniști, optimizând recomandările de conexiuni. **Twitter** a utilizat elemente de graf pentru a determina sugestii de „cine să urmărești”. **Neo4j** este folosit de organizații pentru detectarea fraudelor financiare (conexiuni între tranzacții suspecte și entități) sau de companii de transport pentru optimizarea rutelor. Un exemplu public este utilizarea Neo4j în Panama Papers pentru a identifica rețele complexe de entități și proprietari de firme offshore.

În concluzie, NoSQL reprezintă o familie diversă de tehnologii, fiecare cu propriul punct forte. Alegerea modelului NoSQL potrivit depinde de cerințele aplicației: dacă avem date foarte conectate – un graf DB poate fi indicat; pentru conținut flexibil de tip document – un document store e preferat; pentru streaming de evenimente la volum mare – un cluster de coloane largi poate fi soluția; pentru cache ultra-rapid – un key-value in-memory este ideal etc. Adesea însă, în sisteme complexe apare nevoia de a **combina** mai multe tehnologii pentru a obține ce e mai bun din fiecare. Astfel ajungem la conceptul de **modele hibride** și persistenta poliglotă.

3.3 Modele hibride

Prin **modele hibride** ne referim la abordări și sisteme de baze de date care îmbină caracteristicile modelului relațional cu cele ale modelelor NoSQL, fie într-o singură platformă unificată, fie într-o arhitectură ce folosește mai multe sisteme specializate împreună. Necesitatea unor soluții hibride a apărut deoarece organizațiile doresc atât beneficiile NoSQL (scalabilitate, flexibilitate), cât și avantajele modelului relațional (consistență tranzacțională, bogăția interogărilor SQL) în același sistem.

Vom discuta trei direcții principale:

(a) sisteme multimodel care suportă mai multe paradigme de date nativ

(b) extensii NoSQL în SGBD relaționale tradiționale

(c) arhitecturi poliglote ce combină mai multe baze diferite într-o soluție.

a. Baze de date multimodel: Acestea sunt sisteme proiectate de la zero pentru a suporta **nativ mai multe modele de date**. În loc să se limiteze la un singur tip (doar tabelar, doar document sau doar graf), un SGBD multimodel permite stocarea și interogarea datelor sub diferite forme, oferind un singur motor unificat.

De exemplu, **OrientDB** și **ArangoDB** sunt baze de date multimodel care permit atât stocarea de documente JSON, cât și a datelor de graf și a perechilor cheie-valoare, cu un limbaj de interogare comun.

ArangoDB suportă trei modele (documente, graf, cheie-valoare) într-o singură arhitectură, permițând dezvoltatorului să aleagă cea mai convenabilă reprezentare pentru fiecare entitate și chiar să combine interogări (ex. o interogare poate naviga relații de graf între documente).

OrientDB oferă de asemenea mod de operare ca SGBD de graf și document, plus elemente de stocare pe coloane.

Alt exemplu este **Couchbase Server**, care combină funcționalități de magazin de documente JSON cu indexare tip SQL (prin N1QL, un limbaj asemănător SQL) și caching in-memory (inspirat de Memcached).

Microsoft Azure Cosmos DB este un serviciu cloud multimodel care, deși intern folosește un model de date uniform, expune mai multe API-uri compatibile (documente tip MongoDB, tabele cheie-valoare tip Azure Table, grafuri Gremlin, și chiar SQL pentru modelul de documente), permițând dezvoltatorilor să lucreze cu modelul preferat sub acoperișul aceluiași serviciu.

Avantajul acestor sisteme multimodel este **flexibilitatea ridicată**: o singură soluție poate gestiona tipuri variate de date, reducând complexitatea infrastructurii (nu mai ai nevoie de 3-4 sisteme separate pentru fiecare tip de date)[12]. Totuși, există și compromisuri – implementarea unui motor care să exceleze simultan pe modele diferite este dificilă, astfel că performanța pe un anumit tip de sarcină poate să nu atingă nivelul unei soluții specializate. În plus, maturitatea

uneltelor de jur (driver-e, instrumente de administrare) poate fi mai scăzută comparativ cu sistemele consacrate pe modelul lor.

b. Extinderea SGBD relaționale cu funcționalități NoSQL: O altă abordare hibridă este **îmbogățirea sistemelor relaționale existente** cu capacități non-relaționale. Marii producători de baze de date au recunoscut tendința NoSQL și au integrat funcții noi pentru a reține utilizatorii care altfel ar migra la soluții NoSQL.

De exemplu, **IBM DB2** (începând cu versiuni ~2013) a introdus **BLU Acceleration** și suport pentru stocarea documentelor JSON și a grafurilor direct în baza relațională[13]. Asta înseamnă că într-un tabel DB2 poți avea coloană de tip JSON, interogabilă cu expresii JSONPath, sau poți crea grafuri (noduri și muchii) ca o vedere peste date relaționale, permițând interogări de tip graf.

Similar, **Oracle** și **Microsoft SQL Server** suportă date JSON: pot indexa conținut JSON în coloane, permițând practic să folosești baza relațională ca un depozit de documente.

PostgreSQL a făcut un pas major în această direcție – pe lângă tipul JSONB (binar JSON cu indexare eficientă) suportând și operatori NoSQL – PostgreSQL are și extensii precum **HStore** (perechi cheie-valoare înregistrate în coloană) și module pentru graf (ex. pgRouting pentru grafuri rutiere, sau funcții de tip closure table pentru relații ierarhice).

Astfel, un SGBD relațional modern poate acționa **hibrid**, permițând simultan stocarea datelor foarte structurate în tabele normale și a datelor semi-structurate în coloane JSON sau scheme flexibile.

Această soluție este atractivă deoarece organizațiile își pot păstra infrastructura și cunoștințele SQL, adăugând totodată suport pentru noi tipuri de date.

Dezavantajul potențial este că la bază motorul rămâne unul relațional, astfel încât unele beneficii NoSQL (scalarea masivă orizontală, arhitectura share-nothing) nu se obțin automat – deși aceste SGBD extinse pot manipula JSON, ele tot pe un nod central sau cluster restrâns rulează în majoritatea cazurilor, deci dacă cerința principală e scalarea la nivel NoSQL, nu întotdeauna un RDBMS extins e suficient.

c. Arhitectura poliglotă (Polyglot Persistence): În loc să aștepte un singur produs „să le facă pe toate", multe companii adoptă o abordare **poliglotă**, folosind *împreună* mai multe sisteme de baze de date, fiecare pentru ceea ce se pricepe mai bine. Ideea de **persistență poliglotă** este că într-o aplicație complexă poți folosi de exemplu un SGBD relațional pentru date critice tranzacțional (ex: informații financiare, date master despre clienți), un magazin de documente pentru stocarea conținutului nestructurat sau a log-urilor, și o bază de date de graf pentru analiza relațiilor din rețeaua de clienți – integrându-le la nivelul logicii de aplicație.

Un studiu de caz ar fi o aplicație e-commerce modernă: **stocarea produselor și a comenzilor** s-ar putea realiza într-o bază relațională (asigurând integritatea tranzacțională a comenzilor și inventarului), **recenziile sau comentariile clienților** ar putea fi stocate într-o bază NoSQL de documente (având format flexibil, posibil cu volum mare, nu afectează tranzacțiile de bază), iar **sistemul de recomandări** pentru produse ar putea folosi o combinație de date: un graf al achizițiilor comune (poate într-o bază de graf sau într-un motor de procesare tip Spark) și un cache cheie-valoare (Redis) pentru rezultate de recomandare frecvent accesate. Această coabitare a mai multor tehnologii permite optimizarea fiecărui modul, dar aduce provocări de orchestrare: trebuie asigurată sincronizarea datelor între sisteme, echipa de dezvoltare trebuie să stăpânească multiple API-uri și limbaje de interogare, iar infrastructura devine mai complexă. Cu toate acestea,

persistența poliglotă este din ce în ce mai folosită în arhitectura de microservicii: fiecare microserviciu își poate alege tipul de bază de date potrivit nevoilor sale (un serviciu folosește un PostgreSQL, altul un MongoDB, altul un Neo4j, etc.), comunicând între ele prin API-uri și menținându-și propriul *storage* independent.

d. NewSQL – un hibrid de model și performanță: Am menționat deja NewSQL la secțiunea relațională, dar îl reiterăm aici ca făcând parte din tendința hibridă. **NewSQL** desemnează sisteme care mențin **modelul relațional și SQL-ul** familiar, însă intern arhitectura este re-gândită pentru a fi distribuită, astfel încât să ofere *scalabilitatea și viteza NoSQL-urilor* cu *consistența ACID a SQL-ului*. Google Spanner, CockroachDB, Amazon Aurora, Microsoft Azure SQL Hyperscale pot fi văzute ca exemple de NewSQL (deși unele sunt extinderi ale produselor existente). NewSQL este un răspuns direct la dilema CAP: încearcă să furnizeze Consistență și Partiționare simultan (și să mențină cât mai mult posibil Disponibilitatea), prin inovații la nivel de replicare sincronică, acordarea de ceasuri logice sau fizice precise și distribuirea datelor cu minimizarea operațiilor distribuite blocante. Un citat relevant din documentația Microsoft despre arhitecturi cloud notează: *„A apărut un nou tip de baze de date, numit NewSQL, care extinde motoarele relaționale pentru a suporta scalarea orizontală și performanța scalabilă specifică sistemelor NoSQL”*[5]. Așadar, NewSQL poate fi considerat o soluție hibridă la nivel de concept (îmbină proprietăți din cele două lumi), deși din punct de vedere al modelului de date rămâne în domeniul relațional.

Exemple de soluții hibride din industrie:

- ✓ **IBM Db2 și Oracle multimodel:** După cum am menționat, IBM și Oracle au adăugat în SGBD-urile lor flagship suport pentru JSON, XML și graph. IBM promovează Db2 ca platformă „unificată” unde poți rula atât interogări SQL clasice, cât și interogări **NoSQL** (de exemplu, folosind **MongoDB API compatibil** – Db2 are un modul care înțelege protocolul MongoDB, astfel încât aplicațiile pot comunica cu Db2 ca și cum ar fi un Mongo). Oracle, similar, are Oracle JSON Document Store în cadrul Oracle Database, și un modul Oracle Spatial and Graph pentru management de date graf. Astfel, clienții acestor platforme pot avea o **experiență hibridă** fără a ieși din ecosistemul produsului.
- ✓ **OrientDB și ArangoDB în aplicații reale:** OrientDB a fost folosit, de exemplu, de către compania **Cisco** pentru un sistem de management al rețelei, unde aveau nevoie să stocheze atât informații structurale (tipuri de dispozitive, atribute) cât și relații topologice complexe – combinând astfel modelul document cu cel de graf. ArangoDB este folosit de aplicații de analytics care vor să proceseze log-uri (documente JSON), dar și să extragă relații de cauzalitate sau secvențe (unde un graf poate modela evenimentele într-un flux). Faptul că ambele tipuri de date sunt în aceeași bază reduce latența integrării (nu trebuie să exporți dintr-un sistem în altul, ci poți face o interogare combinată).
- ✓ **Polyglot în e-commerce și servicii financiare:** Este din ce în ce mai obișnuit ca un serviciu complex să fie construit pe multiple subsisteme. De exemplu, **Netflix** (un exemplu clasic) folosește **Cassandra** pentru managementul conținutului și streaming, **Dynomite** (un wrapper peste Redis) pentru cache distribuit de sesiuni, **MySQL** pentru datele conturilor de utilizator (consistente și critice), și **ElasticSearch** pentru funcții de căutare text și recomandări. În domeniul financiar, o aplicație de banking modern poate folosi un mainframe/DB2 sau Oracle pentru registre contabile (consistență strictă), un sistem NoSQL precum **MongoDB** pentru a stoca istoricul de tranzacții al clienților în format de document (volum mare, query flexibil pe diverse atribute), și un engine de graf cum e

Neo4j pentru a rula algoritmi de detectare a fraudelor între conturi. Aceste componente sunt orchestrate împreună pentru a oferi o soluție unitară – arhitectura trebuie să aibă grijă de coerența globală (de exemplu, dacă o tranzacție e salvată în RDBMS, să apară și în MongoDB, eventual asincron prin evenimente).

În concluzie, **modelele hibride** reflectă realitatea că **un singur tip de bază de date rareori mai satisface toate cerințele** diverselor aplicații moderne. Fie prin produse multimodel, fie prin combinarea mai multor sisteme specializate, arhitecții de sisteme caută să obțină un echilibru optim între consistență, performanță, flexibilitate și simplitatea dezvoltării.

Bibliografie :

[1] [2] [3] [4] [5] [10] [15] [16] Relational vs. NoSQL data - .NET | Microsoft Learn
https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data
[6] [7] [9] NoSQL databases explained: Unlock data flexibility and real-time performance | LogicMonitor
https://www.logicmonitor.com/blog/what-is-nosql
[8] Microsoft Word - 06 art. Ularu si Puican revizuit.doc
https://rria.ici.ro/documents/370/06-art.-Ularu-si-Puican-revizuit-1.pdf
[11] [12] [13] Hybrids, in-memory and as a service: the best databases of 2025
https://pandorafms.com/blog/database-types-and-best-ones/
[14] Principles of Distributed Database Systems - SpringerLink
https://link.springer.com/book/10.1007/978-3-030-26253-2
[17] Multi-model database systems as a new trend!
https://db-engines.com/de/blog_post/29