

VÝVOJ DATABÁZOVÝCH SYSTÉMOV

Relačné a nerelačné databázy

Študijná kapitola k prednáške

Od súborových systémov a Coddovho relačného modelu po NoSQL, distribuované databázy, cloud a vektorové vyhľadávanie.

Cieľ kapitoly

Cieľom kapitoly je vysvetliť, prečo databázové systémy vznikli, ako sa vyvíjali ich dátové modely a architektúry a podľa akých kritérií sa dnes rozhoduje medzi relačným, dokumentovým, key-value, wide-column a grafovým prístupom. Dôraz je na porozumení kompromisom, nie na jednoduchom delení technológií na „staré“ a „nové“.

1. Databáza, DBMS a dátový model

Databáza je organizovaná kolekcia údajov. Databázový systém, resp. systém riadenia bázy dát (DBMS – Database Management System), je softvér, ktorý umožňuje údaje ukladať, vyhľadávať, meniť, chrániť, zdieľať a obnovovať. Dátový model určuje, akým spôsobom sú dáta logicky organizované a aké vzťahy medzi nimi vieme vyjadriť. Dotazovací jazyk je spôsob, akým používateľ alebo aplikácia formuluje požiadavky voči DBMS.

Databáza ≠ DBMS ≠ SQL.

Pojem	Význam	Príklad
Databáza	Konkrétne uložené dáta a ich štruktúra	evidencia študentov, predmetov a známok
DBMS	Softvér riadiaci databázu	PostgreSQL, Oracle Database, MongoDB
Dátový model	Logický spôsob reprezentácie údajov	relačný, dokumentový, grafový
Dotazovací jazyk/API	Prostriedok komunikácie s DBMS	SQL, Cypher, MongoDB Query API

PostgreSQL sa napríklad označuje ako objektovo-relačný DBMS. Relačný model však nie je synonymom jedného produktu a SQL nie je databáza – ide o jazyk používaný množstvom databázových systémov.

2. Pred databázami: súborovo orientované spracovanie

Prvé informačné systémy ukladali dáta priamo do súborov. Každá aplikácia poznala ich konkrétny formát a sama implementovala logiku načítania, vyhľadávania a aktualizácie. Takýto prístup funguje pri malých a izolovaných úlohách, no pri rastúcom objeme údajov prináša zásadné problémy.

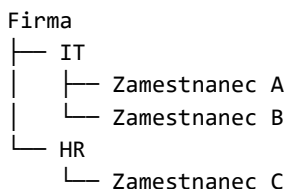
```
studenti.dat
predmety.dat
znamky.dat
ucitelia.dat
```

- silná závislosť programu od fyzickej štruktúry súborov;
- duplicitné dáta v rôznych aplikáciách;
- ťažšie udržiavanie konzistencie;
- komplikovaný súbežný prístup viacerých používateľov;
- náročné zabezpečenie, auditovanie, zálohovanie a obnova;
- potreba programovať nové procedúry pri každom novom type otázky nad dátami.

Jednou zo základných motivácií vzniku DBMS preto bolo oddeliť správu údajov od aplikačnej logiky a poskytnúť nad dátami spoločnú, kontrolovanú vrstvu.

3. Hierarchické a sieťové databázy

V 60. rokoch sa rozšírili hierarchické databázy. Dáta organizovali do stromovej štruktúry rodič–potomok. Takýto model bol efektívny pri úlohách, ktoré prirodzene kopírovali hierarchiu, no bol rigidný pri zložitejších vzťahoch. IBM vo svojich historických materiáloch uvádza hierarchické systémy ako významného predchodcu relačných databáz.



Sieťový databázový model umožnil záznamom zložitejšie väzby a viacero ciest medzi objektmi. Programátor však často musel poznať navigačné cesty v dátach. Aplikácia teda bola stále výrazne previazaná so spôsobom, akým boli vzťahy fyzicky alebo logicky reprezentované.

4. Coddov relačný model: zásadný obrat

V roku 1970 publikoval Edgar F. Codd z IBM prácu „A Relational Model of Data for Large Shared Data Banks“. Relačný model navrhol reprezentovať údaje pomocou relácií a manipulovať s nimi deklaratívnejším spôsobom, ktorý používateľa oddeľuje od detailov fyzického uloženia. Coddova práca sa stala teoretickým základom relačných databáz.

Kľúčový posun nebol iba v tom, že údaje „vyzerajú ako tabuľky“. Podstatná bola dátová nezávislosť: používateľ alebo aplikácia môže formulovať otázku nad logickou štruktúrou dát bez explicitného programovania fyzickej navigácie po uložených záznamoch.

STUDENT		
ID	Meno	Priezvisko
1	Peter	Novák
2	Jana	Malá

V relačnej terminológii je relácia množina n-tíc (tuples) s atribútmi. Praktické databázové systémy túto abstrakciu sprístupňujú najmä prostredníctvom tabuliek, riadkov a stĺpcov.

5. System R, SQL a optimalizácia dotazov

V roku 1973 začalo IBM pracovať na projekte System R, ktorý mal ukázať, že relačný model možno implementovať ako praktický databázový systém. System R sa stal významným testovacím prostredím pre SQL a pre vývoj automatickej optimalizácie dotazov. Donald

Chamberlin a Raymond Boyce vytvorili jazyk SEQUEL, neskôr známy ako SQL, a Patricia Selinger významne prispela nákladovým optimalizátorom dotazov.

```
SELECT meno, priezvisko
FROM student
WHERE rocnik = 2
ORDER BY priezvisko;
```

SQL je deklaratívny jazyk: používateľ prevažne opisuje, aký výsledok chce. DBMS vytvára vykonávací plán – napríklad volí index, poradie JOIN operácií alebo spôsob prístupu k tabuľke. Táto schopnosť je jedným z dôvodov, prečo sa relačné systémy stali univerzálnym základom podnikových informačných systémov.

6. Rozmach relačných databáz

Od konca 70. a počas 80. rokov sa relačné systémy komercializovali a postupne sa stali dominantným modelom pre podnikové spracovanie dát. IBM uvádza prvé dodanie DB2 v roku 1983; Oracle patrilo k prvým komerčným relačným produktom. [2][3]

Dôležitou vetvou vývoja bol akademický projekt POSTGRES na University of California, Berkeley, vedený Michaelom Stonebrakerom. Implementácia POSTGRES sa začala v roku 1986. Z tohto projektu sa neskôr vyvinul PostgreSQL, dnešný objektovo-relačný open-source DBMS. [4]

Medzi známe relačné databázové systémy dnes patria PostgreSQL, MySQL, MariaDB, Oracle Database, Microsoft SQL Server, IBM Db2 a ďalšie.

7. Základy relačného návrhu

Relačná databáza rozdeľuje dáta do logicky súvisiacich relácií. V praxi sa väzby medzi tabuľkami najčastejšie vyjadrujú pomocou kľúčov.

Primárny kľúč (PRIMARY KEY)

Primárny kľúč jednoznačne identifikuje riadok. Môže byť prirodzený (napr. stabilný jednoznačný identifikátor z domény) alebo umelý/surrogate (napr. generované číslo alebo UUID).

Cudzí kľúč (FOREIGN KEY)

Cudzí kľúč odkazuje na kandidátny alebo primárny kľúč inej relácie a umožňuje DBMS kontrolovať referenčnú integritu.

```
CUSTOMER(customer_id PK, name, email)
ORDER(order_id PK, customer_id FK → CUSTOMER.customer_id, created_at)
```

Obmedzenia a integrita

Databáza môže okrem kľúčov vynucovať UNIQUE, NOT NULL, CHECK a ďalšie obmedzenia. Integrita údajov tak nie je ponechaná iba na aplikačný kód.

8. Normalizácia a prečo vznikla

Normalizácia je súbor zásad, ktorých cieľom je navrhnuť relácie tak, aby sa obmedzila zbytočná redundancia a anomálie pri vkladaní, aktualizácii a mazaní údajov. Pri výučbe sa najčastejšie pracuje s prvou, druhou a treťou normálnou formou; v pokročilejšom návrhu aj s BCNF a ďalšími formami.

Forma	Zjednodušená myšlienka
1NF	Hodnoty atribútov sú atomické v zmysle zvoleného relačného návrhu; neopakujeme skupiny stĺpcov.
2NF	Ne-kľúčové atribúty sú plne závislé od celého kandidátneho kľúča, nie iba od jeho časti.
3NF	Ne-kľúčové atribúty nemajú nežiaducu tranzitívnu závislosť od kľúča cez iný ne-kľúčový atribút.
BCNF	Každý determinant netriviálnej funkčnej závislosti je superkľúč.

Normalizácia nie je dogma. Pri analytických systémoch, cache alebo distribuovaných databázach sa údaje môžu zámerne denormalizovať, ak to zodpovedá dominantným prístupovým vzorom. Ide o vedomý kompromis medzi integritou, jednoduchosťou aktualizácií a výkonom čítania.

9. Transakcie a ACID

Transakcia združuje jednu alebo viac operácií do logického celku. Klasické databázové systémy opisujú dôležité transakčné vlastnosti skratkou ACID.

Vlastnosť	Význam
Atomicity – atomicita	Transakcia sa z pohľadu výsledku uskutoční ako celok alebo sa jej zmeny vrátia späť.
Consistency – konzistencia	Úspešná transakcia zachová všetky pravidlá a invarianty, ktoré systém definuje.
Isolation – izolácia	Súbežné transakcie sa navzájom neovplyvňujú spôsobom, ktorý porušuje garantovanú úroveň izolácie.
Durability – trvácnosť	Po potvrdení transakcie majú výsledky prežiť následné zlyhanie podľa garancií DBMS.

```
BEGIN;
UPDATE account SET balance = balance - 100 WHERE id = 1;
UPDATE account SET balance = balance + 100 WHERE id = 2;
COMMIT;
```

ACID nie je výlučnou vlastnosťou „SQL sveta“. Moderné NoSQL systémy môžu podporovať rôzne formy transakcií a silnej konzistencie. Rozsah a náklady týchto garancií však závisia od konkrétneho systému.

10. Prečo vznikla nová vlna nerelačných databáz

S rozmachom webových služieb, vyhľadávačov, sociálnych sietí, mobilných aplikácií a cloudovej infraštruktúry sa objavili workloady s enormným objemom dát, vysokou rýchlosťou zápisu, globálnou distribúciou a rýchlo sa meniacimi dátovými štruktúrami. Významným výskumným míľnikom bol Google Bigtable (publikovaný v roku 2006), navrhnutý na petabajty dát rozložené cez tisíce komoditných serverov.

NoSQL preto nie je jedna databáza ani jeden presne definovaný model. Je to zastrešujúce označenie rôznych nerelačných databázových rodín. MongoDB vo svojom prehľade uvádza štyri hlavné rodiny: dokumentové, key-value, wide-column/column-family a grafové databázy.

Výraz „NoSQL“ je najlepšie chápať ako rodinu alternatívnych dátových modelov a architektúr, nie ako tvrdenie „SQL sa nesmie používať“.

11. Hlavné rodiny NoSQL databáz

11.1 Dokumentové databázy

Dokumentové databázy ukladajú záznamy ako dokumenty, typicky v JSON/BSON-podobnej štruktúre. Dokument môže obsahovať vnorené objekty a polia, takže jedna aplikačná entita sa často uloží ako jeden agregát. MongoDB zároveň upozorňuje, že označenie „schemaless“ je zavádzajúce: dokumentová databáza má dátový model, len býva flexibilnejší a jeho zmena môže byť menej nákladná.

```
{
  "customer_id": 125,
  "name": "Peter Novak",
  "address": {"city": "Nitra", "country": "Slovakia"},
  "orders": [{"product": "Notebook", "price": 1200}]
}
```

Typické využitie: produktové katalógy, content management, profily používateľov, aplikačné dáta s prirodzene vnorenou štruktúrou.

11.2 Key-value databázy

Key-value model pracuje s jedinečným kľúčom a hodnotou. Je vhodný vtedy, keď aplikácia pozná kľúč a potrebuje veľmi efektívne načítať alebo zmeniť zodpovedajúcu hodnotu. Redis dokumentácia opisuje uložené objekty práve ako páry kľúč–hodnota, pričom dnešný Redis nad týmto základom podporuje bohaté dátové typy.

```
session:ab91f2 → {user_id: 42, expires: ...}
cart:customer:42 → {...}
```

Typické využitie: cache, sessions, nákupné košíky, počítadlá, fronty, real-time stav.

11.3 Wide-column / column-family databázy

Wide-column databázy organizujú dáta okolo riadkových kľúčov a rodín stĺpcov, pričom jednotlivé riadky nemusia mať identickú sadu stĺpcov. Ich návrh je často úzko spojený s konkrétnymi prístupovými vzormi a horizontálnym rozdelením dát. Google Bigtable je historicky významný predstaviteľ tejto architektonickej línie; Apache Cassandra je známym open-source systémom ovplyvneným podobnými distribuovanými princípmi.

Typické využitie: masívne distribuované datasety, časové rady, telemetria, udalosti, workloady s vysokým objemom zápisov.

11.4 Grafové databázy

Grafové databázy modelujú dáta pomocou uzlov, hrán/vzťahov a vlastností. Výhodou je prirodzená práca s problémami, pri ktorých sú vzťahy rovnako alebo viac dôležité než samotné

entity. Neo4j uvádza medzi typickými scenármi fraud detection, knowledge graphs, odporúčania, identity & access management a správu komplexných sietí.

```
(Peter)-[:FRIEND_OF]->(Jana)
(Peter)-[:WORKS_AT]->(FirmaX)
(Jana)-[:LIKES]->(ProduktY)
```

12. Relačné a nerelačné databázy: porovnanie bez mýtov

Oblasť	Relačný prístup	NoSQL prístupy
Dátový model	Relácie/tabuľky, kľúče, integritné obmedzenia	Dokumenty, key-value, wide-column, grafy – podľa rodiny
Schéma	Zvyčajne explicitná a centrálne definovaná	Často flexibilnejšia; stále existuje logická alebo aplikačná schéma
Dotazy	SQL a relačné operácie vrátane JOIN	Rôzne API a jazyky optimalizované pre konkrétny model
Integrita	Silná podpora constraintov a referenčnej integrity	Rozsah garancií závisí od konkrétneho systému
Škálovanie	Možné vertikálne aj horizontálne; moderné RDBMS môžu byť distribuované	Mnohé systémy boli od začiatku navrhnuté pre scale-out
Typické silné stránky	Komplexné transakcie, integrita, ad-hoc dotazy, stabilný model	Flexibilita, špecializované prístupy, distribuované workloady, vzťahové grafy

13. Normalizácia verzus denormalizácia

Relačný návrh často rozdeľuje entity do viacerých tabuliek a pri čítaní ich skladá pomocou JOIN. V dokumentových či wide-column databázach sa často používa agregátový návrh: údaje, ktoré sa čítajú spolu, sa zámerne ukladajú spolu – aj za cenu duplicity.

Relácia: CUSTOMER ← ORDER ← ORDER_ITEM → PRODUCT

Dokument objednávky:

```
{ order_id: 158, customer: {...}, items: [{product_name: ..., price: ...}] }
```

Denormalizácia môže znížiť počet distribuovaných lookupov a zjednodušiť rýchle čítanie. Zároveň však komplikuje konzistentné aktualizovanie duplicitných hodnôt. V NoSQL návrhu sa preto často začína otázkou: „Aké dotazy a prístupové vzory musí aplikácia podporovať?“

14. Distribuované databázy: replikácia, partitioning a CAP

Horizontálne škálovanie rozkladá dáta alebo požiadavky medzi viac uzlov. Dva základné mechanizmy sú replikácia – vytváranie kópií dát – a partitioning/sharding – rozdelenie dát medzi uzly. V praxi sa často kombinujú.

CAP teoréma

CAP teoréma sa týka distribuovaných dátových systémov pri sieťovom rozdelení (partition). V takom stave nemožno súčasne garantovať lineárnu/atomickú konzistenciu a úplnú dostupnosť pre všetky požiadavky. Formálne výsledky Gilberta a Lynchovej nadväzovali na Brewerovu CAP domnienku a ukázali tento limit v asynchrónnych sieťových modeloch.

Písmeno	Význam v CAP
C – Consistency	Operácie sa javia, akoby existovala jedna aktuálna kópia dát (v klasickej formulácii atomic/linearizable consistency).
A – Availability	Každá požiadavka smerujúca na nezhavovaný uzol dostane odpoveď.
P – Partition tolerance	Systém musí fungovať aj keď sa správy medzi skupinami uzlov strácajú alebo sú oneskorené tak, že vznikne sieťové rozdelenie.

Zjednodušenie „vyber si dve z troch“ je nepresné. V reálnom distribuovanom systéme sa partition nedá vždy vylúčiť; rozhodujúci kompromis sa prejaví práve počas partition – systém musí obmedziť buď dostupnosť, alebo konzistenčnú garanciu.

Eventual consistency

Pri eventual consistency môžu repliky určitý čas poskytovať rozdielne hodnoty, ale ak prestanú prichádzať nové zmeny a systém funguje správne, očakáva sa ich konvergencia. Takýto model môže byť vhodný pre niektoré globálne distribuované workloady, no nie je univerzálne prijateľný. Voľba konzistenčného modelu musí vychádzať z obchodných invariantov a následkov dočasne neaktuálnych dát.

15. Ako vybrať vhodný databázový model

Výber databázy by nemal začínať otázkou „SQL alebo NoSQL?“, ale analýzou dát a workloadu.

1. Aké entity, vzťahy a agregáty v doméne existujú?
2. Aké sú najčastejšie dotazy a zápisy?
3. Ktoré operácie musia byť transakčné a aké invarianty nesmú byť porušené?
4. Aká je požadovaná latencia, priepustnosť a dostupnosť?
5. Ako rýchlo rastie objem dát a aká je geografická distribúcia používateľov?
6. Ako často sa mení dátová štruktúra?
7. Aké sú požiadavky na analytiku, full-text, grafové alebo vektorové vyhľadávanie?
8. Aké prevádzkové znalosti a nástroje má tím?

Situácia	Často vhodný východiskový prístup
Finančné transakcie, objednávky, účtovníctvo	Relačný DBMS so silnými transakčnými a integritnými garanciami
Produktový katalóg s rôznorodými atribútmi	Dokumentová databáza alebo relačná DB s JSON podporou
Cache a sessions	Key-value / in-memory datastore
Sociálny graf, fraud ring, závislosti	Grafová databáza

Masívny event/telemetry workload	Wide-column, time-series alebo iný distribuovaný špecializovaný systém
Komplexné ad-hoc reporty nad konzistentnými podnikovými dátami	Relačný analytický/warehouse systém podľa charakteru workloadu

16. Priebežný príklad: vývoj e-shopu TechShop

Nasledujúci príklad prepája predchádzajúcu teóriu do jedného scenára. Cieľom nie je tvrdiť, že e-shop musí používať presne tieto produkty, ale ukázať, ako z požiadaviek a prístupových vzorov prirodzene vzniká voľba dátového modelu. TechShop začína ako malá aplikácia a postupne rastie. Pri každom kroku sa objaví nový problém, ktorý možno riešiť vhodnejšou databázovou technológiou.

16.1 Fáza 1 – e-shop uložený v súboroch

Na začiatku má TechShop iba niekoľko zákazníkov a produktov. Programátor preto môže údaje uložiť do CSV súborov. Takéto riešenie zodpovedá súborovo orientovanému spracovaniu opísanému v kapitole 2.

```
customers.csv
1;Peter Novák;peter@email.sk;Nitra
2;Jana Malá;jana@email.sk;Bratislava
```

```
products.csv
101;Notebook Lenovo;899;12
102;Logitech Mouse;39;45
```

```
orders.csv
5001;1;101;1
```

Kým je systém malý, riešenie je zrozumiteľné. Problém sa objaví pri otázke: „Aký produkt si objednal Peter a na akú adresu ho máme poslať?“ Aplikácia musí sama načítať objednávku, podľa identifikátora vyhľadať zákazníka a následne produkt. Ešte vážnejší problém nastane, ak dvaja zákazníci súčasne kupujú posledný kus produktu. Súbežný prístup, integrita a obnova po chybe zostávajú na aplikačnom kóde.

Prepojenie s teóriou: tento krok ukazuje problémy súborových systémov – dátovú závislosť, duplicitu logiky, súbežnosť a potrebu centrálne riadenej integrity. Práve tieto problémy motivovali vznik DBMS.

16.2 Fáza 2 – relačná databáza pre jadro e-shopu

Po raste TechShopu presunieme jadro systému do relačného DBMS, napríklad PostgreSQL. Zákazníci, objednávky, položky objednávok a produkty majú jasné vzťahy a integritné pravidlá, preto je relačný model prirodzenou voľbou.

```
CUSTOMER(customer_id PK, name, email, city)
PRODUCT(product_id PK, name, price, stock)
ORDERS(order_id PK, customer_id FK, created_at)
ORDER_ITEM(order_id FK, product_id FK, quantity, price)
```

```
CUSTOMER 1 — N ORDERS 1 — N ORDER_ITEM N — 1 PRODUCT
```

Dotaz na objednávky konkrétneho zákazníka možno formulovať deklaratívne pomocou JOIN operácií:

```
SELECT c.name, o.order_id, p.name, oi.quantity, oi.price
FROM customer c
JOIN orders o ON o.customer_id = c.customer_id
JOIN order_item oi ON oi.order_id = o.order_id
JOIN product p ON p.product_id = oi.product_id
WHERE c.name = 'Peter Novák';
```

Pri dokončení nákupu zároveň potrebujeme vytvoriť objednávku, znížiť skladové množstvo a zaznamenať súvisiace zmeny ako jeden logický celok. Tu sa prakticky prejaví význam transakcií a vlastností ACID z kapitoly 9: nechceme, aby po chybe zostala vytvorená objednávka bez zodpovedajúcej zmeny skladu alebo naopak.

Prepojenie s teóriou: primárne a cudzie kľúče vynucujú identitu a referenčnú integritu; normalizácia obmedzuje zbytočné duplicity; SQL oddeľuje otázku „čo chceme“ od fyzického spôsobu vykonania; transakcie chránia kritické invarianty.

16.3 Fáza 3 – heterogénny produktový katalóg a dokumentový model

TechShop neskôr začne predávať veľmi rozdielne kategórie. Notebook má CPU, RAM a kapacitu SSD, televízor uhlopriečku, typ panela a HDR, zatiaľ čo oblečenie má veľkosť, farbu a materiál. V relačnom modeli sa to dá navrhnuť viacerými korektnými spôsobmi, no pri rýchlo sa meniacej a heterogénnej sade atribútov môže byť dokumentový model pohodlnejší.

```
{
  "_id": 101,
  "name": "Lenovo ThinkPad",
  "category": "notebook",
  "price": 899,
  "specifications": {"cpu": "Intel Core i7", "ram": "16 GB", "ssd": "512 GB"}
}

{
  "_id": 501,
  "name": "Tričko",
  "category": "clothing",
  "price": 25,
  "specifications": {"sizes": ["S", "M", "L"], "color": "black", "material": "cotton"}
}
```

Dokumentová databáza, napríklad MongoDB, umožňuje reprezentovať každý produkt ako agregát s vlastnou vnorenou štruktúrou. Neznamená to však, že relačná databáza „nevie“ ukladať heterogénne dáta ani že celý e-shop treba presunúť do MongoDB. Ide o voľbu modelu podľa dominantných operácií a spôsobu, akým aplikácia s katalógom pracuje.

Prepojenie s teóriou: ide o praktický príklad flexibilnejšej schémy a agregátového/denormalizovaného návrhu z kapitol 11 a 13. „Schema-free“ neznamená „bez štruktúry“ – štruktúra dokumentov stále existuje a aplikácia ju musí spravovať.

16.4 Fáza 4 – nákupný košík, sessions a key-value prístup

S rastom návštevnosti vzniká ďalší workload: veľké množstvo krátkodobých operácií nad nákupnými košíkmi a používateľskými sessions. Aplikácia často pozná presný identifikátor používateľa a potrebuje veľmi rýchlo načítať alebo zmeniť jeho stav.

```
cart:user:125 → {101: 2, 102: 1}
session:ab91f2 → {user_id: 125, expires: ...}
```

Takýto prístup dobre zodpovedá key-value modelu. Redis môže slúžiť ako rýchla vrstva pre cache, sessions alebo dočasný stav košíka. Po potvrdení nákupu sa však trvalá objednávka uloží do transakčného jadra systému. Konkrétna produkčná architektúra musí riešiť perzistenciu, expirácie a scenáre výpadku podľa požadovaných garancií.

```
Redis / dočasný stav košíka
    | CHECKOUT
    ▼
PostgreSQL / trvalá objednávka a transakčné údaje
```

Prepojenie s teóriou: databáza sa vyberá podľa prístupového vzoru. Ak dominantná operácia znie „poznám kľúč, načítaj alebo zmeň hodnotu“, key-value model môže byť jednoduchší než všeobecný relačný dotazovací model.

16.5 Fáza 5 – odporúčania a graf vzťahov

Marketing chce odporúčať kompatibilné produkty a analyzovať vzťahy typu „zákazníci, ktorí kúpili tento notebook, kupovali aj...“. V tejto úlohe prestávajú byť najdôležitejšie samotné riadky či dokumenty a do popredia sa dostávajú prepojenia medzi zákazníkmi, produktmi a kategóriami.

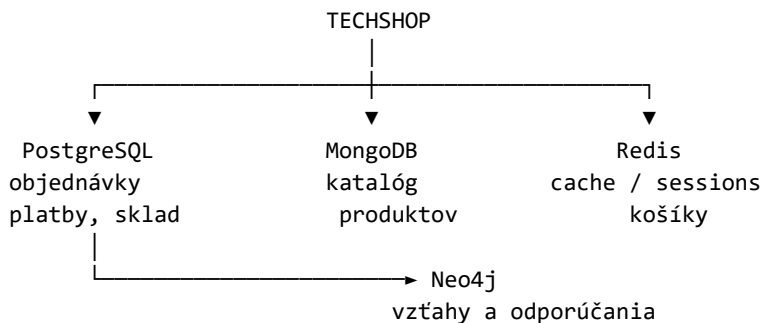
```
(Peter) - [:BOUGHT] -> (Notebook)
(Jana) - [:BOUGHT] -> (Notebook)
(Notebook) - [:COMPATIBLE_WITH] -> (Dock)
(Dock) - [:BOUGHT_WITH] -> (Mouse)
```

Grafová databáza, napríklad Neo4j, reprezentuje entity ako uzly a ich vzťahy ako hrany. Dotazy založené na prechádzaní siete vzťahov tak môžu byť prirodzenejšie vyjadrené grafovým modelom. Relačná databáza vie grafové vzťahy modelovať tiež; grafová databáza je výhodná najmä vtedy, keď je traversal jadrom workloadu.

Prepojenie s teóriou: tento krok ukazuje, že dátový model má zodpovedať otázkam, ktoré nad dátami kladieme. Pri grafoch sú vzťahy prvotriednou súčasťou modelu, nie iba odvodeným výsledkom JOIN operácií.

16.6 Výsledok – polyglotná architektúra TechShopu

Po všetkých krokoch už otázka neznie „SQL alebo NoSQL?“. TechShop môže používať viac technológií, pričom každá má jasne definovanú zodpovednosť. Takáto architektúra je príkladom polyglot persistence.



Dôležitá je aj opačná lekcia: viac databáz nie je automaticky lepšie. Každá ďalšia technológia prináša monitoring, zálohovanie, bezpečnosť, synchronizáciu, prevádzkové znalosti a riziko

nekonzistencie medzi systémami. Polyglot persistence má zmysel až vtedy, keď prínos špecializovaného modelu preváži túto zložitosť.

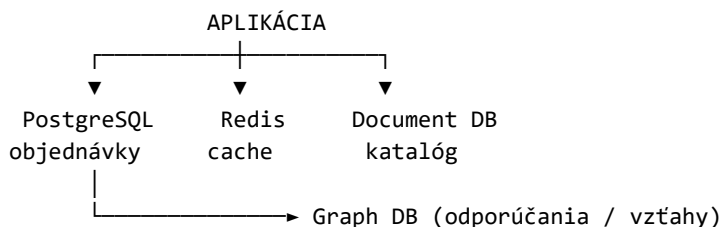
16.7 Čo si z príkladu odniesť

- Súbory ukázali, prečo potrebujeme DBMS a centrálné riadenie dát.
- Relačný model vyriešil jadro s jasnými vzťahmi, integritou, SQL a transakciami.
- Dokumentový model ukázal výhodu agregátov a flexibilnej štruktúry pri heterogénnom katalógu.
- Key-value model ukázal silu jednoduchého prístupu podľa kľúča pri dočasnóm stave a cache.
- Grafový model ukázal výhodu v situácii, keď sa problém sústreďuje na vzťahy a prechádzanie grafu.
- Výsledkom nie je víťaz jednej technológie, ale rozhodovanie podľa dátového modelu, workloadu a požadovaných garancií.

Pedagogická pointa: namiesto memorovania definícií sledujeme reťazec „problém → požiadavka → vhodný model → nový kompromis“. Tak sa historický vývoj databáz prepája s praktickým návrhom dnešných systémov.

17. Polyglot persistence: viac databáz v jednej aplikácii

Moderná aplikácia nemusí mať jedinú databázu. Pojem polyglot persistence označuje využitie viacerých databázových technológií podľa ich silných stránok. E-shop môže používať relačnú databázu pre objednávky, Redis pre cache a sessions, dokumentový store pre katalóg a grafovú databázu pre odporúčania alebo analýzu vzťahov. Priebežný príklad TechShopu v predchádzajúcej časti ukazuje tento princíp na konkrétnych prístupových vzoroch.



Nevýhodou polyglot persistence je vyššia prevádzková a vývojová zložitosť: viac technológií znamená viac monitoringu, zálohovania, bezpečnostných modelov, klientskych knižníc a problémov s konzistenciou medzi systémami. Preto sa ďalšia databáza pridáva vtedy, keď jej prínos preváži túto cenu.

18. Súčasné trendy: cloud, distributed SQL, multimodel a vektory

Vývoj databáz sa dnes neuberá jedným smerom. Viaceré línie sa navzájom približujú a preberajú si vlastnosti.

- Cloud-managed databázy: poskytovateľ automatizuje časť prevádzky, záloh, patchovania a škálovania.
- Distributed SQL / NewSQL: systémy sa snažia kombinovať SQL a relačný model s horizontálnou distribúciou a silnými transakčnými garanciami.

- Multi-model databázy: jeden produkt podporuje viac reprezentácií alebo vyhľadávacích mechanizmov, napr. relačné dáta, JSON, grafové či vektorové operácie.
- In-memory spracovanie: pracovná množina dát sa drží v pamäti s cieľom nízkej latencie; Redis je príkladom dátového servera s bohatými natívnymi dátovými typmi.
- Vektorové vyhľadávanie: embedding reprezentuje objekt ako číselný vektor a databáza hľadá najbližšie vektory podľa zvolenej metriky. Tento mechanizmus je dôležitý pre semantic search a RAG systémy.
- HTAP a real-time analytics: snaha zmenšovať hranicu medzi transakčným a analytickým spracovaním.

Hranica medzi „SQL“ a „NoSQL“ sa preto rozmazáva. Relačné DBMS podporujú JSON, full-text a vektorové rozšírenia; NoSQL databázy pridávajú transakcie, indexy, validačné pravidlá a expresívnejšie dotazovanie.

19. Historická časová os

Obdobie	Vývoj
1950s–1960s	Súborovo orientované spracovanie dát
1960s	Hierarchické a sieťové DBMS
1970	Codd publikuje relačný model [1]
1973+	IBM System R; vývoj SQL a nákladovej optimalizácie [2][3]
late 1970s–1980s	Komercializácia relačných DBMS; Oracle, IBM DB2 a ďalšie
1986+	Berkeley POSTGRES, neskôr PostgreSQL [4]
1990s	Web, klient-server architektúry, open-source RDBMS
2000s	Internetová škála, Bigtable a veľké distribuované dátové systémy [5]
late 2000s–2010s	Široká adopcia NoSQL; document, key-value, wide-column, graph
2010s–2020s	Cloud DB, distributed SQL, multimodel, serverless
2020s	Vektorové vyhľadávanie a databázové funkcie pre AI aplikácie

20. Zhrnutie

História databázových systémov nie je jednoduchý rad náhrad, v ktorom nová technológia zruší predchádzajúcu. Ide o postupné rozširovanie priestoru možných riešení. Súborové spracovanie ukázalo potrebu centrálnej správy dát. Hierarchické a sieťové systémy priniesli organizovanejšie modely, no ostávali navigačne zložité. Coddov relačný model oddelil logický pohľad od fyzickej navigácie a spolu so SQL a optimalizátormi vytvoril základ moderných podnikových databáz.

NoSQL databázy vznikli ako odpoveď na nové dátové modely a distribuované workloady, nie ako univerzálne „lepšie SQL“. Dokumentové, key-value, wide-column a grafové databázy riešia

odlišné typy problémov. Dnešná prax preto často kombinuje viac technológií a hranice medzi kategóriami sa postupne prelínajú.

Použitá a odporúčaná literatúra

- [1] CODD, E. F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, 1970, 13(6), 377–387.
- [2] IBM. The relational database / Edgar F. Codd – historical materials on the relational model and System R. <https://www.ibm.com/history/relational-database> ; <https://www.ibm.com/history/edgar-codd>
- [3] IBM. What is a Relational Database? Historical overview of Codd, System R and DB2. <https://www.ibm.com/think/topics/relational-databases>
- [4] PostgreSQL Global Development Group. A Brief History of PostgreSQL. Current PostgreSQL documentation. <https://www.postgresql.org/docs/current/history.html>
- [5] CHANG, F. et al. Bigtable: A Distributed Storage System for Structured Data. OSDI 2006. Google Research. <https://research.google/pubs/bigtable-a-distributed-storage-system-for-structured-data/>
- [6] MongoDB. Types of Databases – overview of relational and four major NoSQL families. <https://www.mongodb.com/resources/basics/databases/types>
- [7] MongoDB. NoSQL Database Design & Data Modeling. <https://www.mongodb.com/resources/basics/databases/nosql-explained/data-modeling>
- [8] Redis. Redis data types; Keys and values. Official documentation. <https://redis.io/docs/latest/develop/data-types/> ; <https://redis.io/docs/latest/develop/using-commands/keyspace/>
- [9] Neo4j. Graph Database Use Cases; What is a graph database. <https://neo4j.com/use-cases/> ; <https://neo4j.com/docs/getting-started/graph-database/>
- [10] GILBERT, S.; LYNCH, N. Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. SIGACT News, 2002, 33(2), 51–59.