33. NoSQL databáze (porovnání relačních a NoSQL; CAP věta a ACID/BASE principy; typy NoSQL databází; dotazování v NoSQL databázích; agregace dat pomocí Map-Reduce a agregační pipeline).

Nejpoužívanějším typem databází jsou relační databáze, na popularitě však získávají i nerelační databáze, tzv. NoSQL databáze. Zopakujme nyní krátce klíčové vlastnosti relačních databází:

- data jsou organizována do tabulek s pevným schématem, sloupce mají přesně daný datový typ, tabulky jsou relace nad doménami sloupců tabulky
- záznamy v tabulkách mohou být propojeny cizími klíči
- databázi typicky navrhujeme tak, aby splňovala normální formy pro redukci redundance
- dotazování s pomocí jazyka SQL s pomocí příkazu SELECT, jeho sémantika vychází z relační algebry
- DBMS zaručuje ACID:
  - Atomicity – transakce buď celá proběhne, nebo neproběhne vůbec
  - Consistency – Nedojde k porušení integritních omezení daných primárními/cizími klíči
  - Isolation – pokud více transakcí probíhá současně, nesmí se vzájemně ovlivnit
  - Durability – jakmile transakce proběhne, je její výsledek trvale uložen

V poslední době vzhledem k množství dat začíná být kladen vyšší důraz na škálovatelnost databáze (možnost data distributovat apod.). Zároveň pak často v databázích chceme ukládat složitější nestrukturované data, případně speciální datové typy (klíč-hodnota apod.). Schémata databází také nebývají pevná a výhodou je tedy možnost iterativního vývoje. Relační databáze v klasickém pojetí tyto požadavky většinou mají poměrně problém splnit. → proto NoSQL databáze

Pro distribuované systémy platí tzv. CAP teorém:

- Consistency – každý klient vidí stejná data
- Availability – každý požadavek je obsloužen, vždy je možnost zapsat nebo číst data
- Partition Tolerance – databáze je funkční navzdory částečným výpadkům a chybám v síti

Dle CAP teorému je možné v rámci distribuovaného systému uspokojit maximálně 2 z těchto 3 vlastností.

NoSQL databáze

Jak bylo naznačeno, NoSQL jsou nerelační databáze, podporují tedy ukládání nerelačních dat, např. datových struktur typu klíč-hodnota, dokumentů, grafů atd. Zároveň jsou typicky stavěné pro distribuovanou architekturu – platí zde tedy CAP teorém, který většina NoSQL databází řeší omezením konzistence, tzv. přístup BASE:

- Basically Available – aplikace funguje víceméně pořád (požadavek Availability)
- Soft-state – nemusí být konzistentní vždy
- Eventual consistency – nakonec se ale dostane do nějakého konzistentního stavu

Kdybychom měli srovnat BASE s ACID známý z relačních databází, tak u BASE nemáme zaručenu takovou konzistenci (mohou být krátkodobě porušena některá integritní omezení), data jsou dodávána agresivně, jak rychle to jen jde (oproti tomu ACID je konzervativní, aby nebyla porušena konzistence). NoSQL je tedy vhodnější, pokud vyžadujeme distribuované uložení nebo zpracování a rychlost na úkor konzistence. Pro klasické informační systémy je však stále lepší relační databáze.

Databáze klíč-hodnota

(např. Oracle NoSQL) - velká distribuovaná hashovací tabulka

Vyhledávání v těchto databázích probíhá pomocí klíče přes hashovací tabulky, což je velmi efektivní. Hodnota je typicky neinterpretovaná, pro databázi nemá žádnou sémantiku. Vyhledávání na základě hodnoty, případně její části, je tedy brutálně pomalé. Příkladem je Oracle NoSQL:

- klíč v této databázi je dvousložkový (major a minor část). Záznamy se stejným major klíčem musí být uloženy na stejném uzlu
- u záznamů se stejným major klíčem je podporván ACID (protože tyto záznamy jsou na stejném uzlu)

Dokumentové databáze

(např. MongoDB)

Víceméně jako klíč-hodnota (klíč je nějaké ID dokumentu, hodnota pak samotný dokument), ale hodnota je strukturovaná např. jako XML/JSON dokument a je možné dotazovat se i pomocí hodnoty. Dokumenty je možné vnořovat, ale můžeme využívat i odkazů přes ID (podobný princip jako cizí klíče). Vnořováním je typicky realizován vztah 1:N (vnořený dokument pak neoddělitelně patří nadřazenému objektu), zatímco odkazováním vztahy M:N, případně 1:N pokud nechceme, aby nějaký dokument byl vnořen (např. se jedná o samostatnou entitu reálného světa). Příkladem je mongoDB, které přirozeně podporuje práci s JSON/slovníkovými daty. Klíč se realizuje atributem dokumentu _id:

ACID vs BASE

- silná konzistence
- izolovanost
- orientace na commit
- vnořené transakce
- dostupnost?
- konzervativní (pesimistické)
- složitá evoluce schématu

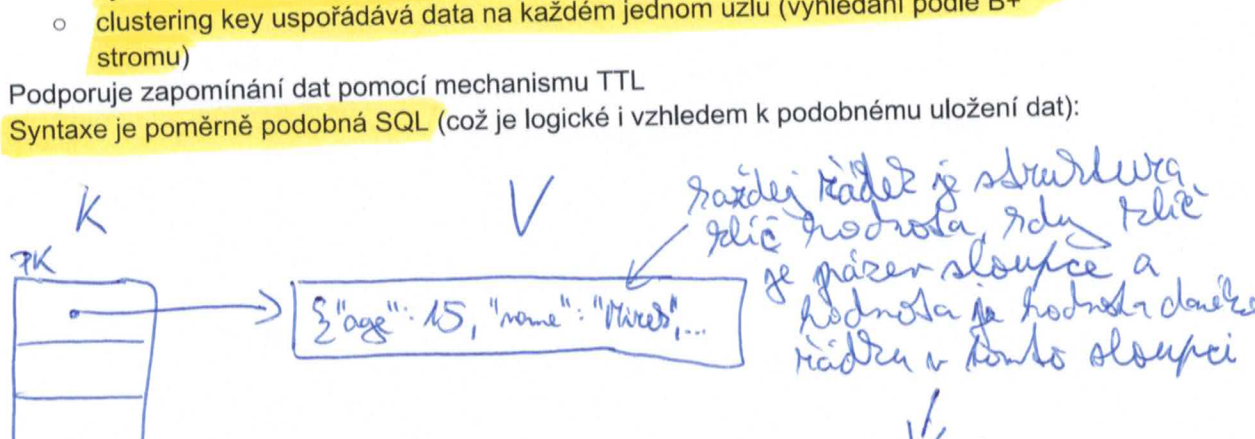
- slabá konzistence
- dostupnost na prvním místě
- přibližné odpovědi jsou OK
- jednodušší, rychlejší
- dodávka dat "jak to jim přijde"
- agresivní (optimistické)
- jednodušší evoluce

```
db.article.insert({
  "name" : "My Article",
  "publish date" : new Date("2013-10-15"),
  "comment" : [],
  "tag" : [ "adventure", "fiction" ]
})

db.article.find({"tag" : "adventure"}).pretty()
{
  "_id" : ObjectID("525c7ed0cc9374393401f5fd"),
  "name" : "My Article",
  "publish date" : ISODate("2013-10-15"),
  "comment" : [],
  "tag" : [ "adventure", "fiction" ]
}

db.article.update(
  {"_id" : new ObjectID("525c7ed0cc9374393401f5fd")},
  {$push : { comment : { name : "Alice", comment : "Awesome post!" } } }
)
```

Sloupcové NoSQL databáze (databáze s širokými sloupci) (např. Cassandra, BigTable)

Databáze se skládá z "tabulek" podobně jako u relačních databází, ale v každém řádku je struktura klíč-hodnota, kdy klíčem je název sloupce a hodnotou hodnota ve sloupci. Každý řádek tedy může mít jiné sloupce. Příkladem takové databáze je Cassandra nebo BigTable.
Podívejme se nyní na vlastnosti databáze Cassandra:

- klíč se skládá ze dvou částí – partition a clustering key:
  - partition key rozděluje data v tabulce mezi uzly (více serverů s databází) – vyhledání uzlu s hledanými daty pomocí hash tabulky
  - clustering key uspořádává data na každém jednom uzlu (vyhledání podle B+ stromu)
- Podporuje zapomínání dat pomocí mechanismu TTL
- Syntaxe je poměrně podobná SQL (což je logické i vzhledem k podobnému uložení dat):



```
CREATE TABLE IF NOT EXISTS temps (
        building text,
        room text,
        time timestamp,
        temperature float,
        PRIMARY KEY (
                (building, room),
                time
        )
);

INSERT INTO temps (building, room, time, temperature)
VALUES ('FIT VUT', 'N203', toTimestamp(now()), 23)
USING TTL 3600;

SELECT * FROM temps
WHERE building = 'FIT VUT' AND room = 'N205'
        AND time > '2019-10-23 12:00:00';
```

Grafové databáze (např. Neo4J)

V grafových databázích chceme uchovávat uzly, vlastnosti uzlů a hrany spojující uzly. Je tak možné reprezentovat sítě a topologie. Speciální kategorií grafových databází jsou tzv. RDF databáze: – modelování ontologií

- RDF je orientovaný ohodnocený graf, kde hrana začíná v subjektu, je ohodnocena predikátem a končí v předmětu (např. Fifinas == subjekt, je studentem == predikát, FIT == předmět).
- Subjekt a predikát je reprezentován URI, předmět pak hodnotou nebo URI
- Nad RDF grafy je možné dokazovat fakta
- K dotazování se využívá jazyk SPARQL

Příkladem grafové databáze je Neo4j, příkladem RDF databáze je AllegroGraph. Ukázka práce s Neo4j:

```
MATCH (m:Movie) WHERE m.title="The Matrix" RETURN m;

MATCH (m:Movie)<-[:ACTED_IN]-(a:Person)
WHERE lower(m.title) CONTAINS "matrix"
RETURN m.title as movie, collect(a.name) as cast;

MATCH
    (m1:Movie)-[r1:ACTED_IN]-(p1:Person),
    (p2:Person)-[r2:ACTED_IN]-(m2:Movie)
WHERE
     p1=p2 AND
     m1.title = "The Matrix" AND
     m2.released > m1.released
RETURN p1.name AS p, m2.title AS m;
```

Databáze časových řad (např. InfluxDB)

Tzv. time series databáze, uchovávají časové řady měření různých hodnot. Časová řada je chronologická posloupnost měření. Každá časová řada je dána svým názvem a hodnotami tagů, ty popisují onu časovou řadu. Jednotlivé časové řady se pak skládají z tzv. polí (field), která mají název a časovou značku. Klíčem je v rámci časové řady časová značka, hodnotou pak naměřená hodnota. Databáze také typicky podporuje seskupení po určitých časových intervalech. Typickým zástupcem je InfluxDB, ukázka práce s ní:

Nedá se:
- Časová řada je dána svým názvem a hodnotou tagů
- „line" protokol

```
SELECT * FROM h2o_feet ORDER BY time DESC LIMIT 1;

SELECT "water_level" FROM "h2o_feet"
WHERE "location" <> 'santa_monica'
AND (water_level < -0.59 OR water_level > 9.95);

SELECT MEAN("water_level"), MAX("water_level")
FROM "h2o_feet" GROUP BY "location";

SELECT COUNT("water_level") FROM "h2o_feet"
WHERE "location"='coyote_creek'
AND time >= '2015-08-18T00:00:00Z'
AND time <= '2015-08-18T00:30:00Z'
GROUP BY time(12m);
```

Dotazování v NoSQL databázích

NoSQL databáze jsou distribuované uložiště typu klíč hodnota:

- Oracle NoSQL: major a minor key
- Cassandra: partition a clustering key
- MongoDB: pole _id v každém dokumentu
- Neo4j: unikátní ID každého uzlu a hrany v grafu
- InfluxDB: název hodnoty, časová značka

Základem je vyhledávání a dotazování přes klíč. Vzhledem k distribuovanosti se typicky skládá ze dvou fází:

1. nalezení uzlu, na kterém jsou data uložena (např. partitioning key, major key)
2. nalezení dat na daném uzlu (např. clustering key, minor key)

Databázové systémy je typicky možné distribuovat dvěma způsoby, tzv. partitioning:

- vertikální – typicky podle schématu, například každá tabulka na jiném uzlu
- horizontální, tzv. sharding – rozdělení dat podle nějakého klíče mezi jednotlivé uzly (v relační databázi bychom si mohli představit rozdělení tabulky na řádky)

NoSQL databáze dělají horizontální partitioning podle hashe klíče. Výhodou shardingu je, že systém zvládá větší zátěž, protože je zátěž rozdělena. Zároveň uzly v jednotlivých shardech mohou mít replikovaná data (ochrana před ztrátou). Nevýhodou naopak je, že operace napříč shardy jsou drahé a přesuny dat mezi shardy jsou složité.

Dotazování na základě klíče je podporováno ve všech databázích, zatímco dotazování podle hodnoty pouze v některých specializovaných (například mongoDB). V rámci dotazování typicky

není podporována join operace, ta je nahrazena vnořováním datových struktur. Zásadním prvkem efektivního vyhledávání je implementace vhodných indexů, které dotazování urychlí.

MapReduce

Pro složitější výpočty se používá model Map-Reduce, který funguje na následujícím principu:

- Funkce Map vezme jednu dvojici klíč-hodnota a vrátí seznam dvojic klíč-hodnota (tuto funkci je možné aplikovat paralelně na celou kolekci). Typově tedy máme $Map(k_1, v_1) -> list(k_2, v_2)$
- Výstupy z jednotlivých funkcí Map jsou poté seskupeny dle klíče a každá skupina je předána funkci Reduce, která z několika hodnot připadajících danému klíči udělá jednu agregovanou hodnotu. Typově: $Reduce(k_2, list(v_2)) -> list(k_3, v_3)$

Například pokud chceme efektivně distribuovaně spočítat, kolikrát se které slovo vyskytuje v nějaké kolekci dokumentů, můžeme použít Map-Reduce následovně:

```
function map(String name, String document):
    // name: document name
    // document: document contents
    for each word w in document:
        emit (w, 1)

function reduce(String word, Iterator partialCounts):
    // word: a word
    // partialCounts: a list of aggregated partial counts
    sum = 0
    for each pc in partialCounts:
        sum += pc
    emit (word, sum)
```

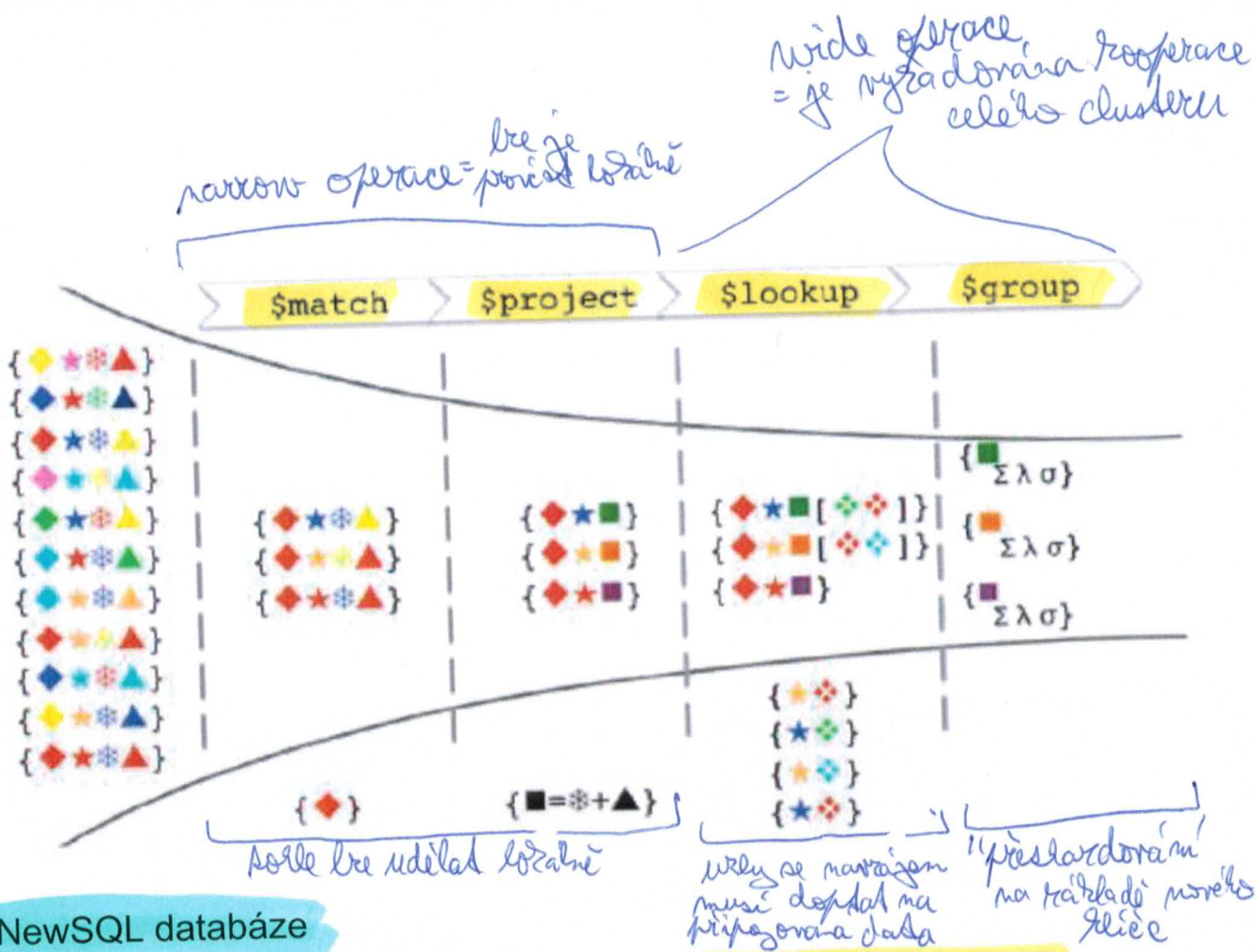
Funkce map může běžet paralelně, nad každým dokumentem zvlášť. Následně nám Map-Reduce framework seskupí slova do dvojic klíč-hodnota, kdy klíčem je slovo a hodnotou je seznam obsahující tolikrát číslici 1, kolikrát se slovo vyskytuje.

na pozadí je však také Map-reduce

Dotazování v MongoDB

V rámci MongoDB se pro vyhledávání a práci s dokumenty používají tzv. agregační pipeliny. Pipeline se skládá z více kroků, každý nějak modifikuje kolekci dokumentů. Výsledkem každého kroku je nová kolekce dokumentů. Základními příkazy agregačních pipeline jsou:

- $match – filtrování dokumentů
- $project – úprava dat v každém dokumentu z kolekce (například výpočet odvozeného atributu)
- $lookup – dodání dat z externího dokumentu na základě reference (cizí klíč)
- $group – seskupení dokumentů podle indexu
- $sort – seřazení dokumentů podle indexu



NewSQL databáze

Jedná se o kompromis mezi NoSQL a relačními databázemi. Cílem je dodat vlastnosti NoSQL (hlavně vysokou škálovatelnost) databázím splňujícím ACID. Typicky z CAP teorému obětujeme částečně dostupnost, abychom dosáhli vysoké konzistence. Příkladem takové databáze je MemSQL nebo CockroachDB.

Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele Fifinas.

- obětují Availability (mají CP)
- splňují ACID
- CockroachDB (backward kompatibilita s Postgresem)
- NewSQL databáze - databáze, co splňují ACID, jsou SQL a přitom jsou distribuované