Pregled i usporedba NewSQL sustava za upravljanje baza podataka

Marjanović, Anamaria

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Anamaria Marjanović

Pregled i usporedba NewSQL sustava za upravljanje baza podataka

Završni rad

Mentor: doc. dr. sc. Danijela Jakšić

Rijeka, 15.09.2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 15.6.2020.

Zadatak za završni rad

Pristupnik: Anamaria Marjanović

Naziv završnog rada: Pregled i usporedba NewSQL sustava za upravljanje baza podataka

Naziv završnog rada na eng. jeziku: Overview and comparison of NewSQL database management systems

Sadržaj zadatka:

Osnovni cilj rada je opisati, testirati i usporediti dva odabrana NewSQL sustava za upravljanje bazama podataka koristeći vlastite primjere. Opisati i objasniti vrste modela podataka koji su podržani NewSQL paradigmom. Opisati i objasniti vrste baza podataka koje su podržane NewSQL paradigmom. Opisati prednosti i nedostatke korištenja NewSQL sustava, kao i razloge njihovog korištenja. Navesti neke primjere NewSQL sustava, kao i primjere industrija u kojima se koriste. Odabrati dva sustava u kojima će se kroz vlastite primjere (relacijskih i nerelacijskih baza podataka te pripadajućih upita i ostalih funkcionalnosti) provesti usporedba i analiza.

Mentor

doc.dr.sc. Danijela Jakšić

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet:

(potpis pristupnika)

Sažetak

Osnovni cilj rada je opisati, testirati i usporediti dva odabrana NewSQL sustava za upravljanje bazama podataka koristeći primjere. Budući da je NewSQL novi pristup relacijskim bazama podataka, opisati će se koji modeli podataka su prihvaćeni NewSQL konceptom. Opisan je NewSQL sustav za upravljanje bazama podataka objašnjavajući svrhu njegovog nastajanja, osnovne značajke po kojima je prepoznatljiv te su navedeni primjeri nekih takvih sustava kao i primjeri industrija u kojima se koriste. Nakon toga su odabrana tri NewSQL sustava za upravljanje bazama podataka (VoltDB, NuoDB i MemSQL) te je provedena usporedba po određenim karakteristikama pronađenim na različitim izvorima.

Ključne riječi

NewSQL baze podataka, VoltDB, NuoDB, MemSQL

Sadržaj

1. Uvod 1
2. Relacijske baze podataka 2
2.1. Relacijski model podataka 3
2.2. ACID transakcije 5
3. NewSQL baze podataka 6
3.1. Povijest i razlog nastajanja NewSQL-a 6
3.2. Karakteristike NewSQL-a 7
3.2.1. Skladištenje u glavnoj memoriji 7
3.2.2. Particioniranje 8
3.2.3. Kontrola konkurentnosti 8
3.2.4. Replikacija 9
3.2.5. Sekundarni indeksi 10
3.2.6. Oporavak od pada sustava 11
3.3. Kategorizacija NewSQL-a 11
3.3.1. NewSQL SUBP-a s novom arhitekturom 11
3.3.2. NewSQL SUBP-a za particioniranje baze podataka 12
3.3.3. NewSQL baza podataka u oblaku 13
3.4. Prednosti i nedostaci NewSQL-a 13
3.5. Usporedba NewSQL SUBP-a sa SQL i NoSQL SUBP-ovima 14
4. Pregled NewSQL baza podataka 15
4.1. VoltDB baza podataka 15
4.2. NuoDB baza podataka 17
4.3. MemSQL baze podataka 18
4.4. Usporedba VoltDB, NuoDB i MemSQL baze podataka 20
5. Zaključak 26
Literatura 27
Popis tablica 29
Popis slika 30

1. Uvod

Nakon stvaranja prvog računala nastala je potreba za skladištenjem velike količine podataka na računalima, a kako je vrijeme odmicalo kod mnogih korisnika javila se želja za obradom podataka u različite svrhe te su zbog toga želje korisnika bivale sve veće i zahtjevnije. S time na umu, definirana je baza podataka koja donosi „višu“ razinu pristupa podacima nego ostali programski jezici. Vizija baze podataka je stvaranje jedinstvenog skupa podataka među kojima postoje odnosi (veze) bez nepotrebne redundancije[1].

Od osamdesetih godina prošloga stoljeća relacijske baze podataka postale su najzastupljenije te se one većinom koriste do danas. Ipak, tim tradicionalnim sustavima za upravljanje podacima usprotivili su se neki zahtjevi koje traže moderne aplikacije poput nepredvidivih količina podataka i skalabilnih performansi. Rješenje na te zahtjeve je osnivanje NoSQL sustava za upravljanje bazama podataka, ali i NoSQL-u su otkriveni nedostaci od kojih je najproblematičniji izostanak ACID garancije. Stoga se taj problem morao riješiti, a to je omogućeno pojavom nove generacije relacijskih baza podataka – NewSQL. Smatra se da su NewSQL baze podataka nastale prepoznavanjem skupa dobavljača koji su uzimali najbolje aspekte SQL baze podataka i dizajniranjem novih proizvoda za modernu arhitekturu, posebno arhitekturu u oblaku. Radi toga postoji nedoumica postoji li u tim bazama podataka neka znanstvena novina ili su se pojavile zahvaljujući napredovanju hardvera. Arhitektura NewSQL baza podataka (arhitektura distribuiranih sustava) značajno smanjuje režijske troškove, sposobna je za rad na velikom broju čvorova i podržava SQL upite na visokoj razini. Dakle, NewSQL je nastao kako bi zadovoljio sve potrebe današnjeg poslovanja, posebice u informacijskim sustavima za pružanje podrške pri izvođenju poslovnih procesa koje se nazivaju OLTP (engl. *Online Transaction Processing*). NewSQL baze podataka su nova klasa relacijskih sustava za upravljanje bazama podataka koje podržavaju horizontalno skaliranje kod transakcijskih sustava. Neki primjeri NewSQL sustava za upravljanje bazama podataka su: NuoDB, VoltDB, ClustrixDB, CockroachDB, MemSQL, Pivotal GemFire XD, Altibase, itd.

Nakon uvoda u NewSQL baze podataka u 1. Poglavlju, u 2. Poglavlju bit će objašnjen osnovni model baza podataka – relacijski model podataka sa svojim osnovnim svojstvima. U 3. Poglavlju bit će riječ o razlozima nastajanja NewSQL baza podataka, ali i njihovim karakteristikama, podjeli po kategorijama, prednostima, nedostacima i njihovoj usporedbi sa relacijskim SQL bazama podataka i NoSQL bazama podataka. Zatim će se u 4. Poglavlju detaljnije prikazati i usporediti odabrane NewSQL baze podataka.

[1] višak istovrsnih komponenata u nekom složenom sustavu bez kojih bi sustav mogao raditi, ali se funkcije umnožavaju zbog sigurnosti

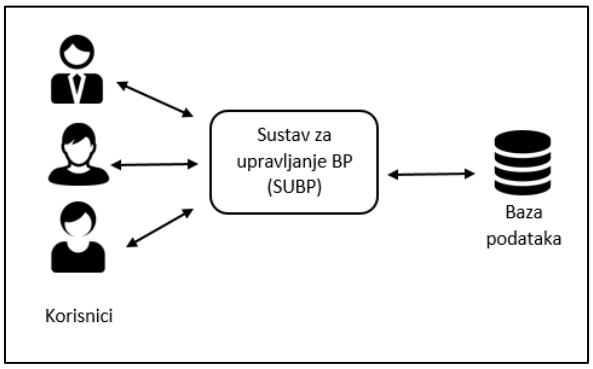
2. Relacijske baze podataka

Razvojem i početkom korištenja baza podataka 60-ih godina dvadesetoga stoljeća koje se definiraju kao skup međusobno povezanih podataka, računalnu opremu na kojoj su pohranjeni i skup programa s pomoću kojih se obavljaju operacije nad njima [1] došlo je do pojave različitih modela podataka. Svaki model podataka je temelj za odgovarajuću bazu podataka te se analogno može zaključiti kako se relacijska baza podataka temelji na relacijskom modelu podataka. Ta baza podataka postala je najrasprostranjenija u današnjem svijetu, a koriste ju mnogi sustavi za upravljanje bazama podataka (u nastavku SUBP) od kojih su neki: Oracle, MySQL, PostgreSQL, MongoDB, IBM Db2, Microsoft SQL Server,… Tablica 1. prikazuje pet najpopularnijih SUBP-ova podataka iz lipnja 2020. godine.

Tablica 1. Najpopularniji SUBP-ovi u lipnju 2020.

	SUBP	**Bodovi**
1.	Oracle	1340.26
2.	MySQL	1268.51
3.	Microsoft SQL Server	1059.72
4.	PostgreSQL	527.00
5.	MongoDB	443.48

Za izrađivanje baze podataka i manipulaciju podataka u istoj, potrebna je programska podrška smještena između korisnika i baze podataka koja se naziva **Sustav za upravljanje bazom podataka** (engl. *Data Base Management System – DBMS*). Osim navedenih zadaća, postoje još neke: fizička nezavisnost podataka, logička nezavisnost podataka, fleksibilnost pristupa podacima, istovremeni pristup do podataka, čuvanje integriteta, mogućnost oporavka nakon kvara, zaštita od neovlaštene uporabe, zadovoljavajuća brzina pristupa i mogućnost podešavanja i kontrole [2].



Slika 1. Sustav za upravljanje bazom podataka

Mogućnost komunikacije između korisnika i SUBP-a omogućeno je jezicima za rad s bazama podataka. Nekada su ti jezici u relacijskim bazama podataka bili podijeljeni po svrhama koje su obavljali: jezik za opis podataka, jezik za rukovanje podacima i jezik za upravljenje

podacima. Međutim, vremenom su se objedinili u jedan integrirani jezik – **Structured Query Language** (u nastavku SQL) koji pripada jezicima za pristupanje bazama podataka 4. generacije. Ostala dva najpoznatija jezika 4. generacije za rad s relacijskom bazom podataka su Query language (QUEL) i Query by Example (QBE).

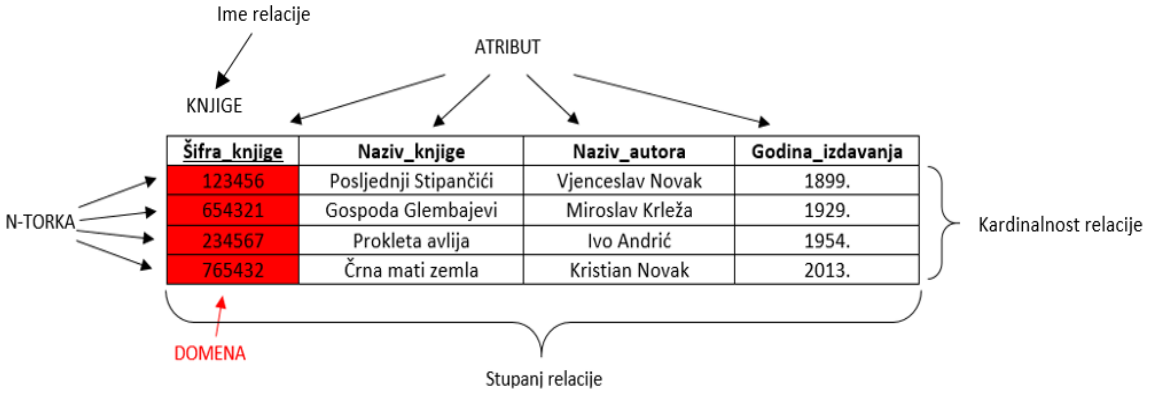
Pojam relacijskih baza podataka pojavio se 1970. godine kada je zaposlenik IBM-a, Edgar Frank „Ted" Codd objavio istraživački rad pod nazivom „*A Relational Model of Data for Large Shared Data Banks*". Iako je u početku bio zanemarivan, postao je općeprihvaćeni 80-ih godina dvadesetoga stoljeća te je tim radom napravio proboj u sustavima relacijskih baza podataka. Na temelju njega su definirani i ustrojeni mnogi koncepti koji se koriste i u današnjim relacijskim SUBP-a. Također, isti autor je objavio pravila pod nazivom „Coddovih 12 pravila" u kojima je htio opisati što je potrebno od SUBP-a kako bi se smatrao sustavom za upravljanje relacijskim bazama podataka. Iako većina SUBP-a ne koristi svih dvanaest pravila, ustanovljeni su minimalni uvjeti potrebni za sustav za upravljanje relacijskim bazama podataka, a to su: prezentacija podatka korisniku u obliku relacije (tablica od stupaca i redaka) i omogućavanje relacijskim operatorima rukovanje podacima u tablicama.

2.1. Relacijski model podataka

Relacijski model podataka prikazuje isključivo logičke segmente podataka. Ustanovljeno je da se relacijski model podataka sastoji od tri značajke: skup koncepata za strukturu baze podataka, skup operacija za provođenje nad podacima i skup pravila integriteta podataka. Osnovne svrhe ovog modela su: nezavisnost podataka, smanjenje zalihosti podataka, omogućavanje razvoja jezika za obradu podataka i prikazivanje modela za opis podataka [3].

Osnovni strukturalni koncepti svakoga relacijskog modela za upravljanje bazama podataka je prikazivanje podataka u obliku relacija sastavljenih od odabranih slogova (n-torki) i atributa. Radi lakšeg razumijevanja u praksi se često koriste drugi termini: tablica (relacija), redak (slog) i stupac (atribut).

Relacija (pravokutna tablica) se prikazuje u obliku dvodimenzionalne tablice koja se sastoji od atributa (stupaca) i slogova (redaka). Prema Codd-u, definicija relacije je: *Neka su dani skupovi $D_1, D_2, \dots, D_n$ (ne obavezno različiti), R je relacija nad ovih n skupova (n>0), ako je to skup n-torki takav da za svaku n-torku vrijedi da je prvi element n-torke iz D_1, drugi iz D_2, n-ti iz D_n*. Svaka ta relacija posjeduje jedinstveno ime po kojemu se razlikuje od ostalih tablica unutar određene baze podataka. **Atribut** (stupac) je opisan kao svojstvo entiteta, a smatra se imenovanim stupcem relacije pri čemu mu je ime jedinstveno. Skup svih mogućih vrijednosti jednog atributa je domena (jedan stupac), dok je broj atributa u jednoj relaciji stupanj (red) relacije. Jedan redak u tablici odgovara jednom **slogu** (n-torki), a broj slogova u jednoj relaciji je kardinalnost relacije. Nije moguće postojanje više jednakih n-torki u jednoj tablici budući da se svaka n-torka mora razlikovati barem po primarnom ključu relacije što znači da je svaki redak jedinstven. Također, još jedno bitno svojstvo za relacije u relacijskom modelu podataka je nebitan redoslijed redaka što znači da se izmijenjeni slijed redaka može ponovno spremiti.



Slika 2. Strukturalni koncepti proizvoljne relacije KNJIGE

Ključ relacije u relacijskom modelu je minimalni skup atributa relacije čije vrijednosti jednoznačno određuju n-torke relacije. Svaka relacija se sastoji od barem jednog ključa. Takvih ključeva može biti više i nazivaju se kandidatima za ključ, a određuju se na temelju dva uvjeta [4]:

1. Uvjet jedinstvenosti – Vrijednost ključa svake n-torke relacije jedinstveno određuje n-torku, odnosno ne postoje dva retka u tablici takva da imaju iste vrijednosti atributa koji čine ključ.

2. Uvjet neredudandnosti – Ne postoji niti jedan atribut kao dio ključa koji se može izostaviti iz ključa a da se pritom 1. uvjet ne gubi, odnosno ključ je unija minimalnog broja atributa.

Izabrani ključ (iz kandidata za ključ) za identifikaciju naziva se primarni ključ, a može postojati i vanjski (strani) ključ ukoliko neki skup atributa u relaciji nije ključ, ali je ključ u nekoj drugoj relaciji.

Nadalje, postoje dva dodatna ograničenja sa svrhom točnog određivanja koje su akcije (operacije) dopustive za rad s relacijama. Oba ograničenja se odnose na integritetna pravila:

1. Integritet entiteta (cjelovitost ključa) – pravilo koje određuje da nijedan od atributa primarnog ključa ne smije biti jednak NULL (nepoznatoj) vrijednosti, odnosno ključ relacije ne smije poprimiti NULL vrijednost.

2. Referencijalni integritet (cjelovitost ovisnosti) – pravilo definirano tako da u prvoj relaciji u kojoj postoji vanjski ključ koji odgovara primarnom ključu druge relacije, svaka vrijednost vanjskog ključa prve relacije mora biti ili jednaka vrijednosti primarnog ključa n-torke iz druge relacije ili jednaka NULL vrijednosti [5].

Rezultat logičkog oblikovanja baze podataka je **relacijska shema** kojom se opisuje građa relacije, a ne opisuje se njezino značenje. Ta shema sadrži ime relacije i imena atributa u zagradama pri čemu je primarni ključ podcrtan (engl. *bold*), a vanjski ključ nakošen (engl. *italic*). Skup relacija određenih relacijskom shemom baze podataka je relacijska baza podataka.

Također, u relacijskim modelima podataka se koristi pojam **relacijska algebra** kojeg je kreirao Codd. Budući da se relacija u relacijskom modelu podataka smatra kao skup, koriste se iste operacije na relacijama kao i na skupovima, Relacijska algebra se sastoji od skupa operatora pomoću kojih se izvode operacije na relacijama relacijske baze podataka. Kao rezultat algebarskih operacija nastaje nova relacija koja zadovoljava navedeni algebarski izraz koji

može biti upit ili pretraživanje. Neke od tih operacija su: unija, presjek, razlika, selekcija, projekcija, Kartezijev produkt, itd.

2.2. ACID transakcije

SQL podržava transakcije kako bi se riješio problem kada jedan ili više korisnika istovremeno žele promijeniti podatke u istoj relaciji. Transakcija je aktivnost (niz aktivnosti) koje vrši jedan aplikacijski program čitajuću ili ažurirajući sadržaj u bazi podataka [5]. Podijeljene operacije unutar transakcije se moraju zajedno obaviti kao nedjeljiva cjelina.

Sa stajališta SUBP-a, četiri svojstva skraćena akronimom ACID (engl. ***A**tomicity, **C**onsistency, **I**solation, **D**urability*) moraju biti zadovoljena kako bi podaci bili valjani, makar i u slučaju pada sustava [6]:

1. Dosljednost: Transakcije se ne mogu samo djelomično izvršiti, sve radnje unutar jedne transakcije moraju završiti bez greške inače baza ostaje nepromijenjena.

2. Konzistentnost: Transakcija mora preobraziti bazu podataka iz jednog konzistentnog stanja u drugo takvo stanje.

3. Izolacija: Paralelno izvođenje više transakcija mora dati rezultat kao da su se izvodile jedna iza druge.

4. Trajnost: Uspješno završene transakcije ostaju trajno zabilježene u bazi podataka i ne gube se pri kvarovima (npr. nestanak struje).

Slika 3. ACID terminologija [7]

3. NewSQL baze podataka

Do sada je većina programera upoznata sa SQL-om i relacijskim SUBP-a. Osnovna načela takve arhitekture postoje već desetljećima. Početkom dvadeset i prvoga stoljeća stigla su NoSQL rješenja poput MongoDB ili Cassandra razvijena za distribuirane, skalabilne potrebe podataka. Dok se razne inačice NoSQL baza podataka još uvijek koriste, postoji još jedna paradigma koja se javlja paralelno s NoSQL-om – NewSQL. NewSQL pokušava kombinirati prednosti relacijskih SUBP-a (konzistencija[2]) i koristi od NoSQL-a (skalabilnost[3]). Takve baze podataka primarno su namijenjene tvrtkama koje upravljaju podacima visokog profita. Iako se različite NewSQL baze podataka razlikuju u unutarnjoj arhitekturi, svi koriste model relacijskih baza podataka i rade uz pomoć SQL-a.

3.1. Povijest i razlog nastajanja NewSQL-a

Promjena u SUBP-a se počela događati kada je istraživački tim predvođen Michael-om Stonebraker-om objavio seminarski rad *„The end of an Architectural Era* (*It's Time for a Complete Rewrite*)". U tom radu je istaknuto da se hardverske pretpostavke relacijske arhitekture baze podataka više ne koriste te da mnoštvo suvremenih opterećenja baze podataka sugerira da jedna arhitektura možda nije optimalna za sva radna opterećenja. Stonebraker i njegov tim predložili su brojne varijante postojećeg dizajna SUBP-a od kojih je svaka optimizirana za specifično opterećenje aplikacije. Istaknula su se dva takva dizajna [8]:

- H-Store: „čista" raspodjela baze podataka u memoriji.
- C-Store: dizajn za stupčastu bazu podataka.

Oba dizajna su postala utjecajna u godinama koje su dolazile i prvi su primjeri onoga što je postalo poznato kao NewSQL.

Budući da se od 2000-ih godina bilježi nagli porast broja SUBP-a, taj porast je „eksplodirao" u razdoblju od 2008.-2009. godine kada su se deseci novih SUBP-a pojavili u tom kratkom razdoblju. Iako su se neki prestali koristiti, neki poput MongoDB, Cassandra i HBase su osvojili značajan tržišni udio. Krajem 2009. godine pojam NoSQL postao je uobičajen za SUBP-a koji se razlikuju od tradicionalnih SQL baza podataka.

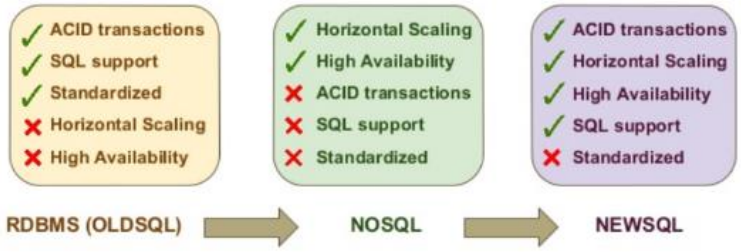
Pojava pojma NewSQL baze podataka dogodila se 2011. godine kada ga je primijenio Matthew Aslett u istraživačkom članku u kojemu je diskutirao o novoj skupini SUBP-a. NewSQL je postao nova generacija modernih relacijskih SUBP-a koji nastoje osigurati skalabilnost NoSQL sustava za radno opterećenje mrežnih transakcija (OLAP) uz uzdržavanje ACID jamstva tradicionalnog sustava baza podataka [10]. Iako postoji više dobavljača na ovom tržištu, svaki se razlikuje po različitoj primjeni.

[2] trajnost ili izdržljivost

[3] sposobnost sustava da se prilagodi povećanim zahtjevima obrade na predvidiv način, bez da postane previše kompleksan, skup i nepraktičan

3.2. Karakteristike NewSQL-a

Unutar karakteristika NewSQL SUBP-a objedinjeni su elementi koji su već od ranije poznati iz relacijskih i NoSQL SUBP-ova, ali postoje i neki novi koncepti potrebni za razumijevanje novosti u ovakvim sustavima. Kako se NewSQL pretežno koristi u aplikacijama s OLTP transakcijama, ovo su ukratko karakteristike tih transakcija: kratkotrajne su, koriste indeksirane pretrage (bez skeniranja tablice), koriste malu količinu podataka po transakciji, imaju mali broj obrazaca (mali broj upita s različitim argumentima) [10]. Ostale važne karakteristike se mogu podijeliti na šest dijelova: skladištenje u glavnoj memoriji, particioniranje[4], kontrola konkurentnosti, replikacija, sekundarni indeksi i oporavak od pada sustava.



Slika 4. Razvoj baza podataka [11]

3.2.1. Skladištenje u glavnoj memoriji

Iako su glavni SUBP-ovi koristili arhitekturu s diskovnom pohranom temeljenu na izvornim SUBP-ovima, ta ideja je postala skupa i imala je ograničen kapacitet. Ideja spremanja baze podataka u potpunosti u glavnu memoriju nije ništa novo. Seminarsko istraživanje na Sveučilištu Wisconsin-Madison ranih osamdesetih godina postalo je temelj mnogim aspektima SUBP-a sa skladištenjem u glavnoj memoriji, uključujući indekse, obradu upita i algoritme oporavka. U istom desetljeću razvijeni su i prvi distribuirani SUBP-ovi koji skladište u glavnoj memoriji. Razvoj takvih sustava nastao je zbog sve većih memorija i pada cijena pa je bilo isplativije držati cijelu bazu podataka u glavnoj memoriji. Novina koja dolazi s glavnom memorijom NewSQL sustava je mogućnost iseljavanja podskupina baza podataka u trajno skladištenje da bi se smanjio njihov trag u memoriji. To omogućuje SUBP-ovima podržavanje baze podataka veće od raspoložive memorije bez potrebe za prebacivanjem na diskovno orijentiranu arhitekturu. Prednost ovog pristupa je veća brzina glavne memorije, sprječavanje stavljanja podataka u pričuvnu memoriju i bolje performanse jer se svim podacima pristupa direktno. Dakle, ovaj pristup je brži nego tradicionalni pa su SUBP-ovi koji ga koriste spretniji pri obradi online transakcija. S obzirom da se u ovom pristupu koriste velike količine podataka u memoriji, postoje mehanizmi za praćenje koji identificiraju podatke koji se ne koriste često i deložiraju ih iz memorije. Svi ti SUBP-ovi zadržavaju ključeve izbačenih podataka u indeksima baze podataka što sprječava potencijalne uštede u memoriji za one aplikacije s puno sekundarnih indeksa.

[4] sposobnost da se sistem izvodi kao da su dva ili više nezavisnih sistema

3.2.2. Particioniranje

Ova sastavnica NewSQL SUBP-a označava način na koji gotovo svi NewSQL SUBP-ovi proširuju bazu podataka dijeljenjem na disjunktne skupove nazvane particijama[5]. Tablice baze podataka u kojima su sadržani podaci horizontalno su podijeljene u više fragmenata čije su granice temeljene na vrijednostima jednog (ili više) stupaca tablice. Povezani fragmenti iz više tablica kombiniraju se zajedno kako bi tvorili particiju kojom upravlja jedan čvor. Taj je čvor odgovoran za izvršavanje svih upita s kojima je potreban pristup podacima pohranjenim u njegovoj particiji. Nakon toga SUBP izvršavanju upita usmjerava na svaki čvor, a zatim kombinira rezultate sa svih čvorova zajedno u jedan rezultat. Svi NewSQL sustavi, osim ScaleArc-a, koji podržavaju izvorne particije, pružaju ovu funkciju.

Neki NewSQL SUBP-ovi koji koriste homogenu, a neki heterogenu arhitekturu gdje se pod homogeno odnosno heterogeno smatraju čvorovi u bazi podataka, a ne općenito baza podataka. Homogene i heterogene arhitekture baze podataka se razlikuju po tome što je na heterogenoj arhitekturi moguće dodavanje resursa za obradu bez potrebe za ponovnim particioniranjem baze podataka. Baze podataka za mnoge OLTP aplikacije imaju ključno svojstvo zbog čega su podložni particioniranju te homogenoj arhitekturi. One organiziraju sheme baze podataka u obliku stabla u kojemu su listovi (potomci) imaju odnos s korijenom preko vanjskog ključa. Tablice su zatim raspoređene na atribute uključene u te odnose tako da su svi podaci za jedan entitet smješteni zajedno u jednoj particiji. To rezultira s prednosti jer omogućuje većini (a možda i svim) transakcijama pristup podacima samo u jednoj particiji. Međutim, NuoDB i MemSQL SUBP-ovi umjesto homogene koriste heterogenu arhitekturu čija je svrha učitavanje što manje količine podataka iz čvorova u kojima su podaci. Heterogena arhitektura podrazumijeva dijeljenje čvorova tako da postoje čvorovi za pohranu i čvorovi za obradu. U MemSQL-u se heterogena arhitektura sastoji od čvorova za prikupljanje podatka i čvorova (listova) za pohranjivanje podataka. Čvorovi (listovi) zaduženi za pohranjivanje izvršavaju dio upita da bi količina podataka koja odlazi na čvorove za izvršavanje bila manja. NuoDB se malo razlikuje od MemSQL-a budući da čvorovi zaduženi za pohranjivanje samo pohranjuju te nemaju drugih mogućnosti, ali o tome će se više govoriti u poglavlju o NuoDB-u.

Još jedan važan segment particioniranja u NewSQL sustavima je podržavanje migracije, ali samo kod nekih sustava. Migracija omogućava SUBP-u da premješta podatke između fizičkih resursa radi ponovnog uspostavljanja ravnoteže, odnosno povećanja/smanjenja kapaciteta SUBP-a bez prekida usluge. Postoje dva pristupa za postizanje migracije: organiziranje baze podataka premještanjem virtualnih particija i premještanje pojedinih slogova ili grupa slogova. Prvi pristup koriste ScaleBase i H-Store SUBP-ovi dok drugi, precizniji pristup koriste Clustrix i AgilData SUBP-ovi.

3.2.3. Kontrola konkurentnosti

Karakteristika kontrole konkurentnosti (istodobnosti) omogućuje krajnjim korisnicima pristup u više baza podataka na više programa, ostvarujući privid da svaki od njih svoju transakciju provodi sam na namjenskom sustavu [10]. Isto tako, najistaknutiji je detalj

[5] logički odvojen dio diska koji operativni sustav tretira kao posebnu jedinicu

implementacije SUBP-ova za obradu transakcija jer utječe na gotovo sve aspekte sustava. Protokol koordinacije transakcija može biti ili sa centraliziranim koordinatorom ili sa decentraliziranim koordinatorom. U sustavu sa centraliziranim koordinatorom, sve transakcije moraju proći kroz koordinator koji odlučuje može li transakcija nastaviti ili ne. U decentraliziranom sustavu svaki čvor održava stanje transakcija koje pristupaju podacima te čvorovi moraju međusobno koordinirati da bi utvrdili sukobljavaju li se transakcije. Ovaj protokol je bolji za skalabilnost, ali satovi čvorova moraju biti jako sinkronizirani kako bi se stvorio globalan redoslijed transakcija za postizanje ispravnog funkcioniranja baze podataka.

U početku su SUBP-ovi koristili dvofazne sheme zaključavanja (2PL). Primjeri takvih SUBP-ova su SDD-1 i IBM-ov R*. Iako su slični, razlika između ta dva SUBP-a je u tome što je koordinacija transakcija u R* bila potpuno decentralizirana dok je u SDD-1 obradom distribuiranih transakcija upravljao centralizirani koordinator. Ipak, NewSQL sustavi koji se temelje na novim arhitekturama izbjegavaju 2PL zbog složenosti rješavanja zastoja. Umjesto 2PL-a, upotrebljavaju se kontrole konkurentnosti s korištenjem vremenskih oznaka (engl. *Timestamp Ordering*).

Najkorišteniji protokol u NewSQL sustavima je decentralizirana više verzijska kontrola konkurentnosti [6] (MVCC). U tom protokolu SUBP stvara novu verziju slogova u bazi podataka kada su podaci ažurirani transakcijom. On omogućuje dugotrajne transakcije za čitanje koje ne blokiraju pisanje, ali i dovršavanje transakcija čak i ako druga transakcija ažurira iste slogove. Gotovo svi NewSQL sustavi temeljeni na novim arhitekturama koriste ovaj protokol, a neki su: MemSQL, HyPer, HANA i CockroachDB.

Ipak, postoje i sustavi koji ne koriste nijednu MVCC varijantu, a jedini takav komercijalni NewSQL SUBP je VoltDB. Ono što ga razlikuje je od MVCC-a je ne miješanje transakcija nego izvršavanje jedne po jedne transakcije uz kontrolu konkurentnosti vremenskih oznaka. Dakle, on naređuje transakcije na temelju logičkih vremenskih oznaka, a zatim planira njihovo izvršavanje na particijama kada dođe njihov red. Završavanjem transakcije, on ima direktan pristup podacima na toj particiji stoga sustav ne mora postavljati zaključavanja na svojim strukturama podataka. Time je omogućeno učinkovito izvršavanje transakcija koje pristupaju samo jednoj particiji bez sukoba s drugim transakcijama. Ako transakcije obuhvaćaju više particija, javlja se nedostatak kontrole konkurentnosti zbog kašnjenja mrežne komunikacije i mirovanja čvorova dok čekaju poruke.

Postoje i sustavi koji kombinirano koriste 2PL i MVCC. Najpoznatija primjena ovog pristupa je MySQL-ov InnoDB, ali se koristi i u Google Spanner-u, NuoDB-u i Clustrix-u. Koristeći kombinaciju, ta shema omogućuje upite samo za čitanje da bi se izbjeglo zaključavanje i ne bi blokiralo pisanje transakcija.

3.2.4. Replikacija

Pojam replikacija baza podataka znači kreiranje i održavanje višestrukih kopija iste baze podataka [13] te svi moderni SUBP-ovi, uključujući NewSQL sustave, podržavaju neku vrstu replikacije baze podataka. Shema repliciranje se naziva *master* shema jer svi čvorovi mogu obaviti ažuriranje. Vezano uz replikaciju, važno je kako SUBP provodi konzistentnost podataka kroz čvorove. Da se transakcija smatra završenom u strogo konzistentnom SUBP-u, ona prvo

[6] metoda kontrole paralelnosti koju obično koriste SUBP-ovi za pružanje istodobnog pristupa bazi podataka

mora biti potvrđena i instalirana na svim replikacijama. Taj pristup ima prednosti što replikacije mogu koristiti upitima, uvijek su dosljedne i sve promjene koje uzrokuje transakcija su vidljive u idućoj transakciji bez obzira na kojem SUBP-u se pristupa. To znači, ako replikacija ne uspije, čvorovi su i dalje usklađeni te nema izgubljenih ažuriranja. Unatoč tomu, da bi se održala usklađenost, sve replike moraju potvrditi transakcije čime se stvaraju dodatni troškovi i zastoji. Iako se svi NewSQL SUBP-ovi služe strogom konzistencijom postoji i eventualna konzistencija u kojoj sve replikacije moraju potvrditi izmjene prije nego SUBP obavijesti aplikaciju da je pisanje uspjelo.

Iduća važna stvar vezana za replikaciju je način na koji se obavljaju replikacije. Prva replikacija je poznata kao aktivno-aktivna replikacija i u njoj svaki čvor istovremeno izvršava upite iz transakcija tj. SUBP upit izvršava paralelno u svim replikacijama. Druga replikacija je poznata kao aktivno-pasivna replikacija te ona provodi transakcije na jednom čvoru nakon čega SUBP prenosi transakcije u ostale replikacije. Tu drugu vrstu replikacije, aktivno-pasivnu replikaciju, koristi većina NewSQL SUBP-ova jer su transakcije nedeterminističke i zbog toga nije važan redoslijed na čvorovima. Postoje i SUBP-ovi s deterministčkim transakcijama koje se izvršavaju jednakim redoslijedom što rezultira istim stanjem baze podataka.

Kako se u modernim informacijskim okruženjima sustavi sve više raspoređuju u više podatkovnih centara između kojih su velike geografske razlike, pojavilo se razmatranje replikacija preko širokopojasne mreže (WAN). Bilo koji NewSQL SUBP može se konfigurirati za sinkrono ažuriranje podataka putem WAN-a, ali to značajno usporava transakcije. Preporučuje se asinkrona replikacija koja je prisutna u NewSQL SUBP-ovima Spanner i CockroachDB.

3.2.5. Sekundarni indeksi

Sekundarni indeksi su načini učinkovitog pristupa zapisima u bazi podataka pomoću nekih informacija koje su različiti od primarnog indeksa (primarnog ključa) što omogućuje SUBP-u da podržava brze upite bez traženja primarnog ključa. U distribuiranim SUBP-ovima postoje dileme oko centralizacije ili decentralizacije. U centraliziranom sustavu postoji jedan sekundarni indeks i on se nalazi na jednom mjestu čime je lakše održavanje. NewSQL sustavi s novim s novim arhitekturama su decentralizirani i koriste particionirane sekundarne indekse kod kojih svaki čvor ima dio sekundarnog indeksa, a ne kopiju. Za testiranje valjanosti indeksa postoje dva različita pristupa koji se koriste: replicirani indeksi i particionirani indeksi. S obzirom na postojanje indeksa s podacima iz kompletne baze podataka u svakom čvoru, replicirani indeksi vraćaju podatke upitu kroz samo jedan čvor. Problem repliciranih indeksa što prilikom izmjene atributa na koje se odnose indeksi, SUBP mora izvršiti distribuiranu transakciju za ažuriranje svih kopija indeksa. Particionirani indeksi se odnosi samo na podatke na jednom čvoru dajući za rezultat povećanu brzinu obrade prilikom izmjene atributa povezanih uz indeks jer se mora ažurirati samo jedan indeks, a ne sve kopije indeksa kao kod repliciranog indeksa.

3.2.6. Oporavak od pada sustava

Dok je kod tradicionalnih SUBP-ova glavna briga osigurati ne gubljenje ažuriranja, od novih SUBP-ova se očekuje da će moderne web aplikacije stalno biti aktivne (on-line) da ne bi došlo do rušenja web mjesta koje uzrokuje velike troškove. U slučaju pada sustava postoje dva pristupa za oporavak sustava. Prvi, tradicionalni pristup bez replikacije, funkcionira na način da se SUBP nakon pada sustava vrati na mrežu, učita zadnju kontrolnu točku koju je uzeo s diska, a zatim ponovno zapisuje dnevnik zapisa (WAL) kako bi vratio stanje baze podataka iz trenutka pada. Taj slučaj nije primjenjiv u distribuiranim SUBP-ovima s replikacijama jer pri rušenju sustava, on promovira jedan od podređenih čvorova kao glavni čvor. Kada se srušeni glavni čvor vrati u aktivnosti, on me može samo učitati kontrolnu točku i ponovno pokrenuti svoj WAL jer je SUBP nastavio obrađivati transakcije čime se stanje baze podataka nastavilo dalje [10]. Tom čvoru nedostaju nova ažuriranja koja su se dogodila dok je bio u mirovanju. Za dohvaćanje tih ažuriranja postoje dva potencijalna načina. Za prvi način čvor koji se oporavlja treba učitati zadnju kontrolnu točku i WAL iz lokalne pohrane, a zatim povući propuštene zapise dnevnika iz ostalih čvorova. Dok god taj čvor može obraditi zapisnik brže nego što se dodaju nova ažuriranja, čvor će se s vremenom vratiti u isto stanje kao i ostali čvorovi replikacije. Druga opcija je da čvor koji se oporavlja odbaci kontrolnu točku i sustav preuzme novu od koje se čvor vraća na oporavljanje. Osim što se smanjuje vrijeme potrebno za oporavak, još jedna prednost je korištenje istog mehanizma i u SUBP-ovim za dodavanje novog čvora replikacije.

3.3. Kategorizacija NewSQL-a

S obzirom na definiciju NewSQL SUBP-a, današnji takvi sustavi se mogu podijeliti u tri skupine prema određenim čimbenicima njihove implementacije. Te tri kategorije su: NewSQL SUBP-a s novom arhitekturom (engl. *New Architectures*), NewSQL SUBP-a za particioniranje baze podataka (engl. *Transparent Sharding Middleware*) i NewSQL baza podataka u oblaku (engl. *Database-as-a-Service*) [10].

3.3.1. NewSQL SUBP-a s novom arhitekturom

Jedna vrsta SUBP-a koji pripadaju NewSQL-ovima jesu SUBP-a iz temelja izgrađeni kao potpuno novi NewSQL SUBP-a. Takvi NewSQL sustavi nove arhitekture omogućuju kreiranje potpuno nove baze podataka po čemu su drugačije od sustava za particioniranje. Osnovne značajke ovih sustava su: podrška za kontrolu istodobnosti čvorova, distribuirana obrada upita, tolerancija grešaka putem replikacija u bazi podataka i kontrola protoka. Neki sustavi koriste skupine (engl. *„cluster“*) sa *shared-nothing*[7] čvorovima u kojemu svaki čvor upravlja podkskupom podataka. Prednost korištenja novog SUBP-a koji je izgrađen za distribuirano izvršavanje je ta da se svi dijelovi sustava mogu optimizirati za okruženje s više

[7] svaki je čvor načinjen od procesora, glavne memorije i diska i komunicira s drugim čvorovima putem mreže za međusobno povezivanje

čvorova. To uključuje stvari poput alata za optimizaciju upita i protokol za komunikaciju među čvorovima. Dakle, većina NewSQL SUBP-a može izravno slati podatke unutar upita između čvorova umjesto da ih se prvo usmjerava na središnji lokaciju kao u nekim sustavima s posredničkim programima. Također, ovi sustavi (isključujući Google Spanner) upravljaju vlastitom primarnom memorijom, bilo u memoriji, bilo na disku. To je važan segment jer omogućuje SUBP-a za „slanje upita podacima“, a ne za „donošenje podataka na upit“ pa se kao rezultat dobije manji mrežni promet jer je slanje upita obično manje mrežnog prometa nego da se prenose podaci u računanje. Međutim, postoje i neki nedostaci od kojih se najviše ističe što su mnoge organizacije oprezne pri uvođenju novih i ne provjerenih tehnologija s velikom instalacijskom bazom. Broj iskusnih ljudi u sustavima je mnogo manji u odnosu na popularne dobavljače SUBP-a što dovodi do mogućeg gubljenja pristupa postojećim alatima za administraciju i izvještaje. Naravno, postoje neki SUBP-a koji uspijevaju riješiti ovaj problem održavanjem kompatibilnosti s MySQL žičanim protokolom. Neki primjeri ovakvih SUBP-a su: Clustrix, Cockroach, Google Spanner, H-Store, HyPer, MemSQL, NuoDB, SAP HANA, VoltDB, itd.

3.3.2. NewSQL SUBP-a za particioniranje baze podataka

U ovom pristupu je omogućeno stvaranje baze podataka iz dijeljene baze podataka na jednom čvoru. Particioniranje u tom pristupu ima karakteristike da svaki čvor, odnosno svaku novu bazu podataka, pokreće isti SUBP, ima samo dio cjelokupne baze podataka i nije predviđeno za nezavisan pristup i ažuriranje od različitih aplikacija. Centralni posrednički program usmjerava upite, koordinira transakcije, upravlja replikacijom i podjelom podataka preko čvorova odnosno služi za komunikaciju između novih čvorova. Dakle, posrednički program predstavlja jednu logičku bazu podataka aplikacije bez potrebe za izmjenom temeljnog SUBP-a. Ključna prednost ove kategorije pristupa je korištenje jednake tehnologije pri čemu nije potrebno mijenjanje aplikacije zato što posrednički program aplikaciji prikazuje jednu logičku bazu podataka. Iako posrednički program olakšava organizaciji skaliranje njihovih baza podataka na više čvorova, takvi sustavi i dalje moraju koristiti tradicionalne SUBP-ove (npr. MySQL, Oracle, itd.) na svakom čvoru. Idući nedostatak je da su ovi SUBP-ovi bazirani na arhitekturi koja koristi pohranjivanje na diskovima i stoga ne mogu koristiti upravitelj skladištenja ili shemu kontrole istodobnosti koji su optimizirani za memorijsko pohranjivanje kao u nekim NewSQL SUBP-ovima izgrađenim na novim arhitekturama. Dokazano je da su naslijeđene komponente arhitekture s pohranjivanjem na diskovima opterećenje i sprječavaju skaliranje tradicionalnih SUBP-ova kako bi se iskoristio veći broj jezgara procesora i veći kapacitet memorije [10]. Još jedan nedostatak očituje se u optimizaciji upita i podataka na particioniranim čvorovima za složene čvorove, ali ipak to omogućuje svakom čvoru primjenjivanje njihove vlastite lokalne optimizacije za svaki upit. Primjeri ovih NewSQL SUBP-a za particioniranje baze podataka su: : AgilData Scalable Cluster, MariaDB MaxScale, ScaleAre, ScaleBase, itd.

3.3.3. NewSQL baza podataka u oblaku

Ovu vrstu nude pružatelji računalskih usluga u oblaku pod konceptom NewSQL „baze podataka kao usluge“ (engl. *Database-as-a-Service*, u nastavku DBaaS) [10]. Koristeći ovaj pristup, organizacije ne moraju održavati SUBP niti na vlastitom hardveru niti na virtualnom stroju u oblaku. Umjesto toga, DBaaS održava fizičku konfiguraciju baze podataka uključujući podešavanje sustava, replikaciju i izradu sigurnosnih kopija. Klijenti dobivaju URL vezu s SUBP-om zajedno s nadzornom pločom i API-om za kontrolu sustava te plaćaju u skladu s očekivanom uporabom resursa u aplikaciji tako što poslužitelj određuje maksimalno korištenje resursa koje je u mogućnosti jamčiti. Jedina dva primjera NewSQL baza podataka u oblaku od 2016. godine su Aurora i ClearDB. Najistaknutiji primjer tih baza podataka je Amazonova Aurora koja koristi strukturiranu pohranu podataka za poboljšanje paralelizma ulaza/izlaza te djeluje na ranije opisani način. ClearDB pruža vlastiti prilagođeni DBaaS koji se može primijeniti na svim glavnim platformama u oblaku te tako ima prednost što može distribuirati bazu podataka između različitih pružatelja usluga u istoj geografskoj regiji kako bi se izbjegli zastoji zbog prekida usluge. Postojalo je još nekoliko tvrtki u tom području (npr. Xeround, GenieDB), ali su propale te su tako prisilile svoje klijente da pronađu novog davatelja usluga i premjeste svoje podatke iz DBaaS-a prije negoli se zatvore.

3.4. Prednosti i nedostaci NewSQL-a

Kroz karakteristike NewSQL SUBP-a već su se pojavile određene prednosti i nedostaci koji su prisutni u radu tih sustava u odnosu na ostale SUBP-ove. Međutim, treba uzeti u obzir i podjelu tih sustava po kategorijama te uvidjeti da ne vrijede sve prednosti i/ili svi nedostaci za sve kategorije. Uočavanje prednosti i nedostataka NewSQL sustava većinom se zasniva na usporedbi NewSQL SUBP-a sa SQL i NoSQL SUBP-ovima. U Tablici 2. su prikazane općenite prednosti i nedostaci NewSQL baza podataka [14].

Tablica 2. Osnovne prednosti i nedostaci NewSQL SUBP-ova

PREDNOSTI	NEDOSTACI
Transakcijska podrška	Nisu postavljeni za opću namjenu kao tradicionalni SQL sustavi
Minimalna kompleksnost aplikacije	Ograničen pristup bogatim alatima SQL sustava
Jaka konzistencija	Memorijske arhitekture nisu učinkovite za terabajte
Prepoznatljiv SQL i standardni alati	
Bogatija analitika utječe na SQL i proširenja	
Moguće povezivanje NoSQL-a s tradicionalnim modelima podataka i upita	

3.5. Usporedba NewSQL SUBP-a sa SQL i NoSQL SUBP-ovima

Ako usporedimo SQL i NewSQL SUBP-ove možemo uočiti da su vrlo slični, ali postoje određene razlike. Oba pristupa podržavaju relacijsko modeliranje na temelju kojega stvaraju univerzalni model za pohranu podataka u računalo. Taj model osigurava konstantnu konzistentnost podataka tijekom provedbe upita. SQL podržava vertikalno skaliranje za koje je potrebno više memorije i mjesta za pohranu, brži procesori što rezultira većim cijenama. Pohranjenim podacima manipuliraju pomoću SQL-a kojeg podržavaju s time da NewSQL koristi poboljšane funkcionalnosti za stari SQL. Uz poboljšane funkcionalnosti, NewSQL je vrlo učinkovit za obradu jednostavnijih i složenijih upita dok SQL ne uspijeva obaviti kompleksnije upite. Još jedna velika razlika između ova dva pristupa je ne podržavanje distribuiranih baza podataka u SQL SUBP-ovima. Naime, SQL SUBP-ovi su monolitni sa stajališta arhitekture jer ne mogu automatski distribuirati podatke i upite u više primjeraka, oni podržavaju arhitekturu s jednim poslužiteljem. Uz posredničke sustave mogu postati distribuirani sustavi, ali i dalje sa slavim performansama. Rastom potrebe za većim brojem operacija nad bazom podataka, jedan poslužitelj nije mogao podnositi sve više transakcija pa je dizajniran NewSQL s distribuiranom arhitekturom koja koristi više poslužitelja.

U usporedbi NoSQL i NewSQL SUBP-ova više je razlika nego sličnosti. NoSQL SUBP-ovi su pri osnutku kao ključne odrednice smatrali odricanje transakcijskih garancija i relacijskog modela u korist eventualne konzistentnosti. To su učinili s uvjerenjem da ti aspekti dotadašnjih SUBP-ova usporavaju njihovu sposobnost skaliranja i postizanja visoke dostupnosti. Iako je NoSQL prvi uveo distribuiranu bazu podataka sa više čvorova, još uvijek (eventualna) konzistencija nije potpuna pa dolazi do grešaka s podacima. Ta situacija se javlja jer se NoSQL ne bavi ACID pravilima za održavanje pouzdane i konzistentne baze podataka. Umjesto toga, NoSQL je fokusiran na BASE svojstva (engl. *Basically Available, Soft state, Eventual consistency*) koja se smatraju suprotnima od ACID svojstava što znači da odustaju od konzistentnosti. NewSQL rješava te greške uvođenjem sigurne konzistentnosti, a tako i ubrzava rad cjelokupnog sustava stoga poduzeća i organizacije koje generiraju terabajte transakcijskih podataka svakodnevno u OLTP sustavima, radije biraju NewSQL. Kao što je spomenuto u prethodnom odlomku, uz SQL i NoSQL podržava vertikalno skaliranje dok NewSQL podržava vertikalno i horizontalno skaliranje što omogućuje rad s Big Data-om pružajući mogućnost istodobnog rada održavajući ACID svojstva.

Sve ove posebnosti koje čine SQL, NoSQL i NewSQL SUBP-ove sličnima ili različitima mogu se rasporediti po osnovnim karakteristikama koje su prikazane u Tablici 3.

Tablica 3. Usporedba SQL, NoSQL i NewSQL SUBP-ova [13]

Karakteristike	SQL	NoSQL	NewSQL
Relacija	Da	Ne	Da
ACID	Da	Ne	Da
SQL	Da	Ne	Da
OLTP	Ne potpuno	Podržano	Da potpuno
Skaliranje	Vertikalno	Vertikalno	Vertikalno + horizontalno
Rukovanje upitima	Da za jednostavne upite, ali ne za složene	Bolje nego SQL za složene upite	Učinkovito za složene i manje upite
Distribuirane BP	Ne	Da	Da

4. Pregled NewSQL baza podataka

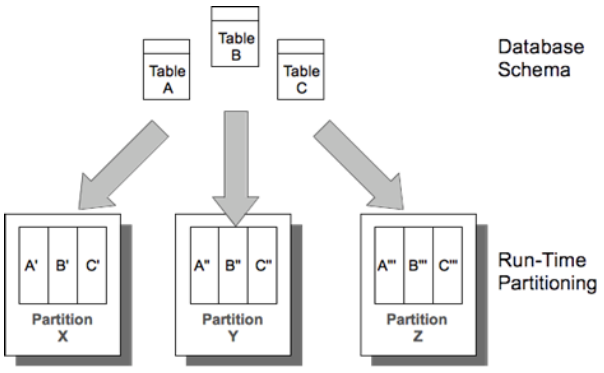
Iako su ranije navedene opće karakteristike NewSQL SUBP-ova, nije nužno da svaka navedena karakteristika vrijedi za sve NewSQL SUBP-ove. Zbog toga će biti odabrani određeni NewSQL SUBP-ovi koji će biti opisani, a zatim uspoređeni unutar tablice po određenim kriterijima koji su pronađeni na različitim izvorima.

4.1. VoltDB baza podataka

VoltDB je u memoriji (engl. *„in-memory“*), horizontalno skalabilna, usklađena sa ACID svojstvima, relacijska baza podataka koju je dizajnirao Michael Stonebraker. Ova baza podataka podržava fleksibilnost JSON-a (engl. *JavaScript Object Notation*) te SQL. VoltDB baza podatka je dizajnirana za poslovne aplikacije visokih performansi te arhitektura ove baze podataka može postići do 45 veću propusnost od trenutnih proizvoda baza podataka. Također, arhitektura omogućuje VoltDB bazama podataka lako skaliranje dodavanjem procesora u skupine kako rastu zahtjevi za količinom podataka i transakcijama. Osnovne mogućnosti ove baze podataka su:

- korištenje pohrane u memoriji kako bi se povećao protok, izbjegavajući skupi pristup disku,
- poboljšanje performansi koje se postiže podjelom cjelokupnog pristupa podacima, izbjegavajući mnoge vremenski zahtjevne funkcije tradicionalnih baza podataka (npr. zaključavanje) i
- skalabilnost, pouzdanost i velika dostupnost koje se ostvaruju podjelom u skupine i replikacijom na više poslužitelja [14].

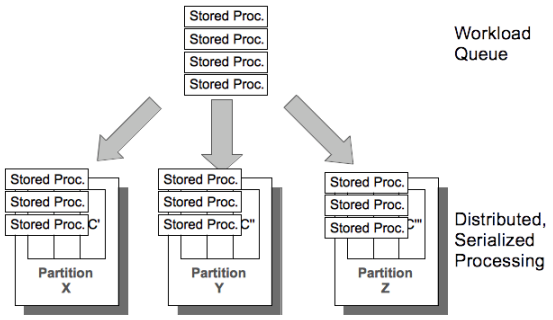
Svaka VoltDB baza podataka optimizirana je za određenu aplikaciju dijeljenjem (na segmente) tablica baze podataka (Slika 5.) i pohranjivanjem procedura koje tim tablicama pristupaju na više segmenata na jednom ili više računala domaćina za stvaranje distribuirane baze podataka. Budući da su podaci i procedure podijeljene, paralelno se može izvoditi više upita. Istodobno se svaka transakcija, zbog neovisnog djelovanje segmenata, može izvršiti do kraja bez prekomjernog zaključavanja pojedinačnih zapisa koji troše velik dio vremena za obradu u tradicionalnim bazama podataka.



Slika 5. Dijeljenje tablica u VoltDB bazama podataka [16]

U VoltDB bazama podataka svaka je pohranjena procedura definirana kao transakcija. Pohranjena procedura ili uspijeva ili se vraća potpuna, osiguravajući dosljednost baze podataka. Analizom i pred kompilacijom logike pristupa podacima u pohranjenim procedurama, VoltDB može distribuirati i podatke i obradu koja je povezana s podacima na pojedinačne skupine u klasteru. Na taj način svaka skupina sadrži jedinstveni dio podataka i obrade podataka. Svaki čvor u klasteru može podržavati više skupina.

Korištenjem serijske obrade, VoltDB osigurava dosljednost transakcija bez dodatnih troškova zaključavanja i dnevnika transakcija, dok particioniranje omogućuje bazi podataka obrađivanje više zahtjeva odjednom. Opće pravilo je, što je više procesora (a samim tim i više skupina) u klasteru, više transakcija VoltDB izvrši u sekundi, pružajući lagan, gotovo linearan put za skaliranje kapaciteta i performansi aplikacije. Kada postupak zahtijeva podatke s više skupina, jedan čvor djeluje kao koordinator i predaje potreban rad drugim čvorovima, prikuplja rezultate i dovršava zadatak. Ova koordinacija čini transakcije sa više skupina nešto sporijima od jednostrukih transakcija. Međutim, arhitektura više paralelnih skupina osigurava maksimalan protok.



Slika 6. Obrada u obliku serija u VoltDB bazama podataka [16]

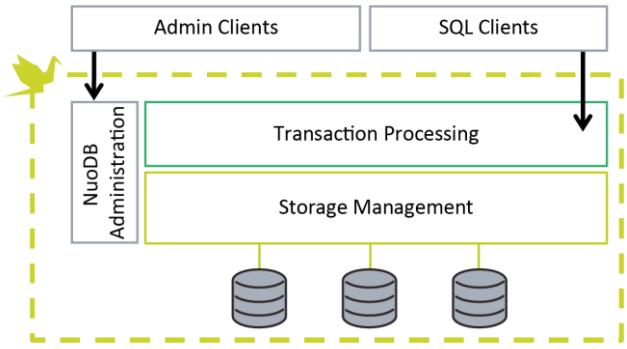
Arhitektura VoltDB-a je optimizirana za maksimalan protok. Svaka transakcija se neprekidno izvodi u vlastitoj niti, minimizirajući pojedinačno kašnjenje po transakciji (vrijeme od početka transakcije do završetka obrade). Ovo eliminira režijske troškove potrebne za zaključavanje i druge administrativne zadatke, smanjujući vrijeme u kojem zahtjevi čekaju na izvršenje. Rezultat toga je veći broj transakcija koje se mogu dovršiti u sekundi nego kod tradicionalnih baza podataka.

VoltDB baze podataka su fokusirane na segment poslovnog računarstva koji se temelji na brzim podacima, odnosno aplikacije visokih performansi koje moraju brzo obraditi velike tokove podataka. To uključuje aplikacije za financijske trgovine, telekomunikacijski tokovi, distribucijski sustavi temeljeni na senzorima, bežično povezivanje, mrežne igre, otkrivanje prijevara, itd. Neke od vodećih svjetskih tvrtki koje koriste VoltDB su: Airpush, Barclays, deltaDNA, SAKURA Internet, Huawei, Nokia Corporation, Mitsubishi Electric, itd. Međutim, VoltDB nije optimalan izbor za prikupljanje i uspoređivanje izuzetno velikih količina podataka koji se moraju ispitivati u više tablica.

4.2. NuoDB baza podataka

NuoDB je SQL orijentirani sustav za upravljanje transakcijskim bazama podataka dizajniran za distribuiranu implementaciju u oblaku kojeg je konstruirao Jim Starkey. Može se kategorizirati kao NewSQL baza podataka koja zadržava karakteristike tradicionalnih SQL baza podataka, a istovremeno uključuje značajke za podršku procesu dodavanja servera (ili čvorova) u računalnim okruženjima u oblaku. Aplikacijski programi komuniciraju s NuoDB-om preko SQL-a, a baza podataka podrazumijeva ACID garanciju radi pouzdanosti transakcija. Promatrajući klijente NuoDB baze podataka, najveći segment zauzimaju klijenti računalnog softvera (41%), zatim klijenti informacijske tehnologije i usluge (7%) i na kraju klijenti financijskih usluga (6%) [18].

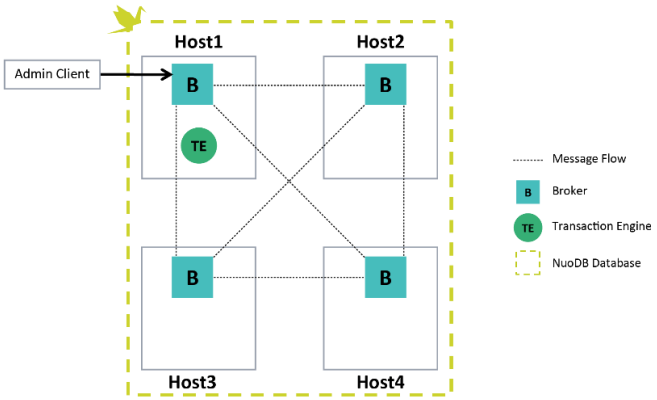
NuoDB arhitektura se razlikuje od uspostavljenih relacijskih pristupa korištenjem troslojne strukture s administrativnim (upravljačkim), transakcijskim (engl. *„Transaction Engine"*) i skladišnim slojem (engl. *„Storage Manager"*). T**ransakcijski sloj** (u nastavku TE) je odgovoran za održavanje dosljednosti i izoliranosti u izvršavanju transakcija. Samo je memorijski sloj pa je učinkovit jer ne zna koliko su podaci trajni, odnosno nema veze s trajnošću. Unutar njega se odvija proces koji omogućuje pristup aplikaciji prema jednoj NuoDB bazi podataka obradom SQL naredbi, memoriranjem podataka i koordinacijom transakcija s drugim TE-ima i upraviteljima pohrane. Uvijek je aktivan i dostupan na zahtjev pred memorije. **Skladišni sloj** (u nastavku SM) pruža uslugu osiguravanja trajnosti. Odgovoran je za stvaranje podataka trajnim (npr. zapisivanjem na disk) i pružanje pristupa podacima kada se dogodi propust u transakcijskom memoriji. Ovakav slojeviti model vrlo olakšava pokretanje sustava, proširenje kapaciteta dodavanjem hardverskih resursa na zahtjev i migraciju podataka. Minimalna NuoDB baza podataka sastoji se od dva procesa, jednog TE i jednog SM, koji se izvode na istom domaćinu. **Administrativni sloj** je zbirka procesa koji se nazivaju brokeri i izvode se na svakom domaćinu na kojem bi baza podataka mogla biti aktivna. Kada se brokeri pokrenu, pokreću se i procesi baze podataka kako bi domaćin postao vidljiv upravljačkoj okolini tj. bio dio upravljačke domene. Domena definira skup resursa dostupnih za pokretanje baza podataka i skup korisnika s dopuštenjima za upravljanje tim resursima. Kada SQL klijent želi komunicirati s TE-om, započinje povezivanje s brokerom. Broker klijenti govori s kojim TE-om uspostavlja vezu. Dakle, sistemski administrator se obraća domeni kao jednoj točki upravljanja, a to se radi putem bilo kojeg brokera. Brokeri pružaju jedinstveno mjesto za povezivanje s domenom, upravljanje i nadgledanje baza podataka i osiguravaju da sustav radi prema očekivanjima.



Slika 7. Arhitektura NuoDB baze podataka [17]

NuoDB baza podataka izgrađena je u obliku dizajna tzv. „*izdržljive distribuirane pred memorije*“ koji koristi skup pred memorija u memoriji za podršku elastičnosti u oblaku, a istovremeno osigurava da su svi podatkovni objekti sigurno pohranjeni i održavani. Također podržava više verzijsku kontrolu istodobnosti (MVCC) za otkrivanje zastoja podataka i rješavanje sukoba pristupa. Uz to, koristi elemente objektno-orijentirane i porukama-orijentirane metodologije, uglavnom komunikacijski format svaki sa svakim (engl. *peer-to-peer*).

Svim podacima u NuoDB bazama podataka se upravlja putem objekata koji se nazivaju tzv. atomi. Atomi su objekti koji sami sebe koordiniraju, a predstavljaju određene informacije (npr. podaci, indeksi ili sheme). To su dijelovi baze podataka koji mogu varirati u veličini za razliku od drugih tradicionalnih struktura na disku koji su određene veličine. Njihova veličina je određena kako bi se povećala učinkovitost komunikacije, minimizirao broj objekata koji se nalaze u pred memoriji i pojednostavilo složeno praćenje promjena. Postoji više vrsta atoma, neki od njih su: katalog atom (prati lokalne i udaljene atome, stvara nove atome, traži sadržaj atoma,…), glavni katalog atom (ima odgovornosti katalog atoma i prati ulazak i izlazak čvorova), upravitelj transakcija atom (promjena i praćenje stanja transakcija), baza podataka atom (registar atoma sheme), itd. Postoji i atom predsjednik koji koordinira pristup upisa u atome kako bi se spriječilo oštećenje podataka. Svaki puta kada TE u NuoDB bazi podataka treba izmijeniti atom (npr. prilikom pisanja podataka), zatražit će dopuštenje od atoma predsjednika. U pred memoriji se nalaze samo potrebni objekti. Nakon što objekt više nije potreban, može se ispustiti iz pred memorije. TE može u bilo kojem trenutku zatražiti objekt koji mu je potreban iz druge TE pred memorije. Ako TE nema predodređeni atom u svojoj pred memoriji, on ne sudjeluje u protokolima za ažuriranje za taj atom.

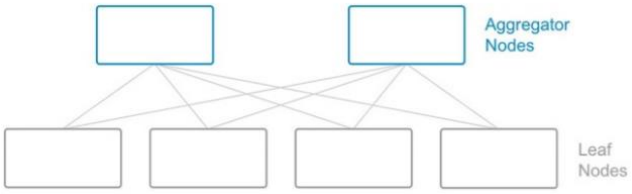


Slika 8. Pristup sistemskog administratora do brokera u NuoDB bazi podataka [17]

4.3. MemSQL baze podataka

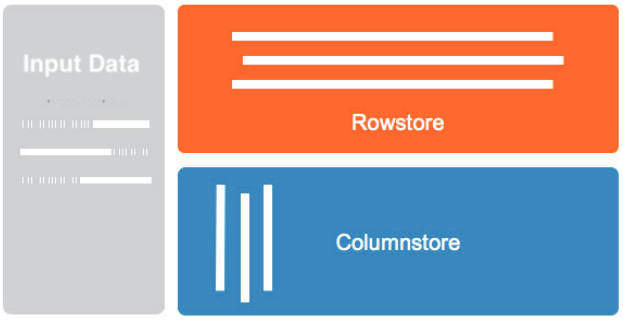
MemSQL je distribuirani, u memoriji relacijski SUBP koji je u skladu sa SQL-om. MemSQL koristi arhitekturu s dvije razine koje se sastoji od čvorova sakupljača (engl. *aggregator nodes*) i čvorova lista (engl. *leaf nodes*). Čvorovi sakupljači su usmjerivači upita koji djeluju kao prolaz kroz distribuirani sustav. Pohranjuju samo meta podatke i referentne podatke. Sakupljači inteligentno distribuiraju upite preko čvorova lista i prikupljaju rezultate koji se šalju klijentu. Povećanjem broja sakupljača poboljšavaju se operacije poput učitavanja

podataka i omogućava se istovremena obrada više zahtjeva klijenata. Čvorovi lista funkcioniraju kao čvorovi za pohranu i računanje. Podaci se automatski distribuiraju preko čvorova listova u skupine kako bi se omogućilo paralelno izvršavanje upita. Povećanje broja čvorova listova povećat će ukupni kapacitet klastera i ubrzati izvršavanje upita, posebno onih koji zahtijevaju velika skeniranja i agregiranja tablica. Dodatni čvorovi listova također omogućuju klasteru da paralelno obradi više upita. Broj raspoređenih čvorova sakupljača i čvorova listova određuje kapacitet i performanse klastera [19].



Slika 9. Arhitektura MemSQL baze podataka [19]

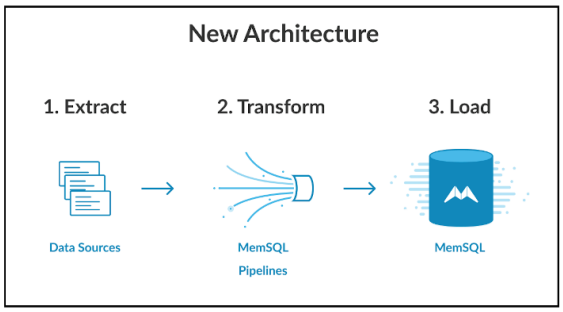
MemSQL baza podataka je distribuirana relacijska baza podataka koja u velikoj mjeri rukuje transakcijama i analitikom u stvarnom vremenu. Dostupna je putem standardnih SQL upravljačkih programa i podržava SQL sintaksu, uključujući spajanja, filtriranja, agregacije i funkcije grupiranja. Ova baza podataka koristi vodoravno skaliranje na instancama u oblaku ili hardveru te tako pruža veliku propusnost na širokom rasponu platformi. Sadrži spremište redova (engl. *rowstore*) u memoriji i spremište stupaca (engl. *columnstore*) na disku za upravljanje istodobnim operativnim i analitičkim radnim okruženjima. Spremište redova u memoriji pruža optimalne performanse u stvarnom vremenu za transakcijska radna opterećenja. Skladište na temelju diska najbolje je za analitička opterećenja u velikim povijesnim skupovima podataka. Ovakve baze podataka mogu učitavati podatke stalno i skupno iz različitih izvora. Popularni izvori učitavanju uključuju: datoteke, klaster i spremišta u oblaku (npr. Amazona S3, HDFS,…)..



Slika 10. Spremište redova i spremište stupaca u MemSQL bazi podataka [20]

MemSQL sadrži tehnologiju za unos podataka nazvanu MemSQL cjevovodi (engl. *MemSQL Pipelines*) kroz koju prolazi velika količina podataka velikom propusnošću u bazu podataka. MemSQL cjevovodi je ugrađena sposobnost za upotrebu koja izdvaja, transformira i učitava vanjske podatke koristeći izvore. Ta sposobnost je idealna za scenarije u kojim se podaci iz podržanih izvora moraju unositi i obrađivati u stvarnom vremenu. Kao što prikazuje Slika 11., to čini MemSQL cjevovode dobrom alternativom za posrednički program za obične

ETL[8] (engl. *Extract, transform, loan*) operacije koje se moraju izvršiti što je brže moguće, eliminirajući na taj način tradicionalne dugotrajne procese.



Slika 11. Korištenje MemSQL cjevovoda u MemSQL bazi podataka [20]

Podržavanje istodobnosti za istovremene korisnike još je jedna sposobnost koju ima MemSQL. Alat za optimizaciju distribuiranih upita ravnomjerno dijeli radno opterećenje obrade kako bi povećao učinkovitost upotrebe procesora. Planovi upita sastavljaju se u strojni kod i pred memoriraju kako bi se ubrzala kasnija izvršavanja. Ključna značajka ovih sastavljenih planova je da oni ne navode vrijednost parametra unaprijed. To omogućuje MemSQL-u da zamijeni vrijednost na zahtjev, što omogućuje brzo izvršavanje sljedećih upita iste strukture. Uz korištenje MVCC-a i podatkovne strukture bez zaključavanja, podaci u MemSQL ostaju vrlo dostupni, čak i usred velikog broja istodobnih čitanja i pisanja u MemSQL bazi podataka.

4.4. Usporedba VoltDB, NuoDB i MemSQL baze podataka

Kriteriji koji su korišteni za usporedbu VoltDB, NuoDB i MemSQL SUBP-ova podijeljeni su u Tablici 4. na dva dijela kojima između kojih je granica crvena horizontalna linija. Prvi dio označuje usporedbu po kriterijima osnovnih karakteristika NewSQL SUBP-ova koje su opisane u poglavlju 3.2. (pohrana u memoriji, particioniranje, kontrola konkurentnosti, replikacija, sekundarni indeksi, oporavak od pada sustava), a drugi dio se odnosi na postojeće kriterije sa odabranih izvora na [18][19][20][21][22].

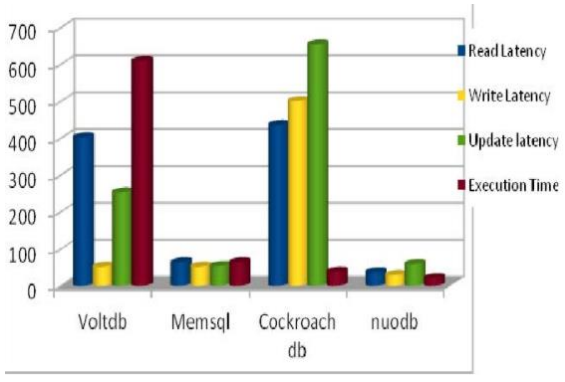
Tablica 4. Usporedba VoltDB, NuoDB i MemSQL SUBP-ova

	VoltDB	**NuoDB**	**MemSQL**
Pohrana u memoriji	Da	Da	Da
Particioniranje	Da (klijenti biraju broj poslužitelja)	Da (nije vidljivo klijentima jer key-value spremnik može razdvajati podatke)	Da

[8] operacije s procesima izdvajanja, transformiranja i učitavanja podataka

Kontrola konkurentnosti	Da	Da, MVCC	Da, MVCC
Replikacija	Da (istovremeno na svim replikama)	Da (replikacija izvor-replika i replikacija s više izvora	Da (istovremeno na svim replikama)
Sekundarni indeksi	Da	Da	Da
Oporavak od pada sustava	Da	Da	Da
Dostupnost	Otvoreni kod	Zatvoreni kod	Zatvoreni kod
Godina razvitka	2010.	2013.	2013.
Usklađenost sa SQL-om	Da (nema vanjskih ključeva)	Da	Da (nema okidača i vanjskih ključeva)
Shema podataka	Da	Da	Da
Skalabilnost	Horizontalna	Horizontalna	Horizontalna
Provjera autentičnosti	Da (korisnici definirani u datoteci za raspoređivanje koja mora biti kopirana na svaki čvor)	Da	Da
Provjera uloga	Da (uloge definirane u shemi a svaka pohranjena procedura određuje koja uloga ju može izvršiti)	Da (SQL)	Da
Tko koristi?	Nokia, Ericsson, Flipkart, hp, Huwei, T-Mobile, OrangeTM	UAExchange, Forbes, Kodiak, Red Hat, MoreMedical	Intel, Microsoft Azure, Uber, Pinterest, Dell, SAMSUNG

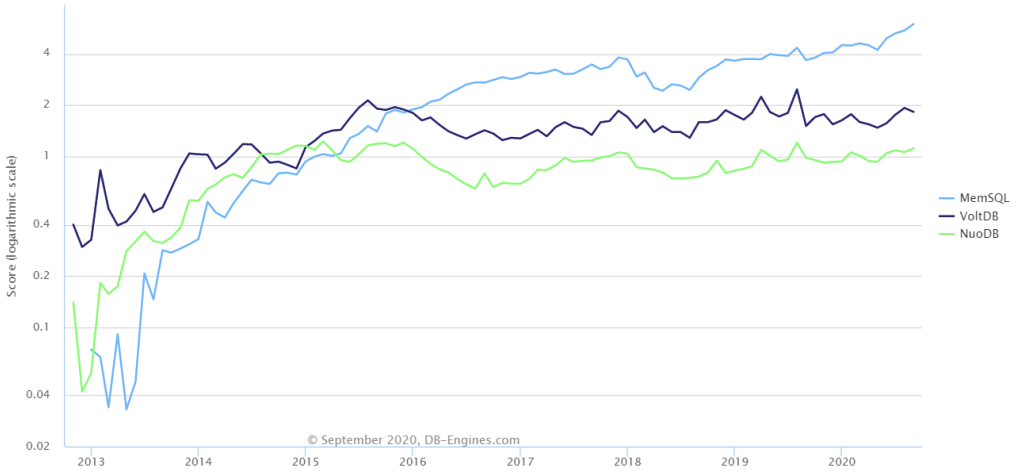
Rad [23] sadrži istraživački rad iz 2017. godine u kojemu su testirani VoltDB, MemSQL, CockroachDB i NuoDB SUBP-ovi te su dobiveni podaci za brzinu čitanja, pisanja, ažuriranja i vrijeme izvršavanja akcija u tim bazama podataka. Dobiveni podaci su preneseni na grafikon na kojemu je uočljivo kako MemSQL i NuoDB imaju slične brzine i vrijeme, VoltDB i CockroachDB imaju sličnu brzinu čitanja dok je brzina pisanja i ažuriranja gotovo dvostruko veća kod CockroachDB-a u odnosu na VoltDB, ali zato VoltDB ima čak do nekoliko puta veće vrijeme izvršavanja od CockroachDB-a.



Slika 12. Grafikon s brzinama za VoltDB, MemSQL, CockroachDB i NuoDB [23]

VoltDB, NuoDB i MemSQL se mogu određenom metodom poredati po popularnosti. Metoda za popularnost obuhvaća parametre za vrednovanje SUBP-a: broj spominjanja SUBP-ova mjeren kao broj rezultata u upitima tražilica (Google i Bing), opći interes za sustav (Google trendovi), učestalost tehničkih rasprava o sustavu, broj ponuda za posao u kojima se spominje, broj profila u profesionalnim mrežama u kojima se spominje i relevantnost u društvenim mrežama (broj Twitter tweetova). Svi navedeni parametri imaju određen maksimalan broj bodova koji je moguće ostvariti, a zatim se zbrajanjem ostvarenih bodova po parametrima dobije konačan broj bodova za pojedini SUBP. Kako bi se eliminirali učinci uzrokovani promjenom količina samih izvora podataka, ocjena popularnosti je uvijek relativna vrijednosti te ju jotrebno tumačiti samo u usporedbi s drugim sustavima [24].

Slika 13. prikazuje stanje po bodovima za VoltDB, NuoDB i MemSQL od njihovog nastajanja 2012. ili 2013. godine do danas. Pri samoj pojavi, VoltDB je imao najviše bodova, oko 0.4, dok je najslabije rangirani bio MemSQL sa oko 0.07 bodova, a NuoDB je imao 0.15 bodova. Nakon rasta bodova za sva tri sustava, 2015. godine dolazi do promjene redoslijeda popularnosti. Tada započinje rast MemSQL SUBP-ova, pad NuoDB SUBP-ova, a VoltDB uz najveće oscilacije postaje središnji. Iako su sva tri SUBP-a imala razdoblja pada i rasta po broju bodova, danas je MemSQL je postao SUBP sa najboljim brojem bodova koji iznosi oko 4, VoltDB je drugi po redu te ima oko 1.8 bodova, a NuoDB ima oko 0.9 bodova.



Slika 13. Popularnost VoltDB, NuoDB i MemSQL-a [25]

Prilikom kreiranja tablica u VoltDB i NuoDB bazama podataka se koristi naredba *create table*, dok se u MemSQL bazi podataka koristi naredba *create* kao što je vidljivo u tablici 5. Nakon toga na slikama 14., 15. i 16. prikazan je primjer kreiranja tablica u tim trima bazama podataka, a nakon toga su odabrani i pokazani primjeri upita u istima.

Tablica 5. Kreiranje tablica u VoltDB, NuoDB i MemSQL SUBP-ovima [26][27][28]

	Sintaksa
VoltDB	CREATE TABLE *table-name* [export-definition | migration-definition] column-definition [,...] [, constraint-definition [,...]]) [ttl-definition] ;
NuoDB	CREATE TABLE [IF NOT EXISTS] [schema_name.]table_name (column_name { data_type | domain_name } [column_constraints] [, ...] [, table_constraints])
MemSQL	CREATE [REFERENCE | TEMPORARY | GLOBAL TEMPORARY] TABLE [IF NOT EXISTS] tbl_name (create_definition,...) [table_options] [[AS] SELECT ...]

U primjeru kreiranja tablice za VoltDB bazu podataka kreirana je tablica *Inventory* koja se sastoji od 5 stupaca (*Company, ProductID, Price, Category, Description*). Prvi stupac *Company* ne smije biti prazan (engl. *null*) što je važno jer se koristi kao particijski stupac u izrazu *partition table*. Također, taj se stupac koristi u ograničenju *primary key*. Dakle, važno je uključiti particijski stupac u bilo koji potpuno jedinstveni indeks za particionirane tablice.

```
CREATE TABLE Inventory (
    Company VARCHAR(32) NOT NULL,
    ProductID BIGINT NOT NULL,
    Price DECIMAL,
    Category VARCHAR(32),
    Description VARCHAR(256),
    PRIMARY KEY (Company, ProductID)
);
PARTITION TABLE Inventory ON COLUMN Company;
```

Slika 14. Primjer kreiranja tablice u VoltDB bazi podataka [26]

Za NuoDB bazu podataka na slici 15. kreirana je tablica koja se naziva *hockey_fans*, a sadrži osam stupaca (*id, name, address, city, zip_code, teamid, gender i phone*). U NuoDB bazi podataka za particioniranje se može koristiti samo jedan stupac. Tip particijskog stupca može biti bilo koji podržani SQL tip osim *blob* ili *clob*. Ako tablica ima primarni ključ ili jedinstveni indeks, particijski stupac se mora dodati kao stupac za indeks kako bi se osigurala jedinstvenost na više paricija. Izuzetak je naveden u primjeru na slici 15. gdje je particijski stupac definiran kao uvijek generiran kao identitet (engl. *generated always as identitiy*). U ovom slučaju, ovaj stupac ne bi trebao biti dodan primarnom ključu i/ili jedinstvenom indeksu.

```
CREATE TABLE hockey_fans
(id         INTEGER     GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
 name       STRING      NOT NULL,
 address    STRING      NULL,
 city       STRING      NOT NULL,
 zip_code   zipcodes,
 teamid     VARCHAR(3)  DEFAULT 'BOS',
 gender     CHAR(1)     CHECK (gender IN ('M','F') ),
 phone      STRING
);
```

Slika 15. Primjer kreiranja tablice u NuoDB bazi podataka [27]

Slika 16. prikazuje primjer kreiranja tablice u MemSQL bazi podataka gdje je kreirana tablica *transaction* sa jedanaest stupaca: *id, explanation, shares, share_price, total_amount, transaction_date, dividend_exdate, misc_expenses, country_abbreviation, correction_date i settlement_date*. Stupac *total_amount* se dobije na način da se pomnože vrijednosti iz stupca *shares* i *share_price*.

```
memsql> CREATE TABLE transaction(
     ->     id BIGINT NOT NULL,
     ->     explanation VARCHAR(70),
     ->     shares DECIMAL(18, 2),
     ->     share_price DECIMAL(18, 2),
     ->     total_amount as shares * share_price PERSISTED DECIMAL(18,2),
     ->     transaction_date DATE,
     ->     dividend_exdate DATE,
     ->     misc_expenses DECIMAL(18, 2),
     ->     country_abbreviation CHAR(6),
     ->     correction_date DATE,
     ->     settlement_date DATE
```

Slika 16. Primjer kreiranja tablice u MemSQL bazi podataka [28]

Unutar tablice 6. nalaze se primjeri određenih upita u VoltDB, NuoDB i MemSQL bazama podataka. Upit u VoltDB bazi podataka dohvaća odabrane stupce za dvije tablice (*employee* i *compensation*) odjednom koje imaju jednake vrijednosti u stupcu *employee_id* koristeći unutarnje spajanje, a stupci su poredani po stupci *lastaname*. U NuoDB bazi podataka odabran je primjer sa rezultatima koji se dobiju nakon što je upit izvršen. U tom primjeru se iz tablice *scoring* izdvajaju stupci *playerid*, *year* i *stint* gdje je vrijednost stupca *year* jednaka 2009, stupac *playerid* počinje slovima „boy“ te su rezultati poredani po stupci *playerid*. Odabrani upit za MemSQL bazu podataka prikazuje stupce *count(*)*, *user_name* i *page_url* iz tablica *clicks*, *users* i *pages* gdje je jednak stupac *user_id* iz tablica *clicks* i *users* te jednak stupac *page_id* iz tablica *pages* i *clicks*. Stupac *count(*)* vraća ukupan broj redaka iz baze podataka koji odgovaraju upitu. Mogućnost *group by* u upitu grupira retke koje imaju iste vrijednosti u sažete retke. Također, s *order by* opcijom su rezultati poredani silazno po stupcu *counts*$^{(*)}$.

Tablica 6. Primjeri upita u VoltDB, NuoDB i MemSQL bazi podataka [29][39][31]

	Primjeri upita

VoltDB	`SELECT lastname, firstname, salary`
	`FROM employee AS e, compensation AS c`
	`WHERE e.employee_id = c.employee_id`
	`ORDER BY lastname DESC;`

NuoDB	`SELECT ALL playerid,year,stint FROM scoring`
	`WHERE year = 2009 AND playerid LIKE 'boy%' ORDER BY playerid;`
	`PLAYERID YEAR STINT`
	`--------- ----- ------`
	`boychjo01 2009 1`
	`boychza01 2009 1`
	`boyddu01 2009 1`
	`boyddu01 2009 2`
	`boyesbr01 2009 1`
	`boylebr01 2009 1`
	`boyleda01 2009 1`
	`boyntni01 2009 1`
	`boyntni01 2009 2`

MemSQL	`memsql> SELECT COUNT(*), user_name, page_url from clicks, users, pages`
	`-> WHERE clicks.user_id = users.user_id AND pages.page_id = clicks.page_id`
	`-> GROUP BY users.user_id, pages.page_id`
	`-> ORDER BY COUNT(*) DESC;`
	`+- ---------+- ----------+- ----------------------------+`
	`| COUNT(*) | user_name | page_url |`
	`+- ---------+- ----------+- ----------------------------+`
	`| 5 | jake | memsql.com |`
	`| 2 | john | http://www.memsql.com/download |`
	`| 1 | jake | developers.memsql.com |`
	`| 1 | jake | memsql.com |`
	`| 1 | jake | http://www.memsql.com/download |`
	`+- ---------+- ----------+- ----------------------------+`
	`5 rows in set (0.00 sec)`

5. Zaključak

Glavna tema ovoga rada su NewSQL sustavi za upravljanje bazama podataka uz dodatan prikaz općenitih informacija o NewSQL bazama podataka i relacijskim bazama podataka. Također, cilj je bio i testirati odabrane NewSQL SUBP-ove na vlastitim primjerima po vlastitim kriterijima, ali zbog nedostatka računala s dobrim kapacitetom obrade i pohrane cilj se promijenio pa su odabrani SUBP-ovi uspoređeni na temelju odabranih izvora na kojima su klijenti recenzirali alate.

Za početak se treba usredotočiti na potrebu za pojavom NewSQL baza podataka i njihovom značaju u svijetu baza podataka. Nastale su zbog potrebe spajanja relacijskih tehnologija koje koriste SQL s distribuiranim bazama podataka NoSQL sustava, ali nisu dizajnirane kako bi zamijenile relacijske SQL modele ili NoSQL. Izgrađene su kao alternativa za obradu podataka prikrivajući nedostatke relacijskog i ne relacijskog sustava baza podataka. Krajnji cilj NewSQL-a je pružiti dostupno rješenje za rukovanje modernim podacima ne ugrožavajući dosljednost podataka i mogućnost brzih transakcija. Međutim, zasad još uvijek nisu postale dominantne u svijetu baza podataka. Jedan od mogućih razloga tomu je relativno kratak period postojanja što rezultira brojnim nepoznanicama i nesigurnostima klijenata pri odlukama o korištenju NewSQL klase baza podataka. Nadalje, za prelazak na korištenje NewSQL SUBP-ova potrebna su financijska sredstva koja klijenti ne žele ulagati radi skromnih iskustava u radu s ovakvim bazama podataka koji ne obećavaju siguran prelazak bez kasnijih poteškoća. Osim manjka iskustva još jedan korak unazad od dominacije NewSQL baza podataka su neki NoSQL sustavi koji smanjuju svoje nedostatke tako što su počeli koristiti neke zahtjeve ACID garancije.

Ipak, ako se klijenti odluče za NewSQL SUBP, vrlo vjerojatno će pri odabiru imati u vidu VoltDB, NuoDB ili MemSQL. Svi oni imaju osnovne karakteristike NewSQL baza podataka, ali se neke karakteristike razlikuju po načinu izvršavanja. Primjerice, dok NuoDB i MemSQL koriste MVCC kontrolu konkurentnosti, VoltDB korištenjem vremenskih oznaka izvršava jednu po jednu transakciju uz kontrolu konkurentnosti. Za svaki NewSQL SUBP moguće je kroz njegovu arhitekturu i način rada shvatiti kako ostvaruje svoje karakteristike. Trendovi s NewSQL bazama podataka brzo se mijenjaju te zadnjih nekoliko godina, od sva tri, najviše dominira MemSQL koji je prilikom pojave bio najmanje popularan. Dakle, NewSQL SUBP-ovi još uvijek nisu jako popularni niti korišteni od strane klijenata, ali ostaje ključna nepoznanica je li bi to bilo drugačije da su se pojavili ranije i je li bi u tom slučaju mogli konkurirati tradicionalnim SQL i NoSQL bazama podataka.

Literatura

[1] Wiederhold Gio, File Organization for Database Design. McGraw-Hill, nepoznato, 1977.

[2] Manger Robert, Baze podataka. Element, Zagreb, 2011.

[3] Varga Mladen, Baze podataka: konceptualno, logičko i fizičko modeliranje podataka. DRIP-Društvo za razvoj informacijske pismenosti, Zagreb, 1994.

[4] Pavlić Mile, Oblikovanje baze podataka. Odjel za informatiku Sveučilišta, Rijeka, 2011.

[5] Kraljević Goran, Baze podataka. Nepoznato, nepoznato, 2019.

[6] Haerder Theo, Reuter Andreas, Principles od Transaction-Oriented Database Recovery. ACM Computing Surveys, nepoznato 1983.

[7] System Design: Database Transactions & ACID Compliance, Medium. Preuzeto 03.09.2020. sa: https://medium.com/@pulkitent/system-design-database-transactions-acid-part-1-45de4c350ff2

[8] Harrison Guy, Next Generation Databases. Apress, Berkeley (Kalifornija), 2015.

[9] NewSQL, Wikipedia. Preuzeto 03.08.2020. sa: https://en.wikipedia.org/wiki/NewSQL

[10] Stonebraker Michael, Madden Samuel, Abadi Daniel, Harizopoulos Stavros, Hachem Nabil, Helland Pat, The End o fan Architectural Era (It's Time for a Complete Rewrite). ACM, Beč (Austrija), 2007.

[11] NewSQL: The Future of Databases?, SlideShare. Preuzeto 03.09.2020. sa: https://www.slideshare.net/ibelmopan/newsql-the-future-of-databases

[12] Pavlo Andrew, Aslett Matthew, What's Really New with NewSQL?. ACM SIGMOD Record, nepoznato, 2016.

[13] SQL vs NoSQL vs NewSQL: The Full Comparison, Xenonstack. Preuzeto 02.09.2020. sa: https://www.xenonstack.com/blog/sql-vs-nosql-vs-newsql/

[14] Mileusnić Vatroslav, Modeliranje i dizajn baze podataka (diplomski rad), Fakultet organizacije i informatike, Zagreb, 2011.

[15] What is NewSQL? What are its advantages and disadvantages compared to NoSQL?, Quora. Preuzeto 07.08.2020. sa https://www.quora.com/What-is-NewSQL-What-are-its-advantages-and-disadvantages-compared-to-NoSQL

[16] Using VoltDB, VoltDB Documentation. Preuzeto 03.09.2020. sa: https://docs.voltdb.com/UsingVoltDB/

[17] Quick Dive Into NuoDB Architecture, NuoDB. Preuzeto 03.09.2020. sa: https://nuodb.com/blog/quick-dive-nuodb-architecture

[18] Companies using NuoDB, Enlyft. Preuzeto 03.09.2020. sa: https://enlyft.com/tech/products/nuodb

[19] Olivierira Joao, Bernardino Jorge, NewSQL Databases – MemSQL i VoltDB Experimental Evaluation. Proceedings of the 9th International Joint Conference on Knowledge Discovery, SCITEPRESS – Science and Technology Publications, nepoznato, 2017.

[20] How MemSQL Works – At a Glance, MemSQL. Preuzeto 09.09.2020. sa: https://www.memsql.com/blog/how-memsql-works/

[21] NewSQL: pregledni rad, Scribd. Preuzeto 10.09.2020. sa: https://www.scribd.com/document/333863618/NewSQL

[22] System Properties Comparison MemSQL vs. NuoDB vs. VoltDB, db-engines. Preuzeto 10.09.2020. sa: https://db-engines.com/en/system/MemSQL%3BNuoDB%3BVoltDB

[23] Kaur Karambir, Sachdeva Monika, Performace evaluation of NewSQL databases. International Conference on Inventive Systems and Control (ICISC), IEEE, India, 2017.

[24] Method of calculating the scores of the DB-Engines Ranking, db-engines. Preuzeto 10.09.2020. sa: https://db-engines.com/en/ranking_definition

[25] Trend of MemSQL vs. NuoDB vs. VoltDB Popularity, db-engines. Preuzeto 10.09.2020. sa: https://db-engines.com/en/ranking_trend/system/MemSQL%3BNuoDB%3BVoltDB

[26] Create table, VoltDB Documentation. Preuzeto 14.09.2020. sa: https://docs.voltdb.com/UsingVoltDB/ddlref_createtable.php

[27] Create table, NuoDB Docs. Preuzeto 14.09.2020. sa: https://doc.nuodb.com/nuodb/4.0.x/reference-information/sql-reference-information/sql-statements/create-table/

[28] Create table, MemSQL Docs. Preuzeto 14.09.2020. sa: https://docs.memsql.com/v7.1/reference/sql-reference/data-definition-language-ddl/create-table/

[29] Select, VoltDB Documentation. Preuzeto 15.09.2020. sa: https://docs.voltdb.com/UsingVoltDB/sqlref_select.php

[30] Select, NuoDB Docs. Preuzeto 14.09.2020. sa: https://doc.nuodb.com/nuodb/4.0.x/reference-information/sql-reference-information/sql-statements/select/

[31] Select, MemSQL Docs. Preuzeto 14.09.2020. sa: https://docs.memsql.com/v7.1/reference/sql-reference/data-manipulation-language-dml/select/

Popis tablica

Tablica 1. Najpopularniji SUBP-ovi u lipnju 2020. 2
Tablica 2. Osnovne prednosti i nedostaci NewSQL SUBP-ova 13
Tablica 3. Usporedba SQL, NoSQL i NewSQL SUBP-ova [13] 14
Tablica 4. Usporedba VoltDB, NuoDB i MemSQL SUBP-ova 20
Tablica 5. Kreiranje tablica u VoltDB, NuoDB i MemSQL SUBP-ovima [26][27][28] 23
Tablica 6. Primjeri upita u VoltDB, NuoDB i MemSQL bazi podataka [29][39][31] 25

Popis slika

Slika 1. Sustav za upravljanje bazom podataka 2
Slika 2. Strukturalni koncepti proizvoljne relacije KNJIGE 4
Slika 3. ACID terminologija [7] 5
Slika 4. Razvoj baza podataka [11] 7
Slika 5. Dijeljenje tablica u VoltDB bazama podataka [16] 15
Slika 6. Obrada u obliku serija u VoltDB bazama podataka [16] 16
Slika 7. Arhitektura NuoDB baze podataka [17] 17
Slika 8. Pristup sistemskog administratora do brokera u NuoDB bazi podataka [17] 18
Slika 9. Arhitektura MemSQL baze podataka [19] 19
Slika 10. Spremište redova i spremište stupaca u MemSQL bazi podataka [20] 19
Slika 11. Korištenje MemSQL cjevovoda u MemSQL bazi podataka [20] 20
Slika 12. Grafikon s brzinama za VoltDB, MemSQL, CockroachDB i NuoDB [23] 21
Slika 13. Popularnost VoltDB, NuoDB i MemSQL-a [25] 22
Slika 14. Primjer kreiranja tablice u VoltDB bazi podataka [26] 23
Slika 15. Primjer kreiranja tablice u NuoDB bazi podataka [27] 24
Slika 16. Primjer kreiranja tablice u MemSQL bazi podataka [28] 24