# Usporedba performansi relacijskog sustava i sustava specijaliziranog za pohranu vremenskih serija u radu s podacima za dijagnostiku vibracijskog stanja rotirajućih strojeva

**Ljubas, Dora**

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 388

# USPOREDBA PERFORMANSI RELACIJSKOG SUSTAVA I SUSTAVA SPECIJALIZIRANOG ZA POHRANU VREMENSKIH SERIJA U RADU S PODACIMA ZA DIJAGNOSTIKU VIBRACIJSKOG STANJA ROTIRAJUĆIH STROJEVA

Dora Ljubas

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 388

# USPOREDBA PERFORMANSI RELACIJSKOG SUSTAVA I SUSTAVA SPECIJALIZIRANOG ZA POHRANU VREMENSKIH SERIJA U RADU S PODACIMA ZA DIJAGNOSTIKU VIBRACIJSKOG STANJA ROTIRAJUĆIH STROJEVA

Dora Ljubas

Zagreb, lipanj 2024.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 4. ožujka 2024.

# DIPLOMSKI ZADATAK br. 388

| | |
|---|---|
| Pristupnica: | **Dora Ljubas (0036522948)** |
| Studij: | Računarstvo |
| Profil: | Programsko inženjerstvo i informacijski sustavi |
| Mentorica: | prof. dr. sc. Ljiljana Brkić |

Zadatak: **Usporedba performansi relacijskog sustava i sustava specijaliziranog za pohranu vremenskih serija u radu s podacima za dijagnostiku vibracijskog stanja rotirajućih strojeva**

Opis zadatka:

Vremenske serije su skupovi podataka koji se generiraju tijekom vremena, pri čemu su vremenske oznake integralni dio skupa podataka. Primjeri vremenskih serija uključuju meteorološke podatke (npr. temperatura, brzina vjetra itd.), financijske podatke (npr. cijene dionica, tečajevi valuta itd.), podatke dobivene od strane raznih senzora itd. Količina podataka u vremenskim serijama može biti vrlo velika, posebno ako se podaci prikupljaju visokom frekvencijom (npr. svake sekunde, minute itd.) tijekom dugog vremenskog razdoblja. Specijalizirani sustavi za pohranu i upravljanje vremenskim serijama (engl. Time Series DBMS) su razvijeni kako bi se poboljšala učinkovitost i skalabilnost obrade vremenskih podataka. Proučiti i opisati trenutno postojeća rješenja prikladna za pohranu i upravljanje vremenskim serijama. Korištenje vremenskih serija podataka za dijagnostiku vibracijskog stanja rotirajućih strojeva je ključno za praćenje njihovih performansi. Analizom vremenskih serija vibracijskih podataka mogu se identificirati anomalije u radu strojeva koje mogu ukazivati na nepravilnosti ili potencijalne kvarove. Trenutno rješenje firme Veski, CoDiS, koje se već dugi niz godina koristi za nadzor vibracija i količina prikupljenih i pohranjenih podataka te način njihove pohrane, pruža nezadovoljavajuće performanse nekih dijelova sustava. Većina podataka koji se prikupljaju ima obilježja vremenskih serija. Podaci su pohranjeni u relacijskom sustavu za upravljanje bazom podataka MySQL. Potrebno je proučiti postojeći informacijski sustav CoDiS firme Veski, uočiti njegova ograničenja kako u modelu podataka tako i u programskoj potpori te predložiti i implementirati redizajnirani prototip sustava. U okviru redizajna modela podataka potrebno je uzeti u obzir sustave prilagođene za pohranu vremenskih serija kao što je npr. InfluxDB. Odluku o novom modelu podataka i sustavu za njegovu pohranu donijeti temeljem testiranja čije parametre i provedbu je potrebno osmisliti tijekom izrade rada. Za novi model podataka je potom potrebno implementirati programsku potporu koja će korisniku omogućiti pregled statističkih podataka važnih za nadzor vibracijskog stanja rotirajućih strojeva.

Rok za predaju rada: 28. lipnja 2024.

*Zahvaljujem firmi Veski d.o.o. na omogućenom uvidu u njihov informacijski sustav za praćenje vibracija rotirajućih strojeva te pristupu podacima nad kojima je provedeno testiranje. Posebna zahvala gospodi F. Tonkoviću i D. Cerinskom na pružanju tehničke potpore i izlaženju u susret oko raznih potreba tijekom izrade rada.*

*Zahvaljujem dr. sc. Bojanu Francu na povezivanju s firmom Veski d.o.o. i korisnim savjetima.*

*Posebno zahvaljujem mentorici izv. prof. dr. sc. Ljiljani Brkić na svim smjernicama, vodstvu i pruženoj potpori od preddiplomskog studija, kroz cijeli diplomski studij, te tijekom izrade ovog diplomskog rada. Njezini stručni savjeti, pristupačnost i ukazano povjerenje bili su ključni za kvalitetnu realizaciju ovog rada.*

## Sadržaj

# Uvod

Vremenske serije predstavljaju skupove podataka generirane tijekom vremena, gdje su vremenske oznake integralni dio podataka. Ovi podaci mogu obuhvaćati raznovrsne informacije, od meteoroloških mjerenja poput temperature i brzine vjetra, preko financijskih podataka kao što su cijene dionica i tečajevi valuta, do podataka prikupljenih putem raznih senzora. S obzirom na visoku frekvenciju prikupljanja i duga razdoblja praćenja, količina podataka u vremenskim serijama može biti izuzetno velika.

Specijalizirani sustavi za pohranu i upravljanje vremenskim serijama razvijeni su kako bi se poboljšala učinkovitost i skalabilnost obrade ovih podataka. Ovi sustavi nude napredne značajke optimizirane za rad s vremenskim podacima, omogućavajući brže pretraživanje, analizu i manipulaciju podacima.

U kontekstu dijagnostike vibracijskog stanja rotirajućih strojeva, korištenje vremenskih serija podataka postaje ključno za praćenje njihovih performansi. Analizom vibracijskih podataka mogu se identificirati anomalije koje ukazuju na nepravilnosti ili potencijalne kvarove, omogućavajući preventivno održavanje i smanjenje rizika od skupih zastoja.

Trenutno rješenje firme Veski, *Asset Analytics Toolkit* (skraćeno AAnT), koristi se za nadzor vibracija rotirajućih strojeva. Međutim, sustav je razvijen koristeći danas zastarjelu verziju MySQL relacijske baze podataka. Stoga. cilj ovog diplomskog rada je proučiti postojeći informacijski sustav AAnT, identificirati njegova ograničenja te predložiti i implementirati redizajnirani prototip sustava koristeći specijalizirane sustave za pohranu vremenskih serija, poput InfluxDB-a. Kroz temeljitu analizu i testiranje, donijet će se odluka o novom modelu podataka i sustavu za njegovu pohranu. Na temelju rezultata testiranja, implementirat će se programska potpora koja će korisnicima omogućiti učinkovit pregled statističkih podataka važnih za nadzor vibracijskog stanja rotirajućih strojeva.

# 1. Opis postojećeg sustava

## 1.1. Općenito o sustavu AAnT

Sustav AAnT (skraćenica od engleskog naziva *Asset Analytics Toolkit*) predstavlja alat za nadzor i dijagnostiku vibracija rotirajućih strojeva poput hidroagregata, pumpi, motora i slično. Razvijen je unutar tvrtke Veski 2019. godine. Preteći sustav AAnT-a naziva se CoDiS (skraćenica od engleskog naziva *Computer Diagnostic System*). CoDiS pruža sveobuhvatno rješenje od senzora do korisničkog sučelja, što uključuje prikupljanje, pohranu, obradu i vizualizaciju podataka. Motivacija za razvoj takvog sustava bila je otkrivanje grešaka u radu strojeva u ranoj fazi jer je tada popravak najisplativiji i najjednostavniji. CoDiS je razvijen u LabVIEW-u, visokorazinskom programskom jeziku i razvojnom alatu koji se često koristi zbog svoje jednostavnosti za izradu dijagnostičkih sustava. Međutim, s vremenom je korisničko sučelje razvijeno u LabVIEW-u postalo zastarjelo, pa je razvijen AAnt koji nudi moderno web korisničko sučelje. AAnT sustav sastoji se od 2 modula: AAnT i AAnT Cloud. AAnT prikazuje uživo stanje sustava, dok AAnT Cloud omogućuje statističku analizu podataka.

### Vremenske serije u kontekstu AAnT-a

Podaci vremenske serije (engl. *time series data*), koji se također nazivaju podacima s vremenskim žigom (engl. *time-stamped data*), niz su podatkovnih točaka određenih vremenskim redoslijedom. Takvi podaci obično predstavljaju uzastopna mjerenja iz istog izvora tijekom fiksnog vremenskog intervala i koriste se za praćenje promjena tijekom vremena.

Unutar AAnT sustava podaci vremenskih serija pojavljuju se u 2 slučaja: kao čisti neobrađeni podaci sa senzora te izračunati vektori stanja. U ovom radu fokus je na vektore stanja. Izračun vektora stanja za signal provodi se na temelju podataka s jednog ili više senzora.

## Osnovne funkcionalnosti aplikacije

AAnT Home, početna stranica, pruža pregled svih jedinica unutar postrojenja.

Odabirom jedinice otvara se *live* prikaz njenog stanja. To uključuje sliku oblika jedinice s prikazom trenutnog stanja signala, prikaz posljednjih alarma, linijski graf koji prikazuje sve izmjerene vrijednosti u odabranom intervalu…

Korisnik može odabrati željene signale i vremenski interval. Ako interval pokriva dulji period, podaci se agregiraju jer u tom slučaju nije moguće prikazati sve podatke. Praćenje *live* podataka omogućuje odgovor u stvarnom vremenu na neželjene vibracije i automatsko gašenje jedinice.

Osim prikaza vektora stanja pojedinih signala, AAnT nudi pregled sirovih valnih oblika (engl. *raw wave forms*), čiji je prikaz povezan sa senzorom s kojeg dolaze, a funkcionira na isti način kao i za vektore stanja.

AAnT Cloud prikazuje statistiku rada jedinica i sastoji se od 6 analitičkih modula: nadzorna ploča KPI-a, histogram, regresijska krivulja, evaluacije temeljene na industrijskom standardu, usporedbe razdoblja te repozitorij svih izvještaja stanja.

## Problematika sustava

Sustav generalno ima zadovoljavajuće performanse. Statistička analiza zahtjeva kompleksnije operacije nad podacima, stoga AAnT Cloud radi nešto sporije od AAnT-a, no i dalje unutar granica prihvatljivosti za krajnjeg korisnika.

Glavni problem sustava proizlazi iz upotrebe zastarjele verzije baze podataka. Naime, korištena je verzija MySQL-a 5.5, koja je izdana u prosincu 2010. godine, a podrška za nju je prestala u prosincu 2018. godine. To znači da više nisu dostupna sigurnosna ažuriranja niti ispravci grešaka za tu verziju. Preporučuje se nadogradnja na novije verzije MySQL-a ili zamjena nekim drugim odgovarajućim sustavom za pohranu podataka. U ovom radu istražuje se mogućnost zamjene MySQL-a s InfluxDB-om, sustavom specijaliziranim za pohranu vremenskih serija podataka.

## 1.2. Model podataka u MySQL-u

### O MySQL-u

MySQL je jedan od najpopularnijih otvorenih relacijskih sustava za upravljanje bazama podataka. Razvijen je kao brz, pouzdan i fleksibilan sustav koji podržava širok spektar aplikacija. Ključne karakteristike MySQL-a uključuju:

- Relacijski model: MySQL se temelji na relacijskom modelu podataka, gdje su podaci organizirani u tablicama s definiranim odnosima između njih. Ovo omogućuje strukturiranu pohranu podataka prema definiranim pravilima integriteta.
- SQL (engl. *Structured Query Language*): MySQL koristi SQL kao jezik za upravljanje podacima. SQL omogućuje izvršavanje različitih operacija nad bazama podataka, uključujući upite za čitanje, pisanje i izmjenu podataka, definiranje struktura baza podataka, te upravljanje korisnicima i pravima pristupa.
- Višestruke platforme: MySQL je dostupan na različitim operativnim sustavima, uključujući Linux, Windows i MacOS, što ga čini pristupačnim i fleksibilnim izborom za različite razvojne okoline.

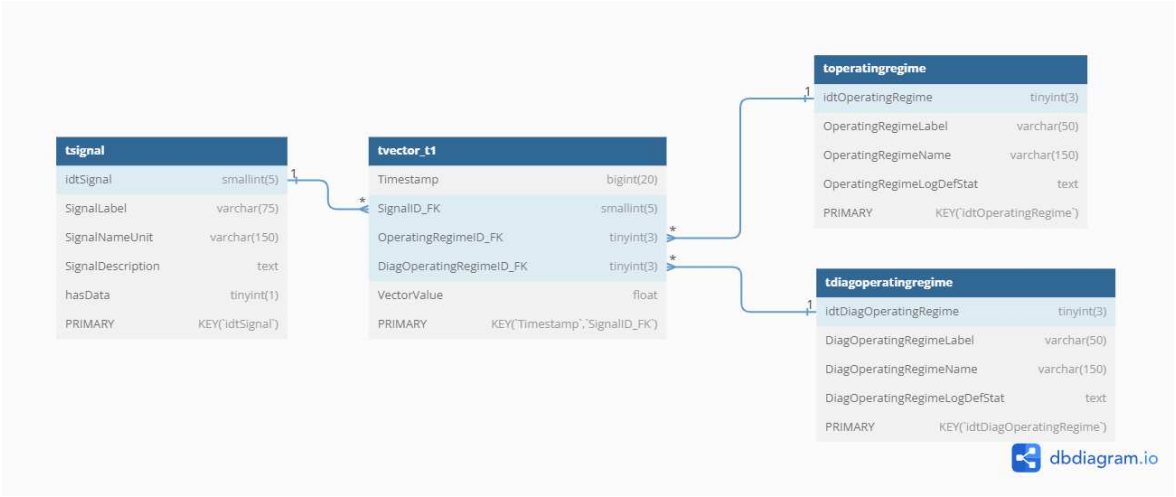
Relacijski sustavi poput MySQL-a nude pouzdanost, konzistentnost i jednostavno rukovanje podacima, što ih čini popularnim izborom za razne aplikacije, uključujući web aplikacije, sustave za upravljanje sadržajem, e-trgovinu i mnoge druge. SQL kao standardni jezik za upravljanje relacijskim bazama podataka omogućuje jednostavno učenje i široku primjenu, čineći MySQL moćnim alatom za pohranu i manipulaciju podacima.

### Model podataka

Za potrebe ovog rada, koriste se podaci koji dolaze iz demo postrojenja koje se sastoji od 12 jedinica. Svaka jedinica ima svoju instancu baze podataka na istom poslužitelju s istom shemom kao i ostale. Shema je prikazana na slici (Slika 1.1).

Glavna tablica naziva se `tvector_t1` i u nju se pohranjuju izračunati vektori stanja za svaki signal. Svaki zapis (koji predstavlja jedno mjerenje) ima vremenski žig u milisekundama (`Timestamp`). Vremenski žig i identifikator signala (`SignalID_FK`) tvore primarni ključ te tablice. Izračunata vrijednost vibracijskog stanja pohranjuje se u atribut `VectorValue` koji je tipa `float`. Dodatno, za svaki zapis pohranjuje se

identifikator operativnog režima i dijagnostičkog operativnog režima rada. Atributi `SignalID_FK`, `OperatingRegimeID_FK` i `DiagOperatingRegimeID_FK`, iako im u nazivu stoji kratica FK koja označava na engleskom *foreign key*, nisu zapravo implementirani kao strani ključevi, već predstavljaju samo konceptualno strane ključeve na opisne tablice `tsignal`, `toperatingregime` i `tdiagoperatingregime`. Svaka opisna tablica ima identifikator (npr. `idSignal`) koji je primarni ključ, oznaku (engl. *label*) koja je konceptualno jedinstvena unutar sustava te naziv koji ne mora biti jedinstven te korisnik ima mogućnost izmjene tog naziva. Tablica `tsignal` ima dodatni atribut `hasData` koji predstavlja zastavicu postoje li zapisi za taj signal u glavnoj tablici.




Slika 1.1 Shema baze podataka

# 2. InfluxDB – sustav za pohranu vremenski serija

Kako je tehnologija napredovala, senzori su počeli emitirati ogromnu količinu podataka vremenskih serija koje je potrebno pohraniti za daljnju analizu. Tradicionalne relacijske baze podataka pokazale su se lošim odabirom za pohranu vremenskih serija jer nisu optimizirane za rad s velikom količinom podataka iz čega je proizašla potreba za namjenski izgrađenom bazom podataka za vremenske serije. Danas najpopularniji takvi sustavi baza podataka su InfluxDB, TimescaleDB, Prometheus, QuestDB, Apache Druid i drugi. Njihova posebnost je što koriste optimizirane strukture podataka poput vremenskih stabala i indeksa kako bi omogućili brze upite i analize. Danas najpopularniji sustav za pohranu vremenskih serija je InfluxDB.

InfluxDB je baza podataka vremenskih serija otvorenog koda napisana u jeziku Go koju je razvio InfluxData. Prva verzija puštena je 2013. godine u rujnu, a veliko poboljšanje došlo je 2019. godine kada je izašao InfluxDB 2.0 koji je riješio mnoge probleme prve verzije. U ovom radu korištena je verzija 2.7.1.

## 2.1. Ključni pojmovi

**Mjerenje (engl. *measurement*)**

Stupac `_measurement` konceptualno predstavlja skupinu mjerenja koja dolaze iz istog izvora te imaju iste atribute. U relacijskom sustavu mjerenje bi najbolje odgovaralo jednoj tablici.

**Vremenski žig (engl. *timestamp*)**

Vremenski žig je osnovni atribut svih podataka vremenskih serija. Svaka tablica podataka u InfluxDB-u sadrži obavezno stupac `_time` u koji se sprema vremenski žig. Na disku, taj vremenski žig pohranjuje se u formatu nanosekundi epohe.

**Polja (engl. *fields*)**

Polja su ključ-vrijednost parovi u koja se tipično spremaju izmjerene vrijednosti. Ključ polja je tipa `string` koji predstavlja ime polja, a vrijednost polja tj. izmjerena vrijednost može biti `string`, `integer`, `float` ili `boolean`.

**Oznake (engl. *tags*)**

Oznake služe za pohranu meta podataka o mjerenjima. Također dolaze u obliku ključ-vrijednost. Razlika oznaka u odnosu na polja je što su vrijednosti oznaka uvijek tipa `string` te što su oznake indeksirane.

**Serije i ključ serije**

Ključ serije definiran je jedinstvenom kombinacijom stupca _measurement, vrijednosti oznaka i ključa polja. Ne postoji jedinstveni identifikator serije kao što u relacijskim sustavima postoji primarni ključ. Ključ serije u InfluxDB-u može se usporediti s UNIQUE ograničenjem u relacijskim sustavima. Seriju čine svi podaci s istim ključem serije koji imaju različite vrijednosti polja za različite vremenske žigove.

**Spremnik (engl. *bucket*)**

Svi InfluxDB podaci pohranjuju se u spremnike koji konceptualno odgovaraju bazi podataka u relacijskim sustavima. Za svaki spremnik moguće je definirati period zadržavanja (engl. *retention period*) nakon kojeg će se zapisi obrisati. Predodređena vrijednost je zauvijek.

**Organizacija**

Organizacija predstavlja hijerarhijsku razinu koja omogućuje strukturiranje i upravljanje spremnicima, kontrolu pristupa korisnika, slaganje nadzornih ploča itd. Može se smatrati ekvivalentom poslužitelju u relacijskim sustavima u kojem se kreira više baza podataka, dodaju razne uloge korisnicima itd.

## 2.2. Ključni koncepti

**Vremenski strukturirano stablo spajanja (engl. *Time Structured Merge Tree* - TSM)**

Za učinkovito sažimanje i pohranu podataka, mehanizam za pohranu grupira vrijednosti polja prema ključu serije, a zatim sortira te vrijednosti prema vremenskom žigu. Tako organizirani podaci se komprimiraju i pohranjuju u stupčasto-orijentiranoj arhitekturi te se to naziva TSM pohrana. Ovakav način pohrane omogućuje značajno smanjenje zauzeća memorije jer kompresija stupaca često daje bolje rezultate nego kompresija redaka. Osim toga, omogućuje brže čitanje podataka jer podaci mogu čitati po ključu serije.

Još jedna prednost TSM datoteka je efikasnije rukovanje vremenskim serijama podataka. Umjesto pohranjivanja svakog pojedinačnog podatka, pohranjuju se samo razlike između uzastopnih vrijednosti u nizu. Ovaj pristup dodatno smanjuje potrebnu memoriju i povećava brzinu pristupa podacima, jer se izračunava samo razlika u odnosu na prethodnu vrijednost, čime se smanjuje količina podataka koju je potrebno pohraniti i obraditi.

Ukratko, vremenski strukturirana stabla spajanja omogućuju optimalno skladištenje i brzi pristup velikim količinama vremenskih serija podataka, koristeći kompresiju i organizaciju podataka na način koji smanjuje memorijske zahtjeve i poboljšava performanse sustava.

**Fragmenti i grupe fragmenata (engl. *Shards and shard groups*)**

InfluxDB organizira vremenske serije u fragmente prilikom pohranjivanja na disk. Svaki fragment definiran je vremenskim intervalom, a može sadržavati više serija podataka (definiranih ključem serije) tj. sve točke iz tih serija podataka koje su unutar zadanog intervala. Interval se određuje na temelju vremena zadržavanja podataka (engl. *retention period*) kao što je prikazano na slici (Slika 2.1). Ako je vrijeme zadržavanja veće od 6 mjeseci (u ovom radu koristi se zauvijek), pojedini fragment sadrži podatke od 7 dana.

| Bucket retention period | Default shard group duration |
|---|---|
| less than 2 days | 1h |
| between 2 days and 6 months | 1d |
| greater than 6 months | 7d |

Slika 2.1 Tablica vremena zadržavanja podataka

**Indeks vremenskih serija (engl. *Time Series Index* - TSI) i kardinalnost**

Kako kardinalnost serija raste, upiti čitanja postaju sporiji. Kardinalnost se odnosi na broj jedinstvenih serija podataka u bazi, što može značajno utjecati na performanse sustava. Indeks vremenskih serija (TSI) osigurava da upiti ostaju brzi čak i kada kardinalnost raste, omogućujući učinkovito pretraživanje i pristup podacima.

TSI organizira ključeve serija tako da ih grupira prvo po mjerenju, zatim po oznakama i na kraju po ključevima polja. Ovakva struktura omogućuje brže pretraživanje jer se podaci pretražuju hijerarhijski, prvo prema vrsti mjerenja, zatim prema specifičnim oznakama, i na kraju prema ključevima polja.

Međutim, prilikom modeliranja podataka, važno je paziti da kardinalnost ne postane prevelika. Razlog tome je što se TSI sprema u RAM-u tijekom rada InfluxDB poslužitelja.

Ako kardinalnost postane prevelika, može doći do preopterećenja RAM-a, što može uzrokovati usporavanje sustava ili čak njegovu nestabilnost.

Dakle, dok TSI omogućuje učinkovito pretraživanje i rukovanje velikim brojem serija podataka, potrebno je pažljivo planirati strukturu podataka kako bi se izbjeglo preopterećenje sustava. Optimalno modeliranje podataka uključuje balansiranje broja mjerenja, oznaka i ključnih polja kako bi se održala visoka performansa bez ugrožavanja stabilnosti sustava.

## 2.3. Lokalna instalacija InfluxDB poslužitelja

InfluxDB dostupan je u Cloud te OSS (engl. *Open Source Software*) inačici. U ovom radu koristi se InfluxDB OSS, koji je potrebno samostalno instalirati i konfigurirati. InfluxDB OSS je dostupan za sve glavne platforme, uključujući Linux, Windows, i MacOS, a može se preuzeti i kao Docker image.

Instalacija je vrlo jednostavna, osobito za Windows platformu. Potrebno je preuzeti ZIP datoteku sa službene InfluxDB web stranice i raspakirati ju. Nakon raspakiranja, InfluxDB poslužitelj se pokreće iz konzole jednostavnom naredbom `influxd`. Nakon što je poslužitelj pokrenut, komunikacija s njim moguća je na više načina:

- InfluxDB CLI (engl. Command Line Interface): CLI omogućuje interakciju s InfluxDB-om putem komandne linije. Potrebno ga je zasebno preuzeti i instalirati. Korištenje CLI-a omogućuje izvršavanje upita, administrativnih zadataka, te uvoz i izvoz podataka.
- Web korisničko sučelje (engl. *User Interface* - UI): Web sučelje dostupno je na adresi `localhost:8086` nakon pokretanja poslužitelja. Ovo sučelje pruža intuitivni vizualni pristup za upravljanje bazom podataka, pregled i analizu podataka, te konfiguriranje sustava.
- Biblioteke za razne programske jezike: InfluxDB podržava integraciju s različitim programskim jezicima kao što su Python, Java, JavaScript, Go, i drugi. To omogućuje programerima da razvijaju aplikacije koje mogu izravno komunicirati s InfluxDB-om, izvodeći upite, pišući podatke i upravljajući bazom podataka putem API-ja specifičnih za određeni jezik.
- HTTP API (engl. *Application Programming Interface*): InfluxDB također pruža HTTP API, što omogućuje slanje upita i upravljanje podacima putem HTTP zahtjeva.

Ovo je korisno za integraciju s drugim sustavima i aplikacijama koje podržavaju HTTP komunikaciju.

InfluxDB OSS pruža fleksibilnost i kontrolu nad instalacijom i konfiguracijom sustava, što je idealno za razvojne okoline i prilagođene implementacije. Njegova kompatibilnost s više platformi i podrška za različite metode komunikacije čine ga svestranim alatom za upravljanje vremenskim serijama podataka.

## 2.4. Flux upitni jezik

Flux upitni jezik (engl. *Flux Query Language*) je napredni funkcionalni skriptni jezik dizajniran za postavljanje upita i analizu podataka, specijalno razvijen za InfluxDB 2.x. Omogućuje korisnicima fleksibilno i moćno rukovanje podacima vremenskih serija.

Svaki upit u Fluxu sastoji se od tri osnovna dijela: izvor podataka, vremenski raspon i filtri podataka:

- Izvor podataka definira se funkcijom `from()`, koja kao parametar prima spremnik (engl. *bucket*) iz kojeg se čitaju podaci. Ova funkcija specificira početnu točku za daljnju analizu podataka.
- Vremenski raspon: Flux zahtijeva definiran vremenski raspon prilikom postavljanja upita na podatke vremenskih serija. Upotrebom operatora za usmjeravanje (`|>`) podaci iz izvora prosljeđuju se funkciji `range()`, koja prihvaća dva parametra: `start` i `stop`. Ovi parametri mogu biti relativni (npr. `range(start: -1h, stop: -10m)` za posljednjih sat vremena do prije 10 minuta) ili apsolutni (npr. `range(start: 2021-01-01T00:00:00Z, stop: 2021-01-01T12:00:00Z)` za specifični vremenski period).
- Filteri podataka: Nakon definiranja izvora i vremenskog raspona, podaci se prosljeđuju funkciji `filter()`. Ova funkcija prima izraz koji se sastoji od `i`/`ili` operacija nad vrijednostima stupaca. Flux prolazi svaki ulazni redak i primjenjuje zadani izraz, ostavljajući u konačnom rezultatu samo retke koji zadovoljavaju specificirane uvjete.

Primjer formiranog jednostavnog upita dan je sljedećim kodom:

```
from(bucket: "example-bucket")
    |> range(start: -15m)
    |> filter(fn:(r)=>r._measurement=="cpu" and r._field=="usage_system")
    |> yield()
```

U upitima nad InfluxDB-om, transformacije podataka igraju ključnu ulogu u izvlačenju korisnih informacija iz vremenskih serija podataka. Ove transformacije najčešće uključuju upotrebu raznih agregatnih funkcija, grupiranje podataka po određenim stupcima i definiranje prozora unutar kojih se primjenjuju te funkcije.

Agregatne funkcije kao što su `mean()`, `max()`, `min()`, `sum()`, `count()`, i mnoge druge, omogućuju korisnicima da sažmu velike skupove podataka na jednostavne, razumljive metrike. Na primjer, funkcija `mean()` izračunava prosječnu vrijednost unutar definiranog skupa podataka, dok `max()` i `min()` identificiraju najveće i najmanje vrijednosti u skupu podataka.

Grupiranje podataka po stupcima omogućuje korisnicima da analiziraju podatke prema specifičnim kategorijama ili atributima. Na primjer, grupiranjem podataka prema oznakama senzora ili geografskim lokacijama, korisnici mogu dobiti uvid u specifične uzorke ili trendove unutar tih grupa. Funkcija `group()` u Fluxu omogućuje definiranje stupaca po kojima će se podaci grupirati, čime se olakšava segmentirana analiza.

Postavljanje prozora, odnosno definiranje vremenskih intervala unutar kojih se primjenjuju agregatne funkcije, omogućuje detaljniju analizu podataka kroz vremenske serije. Funkcija `window()` u Fluxu koristi se za razbijanje podataka na manje, jednako vremenski raspoređene segmente. Na primjer, korisnik može definirati prozor od jednog sata, unutar kojeg će se izračunavati prosječna temperatura. Ovakav pristup omogućuje uvid u promjene podataka kroz definirane vremenske intervale, čime se olakšava detekcija trendova i anomalija.

Primjer upita koji koristi agregatnu funkciju, grupiranje i prozor dan je sljedećim kodom:

```
from(bucket: "example-bucket")
    |> range(start: -1h)
    |> filter(fn:(r)=>r._measurement=="cpu")
    |> group(columns: ["usage_system"])
    |> window(every: 5m)
    |> mean()
```

Tablica 2.1 prikazuje osnovne izraze za manipulaciju podataka u SQL-u i njihov ekvivalent u Flux-u, funkcionalnom jeziku razvijenom za InfluxDB.

Tablica 2.1 Usporedba SQL-a i Flux-a

| SQL | Flux |
|---|---|
| `SELECT x,y` | `keep(columns: ["x", "y"])` |
| `FROM table` | `from(bucket: "example-bucket")` |
| `WHERE column = "a"` | `filter(fn:(r)=>r["column"] =="a")` |
| `WHERE ts > 1651190400` | `range(start: 1651190400)` |
| `GROUP BY column` | `group(columns: ["column"])` |
| `SELECT AVG(x)` | `mean(column: "x")` |
| `ORDER BY time DESC` | `sort(columns: ["_time"], desc: true)` |
| `LIMIT 10` | `limit(n: 10)` |
| `JOIN table2 ON table1.id = table2.id` | `join(tables: {t1: table1, t2: table2}, on: ["id"])` |
| `UNION` | `union(tables: [table1, table2])` |

# 3. Modeliranje i migracija podataka u InfluxDB

Prilikom izrade novog modela podataka primijenjena su četiri različita pristupa. Cilj ovog istraživanja je detaljno ispitati mogućnosti i principe modeliranja podataka u InfluxDB-u te na kraju odabrati optimalan pristup. Kako bi se odredio optimalan pristup, migracija podataka izvršena je za sva četiri izrađena modela, omogućujući kasnije temeljito testiranje njihovih performansi.

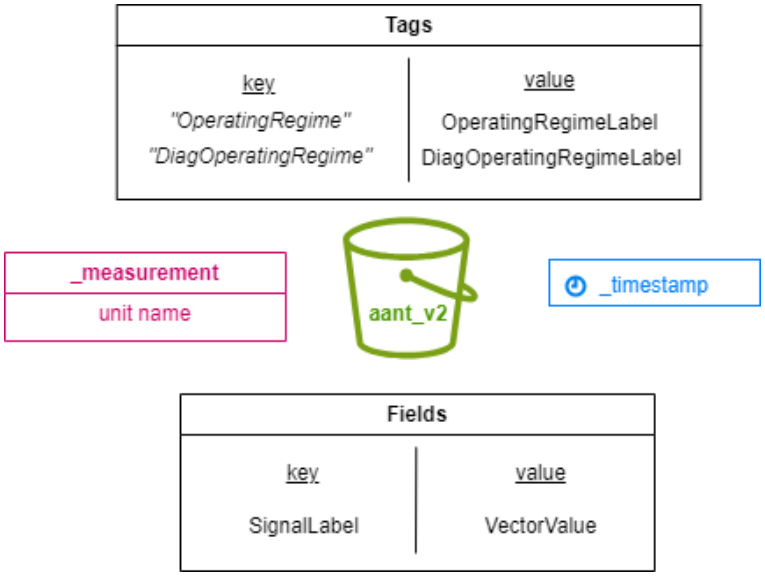
## 3.1. Opis modela

**Prvi pristup**

Pri izradi prvog modela motivacija je osim ubrzanja rada sustava, bila i jednostavnost. Umjesto dvanaest odvojenih baza podataka, sad će se koristiti samo jedan spremnik u kojem će se nalaziti podaci svih jedinica koje pripadaju jednom postrojenju. Time bi se olakšala komunikacija s podatkovnim slojem jer će umjesto dvanaest, biti potrebna samo jedna konekcija na bazu. U InfluxDB-u svi podaci nalaze se u jednoj tablici te nisu potrebna spajanja tablica kako bi se dobio željeni rezultat što dodatno pridonosi jednostavnosti modela. Potencijalna mana ovog pristupa mogla bi biti preveliko opterećenje pri izvođenju upita jer pri svakom upitu na ovaj spremnik koji objedinjuje sve jedinice pretražuje se u prosjeku dvanaest puta više podataka nego u sustavu u kojem svaka jedinica ima svoju bazu podataka. Druga potencijalna mana je što za razliku od originalnog sustava, u ovom pristupu signal nije indeksiran.

Na slici (Slika 3.1) nalazi se konceptualni prikaz modela u Influx-u. Influx tablica ima dva obavezna stupca: `_measurement` (crveno) i `_time` (plavo). S obzirom na to da će se sve jedinice nalazit jednom spremniku, u `_measurement` pohranit će se naziv jedinice (što je prije bio naziv same baze podataka). U stupac `_time` pohranjuje se vremenski žig u preciznosti milisekundi. Kao oznake postavljeni su `OperatingRegimeLabel` i `DiagOperatingRegimeLabel` jer se nad oznakama gradi indeks, a za sve analitičke upite uzimaju se podaci iz samo jednog režima rada stoga će se uvijek primjenjivati filter po režimu rada. Polja će kao ključ imati `SignalLabel`, a kao vrijednost

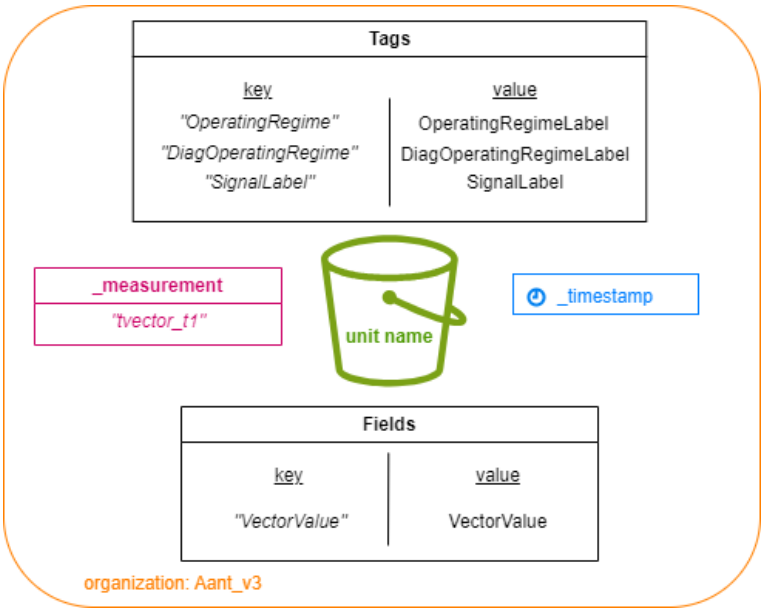
`VectorValue`. Ključ serije će se dakle sastojati od naziva jedinice, operativnog režima, dijagnostičkog režima te oznake signala.




Slika 3.1 Prvi pristup modeliranju podataka u InfluxDB-u

### Drugi pristup

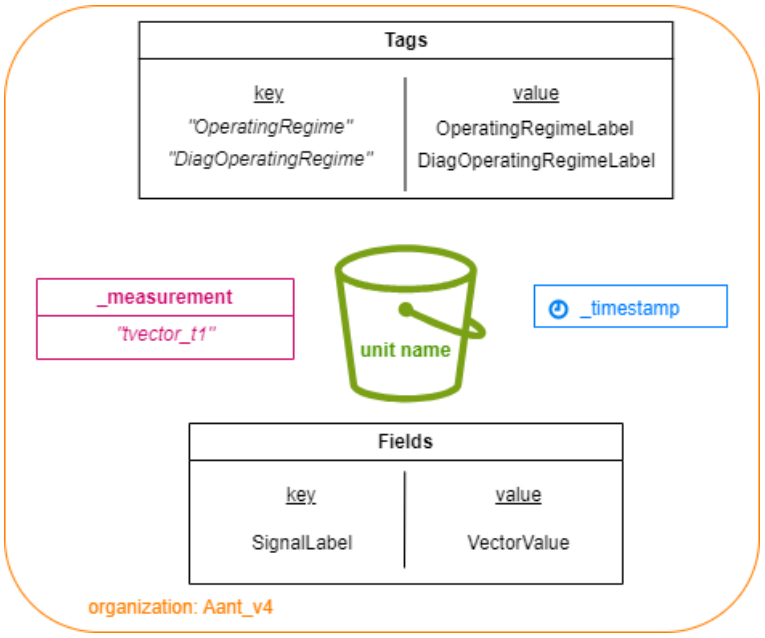
Uzevši u obzir potencijalne probleme prvog pristupa, u drugom pristupu svaka jedinica imat će svoj spremnik, a spremnici bit će grupirani u organizaciju. Svako postrojenje će tako imati svoju organizaciju koja će sadržavati onoliko spremnika koliko ima jedinica u postrojenju. U stupac `_measurement` u kojem je bio pohranjen naziv jedinice u prvom modelu sad ćemo pohraniti naziv tablice kao konstantu „`tvector_t1`“. Druga promjena je prebacivanje oznake signala (`SignalLabel`) u oznaku (*tag*) umjesto ključa polja kako bi dobili indeks i nad signalom. Kao ključ polja postavit ćemo konstantu „`VectorValue`“. Ovom promjenom ključ serije i dalje će se konceptualno sastojat od jedinice, vremena, signala i operativnih režima jer nove dvije postavljene konstante ne igraju nikakvu ulogu u broju različitih kombinacija vrijednosti stupaca koji čine ključ serije. Slika 3.2 prikazuje drugi pristup modeliranju podataka.




Slika 3.2 Drugi pristup modeliranju podataka u InfluxDB-u

### Treći pristup

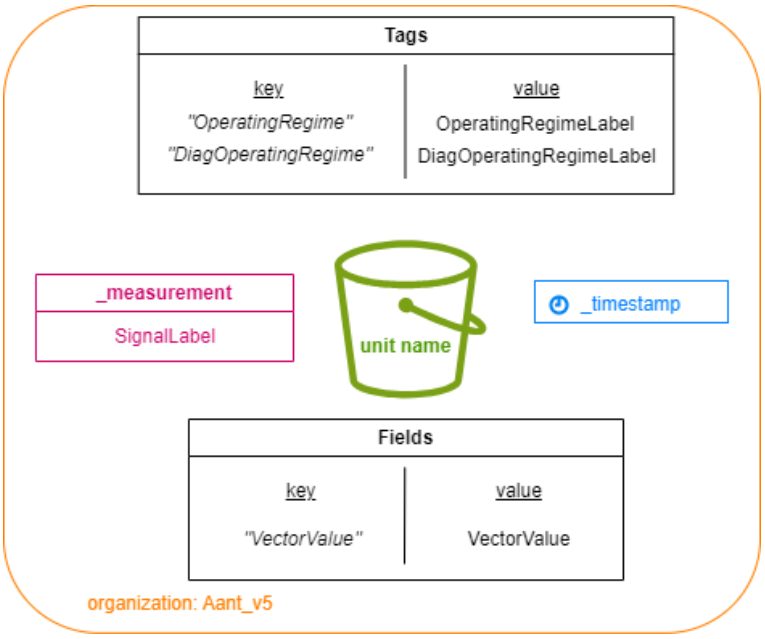
Treći pristup hibrid je prvog i drugog te je prikazan slikom Slika 3.3. Oznaka signala vraća se kao ključ polja umjesto zapravo redundantne konstante „`VectorValue`“, no zadržavaju se odvojeni spremnici za svaku jedinicu. Na ovaj način mogu se odvojeno usporediti dvije promjene uvedene iz prvog u drugi model te utvrditi efikasnost spajanja svih jedinica u isti spremnik kao i potreba da signal bude indeksiran.




Slika 3.3 Treći pristup modeliranju podataka u InfluxDB-u

### Četvrti pristup

U drugom i trećem pristupu u stupac `_measurement` pohranjuje se konstanta „`tvector_t1`“, no postavlja se pitanje može li se taj obavezni stupac u InfluxDB tablicama bolje iskoristiti jer InfluxDB gradi indeks vremenskih serija tako da prvo grupira ključeve serije po mjerenju, zatim po ostalim stupcima. Ako je kao mjerenje zapisana konstanta, tim grupiranjem nikako se ne pridodaje efikasnosti modela. Stoga, ideja ovog pristupa je pohraniti u `_measurement` oznaku signala. Pretpostavka je da bi to moglo prvenstveno ubrzati upite koji koriste signal, no moguća mana ovog modela mogla bi biti usporavanje ostalih upita. Ovaj pristup prikazan je slikom (Slika 3.4).




Slika 3.4 Četvrti pristup modeliranju podataka u InfluxDB-u

### Usporedba pristupa

U tablici Tablica 3.1 prikazane su ključne razlike između četiri pristupa modeliranja podataka u InfluxDB-u te originalnog MySQL relacijskog modela. Prvi pristup spaja sve jedinice u jedan spremnik, dok ostali zadržavaju sličniju strukturu kao u originalnom modelu gdje jedan spremnik odgovara jednoj bazi podataka, a svi spremnici kao i sve baze imaju istu shemu. Posebnost drugog pristupa je što je signal postavljen kao oznaka, a četvrtog što je `_measurement`. Oba pristupa su pokušaji optimizacije filtriranja i grupiranja po signalu, česte operacije u sustavu. Treći pristup je hibrid je prvog i drugog po tome što iz prvog sadrži signal kao ključ polja, a iz drugog odvojene spremnike. Operativni i dijagnostički režim uvijek su pohranjeni kao oznake, a vrijednost vektora stanja uvijek kao vrijednost polja.

Tablica 3.1 Usporedba pristupa modeliranju u InfluxDB-u

| | MySQL | Influx - prvi model | Influx - drugi model | Influx - treći model | Influx - četvrti model |
|---|---|---|---|---|---|
| **Postrojenje** | Poslužitelj baze podataka | bucket - "aant_v2" | organizacija - "aant_v3" | organizacija - "aant_v4" | organizacija - "aant_v5" |
| **Jedinica (unit)** | Baza podataka | posebni stupac _measurement | bucket | bucket | bucket |
| **Vremenski žig** | Stupac „Timestamp“ u tablici tvector_t1 nad kojim je napravljen indeks | posebni stupac _timestamp | posebni stupac _timestamp | posebni stupac _timestamp | posebni stupac _timestamp |
| **Operativni/ dijagnostički režim** | toperatingregime/tdiagoperatingregime tablica, nema indeksa u tablici tvector_t1 | tag – indeksirani stupac (OperatingRegimeLabel) | tag – indeksirani stupac (OperatingRegimeLabel) | tag – indeksirani stupac (OperatingRegimeLabel) | tag – indeksirani stupac (OperatingRegimeLabel) |
| **Signal** | tsignal tablica, indeks u tablici tvector_t1 | field key (nema indeksa) | tag – indeksirani stupac (SignalLabel) | field key (nema indeksa) | posebni stupac _measurement |
| **Vrijednost vektora** | stupac „VectorValue“ u tablici tvector_t1 | field value | field value | field value | field value |

## 3.2. Migracija podataka

**Preduvjeti**

Zbrojeno u svim jedinicama ima oko dvije milijarde podataka koje je potrebno prebaciti iz postojećeg MySQL sustava u novi InfluxDB sustav. To uključuje četiri koraka: čitanje podataka iz MySQL baze, transformacija podataka, upisivanje podataka u InfluxDB te provjera ispravnog broja prebačenih zapisa. Cijeli postupak proveden je u *batch*-evima.

Podjela na *batch*-eve mora zadovoljiti sljedeće kriterije:

- Ne smiju biti preveliki za memoriju programa koji ih obrađuje.
- Pojedini *batch* mora biti deterministički određen kako bi se, u slučaju pogreške tijekom izvođenja programa, moglo utvrditi točno koji su zapisi prebačeni, a koji ne.
- Pri završetku migracije mora biti očuvana konzistentnost i cjelokupnost podataka.

| UNIT | BROJ ZAPISA | BROJ SIGNALA | ZAPIS/SIGNAL |
|---|---|---|---|
| codis-dm_unit1 | 291697574 | 382 | 763606 |
| codis-dm_unit2 | 302779460 | 306 | 989475 |
| codis-dm_unit3 | 322024391 | 308 | 1045534 |
| codis-dm_unit4 | 245897881 | 90 | 2732199 |
| codis-dm_unit5 | 69678842 | 90 | 774209 |
| codis-dm_unit6 | 96981651 | 90 | 1077574 |
| codis-dm_unit7 | 67866400 | 90 | 754071 |
| codis-dm_unit8 | 81054923 | 74 | 1095337 |
| codis-dm_unit9 | 92794874 | 96 | 966613 |
| codis-dm_unit11 | 308992984 | 355 | 870403 |
| codis-dm_unit12 | 57855635 | 330 | 175320 |
| codis-dm_unit13 | 57855635 | 330 | 175320 |

Tablica 3.2 Broj zapisa i signala po jedinici

U tablici (Tablica 3.2) popis je svih jedinica te broja zapisa i različitih signala po jedinici. Prosječan broj zapisa po signalu najviše je oko jedan milijun što nije previše za učitati u program odjednom jer zapisi nisu veliki, stoga je odabrani način podjele na *batch*-eve podjela po signalu. Time je deterministički određeno koji zapisi ulaze u svaki *batch* te je moguća jednostavna provjera da su svi zapisi uspješno spremljeni u InfluxDB.

Prije početka migracije potrebno je stvoriti nove organizacije i spremnike u InfluxDB-u.

**Korištene tehnologije i postupak**

Tehnologija odabrana za migraciju podataka je programski jezik Java u kojem su napisana dva programa: `MigrationOrchestrator` i `MySqlToInfluxMigration`. `MigrationOrchestrator` spaja se na MySQL bazu za odabranu jedinicu te učitava popis signala za koje postoje podaci upitom:

```
SELECT SignalLabel FROM tsignal t WHERE hasData = 1
```

a zatim se za svaki signal poziva `MySqlToInfluxMigration`. Razlog za razdvajanje na dva programa leži u problemu s *Java Heap Space*-om. U prvom pokušaju, jedan je program učitavao redom zapise za sve signale, no u takvom pristupu Javin virtualni stroj (engl. *Java Virtual Machine*) ostao bi bez memorijskog prostora jer se kontinuirano učitavaju novi zapisi, a prebačeni ne stižu dovoljno brzo čistiti iz memorije. Čišćenje memorije u Javi obavlja interni proces zvan *Garbage Collector* te nije moguće ručno očistiti memoriju. Razdvajanjem na dva programa, `MySqlToInfluxMigration` program je zadužen za učitavanje zapisa koji pripadaju samo jednom signalu te kako proces za pojedini signal završi, program se u potpunosti gasi te ponovo pokreće za idući signal. Tako je onemogućeno curenje memorije.

`MySqlToInfluxMigration` spaja se na MySQL za čitanje podataka i na InfluxDB za pisanje podataka. Podaci se čitaju sljedećim upitom:

```
SELECT `Timestamp`, SignalLabel, OperatingRegimeLabel,
       DiagOperatingRegimeLabel, VectorValue
FROM tvector_t1
JOIN tsignal ON tsignal.idtSignal = tvector_t1.SignalID_FK
JOIN toperatingregime ON toperatingregime.idtOperatingRegime =
       tvector_t1.OperatingRegimeID_FK
JOIN tdiagoperatingregime ON
       tdiagoperatingregime.idtDiagOperatingRegime =
       tvector_t1.DiagOperatingRegimeID_FK
WHERE SignalLabel='%s'
```

pri čemu se `%s` zamjenjuje trenutnim `SignalLabel`-om. Nakon toga se svaki zapis transformira u tzv. točku (engl. *Point*) koja će se zapisati u Influx. Isječak programskog koda kojim se stvaraju Influx točke prikazan je slikom Slika 3.5 na primjeru prvog modela.

```
List<Point> points = new ArrayList<>();
while (rs.next()) {
    Point point = Point.measurement(unit)
            .time(Instant.ofEpochMilli(rs.getLong( columnLabel: "Timestamp")), WritePrecision.MS)
            .addField(rs.getString( columnLabel: "SignalLabel"), rs.getDouble( columnLabel: "VectorValue"))
            .addTag("OperatingRegime", rs.getString( columnLabel: "OperatingRegimeLabel"))
            .addTag("DiagOperatingRegime", rs.getString( columnLabel: "DiagOperatingRegimeLabel"));
    points.add(point);
}
```

Slika 3.5 Primjer stvaranja Influx točke

Podaci se pohranjuju u Influx putem `InfluxDBClient` biblioteke za Javu. Ova biblioteka omogućava unošenje podataka u *batch*-evima, pri čemu su bitna dva parametra: `batchSize` i `flushInterval`. Parametar `batchSize` definira broj zapisa koji se upisuju u pojedini *batch*, postavljen na deset tisuća, dok `flushInterval` određuje koliko `InfluxDBClient` vremenski čeka prije nego što upiše podatke koji su se akumulirali, postavljen na 5 sekundi. Svrha zadavanja oba parametra je efikasno zapisivanje. Na primjer, ako imamo 23 000 podataka, pisanje u InfluxDB će se odviti u 3 *batch*-a: prva 2 će sadržavati po 10 000 podataka, što odgovara `batchSize`-u, dok će treći *batch* sadržavati preostalih 3 000 podataka i bit će zapisan nakon 5 sekundi. Bez `flushInterval`-a, posljednjih 3 000 podataka bi se izgubilo (ili bi `batchSize` morao biti znatno manji), dok bez `batchSize`-a zapisi bi se upisivali jedan po jedan, što bi rezultiralo značajno sporijim procesom.

Nakon unosa svih zapisa za pojedini signal provjerava se odgovora li broj zapisa u MySQL-u za taj signal broju upisanih u InfluxDB te ispisuje adekvatna poruka. Program tijekom cijelog procesa migracije podataka ispisuje logove za praćenje napretka i potencijalnih grešaka. Izgled logova prikazan je slikom (Slika 3.6).

```
Processing unit: codis-dm_unit2
Number of signals: 306
------------------------------
Processing signal: AV000.s1.00A
Rows in MySQL: 307857
...processing
Insertion time: 4348ms
Records in flux: 307857
Total processing time: 62645ms
------------------------------
Processing signal: AV000.s1.00Ph
Rows in MySQL: 307857
...processing
Insertion time: 2283ms
Records in flux: 307857
Total processing time: 53313ms
------------------------------
Processing signal: AV000.s2.00A
Rows in MySQL: 307857
...processing
Insertion time: 2411ms
Records in flux: 307857
Total processing time: 53528ms
------------------------------
```

Slika 3.6 Primjer logova programa za migraciju

### Konfiguracija pisanja u InfluxDB

Kako tijekom migracije ne bi došlo do preopterećenja sustava, potrebno je postaviti parametar *cache-snapshot-write-cold-duration* na prikladnu vrijednost. To je parametar kojim se konfigurira interni rad InfluxDB poslužitelja i postavlja se kao zastavica pri pokretanju poslužitelja iz konzole:

```
influxd --cache-snapshot-write-cold-duration=10s
```

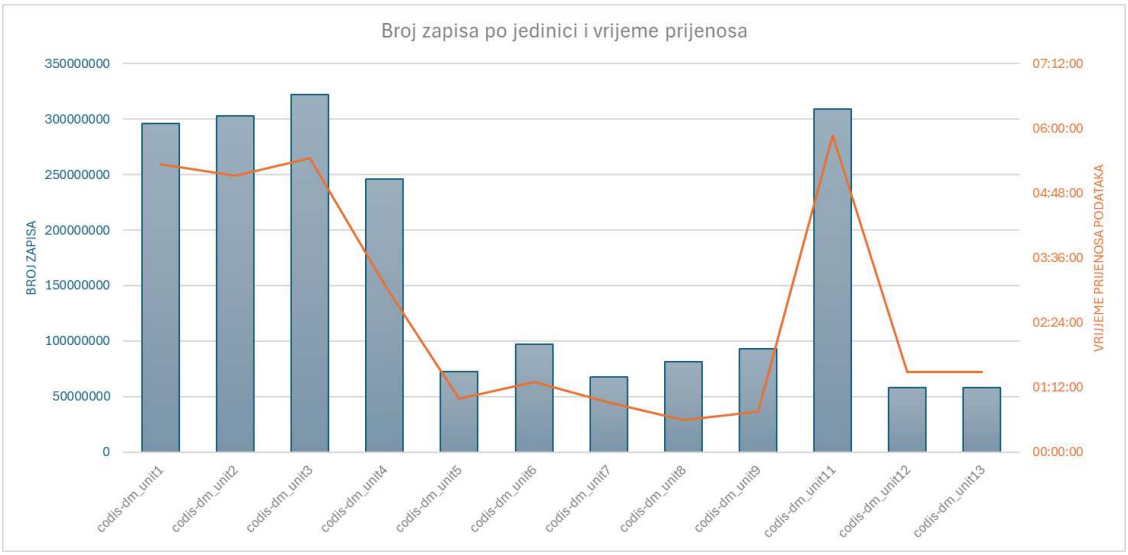
Kao što je opisano u prethodnom poglavlju, InfluxDB organizira podatke u shard-ove definirane vremenskim intervalima. Tijekom zapisivanja podataka u pojedini shard, taj shard prvo se učitava u *cache*, što ga čini vrućim (engl. *hot shard*). Nakon određenog vremena bez novih zapisivanja, taj shard postaje hladan (engl. *cold shard*) te se komprimira i pohranjuje na disk. Vrijeme koje treba proći prije nego što se shard ohladi te zapiše na disk i izbriše iz cache-a definira se parametrom cache-snapshot-write-cold-duration. Predodređena vrijednost ovog parametra je 10 minuta, što znači da treba proći 10 minuta bez novih zapisa u shardu kako bi se on prebacio na disk.

Prilikom zapisivanja povijesnih podataka, kontinuirano se zapisuje u sve shard-ove, što rezultira potrebom da svi shard-ovi budu učitani u cache. Ovaj proces, međutim, može dovesti do zagušenja memorije jer se kontinuirano povećava zauzeće memorije te nadalje to može rezultirati prekidom cijelog procesa migracije.

Opcija zadržavanja podataka u cache-u je korisna kada se uneseni podaci odmah i pregledavaju (na primjer, kod unosa *live* podataka koji se odmah prikazuju na korisničkom sučelju), no kod unosa povijesnih podataka, nema potrebe za zadržavanjem u cache-u. Naprotiv, kako bismo spriječili preopterećenje memorije, želimo da se podaci odmah spremaju na disk stoga postavljamo ovaj parametar na 10 sekundi.

### Trajanje migracije

Migracija svih podataka u prvi model trajala je 32 sata (to se odnosi na vrijeme izvođenje programa, no zbog ručnog pokretanja programa za svaku jedinicu ukupno vrijeme je duže). Na slici (Slika 3.7) prikazan je broj zapisa po jedinici te vrijeme prijenosa svih podataka po jedinici. Uočavamo da je vrijeme otprilike proporcionalno broju podataka iz čega se može zaključiti da je proces migracije tekao bez poteškoća i zagušenja InfluxDB poslužitelja.




Slika 3.7 Broj zapisa po jedinici i vrijeme prijenosa

# 4. Usporedba brzine izvođenja upita

U ovom poglavlju provodi se testiranje MySQL baze podataka u usporedbi s četiri modela u InfluxDB-u kako bi se istražila njihova učinkovitost u izvođenju različitih vrsta upita te odredilo koji je model najprikladnija zamjena za MySQL.

## 4.1. Korišteni alati i postupak

### Mjerenje brzine izvođenja upita

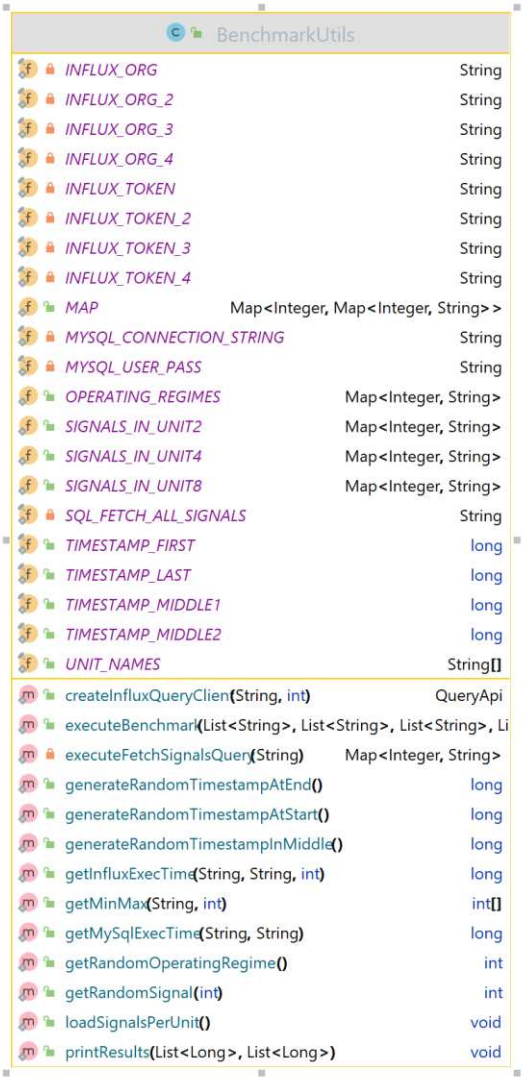
Za testiranje brzine izvođenja upita korišten je programski jezik Java. Za početak, napisana je klasa `BenchmarkUtils` (Slika 4.1) kao potpora za sve testove performansi. Unutar klase definirane su konstante za potrebne za povezivanje s InfluxDB bazom (token, naziv bucketa i organizacije kojoj bucket pripada) te s MySQL bazom (connection string, korisničko ime i lozinka). Klasa omogućuje generiranje slučajnih vremenskih žigova u predefiniranim intervalima kao i dohvaćanje slučajnog operativnog režima i signala iz liste istih. Za signale se učitava popis signala po jedinici iz baze na početku testiranja jer svaka jedinica ima drugačije signale dok su operativni režimi isti za sve jedinice. Klasa nudi metode koje izvršavaju različite vrste upita u MySQL-u i InfluxDB-u te pritom mjere vrijeme izvođenja i/ili broj redaka u rezultatu. Mjeri se isključivo vrijeme potrebno za izvođenje upita, a ne i vrijeme potrebno da se stvori konekcija prema bazi, pripremi upit itd. Također klasa ima metode za ispis rezultata.

Radi jednostavnosti, testiranje je provedeno na tri reprezentativne jedinice: "codis-dm_unit2", "codis-dm_unit4" i "codis-dm_unit8". Među jedinicama pojavljuju se tri kombinacije broja zapisa i signala, a ove jedinice odabrane su kao primjer svake kombinacije - "codis-dm_unit2" sadrži puno zapisa (~300 milijuna) i puno signala (~400), "codis-dm_unit4" sadrži također puno zapisa (~250 milijuna) i manje signala (90), a "codis-dm_unit8" sadrži manje zapisa (~80 milijuna) i signala (90).

Testiranje se sastoji od 2 dijela: testiranje generičkih upita te testiranje tipičnih upita u sustavu.

Za svaku vrstu upita (npr. filtriranje po vremenu) napisana je zasebna klasa. Svaka klasa sadrži predložak upita koji testira za MySQL te za sva četiri InfluxDB modela. Zatim se

formira tristo upita sa slučajno generiranim parametrima upita za svaki model, od čega sto za svaku od tri testirane jedinice, te sprema u listu. Nakon probnih testova, ustanovljeno je da mjerenje vremena izvođenja upita daje najmjerodavnije rezultate ako se za svaki upit izvodi slijedno za model po model te unutar modela za jedinicu po jedinicu. Rezultati spremaju se u CSV datoteku.




Slika 4.1 Klasa BenchmarkUtils

### Analiza dobivenih rezultata

Rezultati dobiveni testiranjem spremljeni u CSV datoteke učitavaju se u Jupyter bilježnicu za daljnju obradu i analizu. Jupyter je iznimno popularan alat u zajednici za podatkovnu znanost, zahvaljujući svojoj fleksibilnosti i interaktivnosti. Omogućuje korisnicima da kombiniraju kôd, tekst, matematičke jednadžbe i vizualizacije u jedinstvenom dokumentu.

U kontekstu analize podataka, Jupyter je idealan za učitavanje, obradu i vizualizaciju dobivenih rezultata. Korištene biblioteke su:

- Pandas: Pandas je osnovna biblioteka za manipulaciju i analizu podataka. Omogućuje jednostavno učitavanje podataka u obliku DataFrame-a, te pruža funkcije za filtriranje, grupiranje i agregaciju podataka.
- Os: Biblioteka os omogućava rad s datotekama i direktorijima te je korištena za učitavanje CSV datoteka.
- Matplotlib: Matplotlib je osnovna biblioteka za kreiranje statičkih, animiranih i interaktivnih vizualizacija u Pythonu. Omogućuje crtanje raznih tipova grafova, uključujući linijske grafove, histograme, dijagram raspršenosti i mnoge druge.
- NumPy: NumPy je biblioteka koja dodaje podršku za velike multidimenzionalne nizove i matrice, s velikim brojem visokih matematičkih funkcija za rad s tim nizovima.

Statistički pokazatelji izračunati u svim testiranjima kao i prikazani grafovi izvedeni su koristeći Jupyter i navedene biblioteke.

## 4.2. Generički upiti

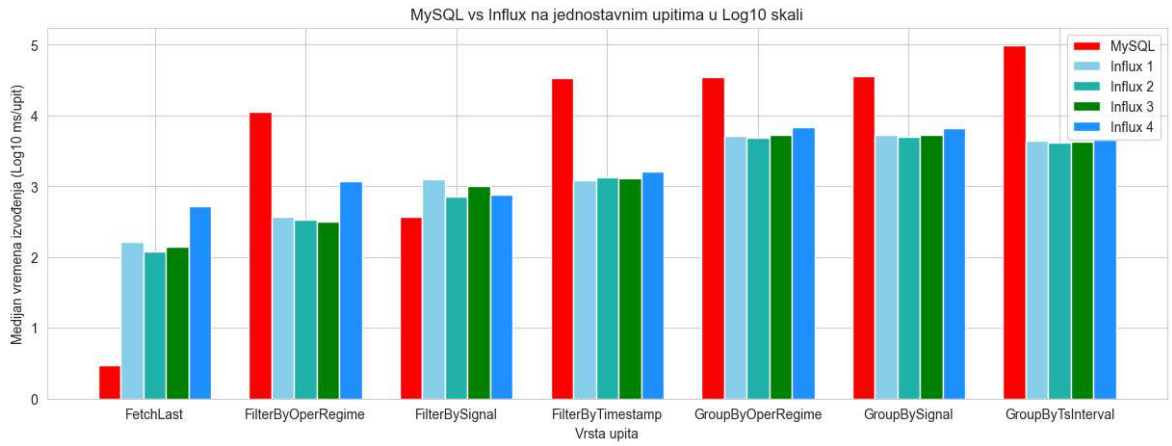
U prvom dijelu testiranja izvode se generički upiti. Ti upiti su jednostavni i fokus je na performansama filtriranja i grupiranja podataka koristeći samo jedan atribut iz modela podataka te jednu agregatnu funkciju. Cilj ovog testiranja je pružiti uvid u izolirane slučajeve kako bi identificirali koje su slabe točke pojedinog sustava te također u kojim vrstama upita koji sustav dominira.

U tablici (Tablica 4.1) nalazi se popis izvedenih upita. Napravljena su tri upita filtriranja podataka: filtriranje po signalu, operativnom režimu te vremenskoj oznaci te također tri upita grupiranja podataka po istom, uz iznimku da se ne grupira po jednom vremenskom žigu, već po intervalu. Zadnji upit je specifičan, radi se o dohvaćanju posljednjeg zapisa. Upiti napisani u SQL-u preslikani su što sličnije u InfluxDB upitni jezik – Flux te je u tablici dan primjer kako to izgleda za prvi od četiri testirana modela.

Tablica 4.1 Popis izvedenih generičkih upita

| Tip upita | Primjer u SQL-u | Primjer u Flux-u (prvi model) |
|---|---|---|
| Filter By Signal | `SELECT COUNT(*) as cnt FROM tvector_t1 WHERE SignalID_FK=485` | `from(bucket: "aant_v2") \|> range(start: 0, stop: now()) \|> filter(fn:(r)=>r["_measurement"]=="codis-dm_unit2") \|> filter(fn:(r)=>r["_field"]=="RV003.Peak2Peak") \|> group() \|> count()` |
| Filter By Operating Regime | `SELECT COUNT(*) as cnt FROM tvector_t1 WHERE OperatingRegimeID_FK=0` | `from(bucket: "aant_v2") \|> range(start: 0, stop: now()) \|> filter(fn:(r)=>r["_measurement"]=="codis-dm_unit4") \|> filter(fn:(r)=>r["OperatingRegime"]=="STOP") \|> group() \|> count()` |
| Filter By Timestamp | `SELECT COUNT(*) as cnt FROM tvector_t1 WHERE `Timestamp`>1633144947000;` | `from(bucket: "aant_v2") \|> range(start: 1633144947, stop: 1713045087) \|> filter(fn:(r)=>r["_measurement"]=="codis-dm_unit2") \|> group() \|> count()` |
| Group By Signal | `SELECT SUM(VectorValue) FROM tvector_t1 GROUP BY SignalID_FK` | `from(bucket: "aant_v2") \|> range(start: 0, stop: now()) \|> filter(fn:(r)=>r["_measurement"]=="codis-dm_unit2") \|> group(columns:["_field"]) \|> sum()` |
| Group By Operating Regime | `SELECT AVG(VectorValue) FROM tvector_t1 GROUP BY OperatingRegimeID_FK` | `from(bucket: "aant_v2") \|> range(start: 0, stop: now()) \|> filter(fn:(r)=>r["_measurement"]=="codis-dm_unit4") \|> group(columns:["OperatingRegime"]) \|> mean()` |
| Group By Timestamp Interval | `SELECT SUM(VectorValue) FROM tvector_t1 GROUP BY FLOOR (`Timestamp`/(1000*86400))` | `from(bucket: "aant_v2") \|> range(start: 0, stop: now()) \|> filter(fn:(r)=>r["_measurement"]=="codis-dm_unit2") \|> window(every: 24h) \|> sum()` |
| Fetch Last | `SELECT VectorValue FROM tvector_t1 WHERE SignalID_FK=495 ORDER BY `Timestamp` DESC LIMIT 1` | `from(bucket: "aant_v2") \|> range(start: 0, stop: now()) \|> filter(fn:(r)=>r["_measurement"]=="codis-dm_unit1") \|> filter(fn:(r)=>r["_field"]=="NONST018.DC") \|> last()` |

Nakon izvođenja upita, dobiveni rezultati ukazuju na značajne razlike u performansama između MySQL-a i InfluxDB-a. Stupčastim dijagramom prikazan je medijan vremena izvođenja za svaku vrstu upita (Slika 4.2). Radi boljeg prikaza, vremena izvođenja prikazana su u logaritamskoj skali baze 10. Već iz dijagrama, vidljivo je da MySQL većinu upita izvodi sporije od InfluxDB-a. MySQL pokazao je bolje vrijeme izvođenja na filtriranju po signalu, što je donekle očekivano s obzirom na to da su zapisi u MySQL modelu particionirani po signalu te u dohvaćanju zadnjeg zapisa što pomalo iznenađuje.




Slika 4.2 MySQL vs InfluxDB na jednostavnim upitima

Za svaku vrstu upita također su izračunati statistički pokazatelji (Tablica 4.2), srednja vrijednost, standardna devijacija, minimalna vrijednost, prvi kvartal, medijan, treći kvartal te maksimalna vrijednost. Iz izračunatih statističkih pokazatelja također je vidljivo da InfluxDB općenito pokazuje manje prosječno vrijeme izvršavanja upita u usporedbi s MySQL-om, no također pokazuje veću varijabilnost, s izuzetno visokim standardnim devijacijama i maksimalnim vrijednostima u usporedbi s MySQL-om. Ovi rezultati sugeriraju da iako InfluxDB može ponuditi bolju skalabilnost za neke upite, potrebno je obratiti pozornost i na konzistentnost performansi. Visoka standardna devijacija ukazuje na veliku razliku u vremenu izvođenja istog upita. Razlog tome može biti promjena parametara upita pri svakom izvođenju, no poželjno bi bilo da brzina izvođenja upita ne ovisi o konkretnim parametrima upita. Drugi razlog može biti što testirane jedinice nemaju jednaku količinu zapisa te broj zapisa koje je potrebno obraditi utječe na vrijeme izvođenja.

Tablica 4.2 Statistički pokazatelji izvođenja jednostavnih upita
(svi brojevi izraženi su u milisekundama)

| FilterByOperRegime | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 10248 | 2825 | 3538 | 3720 | 6306 |
| std | 5149 | 14612 | 20428 | 21346 | 36698 |
| min | 3018 | 48 | 32 | 40 | 378 |
| 25% | 4686 | 153 | 110 | 141 | 829 |
| 50% | 11360 | 376 | 343 | 313 | 1177 |
| 75% | 13968 | 1037 | 955 | 768 | 3057 |
| max | 30415 | 192392 | 271806 | 237264 | 568970 |

| GroupByOperRegime | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 32311 | 7619 | 7392 | 7231 | 9130 |
| std | 15177 | 19514 | 19290 | 14327 | 26098 |
| min | 11352 | 1194 | 1128 | 1170 | 1485 |
| 25% | 12068 | 3162 | 3111 | 3178 | 3522 |
| 50% | 35500 | 5195 | 4839 | 5260 | 6896 |
| 75% | 46797 | 8757 | 9003 | 8952 | 9173 |
| max | 57296 | 296656 | 291859 | 201493 | 379489 |

| FilterBySignal | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 496 | 2259 | 1572 | 2182 | 1743 |
| std | 448 | 2730 | 2286 | 3898 | 2934 |
| min | 52 | 25 | 31 | 23 | 22 |
| 25% | 157 | 93 | 98 | 87 | 76 |
| 50% | 377 | 1266 | 708 | 1012 | 759 |
| 75% | 698 | 3665 | 2041 | 3159 | 2546 |
| max | 2718 | 17061 | 15125 | 53573 | 27867 |

| GroupBySignal | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 30897 | 7521 | 7161 | 7184 | 8168 |
| std | 13843 | 15871 | 14262 | 13982 | 17122 |
| min | 11604 | 1201 | 1119 | 1182 | 1469 |
| 25% | 12249 | 3243 | 3158 | 3193 | 3479 |
| 50% | 35592 | 5338 | 4962 | 5259 | 6639 |
| 75% | 44325 | 8923 | 8818 | 8904 | 9362 |
| max | 49710 | 202427 | 194922 | 199276 | 213130 |

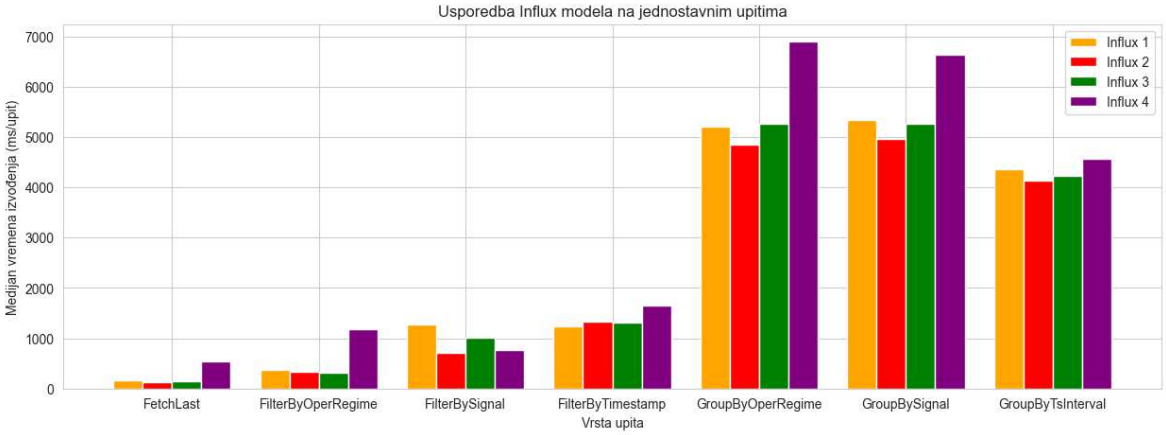
| FilterByTimestamp | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 73015 | 3086 | 2487 | 2781 | 3034 |
| std | 66617 | 11576 | 7301 | 9925 | 7764 |
| min | 3039 | 246 | 226 | 262 | 331 |
| 25% | 32888 | 597 | 596 | 594 | 774 |
| 50% | 34018 | 1239 | 1329 | 1304 | 1645 |
| 75% | 99552 | 2927 | 2864 | 2942 | 3250 |
| max | 228282 | 145789 | 97374 | 122887 | 88836 |

| GroupByTsInterval | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 103725 | 7218 | 6886 | 7092 | 8089 |
| std | 62105 | 16691 | 14317 | 16448 | 18521 |
| min | 29446 | 1259 | 1183 | 1238 | 1556 |
| 25% | 55950 | 2970 | 2864 | 2919 | 3341 |
| 50% | 98234 | 4355 | 4138 | 4232 | 4573 |
| 75% | 126096 | 7080 | 6566 | 7053 | 8549 |
| max | 243393 | 231744 | 172591 | 215191 | 239492 |

| FetchLast | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 12 | 453 | 346 | 330 | 1126 |
| std | 86 | 704 | 1442 | 287 | 954 |
| min | 1 | 118 | 72 | 100 | 418 |
| 25% | 2 | 139 | 95 | 124 | 490 |
| 50% | 3 | 164 | 119 | 140 | 531 |
| 75% | 4 | 672 | 398 | 682 | 2196 |
| max | 1006 | 7665 | 22453 | 1802 | 8627 |

Na slici (Slika 4.3) prikazan je medijan vremena izvođenja upita za InfluxDB modele bez logaritamskog skaliranja i vremena izvođenja MySQL-a kako bi se bolje mogli usporediti InfluxDB modeli međusobno. Primjetno je da dohvaćanje zadnjeg zapisa, koje se pokazalo sporije za Influx nego za MySQL, za sva 4 modela ima medijan izvođenja ispod jedne sekunde što je iz korisničke perspektive prihvatljivo. Filtriranje podataka pokazuje se znatno brže nego grupiranje podataka. Četvrti model pokazao se uvjerljivo najsporijim modelom te već iz ovih rezultata jednostavnih upita daje se naslutiti da četvrti model vjerojatno nije dobra zamjena. Najbolje performanse pokazuju drugi i treći model iz čega prvenstveno možemo zaključiti da više spremnika umjesto samo jednog poboljšava performanse sustava, no ostaje pitanje je li bolje pohraniti signal kao oznaku (kao što je u

drugom modelu) ili polje (kao što je u trećem). Pohraniti signal kao `_measurement` nije najbolje rješenje osim ako se u većini upita tipičnih za sustav ne filtrira po signalu jer ovaj pristup optimizira upravo te operacije, a ostale usporava.




Slika 4.3 Usporedba InfluxDB modela na jednostavnim upitima

## 4.3. Upiti specifični za sustav

U ovom dijelu testiranja, izvodit će se upiti specifični za ovaj sustav kako bi se utvrdilo je li InfluxDB prikladan za ovaj sustav i ako je, koji InfluxDB model je najprikladniji. U tablici (Tablica 4.3) nalazi se popis upita izvedenih u ovom dijelu testiranja. Izdvojena su dva upita koja se koriste za live prikaz podataka u AAnT modulu te jedan upit koji se koristi za prikaz statistike u AAnT Cloud-u. Prvi AAnT upit filtrira podatke po signalu te vremenskom intervalu unutar kojeg se podaci još i grupiraju po vremenu te se za njih računa minimalna, maksimalna i srednja vrijednost. Drugi upit također kombinira filtriranje po signalu i vremenskom žigu, no vraća broj zapisa te posljednji vremenski žig.

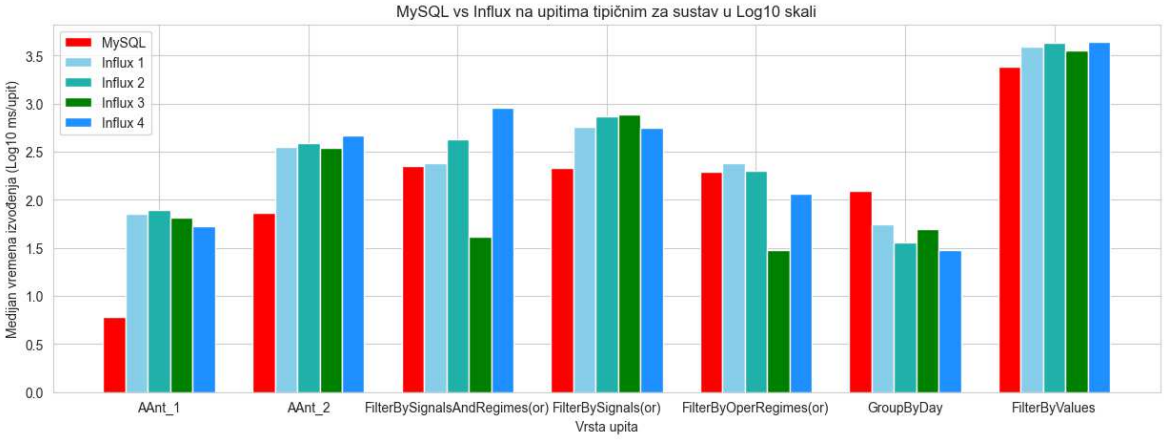
Izdvojeni AAnt Cloud upit vrlo je složen stoga je ovdje razlomljen na nekoliko upita kako bi se mogao detaljnije analizirati. Testira se filtriranje po više signala (koristeći operator ili), više operativnih režima (također operator ili) te jedno i drugo u kombinaciji za različit broj signala i režima rada. Zatim dolazi upit koji grupira podatke po danu (u određenom vremenskom intervalu za jedan signal te jedan operativni režim) te računa prosječnu vrijednost zaokruženu na određenu preciznost. Zadnji upit specifičan je po tome što filtrira podatke po vrijednosti vektora stanja.

Tablica 4.3 Popis izvedenih specifičnih upita

| | Primjer u SQL-u | Primjer u Flux-u (za treći model) |
|---|---|---|
| AAnt_1 | `SELECT`<br>`(Timestamp-1672512860000) DIV 235000 as ts`<br>`  ROUND(AVG(VectorValue),5) as avg,`<br>`  ROUND(MIN(VectorValue),5) as min,`<br>`  ROUND(MAX(VectorValue),5) as max`<br>`FROM tvector_t1`<br>`WHERE SignalId_FK = 301`<br>`  AND Timestamp >= 1672512860000`<br>`  AND Timestamp < 1672853610000`<br>`GROUP BY ts` | `data = from(bucket: "codis-dm_unit2")`<br>`\|> range(start: 1672512860, stop: 1672853610)`<br>`\|> filter(fn:(r)=>r["_measurement"]=="tvector_t1")`<br>`\|> filter(fn:(r)=>r["_field"]=="AV014.s2.00Ph")`<br>`\|> group()`<br><br>`data`<br>`\|> aggregateWindow(every:235s,fn:max,createEmpty:false)`<br>`\|> yield(name: "max")`<br>`data`<br>`\|> aggregateWindow(every:235s,fn: min,createEmpty: false)`<br>`\|> yield(name: "min")`<br>`data`<br>`\|> aggregateWindow(every:235s,fn:mean,createEmpty: false)`<br>`\|> yield(name: "mean")` |
| AAnt_2 | `SELECT`<br>`  MAX(t.Timestamp) as ts_end,`<br>`  COUNT(*)-1 as num_of_rows`<br>`FROM tvector_t1 t`<br>`WHERE SignalID_FK = 54`<br>`AND t.Timestamp>=1640995200000` | `data = from(bucket: "codis-dm_unit2")`<br>`\|> range(start: 1694677399, stop: now())`<br>`\|> filter(fn:(r)=>r["_measurement"]=="tvector_t1")`<br>`\|> filter(fn:(r)=>r["_field"]=="RV002.EqPeak")`<br>`\|> group()`<br><br>`data`<br>`\|> count()`<br>`\|>keep(columns: ["_value"])`<br>`\|>yield(name: "count")`<br>`data`<br>`\|> max(column:"_time")`<br>`\|>keep(columns: ["_time"])`<br>`\|>yield(name: "max_timestamp")` |
| AAntCloud – Filter By Signals And Operating Regimes | `SELECT AVG(VectorValue)`<br>`FROM tvector_t1`<br>`WHERE Timestamp`<br>`  BETWEEN 1622774731000`<br>`  AND 1679678588000`<br>`AND SignalID_FK IN (41,489,51)`<br>`AND OperatingRegimeID_FK in (8,6)` | `from(bucket: "codis-dm_unit2")`<br>`\|> range(start: 1622774731, stop: 1679678588)`<br>`\|> filter(fn:(r)=>r["_measurement"]=="tvector_t1")`<br>`\|> filter(fn:(r)=>r["_field"]=="TRG000.RotSpeed"`<br>`  or r["_field"]=="RV007.Peak2Peak"`<br>`  or r["_field"]=="RV000.EqPeak")`<br>`\|> filter(fn:(r)=>r["OperatingRegime"]=="UNKNOWN RS"`<br>`  or r["OperatingRegime"]=="PUMP RS")`<br>`\|> group()`<br>`\|> mean()` |
| AAntCloud – Filter By Signals | `SELECT AVG(VectorValue)`<br>`FROM tvector_t1`<br>`WHERE Timestamp`<br>`  BETWEEN 1638607133000`<br>`  AND 1649334782000`<br>`AND SignalID_FK IN (81,20)` | `from(bucket: "codis-dm_unit2")`<br>`\|> range(start: 1638607133, stop: 1649334782)`<br>`\|> filter(fn:(r)=>r["_measurement"]=="tvector_t1")`<br>`\|> filter(fn:(r)=>r["_field"]=="AV005.Rest"`<br>`  or r["_field"]=="AV001.EqPeak")`<br>`\|> group()`<br>`\|> mean()` |
| AAntCloud – Filter By Operating Regimes | `SELECT AVG( VectorValue)`<br>`FROM tvector_t1`<br>`WHERE Timestamp`<br>`  BETWEEN 1618067873000`<br>`  AND 1651124474000`<br>`AND SignalID_FK IN (447)`<br>`AND OperatingRegimeID_FK in (6,7)` | `from(bucket: "codis-dm_unit2")`<br>`\|> range(start: 1618067873, stop: 1651124474)`<br>`\|> filter(fn:(r)=>r["_measurement"]=="tvector_t1")`<br>`\|> filter(fn:(r)=>r["_field"]=="RV006.s1.00A")`<br>`\|> filter(fn:(r)=>r["OperatingRegime"]=="PUMP RS"`<br>`  or r["OperatingRegime"]=="UNKNOWN")`<br>`\|> group()`<br>`\|> mean()` |

| | | |
|---|---|---|
| AAntCloud – Filter By Signal And Operating Regime, Group By Day | <pre>SELECT<br>ROUND(AVG(VectorValue)/0.1,0)*0.1<br>  AS zaokruzeno,<br>LEFT(FROM_UNIXTIME(TIMESTAMP/1000),10)<br>  AS tabletime<br>FROM tvector_t1<br>WHERE Timestamp<br>  BETWEEN 1610698726000<br>  AND 1651253223000<br>  AND SignalID_FK IN (299)<br>  AND OperatingRegimeID_FK in (6)<br>GROUP BY tabletime</pre> | <pre>import "math"<br>from(bucket: "codis-dm_unit2")<br>\|> range(start: 1610698726, stop: 1651253223)<br>\|> filter(fn:(r)=>r["_measurement"]=="tvector_t1")<br>\|> filter(fn:(r)=>r["OperatingRegime"]=="PUMP RS")<br>\|> filter(fn:(r)=>r["_field"]=="RV006.EqPeak")<br>\|> group()<br>\|> aggregateWindow(every:1d,fn:mean,createEmpty:false)<br>\|> map(fn: (r) => ({<br>  x: r._time,<br>  y: math.round(x: r._value/0.1)*0.1<br>}))</pre> |
| AAntCloud – Filter By Values | <pre>SELECT *<br>FROM tvector_t1<br>WHERE SignalID_FK = 277<br>  AND VectorValue<br>  BETWEEN 38.731 AND 41.869</pre> | <pre>from(bucket: "codis-dm_unit2")<br>\|> range(start: 0, stop: now())<br>\|> filter(fn:(r)=>r["_measurement"]=="tvector_t1")<br>\|> filter(fn:(r)=>r["_field"] == "AV010.Rest"<br>  and r["_value"] > 38.731<br>  and r["_value"] < 41.869)</pre> |

Rezultati testiranja vremena izvođenja upita specifičnih za sustav, kontradiktorni su očekivanim. Na jednostavnim upitima InfluxDB je pokazivao bolje performanse, no na ovim složenijim upitima, MySQL za većinu vrsta upita ima bolje vrijeme izvođenja što se može vidjeti po medijanima vremena izvođenja (Slika 4.4). Posebno, MySQL nadmašuje InfluxDB u AAnt upitima. Samo za jednu vrstu upita, upit koji filtrira podatke po signalu i operativnom režimu te grupira po danu, MySQL je sporiji. Dobar je pokazatelj da Influx brže grupira podatke po danu, ipak je to sustav specijaliziran za rad s vremenima, no problematično je da na ostalim operacijama pokazuje lošije performanse s obzirom na to da ipak ima dosta drugih operacija koje se provode u tipičnim upitima.




Slika 4.4 MySQL vs InfluxDB na složenim upitima

U tablici (Tablica 4.4) prikazani su svi statistički pokazatelji za pojedine vrste upita iz kojih je ponovno vidljivo da InfluxDB pokazuje značajno veće standardne devijacije, što ukazuje na veću varijabilnost performansi. Jedan od mogućih razloga lošijih performansi InfluxDB-a je što se InfluxDB ne nosi dobro s velikom kardinalnošću serija ključeva, što je s ovim podacima slučaj. Drugi razlog je što je u nekim upitima potrebno vratiti više rezultata tj. koristiti više agregatnih funkcija što u Flux-u nije moguće u istom upitu, već je potrebno za svaki željeni rezultat stvoriti zasebnu privremenu tablicu tj. napisati zasebni podupit.

Tablica 4.4 Statistički pokazatelji izvođenja složenih upita
(svi brojevi izraženi su u milisekundama)

| AAnt_1 | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 47 | 528 | 630 | 511 | 404 |
| std | 146 | 2310 | 2888 | 2360 | 1720 |
| min | 1 | 8 | 7 | 7 | 7 |
| 25% | 3 | 19 | 23 | 16 | 13 |
| 50% | 6 | 71 | 78 | 65 | 53 |
| 75% | 22 | 188 | 205 | 150 | 153 |
| max | 1275 | 28352 | 34608 | 29507 | 18370 |

| AAnt_2 | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 171 | 971 | 1069 | 972 | 912 |
| std | 202 | 1342 | 1430 | 1411 | 1143 |
| min | 2 | 8 | 10 | 7 | 9 |
| 25% | 24 | 33 | 69 | 32 | 46 |
| 50% | 73 | 353 | 387 | 343 | 469 |
| 75% | 262 | 1469 | 1617 | 1349 | 1287 |
| max | 820 | 6744 | 7610 | 8003 | 5493 |

| AAnt_Cloud - Filter By Signals And Operating Regimes (or) | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 300 | 824 | 1286 | 744 | 1914 |
| std | 287 | 1463 | 2007 | 1731 | 3076 |
| min | 4 | 9 | 10 | 7 | 10 |
| 25% | 136 | 47 | 56 | 19 | 112 |
| 50% | 223 | 239 | 421 | 42 | 907 |
| 75% | 369 | 888 | 1677 | 567 | 2421 |
| max | 2700 | 11122 | 14539 | 17137 | 28892 |

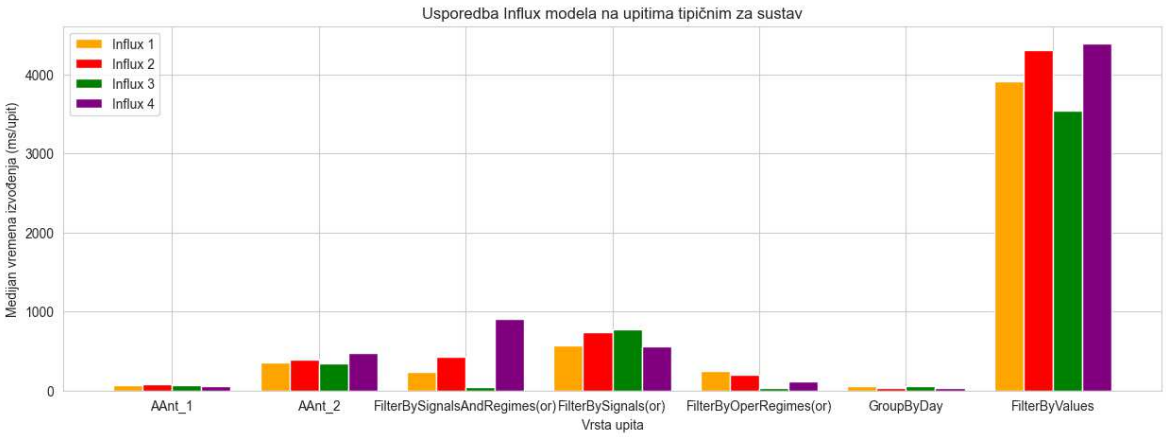
| AAnt_Cloud - Filter By Signals (or) | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 305 | 1189 | 1297 | 1516 | 1493 |
| std | 248 | 1816 | 1541 | 1742 | 1932 |
| min | 21 | 13 | 12 | 10 | 11 |
| 25% | 153 | 167 | 216 | 244 | 98 |
| 50% | 216 | 571 | 741 | 779 | 559 |
| 75% | 409 | 1619 | 1711 | 2315 | 2347 |
| max | 1722 | 22162 | 9228 | 8530 | 9859 |

| AAnt_Cloud - Filter By Operating Regimes (or) | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| count | 900 | 900 | 900 | 900 | 900 |
| mean | 275 | 1339 | 1269 | 293 | 889 |
| std | 290 | 2906 | 2642 | 826 | 2013 |
| min | 2 | 6 | 8 | 5 | 6 |
| 25% | 114 | 45 | 36 | 19 | 36 |
| 50% | 196 | 243 | 200 | 30 | 115 |
| 75% | 339 | 1255 | 1253 | 197 | 685 |
| max | 3550 | 28162 | 26854 | 9982 | 21241 |

| AAnt_Cloud - Filter By Signal,Operating Regime, Group By Day | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| count | 600 | 600 | 600 | 600 | 600 |
| mean | 376 | 316 | 264 | 592 | 255 |
| std | 668 | 713 | 628 | 1276 | 631 |
| min | 4 | 8 | 9 | 8 | 7 |
| 25% | 74 | 24 | 21 | 20 | 16 |
| 50% | 123 | 55 | 36 | 50 | 30 |
| 75% | 251 | 249 | 194 | 587 | 198 |
| max | 3798 | 5991 | 6364 | 8792 | 4976 |

| AAnt_Cloud - Filter By Values (and) | | | | | |
|---|---|---|---|---|---|
| | MySQL | Influx v1 | Influx v2 | Influx v3 | Influx v4 |
| mean | 2263 | 5838 | 6124 | 5488 | 6557 |
| std | 1532 | 7060 | 6886 | 6668 | 6983 |
| min | 0 | 58 | 64 | 51 | 62 |
| 25% | 690 | 1148 | 1313 | 842 | 1812 |
| 50% | 2437 | 3910 | 4307 | 3543 | 4391 |
| 75% | 3399 | 6402 | 7379 | 6805 | 8373 |
| max | 5960 | 45535 | 33808 | 34935 | 35270 |

Zaključno, ovakvi rezultati ukazuju na odličnu optimizaciju postojećeg sustava za upite koji se tipično izvode te dovodi u pitanje koliko bi InfluxDB bio adekvatna zamjena za takav sustav. U nastavku je prikazana usporedba InfluxDB modela (Slika 4.5). Primjetno je da je filtriranje po vrijednostima polja daleko najsporija operacija te usko grlo svakog modela. Razlog tome je što nema nikakvih indeksa (ili neke druge optimizacije) nad vrijednostima polja te se ti podaci moraju uvijek pretraživati slijedno. Prvi i drugi model imaju srednje vrijeme izvođenja od četiri uspoređivana modela, pri čemu je prvi model iznenađujuće brži od drugog (u većini slučajeva) s obzirom na to da u prvom jedan spremnik sadrži sve podatke te je znatno veća količina podataka koje je potrebno obraditi pri svakom upitu. Četvrti model najbrži je pri upitu koji filtrira podatke po više signala (što je i očekivano zbog pohranjivanja signala kao `_measurement`), no to je jedini upit u kojem dominira. Treći model pokazuje se kao optimalan jer ima najbolje vrijeme u većini slučajeva, a u onima u kojima nema najbolje vrijeme nije puno lošiji od drugih.




Slika 4.5 Usporedba InfluxDB modela na složenim upitima

## 4.4. Posebna opažanja

**Predmemoriranje rezultata upita kod MySQL-a (engl. *caching*)**

Tijekom prvog izvođenja testova, u programskim logovima zabilježeno je da se neki upiti u MySQL-u izvršavaju za nula milisekundi (Slika 4.6). To je dovelo do sumnje u stvarnu brzinu izvršavanja tih upita i postavilo pitanje o ispravnosti postavki testiranja.

```
-------------------------------
SELECT sum(VectorValue) FROM tvector_t1 GROUP BY OperatingRegimeID_FK
MySQL: 0
from(bucket: "aant_v2")
    |> range(start: 0, stop: now())
    |> filter(fn: (r) => r["_measurement"] == "codis-dm_unit1")
    |> group(columns: ["OperatingRegime"])
    |> sum()
Influx: 5540
-------------------------------
SELECT avg(VectorValue) FROM tvector_t1 GROUP BY OperatingRegimeID_FK
MySQL: 0
from(bucket: "aant_v2")
    |> range(start: 0, stop: now())
    |> filter(fn: (r) => r["_measurement"] == "codis-dm_unit4")
    |> group(columns: ["OperatingRegime"])
    |> mean()
Influx: 9648
-------------------------------
SELECT sum(VectorValue) FROM tvector_t1 GROUP BY OperatingRegimeID_FK
MySQL: 1
from(bucket: "aant_v2")
    |> range(start: 0, stop: now())
    |> filter(fn: (r) => r["_measurement"] == "codis-dm_unit8")
    |> group(columns: ["OperatingRegime"])
    |> sum()
Influx: 1340
```

Slika 4.6 Isječak logova

Pri završetku testova, izdvojene su tri vrste upita za koje je primijećen taj slučaj (Tablica 4.5). Prosječno vrijeme izvođenja upita u ovim primjerima sporije je za InfluxDB nego MySQL, no ako pogledamo maksimalno vrijeme uočavamo da je ipak 5-6 puta sporije kod MySQL-a te da su sva minimalna vremena manja od jedne milisekunde te je potrebno zapitati se zašto je to tako. Odgovor leži u predmemoriranju rezultata upita. Naime, MySQL poslužitelj pohranjuje tekst `SELECT` naredbe s odgovarajućim rezultatom koji je poslan klijentu. Ako kasnije na poslužitelj dođe identičan upit, poslužitelj dohvaća rezultat iz predmemorije umjesto da ponovno analizira i izvršava upit. Bitno je napomenuti da je ova funkcionalnost proglašena zastarjelom na verziji 5.7, a ukinuta 8.0, no ovaj sustav izgrađen je na verziji 5.5 stoga je ta funkcionalnost i dalje prisutna te znatno utječe na dobivene rezultate. Kod InfluxDB-a se ne vidi takav utjecaj predmemoriranja jer Influx memorira cijele vremenske fragmente kojima se nedavno pristupalo, umjesto konkretnih rezultata upita, a u ovom testiranju zbog slučajnog odabira parametara upita pristupa se nasumično različitim fragmentima u svakom upitu pa se memorirani fragmenti iz nedavnih upita ne koriste.

Tablica 4.5 Upiti s minimalnim vremenom izvođenja kod MySQL-a manjim od 1 ms (svi brojevi izraženi su u milisekundama)

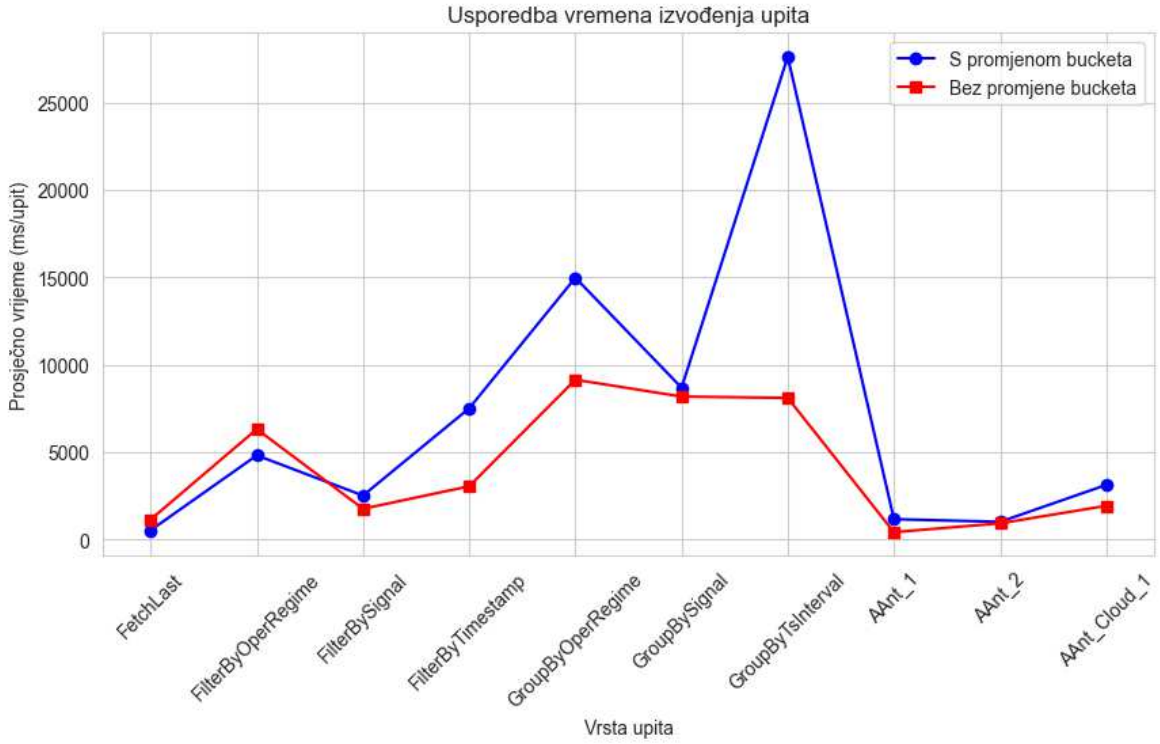
| Tip upita | MySQL AVG | Influx AVG | MySQL MAX | Influx MAX | MySQL MIN | Influx MIN |
|---|---|---|---|---|---|---|
| GroupByOperatingRegime | 3140,18 | 6117,96 | 62735 | 13274 | <1 | 1242 |
| GroupBySignal | 3561,3 | 5955,9 | 51371 | 13198 | <1 | 1278 |
| FetchLast | 20,95 | 29,37 | 758 | 115 | <1 | 14 |

Kako bi se moglo realnije usporedit vrijeme izvođenja iz statističke perspektive računanja srednje vrijednosti ili medijana, poželjno je bilo ipak ukloniti utjecaj predmemoriranja kod MySQL-a. To je izvedeno postavljanjem `SQL_NO_CACHE` zastavice nakon ključne riječi `SELECT` u MySQL upitima. Svi rezultati u radu dobiveni su korištenjem ove zastavice jer inače bi rezultati još više favorizirali MySQL.

**Predučitavanje indeksa vremenskih serija kod InfluxDB**

U ovom testiranju istražena je razlika između izvođenja sto uzastopnih upita na istom spremniku i izvođenja istih upita naizmjenično na različitim spremnicima. Rezultati (Slika 4.7) pokazuju da je prosječno vrijeme izvršavanja upita duže za većinu vrsta upita kada se spremnik mijenja za svaki upit. Ovaj rezultat je očekivan, jer InfluxDB prilikom spajanja klijenta na spremnik (bilo da je to Java program, InfluxDB konzola ili neki drugi klijent) najprije učitava indeksno stablo vremenskih serija u predmemoriju. Kontinuiranom promjenom spremnika za svaki upit, indeksno stablo se iznova učitava, pa izmjereno vrijeme trajanja upita uključuje i učitavanje stabla i samo izvršavanje upita. Kada se više upita izvršava uzastopno na istom spremniku, samo prvi upit ima produljeno vrijeme izvršavanja, dok su ostali upiti brži.

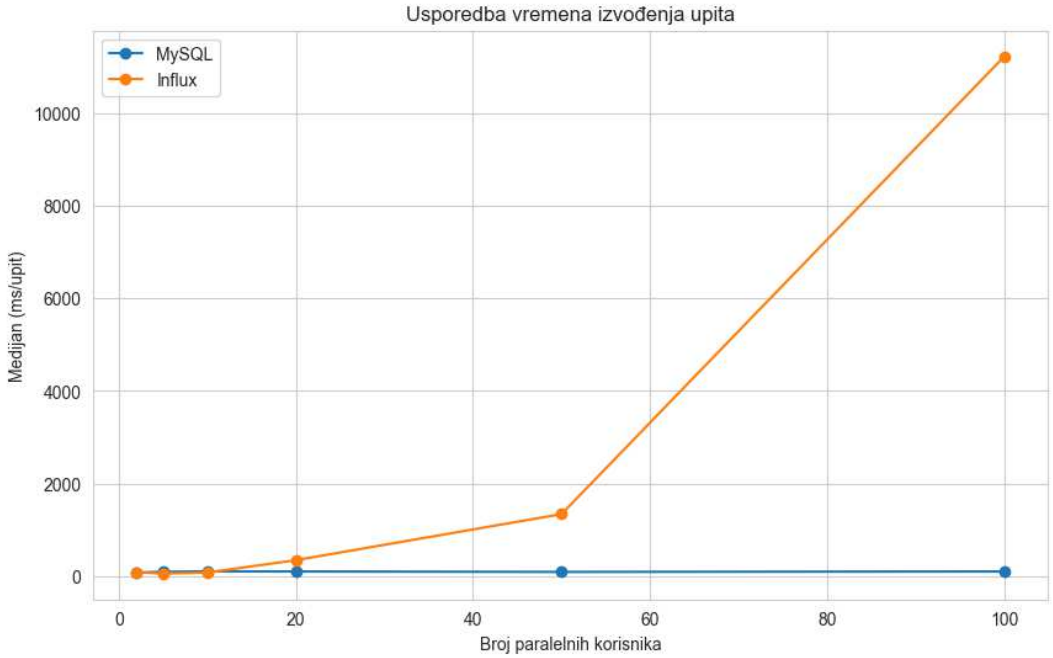
Posebno je uočljiva razlika u vremenu izvođenja upita koji filtriraju i grupiraju podatke po vremenu. Ovo se može objasniti činjenicom da InfluxDB pamti vremenske fragmente podataka kojima se posljednje pristupalo, a takvo memoriranje nema učinka kada se spremnik stalno mijenja.




Slika 4.7 Usporedba vremena izvođenja s promjenom i bez promjene spremnika

### Paralelno izvođenje upita

Malo je vjerojatno da će sustav u jednom trenu koristit samo jedan korisnik, stoga je potrebno ispitati kako se sustav ponaša pri većem opterećenju. Simulacija rada više paralelnih korisnika izvodi se pomoću dretvi u Javi. Stvorena lista upita, koja sadrži više vrsta upita popunjenih različitim nasumičnim vrijednostima, kao i u ostalim testovima, predaje se svakoj dretvi. Svaka dretva simulira jednog korisnika, koji izvodi upite iz liste nasumičnim redoslijedom. Testirano je opterećenje sustava s dva, pet, deset, dvadeset, pedeset i sto paralelnih korisnika. Dobiveni rezultati (Slika 4.8) ukazuju na to da MySQL pokazuje jednake performanse pri radu dva ili sto paralelnih korisnika što znači da je sustav vrlo efikasan. S druge strane, Influx počinje usporavati povećanjem broja korisnika, što je posebno izraženo pri radu sto paralelnih korisnika kada medijan vremena izvođenja upita prelazi deset sekundi.




Slika 4.8 Usporedba vremena izvođenja u ovisnosti broja korisnika

U tablici (Tablica 4.6) prikazani su svi statistički pokazatelji vremena izvođenja upita u InfluxDB-u i MySQL-u pri različitom broju paralelnih korisnika. Uočljiva je velika standardna devijacija u podacima što se može pripisati različitim vrstama upita koji inače imaju različita vremena izvođenja. No kao što je i iz grafičkog prikaza vidljivo, MySQL pokazuje stabilan medijan vremena izvođenja, dok kod InfluxDB-a medijan raste s povećanjem broja korisnika. Srednja vrijednost blago raste i kod MySQL-a, a to je uzrokovano porastom maksimalne vrijednosti. Taj porast može se protumačiti kao potencijalno zagušenje koje je u nekom trenutku tijekom testiranja značajno usporilo određeni upit. No, InfluxDB ima veće prosječne i maksimalne vrijednosti, što sugerira da su zagušenja kod njega bila češća ili izraženija.

Tablica 4.6 Statistički pokazatelji vremena izvođenja u ovisnosti broja korisnika
(svi brojevi izraženi su u milisekundama)

| 2 | | |
|---|---|---|
| | MySQL | Influx |
| mean | 172 | 406 |
| std | 309 | 964 |
| min | 2 | 8 |
| 25% | 19 | 30 |
| 50% | 72 | 82 |
| 75% | 221 | 246 |
| max | 3077 | 9130 |

| 5 | | |
|---|---|---|
| | MySQL | Influx |
| mean | 245 | 374 |
| std | 434 | 1515 |
| min | 1 | 7 |
| 25% | 9 | 24 |
| 50% | 89 | 52 |
| 75% | 313 | 143 |
| max | 4675 | 19307 |

| 10 | | |
|---|---|---|
| | MySQL | Influx |
| mean | 438 | 311 |
| std | 1114 | 897 |
| min | 1 | 6 |
| 25% | 15 | 35 |
| 50% | 99 | 73 |
| 75% | 286 | 168 |
| max | 13902 | 19670 |

| 20 | | |
|---|---|---|
| | MySQL | Influx |
| mean | 704 | 1370 |
| std | 2477 | 8439 |
| min | 1 | 13 |
| 25% | 17 | 202 |
| 50% | 96 | 341 |
| 75% | 261 | 628 |
| max | 36770 | 158619 |

| 50 | | |
|---|---|---|
| | MySQL | Influx |
| mean | 1643 | 1636 |
| std | 7708 | 2191 |
| min | 1 | 7 |
| 25% | 11 | 1009 |
| 50% | 87 | 1337 |
| 75% | 301 | 1736 |
| max | 130772 | 40631 |

| 100 | | |
|---|---|---|
| | MySQL | Influx |
| mean | 3715 | 30481 |
| std | 15437 | 56821 |
| min | 1 | 28 |
| 25% | 15 | 3775 |
| 50% | 96 | 11234 |
| 75% | 409 | 28597 |
| max | 127202 | 689823 |

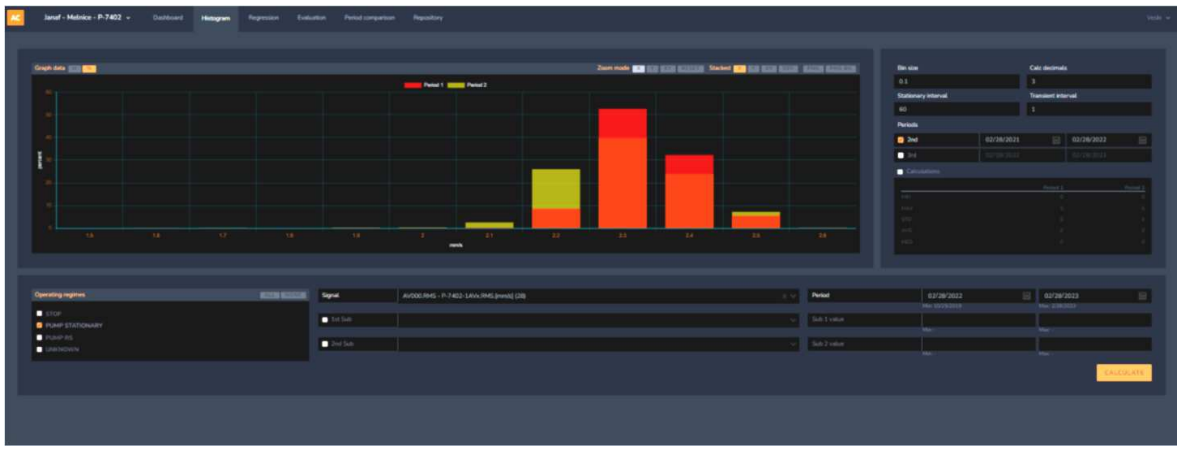
Pri interpretaciji svih rezultata, potrebno je uzeti u obzir hardverska ograničenja. Naime, MySQL poslužitelj radi na zasebnom virtualnom stroju, dok je InfluxDB poslužitelj pokrenut na istom virtualnom stroju na kojem se vrti i Java Benchmark program. Pri simulaciji većeg broja korisnika, pokreće se veći broj dretvi koje zauzimaju RAM memoriju InfluxDB poslužitelju, što potencijalno usporava njegov rad.

# 5. Poslužiteljska aplikacija

Ovo poglavlje opisuje poslužiteljsku aplikaciju razvijenu za komunikaciju klijentske AAnT Cloud aplikacije s InfluxDB poslužiteljem baze podataka. Svrha razvoja testne aplikacije je vidjeti je li tehnički izvedivo zamijeniti MySQL s InfluxDB-om i dobiti iste funkcionalnosti aplikacije te koliko bi takav programski sustav bio kompliciran. Stoga, ova aplikacija razvijena je za jedan statistički prikaz kao dokaz koncepta (engl. *Proof of concept* – POC) da je moguće dobiti isti rezultat iz novog modela podataka kao i iz postojećeg.

### Opis odabranog statističkog prikaza

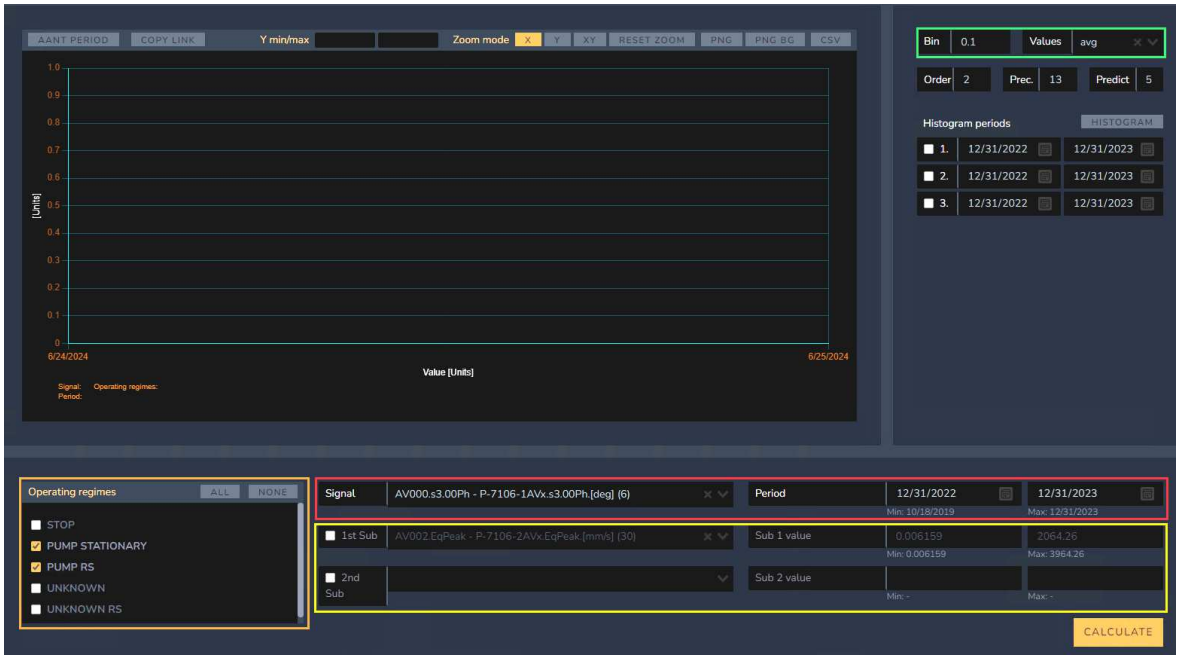
Odabrani statistički prikaz za dokaz koncepta je histogram koji omogućuje usporedbu trenda vrijednosti signala unutar raznih perioda (Slika 5.1)



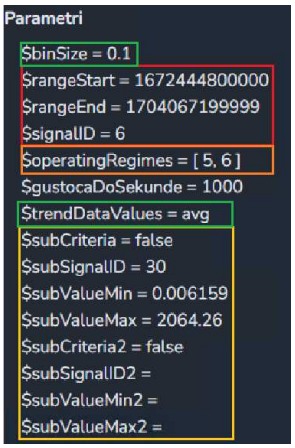
Slika 5.1 Snimka zaslona statističkog prikaza histogram

Na ekranu je dostupno mnogo izbornika kojima se odabiru parametri analize (Slika 5.2). Prvi korak je odabir željenog signala čiji trend želimo vidjeti i perioda u kojem to želimo vidjeti (Slika 5.2 – označeno crveno) te odabir operativnog režima (Slika 5.2 – označeno narančasto). Moguće je definirati dodatna dva podkriterija, uz operativni režim, kojima sužavaju analizu (npr. Operativni režim: „PUMP STATIONARY“, dodatni kriterij: temperatura ulja u ležaju od 30-40 °C) (Slika 5.2 – označeno žuto). U gornjem desnom uglu ekrana (Slika 5.2 – označeno zeleno) nalaze se kontrole za podešavanje točnosti prikaza signala te odabir agregatne funkcije. Ako je bin=1 rezolucija je na razini cijelog

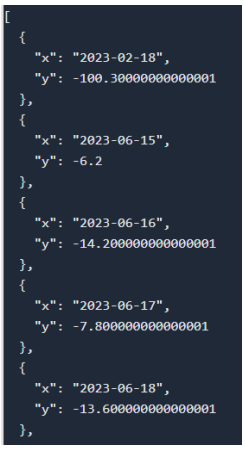
broja, a ako je bin=0.1 rezolucija prikaza histograma je na razini desetinke signala. Dostupne agregatne funkcije su: minimalna, maksimalna, srednja vrijednost te medijan. Nakon odabira svih željenih parametara, pritiskom na gumb „Calculate“ izvršava se zahtjev na poslužiteljsku aplikaciju koja šalje upit na bazu podataka. Zahtjev na aplikaciju sastoji se od parametara prikazanih na slici (Slika 5.3). Parametri su označeni istim bojama kao izbornici kojima se odabiru vrijednosti na snimci ekrana. Rezultat upita je u obliku liste x i y koordinata, pri čemu x predstavlja dan, a y vrijednost odabrane agregatne funkcije za pojedini dan (Slika 5.4).




Slika 5.2 Izbornici za postavljanje parametara analize




Slika 5.3 Parametri zahtjeva




Slika 5.4 Format odgovora

Upit koji ide na MySQL bazu snimljen je alatom Clockwork. Clockwork je alat za praćenje performansi i otklanjanje pogrešaka web aplikacija, posebno popularan među PHP razvojnim programerima, a postojeća klijentska aplikacija razvijena je upravo u PHP-u. Integrira se s aplikacijom i omogućava praćenje različitih aspekata izvođenja aplikacije, uključujući vrijeme izvođenja upita, memorijsko korištenje, te HTTP zahtjeve i odgovore.

Upit je vrlo kompleksan, posebice u slučaju korištenja podkriterija (Slika 5.5). Prvi dio upita (Slika 5.5 - označeno crveno) filtrira podatke prema odabranim signalima, režimima rada i vremenskom intervalu. Zatim se za svaki kriterij stvaraju odvojene privremene tablice. SUB0 (Slika 5.5 - označeno plavo) predstavlja tablicu u kojoj su podaci o signalu čiji se trend se analizira, dok je SUB1 (Slika 5.5 - označeno narančasto) tablica za podkriterij (da postoji više podkriterija, stvarale bi se tablice SUB2, itd.). Tablica SUB0 se spaja s podkriterijima (Slika 5.5 - označeno zeleno), uzimajući samo zapise za koje postoji mjerenje koje zadovoljava glavni kriterij i podkriterij u istom trenutku (tj. postoji zapis s vremenskim žigom u obje tablice). Ovaj korak osigurava da u analizu ulaze samo mjerenja za koja je podkriterij zadan ako postoji, ako podkriterija nema ovaj korak se preskače. Zadnji korak (Slika 5.5 - označeno žuto) je računanje maksimalne vrijednosti po danu, zaokružene na zadanu preciznost.

```
DROP TEMPORARY TABLE IF EXISTS MAIN;

CREATE TEMPORARY TABLE MAIN
    SELECT TIMESTAMP, SignalID_FK, OperatingRegimeID_FK, VectorValue, "D" AS S0, "D" AS S1, "D" AS S2, "D" AS S12
    FROM `codis-dm_unit2`.tvector_t1
    WHERE TIMESTAMP between 1651190400000 and 1654387199999
    AND SignalID_FK IN (39,26)
    AND OperatingRegimeID_FK in (5,7,8);
```

```
DROP TEMPORARY TABLE IF EXISTS SUB0;

CREATE TEMPORARY TABLE SUB0
    SELECT TIMESTAMP, SignalID_FK, OperatingRegimeID_FK, VectorValue, "D" AS S0, "N" AS S1, "N" AS S2, "N" AS S12
    FROM MAIN
    WHERE SignalID_FK = 39;
```

```
DROP TEMPORARY TABLE IF EXISTS SUB1;

CREATE TEMPORARY TABLE SUB1
    SELECT TIMESTAMP, SignalID_FK, OperatingRegimeID_FK, VectorValue, "N" AS S0, "D" AS S1, "N" AS S2, "D" AS S12
    FROM MAIN
    WHERE SignalID_FK = 26
    AND VectorValue between -179.932 and 179.983;
```

```
DROP TEMPORARY TABLE IF EXISTS final;


CREATE TEMPORARY TABLE FINAL
    SELECT u.TIMESTAMP, u.VectorValue, u.SignalID_FK, u.OperatingRegimeID_FK
    FROM (
            SELECT * FROM SUB0
            UNION ALL
            SELECT * FROM SUB1
        ) AS u
    GROUP BY u.TIMESTAMP
    HAVING count(u.TIMESTAMP)=2;
```

```
SELECT IF(zaokruzeno IS NULL, NULL, zaokruzeno) AS y, tabletime AS x, valuetimestamp
FROM
    (SELECT
        round( MAX(VectorValue) / 0.1, 0) * 0.1 AS zaokruzeno,
        LEFT(FROM_UNIXTIME(TIMESTAMP / 1000),10) AS tabletime,
        TIMESTAMP AS valuetimestamp
    FROM `FINAL` GROUP BY `tabletime`) AS a
GROUP BY `tabletime`
```

Slika 5.5 Upit u MySQL-u za dohvaćanje podataka za histogram

Za izradu poslužiteljske aplikacije odabran je model razvijen u trećem pristupu jer je taj model pokazao najbolje performanse pri testiranju upita tipičnih za sustav. Primjer upita koji vraća isti rezultat u Flux-u koristeći treći model kao upit u MySQL-u prikazan na slici (Slika 5.6).

```
import "math"
data = from(bucket: "codis-dm_unit2")
  |> range(start: 1654108695, stop: 1656700695)
  |> filter(fn: (r) => r["_measurement"] == "tvector_t1")
  |>filter(fn: (r) => r["_field"] == "AV003.DC" or r["_field"] == "AV013.RMS")
  |> filter(fn: (r) => r["OperatingRegime"] == "PUMP STATIONARY"
      or r["OperatingRegime"] == "UNKNOWN"
      or r["OperatingRegime"] == "UNKNOWN RS")

d0 = data
|> filter(fn: (r) => r["_field"] == "AV003.DC")

d1 = data
|> filter(fn: (r) => r["_field"] == "AV013.RMS" and r["_value"] > 0.0122679 and r["_value"] < 9.1142699 )

union(tables: [d0,d1])
  |> group()
  |> aggregateWindow(every: 1d, fn: max, createEmpty: false)
  |> map(fn: (r) => ({
    x: r._time,
    y: math.round(x: r._value/0.1)*0.1
}))
```

Slika 5.6 Upit u InluxDB-u za dohvaćanje podataka za histogram

### Tehnologija i arhitektura programskog rješenja

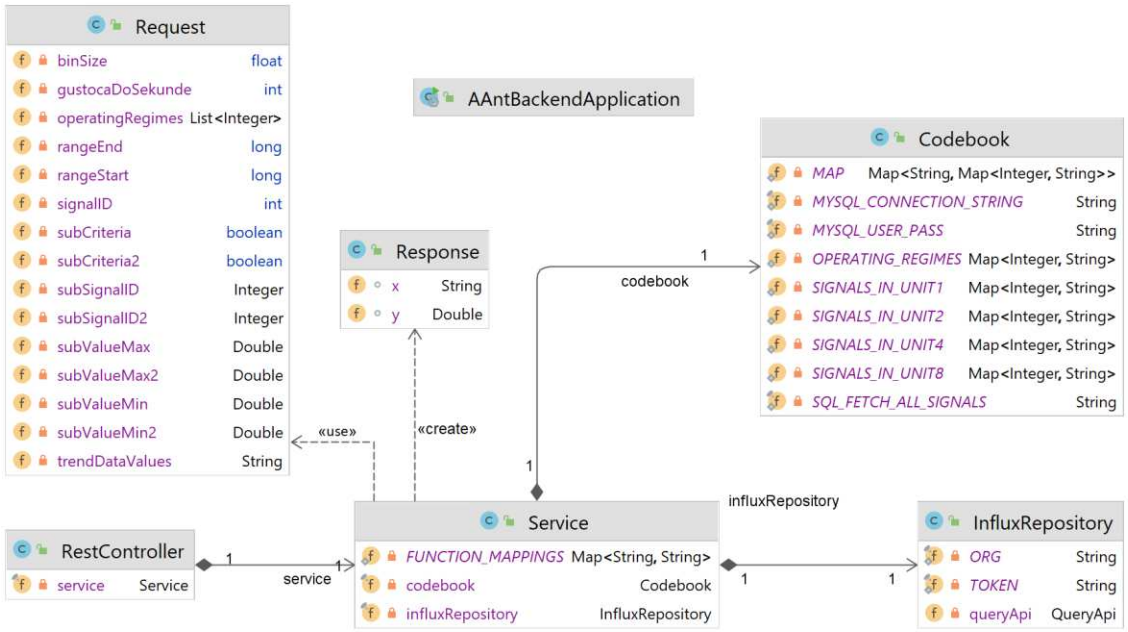
Cilj pri izradi POC aplikacije je omogućiti primanje zahtjeva i slanje odgovora istog oblika kao što prima i šalje postojeća aplikacija. Možemo zamisliti trenutnu poslužiteljsku aplikaciju kao crnu kutiju koju je potrebno zamijeniti.

Za izradu korišten je programski jezik Java verzije 17 te radni okvir Spring Boot verzije 3.3.0. Projekt je kreiran kao Maven projekt što omogućuje jednostavno dodavanje potrebnih biblioteka kao ovisnosti. Korištene biblioteke su:

- Spring Boot Starter Web - za izradu web aplikacija pomoću Spring Boota, uključujući RESTful web servise
- Spring Boot DevTools - za ubrzanje razvoja Spring Boot aplikacija omogućavanjem automatskog ponovnog učitavanja i drugih razvojnih alata
- Lombok - za smanjenje tzv. boilerplate koda u Java aplikacijama pomoću anotacija za generiranje uobičajenih metoda poput getter-a, setter-a, konstruktora, itd.
- InfluxDB Client - za komunikaciju s InfluxDB bazom podataka iz Java aplikacije
- MySQL Connector J - za povezivanje Java aplikacija s MySQL bazom podataka

Arhitektura programskog rješenja je jednostavna. Naslanja se na standardnu arhitekturu Spring aplikacija koja sadrži klase podijeljene u tri skupine klasa koje upravljaju tokom podataka: upravljači, servisi, repozitoriji te modele podataka. Ovo jednostavno rješenje

sadrži po jedan upravljač, servis i repozitorij te dva modela: model zahtjeva i odgovora (Slika 5.7).




Slika 5.7 Dijagram klasa

`RestController` je klasa koja prima HTTP zahtjeve i vraća odgovore. U ovom slučaju (Slika 5.8) mapiran je samo POST zahtjev pri čemu se u putanji zahtjeva navođenje ime jedinice iz koje dohvaćamo podatke, a u tijelo zahtjeva šalju se polja sa slike (Slika 5.3) koji se mapiraju u objekt `Request`. Odgovor na zahtjev je lista objekata `Response` pri čemu jedan objekt sadrži jedan par x i y, a lista se serijalizira u oblik kao na slici (Slika 5.4)

```
@org.springframework.web.bind.annotation.RestController
@RequiredArgsConstructor
public class RestController {
    1 usage
    private final Service service;
    no usages
    @PostMapping("/{unit}")
    public List<Response> getTrendData(@RequestBody Request request, @PathVariable("unit") String unit) {
        return service.getTrendData(request, unit);
    }
}
```

Slika 5.8 RestController klasa

Servisni sloj aplikacije zadužen je za poslovnu logiku. U metodi servisne klase (Slika 5.9) priprema se poziv na repozitorij te mapira odgovor upravljaču. Postoje dva slučaja upita koji ide na repozitorij: upit s i bez podkriterija te se ovisno o tome pozivaju različite funkcije iz repozitorija: `fetchTrendData` ili

`fetchTrendDataWithSubCriteria`. Repozitorij vraća odgovor u obliku `FluxRecord`-a (to je klasa iz InfluxDB Client biblioteke koja modelira rezultat koji vraća InfluxDB) te se taj odgovor treba preslikati u odgovor tražen na klijentskoj strani aplikacije.

```
public List<Response> getTrendData(Request request, String unit) {
    List<String> operatingRegimes = request.getOperatingRegimes().stream().map(a -> codebook.getOperatingRegime(a)).toList();

    var records = !request.isSubCriteria() && !request.isSubCriteria2()
            ?
            influxRepository.fetchTrendData(unit,
                    start: request.getRangeStart() / 1000,
                    stop: request.getRangeEnd() / 1000,
                    codebook.getSignal(unit, request.getSignalID()),
                    operatingRegimes,
                    FUNCTION_MAPPINGS.get(request.getTrendDataValues()),
                    request.getBinSize()
            )
            :
            influxRepository.fetchTrendDataWithSubCriteria(
                    unit,
                    start: request.getRangeStart() / 1000,
                    stop: request.getRangeEnd() / 1000,
                    codebook.getSignal(unit, request.getSignalID()),
                    operatingRegimes,
                    FUNCTION_MAPPINGS.get(request.getTrendDataValues()),
                    request.getBinSize(),
                    request.isSubCriteria(),
                    request.isSubCriteria() ? codebook.getSignal(unit, request.getSubSignalID()) : null,
                    request.getSubValueMin(),
                    request.getSubValueMax(),
                    request.isSubCriteria2(),
                    request.isSubCriteria2() ? codebook.getSignal(unit, request.getSubSignalID2()) : null,
                    request.getSubValueMin2(),
                    request.getSubValueMax2()
            );

    List<Response> responses = new ArrayList<>();
    for (var record : records) {
        responses.add(new Response((String.valueOf(record.getValueByKey("x")).substring(0, 10)),
                ((Number) record.getValueByKey("y")).doubleValue()));
    }
    return responses;
}
```

Slika 5.9 Metoda servisne klase

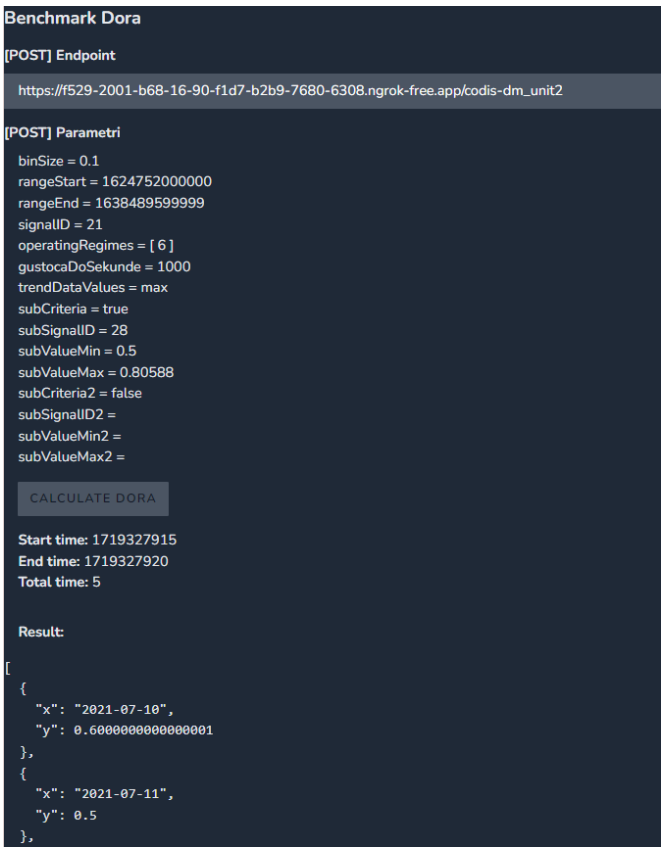
Metode u repozitoriju sadrže podatke potrebne za spajanje na InfluxDB poslužitelj (token, organizacija…) te imaju sljedeći tijek operacija: definira se predložak upita, upit se popunjava vrijednostima poslanim kao parametri metode, koristeći InfluxDB Client upit se izvršava na InfluxDB poslužitelju te se vraća rezultat.

Nova aplikacija čita podatke za statistički prikaz isključivo iz InfluxDB-a, no zahtjev koji se šalje iz klijentske aplikacije sadrži identifikatore signala, režima itd., a InfluxDB koristi oznake. S obzirom na to da se ovaj pristup oslanja na zamjenu isključivo poslužiteljske strane aplikacije te nije moguće promijeniti da u zahtjevu dođu potrebne oznake, napravljena je privremena pomoćna klasa `Codebook` koja pri pokretanju poslužiteljske

aplikacije učitava mapu svih identifikatora i odgovarajućih oznaka signala iz stare MySQL baze u memoriju aplikacije. Servisni sloj aplikacije stoga ima dodatnu ulogu da identifikatore iz zahtjeva, koristeći mape iz `Codebook`-a, pretvori u oznake koje su potrebne za upit na InfluxDB. U budućoj pravoj realizaciji programskog rješenja, ovo bi trebalo biti izbačeno te umjesto tog formiran zahtjev adekvatno za novi model.

### Izlaganje i testiranje aplikacije

Kako bismo se uvjerili da nova poslužiteljska aplikacija vraća iste rezultate za upite kao i postojeća, napravljena su dva posebna prikaza za testiranje na klijentskoj aplikaciji na putanjama /`bmtrendlocal` i /`bmtrenddora`. Oba prikaza sadrže formiran zahtjev nakon odabira parametara u izborniku, gumb za izračun koji šalje zahtjev poslužiteljskoj aplikaciji, vrijeme do dobivanja odgovora te rezultat. Dodatno, /`bmtrenddora` sadrži polje za upis endpoint-a na kojem se nalazi nova aplikacija (Slika 5.10).




Slika 5.10 Snimka ekrana za testiranje

Kako bi aplikacija bila dostupna klijentu, potrebno ju je izložiti (eng. *deploy*). Za to je korišten alat Ngrok koji omogućava izlaganje lokalnog poslužitelja na internetu preko sigurnog tunela, omogućavajući javni pristup lokalnim aplikacijama. Za instalaciju Ngroka

potrebno je napraviti korisnički račun te nakon instalacije generirati autentifikacijski token. Taj token se dodaje u lokalnu konfiguraciju Ngroka naredbom:

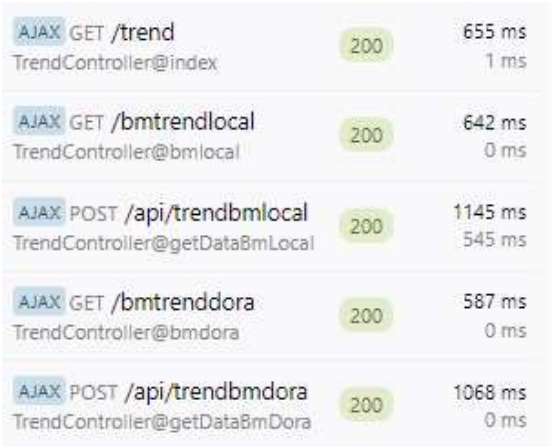
```
ngrok config add-authtoken <your-token>
```

Zatim je potrebno lokalno pokrenuti SpringBoot aplikaciju te izložiti aplikaciju naredbom:

```
ngrok http http://localhost:8080
```

Ngrok dodjeljuje aplikaciji privremenu prolaznu domenu pri svakom pokretanju.

Koristeći Clockwork praćeno je vrijeme izvođenja zahtjeva. Na slici (Slika 5.11) prikazan je primjer praćenja vremena izvođenja upita bez podkriterija. Prvi zahtjev predstavlja dohvaćanje izbornika u kojem biramo parametre analize. Zatim prelazimo na prikaz `/bmtrendlocal` s kojeg šalje POST zahtjev na staru poslužiteljsku aplikaciju koja koristi MySQL bazu podataka. Isto se ponavlja za novu poslužiteljsku aplikaciju koja koristi InfluxDB. Uočeno je (nakon provedbe nekoliko zahtjeva) da u slučaju bez podkriterija stara i nova aplikacija vraćaju rezultat u sličnom vremenu. Na slici (Slika 5.12) prikazan je primjer praćenja vremena izvođenja upita s podkriterijem. Pokazuje se da novoj aplikaciji treba znatno duže da vrati rezultat za upit s podkriterijem. To još jednom ukazuje na odličnu izvedbu i optimizaciju postojećeg sustava te potvrđuje da InfluxDB nije dovoljno dobra zamjena postojećem sustavu.




Slika 5.11 Praćenje izvođenja upita bez podkriterija


Slika 5.12 Praćenje izvođenja upita s podkriterijima

# Zaključak

Ovaj diplomski rad bavio se usporedbom performansi relacijskog sustava MySQL i specijaliziranog sustava za pohranu vremenskih serija InfluxDB u kontekstu obrade podataka za dijagnostiku vibracijskog stanja rotirajućih strojeva. Kroz detaljnu analizu postojećeg rješenja AAnT firme Veski, identificirani su ključni problemi i ograničenja tog sustava. Provedeno je istraživanje koncepata pohrane i modeliranja podataka te načina postavljanja upita za analizu podataka u InfluxDB-u. Nakon tog razvijena su četiri modela podataka u InfluxDB-u u pokušaju dobivanja ekvivalentnih performansi MySQL-a i InfluxDB-a. Testiranje performansi provedeno je prvo na generičkim upitima koji filtriraju ili grupiranju podatke po jednom atributu. Na tim jednostavnim testovima InfluxDB je pokazao načelno brže vrijeme izvođenja upita. Zatim je provedeno testiranje nad upitima tipičnim za promatrani sustav te su dobiveni rezultati kontradiktorni očekivanim, kao i rezultatima dobivenim mjerenjem vremena izvođenja generičkih upita. Ovi su upiti složeniji i očito je da je trenutni sustav odlično optimiziran upravo za svoje tipične upite. Razvijena je jednostavna poslužiteljska aplikacija kao dokaz koncepta da je moguće koristiti InfluxDB i dobiti isti rezultat za upite generirane iz AAnT korisničkog sučelja kao koristeći MySQL, no nažalost ne s jednakim performansama. S obzirom na to da su upiti, iako se radi o podacima vremenskih serija, djelomično relacijske naravi, u budućnosti bi se mogao razmotriti TimescaleDB kao zamjena za MySQL iz razloga što je TimescaleDB u suštini relacijski sustav sa proširenjem za efikasniji rad s vremenskim serijama.

# Literatura


[1] L. Lourenco*, Time Series Database vs Relational Database*, CrateDB (listopad 2023.) Poveznica: https://cratedb.com/blog/time-series-database-vs-relational-database; pristupljeno 26. lipnja 2024.

[2] Timeplus Team, Time-Series Database vs Relational Database: Key Differences, Timeplus (prosinac 2023.) Poveznica https://www.timeplus.com/post/time-series-database-vs-relational; pristupljeno 26. lipnja 2024.

[3] S. Yfantidou, S.N.Z. Naqvi, *Time Series Databases and InfluxDB*, Universite libre de Bruxelles, 2017.

[4] InfluxData Inc., *InfluxDB schema design*, InfluxData Documentation, (2024). Poveznica: https://docs.influxdata.com/influxdb/v2/write-data/best-practices/schema-design/; pristupljeno 26. lipnja 2024.

[5] InfluxData Inc., *Query data in InfluxDB*, InfluxData Documentation, (2024). Poveznica: https://docs.influxdata.com/influxdb/v2/query-data/; pristupljeno 26. lipnja 2024.

[6] O. Orešković, F. Tonković, *Upute za korištenje programskog paketa AAnT*, Interni dokument tvrtke Veski d.o.o. (ožujak 2023.)

[7] O. Orešković, F. Tonković, *Upute za korištenje analitičkog paketa AAnT Cloud*, Interni dokument tvrtke Veski d.o.o. (veljača 2023.)

[8] Oracle, *MySQL Product Support EOL Announcements,* MySQL Support, (2024). Poveznica: https://www.mysql.com/support/eol-notice.html; pristupljeno 26. lipnja 2024.

[9] A. Topbas, *Spring Boot Code Structure: Package by Layer vs Package by Feature*, Medium (svibanj 2024.), Poveznica: https://medium.com/@akintopbas96/spring-boot-code-structure-package-by-layer-vs-package-by-feature-5331a0c911fe; pristupljeno 26. lipnja 2024.

[10] Ngrok, *Setup & Installation on Windows*, Ngrok (2024.) Poveznica: https://dashboard.ngrok.com/get-started/setup/windows; pristupljeno 26. lipnja 2024.

# Sažetak

Naslov: Usporedba performansi relacijskog sustava i sustava specijaliziranog za pohranu vremenskih serija u radu s podacima za dijagnostiku vibracijskog stanja rotirajućih strojeva

Sažetak: Ovaj diplomski rad bavi se usporedbom performansi relacijskog sustava MySQL i specijaliziranog sustava za pohranu vremenskih serija InfluxDB u kontekstu obrade podataka za dijagnostiku vibracijskog stanja rotirajućih strojeva. Provedena je analiza sustava AAnT razvijenog u firmi Veski. Opisuju se ključni pojmovi i koncepti u InfluxDB-u. Razvijena su četiri pristupa modeliranju podataka u InfluxDB-u te provedeni testovi performansi nad jednostavnim upitima te upitima tipičnim za promatrani sustav te doneseni zaključci o isplativosti zamjene MySQL-a s InfluxDB-om u promatranom slučaju. Na kraju je razvijena jednostavna aplikacija koja povezuje postojeće AAnT korisničko sučelje s InfluxDB bazom podataka.

Ključne riječi: MySQL, relacijske baze podataka, vremenske serije podataka, InfluxDB, testiranje performansi

# Summary

Title: Performance comparison of relational databases and time series databases in handling vibration state of rotating machines

Summary: This thesis focused on comparing the performance of the relational database system MySQL with the specialized time series database system InfluxDB in the context of processing data for diagnosing the vibrational condition of rotating machinery. An analysis was conducted on the AAnT system developed by the company Veski. Key concepts and principles of InfluxDB were described. Four different approaches to data modelling in InfluxDB were developed, and performance tests were conducted on both simple queries and queries typical for the observed system. Conclusions were drawn regarding the feasibility of replacing MySQL with InfluxDB in the given scenario. Finally, a simple application was developed to connect the existing AAnT user interface with the InfluxDB database.

Keywords: MySQL, relational databases, time series data, InfluxDB, performance testing

# Skraćenice

| | | |
|---|---|---|
| SQL | *Structured Query Language* | strukturirani jezik upita |
| TSBS | *Time Series Database* | baza podataka vremenskih serija |
| RDBMS | *Relational Database Management System* | sustav za upravljanje bazama podataka |
| TSM | Time Structured Merge Tree | vremenski strukturirano stablo spajanja |
| TSI | Time Series Indeks | indeks vremenskih serija |
| CLI | Command Line Interface | sučelje naredbenog retka |
| UI | User Interface | korisničko sučelje |
| API | Application Programming Interface | programsko sučelje aplikacije |
| OSS | Open Source Software | softver otvorenog koda |
| RAM | Random Access Memory | memorija s nasumičnim pristupom |
| POC | Proof Of Concept | dokaz koncepta |