

VILNIAUS UNIVERSITETAS  
INFORMATIKOS INSTITUTAS  
PROGRAMŲ SISTEMŲ KATEDRA

# **Rodiklių duomenų apdorojimo sistemų generavimas iš duomenų modelių**

## **Generation of Indicator Data Processing Systems from Data Models**

Bakalauro baigiamasis darbas

Atliko:	Vytautas Žilinas	(parašas)
Darbo vadovas:	lekt. Andrius Adamonis	(parašas)
Recenzentas:	assoc. prof., dr. Karolis Petrauskas	(parašas)

Vilnius – 2019

# Santrauka

Šį darbą sudaro teorinė ir eksperimentinė dalis. Teorinėje dalyje apibrėžiamas rodiklis, jo struktūra ir struktūros pokyčiai. Apibrėžiami kokie yra įmanomi pokyčiai ir kaip eksperimentinėje dalyje kuriamas sprendimas prie pokyčių prisitaikys. Specifikuojama duomenų struktūra ir duomenų struktūrų apjungimo ir skirtumo operacijas. Apibrėžiamas srautinis apdorojimas ir pasirenkama sistema, kuri bus naudojama eksperimentinėje dalyje. Remiantis gautais rezultatais nustatoma, kad šiam uždaviniui spręsti pasirenkama "Heron" srautinio apdorojimo sistema. Remiantis pasirinkta srautinio apdorojimo sistema ir apibrėžta rodiklių duomenų struktūra, eksperimentui nusprendžiama generuoti srautinio apdorojimo modulius parašytus "Python" programavimo kalba. Eksperimentinėje dalyje remiantis pasiūlyta architektūra realizuojama bandomoji sistemos versija. Atliekant skirtingų kiekių rodiklių duomenų ir rodiklių duomenų pokyčių simuliaciją stebėjimais analizuojamas šios sistemos tinkamumas apibrėžtam uždaviniui spręsti. Gauti tyrimų rezultatai lyginami, pateikiamos išvados. Taip įrodoma, kad toks sprendimas gali būti įgyvendinamas ir kad kodo generavimas ir srautinio apdorojimo sistema "Heron" yra tinkamas sprendimas kintančių rodiklių uždaviniui spręsti.

**Raktiniai žodžiai:** srautinis apdorojimas, kodo generavimas, rodiklis, rodiklio duomens pokyčiai

## Summary

This work consists of a theoretical and an experimental part. Theoretical part defines the indicator, its structure and changes in indicator structure. It define what changes are possible and how the developed solution in the experimental part will adapt to the changes, specifies the data structure and data structure merging and difference operations, defines stream processing and selects the system to be used in the experimental part. Based on the results it is determined that "Heron" stream processing system is chosen to solve this task. Based on the selected stream processing system and the defined data structure of indicator, it is decided to generate streaming modules written in "Python" programing language. In the experimental part, a pilot version of the system is implemented based on the proposed architecture. By doing the simulation using varying amounts of indicators and indicator changed, the suitability of this system for a defined task is tested. The results of this research are compared and conclusions are given. This demonstrates that such a solution can be implemented and that the code generation and streaming system "Heron" is the right solution to deal with the challenge of changing indicators.

**Keywords:** stream processing, code generation, indicator, indactor structure change

## TURINYS

IVADAS .....	5
1. RODIKLIŲ DUOMENYS .....	8
1.1. Pirminiai rodiklių duomenys .....	8
1.2. Rodiklių duomenų modelis .....	9
1.2.1. Pirminis raktas .....	9
1.2.2. Pirminių rodiklių duomenų apribojimai.....	10
1.2.3. Rodikliai .....	10
1.3. Rezultatų struktūra .....	12
1.4. Rodiklių duomenų struktūros kitimas .....	12
2. SRAUTINIO DUOMENŲ APDOROJIMO SPRENDIMŲ ANALIZĖ .....	14
2.1. Srautinis duomenų apdorojimas .....	14
2.2. Srautinio duomenų apdorojimo sistemos .....	15
2.3. Pristatymo semantika .....	15
2.4. Uždelstumas .....	16
2.5. Pralaidumas .....	16
2.6. Abstrakcijos lygmuo .....	17
2.7. Apibendrinimas .....	18
3. SRAUTINIO DUOMENŲ APDOROJIMO SISTEMŲ ARCHITEKTŪRA .....	19
3.1. Srautinio apdorojimo sistemos komponentai .....	19
3.2. Pagal rodiklių duomenų modelį sugeneruotos srautinio apdorojimo sistemos .....	20
4. EKSPERIMENTAS IR SUKURTO SPRENDIMO SAVYBĖS .....	22
4.1. Eksperimento tikslas .....	22
4.2. Eksperimento vykdymo aplinka .....	22
4.3. Konceptinis sprendimas .....	22
4.3.1. Sistemų generavimo komponentas .....	23
4.3.2. Apdorotų rodiklių duomenų bazė .....	25
4.3.3. Papildoma programinė įranga .....	25
4.4. Eksperimentas .....	26
4.4.1. Pradiniai duomenys .....	26
4.4.2. Eksperimento eiga.....	27
4.4.3. Eksperimento rezultatai .....	28
4.5. Sprendimo savybės.....	28
REZULTATAI .....	29
IŠVADOS .....	30
LITERATŪRA .....	31
PRIEDAI .....	32
1 priedas. Pirminių rodiklių duomenų generavimo įrankis .....	33
2 priedas. Rodiklių duomenų modelis JSON formatu.....	34
3 priedas. Sugeneruota srautinio apdorojimo sistema "Heron" sąsajoje .....	36
4 priedas. Sugeneruotos srautinio apdorojimo sistemos rezultatai .....	37
5 priedas. Atnaujintas rodiklių duomenų modelis JSON formatu .....	38
6 priedas. Atnaujintos srautinio apdorojimo sistemos "Heron" sąsajoje .....	40
7 priedas. Atnaujintos srautinio apdorojimo sistemos rezultatai .....	41

## Įvadas

Šiame darbe yra nagrinėjamas rodiklių duomenų apdorojimas ir kuriamas koncepcinis sprendimas galintis prisitaikyti, kai keičiama rodiklių duomenų struktūra. Rodiklių duomenimis vadiname pasikartojančių įvykių parametrus aprašančius duomenis. Pavyzdžiui, įvairių matuoklių – temperatūros, resursų suvartojimo – fiksuojamus rodmenis, kas mėnesinius veiklos indikatorius, tokius kaip veiklos finansines ataskaitas ar veiklos procesų indikatorius. Taip pat rodiklių struktūra gali keistis laikui bėgant: objektų atributų taksonomija (pvz. mirties priežasčių sąrašas, finansinių sąskaitų sąrašas) arba įrašo atributų sąrašai. Surenkamu rodiklių duomenų kiekis visada didėja, taip pat ir duomenų kiekis, kuriuos reikia apdoroti pagal rodiklius auga, todėl standartiniai sprendimai, pavyzdžiui, reliacinės duomenų bazės netinka dėl ilgos apdorojimo trukmės. Rodiklių duomenų bazės pasižymi tuo, kad duomenys į jas patenka iš daug skirtingų tiekėjų ir patekimo laikas tarp tiekėjų nėra sinchronizuojamas, o suagreguotą informaciją vartotojai gali užklausti bet kurio metu. Todėl šiame darbe bus nagrinėjamas srautinis duomenų apdorojimas, kuris patenkančius duomenis apdoroja realiu laiku, ir saugos jau apdorotus.

Realaus laiko duomenų apdorojimas (angl. Real-time data processing) yra jau senai nagrinėjamas, kaip vienas iš būdų apdoroti didelių kiekių duomenis (angl. Big data). Vienas iš realaus laiko apdorojimo sprendimų yra srautinis duomenų apdorojimas. Srautinis duomenų apdorojimas (angl. stream processing) – lygiagrečių programų kūrimo modelis, pasireiškiantis sintaksiškai sujungiant nuoseklius skaičiavimo komponentus srautais, kad kiekvienas komponentas galėtų skaičiuoti savarankiškai [Bea15]. Darbe yra analizuojamos jau egzistuojančias srautinio apdorojimo sistemų programas pagal srautinio apdorojimo programų savybes aprašytas Michael Stonebraker, pasirenkama vieną srautinio apdorojimo programa ir pagal ją sukuriamas koncepcinis sprendimas.

Kadangi rodikliai laikui bėgant gali būti keičiami reikia, kad sprendimas, galėtų prisitaikyti prie pokyčių. Yra keli būdai kaip tai tokie sprendimai gali būti kuriami:

- Rankinio atnaujinimo sprendimas. Sukuriamas sprendimas pagal esamus reikalavimus, o atsiradus naujiems reikalavimams būtų kuriamos naujos arba keičiamos esamas apdorojimo sistemos.
- Universalus sprendimas. Sukuriamas universalus parametrizuojamas sprendimas ir pritaikomas užduotims nustatant parametrus.
- Kodo generavimo sprendimas. Sukuriamas sprendimas, kuris generuoja srautinio duomenų apdorojimo sistemas pagal iš anksto aprašytą struktūrą.

Tinkamas sprendimas turi būti išrinktas pagal sprendžiamą problemą. Jei nėra numatomas kitimas,

pagal ką turi būti apdorojami duomenys, tai galima pasirinkti ir rankinio apdorojimo sprendimą, kadangi nėra didelės tikimybės, kad teks keisti sprendimą. Toks sprendimas tikėtų apdorojant išmaniųjų skaitiklių duomenis [Nev17]. Universalus sprendimas taip pat gali būti tinkamas jei pirminiai rodiklių duomenis yra specifiški ir yra poreikis juos visus apdoroti. Toks sprendimas gali būti aktualus apdorojant duomenis iš sensorių, kurie matuoja namų būseną (temperatūra, drėgmė ir t.t.) ir bet koks naujas sensorius taip pat turi būti prijungtas ir apdorotas [Yan17]. Šiame darbe buvo norima nagrinėti kodo generavimo sprendimo tinkamumą užduotims.

Pagal Jack Herrington 2003 metų knygą "Code Generation in Action" kodo generavimas - tai rašymas programinės įrangos, kuri rašys reikiamą programinę įrangą problemai spręsti. Tai daroma tokiais atvejais, kai sprendžiama problema reikalauja daug rankinio darbo, kurį įmanoma automatizuoti. Kuo didesnio sprendimo reikalauja uždavinys tuo patraukliau tampa naudoti kodo generavimą sprendimo kūrimui. Kodo generavimas suteikia tokius privalumus:

- Architektūrinį nuoseklumą:
  - Verčia programuotojus labiau mąstyti apie architektūrą.
  - Jei sunku "priversti" generatorių generuoti reikiamą kodą, problema gali būti architektūroje.
  - Geros dokumentacijos buvimas sumažina problemą, kai nariai palieką projektą.
- Abstrakciją:
  - Programuotojai galės kurti naujus šablonus, kurie leis esamą funkcionalumą pritaikyti kitomis kalbomis, sprendimais daug paprasčiau negu rankomis parašytą kodą.
  - Verslo analitikai gali apžvelgti ir patvirtinti sprendimo abstrakciją.
  - Abstrakcija padės paprasčiau paruošti dokumentaciją, testavimo atvejus, produkto palaikymo medžiagą ir t.t.
- Aukštą komandos moralę - rašomas kodas bus nuoseklus ir kokybiškas todėl kels komandos pasitikėjimą.
- Tinkamas sprendimas Judriajam programavimui, kadangi kodo generavimo kuriami sprendimai yra lankstesni, tai leidžia ateityje juos lengviau keisti ir atnaujinti.

Kodo generavimas tampa tikrai naudingas tada, kai jis naudojamas didelių kiekių rankiniam kodavimui pakeisti [Her03].

Darbe nagrinėjamas architektūra ir koncepcinis sprendimas generuojantis srautinio apdorojimo sistemas pagal rodiklių duomenų modelius. Rodiklių duomenis apdorojantis koncepcinis sprendimas turi pasižymėti tokiomis savybėmis:

- Srautinių apdorojimo sistemų generavimas pagal deklaratyvų aprašymą.

- Galimybė keisti esamas apdorojimo sistemas, kai pakeičiamas rodiklių duomenų modelis.
- Išvestinių rodiklių gavimas iš daugiau nei vieno rodiklio transformacijos.
- Išvestinių rodiklių apdorojimas pagal iš anksto apibrėžtas funkcijas.

Tikslas: Sukurti rodiklių duomenų srautinio apdorojimo sistemos, pritaikomos prie duomenų struktūrų naudojant kodo generavimą, architektūrą.

Uždaviniai:

1. Apibrėžti rodiklių duomenų modelį ir galimus rodiklių duomenų struktūros pokyčius.
2. Atlikus šaltinių analizę pasirinkti srautinio duomenų apdorojimo programinę įrangą.
3. Sudaryti srautinio apdorojimo sistemos architektūrą, pagal ją sukurti koncepcinį sprendimą.
4. Nustatyti sukurto koncepcinio sprendimo ir architektūros savybes atliekant bandymus su rodiklių duomenimis.

# 1. Rodiklių duomenys

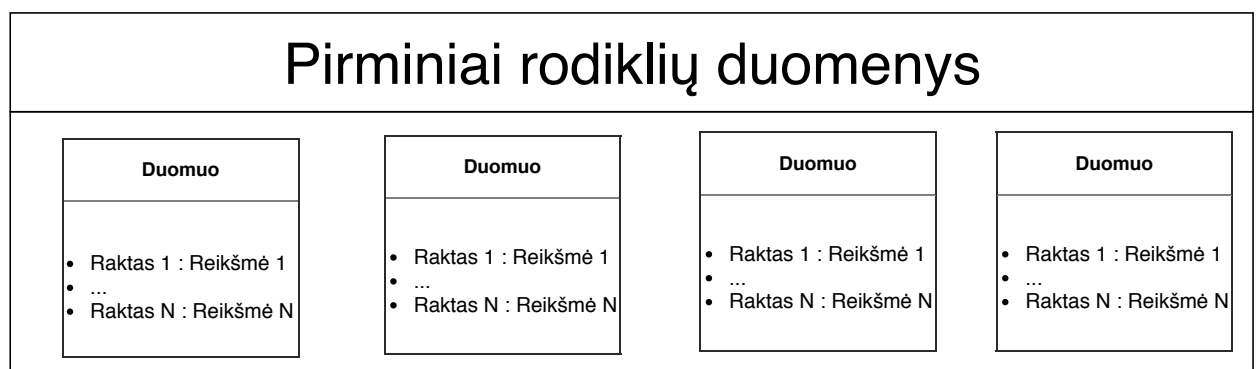
Rodiklių duomenys - pasikartojančių įvykių parametrus aprašančius duomenys. Rodiklių duomenys pasižymi tuo, jog tik surinkti ir apdoroti tampa aktualus. Taip pat rodikliai yra nepastovūs, jų struktūra gali būti keičiama - atsirasti naujų savybių, kurias reikia surinkti arba yra pašalinamos jau esamos savybės. Rodiklių duomenys yra renkami ir analizuojami labai skirtingiems poreikiams - visur kur reikalinga statistika ir apdorojami didelį kiekį duomenų yra naudojami rodikliai: tai gali būti įmonės vedamą "Microsoft Excel" skaičiuoklė, kurioje užrašyta kiek buvo mokėta už elektrą kiekvienais metais per paskutinius 10 metų arba juodosios skylės nuotrauka padaryta koreliuojant didelio duomenų kiekį iš 8 skirtingų teleskopu ir gauti rodikliai rodo taškus ant paveikslėlio, kurio dydis yra  $10^{12}$  kartų mažesnis nei pradiniai duomenys [AAA<sup>+</sup>19].

Pirmiausiai turime apibrėžti bendrą rodiklių duomenų modelį, kurio pagalba galėtume apibrėžti rodiklius.

## 1.1. Pirminiai rodiklių duomenys

Rodiklių duomenys yra gaunami apdorojus pirminius rodiklių duomenis, kurie susidaro iš raktų ir reikšmių žodyno (1 pav.), kur raktas gali būti tik tekstinio tipo, o reikšmė - tekstas arba skaičius. Duomenų raktai ir raktų kiekis gali nesutapti, vieni žodynai gali turėti daugiau raktų-reikšmių elementų, kitų raktai gali būti visiškai nesusiję su apdorojamais rodikliais.

Kadangi pirminiai rodiklių duomenys gali būti gaunami iš skirtingų šaltinių, skirtingais laikais, todėl kuriamas sprendimas turi galėti priimti duomenis asinchroniškai ir iš skirtingų šaltinių, o sprendimas realiu laiku turi sugebėti apdoroti reikiamus duomenis, o nereikalingus atmesti.



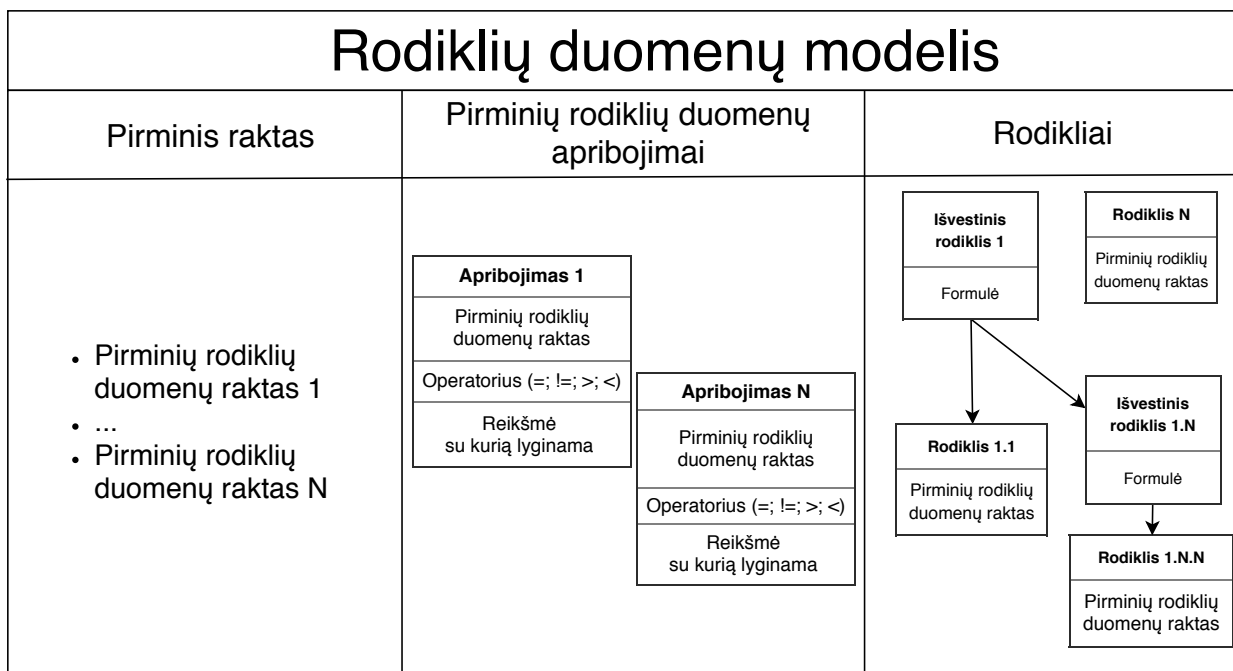
1 pav. Pirminiai rodiklių duomenys



## 1.2. Rodiklių duomenų modelis

Norit apdoroti daug skirtingų rodiklių turime apsibrėžti bendrą rodiklių duomenų modelį (2 pav.), pagal kurį bus generuojamos srautinio duomenų apdorojimo sistemos. Siūlomas apibrėžimas susidaro iš trijų dalių:

- Rodiklių duomenų pirminis raktas sudarytas iš pirminių rodiklių duomenų raktų rinkinio.
- Pirminių rodiklių duomenų reikšmių srities apribojimai.
- Rodiklių.



2 pav. Rodiklių duomenų modelis

### 1.2.1. Pirminis raktas

Pirma rodiklio apdorojimo apibrėžimo dalis nusako raktų rinkinį reikšmių pagal kurias grupuojami apdoroti duomenis. Pavyzdžiui:

- Statistikos srityje: norint surinkti duomenis aprašančius **skirtingų savivaldybių** nekilnojamo turto kainas pagal **metus**, raktų rinkinys atrodys taip:

{ "Savivaldybė", "Metai" }

- Sensoriniai rodikliai: norint surinkti vienos dienos duomenis aprašančius **skirtingų sensorių** parodymus **kiekvieną valandą**, **kiekvienam ofiso kabinetui**, raktu rinkinys atrodys taip:

{ "Sensoriaus tipas", "Valanda", "Kabineto Numeris" }

- Buhalterijoje: norint surinkti duomenis parodančių kiek ir **kokie skyriai** patyrė išlaidų kiekvieną šių **metų mėnesį**, raktų rinkinys atrodys taip:

$$\{ "Skyriaus pavadinimas", "Mėnesis" \}$$

Raktų kiekis neturi būti ribojamas, kadangi realiame pasaulyje yra neribotas kiekis duomenų pagal kuriuos galima grupuoti. Visų raktų reikšmių kiekių sandauga - apdorotų rodiklių duomenų pirminių raktų aibė.

### 1.2.2. Pirminių rodiklių duomenų apribojimai

Antra rodiklio duomenų modelio dalis susidaro iš rinkinio apribojimų, kurie aprašo kokie duomenys turi būti apdorojami. Apribojimai yra aprašomi predikatais. Pavyzdžiui:

- Statistikos srityje: norint surinkti žmonių, **vyresnių nei 60 metų** atlyginimus pagal kalendorinius metus, apribojimų sąrašas atrodys taip:

$$"Amžius" > 59$$

- Sensoriniai rodikliai: norint surinkti **tik savaitgalio** sensorių rodiklius, apribojimų sąrašas atrodys taip:

$$"Savaitės Diena" > 5, "Savaitės Diena" < 8$$

- Buhalterijoje: norint apdoroti tam tikrus duomenis **tik "Marketingas" skyriaus** apribojimų sąrašas atrodys taip:

$$"Skyriaus pavadinimas" == "Marketingas"$$

Apribojimų kiekis yra neribojamas. Šie apribojimai turi leisti naudoti tuos pačius duomenis skirtingiems rodikliams ir rodiklių rinkiniams apdoroti. Pirminiai rodiklių duomenys turi būti ribojami pagal reikšmes, kurios turi priklausyti apribojimų aibių sankirtai. Jei yra pateikti du apribojimus, tai pirminio rodiklio duomens ribojamos reikšmės turi patekti į abiejų apribojimų aibes.

### 1.2.3. Rodikliai

Ši rodiklių duomenų modelio dalis sudaryta iš dviejų skirtingų tipų komponentų:

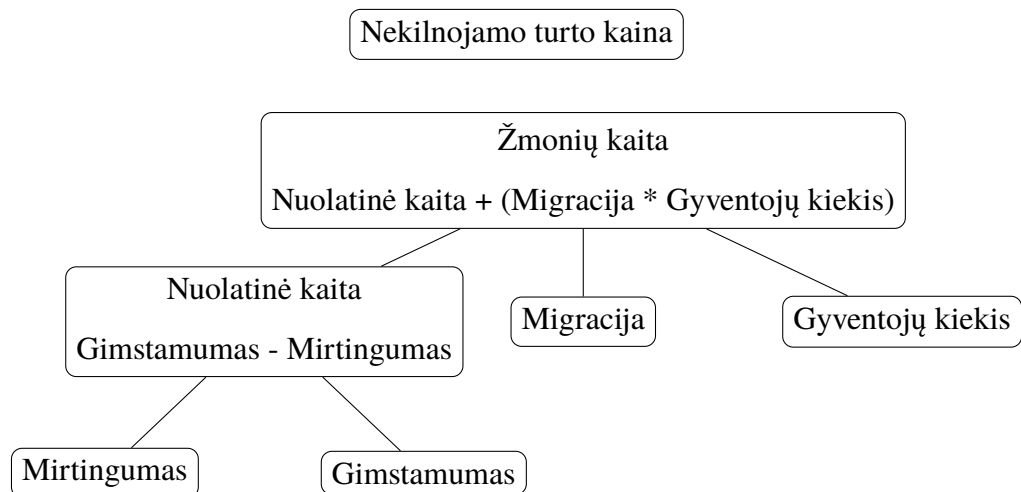
- Rodiklis - iš pirminių rodiklių duomenų gaunami parametrai. Rodiklių duomenų modelyje

rodiklis yra sudarytas iš pavadinimo ir pirminių rodiklių duomenų rakto.

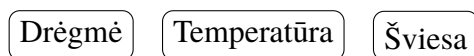
- Išvestinis rodiklis - iš vieno ar daugiau rodiklių išskaičiuojami parametrai. Rodiklių duomenų modelyje išvestinis rodiklis yra sudarytas iš pavadinimo ir rodiklių, nuo kurių priklauso išvestinis rodiklis, transformacijos aprašo - formulės.

Rodiklių duomenų modelyje komponentų priklausomybės sudaro priklausomybių medį, kurio viršūnės - rodikliai, o keliai - priklausomybės tarp rodiklių. Priklausomybių medyje kabančios (galinės) viršūnės visada bus rodiklio tipo, o visos kitos išvestinių rodiklių tipo. Pavyzdžiui:

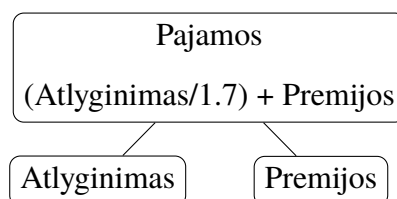
- Statistikos srityje: norint surinkti rodiklių duomenis nekilnojamo turto kainų ir žmonių kaitos šalyje ( $\text{Žmonių kaita} = (\text{Gimstamumas} - \text{Mirtingumas}) + (\text{Migracija} * \text{Gyventojų kiekis})$ ), renkamų rodiklių medžiai apsirasytų taip:



- Sensoriniai rodikliai: norint surinkti drėgmės, temperatūros ir šviesos sensorių rodiklius, rodiklių medžiai atrodys taip:



- Buhalterijoje: norint suskaičiuoti kiek įmonės darbuotojai gauna pajamų ( $\text{Pajamos} = (\text{Atlyginimas}/1.7) + \text{Premijos}$ ) rodiklių medis bus toks:



### 1.3. Rezultatų struktūra

Apdorotų rodiklių rezultatai gauti iš pirminių rodiklių duomenų gali būti užrašyti lentelę. Lentelės duomenų pirminis raktas yra sudarytas iš pirminių rodiklių duomenų reikšmių kombinacijos gautos pagal rodiklio modelio pirminį raktą. Rezultatų lentelės stulpeliuose aprašytas pirminis raktas ir išvestiniai rodikliai. Rezultatų lentelės išvestinių rodiklių kolonėlės susidaro iš apdorotų rodiklių duomenų.

Rezultatai			
Pirminis raktas	Rodiklis 1	...	Rodiklis N
[Duomo[Raktas1], ..., Duomo[RaktasN]]	Apdoroti duomenys 1	...	Apdoroti duomenys N
...	...	...	...
[Duomo[Raktas1], ..., Duomo[RaktasN]]	Apdoroti duomenys N	...	Apdoroti duomenys N

3 pav. Rezultatų struktūra

### 1.4. Rodiklių duomenų struktūros kitimas

Rodiklių duomenų struktūra gali būti keičiama laikui bėgant. Buhalterijos srityje tai gali būti dėl naujo įstatymo, kuris pakeičia kažkurių rodiklių skaičiavimo formulę. Tai gali būti naujos rūšies išmaniųjų namų sensorius, kuriuo duomenys norima irgi rinkti, kad galėtume matyti pilną vaizdą. Statistikos srityje gali atsirasti poreikis išskaidyti tam tikrus rodiklius į mažesnes dalis, pavyzdžiui rinkti ne migracijos rodiklį kaip vieną, o atskirai emigraciją ir imigraciją.

Pagal mūsų aprašyta rodiklių duomenų modelį, mes taip pat turime apsibrėžti kas gali keistis:

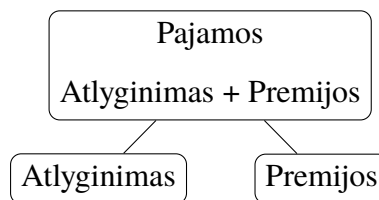
- Pirminis raktas negali keistis, nes naujos reikšmės tampa nepalyginamos su anksčiau surinktomis. Tarkim buvo renkami rodikliai metinių duomenų, o pagal poreikius reikia rinkti mėnesinius duomenis, ir lyginimas su anksčiau surinktais tampa neįmanomas. Todėl bet kokius pirminio raktus pokyčius turime laikyti nauju rodikliu.
- Apribojimai gali keistis, tačiau šie pokyčiai turi turėti prasmę ir naudojami tik tam, kad būtų išlaikytos tos pačios pirminių rodiklių duomenų aibės. Tarkime mums reikėjo apdoroti duomenis iš visų skyrių išskyrus administraciją ir tokiam uždaviniui buvo sukurtas apribojimas - "*Skyriaus pavadinimas*" != "*Administracija*". Tačiau po laiko dalis administracijos skyriaus atsiskyrė ir susikūrė naujas HR skyrius. Norint,

kad būtų išlaikyta apdorotų rodiklių lyginimo prasmė, nauji apribojimai atrodys taip:  
"Skyriaus pavadinimas"!="Administracija", "Skyriaus pavadinimas"!="HR"

- Rodikliai ir išvestiniai rodikliai gali būti keičiami: rodiklių medžio viršūnės gali būti pridedamos ir pašalinamos. Operacijos su rodikliais neturi įtakoti kitų rodiklių būsenos ir modelį. Tarkime jog skaičiavome pajamas naudodami formulę:

$$Pajamos = Atlyginimas + Premijos$$

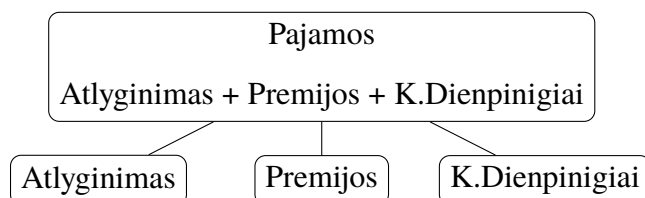
Ir pagal šią formulę buvo sukurtas toks rodiklių medis:



Po tam tikro laiko atsirado poreikis pajamas skaičiuoti pagal formulę:

$$Pajamos = Atlyginimas + Premijos + Komandiruočių dienpinigiai$$

Pagal naują formulę rodiklių medis turės atrodyti taip:



## 2. Srautinio duomenų apdorojimo sprendimų analizė

### 2.1. Srautinis duomenų apdorojimas

Siekiant apžvelgti modernius srautinio duomenų apdorojimo sprendimus turime apsibrėžti jų savybes. 2005 metais Michael Stonebraker apibrėžė 8 taisykles realaus laiko (angl. real-time) srautinio duomenų apdorojimo architektūrai [SÇZ05]:

- 1 taisyklė: Duomenys turi judėti. Žemo uždelstumo užtikrinimui sistema turi apdoroti duomenis nenaudojant duomenų saugojimo operacijų. Taip pat sistema turi ne pati užklausti duomenų, o gauti juos iš kito šaltinio asinchroniškai.
- 2 taisyklė: Duomenų transformacijos turi būti vykdomas SQL pobūdžio užklausomis. Žemo abstrakcijos lygmens srautinio apdorojimo sistemos reikalauja ilgesnio programavimo laiko ir brangesnio palaikymo. Tuo tarpu aukšto abstrakcijos lygmens sistema naudojanti SQL užklausas, kurias žino dauguma programuotojų ir yra naudojamos daugelyje skirtingų sistemų, leidžia efektyviau kurti srautinio apdorojimo sprendimus.
- 3 taisyklė: Architektūra turi susidoroti su duomenų netobulumais. Architektūra turi palaikyti galimybę nutraukti individualius skaičiavimus tam, kad neatsirastų blokuojančių operacijų, kurios sustabdo vieno modulio veikimą ir tuo pačiu visos architektūros veikimą. Taip pat ši architektūra turi sugebėti susidoroti su vėluojančiomis žinutėmis, pratęsiant laiko tarpą, per kurį ta žinutė turi ateiti.
- 4 taisyklė: Architektūra turi būti deterministinė. Kiekvieną kartą apdorojant tuos pačius duomenis rezultatai turi būti gaunami tokie patys.
- 5 taisyklė: Architektūra turi gebėti apdoroti išsaugotus duomenis ir realiu laiku gaunamus duomenis. Sistema parašyta su tokia architektūra turi galėti apdoroti jau esančius duomenis taip pat kaip ir naujai ateinančius. Toks reikalavimas buvo aprašytas, nes atsirado poreikis nepastebimai perjungti apdorojimą iš istorinių duomenų į realiu laiku ateinančius duomenis automatiškai.
- 6 taisyklė: Architektūra turi užtikrinti duomenų saugumą ir apdorojimo prieinamumą. Kadangi sistema turi apdoroti didelius kiekius duomenų, architektūra klaidos atveju, turi sugebėti persijungti į atsarginę sistemą ir tęsti darbą toliau. Taip pat tokios klaidos atveju atsarginė sistema turi būti apdorojusi visus duomenis ir sugebėti iš karto priimti naujus duomenis, o ne apdoroti duomenis iš pradžių.
- 7 taisyklė: Architektūra turi užtikrinti sugebėjimą paskirstyti sistemos darbus automatiškai.

Srautinio apdorojimo sistemos turi palaikyti kelių procesoriaus gijų operacijas. Taip pat sistema turi galėti veikti ant kelių kompiuterių vienu metu ir prireikus paskirstyti resursus pagal galimybes.

- 8 taisyklė: Architektūra turi apdoroti ir atsakyti akimirksniu. Anksčiau minėtos taisyklės nėra svarbios, jeigu sistema nesugeba greitai susidoroti su dideliu kiekiu naujų duomenų. Todėl turi būti naudojamas ne tik teisingas ir greitas srautinio apdorojimo sprendimas, bet ir gerai optimizuota sistema.

Todėl norint sužinoti tam tikro srautinio duomenų apdorojimo sprendimo tinkamumą uždaviniui reikia išanalizuoti jų savybes.

## 2.2. Srautinio duomenų apdorojimo sistemos

Norime nustatyti srautinio apdorojimo sistemų savybes pagal:

- Pristatymo semantika (angl. delivery semantics) - apibrėžia pagal kokį modelį bus pristatyti duomenys. Egzistuoja trys semantikos [Kah18]:
  - Bent vieną kartą (angl. at-least-once) užtikrina, kad duomenys bus apdoroti bent kartą, bet gali atsirasti dublikatų.
  - Ne daugiau vieno karto (angl. at-most-once) užtikrina, kad duomenys bus apdoroti daugiausiai tik vieną kartą, bet gali atsirasti praradimų.
  - Tiksliai vieną kartą (angl. exactly-once) užtikrina, kad duomenys bus apdoroti tik vieną kartą ir klaidos bus suvaldytos.
- Uždelstumas (angl. latency) - apibrėžia laikų sumą - kiek laiko trūko viena operacija ir kiek laiko ši operacija turėjo laukti eilėje kol bus pabaigtos kitos operacijos [KRK<sup>+</sup>18].
- Pralaidumas (angl. throughput) - apibrėžia kiek pavyks įvykdyti operacijų per tam tikrą laiko tarpą.
- Abstrakcijos lygmuo (angl. abstraction) - apibrėžia kokio lygmens programavimo sąsają pateikia sprendimas.

Tam išnagrinėsime ir palyginsime, kaip šitas savybes išpildo keturi egzistuojantys srautinio apdorojimo sprendimai - "Apache Storm", "Apache Spark", "Apache Flink" ir "Heron".

## 2.3. Pristatymo semantika

"Apache Spark" ir "Apache Flink" sprendimų pristatymo semantika yra tiksliai vieną kartą (angl. exactly-once), tai reiškia, kad visi duomenys bus apdoroti tik vieną kartą. Tačiau tam, kad

užtikrinti šią semantiką sprendimas sunaudoja daug resursų, kadangi reikia užtikrinti, kad operacija bus vykdoma būtent vieną kartą kiekviename srautinio apdorojimo žingsnyje: duomenų gavime, kuris stipriai priklauso nuo duomenų šaltinio, duomenų transformacijos, kurią turi įvykdyti srautinio apdorojimo sprendimas, ir duomenų saugojime, tai turi būti užtikrinta sprendimo ir naudojamos saugyklos [ZHA17].

„Apache Storm“ pristatymo semantika yra bent vieną kartą (angl. at-least-once), tai reiškia, kad į šį sprendimą siunčiami duomenys bus visada apdoroti, tačiau kartais gali būti apdoroti kelis kartus [DAM16]. Jei uždavinys nereikalauja tiksliai vieno karto apdorojimo, tai geriau rinktis bent vieną kartą ar ne daugiau vieno karto semantikas, kadangi jos neturi papildomų apsaugų, kurios reikalingos tiksliai vieno karto apdorojimui, ir todėl veikia greičiau [ZHA17].

„Heron“ sprendimų pristatymo semantika gali būti keičiama, programuotojas kurdamas srautinio apdorojimo sistemą šiam sprendimui aprašo kokio tipo pristatymo semantikos kuriama sistema reikalauja [Ram17].

## **2.4. Uždelstumas**

Uždelstumas - laiko trukmė, kuri parodo kaip greitai sprendimas įvykdo vieną operaciją, nuo jos patekimo į eilę iki šios operacijos apdorojimo pabaigos. Pagal [LLD16] aprašytus Martin Andreoni Lopez „Apache Storm“, „Apache Spark“ ir „Apache Flink“ bandymus galima matyti, kad būtent „Apache Storm“ turi mažiausią uždelstumą. Kadangi parinkus tinkamą paralelizmo parametą šis sprendimas su užduotimi susidorojo net iki 15 kartų greičiau. Antroje vietoje liko „Apache Flink“, o po jos „Apache Spark“.

„Heron“ sprendimas yra sukurtas siekiant pagerinti „Apache Storm“ sprendimo greیتaveiką, todėl jo uždelstumas yra dar mažesnis nei „Apache Storm“ [KBF<sup>+</sup>15].

## **2.5. Pralaidumas**

Pralaidumas apibrėžia kokį kiekį procesų sprendimas gali įvykdyti per tam tikrą laiko tarpą. 2016 metais Sanket Chintapalli išmatavo „Apache Storm“, „Apache Spark“ ir „Apache Flink“ sprendimų pralaidumą ir uždelstumą bei palygino rezultatus. Kaip ir anksčiau manyta, „Apache Spark“ turėjo aukščiausią pralaidumą iš visų, kadangi jis vienintelis duomenis apdoroja mikropaketais[CDE<sup>+</sup>16]. Antroje vietoje liko „Apache Flink“, kuris yra subalansuotas pralaidumo atžvilgiu ir paskutinis liko „Apache Storm“, kuris turi žemą uždelstumą, todėl nukenčia pralaidumas. „Heron“ tuo tarpu turi aukštesnį pralaidumą ir žemesnį uždelstumą nei „Apache Storm“ [Ram15].



## 2.6. Abstrakcijos lygmuo

”Apache Storm” parašytos sistemos yra žemo abstrakcijos lygmens, tai reiškia, kad turi būti aprašyti visi srautinio apdorojimo moduliai: `setSpouts(..)`, kuriame nustatoma duomenų įeiga ir koks bus paralelizmo lygis, `setBolt(..)`, kuriame nustatomi apdorojimo moduliai, kokius duomenis gaus iš prieš tai buvusio modulio ir paralelizmo lygis. Kiekvieno modulio `execute()` metodas aprašo, kaip šis modulis turi apdoroti duomenis [tut18]. Šio sprendimo programų kūrimo laikas užtruks ilgiau negu kitiems sprendimams su aukštu abstrakcijos lygmeniu, tačiau žemas abstrakcijos leidžia rašyti daug greičiau veikiančias sistemas, kadangi programuotojas turi pilną kontrolę.

”Apache Spark” parašytos programos yra aukšto abstrakcijos lygmens. Sistemos aprašomos funkciškai, todėl kodo rašymas trunka daug trumpiau ir jį yra daug lengviau skaityti. Tačiau prarandama galimybė optimizuoti ir paralelizmo klausimas paliekamas sprendimui. Kadangi ”Apache Spark” yra ne pilnai srautinis, o mikro-paketinis (angl. *micro-batching*) sprendimas, todėl vartotojas turi apsirašyti kokio dydžio paketais bus renkami duomenys [SS15].

”Apache Flink” parašytos programos yra aukšto abstrakcijos lygmens. ”Apache Flink” sprendimas pats užsiima resursų distribucija, todėl programuotojui lieka tik parašyti veikianti kodą, o sistema pati susitvarkys su paralelizmu [Doc18]. Tačiau tai reiškia, kad su šiuo sprendimu parašytos programos nepavyks optimizuoti taip pat gerai kaip žemo abstrakcijos lygmens sprendimus.

”Heron” sprendimas turi skirtingus API, kurie naudoja skirtingus abstrakcijos lygius, kuriuos galima rinktis pagal tinkamumą sprendžiamai problemai. Darbo rašymo metu ”Heron” turi 4 skirtingus API:

- ”Heron Streamlet API” - aukšto abstrakcijos lygmens API, rašomas su ”Java” programavimo kalba. Panaši sintaksė į ”Apache Flink” rašomų sprendimų.
- ”Heron ECO API” - eksperimentinis aukšto abstrakcijos lygmens API, rašomas su ”Java” programavimo kalba. Skiriasi nuo ”Heron Streamlet API”, nes modulių apdorojimo eiliškumas apsirašo YAML formatu, kas leidžia keisti sukurtos srautinio duomenų apdorojimo programos struktūrą nekeičiant kodo.
- ”Heron Topology API for Java” - žemo abstrakcijos lygmens API, rašomas su ”Java” programavimo kalba. Rašomas identiškas kodas, kaip ir ”Apache Storm”, kadangi ”Heron” sprendimas buvo sukurtas siekiant pagerinti ”Apache Storm”.
- ”Heron Topology API for Python” - žemo abstrakcijos lygmens API, rašomas su ”Python” programavimo kalba. Su šiuo API kuriami srautinio apdorojimo sprendimai yra panašūs į ”Apache Storm”, tik su ”Python” programavimo kalbos privalumais.

## 2.7. Apibendrinimas

Pagal atliktą analizę, sudaryta 1 lentelė. Iš šių keturių sprendimų pasirinktas vienas, kuris labiausiai tinka rodiklių duomenų srautinio apdorojimo sistemų generavimui, pagal aprašytas savybes. Tinkamas sprendimas turi pasižymėti:

- Apdorojimo greičiu, svarbu, kad duomenys būtų apdorojami kai tik patenka į sistemą, todėl turi būti žemas uždelstumas.
- Tiksliai vieną kartą apdorojimu, kadangi norint gauti tikslius rodiklių duomenis turi neišsikreipti pirminių rodiklių duomenų aibė apdorojimo metu.
- Žemu abstrakcijos lygiu, nes tai leidžia generuoti iš anksto optimizuotus srautinio apdorojimo sprendimus.

1 lentelė. Srautinių duomenų apdorojimo sprendimų palyginimas

Charakteristika	"Apache Storm"	"Apache Spark"	"Apache Flink"	"Heron"
Pristatymas	Bent vieną kartą	Tiksliai vieną kartą	Tiksliai vieną kartą	Pasirenkamas
Uždelstumas	Žemas	Aukštas	Vidutinis	Žemas
Pralaidumas	Žemas	Aukštas	Vidutinis	Vidutinis
Abstrakcijos lygis	Žemas	Aukštas	Aukštas	Pasirenkamas

Pagal analizę (1 lentelė.) ir apsibrėžtus reikalavimus mūsų sprendžiamam uždaviniui tinkamiausias srautinio apdorojimo sprendimas yra "Heron".

### 3. Srautinio duomenų apdorojimo sistemų architektūra

Šiame skyriuje aprašoma pasirinktos srautinio apdorojimo programinės įrangos "Heron" srautinio apdorojimo sistemų užrašymą ir kaip turi atrodyti srautinio apdorojimo sistemos pateikiamos į ją.

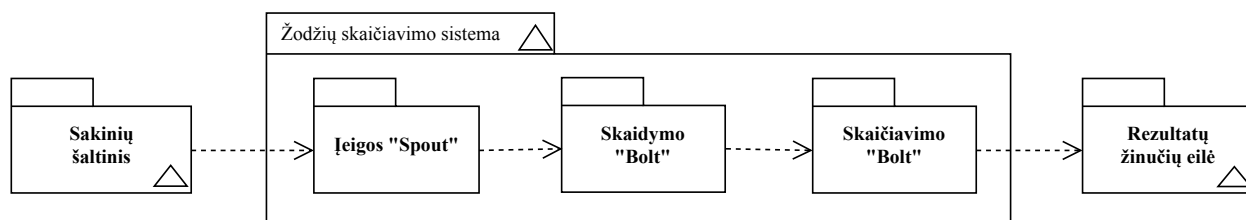
#### 3.1. Srautinio apdorojimo sistemos komponentai

"Heron" programinei įrangai parašytos srautinio apdorojimo sistemos kuriamos iš trijų dalių:

- "Spout" komponentų - duomenų įeigos komponentai. Gali būti daugiau nei vienas, jeigu yra daugiau nei vienas duomenų šaltiniai, pavyzdžiui, žinučių eilė ir duomenų bazė.
- "Bolt" komponentų - duomenų apdorojimo komponentai. Gali būti daugiau nei vienas. Gali gauti duomenis iš "Spout" komponentų ir iš kitų "Bolt" komponentų.
- "Topology" aprašo skirta aprašyti srautinio apdorojimo sistemos pavadinimą, konfigūracijas (pristatymo semantika, kontrolinių saugojimo tašku intervalas ir t.t.) ir specifikuoti, kaip tarpusavyje sujungti "Spout" ir "Bolt" komponentai ir jų paralelizmo lygis.

Konkretus pavyzdys srautinio apdorojimo sistemos gali atrodyti taip: Tarkime norime sudaryti srautinio apdorojimo sistemą, kuriai bus siunčiami sakiniai, o ji skaičiuos kiekvieno žodžio pasikartojimus per visą laiką. Vienas iš būdų įgyvendinti tokią sistemą (5. pav) būtų:

- Įeigos "Spout" komponentas gaunantis sakinius iš žinučių eilės ir perduodantis(angl. emit) sakinius toliau.
- Skaidymo "Bolt" komponentas gaunamus sakinius suskaidantis į žodžius ir perduodantis juos toliau.
- Skaičiavimo "Bolt" komponentas saugantis pasikartojančių žodžių žodyną, kur raktas - žodis, o reikšmė - kiek kartų pasikartojo. Gavęs žodi jis padidina jo skaitiklį žodyne ir perduoda žodyną į "Rezultatų" žinučių eilę.



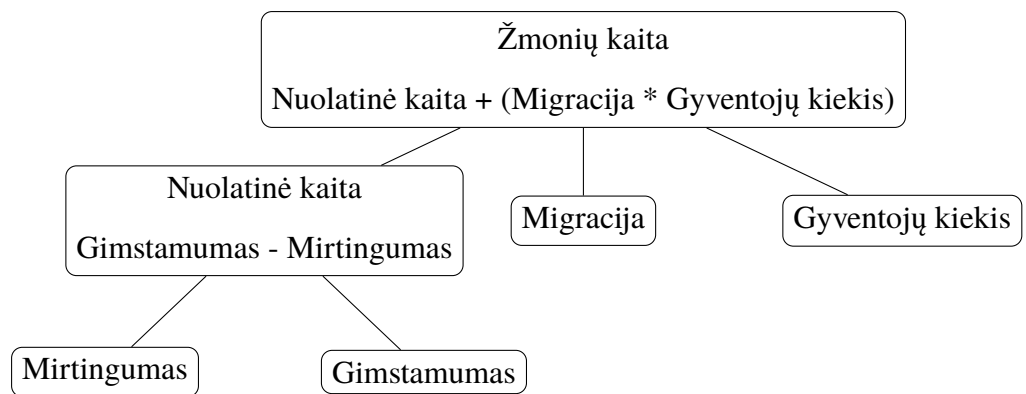
4 pav. Srautinio apdorojimo sistemos pavyzdys

### 3.2. Pagal rodiklių duomenų modelį sugeneruotos srautinio apdorojimo sistemos

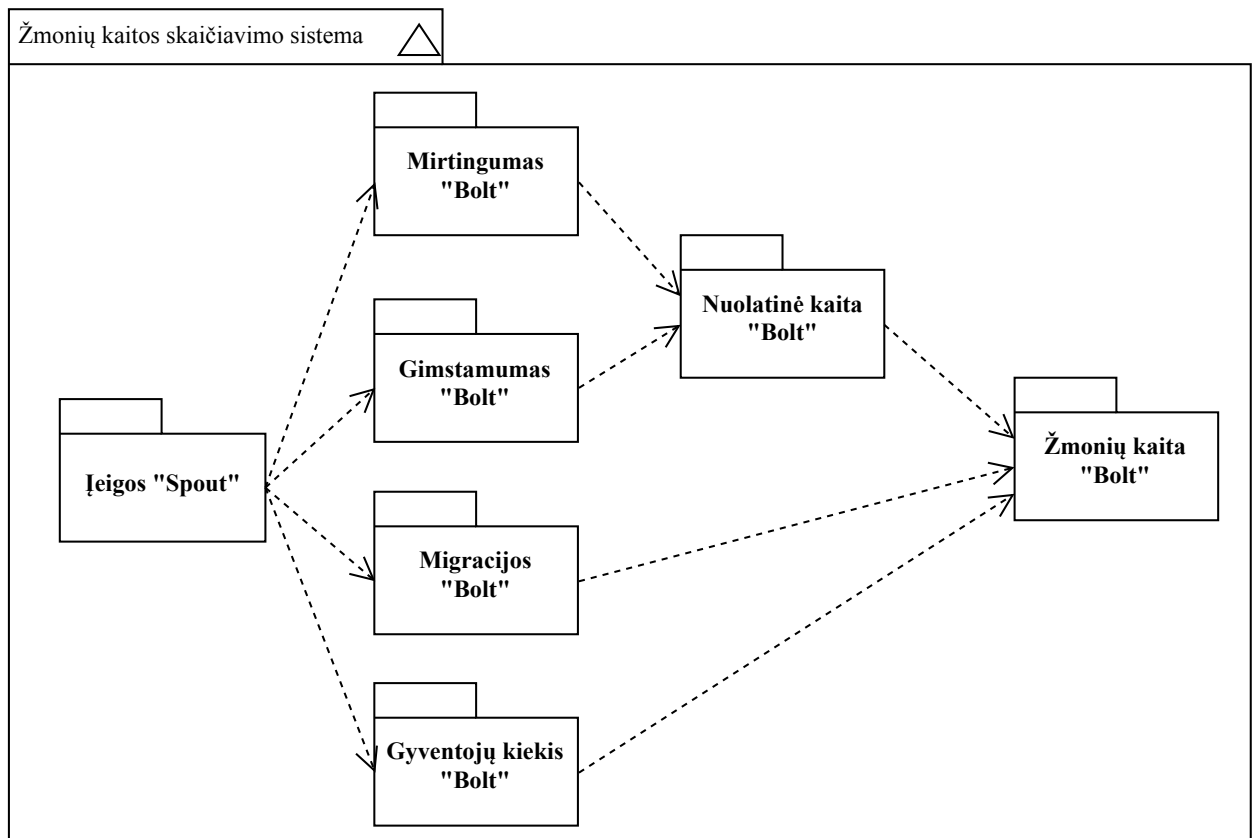
Srautinio apdorojimo sistemos pagal rodiklių duomenų modelį generuojamos taip:

1. Sugeneruojamas "Spout" tipo komponentas į kurį duomenys ateina iš žinučių eilės. Šis komponentas filtruoja duomenis pagal rodiklių duomenų modelyje apibrėžtus apribojimus. Taip pat šiame komponente prie pradinių rodiklių duomenų pridedama papildoma informacija:
  - Pirminis raktas sudarytas pagal rodiklių duomenų apibrėžimą ir pirminius rodiklių duomenis.
  - Sugeneruotas unikalus identifikatorius, kuris naudojamas išvestinių rodiklių kombinacijos veiksams.
2. Sugeneruojami "Bolt" tipo komponentai, pagal rodiklių duomenų modelio rodiklius. Pagal kiekvieną rodiklį sugeneruojamas vienas "Bolt" tipo komponentas. "Bolt" komponentai generuojami iš dviejų tipų rodiklių:
  - Paprasti rodiklių "Bolt" komponentai, kurie iš pirminių duomenų ištraukia elementą, pagal raktą aprašyta rodiklio modelyje ir perduoda jį toliau.
  - Išvestiniai rodiklių "Bolt" komponentai, kurie gauna duomenis iš vieno arba daugiau "Bolt" komponentų ir su jais atlieką transformacijos aprašytus rodiklio modelyje. Transformacijos gali būti bet kokie iš anksto aprašyti veiksmai su bet koku kiekiu duomenų. Pavyzdžiui: norint gauti išvestinį rodiklį, kuris skaičiuoja dviejų pirminių rodiklių sumą turi būti:
    - Srautinio apdorojimo sistemoje iš anksto aprašyta funkcija "suma", kuri priima du skaičius ir gražina vieną.
    - Rodiklio modelyje transformacija aprašyta - suma(rodiklis1, rodiklis2).
3. Sugeneruojamas "Topology" aprašas, kuriame aprašomos "Spout" ir "Bolt" komponentų jungimaisi tarpusavyje pagal tai kaip jie aprašyti rodiklių duomenų modelyje. Visi paprastų rodiklių "Bolt" komponentai gauna duomenis iš "Spout", o išvestinių rodiklių "Bolt" komponentai gauna duomenis iš kitų paprastų arba išvestinių "Bolt" komponentų.

Pavyzdžiui, jei turime rodiklį:



Tada sugeneruota srautinio apdorojimo sistema atrodo taip:



5 pav. Sugeneruotos srautinio apdorojimo sistemos pavyzdys

## **4. Eksperimentas ir sukurto sprendimo savybės**

### **4.1. Eksperimento tikslas**

Eksperimento tikslas - sukurti koncepcinį sprendimą, kurio pagalba patikrinti, ar sudaryta architektūra išpildo šias savybes:

- Srautinių apdorojimo sistemų generavimas pagal deklaratyvų aprašymą.
- Galimybė keisti esamas apdorojimo sistemas, kai pakeičiamas rodiklių duomenų modelis.
- Išvestinių rodiklių gavimas iš daugiau nei vieno rodiklio transformacijos.
- Išvestinių rodiklių apdorojimas pagal iš anksto apibrėžtas funkcijas.

Siekiant tai patikrinti su sukurtu sprendimu atlikti šie bandymai:

1. Sukurtas pavyzdinis rodiklių duomenų modelis, aprašytas deklaratyviai ir pateiktas į koncepcinį sprendimą.
2. Atnaujintas rodiklių duomenų modelis ( pridėtas naujas išvestinis rodiklis) ir pateiktas į koncepcinį sprendimą.

### **4.2. Eksperimento vykdymo aplinka**

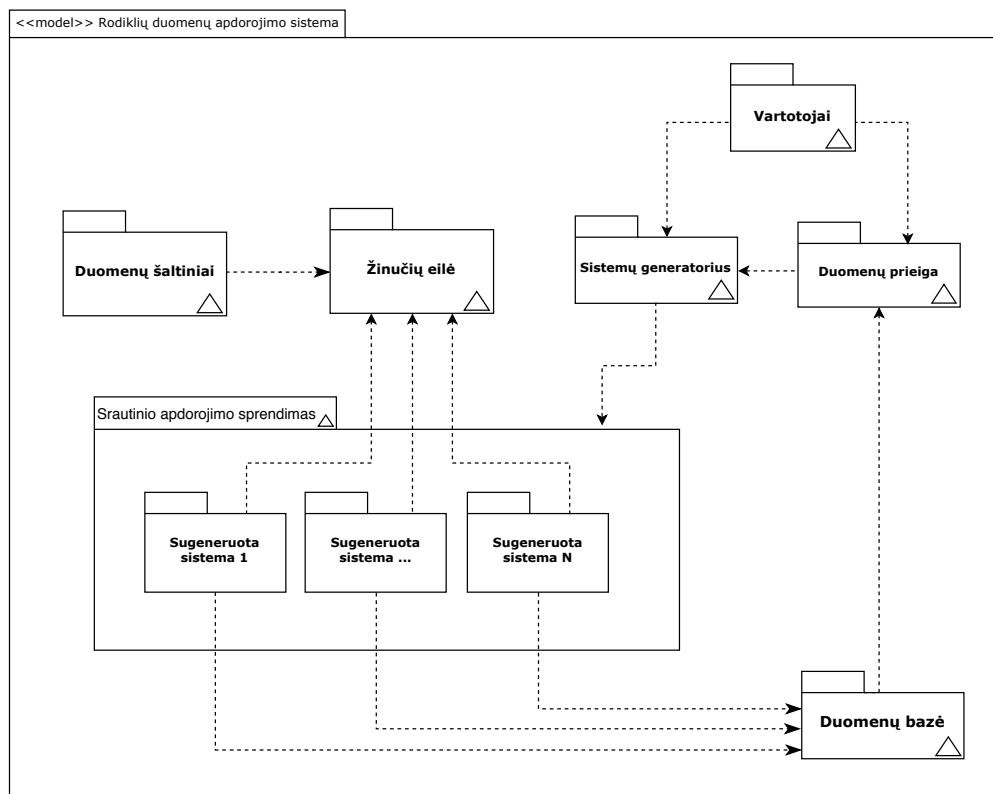
Kompiuterio, kuriame buvo vykdomas eksperimentas specifikacijos:

- Procesorius - Intel Core i7-7700k 4,5GHz
- Sisteminis diskas - 512 GB SSD
- Operatyvioji atmintis - 16 GB
- Operacinė sistema - Ubuntu 18.04

### **4.3. Koncepcinis sprendimas**

Siekiant nustatyti sudarytos architektūros savybes sukurta koncepcinis sprendimas (6 pav.). Sudarytas iš šių pagrindinių komponentų:

- Sistemų generavimo komponentas - programa generuojanti srautinio apdorojimo sistemas.
- Duomenų bazės komponentas - duomenų bazė sauganti rodiklių duomenų apdorojimo rezultatus
- Srautinio apdorojimo sprendimas - programinė įranga, kurioje veikia sugeneruotos srautinio apdorojimo sistemos.



6 pav. Konceptinio sprendimo architektūra

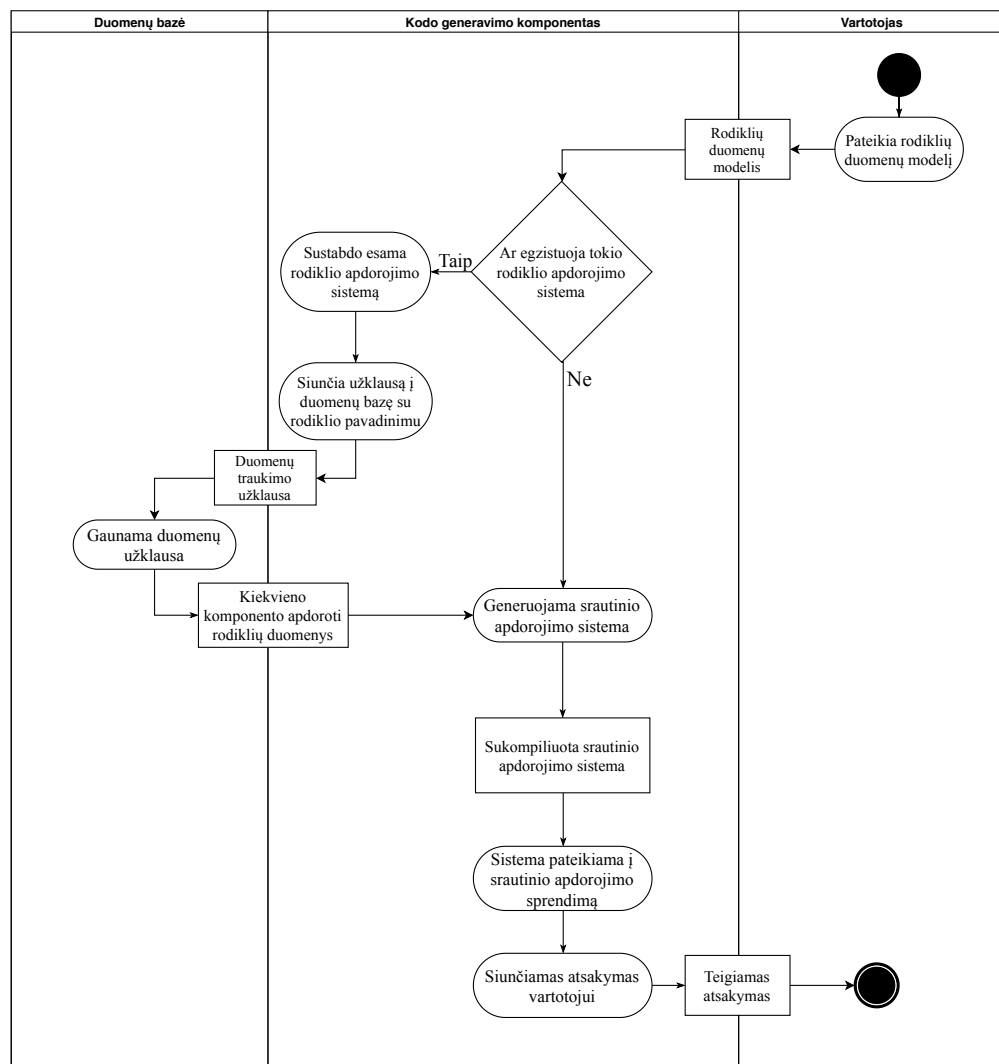
#### 4.3.1. Sistemų generavimo komponentas

Sistemų generavimo komponentas - programa sukurta su ".NET core" biblioteka, nes reikalingas taškas (angl. endpoint) deklaratyviai aprašyto rodiklio modelio pateikimui. Srautinio apdorojimo sistemų kodo generavimas yra vykdomas pagal šablonus, kurie sukurti kiekvienam srautinio apdorojimo sprendimo komponento tipui. Šablono laukai, kuriuos reikia pakeisti, užrašyti specialiais identifikuojančiais elementais. Generavimo metu šie elementai keičiami reikiamu kodu užduočiai spręsti. Šiame komponente kodo generavimas veikia taip (7 pav.):

1. Gaunamas rodiklių duomenų modelis.
2. Tikrinama ar jau egzistuoja tokia srautinio apdorojimo sistema. Tai daroma kreipiantis į duomenų bazę ir tikrinant, ar egzistuoja įrašų, kurių raktai sutampa su generuojamos sistemos pavadinimu.
3. Jei egzistuoja:
  - (a) Sustabdome veikiančią sistemą. Tai darome kviesdami stabdymo komandą per Ubuntu operacinės terminalą.
  - (b) Iš duomenų bazės traukiame jau apdorotus rodiklių duomenis ir juos perduodame į generavimo dalį.
4. Generuojamas srautinio apdorojimo sistemos įreigos "Spout" tipo komponentą. Jame įrašo-

mas žinučių eilės grupės pavadinimą, sąlygas pirminių rodiklių duomenims pagal apribojimus ir pirminiam rodikliui sugeneruojamas unikalus identifikatorius, kurio pagalba išvestinių rodiklių apdorojimo komponentas išskaičiuoja duomenis teisinga tvarka.

5. Pagal rodiklius rekursyviai generuojami duomenų apdorojimo "Bolt" tipo komponentai. Jei toks komponentas jau egzistuoja, sugeneruotas kodas yra užpildomas rezultatais gautais iš duomenų bazės.
6. Generuojamas sistemos "Topology" tipo aprašą, kuriame įrašyta komponentų jungimosi tarpusavyje logiką ir kitos sistemos konfigūracijas.
7. Visos sugeneruotas kodas talpinamas į vieną aplanką ir sukompiliuojamas "Pants" programinės įrangos pagalba.
8. Sukompiliuota sistema pateikiama į srautinio apdorojimo programą.



7 pav. Srautinio apdorojimo sistemų generavimo komponento veikimas



#### 4.3.2. Apdorotų rodiklių duomenų bazė

Duomenų bazė koncepciniam sprendime saugo apdorotus rodiklių duomenis. Kadangi reikia turėti apdorotus išvestinių ir pagrindinių rodiklių duomenis, visi srautinio apdorojimo sistemų komponentai yra prijungti prie duomenų bazės ir apdorojė duomenis talpina juos į duomenų bazę perrašydami ten jau esamus. Duomenys iš duomenų bazės naudojami dviem poreikiams:

1. Gauti apdorotus rodiklių duomenis. Siunčiant užklausą su srautinio apdorojimo sistemos pavadinimu į koncepciniam sprendime sukurta tašką gražinamas atsakymas su apdorotais rodikliais.
2. Srautinio apdorojimo sistemų generavimo komponentas rezultatų duomenų gavimui. Kadangi po rodiklių pridėjimo arba pašalinimo apdorojimas turi vykti toliau, po srautinio apdorojimo sistemos sustabdymo, kodo generavimo komponentas trauks rezultatus ir dės juos į atnaujintos srautinio apdorojimo sistemos nepakitusius komponentus.

Saugoti apdorotus duomenis buvo pasirinkta "Redis" duomenų bazė. Ji pasirinkta kadangi apdoroti rodiklių duomenis užima nedaug vietos, o labai svarbu greitai pasiekti ir įdėti duomenis. "Redis" duomenų bazių valdymo sistema tinka, nes duomenis saugomi operatyvioje atmintyje, kas leidžia duomenis ištraukti ir įdėti greičiau nei tradicinėse duomenų bazėse [Car13].

Apdoroti rodikliai saugomi unikalioje eilėje pagal srautinės apdorojimo sistemos ir rodiklio pavadinimų kombinaciją. Pavyzdžiui: renkant darbuotojų atlyginimus per metus pagal uždarbio ir premijos kombinaciją, apdoroti rodiklių duomenys būtų saugomi pagal raktus: "darbuotojų-atlyginimas:uždarbis", "darbuotojų-atlyginimas:premijos" ir "darbuotojų-atlyginimas:atlyginimas"

#### 4.3.3. Papildoma programinė įranga

Koncepcinio sprendimo veikimui taip pat reikia papildomos programinės įrangos, kurį užsiima duomenų perdavimu, srautinio apdorojimo sistemų kompiliavimu.

- Žinučių eilės pagrindinis uždavinys - perduoti duomenis į srautinio apdorojimo sistemą. Žinučių eilės naudoja publikavimo-prenumeratos modelį (angl. publish-subscribe pattern), komponentai norintys gauti duomenis užsiregistruoja žinučių eilėje, o komponentai siunčiantis tiesiog perduoda juos į žinučių eilę. Taip yra įgyvendinamas asinchroninis bendravimas, komponentai perduoda duomenis į žinučių eilę, o komponentas norintis gauti duomenis gali juos pasiimti bet kuriuo momentu iš eilės. Koncepcinio sprendimo įgyvendinimui pasirinkta Apache Kafka žinučių eilė.
- Apache Kafka reikalauja Apache Zookeeper, kuris atsakingas už Apache Kafka žinučių eilės

konfigūracijos valdymą.

- Sugeneruotos sistemos kompiliavimui naudojama "Pants" kompiliavimo sistema.

Taip pat norint eksperimentų atlikimui yra reikalinga programinė įranga:

- Pirminių rodiklių duomenų siuntimo komponentas. Šis Python kalba parašytas komponentas į žinučių eilę reguliaru intervalu siunčia duomenis sugeneruotus su <https://mockaroo.com/> įrankiu.
- Postman programinė įranga, kuri naudojama išsiųsti rodiklių duomenų modelį į koncepcinio sprendimo generavimo tašką.
- Naršyklė, kuri naudojama stebėti sugeneruotas ir pateiktas srautinio apdorojimo sistemas per "Heron" sąsają ir gauti duomenis iš apdorotų rodiklių duomenų bazės.

## 4.4. Eksperimentas

Eksperimento vykdymui sudarytas rodiklių duomenų uždavinys: turime išmanaus ofiso sensorius - drėgmės, temperatūros, šviesos. Norima surinkti valandinius dienos (nuo 8 valandos iki 20 valandos) pagal kabineto numerį rodiklius. Taip pat norima apskaičiuoti išvestinį juntamosios temperatūros rodiklį pagal formulę[ABP13]:

$$Juntamoji\ temperatūra = Temperatūra - (0.55 * (1 - \frac{Drėgmė}{1000}) * (Temperatūra - 14.5))$$

Taip pat tarkime, kad po laiko buvo nuspręsta pridėti naują bendro pojūčio išvestinį rodiklį, kuris išskaičiuojamas taip:

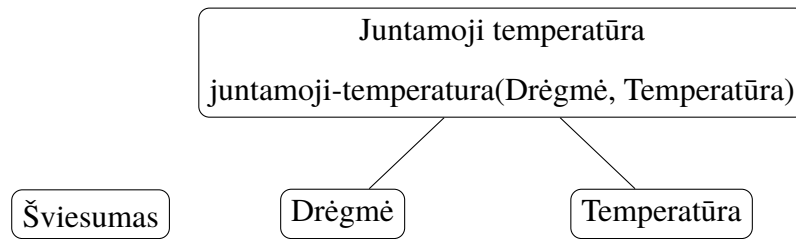
$$Bendras\ pojūtis = \frac{Šviesumas}{75} * \frac{Juntamoji\ temperatūra}{25}$$

### 4.4.1. Pradiniai duomenys

Pradinis rodiklių duomenų modelis pagal uždavinį atrodo taip:

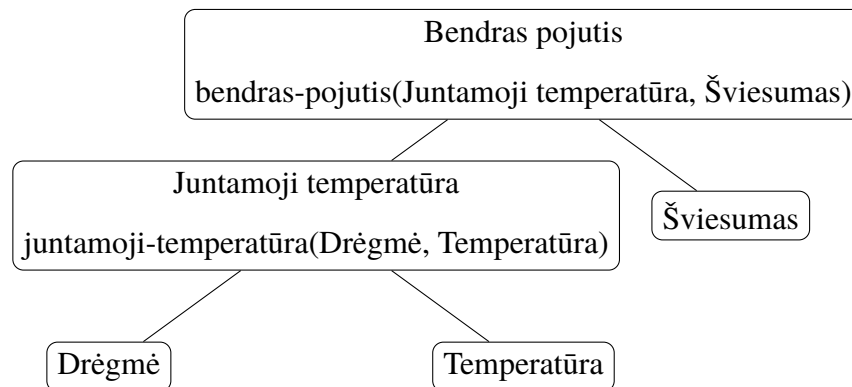
- Pirminis raktas: "Valanda", "Kabinetas"
- Pirminių rodiklių duomenų apribojimai: "Valanda" > 7, "Valanda" < 21

- Rodikliai:



Atnaujintas rodiklių duomenų modelis atrodo taip:

- Pirminis raktas: "Valanda", "Kabinetas"
- Pirminių rodiklių duomenų apribojimai: "Valanda" > 7, "Valanda" < 21
- Rodikliai:



Taip pat sugeneruoti pirminiai rodiklių duomenys naudojant <https://mockaroo.com/> įrankį (Priedas nr. 1). Šių duomenų raktai atrodo taip:

- Kabineto numeris - reikšmės iš aibės - 100, 101, 102, 103, 104
- Valanda - reikšmės nuo 0 iki 24
- Drėgmė - reikšmės nuo 0 iki 100
- Temperatūra - reikšmės nuo 10 iki 30
- Šviesumas - reikšmės nuo 0 iki 100

#### 4.4.2. Eksperimento eiga

Eksperimentas buvo vykdomas tokia eiga:

1. Pradinių rodiklių duomenų modelis užrašomas JSON formatu (Priedas nr. 2) ir naudojant Postman programą paduodamas į sprendimą.
2. Atsidarę "Heron" sąsają naršyklėje matome sugeneruotos srautinio apdorojimo sistemos diagramą (Priedas nr. 3).

3. Su pirminių rodiklių duomenų siuntimo programa pradedame siųsti duomenis į žinučių eilę.
4. Naršyklėje siusdami užklausa į sprendimo tašką gauname apdorotus rodiklių duomenis iš duomenų bazės (Priedas nr. 4).
5. Atnaujintas rodiklių duomenų modelis užrašomas JSON formatu (Priedas nr. 5) ir naudojant Postman programą paduodamas į sprendimą.
6. Atnaujinę "Heron" sąsają naršyklėje matome, jog buvo atnaujinta srautinio apdorojimo sistema (Priedas nr. 6).
7. Naršyklėje siusdami užklausa į sprendimo tašką gauname apdorotus rodiklių duomenis iš duomenų bazės, kuriuose jau yra naujas rodiklis(Priedas nr. 7).

#### **4.4.3. Eksperimento rezultatai**

Stebint eksperimentą gauti rezultatai:

- Sėkmingai sugeneruota srautinio apdorojimo sistema iš rodiklio duomenų modelio.
- Srautinio apdorojimo sistema sėkmingai apdorojo pirminiai rodiklių duomenys pagal rodiklių duomenų modelį.
- Išvestinių rodiklių "Bolt" komponentai teisingai išskaičiavo rezultatus pagal iš anksto aprašytas funkcijas naudojant duomenis iš daugiau nei vieno "Bolt" komponento.
- Sėkmingai atnaujinta srautinio apdorojimo sistema pakeitus rodiklių duomenų modelį.
- Atnaujinus sistemą nepakeisti "Bolt" tipo komponentai toliau pratęsė rodiklių apdorojimą.

#### **4.5. Sprendimo savybės**

Atliktas eksperimentas įrodo, jog šis sprendimas atitinka visas užsibrėžtas savybes.

## Rezultatai

1. Apibrėžtas rodiklių duomenų modelis ir galimi duomenų struktūros pokyčiai.
2. Atlikta srautinio apdorojimo sprendimų analizė ir pasirinktas tinkamas srautinių rodiklių duomenų apdorojimo sistemų generavimui.
3. Sudaryta srautinio apdorojimo sistemos architektūra tinkama spręsti rodiklių apdorojimo uždavinį.
4. Sukurtas koncepcinis sprendimas pagal sudaryta architektūrą.
5. Atlikti eksperimentai su sukurtu sprendimu ir apžvelgtos jo savybės.

## **Išvados**

- Išvada 1
- Išvada 2

## Literatūra

- [AAA<sup>+</sup>19] Kazunori Akiyama, Antxon Alberdi, Walter Alef, Keiichi Asada ir k.t. First m87 event horizon telescope results. iii. data processing and calibration. *The Astrophysical Journal Letters*, 875(1):L3, 2019.
- [ABP13] G Brooke Anderson, Michelle L Bell ir Roger D Peng. Methods to calculate the heat index as an exposure metric in environmental health research. *Environmental health perspectives*, 121(10):1111–1119, 2013.
- [Bea15] Jonathan Beard. A short intro to stream processing. <http://www.jonathanbeard.io/blog/2015/09/19/streaming-and-dataflow.html>, 2015-09.
- [Car13] Josiah L Carlson. *Redis in action*. Manning Publications Co., 2013.
- [CDE<sup>+</sup>16] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar ir k.t. Benchmarking streaming computation engines: storm, flink and spark streaming. *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, p. 1789–1792. IEEE, 2016.
- [DAM16] PRITHIVIRAJ DAMODARAN. “exactly-once” with a kafka-storm integration. <http://bytecontinuum.com/2016/06/exactly-kafka-storm-integration/>, 2016.
- [Doc18] Flink Documentation. Flink datastream api programming guide. [https://ci.apache.org/projects/flink/flink-docs-release-1.5/dev/datastream\\_api.html](https://ci.apache.org/projects/flink/flink-docs-release-1.5/dev/datastream_api.html), 2018.
- [Her03] Jack Herrington. *Code generation in action*. Manning Publications Co., 2003.
- [Yan17] Shusen Yang. Iot stream processing and analytics in the fog. *IEEE Communications Magazine*, 55(8):21–27, 2017.
- [Kah18] Ensar Basri Kahveci. Processing guarantees in hazelcast jet. <https://blog.hazelcast.com/processing-guarantees-hazelcast-jet/>, 2018.
- [KBF<sup>+</sup>15] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy ir Siddarth Taneja. Twitter heron: stream processing at scale. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’15, p. 239–250, Melbourne,

Victoria, Australia. ACM, 2015. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2742788. URL: <http://doi.acm.org/10.1145/2723372.2742788>.






- [KRK<sup>+</sup>18] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen ir Volker Markl. Benchmarking distributed stream processing engines. *arXiv preprint arXiv:1802.08496*, 2018.
- [LLD16] Martin Andreoni Lopez, Antonio Gonzalez Pastana Lobato ir Otto Carlos Muniz Bandeira Duarte. A performance comparison of open-source stream processing platforms. *2016 IEEE Global Communications Conference (GLOBECOM)*:1–6, 2016.
- [Nev17] Mantas Neviera. Išmaniųjų apskaitų didelių duomenų kiekių apdorojimas modernioje duomenų apdorojimo architektūroje, 2017.
- [Ram15] Karthik Ramasamy. Flying faster with twitter heron. [https://blog.twitter.com/engineering/en\\_us/a/2015/flying-faster-with-twitter-heron.html](https://blog.twitter.com/engineering/en_us/a/2015/flying-faster-with-twitter-heron.html), 2015-06.
- [Ram17] Karthik Ramasamy. Why heron? part 2. <https://streaml.io/blog/why-heron-part-2>, 2017-09.
- [SÇZ05] Michael Stonebraker, Uğur Çetintemel ir Stan Zdonik. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47, 2005.
- [SS15] Abdul Ghaffar Shoro ir Tariq Rahim Soomro. Big data analysis: apache spark perspective. *Global Journal of Computer Science and Technology*, 2015.
- [tut18] tutorialspoint.com. Apache storm - core concepts. [https://www.tutorialspoint.com/apache\\_storm/apache\\_storm\\_core\\_concepts.htm](https://www.tutorialspoint.com/apache_storm/apache_storm_core_concepts.htm), 2018.
- [ZHA17] Ji ZHANG. How to achieve exactly-once semantics in spark streaming. <http://shzhangji.com/blog/2017/07/31/how-to-achieve-exactly-once-semantics-in-spark-streaming/>, 2017.



## Priedas nr. 1

### Pirminių rodiklių duomenų generavimo įrankis

office-sensor-indicator Save Changes

Field Name	Type	Options
RoomNumber	Custom List 	100, 101, 102, 103, 104 random blank: 0 % fx ✕
Hour	Number 	min: 0 max: 23 decimals: 0 blank: 0 % fx ✕
Humidity	Number 	min: 0 max: 100 decimals: 0 blank: 0 % fx ✕
Temperature	Number 	min: 10 max: 30 decimals: 0 blank: 0 % fx ✕
Brightness	Number 	min: 0 max: 100 decimals: 0 blank: 0 % fx ✕

## Priedas nr. 2

### Rodiklių duomenų modelis JSON formatu

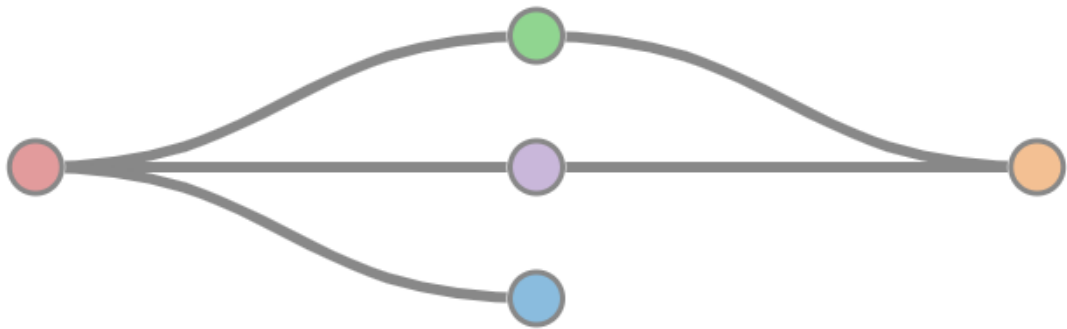
```
{
  "IndicatorId": "32c24420-afbd-44fb-b045-ef72c48cb04e",
  "Name": "office-sensors-indicator",
  "VersionId": "1-416f51b5-2b8a-4cb0-9978-f713d5990c52",
  "PrimaryKey": [
    "RoomNumber",
    "Hour"
  ],
  "Filters": [
    {
      "FieldName": "Hour",
      "Value": 7,
      "Operator": "MORE"
    },
    {
      "FieldName": "Hour",
      "Value": 21,
      "Operator": "LESS"
    }
  ],
  "Values": [
    {
      "Id": "5dbfcf52-bbce-4ec6-ad31-1cf0812b3106",
      "FieldName": "HeatIndex",
      "Formula": "%65c89c44-f5c5-48d4-ba31-01e3d4b69e42% - (0.55 * (1 - (%117af450-3086-412f-90f5-ad63e1989d04%/1000))) * (%65c89c44-f5c5-48d4-ba31-01e3d4b69e42% - 14.5))",
      "NextValues": [
        {
          "Id": "65c89c44-f5c5-48d4-ba31-01e3d4b69e42",
          "FieldName": "Temperature"
        },
        {
          "Id": "117af450-3086-412f-90f5-ad63e1989d04",
```

```
        "FieldName": "Humidity"
    }
]
},
{
    "Id": "252606d7-9b72-42d5-8923-ac8bce4f60e5",
    "FieldName": "Brightness"
}
]
}
```

### Priedas nr. 3

#### Sugeneruota srautinio apdorojimo sistema "Heron" sąsajoje

Logical Topology



## Priedas nr. 4

### Sugeneruotos srautinio apdorojimo sistemos rezultatai

```
4 {
5   "HeatIndex5dbfcf52bbce4": [↔],
332 "Humidity117af450308641": [↔],
659 "Temperature65c89c44f5c5": [↔],
986 "Brightness252606d79b72": [
987   {
988     "primaryKey": "100_12",
989     "count": 11.0,
990     "sum": 493.0
991   },
992   {
993     "primaryKey": "102_17",
994     "count": 19.0,
995     "sum": 1085.0
996   },
997   {
998     "primaryKey": "104_17",
999     "count": 12.0,
1000     "sum": 629.0
1001   },
1002   {
1003     "primaryKey": "104_15",
1004     "count": 9.0,
1005     "sum": 344.0
1006   },
1007   {
1008     "primaryKey": "103_10",
1009     "count": 13.0,
1010     "sum": 841.0
1011   },
1012   {
1013     "primaryKey": "104_9",
1014     "count": 16.0,
1015     "sum": 759.0
1016   },
1017   {
1018     "primaryKey": "102_15",
1019     "count": 9.0,
1020     "sum": 506.0
1021   },
1022   {
1023     "primaryKey": "103_8",
1024     "count": 14.0,
1025     "sum": 556.0
1026   },
```

## Priedas nr. 5

### Atnaujintas rodiklių duomenų modelis JSON formatu

```
{
  "IndicatorId": "32c24420-afbd-44fb-b045-ef72c48cb04e",
  "Name": "office-sensors-indicator",
  "VersionId": "2-a4c42274-65e9-4e9d-863b-671d51d702cf",
  "PrimaryKey": [
    "RoomNumber",
    "Hour"
  ],
  "Filters": [
    {
      "FieldName": "Hour",
      "Value": 7,
      "Operator": "MORE"
    },
    {
      "FieldName": "Hour",
      "Value": 21,
      "Operator": "LESS"
    }
  ],
  "Values": [
    {
      "Id": "5dbfcf52-bbce-4ec6-ad31-1cf0812b3106",
      "FieldName": "GeneralIndex",
      "Formula": "(%252606d7-9b72-42d5-8923-ac8bce4f60e5% / 75) * (%5dbfcf52-bbce-4ec6-ad31-1cf0812b3106% / 25)",
      "NextValues": [
        {
          "Id": "5dbfcf52-bbce-4ec6-ad31-1cf0812b3106",
          "FieldName": "HeatIndex",
          "Formula": "%65c89c44-f5c5-48d4-ba31-01e3d4b69e42% - (0.55 * (1 - (%117af450-3086-412f-90f5-ad63e1989d04%/1000)) * (%65c89c44-f5c5-48d4-ba31-01e3d4b69e42% - 14.5))",
          "NextValues": [
```

```
{
  {
    "Id": "65c89c44-f5c5-48d4-ba31-01e3d4b69e42",
    "FieldName": "Temperature"
  },
  {
    "Id": "117af450-3086-412f-90f5-ad63e1989d04",
    "FieldName": "Humidity"
  }
]
},
{
  "Id": "252606d7-9b72-42d5-8923-ac8bce4f60e5",
  "FieldName": "Brightness"
}
]
}
]
```

**Priedas nr. 6**

**Atnaujintos srautinio apdorojimo sistemos "Heron" sąsajoje**





## Priedas nr. 7

### Atnaujintos srautinio apdorojimo sistemos rezultatai

```
4 {
5   "HeatIndex5dbfcf52bbce4": [↔],
382 "Humidity117af450308641": [↔],
759 "Temperature65c89c44f5c5": [↔],
1136 "GeneralIndex5dbfcf52bbc": [
1137   {
1138     "primaryKey": "102_22",
1139     "count": 5.0,
1140     "sum": 0.0
1141   },
1142   {
1143     "primaryKey": "100_22",
1144     "count": 19.0,
1145     "sum": 3.696
1146   },
1147   {
1148     "primaryKey": "103_22",
1149     "count": 5.0,
1150     "sum": 2.361
1151   },
1152   {↔},
1157   {↔},
1162   {↔},
1167   {↔},
1172   {↔},
1177   {↔},
1182   {↔}
1187 ],
1188 "Brightness252606d79b72": [
1189   {
1190     "primaryKey": "100_12",
1191     "count": 11.0,
1192     "sum": 493.0
1193   },
1194   {
1195     "primaryKey": "102_17",
1196     "count": 19.0,
1197     "sum": 1085.0
1198   },
1199   {
1200     "primaryKey": "102_23",
1201     "count": 8.0,
1202     "sum": 359.0
1203   },
1204   {
1205     "primaryKey": "104_17",
1206     "count": 12.0,
1207     "sum": 629.0
1208   },
1209   {
```