|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Карло Пест

**ПОРЕЂЕНЈЕ РЕЛАЦИОНИХ И НЕРЕЛАЦИОНИХ БАЗА ПОДАТАКА**

ДИПЛОМСКИ РАД

- Основне академске студије -

Нови Сад, 2024.

**КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА**

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, ИБР: | |  | |
| Тип документације, **ТД**: | | Монографска документација | |
| Тип записа, **ТЗ**: | | Текстуални штампани материјал | |
| Врста рада, **ВР**: | | Дипломски рад | |
| Аутор, **АУ**: | | Карло Пест | |
| Ментор, **МН**: | | др Владимир Димитриески, ванредни професор | |
| Наслов рада, **НР**: | | Поређенје релационих и нерелационих база података | |
| Језик публикације, **ЈП**: | | Српски / ћирилица | |
| Језик извода, **ЈИ**: | | Српски | |
| Земља публиковања, **ЗП**: | | Република Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | 2024. | |
| Издавач, **ИЗ**: | | Ауторски репринт | |
| Место и адреса, **МА**: | | Нови Сад, Tрг Доситеја Обрадовића 6 | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | 9/95/0/2/38/1/0 | |
| Научна област, **НО**: | | Електротехничко и рачунарско инжењерство | |
| Научна дисциплина, **НД**: | | Примењене рачунарске науке и информатика | |
| Предметна одредница/Кључне речи, **ПО**: | | Базе података и информациони системи | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | У библиотеци Факултета техничких наука, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | У овом раду су приказане карактеристике и својства различитих типова и подтипова база података, са акцентом на два основна типа: релационе и нерелационе. Циљ рада јесте да представи свеокупну анализу и међусобно поређенје разних база података са намером да се може изабрати адекватан избор базе података за конкретне услове система. Додатан нагласак је посвећен на сарадњу више различитих база података у једном систему у виду софтверског решења, т.ј. веб апликације. | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: | др Милан Челиковић, доцент |
|  | Члан: | др Марко Вјештица, асистент | Потпис ментора |
|  | Члан, ментор: | др Владимир Димитриески, ван. професор вспрофесорпрофесор |  |

**KEY WORDS DOCUMENTATION**

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual printed material | |
| Contents code, **CC**: | | Bachelor Thesis | |
| Author, **AU**: | | Karlo Pest | |
| Mentor, **MN**: | | Vladimir Dimitrieski, PhD, Аssociate Professor | |
| Title, **TI**: | | Comparison of relational and nonrelational databases | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian | |
| Country of publication, **CP**: | | Republic of Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2024. | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | Novi Sad, Dositeja Obradovica sq. 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 9/95/0/2/38/1/0 | |
| Scientific field, **SF**: | | Electrical and computer engineering | |
| Scientific discipline, **SD**: | | Applied computer science and informatics | |
| Subject/Key words, **S**/**KW**: | | Databases and Information Systems | |
| **UC** | |  | |
| Holding data, **HD**: | | The Library of Faculty of Technical Sciences, Novi Sad | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | This thesis presents the characteristics and properties of different types and subtypes of databases, with an emphasis on two basic types: relational and non-relational. The goal of the work is to present a comprehensive analysis and mutual comparison of various databases with the intention of being able to choose an adequate choice of database for the specific conditions of the system. Additional emphasis is devoted to the cooperation of several different databases in one system in the form of a software solution, i.e. web applications. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: | Milan Čeliković, PhD, Assistant Professor |
|  | Member: | Marko Vještica, PhD, Teaching Assistant | Menthor's sign |
|  | Member, Mentor: | Vladimir Dimitrieski, PhD, Аssociate Professor |  |

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ ⚫ **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА** | Лист/Листова: |
|  |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | Основне академске студије  Основне струковне студије |
| --- | --- |
| Студијски програм: | Примењено софтверско инжењерство |
| Руководилац студијског програма: | др. Милан Рапаић, редовни професор |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | Карло Пест | Број индекса: | PR 57/2018 |
| Област: | Електротехничко и рачунарско инжењерство | | |
| Ментор: | др. Владимир Димитриески, ванредни професор | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА  ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:**

|  |
| --- |
| **ПОРЕЂЕНЈЕ РЕЛАЦИОНИХ И НЕРЕЛАЦИОНИХ БАЗА ПОДАТАКА** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| * Објаснити основне карактеристике података и база података и кратку историју базу података. * Објаснити категоризацију база података на релационе и нерелационе. * Проучити основне карактеристике релационих база података (SQL, релациони модел, АЦИД прицнипи). * Прегледати подтипове нерелационих база података и њихове карактеристике. * Осмислити и презентовати софтверско решење које користи више различитих база података за реализацију реалног софтверског система (Веб апликација са темом пикадо клуба). |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:  - Студента;  - Ментора |

Образац **Q2.НА.15-04** - Издање 1

**Садржај**

# Увод

Податак – необрађена, „сирова“ чињеница, претворена у формализован облик (број, реч, слика, итд.).

Човеков свакодневни живот је реализован на основу разних података. Било то његови лични подаци, подаци о његовој околности, пословном или приватном свету па чак и непотребни подаци, наш редован ток размишљања и догађаја је врло често везан, донекле и зависан, од података. Развојом технологије кроз време, човек више није једини који зависи од података, већ су ту зависност и задобиле машине. Kомпјутери, интернет сервери, било који електронски уређаји организују , извршавају и коригују свој рад на основу одређених података.

*„Data is the new oil“ - Clive Humbly, 2006*

Порастом зависности од података, велика пажња је посвећена на развијање софтверских система чија је главна функција рад са подацима. То укључује: складиштење, претрагу, филтрирање, обраду података и разне друге могућности које зависе од система и његових корисника. База података представља складиштен скуп података, а поменути софтверски системи се називају системи за управљање базама података.

Прве софтверске реализације база података су се појавиле 1960-тих година, када је Чарлс Бенчман дизајнирао први облик софтверске базе података, назван *Integrated Data Store*. Након тога, 1970-тих, компаније ИБМ је развила *Information Management System*. Ова два примера база података су биле навигационе, јер је проналазак података захтевао мануелну навигацију кроз податке да би се пронашао потребан податак, оне такође нису имале способност да собствено памте односе између података, такве базе података се могу називати нерелационе. У каснијим 1970-тим годинама, дошло је до прве појаве, такозване, релационе базе података. Релациона база података омогућује чување веза између података, што уводи нову димензију при складиштењу података. Подаци сада нису више везани само за појединачне вредности или елементе, већ је велики значај податка постао његова повезаност са другим подацима. Прве релационе базе података су биле ИНГРЕС, чија је идеја осмишљена и реализована на Беркли универзитету у Калифорнији, и Сyстем Р од компаније ИБМ. Сyстем Р је била прва база података која је користила познати програмски језик *Structured Query* *Language* (SQL) који је наставио да буде коришћен у изради будућих релационих база података. До садашњег времена, технологија је знатно напредовала и то је довело до повећања способности рачунара. Базе података, релационе и нерелационе, су постале више оптимизоване и појавиле су се разне верзије база података и система за управљање базама података који су биле дизајниране са посебним наменама у виду.

У наставку рада ће детаљније бити разматрани различити типови база података који постоје и активно се користе у изради софтверских система, где ће се посебно нагласити разлике између релационих и нерелационих типова база података, ради утврђивања најпогоднијег типа базе података за одређену намену и модел података.

# Релационе базе података

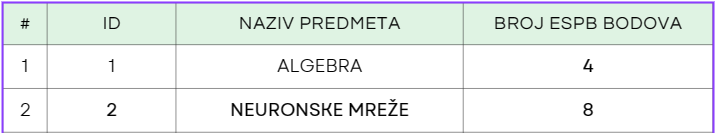
Основна идеја релационих база података је да постоји начин памћења конекција између елемената, међутим временом су релационе базе података постале стандардизоване тако да постоје додатни услови која свака релациона база података треба да испуни.

# Релациони модел података

# Један од услова јесте поштовање релационог модела података. Основна идеја јесте да су подаци организовани у виду табела. Сваки тип ентитета (податка) је складиштен у одговарајућу табелу која у себи складишти разне податке о том типу ентитета, та табела има редове и колоне, где колоне представљају својства ентитета, тј. обележја, а редови представљају појединачне елементе (ентитете). Сваки ентитет који постоји мора да има свој јединствени идентификатор, тај идентификатор се назива примарни кључ и не могу да постоје два елемента у истој табели са истим примарним кључем. Сама вредност обележја које представља примарни кључ може бити у разним облицима (број, текст, итд.).

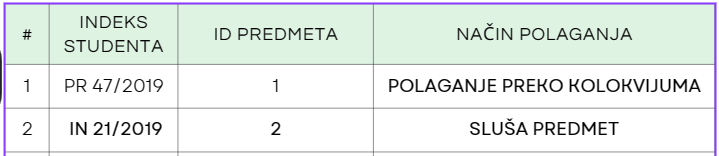
Слика 1. Пример табеле Студената

На слици 1. може се видети пример једне табеле у релационом моделу, у овој табели ће се складиштити информације о свим студентима, где ће се памтити њихово име, презиме, година студирања и њихов индекс. Пошто индекс представља нешто јединствено за сваког студента, он ће служити као примарни кључ у табели.

Слика 2. Пример табеле Предмета

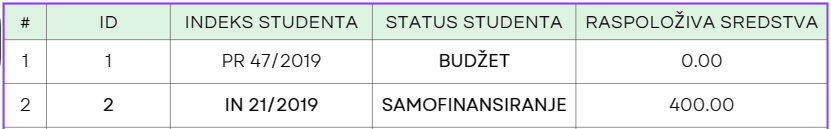
На слици 2. је приказана табела предмета, подаци о предмету су његов назив, број ЕСПБ бодова и предметов јединствени идентификатор (**ID**) који служи као примарни кључ.

Везе између ентитета у моделу се могу реализовати на 2 начина:

1. Помоћу засебних табела

Слика 3. Пример табеле Слушање\_Предмета

Табела са слике 3. функционише као **повезник** између табела Студент и Предмет и показује који студент слуша који предмет и на који начин, то омогућујемо помоћу њихових јединствених идентификатора, тј. примарних кључева. Када се примарни кључ једног типа ентитета налази у другом, тај тип обележја се назива страни кључ. У овом случају постоје два страна кључа и њихов спој заправо представља примарни кључ ове табеле.

1. Помоћу простирања примарног кључа

Слика 4. Пример табеле Студентски\_Профил

Табела Студентски\_Профил са слике 4. садржи податке о студентским профилима на веб-сервису за студенте. Потребно је знати којем студенту профил припада, пошто профил може да припада само једном студенту, студентов примарни кључ се складишти као страни кључ у табели за студентске профиле.

Начин моделовања везе између ентитета зависи од типа њихове везе и самих карактеристика модела података. Програмер, најчешће, има слободу да моделује своју базу података како њему највише одговара, у складу са моделом података и релационим моделом, функционалностима које софтверски систем треба да извршава и другим факторима битних при дизајнирању софтверског система.

# АЦИД (ACID) принципи

АЦИД принципи представљају основне особине које су временом доказане као потребне у свим релационим базама података. Битни принципи су следећи:

* Atomicity (Атомичност) – пошто се једна трансакција над базом података може садржати из више под-захтева, база података мора да гарантује да ће се трансакција извршити исправно пре него што се деси било каква измена података у бази, то јест, уколико бар један део трансакције буде неуспешан, база мора да остане у непормењеном стању.
* Consistancy (Конзистентност) – база података мора увек да буде у важећем стању након сваке трансакције, то значи да се мора осигурати да сви подаци који се уносе у базу морају да буду валидни, у складу са ограничењима табела и њихвоих обележја.
* Isolation (Изолованост) – извршавање једне трансакције над базом података мора да буде изоловано од стране других трансакција.
* Durability (Истрајност) – измене извршене над базом података, после успешне трансакције, морају трајно да буду складиштене и запамћене.

Испуњавањем АЦИД принципа побољшава се интегритет података које користимо у софтверском систему, добијају се безбеднија ажурирања података и обезбеђује се поуздан извор података.

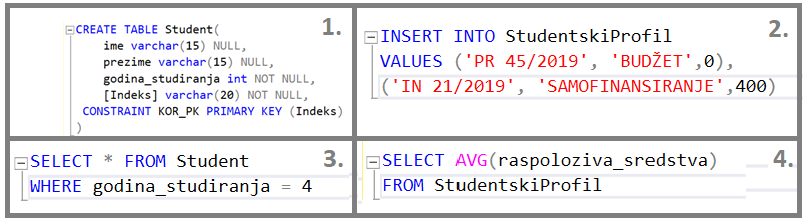
# SQL

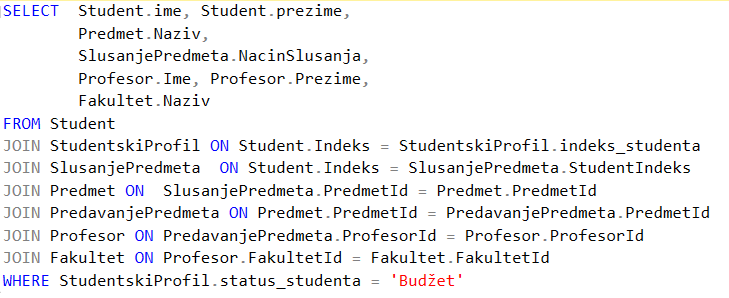
SQL (Structured Query Language) је стандардизован програмски језик за рад са релационим базама података. Настао је 1970-тих и подржан је од стране свих актуелних система за управљање релационим базама података.

*„Data is the new oil...“ - Clive Humbly, 2006*

*„...and SQL is the drill.“ Непознато*

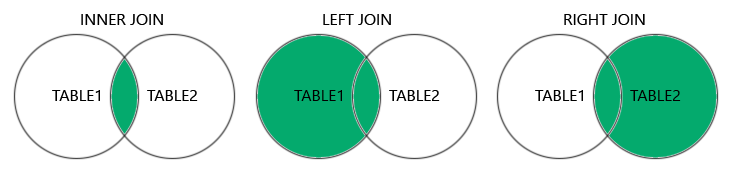
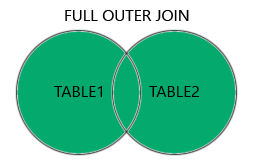
SQL омогућује разне операције над подацима које се извршавају путем SQL упита.

Слика 5. SQL упити за (1.) креирање табеле, (2.) унос података, (3.) претрагу података и (4.) обраду података.

Разумевањем синтаксе SQL програмског језика, можемо креирати комплексне упите који узимају податке из више табела на основу њихових веза, омогућавајући обраду веће количине података у једном позиву.

Слика 6. пример добављања података из свих табела у једном SQL упиту.

На слици 6. се може видети пример комплексног SQL упита који обједињује податке из више табела ради придобијања веће количине података. У овом примеру се добављају подаци о томе који студент слуша који предмет, на који начин, код ког професора и на којем факултету. Такође се додаје додатан услов да се разматрају само студенти на буџету.

На слици 6. је додатано приказан још један битан део SQL језика, а то је спајање табела, то јест JOIN.

Слика 7. различити типова спајања табела.

Постоји више врста спајања табела у зависности од постојања одговарајуће везе по којој се спајају:

* unutrašnje spajanje (**INNER JOIN**) – pridobijanje podataka koji imaju odgovarajuću vezu između dve tabele, ukoliko neki entitet iz tabele 1. nema vezu sa nekim entitetom iz tabele 2 ili obrnuto. on se neće prikazati.
* Spoljašnje spajanje (**OUTER JOIN**) – spoljašnje spajanje može da bude sa leve strane (**LEFT JOIN**) i sa desne strane (**RIGHT JOIN**). Levo spajanje uzima sve podatke iz leve tabele (od znaka ’**=’** ) i vezane podatke iz tabele sa desne strane, dok je za desno spajanje obrnuto. Razlika u odnosu na unutrašnje spajanje jeste što će se uzeti u obzir entiteti iz jedne tabele koji nemaju odgovarajuću vezu sa drugom tabelom.
* Potpuno spajanje (**FULL JOIN**) – potpuno spajanje predstavlja pridobijanje podataka iz obe tabele na osnovu neke veze bez obzira da li neki entiteti jedne tabele imaju veze sa entitetima druge tabele i obrnuto.

SQL је веома способан алат који нам не служи само као посредник у добављању и складиштењу података, већ има способност да уједно врши и комплексну обраду над подацима која уклања потребну логику обраде података коју би иначе програмер морао да имплементира, самим тиме, SQL је дефакто програмски језик за рад са релационим базама података који је активно ажуриран на сваких пар година и присутан је у мноштву софтверских система широм света.

# Приступи за генерисање модела података

Бази података се често приступа путем неке апликације, овај приступ је чест у вишеслојним софтверским системима који користе базу података. Ово уједно значи да је потребно дефинисати модел података на два места, једном у бази података и једном на месту апликације која придобија те податке. Ради уштеде времена и осигурања да се подаци дефинишу подједнако на оба места, настале су технологије базиране на генерисању кода које користе дефинисан модел на једном месту и креирају идентичан и прилагођен модел на другом месту. Постоје два приступа која користе овакве технологије.

* Code-First приступ – генерисање базе података на основу дефинисаног модела у апликацији.
* Database-First приступ – генерисање кода на основу дефинисане базе података.

Алати који се користе ради генерисања кода често могу да функционишу на оба начина, односно могу да генеришу модел података на основу постојуће базе података и обрнуто. Ти алати се зову ORM (Object Relational Mapping) алати.

Неки од тих алата су следећи:

1. Entity Framework – ORM алат најчешће коришћен у C# језику.
2. **Hibernate – ORM алат који користи Java класе.**
3. **Django ORM – ORM алат који користи програмски језик Python.**
4. **Sequelize – модел дефинисан помоћу JavaScript и TypeScript класа.**
5. **Laravel – алат користан у PHP језику.**

# Нерелационе базе података

База података се сматра нерелационом ако не поштује нека од правила која релационе базе морају да поштују. Сва правила која су осмишљена за релационе базе података служе ради увођења стандардног принципа функционисања и осигуравања да при раду са релационим базама података, програмер зна тачно која ограничења постоје и на који начин он може да користи ту базу, овај приступ нам даје сигурност, предвидљивост и основну идеју како да моделујемо нашу структуру података. Међутим, релациони модел података не мора увек да буде идеалан начин моделовања неког модела података, зато су временом постале присутније нерелационе базе података. Поента нерелационих, то јест NoSQL, база података је флексибилност, прилагодивост и слобода програмера да моделује своје податке како жели у бази података.

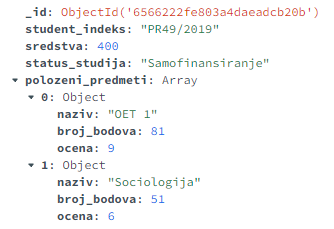
Временом, приликом дизајнирања нерелационе базе података, фокус је прешао са креирања базе података која се може користити за више случајева на креирање базе података која је прилагођена посебном начину коришћења и околностима. Зато су настали разни типови база података које функционишу на веома различите начине, неке сличне релационим, а неке са комплетно другачијим функционисањем. У наставку ће бити разматрани неки од типова нерелационих база података, заједно са њиховим карактеристикама, ограничењима и начинима коришћења.

# Document базе података

Први тип нерелационих база података који ће бити разматран је Document тип базе података. Овај тип је јендоставан пример концепта нерелационе базе података, јер не чува податке о везама између ентитета, не користи SQL програмски језик и нема предефинисан модел података којем је потребно прилагодити податке.

* + 1. **Структура података**

База података типа Document је структуирана тако да је подељена у документа, где сваки документ има више колекција података. Типови података могу бити једноставни, као што су бројеви и текст, а могу и да буду комплексни објекти са више елемената, слике у бинарном формату, низови података и разне друге комплексне структуре. Једино огранишење је да сваки податак мора да има уникатан ID да бих се могао разликовати од других и лакше пронаћи.

Фокус овог типа базе података је на подацима. Њихова структура није ограничена са предефинисаним моделом или стандардом, него је препуштено програмеру да ту структуру организује како највише одговара њему и/или систему. Овај аспекат је од великог значаја када је структура података неког софтверског система од променљиве или непредвидљиве природе, сама база података ће складиштити све податке које јој дамо (под условом да су валидног облика, то јест да се могу дефинисати JSON нотацијом), а њихова обрада и контрола је препуштена програмеру. Пример прилагођене структуре података се може видети на следећој слици.

Слика 8. Пример структуре података складиштене у Document бази података.

Уколико желимо да проширимо већ споменути тип ентитета Студентски\_Профил са опцијом чувања података о положеним предметима, у релационим базама података бих морали да направимо додатну табелу која ће чувати податке о положеним предметима, где је потрено памтити везу са студентским профилом и са предметом. Међутим, у Document базама података можемо чувати све ове потребне податке у једној табели, односно колекцији. На слици 8. се може видети како су подаци о положеним предметима чувани у виду листе, односно низа података о положеним предметима унутар студентског профила студента који их је положио.

Овим приступом су сви потребни подаци на једном месту и нема потребе за креирање додатке табеле која ће складиштити податке о положеним предметима па затим памтити везу између табела. На овакав начин можемо читаву структуру података прилагодити на разне начине и омогућити лакши приступ и преглед података. Такође нема потребе за приступ више табела у једном позиву и није потребно вођење рачуна о везама између ентитета у бази података.

Наравно, коришћењем овог метода за чување података, губимо на сигурности и интегритету података која је аутоматски успостављена приликом рада са релационим базама података. Као што је већ напоменуто раније, сви стандарди уведени за релационе базе података су ту да осигурају интегритет и конзистентност података, занемарујући ова ограничења, жртвујемо предвидивост и сигурност за флексибилност и адаптивност. Добили смо флексибилни модел, али сада морамо да водимо рачуна о подацима које убацујемо, пре него што их убацујемо, јер сама база података неће вршити никакву валидацију података за нас, осим провере валидне (JSON) структуре података. Могуће је постојање веза између ентитета у бази података, али је то такође употпуности препуштено програмеру да води рачуна.

* + 1. **Приступ бази података**

База података документ типа која ће се користити ради примера у овом раду је MongoDB, она рукује са корисним алатом званим MongoDB Compass путем којег можемо да управљамо са базом података директно, слично као алатка SQL Server Management Studio код релационих база података. Путем MongoDB Compass алата, можемо да креирамо нове документе и колекције, а поред тога је могуће вршити унос, брисање и измену постојећих података.

Овај тип база података такође има опцију претраге података путем упита. Слично као SQL код релационе базе података, користе се неки језик упита. Језик упита зависи од базе података, а код MongoDB, упити за претрагу се састоје из низа парова ОБЕЛЕЖЈЕ : ВРЕДНОСТ, где се од базе података тражи да нађе податке које имају наведено обележје са траженом вредности, уз вредност је могуће убацити додатне операторе који указују да се тражи нека вредност у зависности од задате, мања вредност, већа, различита итд. MongoDB даје могућност коришћења мноштва оператора која могу пружати велику корист приликом претрагре и обраде података. На слици 9. се може видети пример упита који се уносе у MongoDB Compass, први упит проналази студентске профиле са одређеном сумом средстава и статусом студија, док други упит претражује студентске профиле са сумом средстава већом од наведене.

Слика 9. пример упита за претрагу студентских профила.

* + 1. **Разлог употребе**

Document базе података су добар пример предности и мана нерелационих база података. Фокусом на чување података без фиксне структуре омогућује флексибилније руковање подацима, а поред тога, омогућавају веома брза читања и писања података у базу података са брзим одзивима и трансакцијама и краткотрајним упитима.

Хоризонтална скалабилност је такође велики фактор, јер су доцумент базе података веома способне да раде у дистрибуираним системима ослањајући се на репликацију. Поред тога су такође одличан алат за системе чија се структура података мења током развоја, те је ефикасно могућа измена података без великих операција над табелом. Док бих код релационе базе података измена неке карактеристике захтевало брисање и креирање табеле изнова, Доцумент базе података са својом флексибилности омогућавају ову измену без великих и/или скупих операција. У наставку следе примери софтверских система који данас користе Document базе података.

* BOSCH
* Sanoma
* SEGA
* ZF Technologies
* Forbes

Поред MongoDB постоје разне алтернативе за нерелационе базе података документ типа, а неке од њих су:

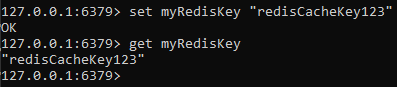
* Amazon DocumentDB – Амазонова база података која је креирана да сарађује са AWS (Amazon Web Services) који нуде велику помоћ у виду репликације и бекапа података.
* Cosmos DB – подржава различите структуре података поред JSON типа као што су Key-Value, Graph и Columbnar структуре. Лака глобална дистрибуираност. Подршка за SQL. Конфигурабилна конзистенција података.
* ArangoDB – подржава различите структуре података поред JSON типа као што су кеy-валуе и граф структуре, где је велики фокус посвећен на граф структуру. Користи AQL (Arango Query Language) који је сличан SQL-у. Трансакције над подацима различитих модела.
* Couchbase Server – Multi-Dimensional Scaling који олакшава дистрибуцију и репликацију података. Конфигурабилна конзистенција података. Користи N1QL упитни језик сличан SQL-у. Интегрисано кеширање често приступљених података. Лагано интегрисана у друге технологије и базе података.
* CouchDB – користи MapReduce упитни језик базиран на JavaScript-у. уграђене могућности репликације и дистрибуираности које олакшавају конфликте приликом репликације података. Могућа синхронизација са мобилним уређајима.
  1. **Key-Value Store (Кључ-Обележје)**
     1. **Структура података**

Један од најједноставнијих примера нерелационих база података су Key-Value Store-ови, односно Key-Value базе података. Оне фукнционишу на једноставан принцип, сви подаци су организовани у Вредност-Кључ парове (Key-Value Pair). Свака вредност је дефинисана са својим јединственим кључем и коначан изглед базе података подсећа на структуру речника или Хеш-Сет колекције.

Често коришћена Key-Value база података је Redis (Remote Dictionary Server), овај тип ће се користити приликом анализе овог типа базе података.

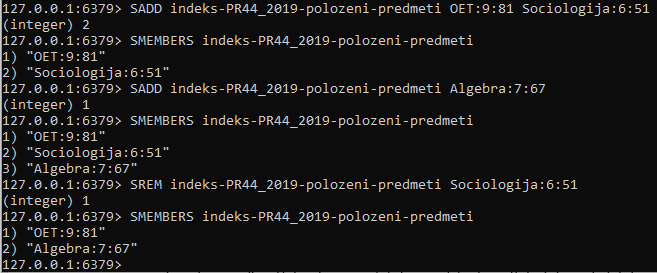
Циљ Redis базе података, и Key-Value база података уједно, је да обезбеди брз и ефикасан начин чувања података једноставне структуре. Типови података који могу да се чувају у Key-Value базама података могу бити основног типа, као број или текст, а могу бити и мало сложенији објекти са више својства, слике у бинарном облику и низови.

* + 1. **Приступ Подацима**

Унос и читање података се обавља путем кључева, где се при уносу наведе за који кључ се додељује одређена вредност, а при читању са затражи вредност са одређеним кључем. Пример уноса и читања података се може видети на следећој слици.

Слика 9. пример уноса и прегледа неке вредности путем Редис базе података.

На слици 9. се може видети једноставан пример меморисања неке вредности, у овом случају је то неки текст, али вредност може да буде разне природе у зависности од типа информације и начина на који ће се користити.

Redis такође има подршку да ради са колекцијама, односно сетовима, обезбеђујући разне операције над низовима и подацима у њима. На слици 10. се могу видети операције уноса, прегледа и брисања елемената над колекцијом положених предмета неког студентског профила. Може се приметити да су подаци о положеним предметима сачувани са произвољним обликом где су назив предмета, оцена и број бодова раздвојени са две тачке. Приликом придобијања ових података, програмер бих морао да парсира ове податке ради одвајања појединачних информација. Ограничењем комплексности структуре података у бази података, побољшана је брзина и доступност података, али изгубљена је могућност вршења комплексних обрада података од стране базе података као што је могуће у релационим базама података.

Слика 10. пример рада са колекцијом положених предмета неког студента.

* + 1. **Разлог употребе**

Key-Value базе података, са тиме што су подаци „лакши“ јер није потребно чувати њихове везе, њихово складишетње и приступање раде брзо и ефикасно. Ово омогућује брз рад са великом количином података, поготово у дистрибуираним системима где је велика учесталост приступа бази података од стране разних елемената система. Key-Value базе података су пројектоване да функционишу у оваквим системима и примуњује се у разне сврхе као што су:

* кеширање података,
* праћење сесије корисника,
* eCommerce,
* подаци ИоТ (Internet of Things) уређаја и сензора,
* географски подаци,
* подаци за видео игре итд.

Свеокупно, Key-Value базе података представљају лагано, флексибилно, скалабилно и адаптивно решење за складиштење података једниставне структуре у дистрибурианим системима, обезбеђујући брз приступ и складиштење података са високим перформансама и високом толеранцијом за високу учесталост приступа бази података.

Неки од других примера актуелних Key-Value база података су:

* Riak – висока приступност и скалабилност са толеранцијом на испаде. Децентрализована архитектура са ефикаснијом перзистенцијом података наспрам Redis базе података.
* Amazon DynamoDB – флексибилнија структура која омогућује складиштење комплекснијих података са подршком AWS технологије. Висока скалабилност. Чест избор за Wеб апликације, видео игре и ИоТ технологију.
* Google Cloud Bigtable – подаци комплексније структуре, Time-Series подаци, аналитика и машинско учење. Подршка за Гугл услуге и системе.
* RocksDB – уградива библиотека која се често користи у склопу других апликација. Ефикасно складиштење података у структуру названу LSM (Log Structured Merge) дрво.
  1. **Тime-Series базе података**

Претходно је споменуто да су нерелационе базе података дизајниране са специфичном намером коришћења у плану, овај тип база података је идеалан пример тога. Time-Series базе података рукују са подацима за које је време најзначајнији аспекат, омогућавајући поред тога брзу обраду тих података ради креирања неке статистике или разматрања промене карактеристика података.

Time-Series као појам у базама података је присутан већ неко време, где су прве базе података креиране за разматрање стања тржишта, некретнина и деоница. Међутим, у скорије доба су Time-Series базе података добиле значајну пажњу и постале су једне од најизучаванијих технологија. Разлог за повећање интереса у Time-Series базе података је повећана зависност софтверских система од врло учесталих мерења, кроз разне апарате и сензоре. Било је потребно дизајнирати неки систем који ће ефикасно склаидштити те податке и анализирати их ради потребних анализа у систему.

InfluxDB представља један актуелан пример Time-Series базе података. Дизајнирана од стране InfluxData групе, са својим вискоим перформансама и интеграцијом са Клауд (Cloud) системима постала је најкоришћенија Time-Series база података у данашњици.

* + 1. **Структура података**

У InfluxDB базама података не постоји шема података, једино правило које постоји јесте да свако унето мерење мора да има неку временску величину као података, уколико се та вредност не унесе онда ће база података да претпостави да је временска вредност једнака тренутном времену у наносекундама. Сваком мерењу се могу доделити додатни параметри који су битни за то мерење, ови параметри могу да буду таг-ови или поља (енг. Field). Таг-ови су индексирани и намењени да садрже податке по којима ће се претраживати и/или груписати подаци и они су индексирани ради боље претраге, а поља садрже неке битне вредности које ће се анализирати и нису индексирана. У наредном примеру са ваздушним сензорима таг-ови представљају идентификациони параметар сензора а међу пољима су количина угљен-диоксида, влажност ваздуха и температура ваздуха. Број таг-ова и поља није ограничен и дефинише се приликом уноса података.

InfluxDB база података групише колекције података у “корпе”, односно Бакет-е, где свака корпа може да има више различитих типова података, односно мерења. Сви подаци у корпи се гледају као мерења и имају време као обавезан параметар. Приликом креирања корпе потребно је такође одабрати временски период важења података, свако унето мерење ће бити избрисано из базе података након истека изабраног времена. Ова појава брисања података из базе након неког времена се зове полиса задржавања (енг. Retention Policy).

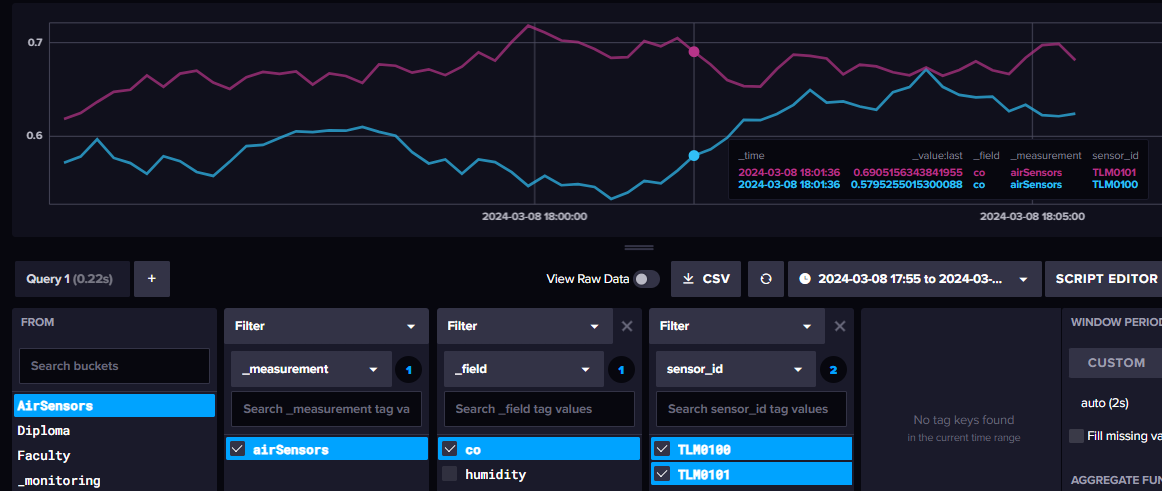
Потребно је назначити да се оваквом организацијом губи способност за комплексне упите који претражују различите врсте мерења у исто време, стога InfluxDB и остале Time-Series базе података нису погодне за руковање са структурама које имају високе кардиналитете. Такође уколико комплексност података постане превелика могуће је приметити високо оперећење процесорксе јединице рачунара и може захтевати велику количину меморијског простора.

* + 1. **Приступ подацима**

Подаци се могу унети појединачно, путем разних фајлова са подацима или преко неке УРЛ адресе, ово омогућује брз унос велике количине података на разне начине. На следећој слици се може видети потребна команда за унос података путем УРЛ адресе до Ексел листа која ће у адекватну корпу да унесе велики број (преко 460 000) мерења осам различитих сензора који региструју податке о количини угљен диоксида, температури и влажности ваздуха у одређено време.

Слика 12. Команда за унос података о мерењима параметара ваздуха.

Самој бази се може приступити на више начина, један од најчешћих начина је приступ адреси <http://localhost:8086> на којој се налази контролна табла базе података која је покренута на локалном рачунару. Путем ове контролне табле могуће је извршавање разних операција над базом података, укључујући креирање нових корпи, креирање конфигурације за унос података у базу путем многих технологија и уређаја (C#, C++, Python, Apache Cassandra, Couchbase итд.) и најкључније операције анализу података у бази података. Контролна табла омогућује анализу разних мерења на више начина: путем разних графова, кружних дијаграма, табела и сирове вредноси података. На следећој слици се може видети промена вредности количине угљен-диоксида у ваздуху на два различита сензора у временском размаку од 10 минута визуелизана уз помоћ графа.

Слика 13. Промена количине угљен-диоксида у ваздуху приказа путем графа.

* + 1. **Разлог употребе**

Time-Series подаци су постали веома присутни развојем еЦоммерце и ИоТ (Internet of Things) индустрије, самим тиме је и употреба Time-Series база података такође постала веома учестала и присутна управо због тога што је креирана са намером да ради искључиво са тим типом података. Обезбеђујући систем који адекватно рукује са великом количином података и омогућује ефикасан унос и обраду тих података, као и њихово брисање након обраде, Time-Series базе података су постале дефакто алат у софтверским системима који раде са Time-Series подацима. Неки од примера софтверских система који користе Time-Series базе података су:

* Amazon – чување података ИоТ сензора у складиштима.
* Uber – географксо праћење возила и корисника.
* LBBC – праћење сензора ваздушних турбина.
* Olympus Controls – праћење биометричких сензора на роботским рукама приликом фабричке производње.

Овај тип база података представља фантастичан пример идеалног алата за специфичну употребу, што је заправо и један од најважнијих аспеката нерелационих база података.

Примери Time-Series база података поред InfluxDB:

* QuestDB – обрада Time-Series података путем SQL упита. Подршка за извршавање упита путем више нити. Веома поуздане перформансе.
* Amazon Timestream – подршка за AMS. Serverless архитектура. Подршка за SQL.
* Apache Druid – дистрибуирана OLAP (Online Analytical Processing) база података са фокусом на колоне. Подржава угњеждене податке. Подршка за SQL упите. Лакша хотизонтална скалабилност.
  1. **Граф базе података**

Граф базе података имају кључну карактеристику, иако се сматрају нерелационим јер не користе SQL и релациони модел, њихов велики фокус је на везама између ентитета. Структура података у граф базама изгледа као граф, подељен на чворове и везе између њих. Ентитети у бази података представљају чворове (nodes), док њихове релације представљају везе (edges).

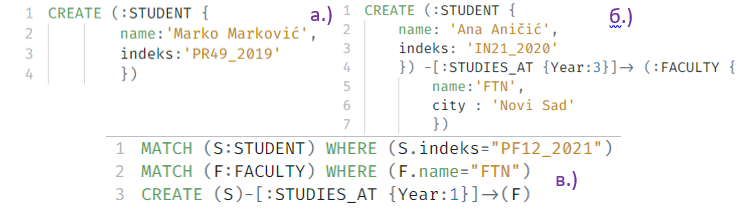
Граф базе података имају сличне аспекте као релационе базе података по питању веза између ентитета, али Граф базе података дају много већи акценат на њих и могу да буду много ефикасније у системима где структура података таква да су ентитети често повезани и то са потенцијално великим бројем других ентитета. У следећој табели ће бити разматране разлике између релационе базе података и Граф база података.

|  |  |  |
| --- | --- | --- |
| Карактеристика | Релациона база података | Граф база података |
| Структура података | Табеле са редовима и колонама | Чворови и везе са својствима |
| Односи између ентитета | Дефинисани са страним кључевима и повезним табелама | Дефинисани са везама између чворова |
| Флексибилност | Круте и ограничене структуре | Флексибилне структуре |
| Комплексне обраде података | Захтевају комплексне упите | Брзи упити и брзи одзиви |
| Препоручена намена | Систем базиран на трансакцијама са једноставнијим везама између ентитета | Системи са високо повезаним ентитетима |

*Табела 1. преглед односа неких карактеристика између релационих и граф база података.*

Граф база која ће се користити као пример је Neo4J. Пројектована од стране истоимене компаније, једна је од најчешће коришћених Граф база података која додатно и поштује АЦИД принципе. Омогућује веома брз почетак рада као и ефикасну визуелизацију података и њихових веза, уз помоћ додатних библиотека као Workplace или преко претраживача. Језик који користи Neo4Jбаза података ради извршавања упита се зове Cypher (Сајфер), он је креиран у току дизајнирања Neo4J базе података и прилагођен потребама базе података.

* + 1. **Структура података**

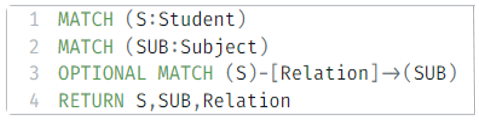
Као што је споменуто раније, подаци у Neo4J бази података су организовани у чворове и гране, где сваки елемент може да има додатна поља са информацијама и мора да има јединствено идентификационо поље које је често аутоматски генерисано (примарни кључ). Сваки чвор или грана мора да припада неком типу, овај приступ олакшава претрагу поља у будућем раду и визуелизацији, међутим ово не мора да значи да сви објекти једног типа имају фиксну структуру. Приликом креирања новог објекта неког типа, могуће је убацити поља која други објекти тог типа немају, ово омогућује креирање флексибилнијег модела без структурних ограничења. Поља могу бити разних једноставних типова као што су текст, број или боолеан вредност, а могу и да складиште низове података у виду листе. Иако је могуће да се у Neo4J бази података складишти слика, строго је препоручено да се та појава избегава јер Neo4J база података није оптимизована да ради са подацима у том формату.

Слика 11. Сајфер упити за креирање чвора типа Студент (а.), везе која повезује студента и факултет (в.) и комбинованог упита за креирања објекта Студент, Факултет и везе између њих (б.)

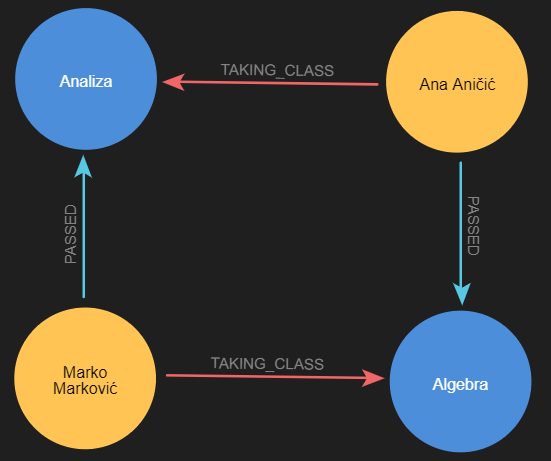
На слици 11. се могу видети примери упита за креирање чворова и веза уз помоћ Сајфер упита. Може се приметити да је креирање веза (в.) мало другачије него креирање самих чворова (а.), јер је потребно пронаћи елементе између којих се веза креира, те је потребно их пронаћи путем MATCH команде или креирати их заједно са чворовима које спаја (б.). Упитни језик Сајфер је пројектован тако да је лагано могуће креирати више елемената у једном упиту који су повезани неким везама, ово омогућује унос велике количине уланчаних елемената у једном упиту. Neo4J подржава убацивање ограничења на поља чворова и/или веза. На следећој слици се може видети упит који поставља ограничење да свако поље индекс чворова типа Студент мора да буде уникатно.

Слика 12. Сајфер упит за постављање ограничења уникатног индекса на чворове типа Студент.

* + 1. **Приступ подацима**

Граф базе података посвећују велику пажњу на визуелизацију података, омогућавајући једноставан и прегледан приказ потребних елемената и њихових веза путем Neo4J алата за приказивање података као што су Аура или Workplace. На слици 13. се може видети пример Сајфер упита којим се претражују сви студенти, предмети и њихове међусобне везе (може се приметити да је у трећој линији упита додата кључна реч OPTIONAL, ово омогућује приказ студената и предмета који немају међусобне везе), а на слици 14. се налази визуелизација придобијених података.

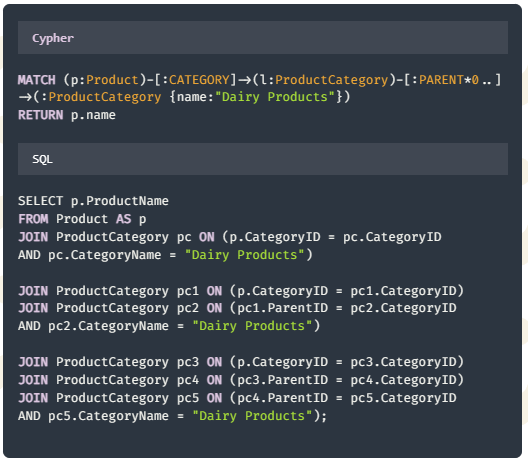
Слика 13. Сајфер упит за преглед студената, предмета и њихових веза.

Слика 14. Граф студената, предмета и њихових веза.

Постоје разни други Цyпхер упити који могу учинити приказ података још детаљнијим у складу са потребама претраге, такође је путем Neo4J алата за приказ података могуће прилагођавање приказа са додатним факторима (боје чворова и веза, позиције, текст видљив над елементима итд.). Разумевањем способности Neo4J базе података и упитног језика Сајфер могуће је изгенерисати детаљану слику елемената неког софтверског система што може служити као велика помоћ приликом разних увиђаја система.

* + 1. **Разлог употребе**

Граф базе података су саме за себе веома специфичан случај. Спадају у нерелационе базе података али читав њихов фокус је управо на ентитетима и њиховим везама, то јест релацијама. Често су поређене са релационим базама података по својој способности руковања са везама између елемената, где су доказале да занемаривањем неких од ограничења која релационе базе података постављају можемо да доведемо способност руковања са релацијама на потпуно други ниво. Сам упитни језик Сајфер може служити као идеалан пример тога, као што се може видети на слици 15. где се види колико мало кода је потребно да замени комплексан SQL упит.

Наравно, као и у другим случајевима, ово доводи до неких последица. Граф базе података не могу да раде са толико комплексним подацима као релационе базе података (слике) и подаци не могу да имају толико високу сигурност и интегритет који омогућавају сва правила релационе базе података, додатно се сматра да је много комплекнији процес упознавања са граф базама података наспрам релационе базе података. Граф базе податаке још увек спадају у новије технологије које нису имале прилику да се распростране по тржишту, те је количина доступне документације ограничена.

Слика 15. поређење комплексности Сајфер и SQL упита.

Сумирано, граф базе података омогућују флексибилно и прегледно чување података које додају потпуно нови вид интеракције са подацима. Представљају перфектно решење за софтверске системе који рукују са великим бројем међусобно повезаних елемената и веома су способне да те везе подигну на нови ниво у поређењу са другим базама података а нарочито са релационим.

Чести случајеви коришћења граф база су:

* бизниси који прате интересовања корисника ради прилагођавања препорученог садржаја,
* системи за детекцију преваре као што користе JP Morgan Chase и Intuit,
* системи за здравствено осигурање (United Health Group) који прате односе клијената и провајдера осигурања итд.

Друге честе граф базе података су:

* Amazon Neptune – Амазонов алат који подржава рад са АWС системима. Подржава РДФ модел. Подржава Gremlin и SPARQL упитне језике.
* JanusGraph – пројектована по узору на Apache Cassandr-u, Hbase i Google Bigtable. Velike tolerancija na ispade. Велике толеранција на испаде. Разна решења за индексирање са подршком за Gremlin упитни језик.
  1. **Search-Engine базе података**

Следећи пример нерелационих база података су Search Engine базе података. Ове нерелационе базе података су дизајниране да олакшају и убрзају претрагу података. Приликом уноса података, база података памти и индексира одређене карактерестике података да би могла у будуће лакше да изврши претрагу над њима.

Прва Search Engine база података је била Apache Lucene која је почела као Open Source библиотека, то значи да је њен изворни код био доступан свима да раде са њим и коригују за своје потребе. Lucene је била кључан корак за даљи развој Search Engine база података, служећи као темељ за данас често коришћене Elasticsearch и Apache Soir (Соар) базе података.

Search-Engine база података која ће служити као пример у овом раду је Elasticsearch. Као сама база података, Еластицсеарцх припада Open Source комплету званим Elastic Стацк који садржи пар додатних алата који могу да помогну Elasticsearch бази података при раду. Ти алати су Kibana, Beats i Logstash. Kibana служи као помоћни алат за рад са подацима, преко њега се могу вршити визуеализација и упити над базом података, док Beats и Logstash помажу при прикупљању података и вођења евиденције над њима.

* + 1. **Структура података**

Elasticsearch база података складишти податке у документима у JSON формату, самим тиме важи да било који податак може да се складишти уколико може да се запише у JSON нотацији. Elasticsearch прихвата податке у JSON нотацији, додељује им уникатан идентфикатор, индексира их по вредностима које садрже и складишти те JSON објекте у структуру која се зове обрнути индекс. Ова структура омогућује ефикасно праћење уникатних вредности текста свих докумената у бази података и омогућује веома брзо претраживање свих докумената по текстуалној вредности. Постоје, такође и друге структуре података које Elasticsearch користи за складиштење нумеричких и боолеан (true/false) вредности. Elasticsearch користи динамично мапирање приликом креирања индекса над пољима података који се уносе, ово омогућује да Elasticsearch сам одлучи како ће индексирати неки податак али је такође могуће дефинисати посебно мапирање података у складу са неким правилима које корисник жели да поштује. Овај приступ је адекватан уколико желимо да се неки подаци индексирају на више одређених начина или ако радимо са подацима специфичног језика па се захтева посебан начин индексирања.

* + 1. **Приступ подацима**

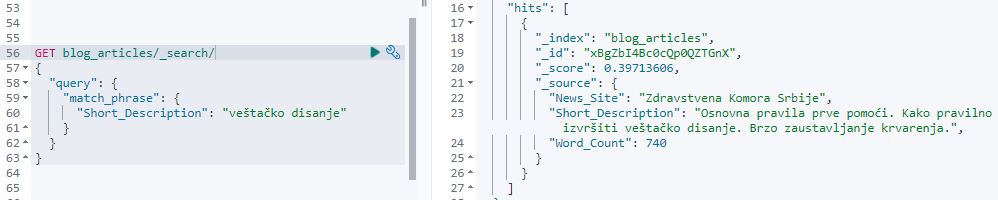
Команде, то јест упити над базом се врше у виду HTTP захтева. Пошто база података обслужује корисника на адреси <http://localhost:9200/>, нема потребе да се користи посебан упитни језик, јер слањем HTTP захтева на ту адресу са потребним подацима могуће је извршавати све операције над базом података и сва комуникација може бити одвијена путем HTTP позива и одговора. Elasticsearch база података чува податке флексибилном структуром у колекцијама које се зову документи, документи се налазе на чворовима који представљају уређаје на којим је конфигурисана Elasticsearch база података а чворови припадају једном Кластеру. Оваква организација омогућује ефикасну скалабилност и дистрибуираност.

Слика 14. HTTP позиви за креирање документа 1.), претрагу свих података 2.), унос података 3.) и 4.) и брисање података 5.)

На слици 14., уз помоћ Kibanaалата, се могу видети упити за креирање колекције података, односно документа, унос конкретног објекта, преглед података, вршење упита за претрагу података и брисања података. Као што је наведено раније, сви ови захтеви представљају HTTP позиве где позив GET често служи за придовијање података, DELETE позив за брисање а PUT и POST служе за креирање података и/или колекција. Приликом генерисања неког објекта, потребно му је доделити уникатно идентификационо поље \_ID, уколико је коришћен PUTпозив приликом уноса објекта потребно је навести вредност идентификационог поља у склопу HTTP позива, док POST позив омогућује да Еластицсеарцх том објекту, уколико није наведено у HTTP позиву, аутоматски додели уникатну вредност идентификационог поља.

Често коришћен начин претраге података је претрага пуног текста (full text search). Овај приступ се заснива на претрази свих поља докумената за вредност неког текста која је затражена у упиту. Elasticsearch је креиран да врши оволико детаљне претраге, ослањајући се на своју способност индексирања података, те процесс претраге свих поља свих података у свим документима може да се изврши у веома кратком периоду. Наравно, у склопу самог упита, можемо да специфицирамо нашу претрагу.

Слика 15. Пример претраге пуног текста са командом MATCH.

На слици 15. се може видети пример претраге новинарског артикла по пуном тексту, где је као параметар задат текст вештачко дисање а изабрана команда MATCH. Elasticsearch ће прегледати све своје документе за податке које имају вредност тог текста у свом садржају, где ће за сваког доделити и фактор важности у зависности колико сматра да је тај податак тражени. Може се уочити на слици да је Elasticsearch пронашао један артикал који има садржај о вештачком дисању, што је тражено, и један артикал који исто има тражени текст у себи, али са другачијим редоследом речи него што је заправо тражено и о потпуно другој теми. Због тога се он налази после првог артикла који се више подудара са траженим текстом. Ова појава може да буде веома проблематична у неким случајевима и можемо ју избећи коришћењем друге команде MATCH\_PHRASE која захтева да се наш текст који смо унели као параметар појављује у податку у истом редоследу како је задат и тиме обезбедити детаљнију и тачнију претрагу. Пример позива са MATCH\_PHRASE командом се налази на слици 16.

Слика 16. пример прецизније претраге пуног текста са командом MATCH\_PHRASE.

Постоје разне друге команде за још детаљније претраге података, као и додатни параметри које можемо додати у команде ради побољшавања корисниковог искуства приликом претраге података на некој интернет страници или апликацији и учинити претрагу много ефикаснијом.

* + 1. **Разлог употребе**

Search Engine базе података су врло способне да претражују податке, било то структуирани или неструктуирани, на основу приложеног текста, за то се ове базе података често користе ради аутоматског довршавања речи приликом претраге на некој веб страници. Када корисник крене да уписује неке речи у поље за претрагу, Search Engine базе података су довољно брзе да претраже потребне податке са тим текстом и пронађу потребне податке или доврше речи у пољу за претрагу са могућим потребним текстом. Због овога коришћење Search Engine база података знатно може да побољша корисниково искуство и операбилност неког система.

Такође, ове базе података су изузетно веште у сортирању података по одређеним обележјима да би преглед тих података био прилагођен неким потребама. Поред тога, одличан су избор за вршење евиденције података, односно logging. Један дистрибуирани систем, који може да користи разне софтверске јединице и платформе, може да препусти евидентирање података некој Search Engine бази података без бриге о перформансама.

Претрага велике количине података може да се врши путем уграђених алгоритама за претрагу, али могуће је и имплементирати неке прилагођене алгоритме и путем њих вршити претрагу. Ово је користан аспекат Search Engine база података јер даје кориснику још већу флексибилност приликом руковања подацима. Неке велике фирме могу да уложе велику пажњу у креирање собственог алгоритма за претрагу, рецимо путем машинског учења, које постаје све више заступљено временом, и онда путем тог алгоритма извршавати претрагу података на начин прилагођен њиховим потребама.

Разумевањем фукнционисања Elasticsearch базе податак и структуре HTTP позива, могуће је вршење разних операција над Elasticsearch бази података. Elasticsearch , заједно са другим алатима у Elastic Stack-у, представља веома способно решење за дистрибуиране системе који раде са великом количином текстуалних података, где алати као Kibana омогућавају додатне аналитичке способности и интеграцију са другим системима. Данас је Elasticsearch знатно присутан у разним софтверским системима, као што су Github, Amazon и Wikipedia, где служи као велики помоћник у претрази велике количине података за специфични тражени кључан садржаја.

* 1. **Columbnar базе података**

Последњи тип нерелационе базе података који ће се разматрати јесте Columbnar тип. Овај тип база података има специфичан начин рада са подацима где је већи фокус посвећен на колоне у табелама уместо на редове. Овим приступом олакшавамо чување и обраду велике количине података, јер чување података по колони захтева мање меморије и лакше је за груписање. Самим тиме су захтеви читања и писања веома брзи при коришћењу овог типа нерелационих база података.

Главни представник овог типа нерелационих база података је **Apache Cassandra** која је убрзо након настанка постала једна од најчешће коришћених база података у софтверским системима широм света.

* + 1. **Структура података**

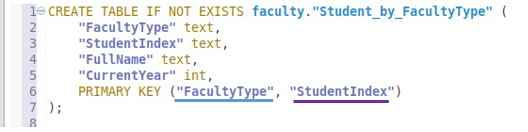
Cassandra је слична релационим базама података по томе што се подаци чувају у табелама које имају колоне и редове, ти подаци могу бити разних типова од једноставних као што су бројеви и текст до комплекснијих типова као што су листа, сет, мапа и тупле који могу да складиште више елемената у једном пољу табеле. Упитни језик који се користи за рад са Cassandra је CQL (Cassandra Query Language).

Начин на који Cassandra чува податке је другачији него код релационих база података. Иако и једна и друга чувају податке у виду табела са редовима и колонама, релационе базе података користе један систем за управљање базама података на којем су дефинисане разне базе података и унутар њих табеле, док Cassandra групише табеле у кључне просторе (Keyspace), а подаци из тих табела су распрострањени на више **чворова** у склопу једног **Кластера** (Cluster). На овај начин Cassandra групише податке у свом систему на више места у зависности од вредности кључних обележја. Подаци из тих табела такође могу бити реплицирани на више чворова ради безбедности и лакше приступности. Поступак, односно стратегија, за расподелу података по чворовима је конфигурабилна са циљем да обезбеди програмеру да одлучи како му најбоље одговара да се подаци расподеле.

Слика 17. CQL упит за креирање Keyspace-а Факултет.

На слици 17. се може видети упит за креирање једног кључног простора, односно кеyспаце-а. Приликом дефинисања једног кључног простора, потребно је дефинисати на који начин ће се водити рачуна о репликацији података. У пољу ‘class’ је потребно навести стратегију репликације и у зависности од стратегије потребно је навести додатне податке о процесу репликације, у овом примеру се користи ‘SimpleStrategy’ где је само потребно навести фактор репликације који одређује на колико чворова ће подаци бити реплицирани приликом уноса података. Постоји и ‘NetworkTopologyStrategy’ која омогућује додатну конфигурацију која је корисна ако је систем комплекснији и дистрибуиран, ова стратегија може да конфигурише распоред реплика у односу на редослед и близину чворова.

У случају са слике 17. ће се подаци из свих табела реплицирати на три чвора након уноса, самим тиме је приступ подацима лакши јер се оптерећење приступа може расподелити на три чвора подједнако, али унос података спорији јер се ти подаци морају реплицирати на више чворова.

Још једна сличност са релационим базама података, табеле у Cassandra бази података морају имати дефинисан примарни кључ који обезбеђује да не постоје два елемента у табели са истом вредности примарног кључа, али тај примарни кључ има додатну корист. Уколико табела има само једно поље дефинисано као примарни кључ онда функционише исто као код релационих база података, али уколико постоје више обележја која представљају примарни кључ онда се та обележја деле на партиционе кључеве (Partition Key) и кључеве за груписање (Clustering Key). Ови елементи утичу на расподелу података по чворовима унутар кластера. Сви елементи са истим партиционим кључем ће се налазити на истом чвору и биће расподељени (сортирани) на основу вредности кључева за груписање. Ово обезбеђује да су сви подаци који имају нешто слично налазе на истом месту, што је веома корисно уколико је потребно приступити већем броју података са истом вредности партиционог кључа.

Слика 18. CQL упит за креирање табеле Студент.

На слици 18. се може видети дефиниција табеле Студент која има поља за тип факултета на којем студент студира, његов студентски индекс, година студирања и име и презиме. Приликом дефинисања примарног кључа, пошто је поље тип факултета наведено прво оно представља партициони кључ, сва обележја наведена после првог представљају кључеве за груписање и то је у овом случају само индекс студента. У конкретном примеру, сви студенти на факултету ‘FTN’ ће се складиштити на истим чворовима и бити сортирани по вредности индекса у растућем редоследу. Редослед сортирања је такође могуће конфигурисати.

* + 1. **Приступ подацима**

Потребно је нагласити да је строго препоручено да претрага података из одређене табеле буде само по вредности партиционог кључа. Додатним командама, могућа је претрага без навођења партиционог кључа али може довести до веома спорих упита и није саветовано јер начин распоређивања података по чворовима на основу партиционог кључа знатно отежава процес претраге тих података по другим обележјима. Може се приметити да овим ограничењем жртвујемо комплексност упита за бржи приступ подацима, самим тиме структуре табела се морају креирати са овим ограничењем у виду. Овакав приступ често доводи до појаве да свака табела одговара једном конкретном упиту који ће бити упућен бази података. На следећој слици се може видети CQL упит за преглед података.

Слика 19. CQL упит за креирање табеле Студент.

* + 1. **Разлог употребе**

Сумирано, Cassandra обезбеђује један комплексан систем за управљање базама података који омогућује детаљно конфигурисање за потребе дистрибуираних система. Давајући програмеру слободу да коригује флексибилност система и сигурност и доступност његових података. Cassandrin начин форматирања табела може да служи као одличан пример система који узима већ познате функционалности и модификује их кроз разне измене које у једну руку ограничавају употребу а у другу руку знатно убрзавају придобијања и уноса података да бих свеобухватно обезбедиле идеалан систем за одређену сврху.

У наставку су примери софтверских система који данас активно користе Цассандра базу података:

* Activision – систем који води рачуна о обавештавању корисника са персонализованим понудама.
* Apple – више од 10 петабајта података на преко 75 000 чворова.
* Discord
* Ebay – преко 400 милиона читања и 100 милиона писања у базе података на дан.

Неке од других често коришћених база података Columbnar типа су:

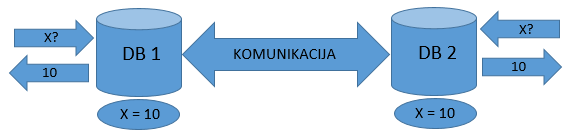
* ScyllaDB – оптимизована за веома учестале упите и брзе одговоре. Фокус на комплексан оквир чувања података назван Seaster Framework који је оптимизован за модеран хардwаре и процесоре са више језгара. Свако језгро процесоре рукује са једним чвором базе података. Базиран на C++.
* Apache Kudu – фокус над обрадом података који се често мењају. Подржава чување података оријентацијом на редове и колоне. Висока конзистенција података.
* Snowflake – Cloud инспирисана база података која подржава SQL упите. Оптимизована за OLAP системе са фокусом на анализу структуираних и слабоструктуираних података. Централизована архитектура складиштења података. Подршка за Amazon и Google сервисе који користе Cloud технологију.
* Google BigQuery – server-less OLAP архитектура са подршком за SQL упите. Користи Google-ов дистрибуирани фајл систем Colossus као и друге Google-ове услуге. Чување и обрада структуираних података.
  1. **Сумирање типова нерелационих база података**

Нерелационе базе података су креиране са принципом да уводе другачији начин руковања са подацима, ослобађањем програмера од рестрикција релационих база података дајемо слободу да се креирају разноразни системи који гледају на те податке на уникатне начине. Било то фокус на начин чувања података, флексибилнију структуру, евиденцију или приказ, па чак и посвећивање веће пажње на конфигурацију система који ће да рукује са тим подацима, нерелационе базе су фантастичан пример креирања специфичног алата за специфичну корист ради максималног добитка. У наставку је кратак опис свих типова нерелационих база података које су разматране у овом раду и њихове главне особине.

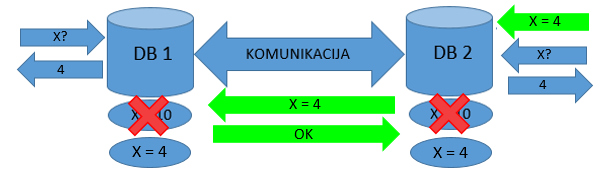
* Key-Value Store – брз приступ подацима једноставне структуре, идеалан за кеширање и чување једноставних података којима се често приступа.
* Document – поуздана база података за системе са врло учесталим трансакцијама над подацима са флексибилном структуром.
* Граф базе података – идеална база података за системе са комплекснијим моделом података где је веза између ентитета у систему од подједнаке, ако не и веће значајности у односу на саме податке ентитета.
* Time-Series – веома погодна за велику количину података чија је кључна вредност време и њихову анализу.
* Search Engine– база података која се фокусира на ефикасну и брзу претрагу података. Идеална за системе у којима се често претражује велика количина података по одређеном (најчешће текстуалном) својству.
* Columbnar – високо конфигурабилна база података са акцентом на дистрибуираност и скалабилност. Ограничена али ефикасна претрага. Веома брзи захтеви читања и писања.

1. **Класификација споменутих база података (CAP теорема)**

До сада су биле разматране разне базе података, заједно са њиховим предностима, манама и наменама. Међутим, у данашњим околностима приликом дизајнирања дистрибуираних система постоје три карактеристике које се сматрају врло битним и то су: конзистенција (Consistancy), доступност (Availability) и толеранција на партиционисање услед прекида конекције између одвојених елемената система (Partition Tolerance). ). У наставку ће бити опширније објашњено шта свака од ове три карактеристике значи.

Конзистенција представља способност система да на свим приступним тачкама система обслужује кориснике са истим и тачним подацима. Уколико један систем или база података има више чворова или елемената који рукују са истим подацима, уколико систем жели да буде конзистентан, онда корисници требају да добију исту информацију без обзира којем чвору приступају.

Слика 20. Конзистентан дистрибуиран систем

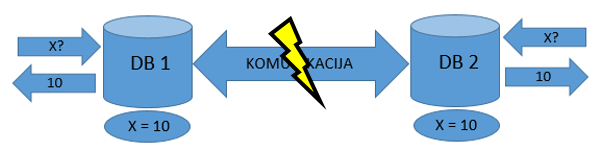
На слици 20. је приказан пример конзистентног система који представљају два чвора са својим базама података **DB 1** и **DB 2**, ова два чвора међусобно комуницирају путем неког комуникационог канала и међусобно преносе информације о уписима И променама података. Путем овог преношења података, односно репликације, омогућујемо да сви корисници добијају исте податке И самим тиме систем је конзистентан. Пример ове комуникације се налази на слици 21. где је зеленим стрелицама представљена комуникација система приликом уноса, односно промене неког податка.

Слика 21. Комуникација између чворова система приликом уноса или промене података.

Доступност представља способност дистрибуираног система да у сваком тренутнку буде доступан свим корисницима и да се сваки упит над елементима система извши успешно без обзира којем елементу система се приступа. Доступност се често постиже повећањем броја елемената којима корисници приступају И осигуравањем да сваки од тих елемената може сам да обавља сву потребну комуникацију са корисницима. Системи који дају акценат на доступност често до неке мере занемарују конзистентност система да бих осигурали брзе одговоре сваког елемента система.

Као што се може видети на слици 22. приликом промене вредности податка X, чвор **DB 2** не иницира сигнал ка чвору **DB 1** са информацијом да се вредност променила И у једном тренутнку ова два чвора ће руковати са другачијим информацијама. Ово не значи да системи који дају акценат на доступност немају никакав систем репликације података, него указује да је процес репликације мање битан те се извршава у мање учесталом периоду него код система који практикују конзистенцију над приступности.

Слика 22. Пример система са акцентом на доступности.

Толеранција над партиционисањем је веома битна карактеристика код дистрибуираних система. Баш због саме природе дистрибуираних система која захтева да су елементи система одвојени и да комуницирају, систем мора да осигура функционисање уколико дође до прекида у комуникацији између елемената система и ово осигурање се сматра Толеранцијом партиционисања. Поред омогућавања да елементи система функционишу иако немају међусобну конекцију, толеранција партиционисања се може додатно омогућити додавањем додатних веза између елемената, међутим ове методе знају да захтевају велику инвестицију и захтевају знатну припрему. Постоје и разне друге методе које софтверски системи користе за ове случајеве.

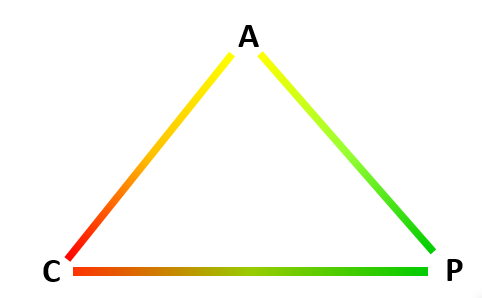
Слика 23. Пример система који функционише иако је дошло до испада приликом комуникације.

Истраживањем је дошло до закључка да су навдене три карактеристике, Конзистентнист (C), доступност (А) и толеранција партиционисања (P) најбитније карактеристике код дистрибуираних система, међутим испуњавање све три карактеристике се сматра немогућим јер не постоји начин да два елемента имају конзистентне вредности ако нису повезани, а рад са мањом количином елемената не испуњава потребну доступност система. Ово запажање је приметио Ерик Бревер који је дошао до закључка да један дистрибуирани систем може да испуни само две од три најбитније карактеристике дистрибуираних система и самим тиме засновао Бреwерову теорему која се често назива и CAP Теорема. Додатним разматрањем да није ефикасно имати дистрибуиран систем који не може да функционише приликом појаве испада комуникације, донет је закључак да је једино могуће жртвовати конзистентност или доступност.

Начин на који се може жртвовати доступност и задржати конзистентност при испаду јесте да други елемент престане са радом док се комуникација не успостави поново. Гарантујемо конзистентност тако што ће све операције бити запамћене и могуће је руковати само са актуелним подацима. Постоји и појам евентуалне конзистенције који значи да ће систем постати конзистентан убрзо након поновог успоставјања везе поја је била прекинута између елемената система.

Пример система који захтева конзистенцију над достпуности јесу трансакциони системи, на пример банкарски системи. Уколико желимо да имамо дистрибуирани систем банкомата, неопходно је обезбедити конзистентан систем који може добро да се заштити од неадекватног руковања подацима. Рецимо да се десио случај да се веза између неких банкомата и банке прекинула, у овом случају би било катастрофално омогућити да банкомати врше било какву исплату новца, те је конзистентност битнија него доступност.

Неки системи и базе података се фокусирају на доступност наспрам конзистентности. Ово указује да уколико се деси прекид у вези између елемената система, елементи ће наставити да раде и обслужују кориснике, а након успостављања прекинуте везе ће синхронизовати податке који су се променили. Јасно је да овим поступком може доћи до неслагања података на два елемента у систему, међутим у неким системима ово не мора да представља велики проблем. Пример овог случаја јесте интернет странице YouTube, приликом прегледа информација о броју прегледа неког видео снимка може доћи до појаве да једном кориснику буде другачији број прегледа него другом. Кад су у питању велики бројеви где је баш тачна вредност од мање користи него оквирна вредност, корисније је омогућити достпуност наспрам конзистентности.

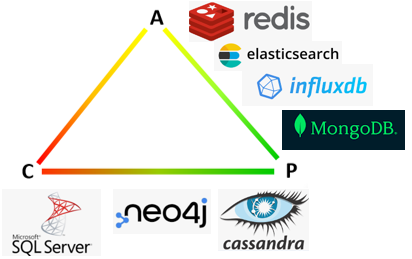
Укратко, што је комплекснија структура података која може да садржи осетљиве и битне податке, то је већа потреба да систем буде конзистентан. Док код случаја када се ради са великом количином података где нема потребе за великим детаљисањем приликом рада са тим подацима, даје се много већи акценат на доступност система. Одлука да ли ће се више посвећивати пажња на конзистентност или доступност у потпуности зависи од услова у којима систем ради и структуре података.

Слика 24. CAP троугао.

Често се поставља питање да ли је могуће направити дистрибуиран систем који је и конзистентан и доступан. Иако је могуће делимично избећи појаву партиционисања путем омогућавања већег броја веза између елемената система, појава партиционисања је веома непредвидљива и без припреме за тај случај могућа је појава великих проблема у дистрибуираним системима. Наравно, конзистентност и доступност је могућа у не дистрибуираним системима без великих проблема.

CAP теорема представља корисну класификацију у раним фазама пројектовања система, након установљавања битних карактеристика система, могуће је уз помоћ CAP теореме одабрати идеалну базу података за пројектовани софтверски систем.

У наставку ће бити сагледане карактеристике свих база података које су биле разматране у раду из угла CAP Теореме, где ће се за сваку базу података навести да ли припада CP (Конзистентни са толеранцијом на партиционисање) или AP (Доступни са толеранцијом на партиционисање)

* SQL сервери – **CP** – релационе базе података посвећују велики ниво пажње да су сви подаци који се уносе у систем исправни и у складу са ограничењима, додавањем на то и поштовање АЦИД принципа, може се приметити велики фокус на конзистенцију.
* MongoDB – **AP** – МонгоДБ омогућује ефективне репликације података путем своје дистрибуиране архитектуре и Схардинг процеса који гарантује евентуалну конзистенцију података, међутим МонгоДБ је пројекатован да подржи велики проток података те је много већа пажња посвећена на доступност приликом рада са МонгоДБ базом података.
* Redis – **AP** – Редис је база података која посвећује велику пажњу на ефикасно складиштење велике количине једноставних података, те је круцијално омогућити брзо прихватање и руковање са тим подацима.
* InfluxDB – **AP** – честа појава код рада са Тиме-Сериес података јесте да је њихова количина веома велика, као и брзина којом се ти подаци уносе у систем. Због ове природе података, Тиме-Сериес базе података морају да омогућавају велику доступност. ИнфлуxДБ постиже евентуалну конзистенцију путем асинхроне реплиакције података кроз елементе у систему.
* Neo4J – **CP** – Нео4Ј подржава АЦИД принципе, самим тиме конзистенција мора бити загарантована. Иако је Нео4Ј подешен да буде толерантан на партиционисање у склопу једног цлустер-а, постоје ограничења приликом рада у веома дистрибуираним системима.
* Elasticsearch – **AP** – Еластицсеарцх је специјализован за рад са великом количином података, што укључује обраду и прихватање тих података. Еластицсеарцх користи методу схардинг-а да реплицира податке на друге чворове у систему што омогућује високу доступност и евентуалну конзистенцију.
* Cassandra – **CP** – Цассандра је специфичан случај из погледа ЦАП теореме због своје конфигурабилне природе. По предвиђеним конфигурацијама, Цассандра даје већи акценат на конзистентност података, поготово са подешавањем стриктне структуре података. Међутим могуће је конфигурање система да се даје већи акценат на доступност него на конзистенцију на разне начине, као што су смањење репликационог фактора неке табеле. Тако да Цассандра може да адаптира своје карактеристике у складу са потребама система и захтеване конзистенције или доступности.

Слика 25. CAP троугао са додатим базама података које су разматране у раду.