УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

Срђа Љ. Бјеладиновић

# РАЗВОЈ МЕТОДОЛОШКОГ ПРИСТУПА ЗА ПРОЈЕКТОВАЊЕ И КОРИШЋЕЊЕ ХИБРИДНЕ *SQL*/*NOSQL* БАЗЕ ПОДАТАКА

Докторска дисертација

Београд, 2018

UNIVERSITY OF BELGRADE
FACULTY OF ORGANIZATIONAL SCIENCES

Srđa Lj. Bjeladinović

# DEVELOPMENT OF A METHODOLOGICAL APPROACH FOR HYBRID SQL/NOSQL DATABASE DESIGN AND USAGE

Doctoral Dissertation

Belgrade, 2018

МЕНТОР:

_____

др Зоран Марјановић, редовни професор

Факултет организационих наука

Универзитет у Београду

ЧЛАНОВИ КОМИСИЈЕ:

_____

др Слађан Бабарогић, ванредни професор

Факултет организационих наука,

Универзитет у Београду

_____

др Иван Луковић, редовни професор

Факултет техничких наука,

Универзитет у Новом Саду

Датум одбране рада: _____

**Наслов докторске дисертације:**
Развој методолошког приступа за пројектовање и коришћење хибридне *SQL*/*NoSQL* базе података.

**Сажетак:**
Ова дисертација је настала из потребе теоретског, развојног и практичног решавања проблема пројектовања и коришћења хибридне *SQL*/*NoSQL* базе података. Различити типови база података, како је описано у наставку, садрже специфичности које је потребно узети у обзир приликом развоја методолошког приступа за реализацију процеса пројектовања и коришћења. Дисертација је написана као резултат спровођења процеса научног истраживања, прегледа области истраживања, анализе постојећих решења и приступа, развоја нових приступа пројектовања и коришћења (са свим њиховим саставним деловима), примене новоразвијених приступа на примере из праксе, тестирања изабраних аспеката перформанси (по одређеним критеријумима) и компаративне анализе постигнутих резултата хибридне базе података пројектоване применом новог приступа и „традиционално“ пројектоване базе података. Циљ ове докторске дисертације је био развој новог приступа за пројектовање хибридне *SQL/NoSQL* базе података и интеграцију и униформно коришћење њених компоненти (*SQL* и *NoSQL* база података). У докторској дисертацији су анализирани погодни критеријуми за доношење одлуке о оправданости преласка са постојеће *SQL* базе података на хибридну базу података. Укључивање аспеката пројектовања нових и редизајна постојећих база података у новоразвијени приступ, проширио је опсег његове могуће примене. За потребе приступа развијена је и нова архитектура, која је омогућила да се над целокупном хибридном базом података (и свим базама података које чине њене саставне компоненте) управља мета подацима, правилима интегритета, правилима мапирања и извршавањем наредби као над јединственом логичком базом податка.

**Кључне речи:** хибридна *SQL*/*NoSQL* база података; методолошки приступ; пројектовање базе података; јединствена логичка база података; структурираност података; интеграција; униформно коришћење
**Научна област**: Техничке науке
**Ужа научна област:** Информациони системи
**УДК:**

**Abstract:**
This dissertation has originated out of the need for theoretical, developmental and practical solving of the issue of the design and integral and uniform usage of hybrid SQL/NoSQL databases. As described in the course of this dissertation, various types of databases contain specificities that have to be taken into account when developing a methodological approach for the realization of aforementioned processes of database design and usage. The dissertation is written based on many activities. It was driven by conducting scientific research, review of research areas, analysis of existing solutions and approaches. Besides that, dissertation included development of new approach for design and usage (with all their integral parts), application of newly developed approach to examples from practice, testing of chosen aspects of performances (based on certain criteria) and comparative analysis of achieved results of hybrid database developed by applying a new approach and 'traditionally' designed database. The aim of this doctoral dissertation was the development of a new approach for the design of hybrid SQL/NoSQL database and integration and uniform usage of its components (SQL and NoSQL databases). A criterion for making a decision on the justification of transition from existing SQL database to hybrid database has been defined in this dissertation. Including aspect of new database design, as well as including aspect of existing database redesign expanded the scope of newly developed approach. For the approach needs, a new architecture has also been developed, which enabled managing metadata, integrity rules, mapping rules and statement execution over the entire hybrid database (including all its component databases) as over a unique logical database.

**Key words:** hybrid SQL/NoSQL database; methodological approach; database design; unique logical database; data structuredness; integration; uniform usage
**Scientific field:** Technical sciences
**Scientific subfield:** Information systems

Мојој мајци Буби, са љубављу и захвалношћу.

# Садржај

# Садржај слика

# Садржај табела

# 1. Увод

Данас се у употреби могу наћи базе података различитих типова (SolidIT, 2018) које поседују специфичне карактеристике. У савременом пословању неретко се дешава да је потребно помирити супротстављене захтеве различитих типова база података, нпр. конкурентно користити податке различитог степена структурираности, или истовремено гарантовати интегритет, али и максимизовати доступност различитих података унутар једног пословног система. Наведено је проузроковало потребу настанка јединствених логичких база података које би као своје компоненте комбиновале базе података различитог типа и у зависности од потребе задовољавале карактеристична својства сваког од типова. То је довело до настанка тзв. хибридних база података. Немогућност адекватне примене постојећих приступа и техника пројектовања на проблем пројектовања хибридне базе података (која обухвата више база података различитих типова) проузроковала је потребу развоја приступа који би у процес пројектовања укључио специфичности различитих типова база података. Поред пројектовања, од значаја је и решавање начина коришћења хибридне базе података, јер тренутно не постоји универзални начин интеграције нити коришћења база података различитог типа, које би чиниле компоненте хибрида. То је отворило простор за дефинисање теоретског оквира, али и практичног решења споменутих проблема.

Узимајући наведено у обзир, може се рећи да је ова дисертација настала из потребе теоретског, развојног и практичног решавања, у домену информационих система присутног, проблема пројектовања и коришћења хибридне базе података. Самим тим, развој методолошког приступа за пројектовање и коришћење хибридне *SQL*/*NoSQL* базе података представља предмет истраживања ове докторске дисертације. Процесу писања дисертације претходили су различити процеси и активности. Дисертација је настала као резултат спровођења процеса научног истраживања, прегледа области истраживања, анализе постојећих решења и приступа и развоја новог приступа пројектовања и коришћења хибридне *SQL*/*NoSQL* базе података. Поред тога, примена новоразвијеног приступа на

примере из праксе, тестирања изабраних аспеката перформанси (по одређеним критеријумима) и компаративна анализа постигнутих резултата хибридне базе података пројектоване применом новог приступа и „традиционално“ пројектоване базе података такође су претходили изради дисертације и заузели су значајно место у истој.

## 1.1. Основ истраживања

Данас, доминантне базе података у употреби су релационе базе података (SolidIT, 2018). Оне су засноване на релационом моделу, а временом су проширене концептима објектно-оријентисане парадигме у објектно-релационе базе података. Због стандардизованог *SQL* (*Structured Query Language*) језика који подржавају, називају се и *SQL* базе података. Главне карактеристике *SQL* база података, поред чињенице да су засноване на релационом моделу и да подржавају *SQL* језик, представљају испуњеност *ACID (Atomicity Consistency Isolation Durability)* особина трансакција и погодност за складиштење и управљање подацима прецизно дефинисане и ретко промењиве структуре, тј. подацима који поседују висок степен структурираности. *SQL* базе података су посебно погодне у раду са структурираним подацима, када се као императив постављају конзистентност података и извршавање комплексних упита, са великим бројем операција спајања.

Значајна експанзија Интернета и сервиса које је он понудио, а посебно улазак у другу фазу његовог животног циклуса, у тзв. *Web* 2.0 фазу (Abiteboul, et al., 2011), допринели су приметном повећању броја корисника друштвених мрежа (*Facebook*, *Twitter*, *LinkedIn* итд.) и различитих портала (*Google*, *Amazon* итд.). Иако је и раније био очигледан тренд експоненцијалног раста количине података у употреби, компаније, које су у свом пословању почеле да користе споменуте сервисе, допринеле су додатном наглашавању потреба брзог манипулисања великом количином података, складиштења великог броја података и употребе података различитог степена структурираности (Bjeladinović, & Marjanović, 2016). Наведене околности нагласиле су потребу употребе база података са флексибилним и лако промењивим шемама, база података које приоритет дају доступности на рачун конзистентности података и које самим тим омогућавају рад са

неструктурираним подацима или подацима ниског степена структурираности (Inmon, Strauss, & Neushloss, 2008). То је допринело настанку новог приступа почетком века: *NoSQL* база података. Данас, термин *NoSQL* базе података обухвата четири типа база података (кључ-вредност базе података, базе података засноване на документима, базе података засноване на фамилији колона и граф базе података), који се могу називати и подтиповима *NoSQL*-а. Заједничко им је одсуство стриктно дефинисане шеме, одсуство униформног језика (иако неки представници ових база података подржавају *SQL* језик, не постоји стандардизовани и униформни језик на нивоу *NoSQL* база података), подршка *BASE (Basically Available, Soft-state, Eventual consistency)* особина (за разлику од *ACID* особина *SQL* база података) и подршка хоризонталне скалабилности (Lake, & Crowther, 2013; Sullivan, 2015).

Популарност и специфичне карактеристике *SQL* и *NoSQL* база података допринели су оправданости њиховог паралелног коришћења у савременом пословању, уз задржавање веће ефикасности примене, свака за своју намену (Avison, & Fitzgerald, 2003; Badia, & Lemire, 2011; da Silva, 2011; Nance, et al., 2013; Nyati, Pawar, & Ingle, 2014). Ради што ефикаснијег испуњавања свих захтева који се данас могу поставити пред савремену базу података, није увек оптимално донети одлуку о избору једног од наведених типова база података. Уместо тога, потребно је омогућити паралелно коришћења *SQL* и *NoSQL* база и њихово повезивање у интегралну логичку целину, а у циљу обухватања најбољих функционалности из оба света. Могуће решење овако сучељених захтева је **пројектовање и коришћење хибридне *SQL*/*NoSQL* базе података**. Хибридна *SQL*/*NoSQL* база података се састоји од *SQL* базе података и једне или више *NoSQL* базе података различитог типа, које третира као своје саставне компоненте и које обједињује у јединствену логичку базу података.

Методолошки приступи и технике за пројектовање *SQL* база података се развијају деценијама уназад (Lazarević, et al., 2006), међутим то није случај са *NoSQL* базама података. *NoSQL* базе података су у већој употреби у последњих десетак година, а приступи њиховог пројектовања су тек у зачетку. Приступи пројектовања *SQL* и *NoSQL* база података анализирани су и у овој дисертацији.

Ипак, постојећи приступи пројектовања *SQL* и *NoSQL* база података садрже специфичности у фазама, техникама и методама пројектовања, који не узимају у обзир специфичности другог типа базе података, осим оног за који су намењени, чиме се отежава, чак и онемогућава, директна примена тих приступа на оба типа базе података (*SQL* и једног или више подтипова *NoSQL*). Самим тим, постојећи приступи могу у одређеном обиму послужити као основа за пројектовање одређених делова њених компоненти, али не могу бити директно примењени на процес пројектовања целокупне хибридне *SQL*/*NoSQL* базе података. Специфичности у пројектовању и интеграцији компоненти хибридне *SQL*/*NoSQL* базе података у јединствену логичку базу података захтевају свеобухватни методолошки приступ пројектовања такве базе података. У овој дисертацији представљен је нови методолошки приступ пројектовања хибридне *SQL*/*NoSQL* базе података.

За приступ подацима и коришћење података из *SQL* система, као што је већ споменуто, користи се *SQL* језик. За разлику од *SQL* база података, *NoSQL* базе података не поседују стандардизовани упитни језик. Различити упитни језици се користе за различите типове *NoSQL* база, а и међу представницима истог типа не постоји консензус произвођача о стандардизацији језика за рад са базом података (Atzeni, Bugiotti, & Rossi, 2014; Badia, & Lemire, 2011; Bondiombouy, 2014). Наведено је отворило простор за развој приступа који би омогућио коришћење хибридне *SQL/NoSQL* базе података у циљу интеграције и униформног коришћења *SQL* и *NoSQL* база података. Новоразвијени приступ за униформно коришћење хибридне *SQL*/*NoSQL* базе података, без обзира да ли се приступа *SQL* или *NoSQL* компоненти, такође је представљен у овој докторској дисертацији.

## 1.2. Структура дисертације

У другом поглављу ове дисертације објашњени су основни концепти и типови база података. С обзиром на то да предложени приступ обухвата пројектовање хибридне *SQL*/*NoSQL* базе података и њено коришћење, сваки од наведених аспеката је описан у засебном поглављу, при чему сваком од њих претходи поглавље са прегледом референтних радова у вези са пројектовањем и коришћењем,

респективно. Иако новоразвијени методолошки приступ обухвата пројектовање и коришћење, због могућности њихове одвојене примене, сваки од та два саставна дела приступа ће бити описан у засебном поглављу.

Опис доступних техника и методолошких приступа за пројектовање *SQL* и *NoSQL* база података дати су у трећем поглављу.

У четвртом поглављу дат је приказ новоразвијеног методолошког приступа за пројектовање хибридне *SQL*/*NoSQL* базе података, чије компоненте чине *SQL* и *NoSQL* базе података. Овај приступ узима у обзир специфичности пројектовања *SQL* и *NoSQL* компоненти хибридне базе података, уводи критеријум за утврђивање оправданости развоја хибридне базе података, у ситуацији када се пројектује нова база података и у ситуацији када се спроводи редизајн постојеће базе података, што је описано у истом, четвртом, поглављу.

У петом поглављу су приказани постојећи приступи у интеграцији *SQL* и *NoSQL* база података, који представљају увертиру за приказ новог приступа коришћења хибридне *SQL*/*NoSQL* базе података.

У поглављу шест представљен је новоразвијени приступ за интеграцију и униформно коришћење *SQL* и *NoSQL* базе података заснован на употреби хибридне *SQL*/*NoSQL* базе података. У овом поглављу описане су новоразвијене компоненте референтне архитектуре, као и динамика њихове примене.

У седмом поглављу, нови приступ је примењен на реалном примеру. Ово поглавље садржи анализу оправданости пројектовања хибридне *SQL*/*NoSQL* базе података, примену новог приступа у пројектовању на конкретном примеру, приказ добијеног решења, поређење остварених резултата тако пројектоване хибридне базе података са традиционалном базом података и приказ униформног начина коришћења хибридне *SQL*/*NoSQL* базе података, без обзира којој компоненти се приступа (*SQL* или *NoSQL*). На крају, дат је закључак у поглављу осам и списак коришћене литературе.

# 2. Основни концепти и типови база података

База података представља колекцију структурираних података која постоји и користи се релативно дуго. Два аспекта база података се посебно издвајају по свом значају (Lazarević, et al., 2006):

- Модел података и
- Системи за управљање базом података (СУБП).

Модел података представља теоријски приступ којим се одређује начин спецификације и пројектовања конкретне базе података, док систем за управљање базом података чини специфична технологија обраде података, тј. софтверски систем који омогућава управљање великом количином података, а који имплементира одређени модел података (Lazarević, et al., 2006). Конкретан модел података одређује логичку структуру података и начин њеног коришћења. Ово уједно представља и основни критеријум разликовања конкретних система за управљање базом података.

## 2.1. *SQL* базе података

На тржишту база података данас су убедљиво најзаступљеније *SQL* базе података (SolidIT, 2018). Под термином *SQL* базе података у пракси се подразумевају релационе и објектно-релационе базе података. С обзиром на то да СУБП и модел података представљају два најважнија аспекта базе података, за оне базе података које се заснивају на релационом моделу (самим тим које користе и релациони СУБП) употребљава се термин релациона база података. Схватајући значај релационог модела, а истовремено поштујући тренд и потребе објектног програмирања, водећи произвођачи релационих база података су понудили подршку за објектне типове и наслеђивање. Тако је релациони модел проширен концептима објектног модела, што је довело до настанка објектно-релационих база података. У основи објектно-релационих база података и даље се налази релациони модел, те се подаци уносе у форми

рекорда (тј. записа базе података, односно слога базе података), али је присутна и подршка за рад са објектним типовима података.

Главне карактеристике *SQL* база података су, поред заснованости на релационом моделу, да захтевају испуњеност *ACID* особина трансакција, да подржавају *SQL* упитни језик и да су погодне за вертикалну скалабилност. *SQL* представља стандардни упитни језик за рад са *SQL* базама података (релационим и уз одређена проширења и за рад са објектно-релационим базама података). Овај стандард је усвојен од стране *American National Standards Institute* (*ANSI*) и постао је стандард који подржавају сви водећи произвођачи *SQL* база података. Свака трансакција над *SQL* базом података мора да испуни *ACID* особине (Lazarević, et al., 2006):

- Атомност (*Atomicity*) – све операције трансакције над базом података се морају успешно обавити или ниједна неће бити реализована;
- Конзистентност (*Consistency*) – пре почетка и након завршетка трансакције база података мора бити у конзистентном стању;
- Изолација (*Isolation*) – ефекти две или више трансакција које се извршавају истовремено морају бити изоловани и
- Трајност (*Durability*) – ефекти завршене трансакције не могу бити изгубљени.

Крајем прошлог века, у употреби су осим *SQL* база података, које су биле доминантне по заступљености (Atzeni, et al., 2013), били присутни и други типови база података, али у знатно мањој мери. У том периоду, избор типа базе података који је био пред организацијама није био превише комплексан. Осим у малом броју случајева када је постојала потреба за специфичним типовима база података, попут објектних база података или *XML* база података, организације су се углавном одлучивале за *SQL* базе података. Временом, постављани су нови типови задатака пред *SQL* базе података, а очекивања од примене база података су се усложњавала. Неке од новонасталих потреба су решаване проширењем функционалности *SQL* база података, попут додавања кориснички дефинисаних типова, у циљу подршке објектних концепата, или додавања „Звезда“ шеме (*Star Schema*), у циљу адекватнијег пројектовања складишта података (*datawarehouse*). Тако на пример *Oracle* СУБП подржава рад са

објектним типом и садржи уграђену подршку за „Звезда“ трансформацију. Експанзија велике количине података различитог степена структурираности, доступних из различитих извора (не само традиционалних извора, већ и новијих попут друштвених мрежа), додатно је нагласила потребу брже обраде података промењиве структуре доступних у различитим форматима.

## 2.2. *NoSQL* базе података

У наведеним околностима *SQL* базе података нису показале завидан ниво перформанси, пре свега због својих карактеристика: неопходности дефинисања стриктне структуре података који се чувају и због безусловног обезбеђивања конзистентности података. То је довело до потребе креирања база података које би за разлику од *SQL* база података акценат ставиле на приказ великих количина података (не нужно структурираних) у најкраћем року, чак и по цену неиспуњавања свих особина *ACID*-а. Формализацију новог приступа поставио је проф. др *Eric Brewer*, својом теоремом *Consistency Availability Partition tolerance* (*CAP* теорема), која је затим потврђена и на *MIT*-у (Gilbert, & Lynch, 2012) и која гласи: „Сваки дистрибуирани систем може имати задовољене максимално две од следеће три карактеристике: конзистентност (*Consistency*), доступност (*Availability*) и толеранција ка отказу мреже (*tolerance towards network Partition*)“ (Brewer, 2000). Ова теорема представља основу *NoSQL* база података. У споменутој теореми идентификоване су *BASE* особине, путем којих се маргинализује значај конзистентности и изолованости из *ACID* особина у корист доступности и перформанси. *BASE* особине чине (Brewer, 2000):

1) Доступност у начелу (*Basically Available*) – није неопходно да сви делови система буду доступни у сваком тренутку да би он био доступан;
2) Конзистентност у начелу (*Soft-state*) – конзистентност система није неопходна у сваком тренутку и
3) Коначна конзистентност (*Eventual consistency*) – не захтева се моментална конзистентност свих делова система након сваке извршене трансакције, али ће се на крају сви чворови поново вратити у конзистентно стање.

*NoSQL* базе података су настале у циљу испуњавања захтева у којима се *SQL* базе података нису показале ефикасним, попут управљања великом количином неструктурираних података или које нису могле да задовоље, попут пружања хоризонталне скалабилности или омогућавања рада са подацима без претходно дефинисане шеме. Изазов обраде велике количине података *NoSQL* базе података неретко решавају применом *Map Reduce* принципа (Bonnet, et al., 2011; Dean, & Ghemawat, 2008; Grolinger, et al., 2014; Liao, Shih, & Chang, 2013).

*NoSQL* базе података, којима је заједничко одсуство формално дефинисане структуре података, не подржавају релациони модел, ни *SQL* упитни језик и заснивају се на *CAP* теореми (и *BASE* особинама). Оне омогућавају примену хоризонталне скалабилности, којом се подаци дистрибуирају на више сервера (Lake, & Crowther, 2013).

Кључ-вредност (*key-value*) базе података заснивају се на истоименом моделу, који други типови *NoSQL* база података (базе података засноване на документима и на фамилијама колона) проширују. Уређени пар кључ-вредност има флексибилну структуру, тј. омогућава да различити парови садрже различите пропертије, што представља значајну разлику у односу на стриктно дефинисану шему *SQL* база података. У овом типу базе података кључ има двоструку функцију:

1) Служи као идентификатор вредности и
2) Представља индексирано поље по којем се филтрира претрага.

Одсуство транспарентности вредности условило је да се у кључ-вредност базама података претрага врши искључиво по кључу. Немогућност филтрирања по садржају вредности представља и значајан ограничавајући фактор приликом извршавања упита над овим типом базе података. Пар кључ-вредност има структуру сличну асоцијативном односно *hash* низу, док вредност може бити састављена од више поља различитих типова података, укључујући и низове и објекте (Gurevich, 2015). Одсуство стриктно дефинисане шеме позитивно утиче на ефикасност складиштења дистрибуираних података по чворовима, док са друге стране отежава складиштење података сложене структуре, због чега се овакви захтеви углавном решавају кроз код апликације (Grolinger, et al., 2013). Због немогућности ефикасног претраживања по опсегу кључева, појавиле су се базе

података која су имплементирале уређене кључеве и оне су назване „уређене кључ-вредност“ (*ordered key-value*) базе података. У овом раду, споменути тип база података се третира као специфична верзија кључ-вредности базе података и неће бити посебно анализиран. Представници кључ-вредност база података су: *Redis*, *BerkeleyDB*, *Voldemort*, *Oracle KV* и *Riak KV*.

Базе података засноване на документима, као што и сам назив сугерише, имају модел документа у својој основи. Концепт документа представља екстензију кључ-вредност модела. Најзначајнија разлика у односу на „традиционални“ кључ-вредност модел је у чињеници да је код документ модела вредност увек сачувана у форми документа. Зато је прихватљиво рећи да базе података засноване на документу користе уређени пар кључ-документ. Овај тип базе података је посебно погодан за системе у којима се подаци на интуитиван начин могу организовати у виду докумената (Grolinger, et al., 2014). Главна повезаност са кључ-вредност базама података видљива је у одсуству шеме и улози кључа као идентификатора, чиме је јединствено одређен сваки документ. Међутим, за разлику од кључ-вредност база података, у базама података заснованим на документима поред претраге по кључу омогућена је претрага и по вредности, тј. по структури документа (Gurevich, 2015). Структура документа може варирати од једноставне до сложене која у себи садржи друге документе. Представници база података овог типа документа најчешће организују по колекцијама (MongoDB, 2018). Свако појављивање документа може имати произвољну структуру, односно није неопходно да сви документи унутар колекције имају исти број и тип поља. Значајно побољшање у односу на кључ-вредност базе података се огледа у видљивости садржаја докумената приликом извршавања упита. Тиме је омогућено да се резултујући подаци филтрирају не само по вредности кључа, као код кључ-вредност база података, већ и по садржају документа, што је додатно допринело популарности овог типа базе података. Главни недостатак огледа се у немогућности спајања података из различитих докумената, изузев у ситуацији када су документа међусобно угњеждена. Представници овог типа *NoSQL* база података су: *MongoDB* и *Couchbase*.

Базе података које се заснивају на фамилији колона настале су као деривати *Google*-ове *BigTable* базе података (Chang, et al., 2008), због чега неки аутори и целу групу база података називају по споменутом решењу. Уколико би се модел овог типа базе података дефинисао преко уређеног пара кључ-вредност, онда би вредност имала структуру фамилије колона. Сваки ред је сачињен од фамилије колона (комбинације колона), при чему сваки ред може имати сопствену фамилију колона. Свака фамилија колона је сачињена од више колона, при чему свака колона садржи пар кључ-вредност и одређена је кључем фамилије колона којој припада (Grolinger, et al., 2013). Неки представници овог типа база података (нпр. *Cassandra* базе података) уводе концепт супер-колоне, којим је дозвољено да вредност колоне буде сложена, тј. да вредност колоне буде нека фамилија-колона (Gurevich, 2015). Најважније побољшање у односу на кључ-вредност базе података огледа се у унапређеном индексирању, јер се у ту сврху, поред примарних кључева фамилије колона, могу користити и кључеви колона. Неки од представника овог типа базе података су: *BigTable*, *Cassandra* и *Hbase*.

Граф базе података се заснивају на моделу графа (Sakr, & Pardede, 2011), који је настао на постулатима теорије графова. Граф представља структуру чије саставне елементе чине чворови и ивице, при чему свака до ових компоненти може имати своје карактеристике. Без обзира што је чвор графа осмишљен да садржи вредности, а ивица графа да представља релације између података, дефинишући тиме „везу“ која се успоставља између чворова, свака од две наведене компоненте може садржати пар кључ-вредност. Иако је тиме примењен концепт кључ-вредност у граф базама података, специфичност модела који користе чине да овај тип базе података не буде уско повезан са кључ-вредност базама података, за разлику од осталих типова *NoSQL* база података. Анализу специфичности рада са граф базама података употребом доступних језика приказали су аутори у раду (Sakr, & Al-Naymat, 2010). Разлог због којег су граф базе података припојене категорији *NoSQL* база података представља одсуство подршке релационог модела, као и потпуни изостанак подршке *SQL* језика. Упркос томе, није ретка ситуација да представници граф база података у потпуности подржавају *ACID* особине трансакција, попут *Neo4J*,

најпознатијег представника граф база података (De Virgilio, Maccioni, & Torlone, 2014).

Појава *NoSQL* база података није угрозила егзистенцију *SQL* база. *SQL* базе задржавају примат у системима где је императив конзистентност података, где се употреба података огледа у доминантном постављању сложених упита и где се од система очекује вертикална скалабилност. Са друге стране, *NoSQL* базе се примарно користе у системима интензивног читања и писања, у системима у којима није унапред позната структура записа, већ је она флексибилна и прилагођава се структури записа који се тек уносе, као и у системима у којима је битна хоризонтална скалабилност. Код *NoSQL* база подаци су обично полу-структурирани, а императив представља брзина одзива, чак и по цену нарушавања конзистентности података. Наведене карактеристике утицале су да *SQL* и *NoSQL* базе наставе да коегзистирају, задржавајући већу ефикасност примене, свака за своју намену.

## 2.3. Појам хибридне *SQL*/*NoSQL* базе података

За разлику од краја прошлог века, данас избор типа базе података није једноставан за организације и зависи од много аспеката пословања. За организације које своје пословање заснивају на широко доступним садржајима на *Web*-у (посебно уколико користе апликације развијене по *Web 2.0* принципу, у којима је садржај неретко повезан са подацима са друштвених мрежа) избор се у начелу своди на неки од типова *NoSQL* база података. За разлику од њих, организацијама чије пословање као приоритет намеће брзо и поуздано извршавање великог броја трансакција, уз гарантовање интегритета података, као природан избор се намеће *SQL* базе података. Међутим, не мали број организација има реалну потребу коришћења погодности које пружају и *SQL* и *NoSQL* базе података.

Ради што ефикаснијег испуњавања свих захтева који се данас могу поставити пред базу података, није увек једноставно донети одлуку о избору једног од наведених типова база, јер не можемо имати гаранцију да је у тренутку доношења та одлука оправдана или сигурно исправна. Уместо тога, потребно је омогућити паралелно коришћења *SQL* и *NoSQL* база и њихово

повезивање у интегралну логичку целину, а у циљу обухватања најбољих функционалности из оба света. Изазови интеграције различитих типова система и униформне размене података између њих постоје већ дуго (ISO, 1997). У прилог томе говори и чињеница да компанија *Facebook*, поред *Cassandra* базе података (*NoSQL* база података посебно развијена за *Facebook*) користи и *SQL* базу података (конкретно *MySQL* СУБП) (Nance, et al., 2013). Осим тога, додатни аргумент представља и чињеница да водећи произвођачи база података, попут *Oracle*-а и *Microsoft*-а, у палети својих производа годинама уназад поседују и *SQL* и *NoSQL* СУБП-ове и да развијају начин њихове интеграције. Пример таквог решења је *Oracle NoSQL* СУБП (Oracle, 2016).

Популарност и специфичност домена примене *SQL* и *NoSQL* база података проузроковала је потребу њиховог паралелног коришћења у савременом пословању. Одређени типови захтева условљавају испуњење *ACID* особина трансакције и формално дефинисање шеме базе, док други као императив постављају хоризонталну скалабилност и флексибилност коју пружа одсуство унапред дефинисане шеме. Једно од решења овако сучељених захтева представља креирање новог типа базе података, **хибридне *SQL*/*NoSQL* базе података**. Овај тип базе података обухвата све посебности *SQL* и *NoSQL* база података. Састоји се од две или више компоненти, при чему сваку компоненту чини одговарајући тип базе података (*SQL* база података или неки од четири подтипова *NoSQL* база података). У овој дисертацији биће приказан нови и оригинални приступ за пројектовање и коришћење хибридне *SQL*/*NoSQL* базе података**.**

# 3. Постојећи приступи и технике за пројектовање *SQL* и *NoSQL* база података

Деценијама уназад, организације користе податке у свим фазама свог животног циклуса, стога подаци представљају један од неопходних ресурса у пословању сваке компаније. У зависности у којој се фази налази сама организација, различита је и улога и значај података за њено функционисање. Нове организације настају на основу података којима њихови оснивачи располажу, а које користе ради темпирања правовременог оснивања и увођења организације на тржиште. Младе организације се позиционирају на тржишту на основу прикупљених и интерпретираних података, организације у зрелом добу труде се да задрже и ојачају позицију на тржишту на основу релевантних и правовремених података, док оне у позном стадијуму пословања настоје да податке искористе у циљу „подмлађивања“ организације, тј. у циљу реализације транзиције из позне у млађе фазе животног циклуса развоја организације. Међутим, заједничко за организације у свим добима је да податке покушавају да искористе зарад доношења добрих пословних одлука и ради стицања конкурентске предности, подизања нивоа задовољства постојећих клијената и стицања нових, а све у циљу оправдања сврсисходности постојања, кроз максимизацију профита.

Да би подаци били употребљиви организацији, неопходно је да буду трајни, тј. перзистентни. Неминовност прикупљања, обраде, организације, складиштења, чувања и коришћења трајних података условила је настанак и коришћење база података. Прве базе података појавиле су се средином прошлог века и биле су у почетку коришћене самостално. Развој техника програмирања омогућио је и израду и повезивање различитих апликација са базама података, да би, након тога, читави информациони системи били развијани, са базом података као једним од најважнијих ресурса. Однос између база података и информационих система и њихова међусобна условљеност су се мењали кроз различита раздобља. Међутим, оно што је заједничко за базе података и информационе системе, још од њихових почетака, јесте потреба њиховог пројектовања. Последица одсуства систематичног и планираног пројектовања како информационог система, тако и базе података

имплицира велике утрошке ресурса, попут новца и времена (Churcher, 2007). Неадекватно пројектована база података, која није пројектована у складу са корисничким захтевима и очекивањима, најчешће узрокује озбиљне последице, које неретко повећавају скривене трошкове. Неадекватна структура базе података онемогућава организацију у испуњењу свих корисничких захтева, а измена и прилагођавање структуре представља додатно ангажовање људских ресурса и додатни утрошак ресурса времена и новца. Неретко се дешава да озбиљне пропусте у пројектовању није лако отклонити чак ни употребом додатних ресурса, па се организације могу наћи пред озбиљним изазовом поновног пројектовања дела или чак целе базе података, односно информационог система, што негативно утиче на профитабилност организације. Због свега наведеног, пројектовање базе података и целокупног информационог система од великог је значаја за њихово касније коришћење и одржавање. Током година, појавили су се методолошки приступи за пројектовање информационих система, применљиви и на базе података, који ће бити анализирани у наставку.

## 3.1. Пројектовање база података

Базе података представљају један од најзначајнијих ресурса сваког информационог система, за који се методолошки приступи пројектовања развијају деценијама уназад. С обзиром на то да не постоји консензус око терминологије у употреби (Avison, & Fitzgerald, 2003; Bugiotti, 2012; Carr, & Verner, 1997; Isaias, & Issa, 2015; Jirava, 2004; Rob, 2004), па се тако неретко исти појам третира као методологија, методолошки приступ или модел, а због значаја за предмет истраживања ове дисертације, овде ће бити направљена дистинкција споменутих појмова, која ће се односити и на остатак дисертације. Како аутори (Avison, & Fitzgerald, 2003) наводе „методологија представља колекцију препоручених фаза, процедура, правила, техника, алата, документације, менаџмента и тренинга неопходних за развој система“. Аутори (Isaias, & Issa, 2015) наводе да методологија може имати више верзија приступа, у зависности од апсеката које укључују и исте називају моделима. Појам модела аутори (Rumbaugh, Jacobson, & Booch, 2004) дефинишу као „репрезентацију битних аспеката онога шта се моделује са одређене тачке

гледишта, док се остатак поједностављује или изоставља“. Узимајући у обзир да се термин модел одомаћено користи у контексту модела података, модела система, али и модела животног века система који се пројектује, у наставку се реч „модел“ конкурентно користи за сваки од три појма, при чему је значење речи у конкретној ситуацији могуће одредити на основу контекста у којем се налази. Због наведеног, као и због значаја појма „методолошки приступ“ за наслов, али и садржај, ове дисертације предстоји његово прецизирање.

Методолошки приступ је могуће дефинисати као појам којим се комбинују аспекти различитих методологија, сродних по својој намени, проширујући тако домен њихове примене. У објашњеном значењу појам методолошких приступа ће бити коришћен и у наставку дисертације. Методолошки приступи дају одговарајућу флексибилност у односу на методологију, јер није потребно дефинисати све аспекте наведене у дефиницији методологије. Такође, у приступима је могуће комбиновати неке од аспеката постојећих методологија. Методолошки приступи који ће бити анализирани у овом раду, примењиви су како на развој информационих система, тако и на пројектовање база података и садрже различите аспекте од других приступа и методологија. Због тога ће у овој дисертацији бити подведени под методолошке приступе, а не „чисте и стриктне“ методологије.

За методолошке приступе анализиране у овом раду од значаја ће бити пре свега следећи аспекти: фазе, процедуре, правила и технике пројектовања. Томе доприноси заједничка карактеристика анализираних методолошких приступа да подржавају фазе пројектовања базе података и фазе спецификације апликација, при чему је подржана паралелна реализација споменутих фаза. Због свега наведеног анализирани методолошки приступи, иако у одређеној литератури наведени као методолошки приступи развоја информационих система (Avison, & Fitzgerald, 2003; Isaias, & Issa, 2015), потврђују адекватност своје примене и на пројектовање база података. С обзиром на то да аутори (Marjanović, Aničić, & Babarogić, 2000) наводе да комплетна спецификација информационог система обухвата: спецификацију базе података, спецификацију апликација и нефункционалну спецификацију, од интереса за овај рад је пре свега спецификација базе података. Због степена заступљености *SQL* база података, њихове нераскидиве повезаности

са информационим системима, као и због значаја за овај рад, општи методолошки приступи пројектовања информационих система, а који су анализирани у следећем потпоглављу, биће описивани пре свега са аспекта пројектовања *SQL* база података.

## 3.2. Приступи и технике пројектовања *SQL* база података

У циљу бољег разумевања постојећих приступа за пројектовање база података, али и целокупних информационих система, биће укратко описан историјски развој процеса пројектовања. Период раног пројектовања база података и апликација са почетка друге половне прошлог века, био је обележен одсуством формализације и одсуством примене методолошких приступа (Avison, & Fitzgerald, 2006a). Структура апликација и база података развијана је „ад-хок“, без опсежног прикупљања, идентификације и анализе корисничких захтева. Због наведеног, аутори (Marjanović, Aničić, & Babarogić, 2014) овај период називају „херојско доба“. Карактеристично за овај период је било да су проблеми, када би се појавили, били решавани програмирањем, па је то уједно била и пожељна вештина, док се познавању пословних процеса и пословања организације није придавало значајније место. Превазилажење технолошких ограничења (попут доступне меморије и других ресурса) неретко је представљао повод за реализацију софтверских пројеката тога времена (Avison, & Fitzgerald, 2003). Постојећи системи су ретко бивали унапређивани, а нове потребе корисника неретко су биле задовољаване израдом нових апликација. Ово су били само неки од фактора који су утицали на неоптимално и непланско коришћење ресурса, на високе трошкове развоја и одржавања апликација крутих и неадекватних функционалности, што је резултирало развојем методолошких приступа. Методолошки приступи пројектовања почињу да се развијају седамдесетих година прошлог века у циљу (Churcher, 2007):

1) Спецификације корака, дефинисања техника, метода и алата пројектовања база података и
2) Формализације процеса развоја апликација.

Први од два наведена циља представља домен спроведених истраживања за потребе развоја новог методолошког приступа пројектовања хибридне базе података, насталог као резултат рада на овој дисертацији. База података представља један од најважнијих ресурса информационог система, на којем је самим тим могуће применити методолошки приступи пројектовања целокупног информационог система.

У наставку поглавља биће приказани најпопуларнији приступи за пројектовање, углавном примењиви на *SQL* базе података, а биће описане и најчешће коришћене технике пројектовања *SQL* база података. Након тога, биће описана тренутна ситуација по питању пројектовања *NoSQL* база података, описани тренутно доступни приступи и расположиве технике за пројектовање *NoSQL* база података.

### 3.2.1. Модел животног циклуса развоја система (*SDLC*)

Први формализовани приступ пројектовања информационих система представља Модел животног циклуса развоја система (*Systems Development Life Cycle*, у наставку *SDLC*) модел. Неки аутори *SDLC* дефинишу као први развојни оквир и користе појам модела за њега (Isaias, & Issa, 2015), док други наглашавају његово природно припадање методолошким приступима, иако у тренутку постављања његових основа *SDLC* није био тако формулисан (Avison, & Fitzgerald, 2006a). Иако је и појам модела прихватљив, због препорука, правила, техника и алата које дефинише уобичајено је да се *SDLC* третира као први методолошки приступ пројектовања информационог система, (Avison, & Fitzgerald, 2003). *SDLC* се и у овој дисертацији третира као први формализовани методолошки приступ пројектовања информационих система, који је примењив и на његове саставне делове, а који је посебно интересантан (као и приступи који га следе) због могућности примене на пројектовање *SQL* база података.

Почеци примене *SDLC*-а везују се за седамдесете и осамдесете године прошлог века. Развијен за потребе пројектовања апликација, временом је постао уобичајени приступ за пројектовање великих система, укључујући и целокупне информационе системе (Isaias, & Issa, 2015). Процес пројектовања информационог

система је у међувремену постао додатно комплексан, јер је обим захтева на које је информациони систем требало да одговори континуирано повећаван. Поред решавања техничких захтева, све се више потенцира сврсисходност информационог система у пословању организације за коју је израђен. Заоштравање конкуренције на тржишту и анализирање сваког сегмента профитабилности пословања, као императив пред информационе системе поставља њихову адекватност намењеној организацији, обим и стабилност имплементираних функционалности, као и задовољство крајњег корисника у коришћењу система. Из тога произилазе и два важна аспекта животног циклуса која наглашавају аутори (Marjanović, Aničić, & Babarogić, 2000): очекивања свих корисника укључених у рад са системом и поседовање стручних знања пројектаната за контекст у којем ће систем функционисати.

Проблем сложености развоја савременог информационог система, *SDLC* је решио поделом целокупног процеса развоја на фазе (Marjanović, Aničić, & Babarogić, 2014). Фаза садржи почетни и крајњи тренутак, скуп специфичних активности које се у њој извршавају, као и алате за праћење извршавања (Marjanović, Aničić, & Babarogić, 2000). Неки аутори (Jirava, 2004) животни циклус развоја третирају као временски оквир, који почиње тренутком рађања идеје за пројектовањем, прати целокупан развој и завршава се увођењем система.

Уопштено, *SDLC* је увео секвенцијално дефинисане фазе развоја. Излаз из сваке фазе представља неопходан улаз наредне фазе, чиме је обезбеђен секвенцијални редослед извршавања. Студија изводљивости, истраживање реалног система, анализа, пројектовање, имплементација, одржавање и анализа повратних информација су фазе које су биле заступљене у почетном *SDLC* приступу (Avison, & Fitzgerald, 2006a).

Поред иновативности у процесу пројектовања који је донео поделом самог процеса на фазе пројектовања, основни *SDLC* приступ имао је доста ограничења. Систематизацију и опис највећих недостатака основног *SDLC* приступа дали су аутори (Avison, & Fitzgerald, 2003):

- Немогућност испуњавања стварних потреба организације - проузрокована је стављањем акцента на унапређење технологије и ефикасности исте, пре него на унапређење функционалности система. Тиме није омогућено

континуирано и подједнако фреквентно увођење промена у пословне процесе и технологију;

- Конзервативан приступ пројектовању система - узрокован је постављањем постојећег система као основе новог система, који се тек пројектује. Главни проблем оваквог начина пројектовања је лимитираност постојећим системом и свођење новог система на побољшану верзију постојећег, са одсуством могућности пројектовања „од нуле“ и креирања суштински адекватнијег система;
- Нестабилност система - директно је повезана са ниским степеном флексибилности система на промене у пословању. Прилагођавање организације променљивим тржишним условима и брзи реинжењеринг пословних процеса не могу се једноставно инкорпорирати у овај методолошки приступ. Измене у систему у циљу адаптибилног одржавања нису практиковане у овом приступу, баш због стриктно секвенцијалних фаза пројектовања које су дефинисане. Поново пројектовање делова система, овим приступом би захтевало пролазак кроз скоро све фазе пројектовања, што захтева велико улагање ресурса (пре свега времена и новца). Уколико се приступи „скраћеном“ процесу измене система, прескакање фаза може довести до неконзистентности система са спецификацијом и званичном документацијом. Такође, процес поновног пројектовања и измене делова постојећег система може проузроковати и недоступност истог;
- Незадовољство корисника проистиче из већ споменутих недостатака, пре свега усмерености *SDLC* приступа на технологију и давање истој предности у односу на организацију и крајњег корисника. Овај приступ дефинише „техничку“ спецификацију система, што неретко утиче на неразумевање функционалности и „изгледа“ будућег система од стране корисника, пре него што систем буде завршен и пуштен у рад.

Временом, настале су специфичне верзије основног *SDLC* приступа. У литератури верзије приступа су често називане моделима Животног циклуса развоја ИС, тј. моделима *SDLC* (Avison, & Fitzgerald, 2003; Avison, & Fitzgerald, 2006a; Isaias, & Issa, 2015), па ће ти термини и у овој дисертацији бити коришћени као истоветни. Главна сврха ових приступа била је да се прошири основни *SDLC*

приступ и да се отклоне неки од његових недостатака. Међу методолошким приступима уочавају се они који су директно повезани са основним *SDLC* приступом (попут „Водопад“ модела или Спиралног), док постоје и они који су се у већој мери дистанцирали од основних постулата *SDLC* приступа (попут Агилног). Због тога аутори праве оштру дистинкцију између *SDLC* приступа и новијих методолошких приступа развоја, при чему савременије методолошке приступе (Објектни, Агилни итд.) третирају као релативно независне приступе, који имају само додирних елемената са основним *SDLC* приступом (Avison, & Fitzgerald, 2003). Општи закључак гласи: *SDLC* представља претечу савремених методолошких приступа и чини њихов основ, али због одсуства његове директне примене данас, као и због удаљености новијих методолошких приступа од основних начела *SDLC*-а, традиционални *SDLC* данас има пре свега историјски значај, али и улогу споне са новијим приступима који су на његовим токовима настали. Ипак и у новијим приступима требало би спровести темељну студију за дефинисање прецизне стратегије развоја великих система, управо по узору на *SDLC*. Иако се брз развој апликација (*RAD*), агилно и објектно-оријентисано пројектовање могу посматрати као својеврсне парадигме, због аспеката које су увели у процес пројектовања и због повезаности са основним начелима *SDLC*-а они ће у овој дисертацији бити третирани као методолошки приступи пројектовања.

Различити су основи класификације приступа заснованих на *SDLC* моделу, а једна од подела, која је наведена у наставку, је по типу (принципу) развоја који испуњавају (Marjanović, Aničić, & Babarogić, 2014):

- Конвенционални развој, који захтева стриктно праћење свих правила инжењерског приступа развоја информационог система;
- Брзи развој, који као приоритет поставља израду решења у најкраћем року, при чему се решење након тога додатно усавршава и
- Формални (трансформациони) развој, који прописује дефинисање формалних модела и поступака развоја, односно који дефинише имплементацију као производ формалне трансформације формалне спецификације информационог система.

Чињеница која ову поделу раздваја од других сличног типа је да споменути аутори наводе да конкретни методолошки приступи могу испуњавати два или чак

сва три принципа развоја, при чему је омогућено флексибилније категорисање методолошких приступа. Као што је већ споменуто, у литератури се појављује велики број методолошких приступа, тј. модела *SDLC*-а. У наставку овог поглавља биће представљени методолошких приступи који се базирају на *SDLC* моделу и који се посебно издвајају по својој заступљености и значају: „Водопад“ модел, *V* модел, Спирални модел, Инкрементални модел, Итеративно-инкрементални модел, Агилни модел, *RAD* модел, Структурни модел, Објектни модел, Прототипски модел и *Contigency* модел (модел „непредвиђене ситуације“).

### 3.2.2. Конвенционални „Водопад“ модел

Модел „Водопад“ представља прво и једно од најзаступљенијих побољшања традиционалног *SDLC* приступа (Isaias, & Issa, 2015). Постулате овог приступа поставио је др *Winston Royce* радом (Royce, 1970) са почетка седамдесетих година прошлог века. У наставку овог потпоглавља дисертације биће изложени најважнији постулати из тог рада. Наведени аутор започиње свој рад констатацијом да су два природна корака у развоју било којег софтверског система анализа и кодирање. Иако та два корака представљају саставни део сваког система, они су довољни искључиво за мале системе. За системе већих размера, наглашена је неопходност увођења додатних фаза развоја и аутор идентификује укупно 7 фаза (Royce, 1970):

1) Идентификација захтева система,
2) Идентификација захтева софтвера,
3) Анализа,
4) Пројектовање,
5) Кодирање,
6) Тестирање и
7) Одржавање.

Фазе су секвенцијалне, при чему излаз из једне фазе представља улаз у наредну. У раду је коментарисана и могућност увођења итерација, при чему би за сваку фазу поред преласка на прву следећу било омогућено и враћање на једну претходну. Од овог концепта се одустало, због *Royce*-овог искуства из праксе, које је сугерисало да постоји основана потреба повезивања повратном везом фаза које нису узастопне,

нпр. фазе тестирања са пројектовањем (прескаче се кодирање), као и пројектовања са фазом идентификације софтверских захтева (прескаче се фаза анализе). Уместо увођења повратних веза, у овом приступу је дефинисано 5 принципа које је потребно испунити током процеса развоја информационог система (Royce, 1970):

1) Принцип „прво урадити нацрт пројекта“ сугерише да би у процес прикупљања корисничких захтева требало укључити и пројектанте, због могућности њихове иницијалне процене потребних ресурса и ограничења.
2) Принцип „документовати пројектовање“ наглашава значај израде документације. У овом приступу посебна пажња је посвећена важности улоге пројектанта при креирању потпуне и тачне документације, па се као решење проблема лоше и непотпуне документације наводи замена главног пројектанта, тј. вође пројекта.
3) Принцип „урадити два пута“ се заснива на увођењу међу-фазе, између идентификације захтева софтвера и фазе анализе, при чему би резултат међу-фазе био симулациони модел система, којим се потенцијално могу отклонити пропусти пре фаза пројектовања, кодирања и тестирања.
4) Принцип „планирање, контрола и праћење тестирања“ подразумева обавезно укључивање специјалиста изван пројекта у фази тестирања. Основна идеја је да специјалисти који нису били укључени као пројектанти система могу објективније да тестирају тај систем. По овом принципу творац приступа даје велики значај тестирању свих појединости система без обзира колико су оне значајне на глобалном нивоу, као и праћењу начина тестирања и постигнутих резултата.
5) Принцип „укључивање клијента“ подразумева да клијенти, на формални начин, буду укључени у процес развоја и то од почетних фаза, ради благовременог сугерисања потребних измена и повећања степена задовољства клијената.

Популарност „Водопад“ проширења *SDLC*-а изнедрила је нове методолошке приступе, као што су Спирални модел и *V* модел. Ипак, поред проширења која је увео, овај модел је задржао и нека ограничења класичног *SDLC* модела. Секвенцијално повезане фазе, без повратне везе, смањују флексибилност у процесу развоја и онемогућују враћање на претходне фазе (Balaji, & Murugaiyan, 2012).

Поред мањка флексибилности, и даље је приметан дуг процес увођења промена у систем, чиме се продужава и време одговара на стварне и текуће потребе организација. „Водопад“ модел је предложио увођење симулационог модела система, па у томе представља искорак у односу на конзервативни модел система традиционалног *SDLC* приступа, ангажованија улога корисника такође представља побољшање у односу на традиционални *SDLC*, док је највећи помак постигнут инсистирањем на ажурној документацији. Детаљнији приказ предности и недостатака „Водопад“ модела доступан је у литератури (Balaji, & Murugaiyan, 2012; Isaias, & Issa, 2015; Royce, 1970).

### 3.2.3. Инкрементални модел

Инкрементални модел представља специфично проширење „Водопад“ модела. Главни сегмент у којем овај модел покушава да унапреди „Водопад“ модел је дужина трајања циклуса пројектовања. Инкрементални модел напушта принцип секвенцијалних фаза без повратне спреге, карактеристичан за „Водопад“ модел и омогућава извршавање целокупног животног циклуса развоја информационог система у кратком временском периоду. То је омогућено кроз постепено пројектовање, имплементацију и тестирање одређених функционалности система, а не целог система одједном (Isaias, & Issa, 2015). Инкремент представља етапу у развоју и након сваке завршене етапе, тј. инкремента, добија се одређена верзија система, која се проверава од стране корисника, након чега се приступа реализацији новог инкремента, све док нека од верзија система не задовољи све захтеве и постане финални систем. Сваки инкремент додаје нове функционалности или проширује постојеће у односу на претходни инкремент и тако утиче на тренутну верзију система (Marjanović, Aničić, & Babarogić, 2014). Активнијим укључивањем клијента и добијањем повратних информација након сваког завршеног инкремента остварују се две главне користи: добијене информације утичу као смернице за наредни инкремент и будући систем се константно тестира, са сваким новим инкрементом. Уколико је реч о мањим системима, верзија система може чинити систем са свим функционалностима, који се након тестирање од стране клијента проширује евентуално новим, све док не обухвати све жељене функционалности и

карактеристике, када постаје финални систем. Уколико је реч о већем систему, резултат реализације инкремента може бити подсистем (скуп функционалности груписан у логичну целину) или делимичан систем (систем у којем нису имплементиране све функционалности). Иако је у основи овог модела секвенцијална зависност фаза развоја, повећање нивоа флексибилности процеса пројектовања постиже се изменама, побољшањима и додавањима нових функционалности између два инкремента (Isaias, & Issa, 2015). Све наведено утиче да је процес развоја растерећенији и бржи, због могућности лакшег увођења накнадних захтева, који су уочени у процесу развоја, а нису били препознати приликом иницијалне спецификације захтева. Два главна проблема која се јављају код овог приступа су: потенцијално скупљи развој, уколико се број верзија пре финалне отргне контроли и усклађивање различитих верзија, односно подсистема (Isaias, & Issa, 2015).

### 3.2.4. Итеративно - инкрементални модел

Главна одлика Итеративног модела је развој целокупног система, проласком кроз све стандардне фазе *SDLC*-а, попут анализе, пројектовања, кодирања и тестирања, при чему се акценат ставља на тестирање. Уколико систем након иницијалне итерације пројектовања не задовољава све идентификоване захтеве, приступа се покретању нове итерације, у којој се такође пролази кроз све наведене фазе развоја. Оног тренутка када се фазом тестирања утврди да систем задовољава све потребне захтеве, обустављају се даље итерације и систем се проглашава завршеним.

Главна разлика Итеративно-инкременталног модела у односу на Итеративни модел, од којег у литератури није увек јасно одвојен (Isaias, & Issa, 2015), представља принцип да се од старта пројектује целокупан систем, са свим функционалностима. Инкрементални модел, са друге стране, од старта пројектује део система, са одређеним функционалностима, при чему се новим инкрементом додају нове функционалности, док се не добије комплетно имплементиран систем.

Природна блискост објашњених појмова инкремента и итерације проузроковала је и рађање приступа који комбинује оба концепта. Итеративно-инкрементални модел омогућава да се систем развија у итерацијама, при чему свака

итерација представља различиту верзију система (или подсистема), тј. различити инкремент, чиме је омогућено да у одређеном тренутку више итерација може бити у току (Marjanović, Aničić, & Babarogić, 2014). У истом тренутку, могуће је и да различити инкременти буду у различитим, али и истим фазама развоја. Такође, као закључак следи да ће неке верзије (или подсистеми), тј. инкременти, бити развијани у више итерација у односу на друге подсистеме.

### 3.2.5. *V* модел

Назив *V* модела представља акроним од валидације и верификације (*Validation* & *Verification*) и представља проширење „Водопад“ модела. Главна идеја овог приступа је давање додатног значаја фази тестирања, која се затим дели на више мањих фаза, при чему свака од тих мањих фаза тестирања има свој пандан у фазама пројектовања и развоја (Balaji, & Murugaiyan, 2012). Свака фаза се ослања на документацију из претходне фазе, што уз споменуто тестирање након сваке фазе развоја чини да документација буде ажурна, а производ благовремено тестиран и одобрен. Овим приступом постиже се виши ниво ефикасности, јер се грешке могу уочити и отклонити у раним фазама развоја.

Распоред фаза није линеаран као код „Водопад“ модела, већ се на дијаграму може приказати у облику латиничног слова „*V*“. Почев од фазе прикупљања захтева, преко спецификације, пројектовања до кодирања (фазе које би одговарале „Водопад“ моделу), фазе имају опадајући смер. Након кодирања следе фазе тестирања које имају узлазни смер. Разлика у почетним фазама развоја у односу на „Водопад“ модел је што се у фази пројектовања дефинишу архитектура система (*High-Level Design* - *HLD*) и софтверске компоненте (*Low-Level Design* - *LLD*). *HLD* и *LLD*, заједно са фазом прикупљања захтева (за целокупни систем) представљају фазе које имају пандане у фази тестирања, као што је већ споменуто. Одговарајуће фазе тестирања су (Balaji, & Murugaiyan, 2012):

- Тестирање програмске јединице (којом се тестира конкретна софтверска компонента, тј. ова фаза тестирања одговара *LLD* фази),
- Фаза Тестирања интеграције (којом се тестира интеграција различитих компоненти унутар архитектуре, тј. ова фаза тестирања одговара *HLD* фази),

- Фаза Тестирања система (у којој се тестира цео систем и ова фаза тестирања одговара фази прикупљања и анализе корисничких захтева за цели систем).

Како наводе споменути аутори, главна предност овог система представља могућност брзог уочавања грешака, чиме се смањују трошкови отклањања грешака и повећава ниво ефикасности. Још једна предност се огледа у чињеници да специјализација фазе тестирања на различите нивое општости омогућава лакше контролисање одговорности у фази тестирања. Као главни недостаци овог приступа наводе се они који важе и за „Водопад“ модел: ригидна структура, са малим простором за флексибилну промену захтева. Измене у захтевима по правилу доводе до потребе измене документације и спровођења нових циклуса тестирања (Balaji, & Murugaiyan, 2012).

### 3.2.6. Спирални модел

Основе Спиралног модела поставио је *Barry Boehm* крајем осамдесетих година прошлог века (Boehm, 1988). Спирални модел ја развијен у циљу увођења анализе ризика, новог аспекта који до ток тренутка није био заступљен ни у једном методолошком приступу за пројектовање информационих система. По концептима које обухвата, Спирални модел се може одредити и као јединствена комбинација прототипског, итеративног и „Водопад“ модела животног циклуса развоја система. Творац приступа дао је детаљну спецификацију фаза, чији опис следи у наставку. Прва фаза у овом приступу представља новину у односу на остале приступе и то је фаза анализе ризика. Приликом иницијалне анализе ризика, врши се идентификација циљева и захтева које систем треба да задовољи, као и анализа ограничења и алтернативних решења. Како творац приступа наводи, на основну претходно описаног, анализирају се и ризици у развоју, а излаз из ове фазе представља улаз за дефинисање плана развоја и тестирања кроз прототип (ређе се могу користити и симулациони модели). Након тога приступа се изради прототипа и од тог тренутка испољава се итеративно својство овог приступа. Иницијални прототип се тестира и за њега се врши анализа ризика. Сваком новом итерацијом прототип се пропушта кроз анализу ризика све до тренутка када прототип не постане довољно робустан да може да се отпочне наредна фаза у развоју (Boehm,

1988). Битно је нагласити да у свакој итерације, нови прототип пролази кроз анализу ризика. С обзиром на то да се сваком новом итерацијом повећава број прототипова, уколико би се овај модел представљао дијаграмом на *X* оси би се налазио број прототипова, тј. број верзија, на *Y* оси напредак, односно трошак (напредовање пројекта нужно условљава увећање трошка, без обзира о којем је односу реч), а линија која би приказивала напредак по итерацијама била би закривљена, у виду спирале, јер се константно повећава број верзија и трошак, све док се не дође до финалног прототипа (Boehm, 1988). Зато овај модел не гради затворену линију својим фазама, већ спиралу, по којој и носи назив. Након одобреног прототипа, прелази се на наредне фазе развоја, које се у потпуности уклапају у фазе „Водопад“ методолошког приступа. После прикупљања захтева и анализе приступа се фази пројектовања, затим имплементације, тестирања и одржавања.

У литератури се може пронаћи шест начела Спиралног модела (Boehm, 2000):

- Начело 1: За Спирални модел потребно је истовремено дефинисати и планирати кључне аспекте, попут концепата, захтева, плана пројекта, модела система и будућег програмског кода,
- Начело 2: Свака итерација у спирали сачињена је од утврђивања захтева, процене ризика, развоја и тестирања и планирања следеће итерације,
- Начело 3: Ниво постигнућа условљен је нивоом прихватљивог ризика,
- Начело 4: Степен детаљности решења условљен је нивоом прихватљивог ризика,
- Начело 5: На свакој контролној тачки учесници оцењују кључне аспекте (испуњени циљеви, коришћена архитектура и функционалности) и
- Начело 6: Процес развоја мора да се фокусира и на целокупан животни циклус организације (укључивање дугорочних аспеката интересовања).

Анализа ризика представља главну предност, али и недостатак у односу на друге приступе. Предност се огледа у могућности детаљне процене ризика и извођења превентивних измена на систему, ради предупређивања нежељених ситуација. Повећани број итерација развоја прототипа, које се спроводе пре почетка развоја самог система, могу негативно утицати на трошкове и потребно време реализације, што уједно представља и највећи недостатак овог приступа.

### 3.2.7. *RAD* модел

Први концепти *RAD* (*Rapid Application Development*) приступа појавили су се још седамдестих година прошлог века, док је прву конкретизацију приступа спровео *James Martin* (Isaias, & Issa, 2015) почетком деведесетих година прошлог века. Овај приступ проистекао је из потребе флексибилније организације процеса развоја система и бржег одговарања на промењљиве потребе организације, уз поштовање одређених стандарда. Као основни циљ било је постављено смањење времена потребног за развој целокупног система. Да би то постигао, *RAD* се дистанцирао од традиционалних приступа развоја, који проширују *SDLC* и задржавају принцип развоја пратећи сукцесивне кораке, већ је понудио еволутивни принцип развоја система кроз инкременте, при чему је сваком инкременту, на основу пословних потреба, додељен одговарајући степен важности (Isaias, & Issa, 2015). Сваки инкремент се развија по принципима спиралног модела. Зато се за овај приступ може рећи да у основи представља комбинацију инкременталног и спиралног модела. *RAD* садржи четири фазе: планирање захтева, пројектовање, имплементација и завршна фаза, при чему се у литератури могу наћи и примери *RAD* приступа са интегрисаном првом и другом фазом у једну (Agarwal, et al., 2000). Прва фаза се истиче као посебно битна, јер се поред анализе захтева, могућности и ограничења и дефинисања рокова, врши и анализа ризика, што такође представља принцип преузет из Спиралног модела.

Главне предности *RAD* приступа су скраћени временски циклус путем утврђивања приоритета инкремената, дефинисаних на основу пословних потреба, као и активно укључивање корисника од почетних фаза развоја. Главни изазов овог приступа представља утврђивање баланса између флексибилног процеса развоја и структурне стабилности система (Isaias, & Issa, 2015). Главни недостатак се огледа у смањењу важности фаза планирања и тестирања (у циљу бржег развоја), што може утицати на ниво квалитета целокупног система. Још један недостатак који се наводи је и могућност постављања неостваривих захтева и рокова од стране менаџмента, јер се на рачун техничког предност даје организационом аспекту развоја (Gottesdiener, 1995).

### 3.2.8. Агилни модел

Потреба дефинисања приступа који би на флексибилнији начин омогућио брз развој информационих система била је присутна деценијама уназад. Временом, јављали су се приступи који су били вођени идејом задовољења наведених потреба, а неки до примера су Спирални и *RAD* приступ. Ипак, споменути приступи су само делимично успели у тим настојањима, приказујући при томе одређене мане, које нису биле присутне код ранијих верзија *SDLC* модела (Спирални модел захтева дуже време за развој прототипа, док *RAD* непрактично умањује значај планирања и тестирања). Почетком двадесетог века, прецизније 2001. године, седамнаест програмера објавило је манифест Агилног приступа.

У манифесту Агилног приступа (Beck, et al., 2001), група аутора наводи 12 принципа развоја софтверског система, који представљају најбољу праксу агилног приступа и којих би пројектанти требало да се придржавају у циљу ефикасног развоја софтвера. Споменутих 12 принципа заснивају се на 4 постулата агилног развоја. Постулати представљају сучељене критеријуме који су уобичајени за процес развоја система. Анализом постулата, може се утврдити који критеријуми имају предност приликом агилног развоја. Ти критеријуми су (Beck, et al., 2001):

1) Активно ангажовање свих индивидуа укључених на пројекту и подстицање интеракције међу њима,
2) Исправни и функционални софтвер,
3) Сарадња са клијентом и
4) Прилагођавање и реаговање на промене.

Овим критеријумима се у директном поређењу даје већи значај од формално дефинисаних процеса и техника, стриктног држања плана пројекта, опширне документације и преговарања услова пројекта.

У литератури се може пронаћи доста варијација агилног приступа, попут екстремног програмирања и *SCRUM*-а, али се упркос томе заједничке фазе могу издвојити. Аутори (Isaias, & Issa, 2015) дефинишу следеће фазе:

1) Селекција пројекта и одобрење - садржи спецификацију захтева, функционалности и очекиваних циљева и анализу ризика. У овој фази се остварује интеракција између пројектаната, менаџера и корисника,

2) Покретање пројекта - за циљ има формирање тима и одређивање алата и архитектуре који ће бити коришћени. Такође, ова фаза остварује интеракцију између пројектаната, менаџера и корисника,
3) Пројектовање и пуштање пројекта - реализују се активности планирања и израде модела система. Ова фаза се реализује по инкрементима, па је од великог значаја правовремена интеракција свих интересних страна. Инкрементално се испуњавају захтеви, све док се не дође до система који обухвата све специфициране функционалности и
4) Тестирање и документовање – акценат је на тестирању целог система и изради документације, иако и фаза пројектовања садржи тестирање конкретних инкремената.

Главне предности агилног приступа представљају брз развој и брзо испуњавање стварних и променљивих потреба пословања, флексибилност развоја и задовољство корисника. Превише флексибилан приступ развоју може проузроковати занемаривање иницијалног плана и одступање од техника и метода развоја, који могу бити корисни. Поред тога недостатак представља и сувише либералан приступ свеобухватности и ажурности документације.

### 3.2.9. Структурни модел

Структурни модел животног циклуса развоја информационих система креиран је осамдесетих година прошлог века од стране *CCTA* агенције (*Central Computer and Telecommunications Agency*) за потребе Владе Велике Британије (Edwards, Thompson, & Smith, 1989). Структурни приступ развоја модела темељи се на традиционалном *SDLC* приступу, прецизније на „Водопад“ моделу, са посебним акцентовањем важности вођења темељне и ажурне документације којом се усмерава пројекат, за разлику од флексибилног креирања документације присутног у *RAD* приступу (Isaias, & Issa, 2015). Разлог његовог настанка био је да се „Водопад“ модел прошири још детаљнијим фазама анализе захтева и функционалности, планирања, као и изградњом пословног модела процеса (*Business Mode*l). Из „Водопад“ модела преузет је принцип сукцесивних фаза, при чему излаз из једне фазе представља улаз у наредну, док се документацији придаје

значај до мере да се наредна фаза не може започети, све док релевантне интересне стране не одобре документацију из претходне фазе (Rob, 2004).

Две најзначајније карактеристике овог приступа су да се сложеност система савладава функционалном декомпозицијом и да се користе методе и дијаграми погодни за приказ логичког модела и структуре система на интуитиван начин (Manteghi, & Jahromi, 2012). Централни ресурс структурног приступа представља пословни модел система, односно модела процеса, а као најпознатије и најчешће коришћене методе за његову израду користе се Структурна системска анализа и *IDEF0* (*Integration DEFinition for Function Modeling*) стандард америчке владе (Lazarević, et al., 2006). Модел процеса се посматра као скуп функција, различите сложености, које на основу улаза производе излаз. За структурни приступ се везује и концепт апстракције, па се функције које су опште од старта приказују, док се оне специфичније постепено уводе. Сложена функција се на нижем нивоу апстракције може описати специфичнијим функцијама, док функције које се даље не могу декомпоновати третирају се као примитивне.

Три најзначајније технике структурног приступа су (Avison, & Fitzgerald, 2006b):

- Моделовање базе података – идентификација, моделовање и документовање свих ентитета будућег система, као и веза које се могу успоставити између њих. Могу се користити различити модели, а најпознатији су МОВ (Модел Објекти-Везе) и *UML* дијаграм класа.
- Моделовање токова података – идентификација, моделовање и документовање токова података унутар система и њиховог односа са функцијама: утврђивање које функције представљају изворе токова података, а које поноре. Такође, евидентирају се и сва складишта података, тј. токови у мировању. Поред наведеног, води се евиденција и о интерфејсима, тј. о ентитетима изван система са којима се врши интеракција путем токова податка. За потребе моделовања токова података може се користити Структурна Системска Анализа.
- Моделовање понашање ентитета - идентификација, моделовање и документовање понашања система. За приказ се најчешће користе случајеви коришћења и дијаграми интеракције (дијаграми секвенци и комуникације).

Број и организација фаза структурног приступа се разликују у различитим радовима (Rob, 2004; Manteghi, & Jahromi, 2012), али су одступања без суштинског значаја. Фазе које се најчешће наводе су: студија изводљивости, анализа и спецификација захтева, израда логичког модела, израда физичког модела, имплементација и тестирање. Неки аутори развој унутар структурног приступа посматрају као континуирани процес, па наводе увођење и одржавање као природне фазе које следе процес иницијалног развоја (Isaias, & Issa, 2015).

Погодност примене структурног приступа се огледа у темељном планирању, које утиче да се пословне потребе детаљно анализирају и спецификују. Ипак, споро је прилагођавање пословним променама, уколико је развој одмакао од фазе планирања. Структурни приступ не захтева експерте за праћење реализације пројекта (не у мери у којој захтева агилни приступ), због стриктне примене правила. Већина корака структурног приступа је подржана *CASE* (*Computer Aided Software Engineering*) алатима. Поред тога, могу се издвојити и следеће погодности: одвајање логичког од физичког модела, што доприноси флексибилнијем процесу развоја, укључивање корисника од почетних фаза и добијање повратних информација, одобравање документације након сваког корака, утврђивање и отклањање пропуста у раним фазама развоја и потенцијална употреба резултата различитих фаза више пута (нпр. дијаграми токова података се могу користити и за неки наредни пројекат, уколико није дошло до реинжењеринга пословних процеса). Главни недостаци, поред спорог прилагођавања пословним променама, представљају могуће повећање утрошка времена и новца услед непотребно детаљних анализа и сложеност дијаграма, уколико се апстраховање не користи у адекватној мери. Детаљнији приказ предности и недостатака Структурног модела доступан је у литератури (Rob, 2004; Manteghi, & Jahromi, 2012; Isaias, & Issa, 2015).

### 3.2.10. Објектно-оријентисани приступ

Објектно-оријентисани (ОО) приступ животног циклуса развоја информационог система се заснива на итеративно-инкременталном приступу. Експанзија објектне програмске парадигме допринела је настанку објектно-оријентисаног приступа (Avison, & Fitzgerald, 2006b). Главна карактеристика овог приступа је да се систем

посматра као скуп објеката (Rumbaugh, Jacobson, & Booch, 2004). Објекат може бити материјални, нематеријални, особа или било који концепт из реалног света. Сваки објекат поседује скуп стања и скуп понашања. Преко скупа понашања утиче се на објекат, чиме се иницира промена стања.

Фазе развоја које су дефинисане овим приступом су (Dennis, Wixom, & Tegarden, 2015):

1) Покретање пројекта – садржи спровођење студије техничке, организационе и економске изводљивости и почетне активности анализе,
2) Елаборација – садржи процесе прикупљања захтева, наставак процеса анализе, процене ризика извођења, логичког пројектовања и комплетирања пословног модела,
3) Израда – углавном се фокусира на имплементацију испројектованог система из претходне фазе, као и евентуално додавање накнадних захтева и финализацију модела и
4) Транзиција на нови систем – фокус ове фазе је на тестирању, евентуалним корективним изменама на основу добијених резултата спроведених тестова, као и на увођењу новог система. Излаз из ове фазе, поред „живог система“, представљају и корисничка документација и план одржавања и унапређења система током процеса употребе.

Свака од фаза се састоји од активности које се могу понављати, како у самој фази, тако и у неким другим фазама. Примера рад, прикупљање захтева се реализује у прве две фазе, док је примарна реализација ове активности у првој фази. Логичко пројектовање се простире на другу и трећу фазу, док је примарна реализација ових активности у другој фази. Активности анализе се спроводе у првој и у другој фази итд. На основу наведеног може се уочити веза са традиционалним *SDLC* моделом. Иако су фазе ОО приступа другачије дефинисане у односу на традиционални *SDLC*, активности унутар сваке фазе у суштини одговарају фазама традиционалног *SDLC* модела. Такође, принцип инкременталног приступа је заступљен, јер свака фаза представља одговарајући инкремент.

ОО приступ користи *UML* (*Unified Modeling Language*) дијаграме као технику представљања различитих аспеката система, а најбитнија су следећа три (Rob, 2004):

1) Функционални аспект - као што и назив сугерише, описује начин функционисања система из угла корисника. Типично се користе описи и дијаграми случајева коришћења за ову фазу. Структурна декомпозиција се изоставља и то уједно представља и највећи недостатак ОО приступа,
2) Статички аспект - описује се класама, методама, атрибутима, везама и порукама и за приказ типично се користе *UML* дијаграми класа и
3) Динамички аспект - Дијаграми интеракције се користе за приказ динамичког аспекта система, исто као и у структурном приступу.

Предност коришћења овог приступа огледа се у увођењу концепата објектне парадигме, употреби стандардизованих и распрострањених *UML* дијаграма и обједињавању неколико других приступа (инкрементални и традиционални). Поред изостанка структурне декомпозиције недостаци су и потенцијална сложеност приступа, као и одсуство јасне границе између фаза анализе и пројектовања.

### 3.2.11. Прототипски модел

Прототип се може дефинисати као „пробни“ модел система, тј. иницијална верзија система која се прелиминарно прави ради представљања жељених концепата система и веза између њих, утврђивања степена испуњености корисничких захтева, анализе потенцијалних одступања и утврђивања начина њиховог отклањања (Isaias, & Issa, 2015). Прототип може бити посматран са засебним животним веком, или као етапа у моделу животног циклуса целокупног информационог система. Прототипски приступ се може дефинисати као брзи и итеративни приступ развоја информационог система, уз контролисане трошкове и укључивање корисника у рану фазу развоја кроз тестирање прототипа (Sommerville, 2001). Итеративност се огледа у чињеници да прототип представља прву верзију система, која се усавршава док не задовољи све прецизирани захтеве, док се основ за тврдњу да

омогућава брзи развој огледа у скраћеном временском циклусу представљања прве верзије система (прототипа) у односу на класичан *SDLC* или „Водопад“ модел.

Неки аутори (Isaias, & Issa, 2015) наводе да је, почев од осамдесетих година прошлог века, управо прототипски развој информационог система био окосница неколико других методолошких приступа (итеративног, спиралног а затим и агилног). *Carr* и *Verner* су крајем прошлог века закључили да су системи развијани путем прототипа динамичнији и да се лакше прилагођавају променљивим потребама корисника, што је и представљало главни разлог да ови аутори издвоје, систематизују и опишу прототипски развој као засебни методолошки приступ (Carr, & Verner, 1997).

Неопходан корак пре започињања развоја прототипа је прецизно дефинисање намене за коју се развија конкретан прототип. Приказивање изгледа и функционалности корисничког интерфејса, валидацију функционалних захтева система и презентовање менаџменту изводљивости развоја целокупног система могуће је постићи израдом више прототипова, при чему сваки од њих треба да има специфичну намену (Sommerville, 2001). Након ове фазе, споменути аутор дефинише и остале фазе овог приступа. Након прецизиране намене, следећа фаза је дефинисање опсега функционалности које прототип треба да садржи. Овде се као питање јавља балансирање броја функционалности и са друге стране времена и трошка потребних за развој прототипа. Како споменути аутор наводи, апстракција одређених делова система и изостављање функционалности прототипа неретко су опредмећени елиминацијом нефункционалних захтева (временом одзива или заузећем меморије). Након описане фазе, приступа се оцењивању прототипа. У зависности од резултата постигнутих у фази тестирања и повратних информација од корисника, систем се проглашава финалним или се покреће нова итерација измене и проширења система.

Постоје три типа прототипског развоја, која су развили и детаљно објаснили аутори (Carr, & Verner, 1997):

1) Истраживачки тип прототипског развоја - заснива се на премиси да се кориснички захтеви све темељније испитују у свакој новој итерацији развоја прототипа. Зато је један од основних циљева овог типа прототипског развоја да се нове верзије прототипа брзо објављују и тестирају. У том контексту

наведени аутори посматрају спирални модел као посебну варијанту истраживачког прототипског модела, где се прототипови повезују кроз узастопне фазе процеса, а које се у основи темеље на фазама „Водопад“ модела.

2) Експериментални тип прототипског развоја - акценат ставља на развој прототипа који се евалуира експерименталним коришћењем од стране корисника. Типичан пример овог приступа је симулациони модел система.
3) Еволутивни тип прототипског развоја - залаже се за формирање верзија прототипа и њихово проширење (додавања инкремента), кроз итерације, тј. овај приступ најприближнији је итеративно-инкременталном приступу. По овом типу, прототип има улогу приближавања система кориснику и представља улазну верзију система за следећу итерацију.

Главне предности употребе прототипског развоја огледају се у лакшем прилагођавању захтева корисника, смањењу ризика развоја, добијању повратне информације од корисника у раним фазама развоја, активнијем укључивању корисника, скраћеном циклусу развоја, повећаној доступности система и повећању шансе за успешном имплементацијом целокупног система, док главне недостатке представљају скраћене фазе анализе и планирања пројекта, мање ажурна документација у односу на традиционални *SDLC*, отежано проширење система функционалностима које су иницијално апстраховане из прототипа у циљу бржег развоја и коришћење финалног система на другачији начин од прототипа (Isaias, & Issa, 2015; Sommerville, 2001).

### 3.2.12. Модел „Непредвиђене ситуације“ (*Contigency* модел)

Овај приступ је настао као покушај креирања методолошког приступа који би омогућио одсуство стриктне примене свих аспеката приступа у свакој ситуацији. Већина приступа има за циљ дефинисање „идеалне ситуације“ у којој је приступ примењив и тежи да што већи опсег могућих ситуација сведе на „идеалну ситуацију“. Ситуација у којој се налази „жива“ организација зависи од њене комплексности и структуре, од типа и учесталости промена, структуре кадра, броја запослених који се суочавају са променом и њихових вештина прихватања

промене, што утиче да одступање од „идеалне ситуације“ (Avison, & Fitzgerald, 2006a). Таква конкретна ситуација се може назвати „непредвиђеном ситуацијом“. Као последица немогућности потпуног прилагођавања једног приступа свим ситуацијама, нити могућности прилагођавања сваке ситуације „идеалној ситуацији“ коју пропагира приступ долази до примене приступа у одређеном степену: неке фазе се изостављају, неке се продужавају, технике се прилагођавају ситуацији конкретне организације и слично. Као што и сам назив сугерише, модел „Непредвиђене ситуације“ дефинише опште смернице развоја, али су фазе, технике, методе, алати и сви остали аспекти прилагодљиви ситуацији: могу бити искоришћени, прилагођени конкретној ситуацији или у потпуности изостављени. На наведено утиче више фактора а неки од најважнијих су тип пројекта, величина пројекта, важност пројекта за организацију и планирано време завршетка пројекта (Avison, & Fitzgerald, 2006a). Како споменути аутори наводе предност овог приступа се огледа у флексибилнијој организацији и коришћењу ресурса и потпуно прилагођавање корака развоја конкретној организацији, док највећи недостатак представљају одсуство формализације фаза, потенцијално прескакање или непотребно скраћивање важних фаза, низак степен ажурности документације, као и висок ризик развоја.

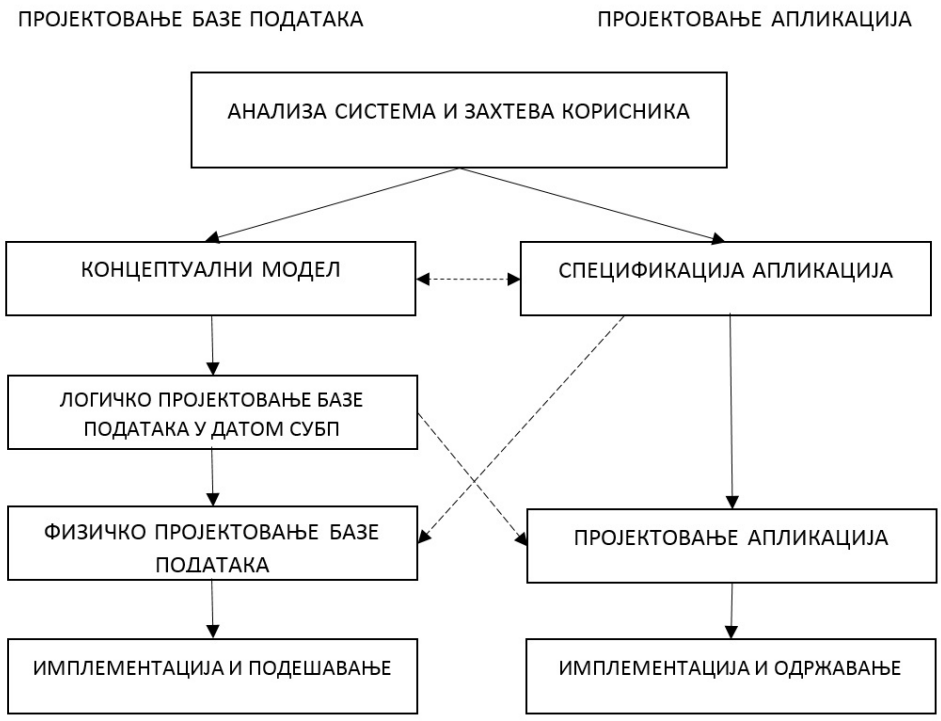
### 3.2.13. ФОН ЛабИС модел

ФОН ЛабИС методолошки приступ животног циклуса развоја софтвера се заснива на системско-теоријском приступу (Lazarević, & Nešković, 1997), који проширује увођењем поступка постепеног развоја система. Овај методолошки приступ, који је настао деведесетих година прошлог века у Лабораторији за информационе системе Факултета организационих наука, прописује употребу савремених техника и метода па представља и одличан основ за едукацију (Marjanović, Aničić, & Babarogić, 2011).

Осим системско-теоријског приступа, за ФОН ЛабИС приступ се такође може рећи да у себи садржи аспекте структурног и објектног приступа, али и других приступа, попут итеративно-инкременталног. Овај приступ, од системско-

теоријског је преузео три главне фазе развоја: идентификација система, реализација система и имплементација система. Шематски приказ фаза пројектовања у овом приступу дат је на Слици 1.

Са Слике 1 се може уочити да се пројектовање базе података и пројектовање апликација могу представити помоћу две гране. Анализа система и захтева корисника заједничка је и за пројектовање базе података и за пројектовање апликација, након чега следе специфичне фазе пројектовања базе података (израда концептуалног модела, логичко, физичко пројектовање базе података, имплементација и подешавање) и специфичне фазе пројектовања апликација (спецификација апликација, пројектовање апликација, имплементација и одржавање).




Слика 1 – Фазе у пројектовању базе података и апликација (Lazarević, et al., 2006)

Од посебног значаја за ову дисертацију су фазе пројектовања базе података ФОН ЛабИС приступа, ради примене на пројектовање *SQL* базе података (Lazarević, et al., 2006):

1) Анализа система и захтева корисника,
2) Израда концептуалног модела,
3) Логичко пројектовање,
4) Физичко пројектовање и
5) Имплементација и подешавање.

Почетна фаза анализе система и захтева корисника за циљ има развој пословног модел система, односно модел функција посматраног система. Методе које се по речима наведених аутора најчешће користе у овој фази су ССА (Структурна Системска Анализа), из скупа конвенционалних метода и пословни случајеви коришћења, из скупа објектних метода. ССА дијаграм највишег степена апстракције се назива дијаграм контекста и на њему се цео систем приказује као једна функција са одговарајућим улазима и излазима. Остали дијаграми се називају дијаграми токова података и њима се постиже декомпозиција и контролисани приказ сложености система. Увођења нових функција даљом декомпозицијом постојећих и побољшање већ идентификованих функција реализује се итеративно-инкрементално.

Прву фазу следе фазе израда концептуалног модела, логичко и након тога физичко пројектовање. Концептуални модел представља слику реалног система који се састоји од објеката, њихових атрибута и међусобних веза. Он представља целокупан садржај информационог система и независан је од технологије која ће бити коришћена за имплементацију. За израду концептуалног модела користе се модели података.

Примена модела податка, који неки аутори (Ponniah, 2007) третирају и као специфичну технику, омогућава да се на основу потребних захтева пословања, свим интересним странама, на разумљив начин представе концепти реалног система који ће бити коришћени у процесу пројектовања. С обзиром на то да у интересне стране у процесу пројектовања спадају крајњи корисници, за које је потребно развити интуитиван модел, али и пројектанти база података, који имају потребу за детаљном спецификацијом базе података која се пројектује, неретко се

користе различити модели података. Модели података који се најчешће користе у интеракцији са корисницима су МОВ, од конвенционалних и *UML* дијаграми класа, од објектно-оријентисаних (Lazarević, et al., 2006). Споменути модели имају графичке симболе па су погодни за приказ корисницима, за разлику од релационог модела, који је, са друге стране, од велике важности пројектантима база података.

Због значаја који концептуално моделовање има за целокупан процес пројектовања, аутори приступа су посебну пажњу посветили техникама применљивим у овој фази, а које су временом препознали и описали (Lazarević, et al., 2006):

- Интеграција подмодела – На основу пословног модела, изграђеног у претходној фази, врши се изградња подмодела података за сваку примитивну функцију. Сваки подмодел примитивне функције добија се интеграцијом токова и складишта, након чега се сви подмодели интегришу у јединствени модел целог система.
- Директно моделовање на основу вербалног описа система – За вербални опис система најчешће се користе описи сценарија препознатих случајева коришћења.
- Коришћење узора, тј. „патерна“, у пројектовању – У циљу елиминисања непотребног вишеструког решавања истог проблема, дефинишу се општа правила, тј. „патерни“, којима се формализује решење препознатог проблема. То решење се може примењивати сваки пут када се пројектант сусретне са истим проблемом.
- Нормализација релација – Представља поступак изградње релационог концептуалног модела, при чему свака релација задовољава одређену нормалну форму. Такође, релациони модел служи и за процес логичког пројектовања структуре базе података.
- Трансформација једног модела у други („директно“ и „инверзно“ инжењерство) – Трансформација каноничког (спецификованог) у имплементациони модел назива се „директно“ инжењерство, док обрнути процес представља „инверзно“ инжењерство.

Логичко пројектовање базе података за циљ има да концептуални модел преведе у логички. Циљ ове фазе пројектовања је да се концептуални модел преведе

у нормализовани релациони модел на основу којег се може изгенерисати шема *SQL* базе података. Примена релационог модела у фази логичког пројектовања дата је у раду (Batini, Ceri, & Navathe, 1992). Превођење је могуће спровести ручно, а могу се користити и *CASE* алати (попут *ErWin*-а или *Embracadera*). Уколико је коришћен нормализовани релациони модел, нема потребе за применом трансформације, већ се из релационог модела скуп релација и ограничења описује *DDL*-ом (*Data Definiton Language* – део *SQL*-а) конкретног система за управљање *SQL* базом података. Са друге стране, уколико је концептуални модел записан помоћу МОВ-а или *UML* дијаграм класа, они се прво преводе у релациони модел, при чему се примењују правила трансформације једног модела у други. На основу релационог модела се затим пишу или генеришу *DDL* наредбе, чиме је завршено логичко пројектовање (Lazarević, et al., 2006).

Физички модел *SQL* база података представља модел који садржи имплементационе специфичности. У овој фази се на основу логичког модела и нефункционалних спецификација врши пројектовање интерне, физичке структуре, која је прилагођена конкретном СУБП-у (Lazarević, et al., 2006). У овој фази се примењују и посебне технике физичког пројектовања:

- Пројектовање дистрибуције базе података – уколико се логичке структуре базе података физички дистрибуирају на различитим СУБП-овима, потребно је пројектовати дистрибуцију базе података, пре свега кроз хоризонталну и вертикалну сегментацију, као и репликацију.
- Денормализација представља технику контролисаног увођења редундансе података, у циљу смањења потребног времена извршавања упита, најчешће оних са интензивним спајањем. Представља „супротан“ поступак од нормализације. Нормализованим моделом се смањује утицај аномалија приликом уноса, ажурирања и брисања редундантних података, због чега техника денормализације има ограничену примену у *SQL* базама података. У *SQL* базама података денормализација је пожељна искључиво као контролисано одступање од нормализованог модела. Пример денормализације представља понављање атрибута `NAZIV_DOBAVLJACA` из релације `DOBAVLJAC` у релацији `FAKTURA`. Приликом извршавања упита којим се приказују назив добављача и подаци о његовим фактурама, наведеном

техником денормализације је елиминисана потреба спајања табела `DOBAVLJAC` и `FAKTURA`.

- Кластеровање представља технику којом се „сродни“ подаци распоређују да буду „физички“ блиски (нпр. у *Oracle*-у, подаци једне табеле да буду у истом блоку или подаци више табела у једном *extent*-у). Под сродним подацима се подразумевају подаци који се често приказују заједно у упитима. Пример представља распоређивање података из релације о неком типу документа и података из релације о ставкама тог типа документа, на физички блиску локацију.
- Техника дефинисања начина приступа подацима омогућава да се у зависности од учесталости приступа подацима одређене табеле или погледа *SQL* базе података дефинише и одговарајући начин приступа. Овом техником могуће је креирати индексе, *hash* функције и материјализоване погледе, у циљу оптимизације заузећа ресурса и времена приступа подацима. Пример може бити приказ дневних трансакција банке из скупа података о петогодишњим трансакцијама. Уз коришћење статистике приступа и осталих параметара може се доћи до закључка да, на пример, за посматрани упит није оптимално приступати целој табели, већ се може користити нека друга структура (нпр. индекс) и одређени начин приступа (функционално скенирање, скенирање по опсегу и слично).

Кроз ФОН ЛабИС приступ су наведене и описане неке од најчешће коришћених техника пројектовања *SQL* база података, које се могу наћи и у другим приступима. Наведени приступ је одабран као домаћин за опис карактеристичних техника пројектовања с обзиром на то да он у себи инкорпорира принципе различитих приступа, при чему се и наведене технике могу такође комбиновати са већим бројем већ описаних приступа. Упоредни приказ најважнијих предности и недостатака анализираних приступа дат је у Табели 1.

Иако технике денормализације и израде концептуалног модела имају своју примену и у приступима пројектовања *NoSQL* база података, при чему се чак денормализација чешће користи за *NoSQL* базе података, специфични аспекти пројектовања *NoSQL* база података условили су настанак и наменских техника и

зачетак нових приступа пројектовања *NoSQL* база података, који ће бити анализирани у наставку.

Табела 1 - Упоредни приказ методолошких приступа за пројектовање базе података

| Верзија/приступ | Предности | Недостаци |
|---|---|---|
| Водопад (*SDLC*) | - Увођење симулационог модела система (искорак у односу на конзервативни модел *SDLC* -а)<br>- Ангажованија улога корисника<br>- Инсистирање на ажурној документацији (највећи допринос)<br>- Прецизно планирање у раним фазама развоја | - Флексибилност (секвенцијално повезане фазе, онемогућено враћање на претходне фазе)<br>- Неадекватан одговор на брзо промењиве и текуће потребе организације (због дугог процес увођења промена у систем) |
| Инкрементални | - Бржи одговор на стварне потребе пословања (постигнуто брзим развојем нове верзије система)<br>- Ангажованија улога корисника<br>- Флексибилност (краће време потребно за добијање повратне информације од клијента) | - Број верзија система пре финалне може брзо нарасти и тиме драстично повећати трошкове<br>- Компатибилност различитих верзија и потенцијални проблем њиховог повезивања |
| Итеративни | - Пројектовање целог система одједном, па увођење побољшања | - Израда почетне верзије система може дуго трајати и повећати трошак |
| Итеративно-инкрементални | - Комбинује итеративни и инкрементални приступ<br>- Омогућен паралелизам у развоју | - Различити делови система се налазе у различитим фазама развоја, што може довести до дисбаланса |
| Прототипски | - Лакше прилагођавање захтевима корисника<br>- Смањење ризика развоја (добијање повратне информације од корисника у раним фазама развоја)<br>- Активније укључивање корисника<br>- Скраћен циклус развоја<br>- Доступност система (док се тестирају одређене специфичне функционалности, могу се развијати друге) | - Неажурна документација<br>- Кратка анализа и планирање пројекта<br>- Потенцијално високи трошкови<br>- Отежано проширење иницијалног система |
| „Непредвиђене ситуације“ | - Флексибилна организација и коришћење ресурса<br>- Потпуно прилагођавање корака развоја конкретној организацији | - Одсуство формализације фаза<br>- Потенцијално прескакање или непотребно скраћивање важних фаза развоја<br>- Низак степен ажурности документације<br>- Висок ризик развоја |
| Спирални | - Детаљна процена ризика и могућност отклањања потенцијалних пропуста система у раној фази развоја<br>- Предности итеративног, прототипског и „Водопад“ модела | - Потенцијални трошак времена и новца услед спровођења више итерација развоја прототипа, пре почетка развоја самог система<br>- Није погодан за мање пројекте |

| | | |
|---|---|---|
| *V* модел | - Брзо уочавање грешака, чиме се и смањују трошкови отклањања грешака и повећава ниво ефикасности<br>- Различити нивои општости омогућавају лакше контролисање одговорности у фази тестирања | - Недостаци као и за „Водопад“ модел: ригидна структура, са малим простором за флексибилну промену захтева<br>- Измене у захтевима по правилу доводе до потребе измене документације и спровођења нових циклуса тестирања |
| *RAD* | - Скраћење временског циклуса, утврђивањем приоритета инкремената на основу пословних потреба<br>- Брже одговарање на потребе корисника<br>- Активно укључивање корисника од почетних фаза | - Баланс између флексибилног процеса развоја и структурне стабилности система<br>- Смањење важности фаза планирања и тестирања<br>- Ажурност документације<br>- Постављање неостваривих захтева и рокова од стране менаџмента |
| Агилни | - Брз развој и брзо испуњавање стварних, променљивих, потреба пословања<br>- Флексибилност развоја<br>- Задовољство корисника | - Превише флексибилан приступ развоју може проузроковати занемаривање иницијалног плана и одступање од техника и метода развоја<br>- Либералан приступ свеобухватности и ажурности документације |
| Структурни | - Темељан приступ планирању утиче да се пословне потребе детаљно анализирају и спецификују<br>- Кораци подржани *CASE* алатима<br>- Одвајање логичког од физичког модела<br>- Укључивање корисника од почетних фаза<br>- Одобравање документације након сваког корака<br>- Утврђивање и отклањање пропуста у раним фазама развоја<br>- Потенцијална употреба резултата различитих фаза више пута | - Споро прилагођавање пословним променама<br>- Могуће повећање утрошка времена и новца услед непотребно детаљних анализа<br>- Потенцијална сложеност дијаграма, уколико се апстраховање не користи у адекватној мери |
| Објектни | - Увођење концепата објектне парадигме<br>- Употреба стандардизованих *UML* дијаграма<br>- Обједињавање неколико других приступа (инкрементални и традиционални) | - Потенцијално велика сложеност приступа<br>- Одсуство јасне границе између фаза анализе и пројектовања |
| ФОН ЛабИС | - Користи концепте структурног, објектног и итеративно-инкременталног приступа<br>- Увођење постепеног развоја софтвера<br>- Прецизно и детаљно дефинисане фазе, са препорученим техникама | - Иако је скраћена брзина реаговања на промене пословања, споријн развој од агилних приступа<br>- Приликом сваке измене потребно је ажурирати документацију, што захтева време |

## 3.3. Приступи и технике пројектовања *NoSQL* база података

Процес пројектовања *NoSQL* база података зависи од типа *NoSQL* базе података која се пројектује (кључ-вредност, документ, фамилија колона или граф) као и од репрезентативних упита прикупљених пре самог пројектовања. Наведено утиче на одабир најпогоднијих техника пројектовања. Уопштено говорећи, идентификоване фазе пројектовања *SQL* база података и модели који се при томе дефинишу (концептуални, логички и физички) нису увек јасно препознатљиви приликом пројектовања *NoSQL* базе података. Формално дефинисана шема, као и неопходност задовољења *ACID* особина, додатно удаљавају постојеће приступе пројектовања *SQL* база података од примене у процесу пројектовања *NoSQL* база података. Са друге стране, концепт података флексибилне структуре, наглашена потреба агрегирања података и пружање хоризонталне скалабилности захтевају посебну пажњу у процесу пројектовања *NoSQL* база података (Sadalage, & Fowler, 2012). Наведено је отворило простор за развој наменских приступа и техника пројектовања *NoSQL* база података.

Иако су *NoSQL* базе података присутне у употреби већ дуже од десет година, тек је последњих неколико година посвећена пажња развоју наменских методолошких приступа за пројектовање *NoSQL* база података. С обзиром на то да су приступи пројектовања *NoSQL* база података тек у повоју, постоји сагласност међу ауторима о неопходности креирања свеобухватног методолошког приступа примењивог на пројектовање свих типова *NoSQL* база података (Atzeni, Bugiotti, & Rossi, 2014; Badia, & Lemire, 2011; da Silva, 2011).

Међу покушајима формализације приступа пројектовања *NoSQL* база података, примењивог на све типове *NoSQL*-а, истиче се по својој свеобухватности методолошки приступ *NoAM* (*NoSQL Abstract Model*) (Bugiotti, et al., 2014). У неколико радова (Bugiotti, et al., 2014; Bugiotti, & Cabibbo, 2013a; Bugiotti, & Cabibbo, 2013b) изложени су постулати *NoAM* приступа. Развоју униформног приступа пројектовања свих типова *NoSQL* база података аутори су приступили увођењем апстракција, чиме су желели да превазиђу ограничавајући фактор у виду типа *NoSQL* базе података за коју се примењује *NoAM* приступ. Творци *NoAM*

приступа идентификовали су пет општих фаза приликом пројектовања *NoSQL* база података (Bugiotti, et al., 2014):

1) Израда концептуалног модела, за представљање релевантних ентитета система и веза које се могу успоставити међу њима,
2) Груписање повезаних података у агрегације,
3) Партиционисање агрегација, фаза којом се агрегације деле на мање (тј. елементарне) податке,
4) Пројектовање *NoSQL* база података највишег нивоа апстракције, где се на основу партиција врши мапирање агрегација у апстрактни *NoAM* међу-модел (*NoAM Intermediate Model*) и
5) Имплементација, фаза у којој се *NoAM* међу-модел мапира у специфичан модел конкретне *NoSQL* базе података.

*NoAM* приступ пружа спону са општим приступима пројектовања, а она се огледа у креирању концептуалног модела. У последње време јављају се и примери примене *UML* дијаграма за потребе израде концептуалног модела *NoSQL* база података (Shin, Hwang, & Jung, 2017). Након израде концептуалног модела приступа се груписању ентитета у агрегиране ентитете, што представља једну од уобичајених техника пројектовања *NoSQL* база података. Агрегирани ентитети садрже у себи концепт пара кључ-вредност, јер сваки агрегирани ентитет има јединствени идентификатор. Разлог формирања агрегираних ентитета је у томе што они представљају међусобно повезане податке, груписане на начин да престављају логичку јединицу приступа и обраде података од стране апликације. Након ове фазе врши се партиционисање агрегираних ентитета на мање елементе, релевантне за приказ резултата често извршаваних упита. Партиционисање се спроводи у циљу оптимизације перформанси. Да би се започела ова фаза потребно је прикупити релевантне представнике упита који се извршавају над *NoSQL* базом података, што представља новину у односу на процес иницијалног пројектовања *SQL* базе података. Услови који су потребни да буду задовољени да би дошло до партиционисања могу се свести на ситуацију када упити или нека функција фреквентно припада само делу агрегираног ентитета, при чему је потребно испунити да подаци којима се често приступа заједно буду унутар једно ентитета. Смештањем података којима се често истовремено приступа у исти ентитет може се повући аналогија са описаном техником кластеровања *SQL* база података. С обзиром на то да *NoAM* међу-модел представља спону између агрегираних података

апликације и података који се налазе у *NoSQL* бази података, превођењем агрегираних података у апстрактни *NoAM* међу-модел врши се припрема за складиштење података у *NoSQL* базу података. Овим је омогућено да све до последње фазе, систем ради са подацима који нису зависни од конкретне *NoSQL* базе података. У последњој фази (имплементације) апстрактни *NoAM* међу-модел се мапира у модел конкретне *NoSQL* базе података, уважавајући при томе све специфичности складиштења и управљања подацима одредишне базе података. Иако креатори приступа наводе могућност примене приступа и на друге типове *NoSQL* база података, приступ је демонстриран над представником кључ-вредност база података, конкретно коришћен је *Oracle KV*. Аутори (de Lima, & dos Santos Mello, 2015) су *NoAM* приступ имплементирали и над *NoSQL* базом података заснованом на документу.

Разноврсност модела података које користе *NoSQL* базе података проузроковала је њихову поделу управо по критеријуму модела на којем заснивају. Изузев граф база, разлике између осталих модела које користе, иако присутне, нису увек јасно разграничене. То се може објаснити чињеницом да фамилија колона и документ базе података представљају деривате кључ-вредност база података. зато ни не чуди што постоји неслагање аутора око категоризације неких конкретних СУБП-ова, попут *SimpleDB*-а. Иако присутан већ годинама *SimpleDB* је у неким радовима сврстан у категорију документ база (Cattell, 2010), у другима у категорију база заснованих на фамилији колона (Grolinger, et al., 2013), док се може наћи податак и да припада базама података заснованих на кључ-вредност пару (SolidIT, 2018).

Значајна особина *NoSQL* база података представљена је извршавањем упита над њеним подацима. Тек извршавањем упита над постојећим подацима, прикупљањем и анализом тих извршавања може се стећи слика о правим потребама корисника. То представља суштинску разлику у односу на *SQL* базе података, за које је приликом пројектовања потребно специфицирати нормализовани модел. Идентификацијом и праћењем извршавања релевантних упита (упити који се често извршавају, имају велики утицај на заузеће ресурса, на перформансе и слично) *SQL* база података, прихватљиво је на контролисани начин увести денормализацију. Код *NoSQL* база података се не примењује нормализација, па је одсуством исте

денормализација постала уобичајена појава. Да би се извршила денормализација на адекватан начин неопходно је познавање упита који се извршавају, пре него што се започне процес пројектовања. Методолошки приступи пројектовања *NoSQL* база података се још увек развијају. Поред приказаног приступа, временом су се издвојиле и технике пројектовања, чијом комбинацијом је могуће направити приступ прилагођен конкретним потребама и конкретном типу *NoSQL* базе података (нпр. техника хијерархијског моделовања се пре свега користи код граф база). У наставку је дат приказ најчешће коришћених техника у пројектовању *NoSQL* база података.

Најобухватнију систематизацију и преглед расположивих техника за пројектовање *NoSQL* базе података дао је аутор (Katsov, 2012):

- Техника денормализације (*Denormalization*) – Као што је већ наведено код пројектовања *SQL* база података: представља контролисано увођење редудансе, ради побољшања перформанси извршавања упита којима се приказују подаци из више ентитета. За разлику од *SQL* база података код којих се само по потреби примењује у фази физичког пројектовања, денормализација код *NoSQL* база података представља основну технику пројектовања. Већина других техника је проширује. Међу најважније предности примене ове технике су смањење броја читања и писања (услед груписања података на „истом“ месту) и брже извршавање сложених упита (на рачун смањеног броја спајања).

- Техника агрегације (*Aggregates*) – Омогућава да се подаци из више ентитета обједине у један агрегирани ентитет у којем је одређени део структуре исти за сва појављивања тог ентитета, док се други део структуре разликује у зависности од конкретног појављивања ентитета. Најважније предности примене агрегације су смањење броја веза између ентитета (самим тим и смањење броја спајања) увођењем угњеждених ентитета и једноставније моделовање хетерогених система креирањем мањег броја агрегираних ентитета промењљиве структуре. Агрегација се најчешће користи у комбинацији са техником денормализације. Пример може бити ентитет `OSOBA` која садржи уобичајене податке о особи, попут `JMBG`, `IME`, `PREZIME` и

*DATUM_RODJENJA*. Поред тога, у зависности од врсте занимања којом се бави, особа може имати специфичне атрибуте попут *SPECIJALNOST* и *DATUM_SPECIJALIZACIJE* за лекара, *TIP_PRAVA* и *LISTA_PRESUDA* за правника *NAUCNA_OBLAST* и *DATUM_IZBORA* за професора. Применом агрегације могуће је направити један агрегирани ентитет особа (и тиме редуковати засебне ентитете лекар, правник и професор). За свако појављивање заједнички атрибути би били *JMBG*, *IME*, *PREZIME* и *DATUM_RODJENJA*, док би се за лекара приказивали и *SPECIJALNOST* и *DATUM_SPECIJALIZACIJE*. По истом принципу, у појављивањима правника и професора поред заједничких били би приказани и по два пропертија специфична за сваког од њих.

- Техника спајања у апликацијама (*Application Side Joins*) –Приликом пројектовања *NoSQL* база података посебну пажњу би требало обратити на могућа спајања ентитета, која неретко имају негативан утицај на перформансе извршавања упита. Предупређивање спајања више ентитета није увек могуће спровести денормализацијом и агрегацијом, те се у тим ситуацијама прибегава имплементацији логике спајања у коду апликације. Везе за које су обе горње границе кардиналности „више“, као и фреквентно ажурирање дела агрегираног ентитета у *NoSQL*-у су најчешће ситуације имплементације спајања у апликацијском коду. Примена ове технике биће објашњена на примеру чувања историје промене цене производа. Уколико би се цена моделовала као део ентитета *PROIZVOD*, честа промена вредности атрибута цене директно би утицала на фреквентно ажурирање ентитета *PROIZVOD* додавањем нове цене, као угњеждене карактеристике производа. Уколико би се структура поједноставила, чувањем једне вредности цене производа, не би било могуће пратити њену историју промене. Настала ситуација се у пракси може решити увођењем засебном ентитета *CENA*, који би се у апликацијском коду спајао са производом којем припада.

- Техника атомске агрегације (*Atomic Aggregates*) – Ова техника служи за премошћивање одсуства *ACID* особина трансакција, које произилази из природних карактеристика *NoSQL* база података (изузимајући граф базе

података). Ипак, у одређеним ситуацијама је потребно задовољити неке од *ACID* особина и један од начина да се то реализује је употребом технике атомске агрегације. Атомска агрегација омогућава ажурирање једног ентитета у трансакцији, уместо више њих, чиме се постиже атомност на нивоу агрегације, а самим тим и на нивоу трансакције која се извршава само над једним агрегираним ентитетом. Примера ради, уколико је потребно водити евиденцију о *e-mail* адреси и броју телефона корисника, под претпоставком да се ти подаци често ажурирају, могуће их је сврстати унутар агрегираног ентитета који ће садржати све контакт податке. Тиме, уместо ажурирања више ентитета ажурира се један, чиме је задовољена и атомност на нивоу агрегираног ентитета и на нивоу трансакције.

- Техника нумерације кључева (*Enumerable Keys*) – Ова техника се примењује за кључ-вредност базе података, посебно код оних које немају уређене вредности кључева. Приликом уноса вредности кључа може се користити *hash* функција или секвенца. Сви кључеви који имају „приближну“ изгенерисану вредност *hash* функције смештају се у исту категорију индексиране листе вредности. Овом техником се убрзава претрага кључева, јер није потребно претражити целокупан опсег вредности, већ у зависности од тражене вредности кључа претражује се конкретна категорија којој кључ припада. Уколико се користи секвенца, могуће је лакше управљати вредностима које се генеришу, а на основу семантике тих вредности лакше извршити претрагу. На пример, за потребе неког система се на крају радног дана креира дневни извештај пословања, при чему број 1 одговара инкременту секвенце која се користи за генерисање вредности примарног кључа сваког новог извештаја. За приказ извештаја који су креирани у претходних недељу дана од конкретног датума довољно је селектовати оне извештаје који имају за 1, 2, 3, 4, 5, 6 и 7 мању вредност кључа од вредности кључа извештаја изабраног датума.

- Техника индексне табеле (*Index Table*) – У контексту чувања поновљених података изворне табеле у засебној и наменској структури, индексне табеле су сличне материјализованим погледима. Техником индексне табеле

омогућено је имплементирање логике извршавања индекса у базама података које не подржавају индексе. Основни принцип је да се поред главне табеле са подацима креира помоћна, тј. индексна табела. У њу се смештају вредности атрибута главне табеле које је потребно индексирати. За сваки приступ конкретној вредности атрибута индексне табеле додељује се вредност примарног кључа из главне табеле. Затим, уколико је потребно приказати кључеве свих записа главне табеле који се филтрирају по вредностима атрибута из индексне табеле, уместо претраге целе главне табеле, довољно је приступити индексној табели. Тиме је обезбеђен ефекат бржег читања података из засебне структуре у односу на главну табелу, тј. постигнут је ефекат индексирања. Са променом у главној табели захтевано је и ажурирање индексне табеле, ради чувања тачности података индексиране табеле. За пример може да послужи табела запослених која као атрибут има назив одељења запосленог. Уколико би се претраживали запослени по називу одељења без примене технике индексне табеле, било би потребно приступити табели запослени и претражити све редове. Уместо тога, креирана је индексна табела, која садржи све вредности назива одељења. За сваку вредност одељења памте се сви кључеви запослених, па претрагом по називу одељења подаци се добијају приступом конкретном реду индексне табеле.

- Техника димензионе редукције (*Dimensionality Reduction*) – Ова техника омогућава да се мулти-димензионални подаци преведу у неки модел *NoSQL* базе података, попут кључ-вредност модела. Димензиона редукција се углавном примењује за географске информационе системе и пример ове технике је *geo-hash* функција. Алтернатива овој техници представља превођење мулти-димензионалних података на скуп дводимензионалних, који се затим деле на мање дијаграме, са којих се „читају“ линије, додељује им се математичка апроксимација, а затим се добијене нумеричке вредности смештају у базу података.

- Техника сложеног примарног кључа (*Composite Key Index*) - Сложени примарни кључеви се неретко користе код пројектовања *NoSQL* база

података, а своју корисност посебно могу демонстрирати код сортираних кључ-вредност база података. Ова техника омогућава да се креира примарни кључ који има више од једног поља. Потреба за оваквом конструкцијом примарног кључа је иста као и код *SQL* база података и користи се пре свега када појављивање једног слога идентификационо и егзистенцијално зависи од појављивања неке друге релације. Типичан пример би био документ са листом својих ставки или запослени са листом исплаћених примања. Ова техника се користи и за креирање мулти-димензионих индекса. За разлику од претходно описане индекс табеле која је садржала један атрибут од значаја, табела са мулти-димензионалним индексима садржи два или више атрибута. Свака јединствена комбинација свих релевантних атрибута представља једно појављивање унутар табеле са мулти-димензионалним индексима.

- Техника агрегације са сложеним кључевима (*Aggregation with Composite Keys*) - Ова техника је уско повезана са претходном и најчешће се користе заједно. Сложени кључеви сортираних кључ-вредност база података омогућавају да се лако изврши агрегирање редова по делу кључа, нпр. да би се приказале све исплате одређеног запосленог потребно је фиксирати први део примарног кључа и приказати све записе који задовољавају услов. С обзиром на то да су слогови сортирани по кључу, све исплате једног запосленог приказују се једна за другом у табели. Измена агрегираних података захтева додатно време и ресурсе у односу на измене појединачних записа.

- Техника инверзна претрага – директна агрегација (*Inverted Search – Direct Aggregation*) - Ова техника представља проширење индекс табеле и комбинује се са њом. Након креирања индекс табеле, објашњене мало раније у раду, поставља се питање да ли је таква структура довољно ефикасна, када постоји велики број дискретних вредности референтног атрибута? У таквом сценарију, приступ индекс табели почиње да личи на приступ целој табели, а уколико је потребно приказати податке почетне, главне табеле, који нису у индекс табели целокупно индексирање губи на значају. У наведеним

сценаријима, да би се побољшале перформансе, поред индекс табеле (инверзне претраге) уводи се и директна агрегација по атрибуту главне табеле који се евидентира за свако појављивање референтног атрибута индекс табеле. Уколико би пример из индекс табеле био проширен, могао би да послужи и за ову технику. Нека главна табела садржи податке о запосленима на пројекту: *PROJEKAT_ID*, *ZAPOSLENI_ID*, *NAUCNO_ZVANJE*, *NAZIV_ODELJENJA* и *ZAVRSENI_FAKULTET*. Индекс табела ће садржати атрибут *NAZIV_ODELJENJA*, за чије вредности се памте кључеви запослених који раде у њему. Поред индекс табеле, креирана је и табела директне агрегације. Она садржи дискретне вредности кључева запослених, којима су додељени остали подаци о запосленом (*NAUCNO_ZVANJE*, *NAZIV_ODELJENJA* и *ZAVRSENI_FAKULTET*). У примеру, потребно је приказати све податке о свим запосленима из конкретног одељења. За приказ свих запослених из једног одељења много ефикаснији приступ је путем индекс табеле, него путем главне табеле. Ипак, уколико постоји велики број одељења и потребно је приказати и остале атрибуте запосленог осим кључа (*NAUCNO_ZVANJE* и *ZAVRSENI_FAKULTET*) индекс табела није довољно ефикасна. Тада се користи табела директне агрегације, да би се на основу кључа запосленог приступило конкретном запису и да би се приказали сви подаци о том запосленом. На овај начин постижу се боље перформансе, јер се уместо приступа главној табели, која садржи податке о свим запосленима на свим пројектима (табела током времена може нарасти у обиму, јер сваки запослени може бити ангажован на „више“ пројеката) приступа табели директне агрегације (која је мањег обима од главне табеле).

- Техника стабла агрегације (*Tree Aggregation*) - Ова техника се заснива на принципу да се стабло или произвољан граф моделују као засебан слог или документ (у зависности који типа *NoSQL* базе података се пројектује). Највећа корист ове технике се остварује када се стаблу или графу приступа само једном. Типичан пример је стабло које се може креирати од коментара на некој *веб* страници. Структура може бити разграната, при чему коментари могу имати одговоре, што се моделује као угњеждени део стабла. Уколико је потребно приказати све коментаре, ова техника је корисна, али

уколико је потребно сложену структуру често ажурирати, ова техника по правилу има лош утицај на перформансе.

- Техника повезивања суседа (*Adjacency Lists*) – овом техником се граф моделује као скуп слогова, тј. чворова, при чему је сваки од њих повезан са листом предака и листом потомака (изузетак представљају корени елемент који нема претка и лист елемент, који нема потомке). Претраживање се може вршити и по кључу директних предака или потомака, мада је овакав тип графа неефикасан када је потребно брзо приказати целокупно подстабло и тада је боље користити технику стабла агрегације.

Поред последње две технике, постоји још неколико техника за дефинисање хијерархије: материјализоване путање и угњеждене хијерархије, али због њихове ређе заступљености у односу на презентоване технике, неће бити детаљније анализиране.

Разлог за мању бројност и развијеност методолошких приступа пројектовања *NoSQL* база података проналази се у чињеници да су *NoSQL* базе података добиле на својој популарности тек последњих десетак година.

Заједничко за обе групе методолошких приступа је да су примењиви за пројектовање само *SQL* или само *NoSQL* база података. Пошто не укључују специфичности пројектовања база података оба типа смањује се и могућност њиховог избора као универзалног приступа пројектовања базе података која у својим компонентама садржи и *SQL* или *NoSQL* базе података, тј. хибридне *SQL*/*NoSQL* базе података. За потребе развоја и коришћења такве хибридне *SQL*/*NoSQL* базе података, по анализираној и тренутно доступној литератури, не постоје свеобухватни приступ пројектовања, већ постоје покушаји интеграције *SQL* и *NoSQL* база података применом различитих делимично примењив приступа и техника, који су приказани у овом поглављу. Наведено је отворило простор за развој новог методолошког приступа пројектовања хибридне *SQL*/*NoSQL* базе података, који је приказан и детаљно објашњен у наредном поглављу дисертације.

# 4. Методолошки приступ за пројектовање хибридне *SQL/NoSQL* базе података

Претходно описани и анализирани приступи пројектовања намењени су конкретном типу базе података (*SQL* или *NoSQL* базе података). Разлике у специфичностима описаних типова база података проузроковале су да ови приступи, иако у одређеној мери садрже преклапање у фазама, техникама и смерницама пројектовања, ипак нису директно примењиви и на *SQL* и на *NoSQL* базе података. Због чињенице да су методолошки приступи пројектовања *NoSQL* база података тек у зачетку, као и због описане оправданости пројектовања хибридне *SQL*/*NoSQL* базе података, која би у зависности од корисничких захтева садржала најбоље из оба света, очигледна је потреба развоја приступа који би задовољио потребе пројектовања како *SQL*, тако и *NoSQL* компоненте хибридне *SQL*/*NoSQL* базе података. Од приступа за пројектовање хибридне *SQL*/*NoSQL* базе података је очекивано да уважи све специфичности пројектовања сваке њене компоненте (*SQL* и *NoSQL* базе података). Новоразвијени приступ за пројектовање хибридне *SQL*/*NoSQL* базе података са пратећим активностима и резултати постигнути приликом тестирања на изабраном примеру из праксе потврђени су у раду (Bjeladinovic, 2018). У наставку ће бити приказани наведени приступ и постигнути резултати.

Међу главним предностима новоразвијеног приступа за пројектовање хибридне базе података су флексибилност и свеобухватност, које се огледају у чињеници да је нови приступ примењив и на базе података које се тек пројектују и на постојеће базе података, за које се по потреби може спровести процес редизајна и трансформације у хибридну базу података. За постојеће базе података врши се анализа актуелних захтева које база података треба да задовољи и по потреби, на основу изабраних критеријума, може се донети одлука о увођењу представника новог типа базе података и интеграцији са постојећом базом у хибридну базу података. У том случају, базе података одређеног типа чине одговарајућу компоненту хибридне *SQL*/*NoSQL* базе података (тј. постоје *SQL* и *NoSQL* компоненте). У процесу преласка на хибридну базу података утврђују се кандидати

за прелазак у другу компоненту хибридне базе података. Додатна флексибилност овог приступа се огледа у томе што су дефинисани критеријуми за прелазак на хибридну базу података, уз подршку задржавања постојеће базе података, као и пројектовања нове базе података као традиционалне *SQL* базе података, када за тиме постоји потреба. Узимајући наведено у обзир, приликом примене новог приступа за пројектовање хибридне *SQL*/*NoSQL* базе података, предложеног у дисертацији, могући су следећи исходи:

1) Спроводи се анализа оправданости редизајна постојеће *SQL* базе података (по критеријуму структурираности података у употреби) и по потреби спроводи се сам процес редизајна и преласка на хибридну *SQL*/*NoSQL* базу података. Да би се реализовала ова могућност, потребно је да већ постоји *SQL* база података, која ће у процесу редизајна бити третирана као иницијална, почетна база података,
2) Спроводи се анализа оправданости задржавања постојеће *NoSQL* базе података и по потреби спроводи се сам процес редизајна и преласка на хибридну *SQL/NoSQL* базу података. Да би се овај процес реализовао, развија се прототипска *SQL* база података, након чега се процес пројектовања наставља као у првом случају. Уколико анализа оправданости покаже потребу коришћења *NoSQL* компоненте унутар хибридне базе података, у ту сврху се може користити постојећа *NoSQL* база података (или неки њен део) и
3) Не постоји база података која би била коришћена и због тога се пројектује нова прототипска *SQL* база података, након чега се процес пројектовања наставља као у првом случају. Резултат тог процеса, под одређеним условима, може бити хибридна *SQL/NoSQL* база података.

Због доминације процеса редизајна постојећих информационих система (и њихових база података) и куповине готових *ERP* (*Enterprise Resource Planning*) система у односу на развој целокупних информационих система од нуле (Panorama, 2016), као и због тренутно вишег степена заступљености *SQL* база података у употреби у односу на *NoSQL* базе података (SolidIT, 2018) у овој дисертацији суштина новог приступа биће приказана пре свега кроз процес редизајна постојеће *SQL* базе података и преласка на хибридну *SQL*/*NoSQL* базу података.

Пројектовање хибридне *SQL*/*NoSQL* базе података од нуле се своди на процес редизајна прототипске *SQL* базе података, при чему су специфичне активности такође приказане и објашњене у наставку.

## 4.1. Концепти новог приступа за пројектовање хибридне *SQL*/*NoSQL* базе података

Као што је већ наведено, устаљена пракса је да приступи пројектовању комбинују различите аспекте других приступа, чиме се задржавају проверени концепти постојећих приступа, уз додавање одговарајућих проширења и унапређења карактеристичних за тај конкретни приступ. На Слици 2 шематски су приказани нови приступ пројектовања хибридне *SQL*/*NoSQL* базе података и аспекти преко којих је нови приступ повезан са постојећим приступима.

Нови приступ третира хибридну *SQL*/*NoSQL* базу података као јединствену логичку базу података, која садржи *SQL* и *NoSQL* компоненту. С обзиром на специфичности процеса пројектовања сваке од наведених компоненти, важна корелација се остварује са приступима пројектовања како *SQL* тако и *NoSQL* базе података. За пројектовање *SQL* базе података изабран је *Chen*-ов *Entity-Relationship* (Објекти-везе) модел (Chen, 1976), док је за пројектовање *NoSQL* базе података искоришћен *NoAM* модел (Bugiotti et al. 2014). Модел Објекти-везе је изабран јер је у пракси најпопуларнији и најчешће коришћен модел за пројектовање *SQL* база података (Lazarević, et al., 2006), док је *NoAM* модел изабран као најобухватнији од приказаних модела за пројектовање *NoSQL* база података.

Слика 2 приказује фазе пројектовања *SQL* базе података: креирање концептуалног, логичког и физичког модела, након чега следи имплементација. Приликом започињања процеса пројектовања хибридне *SQL*/*NoSQL* базе података, прво се утврђује да ли већ постоји *SQL* база података. Уколико је то случај, та база података се узима као основ за анализу и даљи развој хибридне базе података. У супротном, пројектује се прототипска *SQL* база података, која ће служити као полазна тачка даљег развоја хибридне базе података. Пројектовање *NoSQL* базе података се реализује кроз фазе агрегирања и партиционисања, приказаних на Слици 2. Уједно, пројектоване *SQL* и *NoSQL* базе података представљаће


Слика 2 - Шематски приказ новоразвијеног приступа пројектовања хибридне *SQL*/*NoSQL* базе података и његових аспеката

компоненте хибридне *SQL*/*NoSQL* базе података. Свака фаза пројектовања *SQL* и *NoSQL* компоненте може се третирати као својеврсни инкремент, којим се додаје нова вредност хибридној бази података, која се пројектује. С обзиром на то да се развој и додавање инкремената могу понављати у итерацијама када је реч о пројектовању хибридне базе података, евидентно је да нови приступ пројектовања хибридне базе података укључује и аспект итеративно-инкременталног развоја. Свему наведеном претходи прикупљање и анализа захтева и дефинисање критеријума, што су фазе присутне и у различитим верзијама *SDLC* приступа.

Шематски приказ (Слика 2) има за циљ да прикаже општу повезаност новог приступа са постојећим моделима у пројектовању база података конкретних типова (*Enity-Relationship* и *NoAM* модел), као и са аспектима неких општих приступа у пројектовању информационих система, а примењивих и на базе података (*SDLC*, прототипски и итеративно-инкрементални развој).

## 4.2. Фазе и активности новог приступа за пројектовање хибридне *SQL/NoSQL* базе података

*UML* дијаграм активности новоразвијеног приступа за пројектовање хибридне *SQL*/*NoSQL* базе података, са свим фазама и активностима, дат је на Слици 3.

Заједничка карактеристика свих анализираних приступа за пројектовање је да њихова почетна фаза садржи активности прикупљања и анализе захтева. Иста ситуација је и са новим приступом пројектовања хибридне *SQL*/*NoSQL* базе података. Процес пројектовања, по новом приступу, започиње активностима отварања пројекта и прикупљања корисничких захтева. Да ли ће нова база података бити пројектована као хибридна *SQL*/*NoSQL* база података, односно да ли ће постојећа база података бити редизајнирана на начин да постане хибридна база података, зависи од више фактора. На то утичу, пре свега, прикупљени и анализирани захтева које информациони систем који се пројектује (заједно са својом базом података) мора да испуни. Поред тога, на одлуку о развоју хибридне базе података утичу и прикупљене статистике извршавања наредби над базом података. Управо се у томе огледа додатна флексибилност новог приступа, којим се не условљава пројектовање хибридне базе података по сваку цену, јер нови

приступ, под одређеним условима, пружа и могућност задржавања већ постојеће базе података.

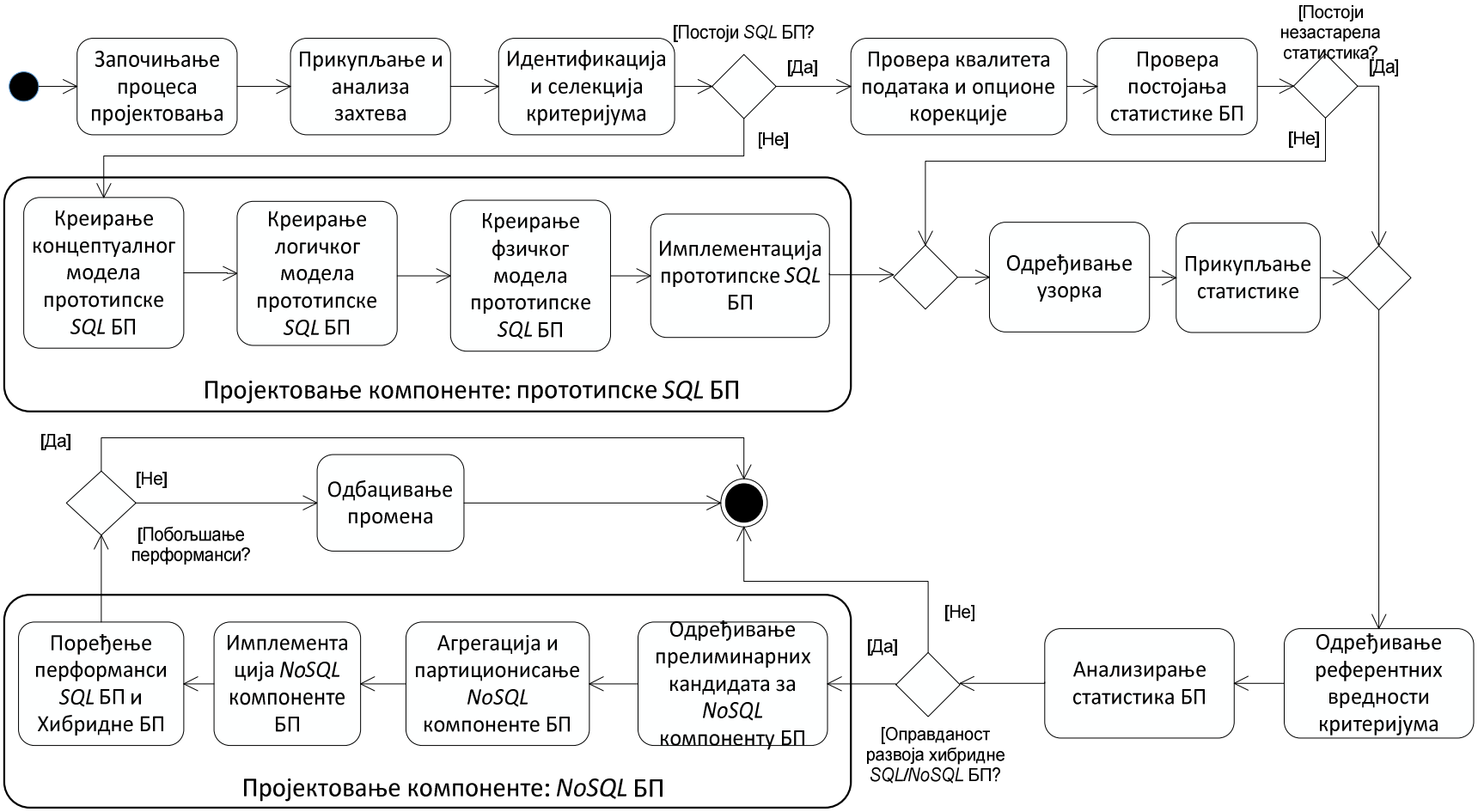
С обзиром на то да захтеви представљају важан аспект за даље пројектовање базе података, потребно их је класификовати и објаснити. Типови захтева од интереса, приликом пројектовања базе података, могу се грубо поделити на (Carkenord, 2009; IIBA, 2015):

- Пословне захтеве;
- Корисничке захтеве;
- Функционалне захтеве;
- Нефункционалне захтеве и
- Техничке захтеве.

Пословни захтеви дају одговор на питање „Зашто се нешто ради?“. Ова група захтева служи за идентификацију пословних проблема (које нова или побољшана база података треба да реши), за спецификацију различитих аспеката пословања (попут пословне терминологије, пословних активности, начина интеракције са другим системима и др.) и за утврђивање жељених циљева ради задовољења пословне сврхе (Carkenord, 2009).

Кориснички захтеви дају одговор на питање „Која су очекивања?“. Они дефинишу које је потребе и очекивања корисника неопходно да задовољи база података у циљу испуњења пословних захтева. Ови захтеви представљају спону између пословних захтева са једне стране и базе података која се пројектује са друге стране (IIBA, 2015). Иако је могуће формулисати и ширу категорију захтева „интересних страна“ (енг. *stakeholder requirements*), којој би припадали и кориснички захтеви (IIBA, 2015), због своје релевантности за нови приступ пројектовања хибридне *SQL/NoSQL* базе података, кориснички захтеви су издвојени као засебна категорија.

Функционални захтеви дају одговор на питање „Шта би систем требало да ради?“. Овом категоријом захтева се прецизира које су очекиване функционалности решења које се пројектује, као и начин на који ће корисник да има интеракцију са решењем.




Слика 3 - *UML* дијаграм активности за нови приступ пројектовања хибридне *SQL*/*NoSQL* базе података

Нефункционални захтеви дају одговор на питање „Колико добро систем ради?“ или „Колико је систем добар у задовољавању постављених захтева?“. Нефункционални захтеви служе за дефинисање оквира услова под којима решење мора задржати пројектоване карактеристике, а које су изражене показатељима успешности попут ефикасности, ефективности, поузданости, лакоће одржавања, перформанси итд.

Технички захтеви дају одговор на питање „Како ће систем бити направљен?“. Ови захтеви настају на основу извршене анализе захтева из претходних категорија и садрже техничке појединости решења које се пројектује, попут типа (*SQL*, *NoSQL*, хибрид) базе података која ће бити коришћена.

Испуњење пословних и функционалних захтева се узима као подразумевано. Уколико се приликом пројектовања базе података занемаре било пословни или функционални захтеви, таква база података неће бити сврсисходна, тј. неће служити својој предвиђеној намени. Слично, уколико се очекивања клијента (финансијера пројектовања) не задовоље, доводи се у питање оправданост и исплативост тако пројектоване базе података.

Ситуација је другачија по питању испуњавања нефункционалних захтева (у наставку дисертације за нефункционалне захтеве биће коришћена скраћеница НФЗ). Иако неки аутори наводе и да је већина проблема у реалном свету нефункционална (Chung, & Leite, 2000), испуњење НФЗ се не узима увек као подразумевано у процесу пројектовања базе података. Објашњење се може наћи у чињеници да процес дефинисања и реализације НФЗ није егзактан, већ је субјективан (Niu, Xu, & Bi, 2013). Самим тим, квантификација НФЗ је врло тешка, али су зато уложени додатни напори у дефинисање квалитативних метода оцене усклађености пројектованог система и нефункционалних захтева (Chung, & Leite, 2000). Још једна битна карактеристика нефункционалних захтева је да неретко могу бити у колизији (нпр. уобичајено супротстављени захтев максимизације перформанси система уз минимизацију употребе ресурса). Због свега наведеног, може се закључити да није изводљиво пројектовати систем или базу података која би обухватила све нефункционалне захтеве, већ је потребно прво класификовати НФЗ, одредити сучељене, усагласити НФЗ са пословним, функционалним и корисничким захтевима и на основу свега наведеног селектовати најзначајније

НФЗ. Наравно, експертско знање и искуство, као и познавање реалног система и сагледавање циљева система који се развија од великог су значаја приликом одабира приоритетних НФЗ.

Детаљну класификацију НФЗ и њихово распоређивање на скали која на једном крају садржи нефункционалне захтеве вођене искључиво пословним аспектом пројектовања, а на другом крају нефункционалне захтеве искључиво вођене техничким аспектом дали су аутори (Niu, Xu, & Bi, 2013). На средини скале концентрисани су нефункционални захтеви подједнако вођени техничким и пословним аспектом пројектовања, чији баланс споменута два аспекта омогућава природније повезивање са конкретним пословним, односно техничким захтевима. Анализом оваквих представника НФЗ уочено је да су управо агилност и перформансе од посебног значаја за нови приступ пројектовања хибридне *SQL/NoSQL* базе података.

Нови приступ за пројектовање хибридне *SQL/NoSQL* базе података користи један аспект перформанси, тј. време извршавања, као главни НФЗ. Уколико потребе захтевају, могуће је одабрати и неки други НФЗ, попут скалабилности (Carkenord, 2009). Пошто је за релевантни НФЗ у овом приступу изабрано време извршавања наредби (које се може третирати као типичан представник показатеља перформанси (Niedritis, Niedrite, & Kozmina, 2011)) други нефункционални захтеви неће бити детаљније обрађивани.

Уколико је већа сложеност механизма кванитифкације вредности по изабраном НФЗ-у и уколико је његова примена захтевнија по питању ресурса (нпр. потребног времена за примену) то би требало узети у обзир приликом планирања активности тестирања, анализе и поређења остварених резултата. Коришћење више НФЗ-ова утицало би на потребно време целокупног процеса пројектовања и захтевало би увођење одређеног система пондерисања НФЗ-ова, а што представља један од даљих праваца истраживања и проширења представљеног приступа. За прикупљање, спецификацију и анализу захтева могу се користи различите технике (нпр. Структурна системска анализа (Isaias, & Issa 2015), *IDEF* и *UML* модели (Kim, et al., 2003)).

Агилност добро дочарава карактеристику базе података да се на бржи или споријим начин прилагођава променама структуре података, тј. да адекватније

одговора изазовима складиштења и коришћења података различитог степена структурираности. Рад са подацима различитог степена структурираности представља једну од окосница коришћених у развоју новог приступа за пројектовање хибридне *SQL/NoSQL* базе података, чиме је и овај НФЗ посредно укључен у новоразвијени приступ. Време извршавања, а посредно и агилност (кроз коришћење података различитог степена структурираности) су присутни и у тестном примеру у Поглављу 7.

Структурираност представља специфичан кориснички захтев који је настао из потребе пословања да се омогући управљање подацима добијеним из различитих извора (нпр. трансакциони банковни подаци, подаци са друштвених мрежа и слично), без обзира на то колико стриктну структуру ти подаци поседују. Под појмом „структурираности података“ подразумева се особина података да поседују мање-више непроменљиву, фиксну, структуру (сви атрибути и њихови типови података познати су приликом пројектовања базе података), која је и формално дефинисана (тачно дефинисани називи и типови колона унутар табела *SQL* база података). Додатно појашњење овог термина може се наћи у литератури (Arenas, et al., 2014; Shepelev, 2011; Lebo, et al., 2017; Nickel, & Tresp, 2013). Највиши степен структурираности одговара структурираним подацима, који поседују стриктно дефинисану структуру, која се не мења (или се мења у занемарљиво малом броју случајева). Са друге стране, неструктурирани подаци имају флексибилну, често променљиву структуру, а неретко њихова структура уопште није ни дефинисана. Самим тим такви подаци поседују најнижи степен структурираности, тј. код њих је присутно потпуно одсуство структурираности. Структурираност података је неопходно узети у обзир приликом пројектовања хибридне базе података.

Циљ овог приступа је да понуди могућности пројектовања и редизајна базе података на основу степена структурираности података, који се изражава наменски развијеним показатељем, који је објашњен касније у овом поглављу. Због својих особина, за складиштење структурираних података типично се користе *SQL* базе података, док се за складиштење неструктурираних података типично користе *NoSQL* базе података. Структурирани (фиксна шема, без промене структуре) и неструктурирани подаци (флексибилна и променљива структура) представљају два екстрема по питању степена структурираности и само у одређеним ситуацијама је

очигледно којој од те две категорије ће припадати неки подаци (нпр. банковне трансакције ће припадати структурираним, а подаци са друштвених мрежа неструктурираним подацима). Проблем распоређивања већине података, који се не могу по „интуицији“ нити са сигурношћу сврстати у одговарајућу од две категорије, захтева увођење критеријума за одређивање степена структурираности таквих података. У литератури се за такве податке може наћи и термин „полу-структурирани“ (Abiteboul, 1997).

Идентификација и селекција критеријума за доношење одлуке о пројектовању хибридне базе података приказана је наредном активношћу на Слици 3. Уопштено говорећи, могуће је бирати различите критеријуме приликом доношења одлуке о пројектовању хибридне *SQL/NoSQL* базе података. Критеријум структурираности података добија све више на значају развојем нових видова комуникације и нових формата података, експоненцијалним растом података у употреби и потребом њиховог повезивања итд. Због тога, као и због раније изнетих аргумената о значају структурираности података, у овом приступу је као подразумевани критеријум управо узет степен структурираности података. Тај критеријум се користи приликом доношења одлуке да ли одређени подаци поседују висок степен структурираности и самим тим представљају кандидате за *SQL* компоненту хибридне базе података или имају низак степен структурираности и самим тим представљају кандидате за *NoSQL* компоненту хибрида.

Доношење одлуке по наведеном критеријуму врши се на основу прикупљене статистике базе података. Термин „статистика базе података“ требало би јасно дистанцирати од значења речи статистика у најширем смислу. У наставку дисертације под статистиком базе података подразумева ће се подаци извршавања свих наредби над базом података, прикупљени од стране сервера базе података. Када је прикупљање статистике активирано, извршавање сваке наредбе генерише одговарајуће податке у речнику података. Познати произвођачи система за управљање базама података, *Oracle* и *Microsoft*, подржавају разноврсне типове статистика извршавања над базом података, који су доступни кроз *Dynamic Performance Views* (Oracle, 2014) и *Dynamic Management Views* (Microsoft, 2017) респективно. На основу прикупљене статистике базе података могуће је одредити различите индикаторе коришћења базе података: број читања/писања, број рекорда

у табели, учесталост промене структуре табела, заузеће ресурса приликом извршавања наредби, селективност упита, најкоришћеније структуре приступа, просечан број сесија по кориснику итд. Овај приступ пројектовању хибридне базе података уводи индикатор УПСП (Учесталости Промене Структуре Података), који представља однос између броја наредби за промену структуре конкретне табеле (*ALTER* наредби) према укупном броју извршених наредби над истом табелом. Наведени индикатор је погодан за исказивање критеријума степена структурираности података, јер подаци са вишим степеном структурираности постижу нижу вредност за изабрани УПСП индикатор, што је у складу са њиховом карактеристиком да поседују ретко променљиву структуру. Супротно томе, подаци са нижим степеном структурираности имају вишу вредност УПСП индикатора, јер су склонији промени структуре. Овде је битно напоменути да се УПСП не мора користити као једини показатељ степена структурираности. Наиме, правац даљег истраживања може бити развој индикатора који би се користио заједно са УПСП, а којим би се одређивало у којој мери је нека пројектована шема, или њен део, прилагођена смештању структурираних података. Примера ради, колона која би чувала напомену и била типа *VARCHAR(4000)*, била би добар кандидат за анализу оправданости складиштења структурираних података. Такође, колоне типа *BLOB* и *CLOB* којима би била додељена велика количина простора за смештање података, могле би бити провераване ради утврђивања оправданости коришћења у структурираној шеми, односно могле би бити потенцијални кандидати за прелазак у *NoSQL* компоненту хибрида.

Битна карактеристика новог приступа пројектовања хибридне *SQL/NoSQL* базе података је одређивање кандидата који ће бити имплементирани као део *NoSQL* компоненте, односно потврда да ли ће и који подаци остати унутар *SQL* компоненте. С обзиром на то да различити типови *NoSQL* база података подржавају различите моделе чувања података (кључ-вредност, документ, фамилија колона и граф) са свим специфичностима које они садрже, нови приступ дефинише препоруке по којима се у зависности од различитих захтева, или њихове комбинације, пројектанти могу одлучити за одговарајући тип:

- Кључ – вредност базе података: погодне за складиштење великих мултимедијалних објеката, попут слика, аудио записа и слично;

- Документ базе података: поља са вишеструким вредностима, подаци у *JSON* формату, подаци са изразито варијабилним атрибутима;
- Фамилија колона базе података: скалабилност, фреквентни уписи великих количина података, потреба за високим степеном редундансе и
- Граф базе података: ентитети са великим бројем међусобних веза, сложени документи, подаци о повезаности корисника друштвених мрежа.

Ипак, ова активност није још увек аутоматизована и основни критеријум избора конкретног СУБП-а се заснива на ограничењима која евентуално постоје у реалном систему (расположивост лиценци одређеног СУБП-а, могућност плаћања лиценци новог СУБП-а, условљеност „наслеђеним“ деловима система који се не могу мењати) и на основу експертских знања и искустава пројектаната. Један од праваца даљег развоја приступа био би креирање правила (по могућству више-критеријумских) која би проширила и формализовала дате препоруке.

Наредна активност је провера постојања *SQL* базе података. Овај конкретан корак омогућава новом приступу за пројектовање хибридне базе података да буде примењив на нове базе података, које се тек пројектују, али и на постојеће базе података. Ако се након извршене анализе испостави оправданим, над постојећом базом података врше се процеси прилагођавања и редизајна, чиме се она трансформише у хибридну *SQL*/*NoSQL* базу података. У ситуацији када се база података пројектује по први пут, нови приступ омогућава, по потреби, да таква база података буде пројектована као хибридна. Уколико се након провере постојања утврди да иницијална *SQL* база података не постоји, врши се транзиција ка пројектовању прототипске *SQL* базе података, што је приказано на дијаграму на Слици 3. На дијаграму се може видети да је целокупан процес пројектовања *SQL* базе података обухваћен новим приступом за пројектовање хибридне базе података. Приказане су активности израде концептуалног модела (употребом *UML* дијаграма класа или *Entity-Relationship* дијаграма), креирање логичког модела (који за резултат има релациони модел добијен применом правила за превођење из *Entity-Relationship* модела) и креирање физичког модела (када се доносе све специфичне одлуке везане за имплементацију, нпр. одабир структуре приступа подацима,

употреба денормализације, партиционисања), након чега следи имплементација прототипске *SQL* базе података са свим појединостима специфицираним у претходним фазама.

За постојећу *SQL* базу података, проверава се ниво квалитета података који је затечен у њој, тј. проверава се ниво квалитета података који се у тренутку примене новог приступа већ налазе ускладиштени у њеним табелама. Уколико се установи да подаци нису задовољавајућег нивоа квалитета (логички пропусти приликом уноса, погрешни називи, недостајуће вредности и слично), примењује се нека од метода за побољшање нивоа квалитета података (Needham, et al., 2009). Контролу нивоа квалитета података могуће је реализовати у различитим фазама пројектовања. У представљеном приступу пројектовања хибридне *SQL*/*NoSQL* базе података, споменута активност се реализује непосредно пре провере статистике. Овакав тренутак реализације активности контроле нивоа квалитета је одређен у циљу смањења вероватноће да подаци нарушеног интегритета обмањујуће утичу на процес одређивања кандидата за *NoSQL* компоненту хибридне базе података.

Након описаних активности, гране дијаграма се спајају (грана пројектовања прототипске *SQL* базе података и грана постојеће *SQL* базе података, али која не садржи статистику). Тада започиње процес прикупљања статистике над *SQL* базом података, над новокреираном прототипском базом података или над постојећом базом података која не поседује незастарелу статистику. Под застарелом статистиком базе података се подразумева статистика над објектима базе података у којима је више од 10% записа промењено од тренутка прикупљања статистике (Oracle, 2018a).

У описана два случаја, потребно је одредити колико дуго ће се прикупљати статистика. Иако постоји директна зависност између дужине прикупљања статистике и њене поузданости, трајање процеса прикупљања зависи од много фактора, а најважнији је колико је времена од целокупног процеса пројектовања на располагању за наведену активност. Уколико се у датом временском оквиру може прикупити статистика у препорученом обиму (типично датом од стране искусних консултаната и пројектаната), нема потребе одређивати узорак. У супротном, уколико у процесу пројектовања није могуће издвојити потребну количину времена за прикупљање статистике препорученог обима, користи се техника узорковања.

Термин „узорковање“, у овом приступу, користи се у значењу одређивања потребног броја *SQL* наредби ради прикупљања репрезентативног дела статистике базе података, над којим ће бити израчунате вредности индикатора, у конкретном случају УПСП индикатора. Након прикупљања статистике (опционо са узорковањем), следи активност одређивања референтне вредности изабраног критеријума. За постојеће *SQL* базе података са незастарелом статистиком грана на дијаграму заобилази прикупљање статистике и врши се транзиција директно ка активности одређивања референтне вредности.

Референтне вредности могу се успоставити на основу искуства експерта, спроведене анкете, дедуктивне методе, применом неке технике одлучивања (*AHP* методе и слично) или на основу идентификованих репрезентативних случајева коришћења (и њихових сценарија).

У оквиру ове докторске дисертације искоришћена је референтна вредност добијена искуствено, а потврђена кроз репрезентативни случај коришћења изабраног примера, који је приказан у поглављу 7 ове дисертације. Конкретна референтна вредност за изабрани пример износи 0,1% (изражена путем УПСП индикатора) и означава да је од хиљаду извршених наредби над неком табелом једна *ALTER* наредба (тј. наредба којом се мења структура слога). Уколико је вредност УПСП индикатора над конкретном табелом мања од 0,1% може се сматрати да је структура одговарајућих података у тој табели релативно стабилна, да ти подаци имају висок степен структурираности, тј. да у задовољавајућој мери одговарају карактеристикама структурираних података. Такви подаци ће остати у *SQL* компоненти. Подаци одређене табеле који имају 0,1% или већу вредност УПСП индикатора сматрају се подацима фреквентно променљиве структуре, тј. подацима ниског степена структурираности. Таква табела из *SQL* компоненте постаје прелиминарни кандидат за имплементацију у *NoSQL* компоненту хибридне базе података.

Након дефинисања референтне вредности, оправданост развоја *NoSQL* компоненте (самим тим и оправданост наставка пројектовања хибридне *SQL*/*NoSQL* базе података) се анализира. На основу статистике базе података, утврђује се које табеле имају вредност УПСП индикатора 0,1% или већу. Ово је иницијални критеријум по којем се утврђује да ли је нека табела уопште

прелиминарни кандидат за прелазак у *NoSQL* компоненту. Уколико је то случај, кандидат за *NoSQL* компоненту ће бити пројектован у складу са фазама *NoAM* приступа (одређивање кандидата, агрегирање, партиционисање и имплементација), јер нови приступ обухвата и фазе *NoAM* приступа, као што је приказано на Слици 3.

У циљу олакшавања примене приступа, посебно када је модел сложен и када сама база података садржи велики број табела, одређене активности приступа су аутоматизоване. Тако на пример, активност прикупљања статистике базе података је у потпуности аутоматизована и реализује се од стране сервера базе података. Поред тога, процеси пројектовања прототипске *SQL* компоненте и *NoSQL* компоненте су аутоматизовани у мери у којој наведени процеси допуштају (нпр. на основу модела података могуће је генерисати *DDL* наредбе за креирање објеката шеме *SQL* компоненте, затим дефинисањем секвенце може се генерисати нова вредност примарног кључа, а на основу *NoSQL* агрегација може се извршити мапирање у међу-модел *NoAM* модела). Упркос наведеном, правац даљег развоја аутоматизације се може огледати у детекцији кандидата чије вредности премашују референтну вредност, чиме би се пројектантима скратило потребно време за идентификацију кандидата, уз истовремено смањење вероватноће превида неког кандидата.

Перформансе конкретних *SQL* и *NoSQL* кандидата се затим упоређују кроз анализу остварених времена извршавања наредби (изабрани НФЗ). За опис основних компоненти времена извршавања може се употребити формула (1):

$$\text{Трошак(в)} = \text{ТрошакКонекције} + \text{ТрошакНаредбе} \quad (1)$$

Уколико се приступа већем броју компоненти, потребно време за конекције се повећава, али оно представља релативно „фиксан“ временски утрошак. За разлику од тога, временски утрошак приликом извршавања наредби биће главна компонента обрачуна укупног временског утрошка. Временски утрошак извршавања наредбе зависиће од типа наредбе, типа базе података над којом се наредба извршава и броја слогова, као што је приказано на резултатима спроведених тестова у Поглављу 7.

Тек након упоређивања остварених времена извршавања наредби истог кандидата имплементираног у *SQL* и у *NoSQL* компоненти доноси се коначна одлука да ли се конкретан кандидат дефинитвно задржава у *NoSQL* компоненти (уколико је остварено краће време извршавања наредби имплементацијом кандидата у *NoSQL*) или се „враћа“ у *SQL* компоненту (уколико је време извршавања наредби кандидата као саставног дела *SQL* компоненте било краће).

У представљеном приступу, критеријум структурираности података је коришћен за иницијално одређивање прелиминарних кандидата за превођење у *NoSQL* компоненту. УПСП индикатор као изабрани показатељ степена структурираности података, а чија се вредност за сваку табелу добија из статистике базе података, коришћен је за одређивање прелиминарних кандидата за *NoSQL* компоненту хибридне *SQL*/*NoSQL* базе података. Након имплементације прелиминарних кандидата унутар *NoSQL* компоненте врши се поређење по изабраном НФЗ (нпр. по времену извршавања наредби). Поређењем остварених вредности по изабраном НФЗ доноси се коначна одлука о томе да ли се кандидат имплементиран у *NoSQL* компоненти дефинитивно задржава као саставни део исте, или ће ипак бити у *SQL* компоненти, чији саставни део је и био пре поређења. Уколико ни један кандидат из иницијалне итерације не покаже побољшање перформанси након имплементације у *NoSQL* компоненти, сви подаци ће остати у *SQL* компоненти, тј. ни једна табела из *SQL* компоненте неће прећи у одговарајућу структуру *NoSQL* компоненте. Након појаве првог кандидата за *NoSQL* компоненту који постигне бољи резултат по изабраном НФЗ (у конкретном примеру краће време извршавања изабраних наредби), оправдано је наставити процес даљег пројектовања хибридне *SQL*/*NoSQL* базе података.

# 5. Постојећи приступи за интеграцију и униформно коришћење *SQL* и *NoSQL* база података

Паралелно коришћење *SQL* и *NoSQL* база података је реалност и очекивано је да ови типови база података наставе да коегзистирају и у наредном периоду. То доводи савремене организације у позицију да се неретко опредељују за употребу различитих технологија, а у циљу остваривања погодности примене адекватне технологије на конкретан домен. Описана ситуација се са аспекта складиштења и коришћења података огледа у употреби база података различитих типова за специфичне потребе. Употребом различитих технологија може се остварити корист по одређеним критеријумима и под одређеним условима (један пример је приказан и у седмом поглављу ове дисертације), али то истовремено доприноси повећању нивоа комплексности целокупног система. У циљу задржавања уобичајеног функционисања система и коришћења додатних функционалности које пружају нове технологије, од великог је значаја омогућити паралелно коришћење тих технологија. Конкретно, са аспекта складиштења и коришћења података, након пројектовања система, потребно је омогућити и паралелно коришћење података, без обзира да ли су они смештени у *SQL* или *NoSQL* базе података. Због одсуства стандардизованог начина интеграције, као и због немогућности директног коришћења података из база података различитих типова (због различитих структура за складиштење података, различитих начина приступа подацима, различитих упитних језика итд.) важно питање које се поставља је:
„*На који начин интегрисати базе података различитих типова и омогућити њихово униформно коришћење као компоненти јединствене логичке, хибридне, базе података?*“.

Проблем паралелног коришћења база података различитих типова не би требало поједноставити до нивоа да се базе података различитих типова могу увек користити паралелно, приступом дистрибуираним (међусобно неповезаним или лабаво повезаним) базама података, а употребом специфичног упитног језика за базу података одговарајућег типа. Оваква поставка коришћења негира интегрисаност база података различитих типова, које се на описани начин квази „паралелно“ користе. Још важније, споменута поставка представља погодну основу

за настанак и неконтролисано ширење редундансе кроз, на споменути начин, дистрибуиране базе података по различитим типовима, што ствара аномалије у процесу складиштења података и коришћења база података, а може негативно утицати и на додатно оптерећење системских ресурса и на погоршање перформанси (одзив базе података, време извршавања итд.). Наведено посебно добија на значају када се говори о *NoSQL* базама података, јер одсуство стриктних механизама контроле правила интегритета (референцијални интегритет, правила интегритета ентитета итд.) још лакше може проузроковати настанак и увећање нежељене редундансе у њима. У ситуацији када се *NoSQL* базе података користе заједно са *SQL* базама података поред тешкоће контролисања и смањивања редундансе у појединачним базама података, велики и додатни изазов представља усклађивање података између *SQL* и *NoSQL* база података, тј. отклањање података који нису поновљени унутар једне базе података, али представљају редундантни податак када се узме у обзир и база података другог типа. Иако *SQL* базе података поседују напредније механизме за умањење редундансе, неопходан је интегралан приступ решавању аномалија уноса, промене и брисања слогова када се оне користе са *NoSQL* базама података. У дисертацији се анализира и предлаже приступ решењу интеграције и паралелног коришћења база података различитих типова, као компоненти јединствене логичке базе података, а не пуко „паралелно“ коришћење база података различитог типа.

Под појмом интеграције компоненти хибридне базе података подразумева ће се повезивање база података различитих типова (које улазе у састав хибрида) у јединствену логичку базу података, над којом ће се, као да је у питању једна инстанца базе података, примењивати „обједињени“ процес администрирања, управљања правилима интегритета и отклањања редундансе података. Под појмом униформног коришћења компоненти хибрида подразумева се пружање осећаја и погодности рада са јединственом базом података, при чему корисник не треба да води рачуна о томе у којој компоненти се налазе подаци, како истима приступа, нити како повезује податке из различитих компоненти у резултат извршавања наредби. Зато је важно, у споменутом контексту коришћења база података различитих типова као компоненти хибрида, термине интеграције и паралелног коришћења третирати нераздвојиво. Под интеграцијом и паралелним коришћењем,

у наставку дисертације, биће разматрани приступи који на различите начин интегришу базе података хетерогених типова у логичку целину.

Прегледом литературе утврђено је више различитих приступа у решавању проблема интеграције и паралелног коришћења *SQL* или *NoSQL* база података и они се могу сврстати у следеће категорије:

- Миграција (са *SQL* на *NoSQL* базу података или обратно),
- Коришћење униформног језика (избором једног од постојећих)
- Креирање новог униформног језика,
- Израда приступа за униформно коришћење *SQL* и *NoSQL* база података.

С обзиром на чињеницу да је фокус ове дисертације на хибридним базама података, посебно ће бити анализирано колико су решења из наведених категорија применљива и на проблем интеграције и униформног коришћења свих база података које улазе у састав хибридне базе података, тј. биће показано да анализирани приступи не задовољавају у потпуности наведене потребе и да су отворили простор за развој новог приступа за интеграцију и униформно коришћење хибридне базе података. Тај приступ коришћења хибридних база података, који је развијен у овој дисертацији биће презентован у наредном поглављу.

## 5.1. Миграција података

Миграција представља процес превођења структуре и података изворне базе података у одредишну базу података. Уколико се описани процес реализује између база података истог типа, реч је о мање захтевном виду миграције. У наведеном случају врши се анализа семантике шеме изворне базе података, њено превођење и прилагођавање семантици шеме одредишне базе података. Осим миграције између база података истог типа, могуће је спровести миграцију и између база података различитог типа. Комплексност овог начина миграције огледа се пре свега у различитости имплементације модела података на којима се заснивају изворна и одредишна база података, а које је потребно ускладити.

Различитост имплементације модела података у различитим типовима база података потиче од карактеристика конкретних типова база података. Наиме, *SQL*

базе података користе релације (које се имплементирају путем табела које задовољавају одређене услове, о којима ће бити речи у наставку), док *NoSQL* базе података користе парове кључ-вредност, документа, фамилије колона или чворове са везама, у зависности од конкретног типа *NoSQL* базе података. У циљу приближавања различитости модела потребно је дефинисати правила превођења између база података одговарајућих типова. У том процесу може се користити директно превођење концепата модела (нпр. превођење пара кључ-вредност или документа у ред табеле) или се, опционо, могу користити медијатори, тј. међу-модели. Сврха међу-модела је да олакша процес превођења, јер се изворни и одредишни модел преводе у међу-модел, који у мањој или већој мери представља компромисни модел, тј. модел који садржи одређене карактеристике како изворног тако и одредишног модела. Иако миграција између база података различитих типова теоретски може обухватити миграцију између *SQL* и *NoSQL* база података у оба смера, доступни радови углавном приказују процес миграције са представника *SQL* база података на представнике *NoSQL* база података.

*Blankers* (Blankers, 2014) даје опште смернице за миграцију база података, при чему акценат ставља на миграцију између два различита СУБП-а *SQL* база података. Општи поступак миграције *SQL* база података на *Oracle 12c SQL* базу података дат је у раду (Oracle, 2013). Група аутора (Potey, et al., 2015) је представила теоријски оквир за конверзију структурираних база података у неструктуриране, као и *cloud* заснован алат који се у ту сврху може користити. Примена приступа демонстрирана је на примеру миграције *SQL* базе података (*MySQL*) на *NoSQL* базу података (*MongoDB*). Миграцијом база података различитих типова бавила се и група аутора која је у свом раду (Hanine, Bendarag, & Boutkhoum, 2016) представила методологију мигрирања *SQL* базе података у *NoSQL* базу података и детаљно описала кораке и технике које она садржи. Поред тога, споменута група аутора је у програмском језику *Java* развила алат за миграцију и дала је приказ рада тог софтвера на примеру миграције података из *MySQL* у *MongoDB*. У раду (Khan, & Mane, 2013) аутори износе аргументе за оправданост преласка са *SQL* на *NoSQL* тип базе података и такође дају пример миграције из *MySQL* у *MongoDB*. Анализа поступка миграције из *SQL* базе података у базе података других типова, пре свега у објектне и *XML* базе података, дат је у раду (Maatuk, Ali, & Rossiter, 2008).

Упркос корисности примене миграције у ситуацији када постоји потреба преласка са изворног на неки други тип базе података, она не испуњава у потпуности иницијални захтев интеграције и паралелног коришћења база података различитих типова и само се у одређеној мери може посматрати као решење наведеног проблема. Наиме, миграцијом се проблем интеграције *SQL* и *NoSQL* база података своди на одабир базе података једног од типова у којој ће бити смештени сви подаци.

Одабир ексклузивне базе података одређеног типа, односно одабир једне компоненте хибридне базе, повлачи са собом немогућност истовременог коришћења више база података различитог типа, чиме се смањује погодност коришћења базе података адекватног типа за специфичне аспекте пословања организације, а што је даље у супротности са основним начелом хибридне базе података: да буде сачињена од база података хетерогених типова.

## 5.2. Униформни упитни језик

Друго решење за интеграцију и паралелно коришћење база података различитих типова представља употреба униформног језика. Униформни језик може имати улогу компоненте која би омогућила извршавање наредби јединствене синтаксе над различитим типовима базе података. Тиме би била могућа интеграција база података различитих типова. Такође, заједнички упитни језик омогућио би да се базе података различитог типа користе паралелно и интегрално, без потребе коришћења засебних упитних језика за сваки тип. Две могућности се намећу као природне опције: коришћење постојећег или развој новог униформног језика. У наставку, прво ће бити размотрена могућност избора униформног језика од постојећих упитних језика, а након тога биће анализирана и могућност развоја новог униформног језика.

### 5.2.1 Избор униформног језика од постојећих упитних језика

Пре разматрања могућности избора једног од постојећих упитних језика за униформни језик база података различитих типова, потребно је направити њихов преглед. Преглед најзначајнијих упитних језика по типовима база података следи у наставку.

Стандардизовани упитни језик за рад са релационим базама података је *SQL*. Он је временом постао нераскидиво својство споменутог типа базе података, па је њихов одомаћен назив постао *SQL* базе података. Стандардизованост *SQL* упитног језика се огледа у чињеници да је дефинисан по стандардима *ANSI* и *ISO*. Године 1974. у *IBM* истраживачкој лабораторији *Chamberlin* је дефинисао основне постулате наведеног упитног језика, који су се заснивали на *Codd*-овом релационом моделу (Lazarević, et al., 2006).

Данас, актуелна је верзија стандарда из 2016. године, која је увела подршку за препознавање шаблона међу редовима (*Row Pattern Recognition*) и подршку за *JSON* формат (Winand, 2017). Ипак, верзија стандарда из 1999. године (*SQL:99*) увела је значајне промене, попут подршке за рад са објектима, чиме је поставила основ савремених објектно-релационих база података, која је прихваћена од стране водећих произвођача система за управљање базама података, попут *Oracle*-а, *Microsoft*-а и *IBM*-а. Тако је *SQL* постао стандард и објектно-релационих база података (до тада релационих) и прерастао у нераскидиви део СУБП-ова попут *Oracle XE/EE*, *Microsoft SQL Server*, *Microsoft Access*, *PostgreSQL*, *IBM DB2* итд.

*SQL* је декларативни језик и као такав дефинише „ШТА“ је потребно урадити, а не и „КАКО“ ће нешто бити урађено. Током времена, декларативни основ је добио и процедуралну надградњу, која за разлику од декларативног језгра није стандардизована, већ се разликује међу произвођачима СУБП-ова. Наиме, сврха процедуралне надградње је иста код свих произвођача, али се синтакса разликује. С обзиром на то да су *Oracle* и *Microsoft* водећи произвођачи СУБП-ова (SolidIT, 2018), две најчешће коришћене процедуралне надградње су *Oracle*-ов *PL/SQL* и *Microsoft*-ов *T-SQL*.

*SQL* базе података се заснивају на релационом моделу. Релација са својим атрибутима може бити представљена као табела са одговарајућим колонама, али таква табела, како наводе аутори, мора испунити следеће услове (Lazarević, et al., 2006):

1) Не постоји дупликат врсте табеле,
2) Редослед врста није значајан,
3) Редослед колона није значајан и
4) Све вредности атрибута у релацијама су атомске.

Узимајући наведено у обзир, као основна структура складиштења података у *SQL* базам података може се третирати табела, која испуњава горе-наведене услове. У остатку рада под термином „табела“ (односно „табела базе података“) подразумева ће се управо табела описаних својстава.

Табела се састоји од једне или више колоне и може садржати ниједан или више редова. Важно својство табеле је да има фиксну структуру, која је иста за сваки ред табеле. Под одговарајућим условима структура табеле се може мењати (нпр. не може се додавати колона која садржи *not null* ограничење, а да се притом не дефинише *default* вредност за постојеће редове и слично), али ефекат те промене се одражава на сваки постојећи ред табеле, као и на сваки ред који ће након измене структуре бити додат. За рад са подацима дефинисане су *SQL* наредбе које су подељене у неколико категорија (Oracle, 2018c):

- Категорија наредби за дефинисање структуре (енг. *DDL - Data Definition Language*) – служи да се креира или мења структура било којег типа објекта. Наредбе, који су репрезентативни представници ове категорије су *CREATE* (креирање конкретног објекта), *ALTER* (измена конкретног објекта), *DROP* (брисање конкретног објекта), *TRUNCATE* (брисање свих редова);
- Категорија наредби за манипулацију подацима (енг. *DML - Data Manipulation Language*) – служи за приказ и ажурирање података, а представници ове категорије су наредбе *INSERT* (унос реда), *UPDATE* (измена једног или више редова), *DELETE* (брисање једног или више редова) и *SELECT* (селектовање података), тј. најчешће коришћена наредба *SQL*-а;

- Категорија наредби за управљање трансакцијом (енг. *Transaction Control Statements*) – служи да се управља извршавањем трансакције: *COMMIT* (потврђивање), *ROLLBACK* (поништавање) и *SAVEPOINT* (маркирање тачака извршавања трансакције);
- Категорија наредби за управљање сесијом (енг. *Session Control Statements*) – омогућава измену сесије (*ALTER SESSION*) и додељивање улога (*SET ROLE*);
- Категорија наредби за управљање системом (енг. *System Control Statement*) – састоји се од једне наредбе којом се мењају подешавања на нивоу система (*ALTER SYSTEM*);
- Категорија уграђених *SQL* наредби (енг. *Embedded SQL Statements*) – омогућава повезивање наредби за дефинисање структуре и манипулацију подацима са процедуралном надградњом *SQL*-а.

За разлику од *SQL* база података, за *NoSQL* базе података не постоји униформни језик за рад са подацима, ни на нивоу свих типова *NoSQL* база података, ни унутар конкретног типа (кључ-вредност базе података, базе података фамилије колона, базе података засноване на документима и граф базе података). За рад са *NoSQL* базама података доступан је већи број различитих језика, који нису директно компатибилни.

Упитни језици кључ-вредност база података заснивају се на концепту који се налази у самом називу овог типа *NoSQL* базе података. Кључеви су показивачи и идентификатори скупа вредности, произвољне структуре. *BerkeleyDB* је представник кључ-вредност *NoSQL* база података, који подржава *SQLite* упитни језик. *SQLite* представља извршно окружење, али и језик који се заснива на *SQL* синтакси. Због великог степена поклапања *SQLite* и *SQL* језика, тј. због истоветности свих релевантних клаузула и наредби које су од интереса за овај рад (*SELECT*, *FROM*, *JOIN*, *WHERE*, *INSERT*, *UPDATE*, *DELETE*), *SQLite* језик (који користи *BerkeleyDB* база података) нема потребе овде засебно представљати, али ће овај језик бити узет у разматрање приликом компаративне анализе језика за рад са *NoSQL* базама података. Као представник упитних језика кључ-вредност *NoSQL* база података, у наставку ће бити представљен *Redis*-ов упитни језик. Поред њега, у наставку ће бити представљени упитни језици за рад са *NoSQL* базама података

заснованим на документима (*MongoDB QL*, *UnQL*), на фамилији колона (*CQL*) и на графовима (*SPARQL*, *Cypher*).

*Redis QL* је упитни језик намењен раду са истоименим СУБП-ом. Основни концепт чине парови кључ-вредност, при чему је структура пропертија повезаних са кључем флексибилна. Одсуство стриктно дефинисане шеме представља препознатљиву карактеристику *NoSQL* база података генерално, а уједно и окосницу разлика у односу на *SQL* базе података. *Redis QL* не подржава синтаксу која се користи приликом креирања *SQL* наредби (подршка или интеграција са *SQL*-ом се не спомиње ни као део плана даљег развоја језика), већ поседује специфичне операције. За извршавање *CRUD* (*Create Read Update Delete*) операција користи наменске операције *SET* (за унос података и за ажурирање података), *GET* (за читање података) и *DEL* (за брисање података) (RedisDB, 2018). Све наведено, уз одсуство плана да се у скорије време изврши интеграција са *SQL*-ом, на којој други произвођачи увелико раде, условила је да се *Redis QL* није изборио за значајнију улогу међу језицима *NoSQL* база података и да је тренутно значајан само по томе што је језик који користи једна од тренутно најпопуларнијих кључ-вредност база података (SolidIT, 2018).

*MongoDB QL* је упитни језик истоимене базе података која припада типу *NoSQL* база података заснованих на документу. Језик за *MongoDB* је развијен у *C++*, а поседује интерактивни *Mongo Shell* интерфејс развијен у *JavaScript*-у (MongoDB, 2018). Аутори (Nakabasami, Amagasa, & Kitagawa, 2013) су разматрали извршавање прилагођених упита над *MongoDB* применом *LINQ* језика у *JavaScript* окружењу. Уопштено говорећи, структура *MongoDB* докумената се базира на *JSON* формату, у којој се пропертијима додељују вредности, при чему, као што је уобичајено за *NoSQL* базе података, структура документа може варирати. По узору на *JSON*, *MongoDB* подржава и сложене структуре, па тако документ може садржати друге документе, низове вредности и низове докумената, над којима се могу извршавати *CRUD* операције применом *MongoDB QL*-а.

Језик *MongoDB QL* користи „дот“ нотацију за позив операција над колекцијама у формату: *bp.kolekcija.operacija()*. Сходно наведеном, може се уочити да начин писања операција у *MongoDB QL* нема сличности са *SQL* упитним језиком,

за разлику од неких других језика *NoSQL* база података који подржавају *SQL* синтаксу, а чији опис следи у наставку. За унос документа у колекцију *MongoDB QL* користи *db.collection.insertOne()* и *db.collection.insertMany()* операције, у зависности да ли се уноси један или више докумената. За приказ података се користи *db.collection.find()* операција, која може бити параметризована условима који се пишу унутар витичастих заграда и који, попут *WHERE* клаузуле у *SQL*-у, филтрирају податке који се приказују. За ажурирање података расположиве су *db.collection.updateOne()* и *db.collection.updateMany()* наредбе, док операција *db.collection.replaceOne()* представља пандан *MERGE* наредби у *SQL*-у. Брисање докумената из колекције постиже се операцијама *db.collection.deleteOne()* и *db.collection.deleteMany()*. Детаљна спецификација како приказаних тако и других подржаних наредби доступна је на веб страници *MongoDB*-а (MongoDB, 2018).

*UnQL* (*Unstructured Query Language*) представља први покушај стандардизације упитних језика *NoSQL* база података, применом синтаксе која је заснована на *SQL*-у. Покренут је 2012. године од стране *Damien Katz*-а (*Couchbase*), *Richard Hipp*-а (*SQLite*), *Erik Meijer*-а (*Microsoft*) и *Gavin Bierman*-а (*Microsoft*) када су као представници компанија у којима раде најавили почетак имплементације новог језика у своје *NoSQL* базе података (van Ombergen, 2014). Међутим, тај план није реализован у потпуности. Од тада, *Microsoft* је понудио два начина за рад са својом *NoSQL* базом података заснованом на документима (*Azure DocumentDB*). Један начин омогућава писање упита употребом *JavaScript* синтаксе, док други подржава *SQL* синтаксу. Ипак, у документацији се не спомиње подршка за *UnQL* упитни језик (Microsoft, 2016). Са друге стране, *CouchDB* је у потпуности имплементирао подршку за *UnQL* (Grolinger, et al., 2013).

Језик *UnQL* чува податке у несортираним колекцијама, чији су елементи типа документа. Сваки документ има дефинисане пропертије, при чему структура пропертија може варирати, што представља типичну карактеристику *NoSQL* база података. Документ и његове пропертије је могуће дефинисати употребом *JSON* формата, а подржани су и *Integer*, *Float* и *String* типови података. Пошто је заснован на *SQL*-овој синтакси, *UnQL* подржава све *CRUD* операције. Наредбе имају исте називе као и у *SQL*-у (*INSERT*, *SELECT*, *UPDATE*, *DELETE*), а подржане су и

*WHERE*, *FROM*, *GROUP BY*, *HAVING* и *ORDER BY* клаузуле, док је директна подршка за *JOIN* изостала (van Ombergen, 2014).

*CQL* (*Cassandra Query Language*) је упитни језик развијен наменски за *CassandraDB*, базе података коју користи *Facebook* (Han, et al., 2011). *CassandraDB* развијена је на основу *Google*-ове *BigTable* базе података која припада колони-фамилија, али садржи и карактеристике кључ-вредност база података, што је у литератури подстакло полемику на тему којем подтипу *NoSQL* базе података *CassandraDB* заправо припада (Bach, & Werner, 2014). Због уске повезаности са *Google*-овим *BigTable*-ом, у овој дисертацији ће *CassandraDB* бити третирана као представник база података заснованих на фамилији колона, а *CQL* као представник језика наведеног типа.

Језик *CQL* податке смешта у фамилије колона. Фамилија колона представља скуп редова, при чему сваки ред може имати произвољан број колона. Синтакса *CQL*-а се може упоредити са синтаксом *SQL*-а. За унос података се користи *INSERT* наредба, за приказ *SELECT*, за филтрирање (само по кључу или индексираној колони) користи се *WHERE*, за сортирање (само по примарном кључу или његовим деловима) користи се *ORDER BY*, док је за груписање и спајање, тј. за *GROUP BY* и *JOIN* из *SQL*-а, подршка у потпуности изостала (Bach, & Werner, 2014).

Од језика за рад са граф базама података посебно се издвајају *SPARQL*, који представља покушај стандардизације упитних језика граф база података и *Cypher*, који има примену у популарном *Neo4J* решењу.

*SPARQL* (*SPARQL Protocol And RDF Query Language*) је стандард *W3C*-а (*World Wide Web Consortium*) који је настао 2008. године у циљу стандардизације упитног језика за *RDF* (*Resource Description Framework*) (W3C, 2008). Комплетан стандард је доступан на Интернету (SparQL, 2016) и у наставку ће бити наведене његове најважније одреднице. *RDF* је осмишљен као основа за процесуирање мета података, тј. као основ за пружање интероперабилности апликација које користе машински код. Уз помоћ *RDF*-а дефинишу се стандардни формати података графова, као и протоколи размене на Интернету. *Neo4J*, најпопуларнији представник граф база података (SolidIT, 2018), подржава семантички *Web* који је одређен *RDF*-ом, што је био услов да *Neo4J* подржи *SPARQL* упитни језик.

Подржана је логика *RDF*-а за успостављање везе између два ентитета, па тако *RDF* предикат може бити услов филтрирања у *SPARQL* упиту. Услову филтрирања претходи декларисање *URI* (*Uniform Resource Identifier*) адресе до скупа података, онтологије или неког другог документа. *SELECT* наредба је преузета из *SQL*-а и служи за креирање листе резултујућих *URI*-а који се приказују у форми табеле. Осим *SELECT* клаузуле и *WHERE* клаузула за филтрирање је преузета из *SQL*-а, мада се у ту сврху може користити и *FILTER*. Клаузула *CONSTRUCT* омогућава прављење резултујућег *RDF* графа. *ASK* клаузула враћа вредност типа *Boolean* и користи се да би се прелиминарно, пре приказивања резултујућег графа, утврдило да ли упит има резултат. За унос података се користи *INSERT*, за измену *MODIFY*, а за брисање *DELETE*.

*Cypher* је декларативни упитни језик развијен за потребе *Neo4J* граф базе података, којим је омогућена претрага и ажурирање графова. Употребом сопственог *Cypher* упитног језика, *Neo4J* је стекао независност у односу на *SPARQL*. Као и сваки декларативни језик, намена му је да одговори на питање „*Шта треба постићи?*“, а не и „*Како постићи?*“. За разлику од императивних језика у којима се у тренутку писања наредби дефинише и како ће иста бити извршена, корисник *Cypher*-а не дефинише експлицитно начин извршавања, нити у том тренутку врши оптимизацију упита. Ипак, *Cypher* поседују подршку интеграције са императивним језицима (попут *Java*-е) и скрипт језицима (попут *Gremlin*-а и *JRuby*-а). Упити *Cypher*-а представљају константне низове карактера, чиме је омогућено њихово складиштење попут компајлираних *SQL* упита. Резултат извршавања упита може садржати чворове са свим атрибутима, чворове са одређеним атрибутима или агрегиране податке (Holzschuher, & Peinl, 2013). Последњих година појавила се и верзија слободно доступне спецификације *Cypher* упитног језика, под називом *openCypher* (OpenCypher, 2016). Синтакса *Cypher*-а користи наредбе из других упитних језика, највише из *SQL*-а, из којег су усвојене клаузуле *WHERE* и *ORDER BY*. За приказ података користи се *RETURN*. Осим приказа података, *Cypher* омогућава и преостале *CRUD* операције. За креирање новог чвора или везе користи се *CREATE* наредба, за унос вредности *SET*, за брисање *REMOVE*, а за измену комбинација *SET* и *FOREACH*.

Дати опис представника језика за рад са различитим типовима *NoSQL* база података отвара простор за њихову компаративну анализу. Анализу је потребно спровести са аспекта могућности униформног коришћења представљених језика на нивоу типа *NoSQL* базе података којој припадају (кључ-вредност, документ, фамилија-колона, граф), на нивоу *NoSQL* база података уопштено, као и са аспекта изводљивости интеграције са *SQL* језиком, који представља стандардизовани језик истоименог типа базе података. Интеграција са *SQL* базама података је круцијална у контексту решавања изазова униформног и истовременог коришћења *SQL* и *NoSQL* база података. Сходно томе, Табела 2 садржи колоне: назив типа базе података којем изворно припада посматрани језик, назив језика, представнике СУБП-ова који користе конкретан језик, као и три колоне са оценама применљивости и лакоће повезивања. Прва од те три колоне описује степен заступљености анализираног језика унутар изворног типа *NoSQL* базе података којој припада, тј. могућност проширивања опсега примене на остале представнике истог типа *NoSQL* базе података. Друга од те три колоне садржи оцене применљивости одређеног језика на остале типове *NoSQL* база података, док последња колона Табеле 2 садржи оцене могућности интеграције конкретног језика са *SQL*-ом. За оцену применљивости језика у сопственом типу *NoSQL* базе података, у свим типовима *NoSQL* база података и у интеграцији са *SQL*-ом коришћена је скала са следеће три оцене: + + *подржано*, + - *делимично подржано*, - - *није подржано*. Оцене су даване са аспекта лакоће интеграције *CRUD* наредби конкретног језика са *CRUD* наредбама других језика, пре свега *SQL*-а. У наставку следи анализа података из табеле.

*BerkeleyDB* и *RedisDB* су анализирани представници кључ-вредност *NoSQL* база података. Разлика између ова два решења је, између осталог, у језику који користе. *RedisDB* користи наменски развијени језик, који не подржава *SQL* синтаксу и који се није наметнуо као водећи језик кључ-вредност база података, ни као универзални језик *NoSQL* база података. Наведено се одразило на оцене могуће интеграције са другим типовима база података, те је *RedisDB* са својим специфичним процедурама постигао минимални скор, тј. рангиран је као језик без могућности шире примене и интеграције. Слична ситуација је и са језиком за рад са *MongoDB* базом података. Упркос већ наведеној чињеници да је у питању

тренутно најпопуларнији представник *NoSQL* база података, његов језик, по истом принципу као и *Redis QL*, остварује минимални учинак по питању анализираних могућности универзалне примене и интеграције база података различитих типова.

Табела 2 - Упоредни приказ упитних језика *NoSQL* базе података, са оценама могућности интеграције

| Тип *NoSQL* базе података | Назив језика | Корисници | Заступљеност унутар типа *NoSQL* базе | Примењивост на друге типове *NoSQL* б.п. | Интеграција са *SQL*-ом |
|---|---|---|---|---|---|
| Кључ-вредност база података | *SQLite (SQL)* | *BerkeleyDB* | + - | + + | + + |
| Кључ-вредност база података | *Redis* | *RedisDB* | - - | - - | - - |
| База података засноване на документу | *MongoDB QL* | *MongoDB* | - - | - - | - - |
| База података засноване на документу | *UnQL* | *CouchDB, Microsoft* (незванично) | + - | + - | + - |
| База података засноване на фамилији колона | *CassandraQL (CQL)* | *CassandraDB* | - - | + - | + - |
| Граф база података | *SparQL* | *Neo4J, AllegroDB* | + - | - - | + - |
| Граф база података | *Cypher* | *Neo4J* | - - | - - | - - |

За разлику од *MongoDB QL*-а и *Redis QL*-а, *BerkeleyDB* подржава *SQLite* језик који је у широј употреби. Језик *SQLite* користи и истоимена база података која је типа *SQL* базе података, а његова изведеност из *SQL*-а омогућава додатне могућности примене (SQLite, 2016). Иако *SQLite* не подржава све наредбе на идентичан начин као *SQL*, све релевантне наредбе за ову анализу (као што је већ појашњено у делу где су описани анализирани језици) су подржане на идентичан начин као у *SQL*-у, те се *SQLite* може третирати као верзија *SQL* синтаксе. То је утицало да *SQLite* језик добије максималну оцену („потпуне подршке“) за интеграцију са *SQL*-ом, што је *SQLite* високо рангирало по наведеном критеријуму, поготову када се узме у обзир да остали језици тренутно не поседују или поседују само делимичну повезаност са релевантним наредбама *SQL*-а, значајним за

интеграцију и универзално коришћење база података различитих типова. Примењивост *SQLite*-а је такође добра и на друге типове *NoSQL* база података.

*SparQL* и *Cypher* представљају наменске језике за рад са граф базама података и као такви имају доста одступања у односу на остале језике, укључујући и *SQL*. Ипак, због подржаности од стране више произвођача СУБП-ова, *SparQL* остварује бољи резултат од *Cypher*-а, у категорији заступљености језика у изворном типу *NoSQL* базе података. Због већег степена подударања наредби са *SQL*-ом, *SparQL* остварује бољи резултат од *Cypher*-а и у категорији интеграције са *SQL*-ом.

По сличном принципу, иако неке интересне стране, упркос најавама, још увек нису почеле да га користе (*Microsoft*), *UnQL* остварује делимичну подршку међу представницима свог типа *NoSQL* базе података (засноване на документима). Због подршке већине истраживаних *SQL* наредби, *UnQL* и *CQL* су рангирани као делимично примењиви и на остале типове *NoSQL* база података, пре свега на оне чији језици подржавају *SQL* синтаксу и делимично примењиви у директној интеграцији са *SQL*-ом, због изостанка неких битних операција, попут *JOIN*-а. *CQL* језик није подржан међу осталим представницима база података заснованих на фамилији колона, али с обзиром да *CQL* у великој мери подржава *SQL* наредбе (чак се релевантне наредбе за ову анализу у потпуности подударају са *UnQL*), остварује бољи резултат у категорији примењивости на друге типове *NoSQL*-а, него унутар база података заснованих на фамилији колона.

На основу свега изложеног, закључује се да не постоји непознаница око језика који се користи на нивоу *SQL* база података. Код *NoSQL* база података је потпуно супротна ситуација, што отежава избор једног од постојећих језика који би био примењив за интеграцију и униформно коришћење база података свих типова. Као што је спроведена анализа показала, постоји значајно одступање између концепата и наредби које користе различити типови *NoSQL* база података. Поред тога, на суштинску немогућност избора једног од постојећих *NoSQL* језика (у форми у којој су доступни без икаквих модификација) у функцији интегратора и униформног језика за рад са базама података различитих типова утиче и одсуство комплетне повезаности са *SQL*-ом. Иако је *SQLite* пришао најближе остварењу последње споменутог циља (као језик који користи *BerkeleyDB*), у питању ипак није

*NoSQL* језик, већ варијанта *SQL* језика, који се користи у *SQL* базама података и који је „позајмљен“ представнику *NoSQL*-а (*BerkeleyDB*). Док се остали *NoSQL* језици не приближе *SQL*-у као стандардизованом језику или док се не развије стандардизовани језик на нивоу свих *NoSQL* база података, избор једног од језика *NoSQL*-а за рад са свим типовима база података није остварив. Такође, због недостатка потпуне подршке *SQL*-а од стране већине *NoSQL* база података, ни *SQL* у свом изворном облику, без одређених модификација и прилагођавања, не може директно да преузме улогу униформног језика и интегратора база података које су различитог типа. То отвара простор за разматрање могућности развоја новог униформног упитног језика за рад са свим типовима база података, што је урађено у наставку дисертације.

### 5.2.2 Развој новог униформног упитног језика

Развој униформног упитног језика представља још један од приступа за решавање проблема интеграције и униформног коришћења *SQL* и *NoSQL* база података. Заједничко за све предлоге решења из ове категорије је став да је потребно одредити униформни језик који би омогућио несметано извршавање униформних наредби над свим типовима база података. С обзиром на стандардизованост језика за рад са *SQL* базама података, евидентно је улагање напора у циљу стандардизације униформног језика за рад са *NoSQL* базама података. У складу са тиме, неки од произвођача *NoSQL* база података опредељују се да имплементирају постојеће упитне језике у своје производе (*BerkeleyDB* подржава *SQLite*, *AllegroGraph* подржава *SparQL*), док други развијају нове језике (*Cypher* је развијен за потребе *Neo4J*-а, *CassandraQL* развијена је за *Cassandra* базу података) (Holzschuher, & Peinl, 2013; Han, et al., 2011). Ипак, одсуство консензуса произвођача *NoSQL* база података око установљења одређеног језика за стандардни упитни језик, као и чињеница да су последњих година настали нови упитни језици за рад са *NoSQL* базама података, додатно отежава коришћење више различитих *NoSQL* база података на интегрисани и униформни начин. На основу наведеног, закључује се да би захтеви, које би требало поставити пред нови упитни језик за универзално коришћење база података различитих типова, требало да гласе:

- Омогућити универзално коришћење *NoSQL* база података свих типова и
- Обезбедити интеграцију новог упитног језика са *SQL* упитним језиком, као стандардизованим језиком за рад са *SQL* базама података.

На тешкоћу реализације првог захтева утиче немогућност да се језик који би тек био развијен наметне као стандардни. Тренутно, без активног укључивања водећих произвођача (или неког реномираног тела за стандардизацију) и њихове колаборације на новом стандардном језику за рад са *NoSQL* базама података, сваки појединачни покушај развоја универзалног језика би деловао као узалудни покушај, јер би само додао један у низу језика који нису прихваћени од стране водећих произвођача база података. Други захтев, иако обиман због великог броја нових језика, отвара простор за интеграцију између сваког новог (али и постојећег) језика за рад са *NoSQL* базама података и *SQL* језика. На основу изложених фактора који утичу на отежано успостављање универзалног језика, не чуди што се последњих година појављују наменски приступи интеграције и униформног коришћења база података различитих типова. Они превазилазе велику комплексност и међусобну супростављеност захтева које би новоразвијени језик морао да испуни (због различитих концепата, структура чувања података, начина приступа итд.) да би био погодан за све типове база података. Такође, споменути наменски приступи елиминишу потребу избора униформног језика од тренутно доступних језика, који не могу у потпуности задовољити потребе рада са различитим типовима база података. Овим се елиминише и мукотрпни процес наметања изабраног језика као униформног језика свим произвођачима различитих типова база података. Све наведено допринело је да ови наменски приступи буду детаљније размотрени у наставку.

## 5.3. Наменски приступи интеграцији и униформном коришћењу *SQL* и *NoSQL* база података

Идентификована ограничења у примени миграције и униформних језика као поступака за решавање проблема интеграције и униформног коришћења база података различитих типова временом су отворила простор и за настанак наменских приступа за решавање споменутих проблема. Заједничко за те приступе и за поступке миграције и коришћења униформног језика је да сви они подржавају концепт вишеслојне архитектуре. Због тога, у наставку ће бити објашњени основни концепти везани за трослојну архитектуру, након чега ће бити приказани приступи који је користе и модификују у циљу интеграције различитих система и база података.

Вишеслојна архитектура у најопштијем смислу представља клијент-сервер архитектуру, која поседује више слојева, повезаних у логички систем. Раздвајањем различитих ресурса система по слојевима постижу се флексибилност и модуларност, чиме се олакшавају измене система, без обзира да ли је реч о изменама због редовног одржавања и прилагођавања или због унапређења. Флексибилност вишеслојне архитектуре се пре свега огледа у њеној карактеристици да приликом промене једног слоја, није увек нужно мењати и остале слојеве. Модуларност архитектуре омогућава да различите апликације користе исте слојеве. Тако на пример, проверу стања на банковном рачуну могуће је извршити са различитих уређаја који користе различите интерфејсе за приступ истим подацима. У зависности да ли је реч о једноставним апликацијама или свеобухватнијим информационим системима богатство садржајем презентационог слоја се може разликовати, али слој података у оба случаја може бити исти.

Типичан представник вишеслојне архитектуре је трослојна архитектура на којој се неретко развијају савремене апликације и информациони системи. Као што и сам назив трослојне архитектуре сугерише, сачињена је од три слоја:

- Презентационог слоја,
- Слоја пословне логике и
- Слоја података.

Презентациони слој је задужен за размену информација између корисника и система. Он омогућава кориснику унос инструкција и приказује му резултат извршавања задате инструкције над системом. У зависности од специфичних потреба, овај слој може садржати један или више корисничких интерфејса који обухватају хардвер и одговарајући софтвер. У зависности на који начин је кориснички интерфејс имплементиран, издвајају се следећи имплементациони типови интерфејса (Myers, 1998):

- Интерфејс текстуалних едитора (командних линија) – представља најстарији вид корисничких интерфејса, који је заснован на текстуалној комуникацији. Корисник уноси одређене наредбе типично путем тастатуре (или неког другог периферног уређаја) у текстуални едитор или командну линију. Потребно је да корисник унапред познаје синтаксу и значење наредби које жели да изврши и чији резултат извршавања жели да види. Пример примене представљају разни видови терминала.
- Графички кориснички интерфејс – заснива се на графичком приказу окружења за комуникацију са системом (форме, иконице, показивачи итд.). Данас је већина корисничких интерфејса заснована на овом типу, јер савремени информациони системи, савремене апликације за управљање различитим аспектима пословања, наменске апликације за извештавање, системи за управљање пословним одлучивањем и слични системи користе графички приказ за реализацију интеракције са корисником.
- Интерфејс заснован на менију – садржи унапред дефинисане опције за рад са системом, које корисник бира кретањем кроз мените и подмените. Након навигације до жељене опције корисник извршава неку од предефинисаних наредби. Оваквим интерфејсом се ограничава скуп функционалности које су кориснику на располагању, али се истовремено смањује и вероватноћа погрешно позване или унете наредбе. Типичан пример примене представљају банковни аутомати (*ATM – Automated Teller Machine*).
- Интерфејс заснован на природним језицима – још увек у повоју, омогућава кориснику да употребом говорног језика или покрета комуницира са системом. Пример примене овог типа интерфејса представљају апликације

„паметних“ агената (нпр. *iPhone Siri* или апликације мултимедијалних система у савременим аутомобилима које препознају покрете и гласовне команде).

Слој пословне логике задужен је за управљање комуникацијом између презентационог слоја и слоја података. Може бити имплементиран употребом различитих технологија, принципа и патерна. Овај слој представља слој архитектуре у којем може доћи до креирања додатних слојева, када трослојна архитектура прераста у вишеслојну.

Слој података представља извор и понор свих података у информационом систему. Овај слој омогућава складиштење, измену, чување перзистентних података и извршавање упита над истима. Информациони систем без перзистентних података не би био сврсисходан, па су од великог значаја пројектовање и имплементација слоја података. Иако је могуће да различити типови фајлова буду извори података (текстуалне, *XML, CSV* и друге датотеке), због својих доминантних функционалности за управљање подацима, базе података се у овој дисертацији третирају као подразумевано имплементационо окружење слоја података.

Трослојна архитектура је послужила као основ наменских приступа за решавање проблема интеграције различитих база података, при чему су је аутори, у различитим радовима на различите начине проширивали. Одређени аутори наводе да пројектовање заједничког корисничког интерфејса може имати улогу интегратора (Zhu, & Wang, 2012). Било је више покушаја стандардизације корисничког интерфејса којима би била омогућена интеграција база података које су различитог типа. Аутори (Atzeni, Bugiotti, & Rossi, 2012) су употребом приступа израде метамодела креирали генерички интерфејс назван *SOS* (*Save Our Systems*). Идеја иза овог приступа је да се апстрахују специфичности тиме што се конкретни интерфејси мапирају у генерички. Приступ су применили на проблем повезивања различитих типова *NoSQL* база података, што су демонстрирали кроз интеграцију *MongoDB*, *Redis* и *HBase* СУБП-ова. Проблемом израде заједничког интерфејса и синхронизацијом упита бавили су се аутори (Polese, & Vacca, 2009). Они су у свом приступу нагласили важност да улога администратора базе података буде додељена

особи од искуства. Таква особа, у садејству са алатом за синхронизацију, има пресудан утицај да отклони кризне ситуације које могу настати када се упити извршавају над застарелим верзијама шема (када упити нису испратили промене у шеми базе података). Аутори (Polese, & Vacca, 2009) предлажу креирање интерфејса за промену упита, који би у позадини вршио трансформацију упита у одговарајућу, нову, шему базе података, као и креирање интерфејс менаџера (*Interface Manager*) који би представљао додатни слој за усклађивање комуникације између корисника и интерфејса алата. У литератури се могу наћи различити приступи за мапирање специфичности између језика за рад са *SQL* и *NoSQL* базама података. Једно решење за интеграцију база података различитих типова, имплементирано кроз мапирање *SQL* упитног језика у наредбе за рад са *HBase* СУБП-ом, представили су аутори (Vilaça, et al., 2013). Ово решење може имати корисну примену посебно за коришћење *SQL* наредби и клаузула које нису подржане од стране *HBase*-а. Сличан приступ у свом истраживању имали су и аутори (Calil, & dos Santos Mello, 2012) када су предложили интерфејс који подржава *SQL* наредбе, а који се може повезати са *SimpleSQL* базом података. Аутори (Tatemura, Hsiung, & Hacigümüş, 2012) су дали обухватнији предлог креирања *SQL* мапера којим би се омогућило коришћење *SQL* упита над *NoSQL* базама података. У споменутом раду представљен је слој назван *Partiqle*, који служи за мапирање *SQL* упита у упите погодне за извршавање над *NoSQL* базама података. Споменуто решење приказано је само на примеру рада са *HBase*-ом. Упркос општем приступу повезивања *SQL* језика са *NoSQL* системима, недостатак овог приступа је неопходност специфичне имплементације конкретног *NoSQL* језика.

Сви до сада анализирани и представљени радови представљају добру стартну позицију за даљу анализу могућности повезивања *SQL* језика са *NoSQL* базама. Иако већина њих пружа могућност повезивања *SQL* језика са *NoSQL* језиком, главни недостаци ових радова остају специфичност СУБП-а на које се односе, као и одсуство директне интеграције са неком конкретном *SQL* базом података. Ово је разматрано у радовима који следе у наставку.

Аутори (Roijackers, & Fletcher, 2013) су такође обрађивали проблем интеграције база података хетерогених типова, али пре свега са аспекта паралелног

читања података из *SQL* и *NoSQL* база података. За наведене активности ови аутори су развили прототипски оквир и мапер за превођење наредби *SQL* упитног језика у *NoSQL* наредбе. Споменути прототипски оквир задужен је да податке добијене из *NoSQL* базе података прикаже у форми релационог модела, чиме допуњује улогу мапера. Примена предложеног приступа демонстрирана је на проблему интеграције *PostgreSQL* и *MongoDB* СУБП-ова. Пример прототипски развијеног оквира за интеграцију *SQL* и *NoSQL* база података, названог *SQL*/*MED*, приказан је и у раду аутора (Melton, et al., 2002). Овај рад је био међу првима који је обрадио наведену проблематику, а приказ примене приступа дат је, такође, на проблему интеграције *PostgreSQL* и *MongoDB* СУБП-ова. Мапирање између *NoSQL* и *SQL* концепата, као и између *SQL* и *XDM (XML Documentation Mapings)* концепата била је тема рада (Valer, Sauer, & Härder, 2013), а све у циљу стварања основе за примену *XQuery* језика (тј. упитног језика *XML*-а) на *NoSQL* базе података. Конкретније решење проблема интеграције и униформног приступа различитим типовима база података дали су аутори који се залажу за уметање наменских слојева интеграције и њихови радови ће бити објашњени у наставку.

У раду (Sellami, Bhiri, & Defude, 2014) предложен је *REST* (*Representational State Transfer*) кориснички интерфејс назван *ODBAPI* који омогућава равноправно извршавање *CRUD* операција *SQL* упитног језика над *SQL* и *NoSQL* базама података. Ово решење не подржава сложене упите нити употребу *JOIN* клаузуле, што представља и његов највећи недостатак, а његова примена је, у споменутом раду, демонстрирана на примеру интеграције *Riak*-а и *CouchDB*-а. Глобална шема за *SQL* и *NoSQL* базе података представља приступ решавања проблема њихове интеграције, који је приказан у раду (Curé, et al., 2011). У овом раду, разлика између императивних језика већине *NoSQL* база података и декларативног *SQL* језика решена је увођењем језика за мапирање, којим се концепти изворне базе података (без обзира да ли је у питању *SQL* или *NoSQL* тип базе података) мапирају у глобалну шему. Условљеност да се чак и *SQL* наредбе корисника морају преводити, као и одсуство подршке за сложене упите и употребу *JOIN* клаузуле представљају највеће недостатке овог приступа. Још један предлог слоја интеграције, назван *Sinew*, објашњен је у раду (Tahara, Diamond, & Abadi, 2014). *Sinew* представља слој који такође омогућава извршавање *SQL* наредби над *SQL* и *NoSQL* базама података,

а његова специфичност се огледа у подршци *JSON* формата, чиме је постигнуто да се подаци добијени из *SQL*-а и *JSON*-а могу заједно користити и да се над њима могу извршавати упити.

Аутори (Alomari, Barnawi, & Sakr, 2015) предлажу заједнички модел података и униформни кориснички интерфејс за *SQL* и *NoSQL* базе података. Тиме су презентациони слој и слој пословне логике, трослојне архитектуре, у потпуности ослобођени било каквих специфичности конкретних база података. На слоју података користи се заједнички модел, добијен употребом развијеног алата за конверзију, трансформацију и размену података између различитих модела података. Примена приступа демонстрирана је на интеграцији *MongoDB*, *Google DataStore* и *SimpleDB* СУБП-овима. Прва два СУБП-а су представници база података заснованих на документима, док се трећи најчешће сврстава у базе података засноване на фамилији колона.

*Lawrence* је у свом раду (Lawrence, 2014) истраживао проблем интеграције уопштено, али и са аспекта интеграције различитих типова *NoSQL* база података, које не подржавају *SQL* упитни језик, већ користе сопствене језике. Овај аутор за решење споменутог проблема предлаже додатни слој за униформни приступ, који проширује трослојну архитектуру. Тај слој третира се као слој виртуелизације и назван је *Unity*. Могућност интеграције представника *NoSQL* база података, који не подржава *SQL* упитни језик, са *SQL* базом података демонстрирана је на примеру интеграције *MongoDB*-а и *MySQL*-а. Слој виртуелизације омогућио је постављање истовремених упита над оба типа базе података (документ *NoSQL* и *SQL*). По мишљењу неких аутора који су обрађивали исту тематику (Alomari, Barnawi, & Sakr, 2015), *Unity* представља једно од најнапреднијих решења за споменути проблем интеграције и униформног приступа. Главна карактеристика овог решења је да су базе података различитих типова искоришћене за складиштење различитих података, што је итекако пожељна карактеристика, јер омогућава да се база података прикладног тип користи за одговарајућу намену и притом је могуће свакој од њих, са аспекта корисника, приступити униформним језиком, употребом *SQL*-а.

## 5.4. Хибридна база података као приступ интеграцији и униформном коришћењу *SQL* и *NoSQL* база података

Анализирани приступи развијани за интеграцију и униформно коришћење база података различитих типова представљају добар основ за решавање питања интеграције и коришћења компоненти хибрида, али их не могу решити у потпуности. У већини радова, за језик који има улогу споне база података различитих типова изабран је *SQL*, због своје стандардизованости и глобалне заступљености међу корисницима база података. Мапирањем хетерогених језика *NoSQL* база података у *SQL* постигнута је униформисаност приступа и интеграција у одређеној мери. Такође, неки аутори (попут *Lawrence*-а) у својим приступима су обезбедили интеграцију, у одређеној мери, проширивањем стандардне трослојне архитектуре, чиме су показали да се, уз одговарајућа проширења, трослојна архитектура може третирати као погодан основ за специфичне потребе интеграције база података различитих типова. Међутим, са аспекта хибридних база података споменути радови оставили су простор за даљу интеграцију и унапређење униформности коришћења различитих база података као компоненти хибридне базе података. Наиме, захтев који се природно поставља пред хибридне базе података је да, поред погодности које доноси пројектовање јединствене логичке базе података, оне кориснику треба да пруже и једноставност рада са јединственом базом података, без обзира којој компоненти хибрида корисник приступа. Пошто базе података различитог типа чине компоненте хибридне базе података, кориснику би требало обезбедити осећај рада са јединственом базом података, уместо осећаја рада са више појединачних база података. Наведено захтева постизање вишег степена униформисаности и интеграције компоненти хибридне базе података. Простор за даље повећање нивоа интегрисаности огледа се пре свега кроз енкапсулацију и обједињену администрацију свих компоненти, без обзира којег су типа. Енкапсулацијом се скрива која база података представља извор података, тј. која компонента хибридне базе података служи за складиштење и читање конкретних података. Тиме се крајњи корисник ослобађа вођења рачуна о физичким локацијама на којима се налазе подаци, већ целокупну хибридну базу података посматра и користи као јединствену логичку базу података. За интегрисану администрацију база података хетерогених типова које чине

компоненте хибридне базе податка, простор се отвара пре свега кроз заједничко администрирање ограничења и правила интегритета, уз тежњу смањења редудансе на нивоу хибридне базе података генерално и на нивоу појединачних компоненти. Тиме се на податке из *NoSQL* база података могу применити неки сигурносни механизми *SQL* компоненте, што доприноси даљој интеграцији компоненти хибридне *SQL*/*NoSQL* базе података. Уочена могућност за даљу интеграцију постојећих *SQL* и *NoSQL* база података и њихово униформно коришћење као компоненти јединствене логичке базе података, искоришћени су за развој новог приступа за коришћење хибридне *SQL*/*NoSQL* базе података, који је описан у наредном поглављу.

# 6. Приступ за интеграцију и униформно коришћење *SQL* и *NoSQL* база података као компоненти хибридне *SQL/NoSQL* базе података

Употреба различитих система за управљање базама података, који подржавају специфичне језике за манипулисање подацима (тј. одсуство стандардизованости језика за рад са свим типовима база података) главни је узрочник ниског степена интеграције података, ускладиштених у базама података различитих типова, а које је потребно истовремено користити. На проблем интегрисаности података из база различитих типова, које се конкурентно користе, надовезује се и проблем униформног приступа тим подацима. Анализа доступних приступа, извршена у претходном поглављу, указала је да постојећи приступи на различите начине и у различитој мери нуде могућа решења наведених изазова интеграције и униформног коришћења база података различитог типа. Тиме је отворен простор за даље истраживање, кроз решавање наведених изазова интеграције и униформног коришћења база података различитих типова и у ситуацији када оне представљају компоненте хибридне *SQL*/*NoSQL* базе података. Као што је објашњено у претходном поглављу, процес интеграције и униформног коришћење база података као компоненти хибрида потребно је да обухвати и додатне специфичности, попут пружања обједињене администрације, контролисања правила интегритета и редуковања редундансе на нивоу целе логичке базе података. Те специфичности нису у потпуности узете у разматрање у постојећим приступима, што је отворило простор за развој новог приступа за интеграцију и униформно коришћење хибридне *SQL* и *NoSQL* база података као компоненти хибридне *SQL*/*NoSQL* базе података.

Хибридна *SQL*/*NoSQL* база података представља јединствену логичку базу података. Међутим, свака компонента хибридне базе података представља конкретни хетерогени тип базе података, са свим својим специфичностима (логичка организација података, структуре за складиштење података итд.), па самим тим и са сопственим језиком за рад са подацима. За рад са подацима *SQL* базе података природно је да се користи *SQL* упитни језик. Међутим, не подржавају

све *NoSQL* базе података *SQL* упитни језик, па се самим тим он не може директно применити за рад са свим компонентама хибридне базе података. Ситуација постаје додатно сложена уколико се узме у обзир да различити типови *NoSQL* база података користе различите упитне језике и да не постоји њихова стандардизованост чак ни на нивоу истог типа *NoSQL* базе података. Наведено је потребно узети у обзир приликом развоја јединственог приступа за униформно коришћење свих компоненти хибридне базе података. Хибридна база података обједињује базе података хетерогених типова, али је пројектована као јединствена логичка база података. Кроз обезбеђивање могућности да корисник на јединствени начин користи податке из база података које су различитог типа (и чине компоненте хибрида), а чијим подацима би иначе приступао употребом различитих језика, постиже се жељени ефекат униформног коришћења јединствене логичке базе података.

Сврха новог приступа је да се корисник растерети вођења рачуна о томе који се подаци налазе у којој компоненти, који језик је потребно користити за њихов приступ, као и у ком конкретном типу базе података ће наредба или њен део бити извршен. Жељени ефекат је да приликом извршавања наредби над хибридном базом података корисник не води рачуна који тип структуре је коришћен за складиштење података које претражује, нити у ком конкретном систему за управљање базом података ће упит или неки његов део бити извршен, на који начин ће бити спојени подаци из различитих извора, нити на који начин ће бити обједињени резултујући подаци. Исто као и приликом рада са једним конкретним системом за управљање базом података (нпр. *Oracle 12c EE*), који користи концепте конкретног типа базе података (*SQL* базу података), од корисника се очекује да унесе наредбу и да је изврши, након чега се кориснику приказује резултат извршавања унете наредбе. Описани поступак представља основни начин извршавања наредби над било којим системом за управљање базом података, без обзира који тип базе података конкретан систем користи. Општи поступак извршавања наредби над једним (конкретним) типом базе података се може описати следећим, упрошћеним, случајем коришћења:

1) Корисник уноси наредбу.
2) Корисник покреће извршавање наредбе.

3) Систем извршава наредбу и приказује резултат извршавања наредбе.

Прва два догађаја представљају акције корисника, док трећи представља одговор система. Под системом се генерално подразумева информациони систем, којем припада и база података одговарајућег типа (*SQL*, *NoSQL* или хибридна).

Једна од водиља приликом развоја новог приступа за коришћење хибридне *SQL*/*NoSQL* базе података била је да се кориснику омогући употреба напредних и адекватних технологија за специфичне домене примене, уз задржавање степена угодности и једноставности у раду као приликом коришћења једног типа базе података. Да би описани ефекат био постигао и у раду са хибридном базом података, догађаји са стране корисника, који су наведени у претходном случају коришћења, су задржани и у опису случаја коришћења за рад са хибридном базом података. Тиме је приликом коришћења хибридне базе података, са аспекта корисника, задржан исти степен једноставности употребе као када се користи *SQL* база података. Са друге стране, догађаји који се извршавају са стране система проширени су неопходним специфичним смерницама за приступ и коришћење података хибридне базе података, а који проистичу из жеље и потребе да се кориснику пружи осећај рада са јединственом логичком базом података. Споменуте смернице се могу формулисати на следећи начин:

- Омогућити кориснику униформни приступ и рад са свим подацима хибридне *SQL*/*NoSQL* базе података, без обзира да ли су ти подаци ускладиштени у *SQL* или *NoSQL* компоненти,
- Наредбу унету од стране корисника декомпоновати и одговарајуће делове наредби извршити над подацима у одговарајућим компонентама хибридне базе података (*SQL* или неком типу *NoSQL* базе података) применом одговарајућих језика за рад са подацима,
- Интегрисати податке из различитих компоненти у јединствени резултат извршавања наредби, који ће бити приказан кориснику употребом стандардизованог језика који корисник користи.

Приликом коришћења хибридне базе података, ефекат једноставности у раду није лако постићи и разликује се од начина на који то постижу изоловани системи за управљање базом података. Таква констатација проистиче из потребе решавања проблема интеграције и униформног рада са подацима из база података различитих

типова. Са друге стране, након решавања додатних изазова које са собом повлачи рад са хибридним базама података, предност се огледа у могућности примене адекватних и са аспекта перформанси оптималних типова база података за конкретне захтеве који су диктирани од стране специфичности домена пословања организације.

Укључивањем идентификованих смерница за рад са хибридном базом података у првобитни, општи, случај коришћења за извршавање било које наредбе у било којем типу базе података добијен је проширени случај коришћења за извршавање наредбе у хибридној бази података. Овај случај коришћења намењен је раду са хибридном базом података и укључен је и у развој новог приступа за коришћење хибридне *SQL*/*NoSQL* базе података, који је представљен у наставку дисертације. Опис проширеног случаја коришћења гласи:

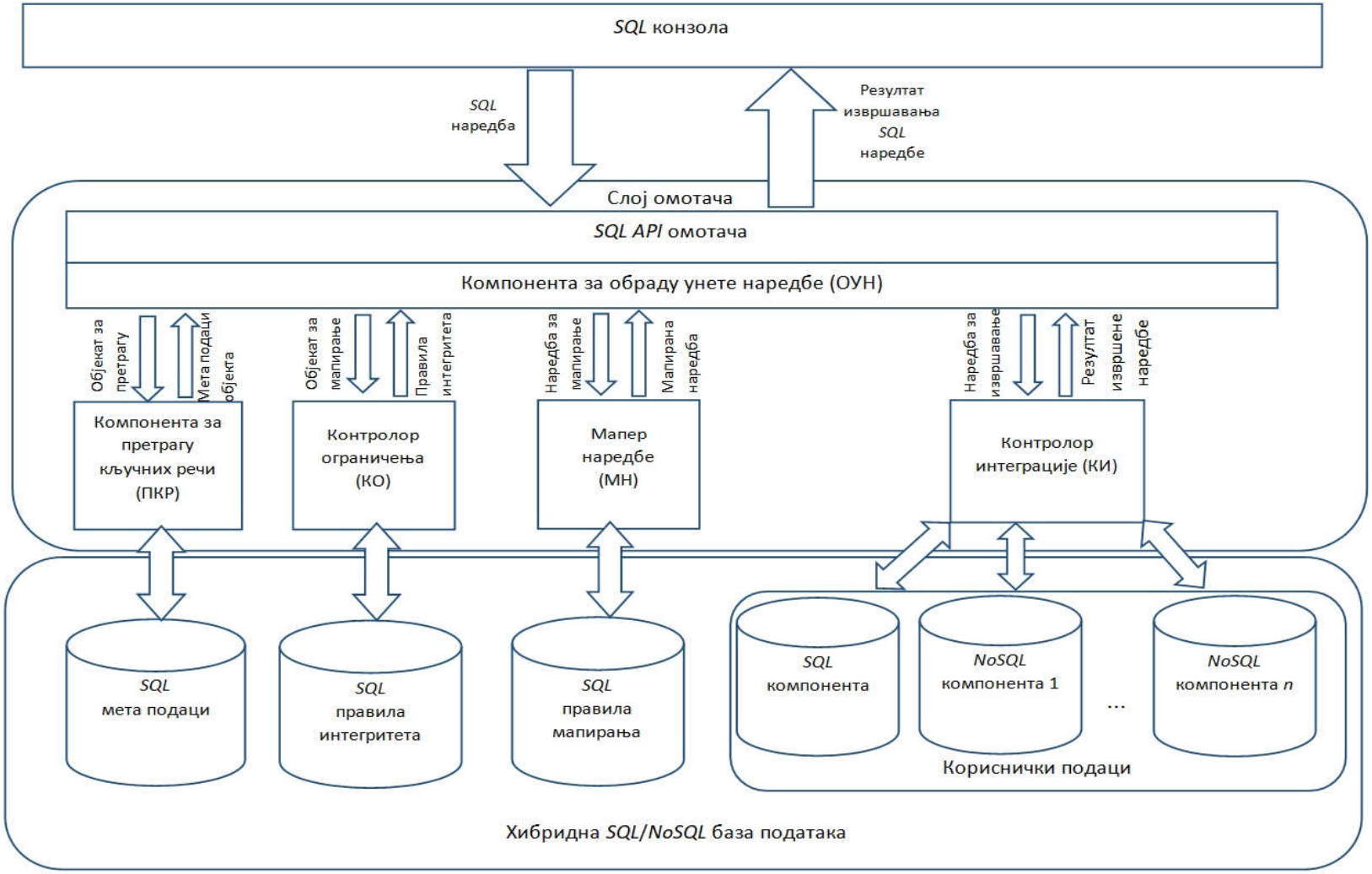
1) Корисник уноси наредбу.
2) Корисник покреће извршавање наредбе.
3) Систем декомпонује наредбу, утврђује у којим компонентама треба извршити одговарајуће делове наредбе и употребом којих језика за рад са подацима.
4) Систем врши прилагођавање изворне наредбе и извршава их на конкретном језику конкретног типа базе података.
5) Систем прихвата резултате извршавања сваке од компоненти, обједињује их, преводи у језик корисника и приказује интегрисан резултат.

Динамику реализације случаја коришћења могуће је приказати употребом неког од *UML* дијаграма за опис динамике. Међутим, пре тога, потребно је креирати референтну архитектуру. За основу архитектуре узета је широко распрострањена (и у претходном поглављу описана) трослојна архитектура која је затим проширена специфичним компонентама у циљу лакшег испуњења потреба интеграције и униформног коришћења база података различитих типова. Та новоразвијена архитектура омогућава коришћење предности сваког појединачног типа базе података који улази у састав хибридне *SQL*/*NoSQL* базе података (чинећи неку њену компоненту), уз истовремену њихову интеграцију у јединствену логичку базу података, којој се приступа и са којом корисник комуницира на јединствен начин. У циљу реализације свега наведеног осмишљене су нове компоненте архитектуре

са наменским функционалностима. У наставку је објашњена свака компонента са својим функционалностима и имплицираним ефектом на целокупну архитектуру.

## 6.1. Приказ развијене архитектуре новог приступа за униформно коришћење хибридне *SQL*/*NoSQL* базе података

У овом поглављу приказана је и описана новокреирана архитектура, која је осмишљена за потребе истовременог, интегралног и униформног коришћења свих компоненти хибридне *SQL*/*NoSQL* базе података, чиме се постиже ефекат рада са јединственом логичком базом података. Ради адекватног испуњења специфичних потреба које са собом доноси коришћење хибридне базе података, за полазну основу је узета традиционална трослојна архитектура, која је затим проширена новим компонентама. Свака компонента унапређене трослојне архитектуре поседује специфичну намену којом се проширују постојеће могућности и функционалности полазне архитектуре, са којом истовремено чини интегралну целину. Та интегрална целина традиционалне трослојне архитектуре и новододатих компоненти ће у наставку дисертације бити третирана као референтна архитектура новоразвијеног приступа за коришћење хибридне *SQL*/*NoSQL* базе података. Графички приказ новоразвијене архитектуре креиране за потребе новог приступа коришћења хибридне *SQL*/*NoSQL* базе података дат је на Слици 4. У наставку следи опис сваке нове компоненте, са наменом која јој је додељена.




Слика 4 - Приказ новоразвијене архитектуре креиране за потребе новог приступа коришћења хибридне *SQL*/*NoSQL* базе података

Први слој који ће бити описан је презентациони слој. Овај слој, као и код традиционалне трослојне архитектуре, садржи кориснички интерфејс који представља тачку приступа и коришћења система од стране корисника. Под системом ће се у даљем раду подразумевати било информациони систем, скуп апликација или конкретна апликација. У сва три случаја, заједничка карактеристика система је да поседује слој података који користи различите базе података хетерогених типова (*SQL* и *NoSQL*) интегрисане у хибридну базу података. Већ је наведено у претходним поглављима да *SQL* упитни језик представља опште прихваћени стандард за рад са најзаступљенијим типом база података (SolidIT, 2018), тј. са *SQL* базама података, па је та чињеница представљала и главну водиљу приликом одабира језика за рад са хибридном базом података. Интеракција између корисника и система, који је заснован на новој архитектури и који користи хибридну базу података, реализоваће се употребом *SQL* упитног језика. Новој архитектури додате су компоненте које анализирају и прилагођавају извршавање *SQL* наредби над свим типовима база података које улазе у састав хибридне базе података. Иако постоје различити типови корисничких интерфејса, као што је изложено у претходном поглављу, за нови приступ коришћења хибридне базе података нису од интереса њихове имплементационе специфичности, јер оне не утичу на њихову намену (прослеђивања унетих података пословној логици).

Због тога, могуће је увести одређени степен апстракције у приказу корисничког интерфејса. Тиме се поједностављује приказ истог на архитектури, уз задржавање његове функционалности. За потребе новог приступа коришћења хибридне базе података одабран је кориснички интерфејс који ради на принципу конзоле, која као улаз прихвата *SQL* наредбе. Предности одабраног типа интерфејса приликом коришћења хибридне базе података се огледају у једноставности рада и приказа, као и у флексибилности коришћења, јер не постоји ниједно ограничење у дефинисању наредби. Све *SQL* наредбе које је могуће откуцати и извршити над базом података кроз постојеће конзоле (попут *Oracle SQL Plus*-а) могуће је откуцати и кроз *SQL* конзолу нове архитектуре, што је од посебне важности приликом креирања ад-хок упита. Такође, приказ резултујућих података извршене наредбе је у складу са *SQL* стандардом, тј. приказује се на начин на који се

приказују и у другим *SQL* конзолама. Примена корисничког интерфејса који прихвата *SQL* наредбе, олакшава коришћења система, јер кориснику нису потребна нова знања нити нове вештине за писање наредби. Поред тога, применом интерфејса који прихвата *SQL* наредбе додатно се наглашава место *SQL*-а у новој архитектури и то као:

- Униформног језика хибридне базе података – са аспекта корисника омогућава коришћење података на униформни начин, без обзира да ли се приступа подацима из *SQL* или неке *NoSQL* компоненте. Корисник уноси наредбу у виду *SQL*-а и добија податке у виду резултата извршавања *SQL* наредбе, док новоразвијене компоненте врше све потребне анализе и подешавања
- Језика интеграције хибридне базе података – повезује податке из база података различитих типова и приказује их кориснику као резултат извршавања наредбе над хибридном базом података. Корисник не треба да размишља који су подаци у којој компоненти хибридне базе података, већ он приступа интегралном систему коришћењем *SQL*-а и добија интегрални резултат извршавања наредбе над целокупним системом, уместо над појединачним компонентама.

За комуникацију са презентационим слојем задужен је омотач, који се лако уочава на графичком приказу архитектуре (Слика 4). Омотач представља важну карактеристику и главну новину новоразвијене архитектуре у односу традиционалну трослојну архитектуру. У циљу реализације интеракције између корисника (који уноси *SQL* наредбе) и система, омотач пружа *SQL API (Application Programming Interface)*. *API* омотача подржава *SQL* стандард и омогућава унос наредби креираних у споменутом језику. Све *CRUD* (*Create Read Update Delete*) операције су подржане омотачевим *API*-ем. Међутим, овде се не завршавају функционалности омотача који је наменски осмишљен за потребе нове архитектуре. Поред наведеног, омотач је задужен и за управљање процесом извршавања наредбе, од тренутка када прихвати наредбу унету од стране корисника, до тренутка када се кориснику прикаже резултат извршавања те наредбе. Због тога се омотач може посматрати и као специфичан контролер који управља реализацијом целокупне интеракције између корисника и система.

Након прихватања упита омотач је задужен за његову анализу, обраду и упућивање на одредишне локације за његово извршавање. Иако је новом архитектуром омогућено коришћење база података различитих типова (*SQL* и подтипова *NoSQL*), *SQL* наредбе није могуће директно применити на сваку од обухваћених база података. Као што је већ објашњено, иако неки произвођачи *NoSQL* система за управљање базом података раде на развоју подршке за *SQL* упитни језик (Grolinger, et al., 2013) или га већ подржавају (SQLite, 2016), споменути језик није примењив на све типове *NoSQL* база података, нити на све *NoSQL* системе за управљање базом података. Примера ради, најзаступљенији *NoSQL* систем за управљање базом података *MongoDB* (SolidIT, 2018), не подржава *SQL* упитни језик, већ користи сопствени *MongoDB Query Language*. Наведено намеће потребу превођења улазне *SQL* наредбе у специфичан упитни језик конкретног система за управљање базом података (уколико одредишни СУБП не подржава *SQL*), што се не може спровести без претходног дефинисања правила превођења. Да би се то реализовало, потребно је анализирати и декомпоновати унету наредбу и утврдити који тип базе података, а затим и који систем за управљање базом података, представља извор читања жељених података, односно дестинацију уписивања, измене, или брисања података. У наставку рада ће за базе података које представљају извор читања података, односно дестинацију уписивања, измене или брисања података бити коришћен заједнички термин одредишна база података. Ситуацију додатно чини комплексном и подршка нове архитектуре за паралелни рад са различитим одредишним базама података. Уколико се упит посматра као *SQL* наредба која се често извршава над хибридном *SQL*/*NoSQL* базом података (и која представља наредбу у којој је могуће истовремено читање података из одредишних база података различитих типова, које чине компоненте хибридне базе података), могу се приметити следећи исходи у његовом извршавању:

1) Упит је потребно извршити над једном или више одредишних *SQL* база података (или над базама података које подржавају *SQL*), те није потребно вршити превођење унетог *SQL* упита,
2) Упит је потребно извршити над једном или више одредишних база података, од којих је барем једна *NoSQL* база података која не подржава

*SQL* упитни језик. У тој ситуацији неопходно је извршити превођење *SQL*-а у специфичан језик одредишне базе података која не подржава *SQL* синтаксу.

Главна разлика између упита и преосталих *CRUD* наредби (а те три наредбе су *Create*, *Update*, *Delete*) је у чињеници да је само за упите „интересантна“ могућност паралелног коришћења више различитих типова одредишних база података. Преостале три наредбе, који год тип базе података да имају за одредиште, биће извршене само над једним типом базе података. И у случају преостале три наредбе, уколико одредишна база података не подржава *SQL*, биће потребно извршити трансформацију унете наредбе у синтаксу одредишне базе података. Због комплексности која може настати приликом рада са упитима када се користи више различитих одредишних база података, које користе различите упитне језике, у наставку дисертације ће логика рада компоненти нове архитектуре бити објашњена на случају када је унета *SQL* наредба упит, који може користи податке из више различитих типова одредишних табела. Логика обраде неке од преостале три наредбе се подводи под логику обраде једноставнијег упита, када целокупан упит користи податке из само једне одредишне базе података.

У сврху детектовања типа одредишне базе података којој припадају објекти унетог упита и у сврху адекватног управљања даљим извршавањем упита (декомпоновањем истог и опционим превођењем у специфичне језике), слој омотача користи специфичне компоненте, које су као и сам омотач, наменски развијене за потребе нове архитектуре. Те компоненте су:

- Компонента за обраду унете наредбе (ОУН),
- Компонента за претрагу по кључним речима (ПКР),
- Контролор ограничења,
- Компонента за мапирање наредбе и
- Контролор интеграције.

Омотач обухвата наведене компоненте, управља комуникацијом између њих и интегрише њихов рад, па се зато може третирати и као слој, који обухвата ове компоненте, што је и приказано на  Слици 4.

### 6.1.1. Компонента за обраду унете наредбе (ОУН)

У циљу одређивања типова одредишних база података, у слоју омотача прво се анализира унети упит. Ова функционалност омотача реализује се у компоненти за обраду унете наредбе (ОУН). Поделом упита на клаузуле ОУН започиње процес „обраде“ унетог *SQL* упита. У почетној фази „обраде“ упита, од посебног значаја је *FROM* клаузула. У њој су наведени називи табела, погледа или функција који су од значаја за извршавање упита. Пошто су табеле најчешће коришћене од три наведена типа објекта (табела, поглед и функција) цео израз у *FROM* клаузули се неретко дефинише као „Израз за навођење табела“ (*TableExpression*) (Oracle, 2018b). Узимајући то у обзир, а у циљу поједностављења, у наставку текста ће *FROM* клаузула бити коришћена за навођење конкретних табела (иако ће се подразумевати да *FROM* поред табела може садржати и погледе и функције). ОУН подржава синтаксу спајања табела и то традиционалну, по којој се табеле одвајају симболом „зарез“ и синтаксу у складу са *American National Standards Institute* препоруком, по којој се за спајање табела унутар *FROM* клаузуле користи кључна реч *JOIN*. У оба случаја, ОУН је у могућности да препозна коришћену синтаксу и да из *FROM* клаузуле извуче називе табела из унетог упита. Кориснику је омогућено да пише *SQL* упите неводећи рачуна о логичкој и физичкој структури података, нити о њиховој организацији у хибридној бази података, јер употребом *SQL* синтаксе свим ресурсима свих база података приступа на униформни начин као деловима јединствене хибридне базе података. ОУН мора да прилагоди унети упит за извршавање, што је условљено типовима одредишних база података у којима ће упит и његови делови бити извршени.

Поред наведеног, специфичан изазов са којим се сусреће ОУН приликом обраде упита је могућност да један упит приступа подацима из различитих објеката, различитих СУБП-ова, односно база података различитих типова, тј. могућност да упит за дестинацију извршавања има хетерогене одредишне базе података, које припадају једној хибридној *SQL*/*NoSQL* бази података. У циљу управљања наведеним изазовом потребно је дати одговоре на следећа питања:

- Како организовати и управљати подацима о типовима објеката, типовима база података којима припадају и специфичним језицима које користе?

- У којој мери је могуће обезбедити управљање ограничењима у хибридној *SQL*/*NoSQL* бази података, сачињеној од база података хетерогених типова?
- На који начин извршити превођење унете наредбе у одговарајућу наредбу специфичног језика одредишне базе податка?
- На који начин у хибридној *SQL*/*NoSQL* бази података повезати податке из база података различитих типова (а које користе различите моделе за складиштење истих) и на који начин интегрисати добијене резултате извршавања упита над различитим компонентама хибридне базе података у интегрални резултат који треба приказати кориснику?

У циљу давања одговора на наведена питања и решавања наведених изазова из којег су питања проистекла, ОУН компонента комуницира са другим наменским компонентама, прослеђује им захтеве, обрађује враћене резултате и интегрише извршавање функционалности тих компоненти кроз управљање целокупним током реализације унете наредбе. Те новоразвијене компоненте, са којима ОУН комуницира, су:

- Компонента за претрагу по кључним речима (ПКР) – ради прибављања мета података о објектима упита (тип објекта, тип базе података у који је смештен итд.),
- Контролор ограничења (КО) – ради прибављања података о правилима интегритета која су дефинисана за објекте из упита,
- Компонента за мапирање наредбе (МН) – ради добијања преведене наредбе у специфичан језик одредишне базе података (уколико одредишна база података није *SQL* база података или база података која има подршку за *SQL* језик) и
- Контролор интеграције (КИ) – којем ће прослеђивати наредбе за извршавање и од којег ће добијати резултате извршене наредбе.

Прегледом листе наведених компоненти са којима ОУН остварује интеракцију, јасно се уочава да ОУН компонента не комуницира директно са слојем података (што је и визуелно приказано на Слици 4). Због те карактеристике, може се пронаћи аналогија између контролора пословне логике на традиционалној архитектури и ОУН компоненте новоразвијене архитектуре. Улога компоненти ПКР, КО, МП и КИ нове архитектуре је да остваре комуникацију са слојем података, прихватајући

захтеве ОУН компоненте и прослеђујући јој резултате извршавања тих захтева над базом података. Ипак, свака од ових компоненти има специфичну улогу, па су тако компоненте ПКР, КО и МП, респективно, задужене за комуникацију са речником мета података, правилима интегритета и правилима мапирања смештених у слоју података, док је контролор интеграције, према инструкцијама ОУН компоненте, задужен за извршавање наредби над различитим компонентама хибридне базе података.

### 6.1.2. Компонента за претрагу по кључним речима (ПКР)

У циљу прилагођавања упита одредишној бази података, ОУН компонента комуницира са још једном наменски развијеном компонентом нове архитектуре, која је названа компонента за претрагу по кључним речима (ПКР). Ова компонента користи податке из речника мета података, који је за потребе нове архитектуре додатно проширен. Поред уобичајених података о објектима, речник мета података у новој архитектури садржи и специфичне податке о објектима хибридне базе података:

- Тип објекта (табела, поглед, документ, парови кључ-вредност, чворови графа итд.)
- Тип базе података којем припада (*SQL* или неки конкретан подтип *NoSQL* базе података: база података заснована на кључ-вредност пару, на документу, на фамилији колона или на графу)
- Конкретан СУБП којем припада (*Oracle*, *PostgreSQL*, *MongoDB* и слично)

Овакав речник елиминише потребу да корисник хибридне базе података зна или води рачуна о томе како су ускладиштени подаци, које структуре се користе за њихово смештање, у којем типу базе података се налазе подаци, односно у којем конкретном СУБП-у. Корисник уноси *SQL* наредбу на начин као да сви жељени подаци припадају табелама јединствене *SQL* базе података. Речник мета података садржи потребне информације о организацији података, чиме је управо омогућено да корисник, куцајући само *SQL* упит, на униформни начин приступа свим подацима хибридне базе података, без обзира да ли се жељени подаци налазе у једној или више компоненти хибрида. Да би се подаци из речника искористили,

када дође време за евентуалну трансформацију унетог упита и његових саставних делова у специфичан језик конкретног СУБП-а, потребно је да се изврши читање података из речника, тј. потребно је да се за називе табела из иницијалног *SQL* упита учитају одговарајући подаци из речника мета података хибридне базе. За то је задужена компонента за претрагу по кључним речима.

Компонента за претрагу по кључним речима је компонента слоја омотача која представља спону између компоненте за обраду унете наредбе и речника мета података хибридне базе података. ОУН компонента детектује кључне речи, које представљају препознате називе табела из *FROM* клаузуле унетог *SQL* упита и затим их прослеђује ПКР компоненти. ПКР компонента користи кључне речи и по њима претражује речник мета података. Мета податке (тип објекта, тип базе података и конкретан СУБП) добијене за сваку кључну реч, тј. за назив сваке табеле, компонента ПКР враћа ОУН компоненти, која анализира враћене мета податке и доноси одлуку да ли је неопходно трансформисати унети упит у језике одредишних база података. Узимајући у обзир два могућа исхода приликом извршавања унетог упита, ОУН користи следеће смернице у реализацији сваког упита:

1) Уколико се упит извршава над *SQL* одредишном базом података, такав упит неће бити превођен у специфичан језик одредишне базе података. Пре него што се изврши упит, биће проверена правила интегритета. Уколико је потребно, упит ће евентуално бити допуњен одговарајућим критеријумима филтрирања или ће бити изгенерисан нови подупит,
2) Уколико се упит извршава над неким типом *NoSQL* базе података, унети *SQL* упит је потребно превести у специфичан језик одредишне базе података, али ће пре мапирања и извршавања упита бити проверена правила интегритета. Уколико буде постојала потреба, упит ће евентуално бити допуњен одговарајућим критеријумима филтрирања или бити изгенерисан додатни упит.

### 6.1.3. Контролор ограничења (КО)

Контролор ограничења задужен је за проверу ограничења и услова спајања објеката из база података различитих типова, које припадају истој хибридној бази података. Основу повезивања објеката, независно од компоненте хибридне базе података којој припадају, чине правила интегритета која важе за било који релациони модел, тј. било коју *SQL* базу података (Lazarević et al., 2006):

- Правило интегритета ентитета, којим се дефинише да ниједан атрибут примарног кључа не може имати непознату (*null*) вредност и
- Правило референцијалног интегритета којим се дефинише начин повезивања примарног и спољног кључа.

Ова правила, карактеристична за *SQL* базу података, изабрана су као основна ограничења хибридне *SQL*/*NoSQL* базе података јер *SQL* база података, која представља значајни део хибрида, у начелу пружа виши степен контроле ограничења од *NoSQL* базе података (Nance, et al., 2013).

Прилагођавањем правила интегритета ентитета и правила референцијалног интегритета потребама хибридне *SQL*/*NoSQL* база података, формулисане су смернице њихове практичне примене. Оне се огледају у дефинисању два начина реализације споменутих правила у хибридној бази података. Сваки објекат хибрида, без обзира у којем типу базе података се налази, повезан је са другим објектима исте или других база података на барем један од два начина:

- Објекат садржи референцу на један или више других објеката преко референцирајућих атрибута (спољних кључева или атрибута који имају ту улогу), без обзира у којој компоненти хибридне базе података се ти објекти налазе и
- Идентификатор објекта (примарни кључ табеле или атрибут другог типа објекта који има ту улогу) је референциран од стране барем једног објекта из исте или неке друге базе података, која представља компоненту хибридне базе података.

Повезивање објеката унутар *SQL* базе података не представља проблем, због подршке примене правила интегритета у *SQL* типу базе података. Међутим, код повезивања са објектима који припадају *NoSQL* типовима база података, јавља се

нови изазов, јер *NoSQL* базе података не подржавају директно правила интегритета *SQL* база података. Улога у премошћивању наведеног изазова додељена је ОУН компоненти, која ће задовољење правила интегритета остварити уз помоћ наменски развијене компоненте назване контролор ограничења (КО). За ОУН компоненту од значаја је да располаже информацијама о објектима и њиховим пропертијима која омогућавају да се задовоље правила интегритета, а које ће из наменске табеле базе података (`PRAVILA_INTEGRITETA`) добијати од КО компоненте. За сваки објекат и за сваку везу коју тај објекат гради са другим објектом у моделу дефинише се по један запис у табели `PRAVILA_INTEGRITETA` који садржи следеће податке:

1) Назив објекта (јединствен на нивоу хибридне базе података),
2) Тип правила интегритета: Правило интегритета ентитета („`INTEGRITET_ENT`“) или правило референцијалног интегритета („`REF_INTEGRITET`“),
3) Идентификатор објекта: колона примарног кључа за *SQL* базе података или одговарајуће поље које ће преузети ту улогу код објеката *NoSQL* база података, а које не сме имати *null* вредност,
4) Рефенцирајући проперти: колона спољног кључа за *SQL* базе података или одговарајуће поље које ће преузети ту улогу код објеката *NoSQL* база података,
5) Назив објекта који се референцира правилом референцијалног интегритета,
6) Назив колоне или поља референцираног објекта.

Ставком 3) описане структуре записа задовољено је правило интегритета ентитета (јер примарни кључ не може имати *null* вредност), док је ставкама 3) – 6) омогућена примена правила референцијалног интегритета. У случају евидентирања правила интегритета ентитета (Табела 3), колона 2) има вредност „интегритет ентитета“, колона 3) се попуњава одговарајућим називом идентификатора, а колоне 4) – 6) се не попуњавају. Запис описане структуре се уноси у табелу `PRAVILA_INTEGRITETA`, без обзира да ли је у питању објекат *SQL* базе података (табела) или објекат неког подтипа *NoSQL* базе података.

Табела 3 - Изглед табеле *PRAVILA_INTEGRITETA* и одговарајућег записа за објекат *TRANSAKCIJA* (правило интегритета ентитета)

| *NAZIV_OBJEKTA* | *TIP_PRAVILA* | *IDENTIFIKATOR_ OBJEKTA* | *REFERENCIRAJUCI _PROPERTI* | *REFERENCIRANI _OBJEKAT* | *REFERENCIRANI _PROPERTI* |
|---|---|---|---|---|---|
| *TRANSAKCIJA* | *INTEGRITET_ENT* | *KOMPANIJA_ID*<br><br>*TRANSAKCIJA_ID* | (*null*) | (*null*) | (*null*) |

У другом случају (Табела 4), када се у табели *PRAVILA_INTEGRITETA* евидентира правило референцијалног интегритета, запис табеле садржи вредности и за колоне 4) – 6). Поновљена вредност назива идентификатора објекта у више слогова за објекат који садржи више правила интегритета представља контролисану редундансу и омогућава да једна табела базе података садржи записе о оба типа правила.

Табела 4 - Изглед табеле *PRAVILA_INTEGRITETA* и одговарајућег записа за објекат *TRANSAKCIJA* (правило референцијалног интегритета)

| *NAZIV_OBJEKTA* | *TIP_PRAVILA* | *IDENTIFIKATOR_ OBJEKTA* | *REFERENCIRAJUCI _PROPERTI* | *REFERENCIRANI _OBJEKAT* | *REFERENCIRANI _PROPERTI* |
|---|---|---|---|---|---|
| *TRANSAKCIJA* | *REF_INTEGRITET* | *KOMPANIJA_ID*<br><br>*TRANSAKCIJA_ID* | *KOMPANIJA_ID* | *KOMPANIJA* | *KOMPANIJA_ID* |

Овде је важно још једном напоменути да за сваки објекат који гради везе са једним или више других објеката, табела *PRAVILA_INTEGRITETA* садржи по један запис по свакој од тих веза, што значи да се назив објекта може поновити у више записа, било као објекат који референцира или који је референциран. У Табели 4 дато је правило референцијалног интегритета између две табеле *SQL* базе података. На исти начин се евидентирају и правила референцијалног интегритета између два објекта који припадају базама података различитих типова, што ће бити приказано на конкретном примеру у наредном поглављу.

С обзиром на то да су наведени подаци јасно структурирани и да је потребно обезбедити њихову конзистентност и трајност, природна солуција је да они буду смештени у тип базе података који је прилагођен за рад са структурираним подацима и који подржава наведене особине трансакција, тј. природно је да *PRAVILA_INTEGRITETA* буде табела у *SQL* бази података. За потребе смештања

наведених података о правилима интегритета нова архитектура омогућава било креирање наменске инстанце *SQL* базе података или доделу одређеног простора постојеће *SQL* базе података. Креирање наменске инстанце *SQL* базе података представља погоднију солуцију, јер раздваја складиштење података о правилима интегритета од корисничких података, чиме је за сваку од ове две категорије података омогућено независно администрирање, флексибилније управљање привилегијама и олакшан опоравак.

Слична ситуација је и по питању правила мапирања и речника мета података. Наиме, правила мапирања и мета подаци такође поседују релативно стриктну структуру, ретко промењиву. Због тога их је, такође, погодно чувати у инстанцама *SQL* база података, по могућству одвојеним од корисничких података (по препоруци већ образложеној код правила интегритета). Узимајући наведено у обзир, на Слици 4 три наменска репозиторијума (правила мапирања, речник мета података и правила интегритета), који су типа *SQL* база података, су приказана као саставни део слоја података, али истовремено и као засебне инстанце *SQL* база података, чији су подаци одвојени од корисничких података. На графичком приказу означени су и комуникациони канали између компоненти омотача и слоја података, тј. јасно је назначено која компонента комуницира са којим репозиторијумом (ПКР са речником мета података, компонента за мапирање наредбе са правилима мапирања, а контролор ограничења са правилима интегритета).

Приликом извршавања упита, а ради задовољења правила интегритета, потребно је упоредити да ли су унете вредности спољних кључева референцирајућег објекта у складу са вредностима примарних кључева референцираних објеката. Пошто подаци могу бити смештени у различитим типовима база података, није могуће директно користити постојеће типове ограничења *SQL* база података, већ је та контрола остварена кроз функцију ОУН компоненте и кроз податке из `PRAVILA_INTEGRITETA`, које ОУН компонента добија од КО. На основу података из табеле `PRAVILA_INTEGRITETA`, приликом уноса и ажурирања података проверава се да ли су унете вредности у складу са постојећим вредностима референцираног објекта и да ли је за идентификатор објекта унета вредност (јер не сме бити *null*). Приликом извршавања упита који приступају подацима из различитих објеката, при чему објекти могу припадати различитим

типовима база података, ОУН проверава податке из *PRAVILA_INTEGRITETA*, такође добијене од КО. Функција ОУН којом се спроводе правила интегритета су:

- Генерисање додатног упита или
- Модификација постојећег упита.

Додатни упити ће бити изгенерисани ради провере конкретних вредности које морају бити у складу са правилима референцијалног интегритета. Примера ради, када се уносе подаци у објекат који референцира неки други објекат, ОУН ће изгенерисати упит којим ће проверити да ли унета вредност пропертија који има улогу спољног кључа одговара некој вредности пропертија који има улогу примарног кључа у шифарнику референцираног објекта. Модификација упита се спроводи када је потребно повезати податке из различитих објеката, који припадају базама података различитих типова. Највећи изазов у том случају представља *JOIN* клаузула. Пошто се она не може директно применити на објекте из база података различитих типова, ОУН дели унети упит на мање упите, при чему сваки од тих упита има једну одредишну базу података. ОУН компонента од КО добија податке о правилима интегритета сваког од тих објеката и на основу тога генерише нове клаузуле тих новодобијених упита (насталих дељењем иницијалног). Тако на пример, уколико је потребно приказати податке из *SQL* табеле која референцира неки документ *NoSQL* базе података, ОУН компонента ће модификовати део *SQL* упита који се извршава над *SQL* базом података, на начин да се у њему филтрирају они записи који имају одговарајући пар из упита који ће бити извршен над *NoSQL* базом података. На тај начин ОУН задовољава правила интегритета, добијена од стране КО компоненте. Практични примери примене новог приступа и његових компоненти на примерима из праксе приказани су наредном поглављу. У наредном поглављу су дати и прикази међу-корака у реализацији неколико случајева коришћења, са објашњењима.

### 6.1.4. Мапер наредбе (МН)

За прилагођавање и превођење корисничког *SQL* упита у специфичан језик конкретног СУБП-а ОУН комуницира са мапером наредбе (МН). Овој комуникацији претходи комуникација између ОУН и КО компоненти. Након

анализираних правила интегритета које је ОУН компонента прибавила од КО, ОУН „се обраћа“ компоненти МН. МН компонента, као што њен назив сугерише, користи правила мапирања, тј. правила превођења. Правила мапирања су саставни део слоја података и садрже правила којима се упити остале наредбе, тј. наредбе за унос, брисање и ажурирање података, трансформишу у синтаксу језика одредишне базе података. Правила мапирања су дефинисана у зависности од типа наредбе, типа базе података и језика који користи конкретан СУБП. За представнике различитих типова база података, који не подржавају *SQL* упитни језик, дати су примери мапирања *SQL* наредби у специфичне језике које користе. Приказ примера правила мапирања за неке СУБП-ове дати су у Табели 5.

Табела 5 - Правила мапирања наредби између језика представника база података различитих типова

| Језик | *SQL* (и *SQLite*) | *SPARQL* | *Cypher* | *CQL* | *UnQL* | *Mongo QL* | *REDIS QL* |
|---|---|---|---|---|---|---|---|
| Модел | Релациони | Граф | Граф | Фамилија колона | Документ | Документ | Кључ-вредност |
| Унос података | *INSERT* | *INSERT* | *SET* | INSERT | *INSERT* | *insertOne()* *insertMany()* | *SET* |
| Ажурирање података | *UPDATE* | *MODIFY* | *SET* и *FOREACH* | *UPDATE* | *UPDATE* | *updateOne()* *updateMany()* | *SET* |
| Брисање података | *DELETE* | *DELETE* | *REMOVE* | *DELETE* | *DELETE* | *deleteOne()* *deleteMany()* | *DEL* |
| Приказ података | *SELECT* | *SELECT* | *RETURN* | SELECT | *SELECT* | *find()* | *GET* |
| Извор | *FROM* | *(FROM) URI* | *MATCH* | FROM | *FROM* | .колекција | / |
| Филтрирање | *WHERE* | *WHERE* / *FILTER* | *WHERE* | *WHERE* (инд.кол.) | *WHERE* | *{field1:value1}* | / |
| Спајање | *JOIN* | Преко патерна | *MATCH* (приближно) | / | / | *$lookup* „ . “ (за *nested*) | / |

У табели 5 су приказане најчешће коришћене наредбе (тзв. *CRUD* операције). Ове операције су објашњене у претходном поглављу ове дисертације, у којем је

извршена и компаративна анализа језика којима припадају. У табели 5 су наведене *SQL* наредбе и клаузуле од значаја за реализацију *CRUD* операција, али и за постављање сложенијих упита, попут оних са *JOIN* и *WHERE*. Прегледом наредба уочава се највећи степен повезаности између *UnQL* и *CQL* са једне стране и *SQL*-а са друге стране, што директно утиче на сложеност процеса мапирања и време потребно за његову реализацију. Одсуство *JOIN* клаузуле у *NoSQL* базама података не представља велики недостатак чак се може рећи и да представља својство које природно произилази као последица структура података које базе података овог типа користе за складиштење података. Међутим, ова карактеристика отежава интеграцију са *SQL* базама података. Због тога, приликом извршавања упита који за одредишне базе података имају и *SQL* и *NoSQL* компоненте хибрида, резултујући подаци, уколико нема додатних услова, биће прво селектовани из *NoSQL* компоненте (извршавањем дела упита који се тиче те компоненте, а који прави ОУН), а затим ће ОУН исте заменити у *WHERE* клаузулу упита који се извршава над *SQL* компонентом, поштујући правила интегритета. Тиме се смањује количина наредби потребна за мапирање. Наведени принцип биће примењен у примерима који следе у наставку.

Функционисање компоненте за мапирање наредбе се може описати на следећи начин: ОУН прослеђује овој компоненти упит, назив објекта, тип објекта, тип базе и конкретан СУБП. На основу улаза, компонента која садржи правила мапирања преводи улазни упит, тако што из предефинисаних правила учитава начин трансформације сваке од клаузула и прилагођава специфичности (попут оператора и слично) одредишној бази података. Имајући у виду да се помоћу компоненте за мапирање наредбе дефинише нова синтакса упита, али да се у њој не извршавају сами упити, ова компонента ће након трансформације вратити „преведени“ упит ОУН компоненти. Описана комуникација којом компонента за мапирање наредбе враћа модификован упит ОУН компоненти додатно наглашава контролорску улогу коју ОУН компонента има током целокупног процеса рада са унетом наредбом, од тренутка њеног прихватања, до тренутка приказа резултата кориснику.

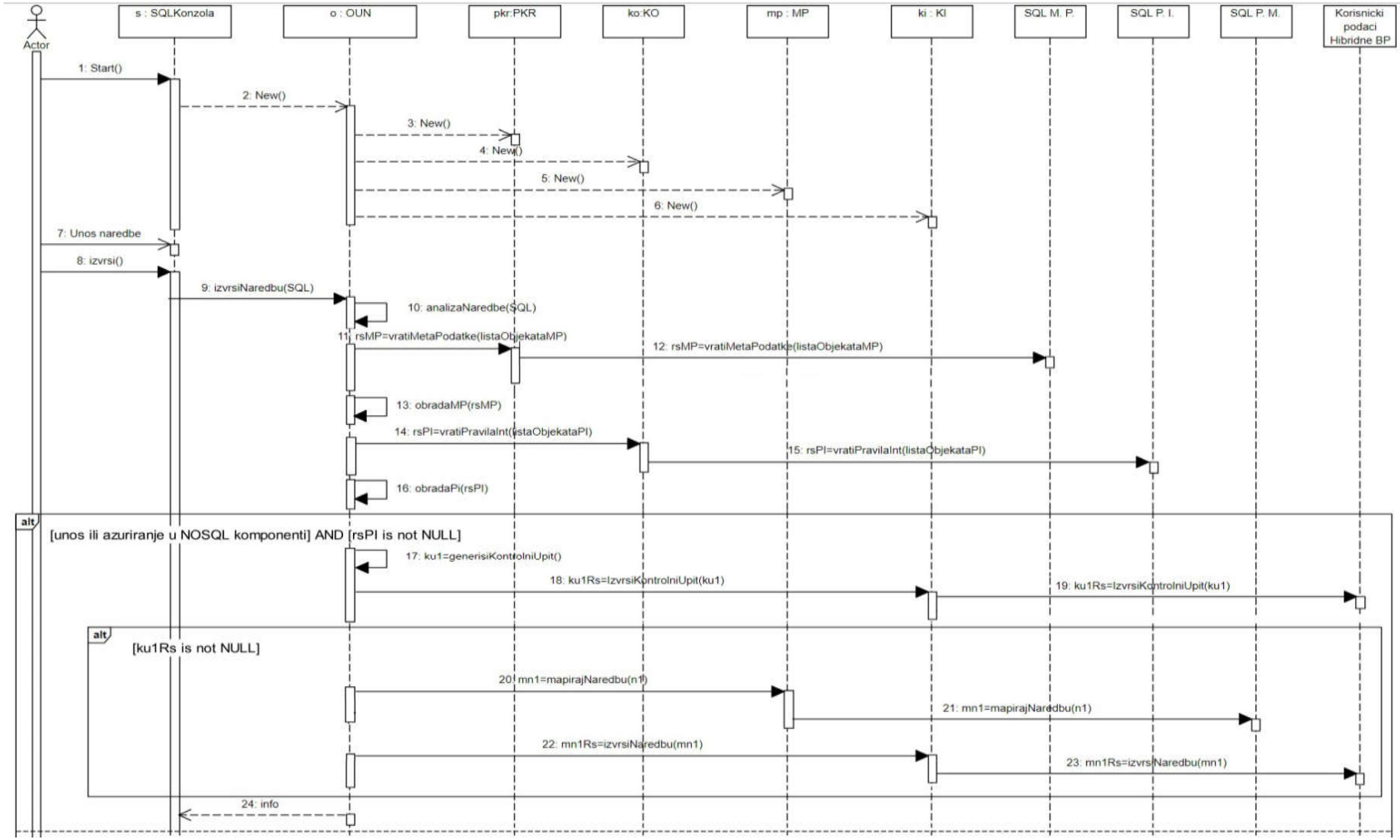
### 6.1.5. Контролор интеграције (КИ)

Контролор интеграције (КИ) је компонента нове архитектуре задужена за извршавање наредби над хибридном базом података. Наредбе које ће бити извршавана над компонентама хибридне базе података КИ добија од ОУН компоненте. КИ успоставља конекцију над одредишним базама података (уколико она није отворена) и извршава одговарајуће наредбе. С обзиром на то да ОУН компонента не комуницира са слојем података, управљање извршењем саме трансакцијом над базом података препуштена је контролору интеграције. Ипак овде треба нагласити да је за управљање целокупним процесом реализације наредбе, од тренутка уноса наредбе од стране корисника до тренутка приказа резултата кориснику, задужена ОУН компонента. То се потврђује и приликом анализе интеракције између ОУН и КИ компоненти. КИ извршава наредбе над базом података добијене од стране ОУН, којој враћа резултат извршавања истих. Које наредбе ће бити извршене над којом компонентом хибридне базе података одлучује ОУН. Такође, након извршених наредби над базом података, резултате враћене од стране КИ компоненте обрађује ОУН компонента. ОУН анализира резултате, по потреби форматира и врши спајање резултата извршавања наредби (типично упита) над различитим компонентама хибридне базе података у интегрални резултат, који ће кориснику бити приказан у форми извршавања *SQL* наредбе из *SQL* конзоле. Тиме је и заокружен целокупан процес реализације наредбе над хибридном базом података. Приказ примене новог приступа коришћења хибридне *SQL*/*NoSQL* базом података на примеру пет наредби, различите сложености и са различитим одредишним базама података приказан је и објашњен у следећем поглављу.

## 6.2. Динамика примене новог приступа за униформно коришћење хибридне *SQL*/*NoSQL* базе података

Динамику примене новог приступа за униформно коришћење хибридне *SQL*/*NoSQL* базе података могуће је приказати употребом наменских дијаграма. У овом поглављу дисертације већ је дат опис општег случаја коришћења у извршавању *SQL* наредбе над хибридном *SQL*/*NoSQL* базом података, као и детаљна спецификација наменски развијене архитектуре за потребе новог приступа. Након спецификације описа случаја коришћења, природан корак у приказу динамике његове реализације представља израда неког од *UML* дијаграма за опис динамике. У ту сврху изабран је *UML* дијаграм секвенци, који служи за приказ комуникације између различитих компоненти референтне архитектуре, уз наглашавање типова порука, услова размене и временских тренутака у којима до размене порука долази. Дијаграм секвенци новог приступа приказан је на Слици 5 и Слици 6.

На дијаграму секвенци се уочавају актер и компоненте архитектуре. Свака компонента је представљена инстанцом одговарајуће класе у формату *nazivInstance*:*NazivKlase* (*oun*:*OUN*, *ki*:*KI* итд.). Дијаграм секвенце поседује одређена специфична проширења. Наиме, осим компоненти презентационог и слоја пословне логике референтне архитектуре, на Слици 5 и 6 је приказан и слој података, са његовим саставним деловима. Овакав приказ (и одступање од устаљеног изостављања слоја података на дијаграму секвенци) погодни су у случају хибридне *SQL*/*NoSQL* базе података ради приказа саставних компоненти хибридне базе података, а за чиме нема потребе приликом приказа система који користе само један тип базе података. Поред тога, описане компоненте, тј. репозиторијуми за складиштење специфичних података (мета подаци о објектима хибрида, правила интегритета и правила мапирања), а за које је препоручено да буду засебне у односу на корисничке податке, такође се уочавају на оваквом, детаљнијем, приказу дијаграма секвенци.




Слика 5 - Дијаграм секвенци (део 1/2) извршавања наредбе применом новог приступа коришћења хибридне базе података


Слика 6 - Дијаграм секвенци (део 2/2) извршавања наредбе применом новог приступа коришћења хибридне базе података

Интеракцију између система и корисника започиње корисник, након чега се креирају одговарајуће инстанце компоненти, неопходне за наставак реализације случаја коришћења уопштено. Након уноса *SQL* наредбе у конзолу, корисник позива методу конзоле за извршавање унете наредбе. Комуникација између корисника и система се одвија путем конзоле, која ће у овом случају проследити унету *SQL* наредбу компоненти која има улогу контролора пословне логике, а то је компонента за обраду унете наредбе (ОУН). Након анализе и декомпозиције унете наредбе по клаузулама и кључним речима, што је на дијаграму приказано тако да ОУН позива своју методу (*analiziraNaredbe(sql)*), ОУН компонента прослеђује кључне речи (тј. вредности назива идентификованих објеката унетог упита) компоненти ПКР, која затим претражује одговарајући репозиторијум мета података објеката. Након претраге, ПКР компонента добијене резултате враћа ОУН компоненти. Сличан ток комуникације се одвија и у следећем кораку, између ОУН и КО компоненте. На основу захтева ОУН компоненте, контролор интеграције претражује податке о правилима интегритета за прослеђене називе објеката и резултате претраге враћа ОУН компоненти.

У зависности да ли је *SQL* наредба упит или као друга опција унос или ажурирање података, ОУН ће предузети следеће кораке. Уколико је у питању унос или ажурирање података, а при томе је КО компонента вратила правила интегритета чију задовољеност треба проверити у контексту вредности које је корисник проследио конзоли за неку од две наведене наредбе, ОУН компонента ће генерисати контролни упит. Контролним упитом ће бити проверено да ли су унете вредности пропертија објекта који имају улогу спољног кључа у складу са већ постојећим вредностима пропертија референцираног објекта, а који има улогу примарног кључа. Контролни упит ће ОУН компонента проследити КИ компоненти, која ће исти извршити над одговарајућом компонентом. Уколико контролни упит у шифарнику не пронађе одговарајуће поклапање са вредношћу пропертија који има улогу примарног кључа, корисник ће бити обавештен о невалидном уносу вредности. У супротном, сценарио (унос или ажурирање, са постојећим правилима интегритета) се наставља са извршавањем мапирањем и извршавањем наредбе, уколико је одредишна наредба *NoSQL* наредба за унос или ажурирање података, односно само извршавањем наредбе, уколико је одредишна

наредба *SQL* наредба за унос или ажурирање података. Под одредишном наредбом се подразумева наредба која се, након евентуалних провера и мапирања, извршава над одредишном базом. Да ли ће нека *SQL* наредба, унета од стране корисника, имати за одредишну наредбу *SQL*, *NoSQL* или обе (укључујући и подтипове *NoSQL*-а) зависи од типова идентификованих објеката и типова база података у којима су смештени ти објекти иницијалне наредбе.

У случају када је унета *SQL* наредба типа упит или уколико за објекте унетих наредби за унос или ажурирање података не постоје дефинисана правила референцијалног интегритета, сценарио се извршава на начин који је описан у наредном *ALT* блоку дијаграма секвенци. За наведене услове проверава се тип одредишне наредбе, а који је утврђен на основу типа идентификованог објекта из корисничке наредбе. Уколико је та одредишна наредба, уз већ наведене услове који морају бити задовољени (унос или ажурирање без правила референцијалног интегритета или упит), типа *NoSQL* спроводи се мапирање, које на основу захтева ОУН компоненте реализује компонента за мапирање наредбе. Након тога следи извршавање наредбе од стране контролора интеграције. Компонента КИ ће извршити мапирану наредбу над конкретним подтипом *NoSQL* базе података, односно над *SQL* базом података, уколико наредба није мапирана.

Последњи услов у реализацији овог општег случаја коришћења представља провера да ли унета наредба (тј. конкретно упит) садржи *JOIN* клаузулу. Уколико је то случај, све док постоје објекти са којима је потребно повезати друге објекте, ОУН модификује делове упита који се извршавају. ОУН компонента то ради тако што резултате извршавања првог упита (приликом првог спајања, а касније резултат извршавања претходне наредбе) повезује са одговарајућом клаузулом филтрирања података наредног упита (*WHERE* клаузула за *SQL* упит). Завршетком овог опционог сегмента са понављањем (само уколико у иницијалном упиту фигурише спајање објеката) подаци се приказују кориснику, чиме је и завршен случај коришћења.

# 7. Примери примене предложеног приступа за пројектовање и коришћење хибридне *SQL/NoSQL* базе података и приказ остварених резултата

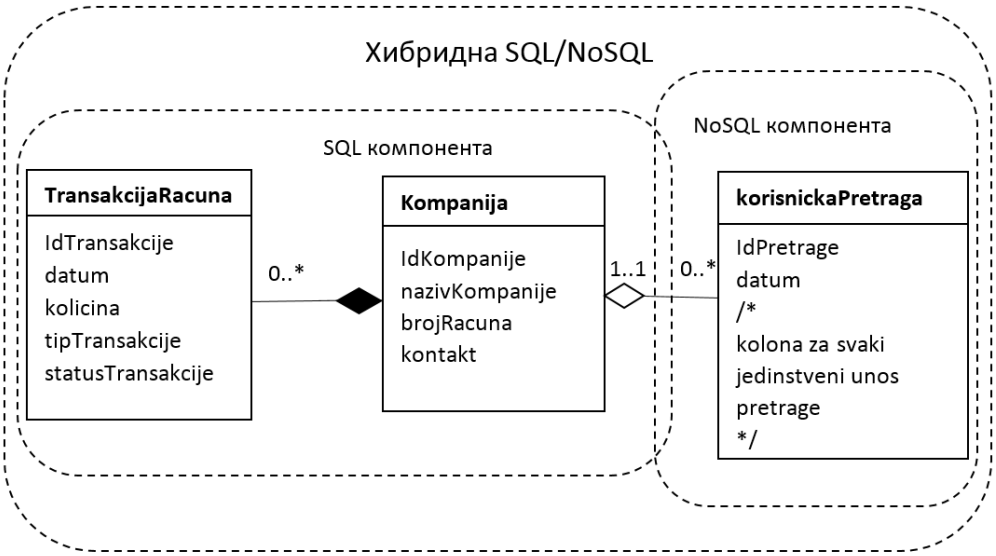
У овом поглављу биће приказани примери практичне примене новоразвијеног приступа за пројектовање и коришћење хибридне *SQL/NoSQL* базе података. На изабраном примеру из праксе примењен је нови приступ пројектовања хибрида. Након тога је извршена компаративна анализа постојеће *SQL* базе података коју организација из примера већ користи, са хибридном базом података креираном применом новог приступа, што је објављено и у раду (Bjeladinovic, 2018). За компаративну анализу изабрано је време извршавања наредби као један од показатеља перформанси.

Друга целина овог поглавља представља приказ примене новог приступа за интеграцију и униформно коришћење *SQL* и *NoSQL* база података, а који се заснива на употреби хибридне *SQL/NoSQL* базе података. За приказ новоразвијеног приступа интеграције и униформног коришћења база података хетерогених типова одабрано је пет случајева коришћења, који су обухватили извршавање наредби два типа, различите сложености, над хибридном *SQL/NoSQL* и „традиционално“ пројектованом *SQL* базом података.

## 7.1 Примена приступа за пројектовање хибридне *SQL/NoSQL* базе података и приказ остварених резултата

За потврду оправданости креирања и увођења хибридне *SQL*/*NoSQL* базе података, као и за приказ погодности примене предложеног приступа за њено пројектовање у реалном животу, искоришћен је пример из праксе. Одабран је угоститељски објекат, конкретно један ресторан, који поседује информациони систем заснован на *SQL* бази података. Поред уобичајених захтева из домена пословања ресторана, руководство објекта је пред свој информациони систем (и базу података коју он користи) поставило и нови нефункционални захтев. То је био повод за проверу оправданости даљег коришћења *SQL* базе података, односно за проверу оправданости увођења хибридне *SQL*/*NoSQL* базе података, за чије пројектовање

би био искоришћен нови приступ, описан у овој дисертацији. За потребе тестирања новог приступа у пракси, на Слици 3 приказан је део концептуалног *UML* дијаграма класа за управљање односом са купцем (*Customer Relationship Management* - *CRM*). Коришћена је техника апстракције у циљу елиминисања класа и атрибута који нису релевантни за пример. Нови приступ пројектовања хибридне *SQL*/*NoSQL* базе података примењен је на модел података са Слике 3.




Слика 7 - *UML* дијаграма класа за део концептуалног модела *CRM*-а ресторана

Ресторан води евиденцију о пословним партнерима (правна лица, тј. компаније) са основним атрибутима (`KOMPANIJA_ID`, `NAZIV_KOMPANIJE`, `BROJ_RACUNA`, `KONTAKT`) приказаним у одговарајућој класи на Слици 7. По правилу, са компанијама које су пословни партнери ресторана, ресторан има важећи уговор за услуживање запослених те компаније. *CRM* подсистем је повезан и са другим подсистемима информационог система, што је на дијаграму симболично приказано везом са класом `TRANSAKCIJA_RACUNA`. Трансакције су идентификационо и егзистенцијално зависне од класе `KOMPANIJA`, која може имати више трансакција. Поред тога, поставља се нови захтев пред постојећи информациони систем. Потребно је водити евиденцију и о свим претрагама корисника, запосленим у конкретној компанији. Компаније од интереса су само оне које су уједно и

пословни партнери посматраног ресторана. За сваког корисника (мора бити запослен у компанији која је уједно и пословни партнер ресторана), евидентира се да ли је на Интернету претраживао конкретан ресторан и ако јесте које је кључне речи користио у претрази. Споменуте податке прикупља интернет страница ресторана. Подаци се смештају у `PRETRAGE_KORISNIKA`, где се за сваку претрагу води евиденција који корисник ју је извршио и коју комбинацију кључних речи је при томе користио. Ово омогућава маркетинг одељењу ресторана да анализира који корисници су претраживали њихов ресторан (уз претходно добијену сагласност компаније, пословног партнера, у којој раде корисници) и које кључне речи су користили у претрази. За кориснике компаније који нису били подстакнути да претражују конкретан ресторан (ниједна претрага тог корисника није евидентирана у претрагама корисника), додатни промоциони канали се користе у циљу њиховог анимирања. Специфичност новог захтева огледа се у недостатку стриктне структуре речи по којима се ресторан претражује, као и у одсуству предефинисаних комбинација тих речи. Поред тога, исте речи претраге или њихове комбинације могу бити поновљене од стране различитих корисника. У циљу провере оправданости имплементације новог захтева у постојећој *SQL* технологији, односно увођења хибридне *SQL*/*NoSQL* технологије за део система који ће имплементирати нови захтев, коришћен је нови приступ пројектовања хибридне *SQL*/*NoSQL* базе података, детаљно објашњен у претходним поглављима дисертације.

На узорку претрага, који је прикупљен пре анализе оправданости пројектовања хибридне *SQL*/*NoSQL* базе података за конкретан ресторан, утврђено је да је од почетка праћења статистике сајт био претражен 51.343 пута. Евидентирано је на том узорку 52 различита уноса који садрже одређене карактеристичне речи (нпр. ресторан, кафана, бар, радно време, недеља, јеловник, цене, паркинг, прославе, музика, амбијент, *Instagram*, *Facebook*, центар града, карта пића, свадба и слично) или њихову комбинацију. Да би се подаци добијени из узорка укључили у креиране тестове, изабране су две расподеле: линеарна и *Pareto*.

Линеарни тренд показује како са повећањем броја претрага линеарно расте и број нових речи које се уносе у претрагу. Погодан за стицање опште слике колико

ће повећање броја претрага утицати на време извршавања наредби, постепеним и уједначено учесталим увођењем нових речи претраге. За разлику од тога, *Pareto* принцип дефинише однос 20:80 и неретко се користи у истраживањима чији домен су информациони системи и технологије (Ebert, 1996; Huang, & Nie, 2010; Zhang, 2008). У конкретном примеру *Pareto* принцип полази од претпоставке, добијене искуствено, прикупљањем података о претрагама корисника у претходном периоду, да ће у првих 20% анализираних претрага, над којима се прикупља статистика базе података, бити 80% различитих речи претраге или њихових комбинација, док ће се у преосталих 80% претрага појавити преосталих 20% речи претраге. *Pareto* принцип дочарава колико првих 20% претрага могу бити интензивни по питању уношења нових речи претраге. За разлику од линеарног тренда, који акценат ставља на уједначено увећање броја претрага, *Pareto* принцип потенцира интензиван унос нових речи у почетним итерацијама претраге. Тиме показује колико база података може бити оптерећена од старта коришћења, јер се уноси свих претрага евидентирају у њој (приказано на моделу на Слици 7). Очекивано је да *Pareto* тренд брже испољи скраћење времена потребног за унос података у хибридну базу података, него у *SQL* базу података, а што је последица карактеристике хибридне базе података да флексибилније управља изменама структуре података.

За потребе спровођења тестова потребно је одредити однос између броја различитих претрага (различитих речи и њихових комбинација) и укупног броја претрага. На посматраном узроку, за линеарни тренд тај однос износи 0,1%. Са друге стране, *Pareto* принцип је примењен у циљу симулирања неравномерне расподеле нових уноса претраге (кључних речи и њихових комбинација) према укупном броју претрага. Тако на пример, на узорку од 50.000 претрага где је унето 50 различитих речи претрага и њихових комбинација, симулирано је да је 40 речи претраге унето у првих 10.000 претрага, док је преосталих 10 речи или комбинација унето у преосталих 40.000 претрага. Овај принцип је сукцесивно примењен и на остале опсеге претрага, па је тако дефинисано да су прве 32 речи или њихове комбинације били унети у првих 1.984 претраге, док је првих 24 речи или њихових комбинација било унето већ у првих 400 претрага. Ова два приступа, са две различите логике расподеле речи претраге и њихових комбинација на целокупан

опсег свих претрага, примењени су у циљу елиминисања фаворизације било *SQL* или хибридне *SQL*/*NoSQL* базе података.

Две могућности имплементације захтева за претрагу су идентификоване и на сваку од њих експериментално су примењена оба принципа (линеарни и *Pareto*), а добијена времена извршавања су упоређена. Споменуте две могућности имплементације су:

- Додавање нове колоне у *SQL* базу података за сваку нову комбинацију претраге која се појави и
- Прелазак на хибридну *SQL*/*NoSQL* базу података, односно увођење *NoSQL* базе података за евидентирање података чија се структура фреквентно мења, уз задржавање осталих података у *SQL* бази података (уз потребно подешавање и интегрисање *SQL* и *NoSQL* компоненти у хибридну базу података).

Применом првог решења, за сваку извршену претрагу, евидентирано је да ли је корисник користио конкретну комбинацију речи. Свака комбинација речи има додељену једну колону, у коју се уноси вредност у зависности да ли је корисник користио ту комбинацију приликом конкретне претраге (вредност „1“ се уноси ако ју је користио, а вредност „0“ ако није). Тиме се елиминише редунданса, али се усложњава структура табеле, фреквентним изменама: за сваку нову реч претраге мења се дефиниција табеле додавањем нове колоне.

Друга опција је примена хибридне *SQL*/*NoSQL* базе података, увођењем *NoSQL* компоненте за податке чија се структура фреквентно мења. *NoSQL* компоненту је потребно интегрисати са подацима из постојеће *SQL* базе података, која ће бити *SQL* компонента хибридне *SQL*/*NoSQL* базе података. У овом решењу, *NoSQL* компонента је додата за евидентирање претрага корисника, док је веза са *SQL* компонентом остварена везом агрегације, приказаном на *UML* дијаграму класа на Слици 7. Остали подаци су задржани у табелама *SQL* базе података (*SQL* компоненти), а обе компоненте (*SQL* и *NoSQL*) су интегрисане у јединствену логичку базу података: хибридну *SQL*/NoSQL базу података. Због флексибилне и независне структуре сваког новог слога у *NoSQL* бази података, очекивано је да ће

и хибридна *SQL*/*NoSQL* база података профитирати у потребном времену извршавања, услед елиминисања времена потребног за додавање нове колоне, које постоји у *SQL* бази података. Наведена два решења су упоређена кроз експерименталне тестове.

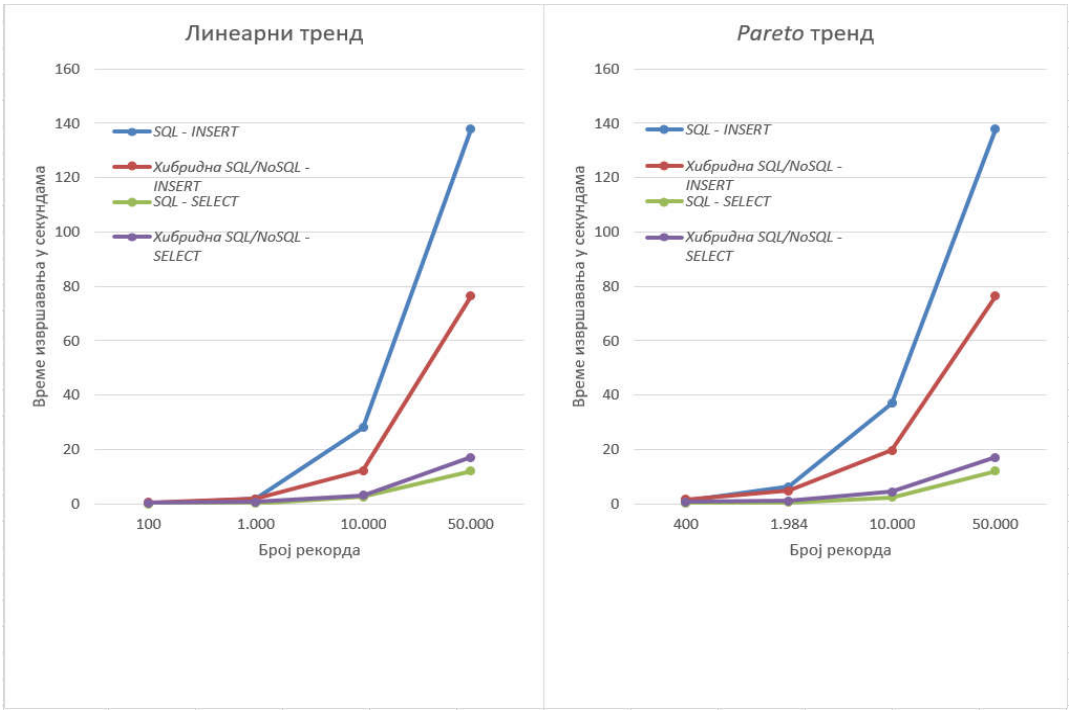
За потребе тестирања коришћен је хардвер са *Intel* i7 процесором са 4 језгра, радног такта 2.9 *GHz*, са 16*GB RAM*-а и *SSD* хард-диском на којем је био смештен целокупан систем, укључујући и базу података. Као представник *SQL* базе података коришћен је *Oracle Enterprise Edition* 12c, а као представник *NoSQL* базе података изабрана је база података заснована на документима *MongoDB* 3.4. Наведена два система за управљање базом података су изабрана, јер су тренутку спровођења тестова били најбоље рангирани представници *SQL* односно *NoSQL* база података (SolidIT, 2018). Као критеријум за мерење перформанси одабрано је укупно време извршавања наредби, односно време за које апликација успостави конекцију над базом (уколико конекција већ није отворена), проследи захтев и добије резултат извршене наредбе над базом података. Време извршавања наредби је коришћено као критеријум поређења и у раду (Bjeladinović, Babarogić, & Marjanović, 2012). Описани ток одговара начину функционисања апликација информационог система које комуницирају са базом података. За мерење укупног утрошка времена коришћен је *NetBeans IDE* (*Integrated Development Environment*) 8.0.2. Иако постоје специјализовани алати за мерење перформанси самих система за управљање базом података, попут *HammerDB* (HammerDB, 2017), *NetBeans IDE* је одабран због могућности обухватања наведених временских трошкова извршавања целокупне операције, а који одговарају времену извршавања операција у свакодневним условима, из апликацијског окружења.

Тестови су извршавани над *SQL* базом података и над хибридном *SQL*/*NoSQL* базом података, са истим сетом података од 100, 1.000, 10.000 и 50.000 записа за линеарни тренд. За *Pareto* тренд, као што је већ објашњено, коришћено је 400, 1.984, 10.000 и 50.000 записа. Брзина извршавања уноса и приказа података су тестирани. Табела 6 показује време извршавања сваке од тестираних наредби (*INSERT* и *SELECT*) у секундама, за *SQL* и хибридну *SQL*/*NoSQL* базу података.

Табела 6 - Експериментални резултати времена извршавања у секундама (*sec.*), за *SQL* и хибридну *SQL/NoSQL* базу података

| | ЛИНЕАРНИ ТРЕНД | | | | *PARETO* ТРЕНД | | | |
|---|---|---|---|---|---|---|---|---|
| Број записа: | 100 | 1.000 | 10.000 | 50.000 | 400 | 1.984 | 10.000 | 50.000 |
| *SQL – INSERT* (*sec.*) | 0,1137 | 1,7598 | 28,1076 | 137,9031 | 1,2714 | 6,38885 | 36,98245 | 137,9031 |
| Хибридна *SQL/NoSQL – INSERT* (*sec.*) | 0,5417 | 1,9399 | 12,3405 | 76,504 | 1,5773 | 4,89875 | 19,66545 | 76,504 |
| *SQL – SELECT* (*sec.*) | 0,05129 | 0,1723 | 2,5808 | 12,0321 | 0,1711 | 0,6121 | 2,44355 | 12,0321 |
| Хибридна *SQL/NoSQL – SELECT* (*sec.*) | 0,3918 | 0,7127 | 3,1542 | 17,095 | 0,7807 | 1,15025 | 4,3813 | 17,095 |

Остварени резултати приказани су и на следећем дијаграму.




а) б)

Слика 8 - Дијаграми остварених резултата применом линеарног (а) и *Pareto* тренда (б).

На дијаграмима се може уочити да је од тестираних операција временски најзахтевнији унос података, и то у *SQL* базу података, са 50.000 записа. Постигнути резултат може се описати следећом формулом (2) за рачунање утрошка времена за унос података у посматраном примеру:

$$\text{Трошак(в)} = \text{ТрошакКонекције} + \text{ТрошакУносаКомпаније} + \\ +\text{ТрошакУносаПретрагеКорисника} \quad (2)$$

Уколико се претходна формула прошири за сваку анализирану технологију, за *SQL* базу података формула (3) гласи:

$$\text{Трошак(в)} = \text{ТрошакКонекције}(\textit{Oracle}) + \\ +\text{ТрошакУносаКомпаније}(\textit{Oracle}) + \\ +\text{ТрошакУносаПретрагеКорисника}(\textit{Oracle}) + \\ +\text{ТрошакИзмењеСтруктуреПретрагеКорисника}(\textit{Oracle}) \quad (3)$$

а за хибридну *SQL*/*NoSQL* базу података формула (4) гласи:

$$\text{Трошак(в)} = \text{ТрошакКонекције}(\textit{Oracle} + \textit{Mongo}) + \\ +\text{ТрошакУносаКомпаније}\ (\textit{Oracle}) + \\ +\text{ТрошакУносаПретрагеКорисника}(\textit{Mongo}) \quad (4)$$

Формуле (3) и (4) показују да је иницијални утрошак времена коришћења хибридне *SQL*/*NoSQL* базе података већи од времена потребног за коришћење *SQL* базе података, јер је потребно успоставити конекцију на две базе података (*Oracle* и *Mongo*) уместо на само једну (*Mongo*). У тестираним случајевима коришћења, трошак конекције се сматра константним, јер на њега не утиче број извршених наредби. Конекција се у свим случајевима коришћења успоставља на исте инстанце базе података и на исти начин, што такође доприноси да трошкови конекције буду третирани као константни. Унос података о компанији не фаворизује ни једну од

база података, јер се у оба случаја ти подаци уносе у *SQL* базу података. Пропорцијално са повећањем броја слогова, потребно време за унос података о компанијама ће се подједнако увећавати, и за *SQL* и за хибридну базу података. За разлику од утрошка времена потребног за унос компаније, ситуација се мења приликом уноса података о претрагама корисника. Као што је већ објашњено, претраге корисника су у хибридној бази података имплементиране у *NoSQL* компоненти, за разлику од традиционалне *SQL* базе података. *NoSQL* компонента хибридне базе података омогућава флексибилност у структури сваког слога који се уноси, што смањује потребно време, исказано кроз компоненту трошка ТрошакУносаПретрагеКорисника (*Mongo*). Због флексибилности структуре сваког појединачног реда, када је потребно додати нове речи претраге, не долази до трошка измене унапред дефинисане структуре. То није случај код *SQL* базе података, у којој је приликом појаве нових речи претраге потребно променити дефиницију табеле, додавањем нове колоне. Овај, у односу на *NoSQL* компоненту хибридне базе података, додатни утрошак времена исказан је кроз компоненту трошка ТрошакУносаПретрагеКорисника(*Oracle*). Након измене структуре табеле ПретрагеКорисника настаје трошак и додавања записа у измењену табелу, исказан кроз компоненту ТрошакИзмењеСтруктуреПретрагеКорисника(*Oracle*). Резултати експеримента су потврдили очекивања, па је применом линеарног тренда хибридна база података била ефикаснија приликом уноса 10.000 записа (просечно измерено време за унос података у *SQL* базу података било је 28,1076 секунди, док је хибридној бази података било потребно 12,3405 секунди) и 50.000 записа (137,9031 секунди за *SQL* базу података наспрам 76,504 секунди за хибридну *SQL*/*NoSQL* базу података). *SQL* база података је била доминантна са мањим бројем записа, попут 100 (0,1137 секунди за *SQL* базу података наспрам 0,5417 секунди колико је било потребно хибридној бази података). Овакав резултат је био очекиван и објашњава се релативно фиксним трошком успостављања две конекције. Са повећањем броја записа, смањивао се и удео трошка конектовања у укупном трошку извршавања, па је тако разлика на 1.000 записа била мања него на 100. Са 1.000 слогова *SQL* база података се временом извршавања значајно приближила *SQL* бази података (1,7598 секунди наспрам 1,9399 секунди). Са 10.000 слогова је хибридна база података имала краће време извршавања, при чему је такав исход потврђен и са 50.000

слогова. Слични резултати су постигнути и са *Pareto* трендом, уз разлику да је по овом принципу чак и уношење 1.984 записа извршено брже од стране хибридне базе података. Објашњење се може наћи у количини извршених наредби за промену структуре (по *Pareto* тренду је било 32 операције промене структуре за 1.984 записа, а по линеарном 10 за 1.000 записа).

Резултати за читање података су такође очекивани. Због непотребности успостављања две конекције, *SQL* база података је била бржа по оба тренда, са свим тестираним количинама записа. Времена извршавања операције читања су значајно краћа од времена уноса података, за представнике оба типа база података, што је уочљиво на Слици 8.

## 7.2 Примена приступа за униформно коришћење *SQL* и *NoSQL* база података заснованог на хибридној *SQL/NoSQL* бази података

Новоразвијени приступ пројектовања хибридне *SQL*/*NoSQL* базе података представљен је у поглављу 6. У наставку ће бити приказана примена новог приступа за униформно коришћење база података различитих типова употребом хибридне *SQL*/*NoSQL* базе података. Као полазиште за пример узет је модел базе података једног реалног система, који је приказан на Слици 7 и већ детаљно објашњен на почетку овог поглавља. На приказаном делу концептуалног модела *CRM*-а ресторана, за примере униформног коришћења *SQL* и *NoSQL* база података применом хибридне базе података посебно су значајне класе `KOMPANIJA` и `PRETRAGE_KORISNIKA`. Као што се може видети на Слици 7, подаци о компанијама се складиште у *SQL* базу податка (тј. у *SQL* компоненту хибридне *SQL*/*NoSQL* базе података), док се подаци о претрагама корисника смештају у *NoSQL* базу податка (тј. у *NoSQL* компоненту хибридне *SQL*/*NoSQL* базе података). У сврху представљања новог приступа у раду са подацима из хибридне базе података, без обзира у којој компоненти се исти налазе или се смештају, употребљено је неколико наредби, различитих степена комплексности.

За приказ погодности униформног, истовременог и интегрисаног коришћења база података различитих типова које пружа примена новог приступа коришћења хибридне базе података, изабране су следеће наредбе:

- Унос података и
- Приказ постојећих података.

С обзиром на то да се снага новог приступа коришћења хибридне базе података, између осталог, заснива на чињеници да је кориснику довољно познавање само стандардизованог *SQL* упитног језика за рад са свим подацима хибридне базе података (тј. да није потребно да корисник води рачуна који подаци су у којем типу базе података, да ли тај тип базе података подржава *SQL* језик и који специфичан језик је потребан у раду са сваким конкретним типом базе података), изабране наредбе су додатно рашчлањене у циљу приказа различитих исхода приликом реализације *SQL* наредби. Поред наведеног, жеља је била да се укључе и наредбе, односно специфични случајеви коришћења извршавања наредбе за унос и наредбе за приказ података, којима се додатно наглашава универзалност и примењивост новог приступа, како у раду са различитим типовима база података које чине компоненте хибридне *SQL*/*NoSQL* базе података, тако и приликом рада само са једним типом базе података (са *SQL* базом података или неким подтипом *NoSQL* база података). У ту сврху одабрани су следећи случајеви коришћења:

1) Унос података (у *SQL* компоненту хибридне *SQL*/*NoSQL* базе података)
2) Унос података (у *NoSQL* компоненту хибридне *SQL*/*NoSQL* базе података)
3) Приказ постојећих података (упит, којим се учитавају подаци само из *SQL* компоненте хибридне *SQL*/*NoSQL* базе података)
4) Приказ постојећих података (упит, којим се учитавају подаци само из *NoSQL* компоненте хибридне *SQL*/*NoSQL* базе података)
5) Приказ постојећих података (упит, којим се истовремено учитавају подаци из *SQL* и из *NoSQL* компоненте хибридне *SQL*/*NoSQL* базе података)

### 7.2.1. Унос података применом новог приступа

Логика случајева коришћења за извршавање наредби за унос, брисање или измену података хибридне базе података представља поједностављену логику случаја коришћења извршавања упита над хибридном базом података. Због тога ће већ објашњена логика извршавања наредби по новом приступу бити прво показана на примеру уноса података и то у два случаја. У првом случају одредишна база

података представља *SQL* компоненту хибрида, док је у другом случају одредишна база података *NoSQL* компонента хибрида. Први случај, када је одредишна база података *SQL* компонента, уједно садржи и најједноставнију логику извршавања, па ће он бити прво приказан и то на примеру уноса података о компанији.

Да би корисник унео податке о компанији, потребно је да унесе и изврши *SQL* наредбу за унос, тј. *INSERT* наредбу. Одговарајућа наредба, за унос компаније „*ABC*“ која има вредност 100 за `KOMPANIJA_ID`, би гласила:

```
INSERT INTO KOMPANIJA (KOMPANIJA_ID, NAZIV_KOMPANIJE) VALUES (100, 'KOMPANIJA ABC');
```

Након уноса *SQL* наредбе од стране корисника, ОУН компонента је прихвата и започиње њену „обраду“. Након синтаксне анализе, утврђују се тип објекта у који се уносе подаци, тип базе података којем припада конкретан објекат, као и конкретан СУБП, а све то због утврђивања језика у који евентуално треба да се мапира наредба. За то је задужена ПКР компонента, која ће у наведеном случају утврдити да је објекат `KOMPANIJA`, у који се уносе подаци, типа табеле, да припада *SQL* бази података, што имплицира да није потребно мапирање у специфичан језик. Затим се проверава да ли нека колона табеле `KOMPANIJA` представља спољни кључ. Овај податак ће добити контролор ограничења, на основу података из табеле `PRAVILA_INTEGRITETA`. Пошто табела `KOMPANIJA` у приказаном моделу не референцира ни један други објекат, није потребно проверавати да ли унете вредности неких колона одговарају вредностима које треба да има спољни кључ. Након извршених анализа, извршава се и сама наредба над одредишном базом података, која је у овом случају *SQL* компонента хибрида. Наредбу над *SQL* базом података ће извршити контролор интеграције, који ће затим резултате извршавања наредбе вратити ОУН компоненти. Иако представља најједноставнији случај, овом наредбом је показана универзалност у коришћењу новог приступа у раду са искључиво *SQL* базом података, тј. *SQL* компонентом хибридне *SQL*/*NoSQL* базе података.

Други случај коришћења садржи комплекснију логику извршавања од претходног, јер приликом његове реализације жељени подаци за одредишну базу података имају *NoSQL* базу, тј. подаци се уносе у *NoSQL* компоненту хибридне базе

података. Нови приступ, као што је већ речено, пружа униформност приликом приказивања, уноса, измене и брисања свих података хибридне базе података, без обзира у којој компоненти се они налазе. Корисник и у другом случају коришћења, кроз *SQL* конзолу система, уноси и извршава *SQL* наредбу за унос података, тј. *INSERT* наредбу. Наредба за унос података о претрази корисника која има вредност 111 за `PRETRAGA_ID`, коју је извршио `KORISNIK` са бројем 123, који ради у компанији са `KOMPANIJA_ID`-ем 100 и који је претраживао ресторан користећи кључне речи „ресторан“ и „*X*“ (при чему је симбол „*X*“ узет да представља назив ресторана) гласила би:

```
INSERT INTO PRETRAGA_KORISNIKA (KOMPANIJA_ID, PRETRAGA_ID, KORISNIK,
RESTORAN_X) VALUES (100, 111, 123, 1);
```

Процес обраде унете наредбе се подудара са процесом описаним у претходном примеру. Након прихватања наредбе, ОУН компонента утврђује назив објекта у који се уносе подаци. У садејству са ПКР компонентом долази до мета података о објекту `PRETRAGA_KORISNIKA`. У датом примеру, као што је већ наведено приликом описа концептуалног модела, објекат `PRETRAGA_KORISNIKA` је типа документ, припада *NoSQL* типу базе података заснованом на документима, односно ускладиштен је у *MongoDB* СУБП. Следеће што се проверава је да ли документ `PRETRAGA_KORISNIKA` референцира неке друге објекте. Пошто се на датом моделу (Слика 7) уочава да је за идентификацију сваког појављивања `PRETRAGA_KORISNIKA` потребно користити вредност атрибута `KOMPANIJA_ID` из `KOMPANIJA` заједно са вредношћу атрибута `PRETRAGA_ID` из `PRETRAGA_KORISNIKA`, очигледно је да постоји зависност између `PRETRAGA_KORISNIKA` и `KOMPANIJA`. То је дефинисано правилима интегритета, као што је приказано у Табели 7.

Табела 7 - Правила интегритета објекта `PRETRAGA_KORISNIKA`, која су ускладиштена у SQL бази података, у табели `PRAVILA_INTEGRITETA`

| NAZIV_OBJEKTA | TIP_PRAVILA | IDENTIFIKATOR_OBJEKTA | REFERENCIRAJUCI_PROPERTI | REFERENCIRANI_OBJEKAT | REFERENCIRANI_PROPERTI |
|---|---|---|---|---|---|
| PRETRAGA_KORISNIKA | REF_INTEGRITET | KOMPANIJA_ID<br>PRETRAGA_ID | KOMPANIJA_ID | KOMPANIJA | KOMPANIJA_ID |

У табели *PRAVILA_INTEGRITETA* заглавља колона садрже општи термин „објекат“ јер ова табела садржи правила интегритета и за објекте *SQL* компоненте (табеле) и за објекте *NoSQL* компоненте (документа, парови кључ-вредност, фамилије колона, чворови графова) хибридне *SQL*/*NoSQL* базе података. Из истог разлога употребљен је и термин „проперти“ (обухвата колоне, поља итд.).

За пример уноса новог појављивања *PRETRAGA_KORISNIKA*, наведено ограничење референцијалног интегритета је значајно јер спречава да се унесе претрага корисника запосленог у компанији која не постоји, тј. која није већ унета у шифарник компанија. У циљу спречавања описане ситуације, ОУН компонента ће креирати додатну *SQL* наредбу којом ће проверити вредност за *KOMPANIJA_ID*, а коју је унео корисник. У наведеном случају, разлог генерисања наредбе у *SQL* синтакси је чињеница да се, у посматраном примеру, подаци о компанији налазе управо у *SQL* компоненти хибридне базе података. Изгенерисана наредба за проверу референцијалног интегритета гласи:

```
SELECT 1 FROM KOMPANIJA WHERE KOMPANIJA_ID = 100;
```

Уколико унета вредност за *KOMPANIJA_ID* (у овом случају 100) има одговарајући пар у шифарнику, резултат извршавања упита ће вратити контролну вредност (у овом примеру је то литерал „1“) и уписивање података у документ *PRETRAGA_KORISNIKA* може да се изврши. Да би се та операција извршила неопходно је *SQL* наредбу за унос мапирати у одговарајућу наредбу за унос документа *MongoDB* језика. Трансформисана наредба ће гласити:

```
DB.PRETRAGA_KORISNIKA.INSERT({KOMPANIJA_ID:100, PRETRAGA_ID:111,
KORISNIK:123, RECI:[”RESTORAN”, ”X”]});
```

Након мапирања наредбе иста се извршава над *NoSQL* компонентом хибридне *SQL*/*NoSQL* базе података, прецизније над *MongoDB* компонентом хибрида. Овим је показана униформност новог приступа и у раду са *NoSQL* компонентом хибридне базе података, јер је корисник извршио унос података на исти начин на који је то урадио и у претходном примеру са *SQL* компонентом: у оба случаја је унео и извршио *SQL* наредбу.

### 7.2.2. Приказ података применом новог приступа

Трећи и четврти пример представљају случајеве коришћења у којима корисник уноси и извршава упит над једним објектом, док ће у петом примеру бити описано извршавање случаја коришћења у којем се резултујући подаци упита учитавају из различитих објеката, база података различитих типова, које чине компоненте хибридне *SQL*/*NoSQL* базе података.

У примеру 3, за приказ података из одређеног објекта хибридне базе података, корисник користити *SQL* синтаксу за писање упита. Уколико је потребно приказати све податке о компанији чији назив је „*KOMPANIJA ABC*“, одговарајући *SQL* упит би гласио:

```
SELECT * FROM KOMPANIJA WHERE NAZIV_KOMPANIJE = 'KOMPANIJA ABC';
```

Упркос томе што овај пример садржи *WHERE* клаузулу којом се филтрирају подаци, упит је ипак релативно једноставне сложености па је зато изабран за почетни пример при демонстрацији рада новог приступа са упитима који се извршавају над хибридном базом података. Једноставности реализације горе-наведеног упита доприноси и чињеница да се подаци о компанији налазе у *SQL* компоненти хибридне базе података, што на основу до сада већ показаног јасно сугерише да неће бити потребно мапирање унете наредбе у специфичан језик одредишне базе података. Ипак, још једном је важно нагласити да приликом куцања оваквог упита корисник не треба да води рачуна да ли резултујући подаци извршавања упита за одредишну базу имају само *SQL*, само *NoSQL* или више различитих компоненти (*SQL* и различити типови *NoSQL* база података) хибридне базе података. На крају овог поглавља биће приказан пример упита који користи податке из *SQL* и *NoSQL* базе података засноване на документима. Пре тога ће бити приказан још један случај коришћења, у којем се подаци учитавају такође из једне одредишне базе података, али за разлику од претходног примера где је она била типа *SQL*, у примеру који следи одредишна база података је *NoSQL* база података заснована на документима.

За приказ случаја коришћења у којем корисник уноси *SQL* наредбу ради приказа података из објекта за који ће се испоставити да има *NoSQL* одредишну

базу података, одабран је пример приказа података о претрагама корисника. Као што је већ објашњено код концептуалног модела, подаци о претрагама ће бити смештени у *NoSQL* компоненту хибридне базе података, иако корисник о томе не мора да води рачуна приликом комуникације са системом, јер ће он и у овом случају куцати наредбе стандардизованог *SQL* језика. Пример *SQL* наредбе за приказ претрага корисника, који има број 123, гласи:

```
SELECT * FROM PRETRAGA_KORISNIKA WHERE KORISNIK = 123;
```

Анализом унете наредбе, ОУН компонента шаље захтев ПКР компоненти за добијање мета података о објекту `PRETRAGA_KORISNIKA`. Из речника мета података добијају се подаци о траженом објекту и у овом примеру то је документ као тип објекта, који припада *NoSQL* типу базе података заснованом на документу, конкретно *MongoDB* СУБП-у. Правила референцијалног интегритета, у овом примеру, неће проузроковати генерисање нових, контролних, упита, као што је то био случај у примеру уноса података у *NoSQL* компоненту. Наиме, пошто је у питању наредба за приказ података из једног документа, није потребно проверавати вредности пропертија који имају функцију спољних кључева, јер су те вредности проверене приликом уноса података. Одсуство спајања табела у унетој *SQL* наредби елиминише могућност да се резултујући подаци упита у овом примеру приказују из више објеката, па ни због тога неће бити потребе да ОУН генерише додатне, контролне, упите. Због свега наведеног, следећи корак у реализацији овог случаја коришћења је мапирање наредбе. Унета *SQL* наредба, након мапирања, имаће следећу синтаксу:

```
DB.PRETRAGA_KORISNIKA.FIND({KORISNIK: 123});
```

Након завршеног процеса мапирања наредбе иста се извршава над *NoSQL* компонентом хибридне *SQL*/*NoSQL* базе података, прецизније над *MongoDB* компонентом хибрида. Као и приликом уноса података у *NoSQL* компоненту хибридне базе података и овим примером приказа података из *NoSQL* компоненте демонстрирана је униформност новог приступа у раду са свим компонентама хибридне базе података. Корисник креира упит на исти начин, употребом *SQL*

синтаксе, без обзира из које компоненте ће бити приказани резултујући подаци извршавања упита.

Пети пример демонстрира примену новог приступа за истовремени приказ резултујућих података из *SQL* и *NoSQL* компоненти хибридне *SQL*/*NoSQL* базе података. За случај коришћења извршавања упита којим се спајају подаци из објеката база података, које су различитог типа, биће искоришћен приказани концептуални модел (Слика 7). Упитом је потребно приказати све податке о компанијама и о претрагама ресторана које су извршили запослени те компаније. Одговарајућа *SQL* наредба би гласила:

```
SELECT * FROM KOMPANIJA JOIN PRETRAGA_KORISNIKA USING (KOMPANIJA_ID);
```

ОУН компонента, по већ описаној процедури, анализира унету наредбу и шаље називе објеката `KOMPANIJA` и `PRETRAGA_KORISNIKA` ПКР компоненти, ради прибављања мета података о споменутим објектима упита. У конкретном примеру, објекат `KOMPANIJA` представља табелу *SQL* базе података (*Oracle* СУБП), док објекат `PRETRAGA_KORISNIKA` представља документ унутар *NoSQL* базе података засноване на документима (*MongoDB*). Наведено имплицира да део упита за табелу `KOMPANIJA` неће бити потребно мапирати у језик одредишне базе података, већ ће се над одредишном *SQL* базом података извршити следећи део упита:

```
SELECT * FROM KOMPANIJA;
```

За документ `PRETRAGA_KORISNIKA` ће бити потребно извршити мапирање у специфични језик одредишне базе података. Међутим, пре трансформације упита (коју обавља компонента за мапирање упита), контролор ограничења, по захтеву ОУН компоненте, ће приступити правилима интегритета. Референцијалном интегритету у овом примеру такође ће одговарати запис из Табеле 7. У циљу спајања адекватних записа из табеле `KOMPANIJA` са подацима из документа `PRETRAGA_KORISNIKA` неопходно је испоштовани наведено правило интегритета. То се постиже извршавањем једног дела упита и заменом његових резултата у други део упита. Уколико у корисничком упиту не фигурише *WHERE* клаузула за *NoSQL* објекат, као што је случај са упитом који се тренутно анализира, тада се прво извршава део упита који се односи на *NoSQL* компоненту. Ово правило новог

приступа дефинисано је ради смањења броја наредби које ће се мапирати у специфичан језик одредишне *NoSQL* базе података. Овим се директно постиже уштеда у времену потребном за мапирање дела упита у специфичан упит одредишне базе података, јер се у том случају изврши мапирани упит без клаузуле за повезивање са подацима из друге компоненте (тј. *SQL* компоненте). Добијени резултати се из *NoSQL* базе података (која извршава *NoSQL* део упита) врате контролору интеграције и ОУН компоненти, која их затим инкорпорира у *WHERE* клаузулу *SQL* упита. Тиме је омогућено повезивање података из две компоненте и њихова интеграција у јединствени резултат у складу са дефинисаним правилом референцијалног интегритета, уз мањи трошак времена и ресурса потребних за мапирање *WHERE* клаузуле у *MongoDB*. Адекватна наредба која ће бити извршена у *MongoDB* СУБП-у је:

```
DB.PRETRAGA_KORISNIKA.FIND({});
```

Резултат извршавања наведеног упита се враћа контролору интеграције који их прослеђује ОУН компоненти. ОУН из датог резултата извлачи вредности за `KOMPANIJA_ID` и преводи их у формат *SQL* табеле, коју динамички додаје *WHERE* клаузули *SQL* упита који треба да се изврши. Након што ОУН модификује иницијални *SQL* упит за приказ свих компанија, резултујући упит који ће се и извршити над *SQL* компонентом, написан у псеудо коду, ће гласити:

```
SELECT * FROM KOMPANIJA WHERE KOMPANIJA_ID IN
LISTA_VREDNOSTI_IZ_NOSQL_UPITA;
```

Након извршавања овог упита и селекције само оних компанија које имају барем једну претрагу, контролор интеграције врши спајање резултујуће табеле *SQL* упита са резултатима извршавања *NoSQL* упита који су у претходном кораку преведени у формат *SQL* табеле, при чему се сада резултујућој табели, поред `KOMPANIJA_ID` додају и остала поља које је вратио *NoSQL* упит. Овим је заокружен процес спајања података из више различитих одредишних база података (у овом случају две), који се у ОУН компоненти налазе у форми резултујуће *SQL* табеле, коју затим наведена компонента приказује кориснику, као интегрални резултат извршавања упита.

Тиме је поред униформности новог приступа у раду са различитим наредбама над различитим компонентама демонстрирана и његова интегришућа функционалност.

# 8. Закључак

Постојећи приступи пројектовања *SQL* и *NoSQL* база података садрже фазе, технике и методе које не узимају у обзир специфичности процеса пројектовања база података другог типа, осим оног за који су намењене. Због тога ови приступи не могу на адекватан начин одговорити специфичним потребама пројектовања хибридне *SQL*/*NoSQL* базе података, које проистичу из различитих карактеристика њених *SQL* и *NoSQL* компоненти.

У овој дисертацији приказан је и детаљно објашњен оригинални и новоразвијени методолошки приступ за пројектовање и коришћење хибридне *SQL/NoSQL* базе података који је примењив за пројектовање нових и редизајн постојећих база података. Специфичности пројектовања *SQL* и *NoSQL* база података су обухваћене представљеним приступом на начин да се *SQL* и *NoSQL* базе података различитих типова третирају као компоненте јединствене логичке, тј. хибридне *SQL/NoSQL*, базе података. У овом приступу су дефинисане неопходне фазе и активности за реализацију процеса пројектовања. Степен структурираности података је коришћен као критеријум за утврђивање оправданости пројектовања хибридне *SQL/NoSQL* базе података и за одабир кандидата за сваку од компоненти хибрида.

Проблем интеграције и униформног коришћења база података различитих типова у литератури је решаван на више начина: миграцијом, избором постојећег или креирањем новог униформног језика, а последњих година и употребом хибридних база података. Миграцијом се проблем интеграције своди на коришћење базе података одређеног типа, чиме се губи флексибилност истовременог коришћења база података различитих типова, док креирање и одабир униформног језика који би глобално послужио за интеграцију база података различитих типова захтева консензус водећих светских произвођача база података. Иако тек у повоју, хибридне базе података представљају један савремени приступ решавању описаног проблема.

Представљени приступ интегрише *SQL* и *NoSQL* базе података као компоненте јединствене логичке, хибридне *SQL*/*NoSQL*, базе података. Уз употребу стандардизованог *SQL* упитног језика приступ омогућава и њихово универзално

коришћење. Униформност коришћења омогућава кориснику унос искључиво *SQL* наредби, без обзира да ли ће се оне извршавати над *SQL* или *NoSQL* компонентом хибрида. Након анализе, декомпозиције и прилагођавања од стране система, унета наредба и њени делови се извршавају над одредишним базама података без обзира којег су типа и резултати извршавања се приказују кориснику у виду обједињених резултата. У дисертацији су дати и прикази примене новог приступа за пројектовање и коришћење хибридне *SQL/NoSQL* базе података на реалном примеру, компаративне анализе перформанси (по изабраном показатељу времена извршавања наредби) хибрида, пројектованог употребом приказаног приступа и традиционално пројектоване *SQL* базе података, као и примери интеграције и униформног коришћења *SQL* и *NoSQL* база података као компоненти хибрида.

Из свега наведеног може се закључити да су општа и посебне хипотезе које су дефинисане приликом пријаве теме докторске дисертације потврђене, а да резултати ове дисертације доприносе сазнањима у вези са процесом пројектовања хибридне *SQL*/*NoSQL* базе података и са процесима интеграције и униформног коришћења *SQL* и *NoSQL* база података као компоненти хибрида.

Дисертација је отворила и простор за даља истраживања. То се пре свега огледа у истраживању додатне аутоматизације одређених активности пројектовања. Редовно праћење статистике извршавања наредби над базом података је неопходан, али и ограничавајући услов, јер кандидати за прелазак у другу компоненту не могу бити аутоматски детектовани.

Такође, представљени приступ због заступљености *SQL* база података предност приликом прототипског развоја даје управо базама података овог типа. Један од праваца даљег истраживања био би да се прецизира критеријум под којим би, приликом пројектовања базе података од почетка, било погодно за прототип изабрати представника *NoSQL* база података. Прецизирање критеријума за одабир *NoSQL* база података одређеног типа могло би се такође разрадити у будућим истраживањима.

Развој и коришћење додатних индикатора којима би се одређивала мера у којој је нека пројектована шема, или њен део, прилагођена смештању структурираних података представљају правце даљег развоја и проширења новог приступа.

## Литература


[1] Abiteboul, S. (1997). *Querying semi-structured data*. In International Conference on Database Theory (ICDT 97), 1-18. London, UK: Springer.

[2] Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M. C., & Senellart, P. (2011). *Web data management*. New York, NY, USA: Cambridge University Press.

[3] Agarwal, R., Prasad, J., Tanniru, M., & Lynch, J. (2000). Risks of Rapid Application Development (RAD_risks). *Communications of the ACM, 43*(11), 77-88. doi:10.1145/352515.352516

[4] Alomari, E., Barnawi, A., & Sakr, S. (2015). Cdport: A portability framework for nosql datastores. *Arabian Journal for Science and Engineering*, *40*(9), 2531-2553. doi:10.1007/s13369-015-1703-0

[5] Arenas, M., Diaz, G., Fokoue, A., Kementsietsidis, A., & Srinivas, K. (2014). A Principled Approach to Bridging the Gap between Graph Data and their Schemas. *PVLDB, 7*(8), 601-612.

[6] Atzeni, P., Bugiotti, F., & Rossi, L. (2012). *Uniform access to non-relational database systems: The SOS platform*. In International Conference on Advanced Information Systems Engineering, 160-174. Berlin, Germany: Springer.

[7] Atzeni, P., Bugiotti, F., & Rossi, L. (2014). Uniform access to NoSQL systems. *Information Systems, 43*(C), 117-133. doi: 10.1016/j.is.2013.05.002

[8] Atzeni, P., Jensen, C. S., Orsi, G., Ram, S., Tanca, L., & Torlone, R. (2013). The relational model is dead, SQL is dead, and I don't feel so good myself. *ACM SIGMOD Record, 42*(2), 64-68. doi: 10.1145/2503792.2503808

[9] Avison, D., & Fitzgerald, G. (2003). Where now for development methodologies? *Communications of the ACM, 46*(1), 78-82. doi:10.1145/602421.602423

[10] Avison, D., & Fitzgerald, G. (2006a). *Methodologies for developing information systems: A historical perspective*. In The Past and Future of Information Systems: 1976-2006 and Beyond, 27-38. Boston, MA, USA: Springer.

[11] Avison, D., & Fitzgerald, G. (2006b). *Information systems development* (четврто издање). London, UK: McGraw-Hill Education.

[12] Bach, M., & Werner, A. (2014). Standardization of NoSQL Database Languages. In Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., & Kostrzewa, D. (уредници), *Beyond Databases, Architectures, and Structures*, *Proceedings of 10th International Conference (BDAS 2014)*, 50-60. Cham, Switzerland: Springer. doi: 10.1007/978-3-319-06932-6

[13] Badia, A., & Lemire, D. (2011). A Call to Arms: Revisiting Database Design. *SIGMOD, 40*(3), 61-69. doi: 10.1145/2070736.2070750

[14] Balaji, S., & Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management, 2*(1), 26-30.

[15] Batini, C., Ceri, S. & Navathe, S. B. (1992). *Conceptual Database Design: An Entity-Relationship Approach*. Redwood City, CA, USA: Benjamin-Cummings.

[16] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development*. Доступно на: http://agilemanifesto.org/, приступано: август 2016.

[17] Bjeladinovic, S. (2018) A fresh approach for hybrid SQL/NoSQL database design based on data structuredness, *Enterprise Information Systems, ISSN (Online)*: *1751-7583*, *12*(8-9), 1202-1220. doi: 10.1080/17517575.2018.1446102

[18] Bjeladinović, S., Babarogić, S., & Marjanović, Z. (2012). *Comparison of relational and NoSQL systems*. In Proceedings of XIII International Symposium SymOrg 2012, 974-980. Beograd, Srbija: FON.

[19] Bjeladinović, S. & Marjanović, Z. (2016). Primena virtuelizacije u cilju integracije relacionih i NoSQL baza podataka, na primeru Oracle-a. *Info M*, *57/2016*, 4-10.

[20] Blankers, T. (2014). *Data Conversion for Modern Applications* [White Paper]. Доступно на Uniface: http://www.uniface.com/wp-content/uploads/2015/11/wp-data-conversion.pdf, приступано: јул 2016.

[21] Boehm, B. (1988). A Spiral model of software development and enhancement. *Computer*, 61-72. Washington, DC, USA: IEEE. doi: 10.1109/2.59

[22] Boehm, B. (2000). *Spiral Development: Experience, Principles, and Refinements*. Pittsburgh, PA, USA: Software Engineering Institute.

[23] Bondiombouy, C. (2014). *Query Processing in Cloud Multistore Systems* (докторска теза), University of Montpellier, Montpellier, France. doi: 10.1504/IJCC.2016.080903

[24] Bonnet, L., Laurent, A., Sala, M., Laurent, B., & Sicard, N. (2011). *Reduce, You Say: What NoSQL Can Do for Data Aggregation and BI in Large Repositories*. In 22nd International Workshop on Database and Expert Systems Applications 483-488. Washington, DC, USA: IEEE. doi: 10.1109/DEXA.2011.71

[25] Brewer, E. (2000). *Towards Robust Distributed Systems*. In Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing. New York, NY, USA: ACM.

[26] Bugiotti, F., & Cabibbo, L. (2013a). *An Object-Datastore Mapper Supporting NoSQL Database Design*. Доступно на: http://cabibbo.dia.uniroma3.it/pub/ondm.pdf, приступано: јул 2016.

[27] Bugiotti, F., Cabibbo, L. (2013b). *A Comparison of Data Models and APIs of NoSQL Datastores*. In Italian Symposium on Advanced Database Systems (SEBD '13), 63-74. Rome, Italy: Roma Tre University.

[28] Bugiotti, F. (2012). *A model oriented approach to heterogeneity* (докторска теза). Roma Tre University, Rome, Italy.

[29] Bugiotti, F., Cabibbo, L., Atzeni, P., & Torlone, R. (2014). *Database design for NoSQL systems*. In International Conference on Conceptual Modeling 223-231. Cham, Switzerland: Springer.

[30] Calil, A., & dos Santos Mello, R. (2012). *SimpleSQL: A Relational Layer for SimpleDB*. In Proceedings of the 16th East European conference on Advances in Databases and Information Systems (ADBIS'12), 99-110. Berlin, Germany: Springer. doi:10.1007/978-3-642-33074-2_8

[31] Carkenord, B. (2009). *Seven Steps to Mastering Business Analysis*. Fort Lauderdale, FL, USA: J. Ross Publishing.

[32] Carr, M., & Verner, J. (1997). *Prototyping and software development approaches*. Hong Kong, HK: City University of Hong Kong.

[33] Cattell, R. (2010). Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record, 39*(4), 12-27. doi:10.1145/1978915.1978919

[34] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS), 26*(2). doi: 10.1145/1365815.1365816

[35] Chen, P. (1976) The Entity-Relationship Model-Toward a Unified View of Data. *ACM Transactions on Database Systems*, *1*(1), 9-36.

[36] Chung, L., & Leite, J.C.S.P. (2000) Non-Functional Requirements in Software Engineering. In: A.T. Borgida et al. (уредници), *Mylopoulos Festschrift, Lecture Notes in Computer Science 5600,* 363-379. Berlin, Germany: Springer.

[37] Churcher, C. (2007). *Beginning Database Design From Novice to Professional.* Berkeley, CA, USA: Apress.

[38] Curé, O., Hecht, R., Le Duc, C., & Lamolle, M. (2011). *Data Integration over NoSQL Stores Using Access Path Based Mappings*. In International Conference on Database and Expert Systems Applications, 481-495. Berlin, Germany: Springer.

[39] da Silva, C. (2011). *Data Modeling with NoSQL: How, When and Why?* (мастер теза). Faculdade de Engenharia da Universidade do Porto, Porto, Portugal.

[40] de Lima, C., & dos Santos Mello, R. (2015). *A Workload-Driven Logical Design Approach for NoSQL Document Databases*. In Proceedings of iiWAS '15, 73, 1-10. New York, NY, USA: ACM. doi: 10.1145/2837185.2837218

[41] De Virgilio, R., Maccioni, A., & Torlone, R. (2014). *Model-Driven Design of Graph Databases*. In International Conference on Conceptual Modeling, 172-185. Cham, Switzerland: Springer.

[42] Dennis, A., Wixom, B. H., & Tegarden, D. (2015) *Systems Analysis and Design: An Object-Oriented Approach with UML* (пето издање). New York, NY, USA: John Wiley & sons.

[43] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107-113.

[44] Ebert, C. (1996) Classification techniques for metric-based software development. *Software Quality Journal, 5*(4), 255-272.

[45] Gilbert, S., & Lynch, N. (2012). Perspectives on the CAP Theorem. *Computer, 45* (2), 30-36. Washington, DC, USA: IEEE. doi: 10.1109/MC.2011.389

[46] Gottesdiener, E. (1995). RAD realities: beyond the hype to how RAD really works. *Application Development Trends*, *August 1995*, 28-38.

[47] Grolinger, K., Hayes, M., W., Higashino, W., L'Heureux, A., Allison D., & Capretz, M. (2014). *Challenges for MapReduce in Big Data.* In Proceedings of the IEEE 10th 2014 World Congress on Services (SERVICES 2014). Washington, DC, USA: IEEE.

[48] Grolinger, K., Higashino, W., Tiwari, A., & Capretz, M. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications, 22*(2). doi: 10.1186/2192-113X-2-22

[49] Gurevich, Y. (2015). Comparative Survey of NoSQL/NewSQL DB Systems (докторска теза), The Open University, Milton Keynes, UK.

[50] HammerDB (2017). *About HammerDB.* Доступно на: http://www.hammerdb.com/about.html, приступано: април 2017.

[51] Han, J., Haihong, E., Le, G., & Du, J. (2011). *Survey on NoSQL database*. In 6th International Conference on Pervasive Computing and Applications (ICPCA), 363-366. Washington, DC, USA: IEEE. doi: 10.1109/ICPCA.2011.6106531

[52] Hanine, M., Bendarag, A., & Boutkhoum, O. (2016). Data Migration Methodology from Relational to NoSQL Databases. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, *9*(12), 2369-2373. doi: 10.5281/zenodo.1123793

[53] Holzschuher, F., & Peinl, R. (2013). *Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j*. In Proceedings of the Joint EDBT/ICDT 2013 Workshops, 195-204. New York, NY, USA: ACM.

[54] Huang, L., & Nie, J. (2010). *Using Pareto Principle to Improve Efficency for Selection of QoS Web Services*. In Proceedings of the 7th IEEE Consumer Communications and Networking Conference (CCNC 10), 854-855. Washington, DC, USA: IEEE.

[55] IIBA (2015) *A Guide to the Business Analysis Body of Knowledge (BABOK Guide)* (треће издање). Toronto, Canada: International Institute of Business Analysis.

[56] Inmon, H. W., Strauss, D., & Neushloss, G. (2008). *DW 2.0 The Architecture for the Next Generation of Data Warehousing.* San Francisco, CA: Morgan Kaufman.

[57] Isaias, P., & Issa, T. (2015). *High level models and methodologies for information systems*. New York, NY, USA: Springer.

[58] ISO (1997). *ISO/IEC JTC 1 /SC 32 Data Management and Interchange*. Доступно на: http://www.iso.org/iso/standards_development/technical_committees/

other_bodies/iso_technical_committee.htm?commid=45342, приступано: јул 2016.

[59] Jirava, P. (2004). System development life cycle. *Scientific papers of the University of Pardubice, 9*(2004), 118-125.

[60] Katsov, I. (2012). *NoSQL Data Modeling Tehniques*. Доступно на: https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/, приступано: јун 2016.

[61] Khan, S., & Mane, V. (2013). SQL support over MongoDB using metadata. *International Journal of Scientific and Research Publications*, *3*(10), 1-5.

[62] Kim, C., Weston, R. H., Hodgson, A., & Lee, K. (2003). The complementary use of IDEF and UML modelling approaches. *Computers in Industry, 50*(1), 35-56.

[63] Lake, P., & Crowther, P. (2013). *Concise Guide to Databases*. London, UK: Springer.

[64] Lawrence, R. (2014). *Integration and Virtualization of Relational SQL and NoSQL Systems including MySQL and MongoDB*. In International Conference on Computational Science and Computational Intelligence, 285-290. Washington, DC, USA: IEEE.

[65] Lazarević, B., & Nešković, S. (1997). Sistemsko-teorijska kritika objektno orjentisanih pristupa razvoju softvera. *Info Science*, *5*(4), 17-23.

[66] Lazarević, B., Marjanović, Z., Aničić, N., & Babarogić, S. (2006). *Baze podataka (treće izdanje)*. Beograd, Srbija: FON.

[67] Lebo, T., Del Rio, N., Fisher, P., & Salisbury, C. (2017). A Five-Star Rating Scheme to Assess Application Seamlessness. *Semantic Web – Interoperability, Usability, Applicability an IOS Press Journal, 8*(1), 43-63.

[68] Liao, C. S., Shih, J. M., & Chang, R. S. (2013). Simplifying map reduce data processing. *IJCSE 8*(3), 219-226.

[69] Maatuk, A., Ali, A., & Rossiter, N. (2008) *Relational Database Migration: A Perspective, School of Computing, Engineering & Information Sciences*. In 19th International Conference on Database and Expert Systems Applications (DEXA'08). Newcastle upon Tyne, UK: Northumbria University.

[70] Manteghi, N., & Jahromi, S. K. (2012). Designing accounting ınformation system using SSADM1 Case Study: South Fars Power Generation Management Company (S.F.P.G.M.C). *Procedia Technology*, *1*, 308-312.

[71] Marjanović, Z., Aničić, N., & Babarogić, S. (2000) *Prošireni model Objekti-veze* [skripta]. Beograd, Srbija: FON LabIS.

[72] Marjanović, Z., Aničić, N., & Babarogić, S. (2011) *FON LabIS metodologija razvoja IS* [skripta]. Beograd, Srbija: FON LabIS.

[73] Marjanović, Z., Aničić, N., & Babarogić, S. (2014) *Prezentacije sa predmeta Projektovanje informacionih Sistema* [materijali sa predavanja], Beograd, Srbija: FON.

[74] Melton, J., Michels, J. E., Josifovski, V., Kulkarni, K., & Schwarz, P. (2002). SQL/MED: a status report. *ACM SIGMOD Record*, *31*(3), 81-89. doi: 10.1145/601858.601877

[75] Microsoft (2016). *Microsoft Azure.* Доступно на: https://azure.microsoft.com/en-us/, приступано: јул 2016.

[76] Microsoft (2017). *Use Dynamic Management Views (DMVs) to Monitor Analysis Services.* Доступно на: https://docs.microsoft.com/en-us/sql/analysis-services/instances/use-dynamic-management-views-dmvs-to-monitor-analysis-services, приступано: новембар 2017.

[77] MongoDB (2018) *The MongoDB 3.2 Manual.* Доступно на: https://docs.mongodb.com/v3.2/, приступано: јул 2018.

[78] Myers, B. (1998) A brief history of human-computer interaction technology. *Interactions*, 5(2), 44-54. New York, NY, USA: ACM. doi: 10.1145/274430.274436

[79] Nakabasami, K., Amagasa, T., & Kitagawa, H. (2013). *Querying MongoDB with LINQ in a Server-Side JavaScript Environment*. In 16th International Conference on NetworkBased Information Systems, 344-349. Washington, DC, USA: IEEE.

[80] Nance, C., Losser, T., Iype, R., & Harmon, G. (2013). *NOSQL vs RDBMS -why there is room for both*. In Proceedings of the Southern Association for Information Systems Conference, 111-116.

[81] Needham, D., Sinopoli, D., Dinglas, V., Berenholtz, S., Korupolu, R., Watson, S., Lubomski, L., Goeschel, C., & Pronovost, P. (2009). Improving data quality control in quality improvement projects. *International Journal for Quality in Health Care, 21*(2), 145-150.

[82] Nickel, M., & Tresp, V. (2013). *An Analysis of Tensor Models for Learning on Structured Data.* In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD 2013), 272-287. Berlin, Germany: Springer. doi: 10.1007/978-3-642-40991-2_18

[83] Niedritis A., Niedrite L., & Kozmina N. (2011) Performance Measurement Framework with Formal Indicator Definitions. In: Grabis J., & Kirikova M. (уредници), *Perspectives in Business Informatics Research. BIR 2011. Lecture Notes in Business Information Processing, vol 90,* 44-58. Berlin, Germany: Springer.

[84] Niu, N., Xu, L. D., & Bi, Z. (2013). Enterprise Information Systems Architecture - Analysis and Evaluation. *IEEE Transactions On Industrial Informatics, 9*(4), 2147-2154.

[85] Nyati, S.S., Pawar, S., & Ingle, R. (2014). *Performance Evaluation of Unstructured NoSQL data over distributed framework*. In International Conference on Advances in Computing, Communications and Informatics (ICACCI), 1623-1627. Washington, DC, USA: IEEE.

[86] OpenCypher. (2016). *OpenCypher*. Доступно на: http://www.opencypher.org/, приступано: август 2016.

[87] Oracle (2013). *Migrating Applications and Databases with Oracle Database 12c*. [White Paper]. Доступно на: http://www.oracle.com/technetwork/database/migration/migrating-to-oracle-database-wp-12c-1896125.pdf, приступано: август 2016.

[88] Oracle (2014). *Oracle Database Performance Tuning Guide 11g Release 2 (11.2)*. Доступно на: https://docs.oracle.com/cd/E11882_01/server.112/e41573.pdf, приступано: новембар 2017.

[89] Oracle (2016). *Oracle NoSQL Database Enterprise Edition, Version 18.1.* [White Paper] Доступно на: http://www.oracle.com/technetwork/products/nosqldb/learnmore/nosql-database-data-sheet-498054.pdf, приступано: август 2016.

[90] Oracle (2018a). *Managing optimizer statsitics.* Доступно на: https://docs.oracle.com/cd/B19306_01/server.102/b14211/stats.htm#i41282, приступано: април 2018.

[91] Oracle (2018b). *Table Expression.* Доступно на:: https://docs.oracle.com/javadb/10.6.2.1/ref/rreftableexpression.html#rreftableexpression, приступано: јул 2018.

[92] Oracle (2018c). *TimesTen In-Memory Database SQL Reference.* Доступно на: https://docs.oracle.com/cd/E11882_01/timesten.112/e21642/state.htm#TTSQL277 приступано: јун 2018.

[93] Panorama (2016). *2016 Report on ERP systems and enterprise software - A Panorama Consulting Solutions Research Report.* Доступно на: http://go.panorama-consulting.com/rs/panoramaconsulting/images/2016-ERP-Report.pdf, приступано: јун 2016.

[94] Polese, G., & Vacca, M. (2009). *A dialogue-based model for the query synchronization problem*. In IEEE 5th International Conference on Intelligent Computer Communication and Processing, 67-70. Washington, DC, USA: IEEE.

[95] Ponniah, P. (2007). *Data Modeling Fundamentals: A Practical Guide for IT Professionals*. Hoboken, NJ, USA: Wiley & Sons, Inc.

[96] Potey, M., Digrase, M., Deshmukh, G., & Nerkar, M. (2015). *Database Migration from Structured Database to non-Structured Database*. In Proceedings of International Conference on Recent Trends & Advancements in Engineering Technology (ICRTAET '15), 1, 1-3.

[97] RedisDB (2018) *Documentation.* Доступно на: https://redis.io/documentation, приступано: јул 2018.

[98] Rob, M. A. (2004). Issues of structured vs. object-oriented methodology of systems analysis and design. *Issues in Information Systems*, *5*(1), 275-280.

[99] Roijackers, J., & Fletcher, G. H. (2013). *On bridging relational and document-centric data stores*. In British National Conference on Databases, 135-148. Berlin, Germany: Springer.

[100] Royce, W. W. (1970). *Managing the development of large software systems*. In Proceedings of IEEE WESCON, 26(8), 328-338.

[101] Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *Unified Modeling Language Reference Manual*. Boston, MA, USA: Addison-Wesley.

[102] Sadalage P. J., & Fowler M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Boston, MA, USA: Addison-Wesley.

[103] Sakr, S., & Al-Naymat, G. (2010). Graph indexing and querying: a review. *International Journal of Web Information Systems*, *6*(2), 101-120.

[104] Sakr, S., & Pardede, E. (2011). *Graph data management: techniques and applications* (прво издање). Hershey, Pennsylvania, USA: IGI Global.

[105] Sellami, R., Bhiri, S., & Defude, B. (2014). *ODBAPI: a unified REST API for relational and NoSQL data stores*. In 2014 IEEE International Congress on Big Data, 653-660. Washington, DC, USA: IEEE.

[106] Shepelev, I.E. (2011). Identification of the hierarchical data structure. *Pattern Recognition and Image Analysis, 21*(2), 211-214.

[107] Shin, K., Hwang, C., & Jung, H. (2017). NoSQL Database Design Using UML Conceptual Data Model Based on Peter Chen's Framework. *International Journal of Applied Engineering Research 12*(5), 632-636.

[108] SolidIT (2018). DB-Engines Ranking. Доступно на: https://db-engines.com/en/ranking, приступано: фебруар 2018.

[109] Sommerville, I. (2001). *Software engineering* (девето издање). Boston, MA, USA: Addison-Wesley.

[110] *SQLite* (2016). Доступно на: https://www.sqlite.org/, приступано: август 2016.

[111] Sullivan, D. (2015). *NoSQL for Mere Mortals*. Boston, MA, USA: Addison-Wesley.

[112] Tahara, D., Diamond, T., & Abadi, D. J. (2014). *Sinew: a SQL system for multi-structured data*. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data, 815-826. New York, NY, USA: ACM.

[113] Tatemura, J., Po, O., Hsiung, W. P., & Hacigümüş, H. (2012). *Partiqle: An elastic SQL engine over key-value stores*. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 629-632. New York, NY, USA: ACM.

[114] Valer, H., Sauer, C., & Härder, T. (2013). *XQuery processing over NoSQL stores*. In Proceedings of the 25th GI-Workshop Grundlagen von Datenbanken 2013, 75-80.

[115] van Ombergen, S. (2014) *A Comparison of Five Document-Store Query Languages - Finding a suitable standard* (мастер теза), University of Amsterdam, Amsterdam, Netherlands.

[116] Vilaça, R., Cruz, F., Pereira, J., & Oliveira, R. (2013). *An Effective Scalable SQL Engine for NoSQL Databases*. In IFIP International Conference on Distributed Applications and Interoperable Systems, 155-168. Berlin, Germany: Springer.

[117] W3C (2008). *SparQL Query Language for RDF*. Доступно на: https://www.w3.org/TR/rdf-sparql-query/, приступано: јул 2016.

[118] Winand, M. (2017). *What's New in SQL:2016*. Доступно на: https://modern-sql.com/blog/2017-06/whats-new-in-sql-2016, приступано: април 2018.

[119] Zhang, H. (2008) On the Distribution of Software Faults. *IEEE Transactions on Software Engineering, 34*(2), 301-302.

[120] Zhu, J., & Wang, A. (2012). *Data Modeling for Big Data* [White paper]. Beijing, China: CA technologies.

## Биографија

Срђа Бјеладиновић рођен је 09.04.1985. године у Београду. Трећу београдску гимназију завршио је 2004. године као носилац дипломе „Вук Караџић“. Исте године уписао је Факултет организационих наука, одсек Информациони системи и технологије. Дипломирао је 17.06.2009. године, са просечном оценом 8,91 и оценом 10 на дипломском раду. Године 2009. уписује дипломске академске студије – мастер на Факултету организационих наука, студијски програм Информациони системи и технологије, студијско подручје Информациони системи. Мастер студије завршава 14.07.2011. године са просечном оценом студирања 10,0 и оценом 10 на завршном мастер раду. Године 2011. уписује докторске академске студије – студијски програм Информациони системи и менаџмент, изборно подручје Информациони системи.

Од 10.2009. године ангажован је као демонстратор на Катедри за информационе системе, од маја 2010. године као сарадник у настави, а од маја 2012. године у звању асистента на споменутој катедри. Неки од предмета на којима бива ангажован су: Базе података, Пројектовање информационих система и Администрација базе података. Од 2011. године, ангажован је и као предавач од стране *Oracle University*. Одржао је више десетина курсева у земљи и иностранству, као и путем Интернета.

Објавио је више радова, међу којима су и радови настали током писања докторске дисертације:

1. **Bjeladinovic, S.** (2018) A fresh approach for hybrid SQL/NoSQL database design based on data structuredness. *Enterprise Information Systems, ISSN(Online)*: *1751-7583*, *12*(8-9), 1202-1220. Milton, Abingdon, UK: Taylor & Francis. IF(2017)=1.683 (M22). doi: 10.1080/17517575.2018.1446102
2. **Bjeladinović, S.**, Marjanović, Z. (2016) Primena virtuelizacije u cilju integracije relacionih i NoSQL baza podataka, na primeru Oracle-a. *Info M*, *57/2016*, 4-10, (M53)

# Изјава о ауторству

Име и презиме аутора _____

Број индекса _____

**Изјављујем**

да је докторска дисертација под насловом

_____

_____

- резултат сопственог истраживачког рада;
- да дисертација у целини ни у деловима није била предложена за стицање друге дипломе према студијским програмима других високошколских установа;
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио/ла интелектуалну својину других лица.

**Потпис аутора**

У Београду, _____

_____

# Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора _____

Број индекса _____

Студијски програм _____

Наслов рада _____

Ментор _____

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла ради похрањена у **Дигиталном репозиторијуму Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског назива доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

**Потпис аутора**

У Београду, _____

_____

# Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

_____

_____

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигиталном репозиторијуму Универзитета у Београду и доступну у отвореном приступу могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство (CC BY)

2. Ауторство – некомерцијално (CC BY-NC)

3. Ауторство – некомерцијално – без прерада (CC BY-NC-ND)

4. Ауторство – некомерцијално – делити под истим условима (CC BY-NC-SA)

5. Ауторство – без прерада (CC BY-ND)

6. Ауторство – делити под истим условима (CC BY-SA)

(Молимо да заокружите само једну од шест понуђених лиценци.
Кратак опис лиценци је саставни део ове изјаве).

**Потпис аутора**

У Београду, _____

_____

1. **Ауторство**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. **Ауторство – некомерцијално**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. **Ауторство – некомерцијално – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. **Ауторство – некомерцијално – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. **Ауторство – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. **Ауторство – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.