VIŠJA STROKOVNA ŠOLA ACADEMIA

MARIBOR

Primerjava zmogljivosti podatkovnih baz PostgreSQL in Oracle na primeru Spring Boot aplikacije pri obdelavi velikih količin podatkov

Kandidat: Gregor Presker

Vrsta študija: študent izrednega študija

Študijski program: Informatika

Mentor predavatelj: mag. Ervin Schaff

Mentor v podjetju: Janko Livić, dipl. inž. inf. tehnol.

Lektorica: Karin Vidmar, prof. slov. in angl.

Maribor, 2024

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani Gregor Presker izjavljam, da sem avtor diplomskega dela z naslovom Primerjava zmogljivosti podatkovnih baz PostgreSQL in Oracle na primeru Spring Boot aplikacije pri obdelavi velikih količin podatkov, ki sem ga napisal pod mentorstvom mag. Ervina Schaffa.

S svojim podpisom zagotavljam, da:

- je predloženo delo izključno rezultat mojega dela;
- sem poskrbel, da so dela in mnenja drugih avtorjev, ki jih uporabljam v predloženi nalogi, navedena oz. citirana skladno s pravili Višje strokovne šole Academia Maribor;
- se zavedam, da je plagiatorstvo – predstavljanje tujih del oz. misli kot mojih lastnih, kaznivo po Zakonu o avtorskih in sorodnih pravicah (Uradni list RS, št. 16/07 – uradno prečiščeno besedilo, 68/08, 110/13, 56/15 in 63/16 – ZKUASP); prekršek pa podleže tudi ukrepom Višje strokovne šole Academia Maribor skladno z njenimi pravili;
- skladno z 32.a členom ZASP dovoljujem Višji strokovni šoli Academia Maribor objavo diplomskega dela na spletnem portalu šole.

Slovenska Bistrica, september 2024 Podpis študenta:

ZAHVALA

Želel bi se zahvaliti mag. Ervinu Schaffu in Janku Liviću, dipl. inž. inf. tehnol. za njuno vodenje in podporo pri pisanju diplomskega dela in izvedbi prakse. Mag. Ervin Schaff je bil moj mentor med pisanjem diplome, vedno me je spodbujal in mi posredoval potrebne povratne informacije, da sem lahko uspešno napisal diplomsko nalogo. Prav tako izražam iskreno zahvalo Janku Liviću, ki me ni le mentoriral med pisanjem diplomske naloge, ampak je tudi igral ključno vlogo pri izvedbi prakse. Njuno strokovno znanje, njuna potrpežljivost in predanost sta bili ključni pri pridobivanju znanja in izkušenj na področju računalniškega programiranja ter pri pisanju diplomskega dela. Vesel sem, da sem se imel priložnost učiti od tako izjemnih mentorjev, strokovnjakov.

POVZETEK

Diplomska naloga je raziskovanje učinkovitosti in zmogljivosti uporabe tehnologij, pri čemer je osrednji poudarek na ogrodju Spring Boot ter bazah podatkov Oracle in PostgreSQL, v kontekstu obdelave masovnih podatkov. Raziskava zajema temeljito analizo teh tehnologij, oceno njihovih prednosti in slabosti ter primerjalno oceno njihove uspešnosti in uporabnosti pri obdelavi velikih količin podatkov.

V diplomski nalogi smo si zastavili cilj odgovoriti na naslednja raziskovalna vprašanja:
a) Kaj je Big Data? b) Ali se hitrost vnosa neskončnega toka podatkov razlikuje pri Oracle bazi v primerjavi z PostgreSQL bazo? c) Kaj so razlogi za razliko v zmogljivosti podatkovnih baz pri obdelavi večjih količin podatkov? č) Ali sta Oracle in PostgreSQL najprimernejši bazi za neskončni tok podatkov?

V ta namen smo najprej opredelili osnovno teorijo, ki je zajemala: razčlenitev Spring Boot ogrodja, preučitev programskega jezika Java in funkcij Spring Boot ogrodja ter njegovih prednosti in slabosti, preučitev baz podatkov Oracle in PostgreSQL. Vsaka podatkovna baza je natančno analizirana, pri čemer so zajete njene lastnosti, prednosti in slabosti, kar na koncu vodi do primerjalne ocene. Predstavili smo še Docker platformo in pojem Big Data.

Izdelali smo aplikacijo s pomočjo ogrodja Spring Boot. Napravili smo analizo specifikacij aplikacije, predstavili smo razvojno okolje, postopek izdelave aplikacije s Spring Boot ogrodjem in postopek izdelave podatkovnega modela v podatkovnih bazah Oracle in PostgreSQL.

Po predstavitvi izdelave aplikacije smo opravili meritve, kjer se je izvedla empirična analiza rezultatov podatkovnih baz Oracle in PostgreSQL. Meritve hitrosti so se natančno beležile in analizirale, to nam je dalo dragocen vpogled v učinkovitost in uspešnost obeh sistemov baz podatkov. Ugotovitve iz faze merjenja kot tudi ugotovitve iz teorije so nam služile kot osnova za potrditev ali zavrnitev zastavljenih hipotez v diplomski nalogi.

V sklepnem delu diplomske naloge smo povzeli ugotovitve iz teoretičnega in empiričnega dela diplomskega dela.

Ključne besede: masovni podatki, Spring Boot, Java, Oracle, PostgreSQL, Docker.

ABSTRACT

Comparison of PostgreSQL and Oracle database performance on the example of Spring Boot application when processing large amounts of data

The thesis is a research of the efficiency and capability of utilizing technologies with a primary focus on the Spring Boot framework and Oracle and PostgreSQL databases, all in the context of processing Big Data. The research includes a thorough analysis of these technologies, an assessment of their strengths and weaknesses, and a comparative assessment of their performance and usability in processing large amounts of data.

In the thesis, we aim to address the following research questions: a) What is Big Data? b) Does the data ingestion speed differ between Oracle and PostgreSQL databases when dealing with an infinite data stream? c) What are the reasons for the performance difference between databases in processing large volumes of data? č) Are Oracle and PostgreSQL the most suitable databases for an infinite data stream?

To this end, we first defined the basic theory, which included: dissecting the Spring Boot framework, examining the Java programming language and Spring Boot framework features, as well as their pros and cons. We also examined the Oracle and PostgreSQL databases, conducting a thorough analysis of each database's characteristics, advantages, and disadvantages, ultimately leading to a comparative evaluation. Additionally, we introduced the Docker platform and the concept of Big Data.

We developed an application using the Spring Boot framework. We conducted an analysis of the application specifications, presented the development environment, as well as the application development process, using the Spring Boot framework, and the process of creating a data model in Oracle and PostgreSQL databases.

Following the application development presentation, we conducted measurements performing an empirical analysis of the results from the Oracle and PostgreSQL databases. Speed measurements were meticulously recorded and analysed, providing valuable insights into the efficiency and performance of both database systems. The findings from the measurement

phase, as well as those from the theoretical research, served as the basis for confirming or refuting the hypotheses set forth in the thesis.

In the concluding section of the thesis, we summarized the findings from the theoretical and empirical parts of the research.

Keywords: Big Data, Spring Boot, Java, Oracle, PostgreSQL, Docker.

KAZALO VSEBINE

KAZALO SLIK

KAZALO TABEL

1 U VOD

1.1 Opis področja in opredelitev problema

OPREDELITEV PODROČJA

Big Data (v nadaljevanju masovni podatki) zajemajo ogromne količine raznolikih podatkov, ki za organizacije predstavljajo velik izziv in priložnost. Zaradi velikega obsega in raznolikosti so potrebne napredne tehnologije in analitika za pridobivanje dragocenih vpogledov. Od interakcij v družbenih medijih do podatkov senzorjev – veliki viri podatkov so raznoliki in številni ter ponujajo potencialne vpoglede, ki lahko spodbujajo inovacije in izboljšajo procese odločanja. Kljub zapletenosti lahko učinkovito izkoriščanje masovnih podatkov privede do izboljšane operativne učinkovitosti, izboljšanih uporabniških izkušenj in konkurenčnih prednosti v današnjem okolju, ki temelji na podatkih. Vendar pa upravljanje, obdelava in analiza teh podatkov predstavlja velike izzive, ki od organizacij zahtevajo, da analizirajo primernost trenutne uporabljene strojne in programske opreme za hrambo in obdelavo podatkov in jo po potrebi nadgradijo ali zamenjajo. Predvsem je pomembno, da se organizacija, ki se ukvarja z masovnimi podatki, odloči za izbiro najprimernejše podatkovne baze, saj le-ta neposredno vpliva na njihovo sposobnost učinkovitega upravljanja, obdelave in pridobivanja vpogledov iz velikih količin raznolikih podatkov. Izbira najprimernejše baze podatkov pomembno vpliva na zmožnost organizacije, da pridobi vrednost iz masovnih podatkov in ohrani konkurenčno prednost v današnjem okolju, ki temelji na podatkih.

OPREDELITEV PROBLEMA

Ena izmed najpomembnejših lastnosti podatkovne baze je njena zmožnost hitre obdelave podatkov, saj to omogoča čimprejšnjo pridobitev relevantnih poročil za pravočasno sprejemanje odločitev v podjetju. Z vidika hitrosti obdelave podatkov bomo primerjali podatkovni zbirki Oracle in PostgreSQL. S predelavo relevantne teorije bomo skušali ugotoviti, ali obstajajo druge bolj primerne podatkovne baze za neskončni tok podatkov. Pri tem smo si zastavili naslednja raziskovalna vprašanja:

V1: Kaj je Big Data?

Podvprašanje 1: Kako bi lahko simulirali Big Data?

V2: Ali se hitrost vnosa neskončnega toka podatkov razlikuje pri Oracle bazi v primerjavi s PostgreSQL bazo?

V3: Kaj so razlogi za razliko v zmogljivosti podatkovnih baz pri obdelavi večjih količin podatkov?

V4: Ali sta Oracle in PostgreSQL najprimernejši bazi za neskončni tok podatkov?

1.2 Namen, cilji in osnovne trditve

NAMEN DIPLOMSKEGA DELA

Namen diplomske naloge, ki smo ga obravnavali, je torej, katera baza podatkov, Oracle ali PostgreSQL, izkazuje večjo hitrost pri obdelavi velike količine podatkov. Želeli smo tudi ugotoviti, katere podatkovne baze so na splošno najprimernejše za obdelavo masovnih podatkov. Z analizo meritev hitrosti je ta diplomska naloga zagotovila uporabne vpoglede v učinkovitost in uspešnost Oracle in PostgreSQL baz podatkov pri obravnavanju nalog obdelave podatkov v velikem obsegu. Z identifikacijo baze podatkov, ki ponuja največjo hitrost delovanja za scenarije masovnih podatkov, lahko razvijalci in organizacije sprejemajo informirane odločitve pri izbiri najprimernejše baze podatkov, s čimer povečajo učinkovitost obdelave velikih količin podatkov.

CILJI DIPLOMSKEGA DELA

- Predstaviti osnovno teorijo tehnologij Spring Boot in Java.
- Predstaviti osnovno teorijo podatkovnih baz Oracle in PostgreSQL ter jih medsebojno primerjati.
- Predstaviti osnovno teorijo masovnih podatkov.
- Predstaviti osnovno teorijo orodja Docker.
- Predstaviti izdelavo Spring Boot aplikacije.
- Opraviti testiranja Oracle in PostgreSQL podatkovnih baz z vidika hitrosti obdelave velikih količin podatkov ter tako odgovoriti na zastavljena raziskovalna vprašanja.

HIPOTEZE

Da bi odgovorili na postavljena raziskovalna vprašanja naloge, smo oblikovali naslednje hipoteze:

- H1: Z datotekami csv lahko simuliramo Big Data – neskončen tok podatkov.
- H2: Hitrost vnosa neskončnega toka podatkov pri Oracle bazi podatkov je precej večja kot pri PostgreSQL.
- H3: Razlika v zmogljivosti med bazama podatkov PostgreSQL in Oracle pri obdelavi večjih količin podatkov v Spring Boot aplikaciji je odvisna samo od specifičnih značilnosti podatkov, ki se obdelujejo, in kompleksnosti poizvedb.
- H4: Za neskončni tok podatkov obstajajo bolj primerne podatkovne baze.

1.3 Predpostavke in omejitve

PREDPOSTAVKE

V diplomski nalogi predpostavljamo, da:

- se obstoječa literatura in viri v času pisanja diplomskega dela ne bo spremenila;
- sta tako Oracle kot PostgreSQL podatkovni zbirki optimalno konfigurirani za testiranje delovanja z ustreznimi nastavitvami in optimizacijami;
- uporabljene poizvedbe za testiranje zmogljivosti podatkovnih zbirk predstavljajo realne primere obdelave podatkov;
- izvedeni testi zmogljivosti zagotavljajo pošteno in objektivno primerjavo med podatkovnima bazama Oracle in PostgreSQL v smislu hitrosti obdelave podatkov v scenarijih obdelave velikih količin podatkov;
- prenos podatkov iz CSV datoteke v podatkovni bazi Oracle in PostgreSQL simulira vnos izseka neskončnega toka podatkov v podatkovni bazi;
- so različice baz podatkov Oracle in PostgreSQL, ki se uporabljajo za testiranje zmogljivosti, reprezentativne za trenutne ali splošno razširjene različice in da so morebitne razlike v zmogljivosti med različicami zanemarljive;
- je strojna oprema, ki se uporablja za testiranje zmogljivosti, zadostna in ustrezno opremljena za podporo delovnih obremenitev in operacij baze podatkov;
- so meritve delovanja podatkovnih baz, zbrane med testiranjem, natančne in zanesljive ter da so vsa odstopanja ali variacije v rezultatih minimalna in sprejemljiva;

- se značilnosti delovanja podatkovne baze, opažene med testiranjem v določenem obsegu, lahko ekstrapolirajo na večje nabore podatkov in višje ravni delovne obremenitve brez znatnega poslabšanja zmogljivosti.

OMEJITVE

Omejitve diplomske naloge so:

- v diplomski nalogi se omejujemo na uporabo slovenskih in angleških virov in literature;
- v teoretičnem delu se omejujemo na temo masovnih podatkov;
- v teoretičnem delu bodo predstavljene splošne značilnosti tehnologij in podatkovnih zbirk, kot so Spring Boot, Java, Oracle, PostgreSQL itd.;
- v empiričnem delu diplomskega dela se omejujemo na uporabo podatkovnih zbirk PostgreSQL (verzija: 14) in Oracle (izdaja: Express Edition (XE), verzija: 21c) za testiranje zmogljivosti podatkovnih zbirk pri obdelavi velikih količin podatkov;
- v empiričnem delu se omejujemo pri izvajanju aplikacije na uporabo enega samega računalnika;
- v empiričnem delu diplomskega dela se omejujemo na osem različnih SQL poizvedb za testiranje hitrosti delovanja obdelave velikih količin podatkov;
- v empiričnem delu diplomskega dela se omejujemo na velikost CSV datoteke, ki predstavlja naše masovne podatke – izsek neskončnega toka podatkov, na skupno sto devetinštirideset tisoč vrstic.

1.4 Uporabljene raziskovalne metode

V teoretičnem delu smo za predstavitev teorije, osnovnih pojmov in teoretičnih izhodišč uporabili deskriptivni pristop. Za teoretična dognanja smo uporabili metodo analize literature in virov, s pomočjo katere smo potrdili ali zavrnili zastavljene hipoteze.

V diplomski nalogi smo potrdili ali zavrnili hipotezo H1 in H4 s pomočjo predelave relevantne teorije.

V empiričnem delu smo uporabili metodo zbiranja podatkov. Podatke smo pridobili s pomočjo aplikacije, ki meri hitrost vnosa podatkov v podatkovni bazi Oracle in PostgreSQL ter hitrost izvrševanja poizvedb v podatkovnih bazah. Meritve hitrosti predstavljajo ključne

podatke, ki smo jih uporabili za potrditev ali zavrnitev zastavljenih hipotez v empiričnem delu naše raziskave.

V diplomski nalogi smo potrdili ali ovrgli hipotezo H2 s pomočjo rezultatov meritev aplikacije.

Na podlagi meritev smo prišli tudi do zaključka glede zmogljivosti Oracle in PostgreSQL baz pri obdelavi velikih količin podatkov.

Hipotezo H3 smo potrdili ali ovrgli s pomočjo logičnega sklepanja glede na znana dejstva o pripravi in izdelavi aplikacije.

V sklepnem delu smo uporabili metodo sinteze, s katero smo na podlagi dobljenih rezultatov prišli do zaključnih ugotovitev.

2 UPORABA IN UČENJE TEHNOLOGIJ

2.1 Spring Boot

Spring Framework je zelo priljubljeno in široko uporabljano Java ogrodje za razvoj spletnih in poslovnih aplikacij. Obravnava potrebe sodobnih poslovnih aplikacij, kot so varnost, podpora za NoSQL baze podatkov, obdelava velikih količin podatkov, integracija z drugimi sistemi, itd (Reddy, 2017). Spring Boot je razširitev Spring Frameworka, poenostavlja proces razvoja spletnih aplikacij in mikrostoritev. To dosega s tremi ključnimi funkcijami (IBM, 2024):

1. samodejna konfiguracija,
2. mnenjski pristop h konfiguraciji,
3. možnost ustvarjanja samostojnih aplikacij.

Te funkcije omogočajo namestitev aplikacije, osnovane na Spring Frameworku, z minimalno nastavitvijo in konfiguracijo. Spring Boot izhaja iz Spring Frameworka in je njegova razširitev. Spring Boot poenostavlja proces ustvarjanja aplikacij, saj odstrani veliko odvečne kode in konfiguracije, ki tradicionalno spremljata razvoj poslovnih Java aplikacij. S Spring Bootom se lahko gradijo samostojne Spring aplikacije, ki so pripravljene za takojšnje izvajanje z zelo malo ali nič potrebe po konfiguraciji (IBM, 2024).

»Ena od ključnih lastnosti Spring Boota je podpora za vgrajene strežnike, kot so Tomcat, Jetty in Undertow. To pomeni, da zunanji spletni strežniki niso več potrebni. Vdelan pristop olajša tudi lokalni zagon in testiranje aplikacij, kar pospeši razvojni cikel« (HackerNoon, 2024).

Podrobnejšo razlago značilnosti ogrodja Spring Boot podajamo v nadaljevanju.

2.1.1 Programski jezik Java

Java je programski jezik, ki temelji na razredih in objektih, zasnovan z minimalnimi implementacijskimi odvisnostmi. Razvijalcem omogoča, da napišejo kodo enkrat in jo poganjajo kjerkoli, saj prevedena Java koda deluje na vseh platformah, ki podpirajo Javo, brez potrebe po ponovnem prevajanju. Od svoje prve izdaje leta 1995 se Java uporablja za razvoj namiznih, spletnih in mobilnih aplikacij. Znana je po preprostosti, robustnosti in varnostnih funkcijah, kar jo naredi priljubljeno za podjetniške aplikacije. Java je enostavna za pisanje, prevajanje in odpravljanje napak. Ponuja tudi podporo za ponovno uporabo kode in modularne

programe, saj se Java aplikacije prevajajo v bitno kodo, ki lahko deluje na kateremkoli Java Virtual Machine (JVM) (Geeksforgeeks, 2024).

Izvedba Java programa vključuje naslednje korake (Geeksforgeeks, 2024):

1. ustvarjanje programa,
2. sestavljanje programa,
3. zagon programa.

Glavne značilnosti programskega jezika Java so (Geeksforgeeks, 2024):

1. Platformska neodvisnost: Java omogoča platformsko neodvisnost, saj kompilator pretvori izvorno kodo v bajtno kodo, ki jo nato JVM izvaja. Ta bajtna koda lahko teče na katerikoli platformi, bodisi Windows, Linux ali macOS.
2. Objektno orientiran programski jezik: Programi se organizirajo kot zbirka objektov, od katerih vsak predstavlja primer razreda. Temeljni koncepti objektno orientiranega programiranja vključujejo abstrakcijo, enkapsulacijo, dedovanje in polimorfizem.
3. Preprostost: Java se uvršča med preproste jezike, saj ne vključuje kompleksnih lastnosti, kot so kazalniki, preobremenitev operatorjev, večkratno dedovanje in eksplicitna dodelitev pomnilnika.
4. Varnost: V programskem jeziku Java se ne uporabljajo kazalniki. Programi v Javi se izvajajo v okolju, ki je neodvisno od operacijskega sistema, kar prispeva k večji varnosti programov.
5. Razdeljenost: Z uporabo programskega jezika Java lahko ustvarimo razdeljene aplikacije.
6. Večnitnost: To je funkcija Jave, ki omogoča sočasno izvajanje dveh ali več delov programa za maksimalno izkoriščanje CPU-ja.
7. Prenosljivost: Koda, napisana na enem računalniku, se lahko izvede na drugem računalniku.
8. Visoka učinkovitost: Arhitektura Jave je zasnovana tako, da zmanjšuje obremenitve med izvajanjem. V nekaterih primerih Java uporablja tudi JIT (Just In Time) prevajalnik, ki prevaja kodo po zahtevi in prevaja le tiste metode, ki so dejansko klicane.
9. Dinamična prilagodljivost: Zagotavlja prilagodljivost pri dodajanju razredov, novih metod obstoječim razredom in celo ustvarjanju novih razredov preko podrazredov.

10. Izvajanje v peskovniku: Java programi se izvajajo v ločenem prostoru, ki uporabnikom omogoča izvajanje njihovih aplikacij, ne da bi vplivali na osnovni sistem.
11. Načelo "Pisati enkrat, izvajati kjerkoli" je ključno načelo Jave, ki omogoča, da se Java programi lahko izvajajo na katerikoli platformi, ne glede na strojno arhitekturo, saj se koda pretvori v neodvisno bajtno kodo.

Velika moč Jave je njena obsežna standardna knjižnica, ki zagotavlja bogat nabor API-jev. API-ji v Javi vključujejo razrede, vmesnike in uporabniške vmesnike. Razvijalcem omogočajo integracijo različnih aplikacij in spletnih strani ter nudijo informacije v realnem času. Java razvijalci potrebujejo Java API-je za (Ravikiran, 2023):

1. Izboljšanje poslovne tehnike: Uvedba API-jev omogoča mnogim podjetjem sproščanje zasebnih podatkov za ustvarjanje novih idej, odpravljanje napak in prejemanje novih načinov za izboljšanje operacij.
2. Ustvarjanje zmogljivih aplikacij: Na primer, API-ji strankam v bančništvu omogočajo upravljanje svojih financ digitalno.

Java močno poudarja varnost, zaradi česar je postala priljubljena izbira za gradnjo aplikacij, še posebej v industrijah, kot so finance ali zdravstvo. Varnost v Javi vključuje obsežen nabor API-jev, orodij in implementacij pogosto uporabljenih varnostnih algoritmov, mehanizmov in protokolov. API-ji za varnost v Javi pokrivajo širok spekter področij, vključno s kriptografijo, infrastrukturo javnih ključev, varno komunikacijo, avtentikacijo in nadzorom dostopa. Tehnologija varnosti v Javi razvijalcu zagotavlja celovit varnostni okvir za pisanje aplikacij, uporabniku ali skrbniku pa ponuja niz orodij za varno upravljanje aplikacij. Mehanizem varnega nalaganja in preverjanja razredov zagotavlja, da se izvaja le legitimna Java koda (Oracle, 2024).

2.1.2 Značilnosti ogrodja Spring Boot

Java Spring Boot je del ogrodij znotraj večjega ekosistema Spring, ki poenostavlja postopek izdelave aplikacij s pomočjo ogrodja Spring. Zagotavlja niz orodij, ki omogočajo enostavno ustvarjanje samostojnih aplikacij na osnovi Springa (Medium, 2023).

V nadaljevanju predstavljamo glavne značilnosti ogrodja Spring Boot (Medium, 2023):

1. Samodejna konfiguracija: Spring Boot uporablja samodejno konfiguracijo, ki avtomatsko konfigurira aplikacijo na podlagi knjižnic in odvisnosti, vključenih v projekt.
2. Neodvisnost: Aplikacije Spring Boot so samozadostne in delujejo neodvisno, kar omogoča enostavno namestitev in upravljanje z njimi.
3. Prilagodljivost nastavitev: Spring Boot zagotavlja smiselne privzete nastavitve konfiguracije, v kolikor je potreba, se lahko te spremenijo oziroma prilagodijo.
4. Podpora za mikrostoritve: Spring Boot je primeren za gradnjo mikrostoritev (podpora za vdelane kontejnerje).
5. Integracija z ekosistemom Spring: Integrira se lahko z drugimi Spring projekti, kot so Spring Data, Spring Security in Spring Cloud, kar omogoča razvoj kompleksnih aplikacij.

Zraven že zgoraj naštetih značilnosti še lahko dodamo (Staver, 2021):

6. Upravljanje z odvisnostmi: Spring Boot implicitno vključi potrebne odvisnosti tretjih oseb za različne vrste aplikacij. Te odvisnosti so zagotovljene preko tako imenovanih začetnih paketov. Začetni paketi so zbirke priročnih opisnikov odvisnosti, ki se jih lahko vključi v aplikacijo.
7. Vgrajeni spletni strežnik: Spring Boot aplikacija vključuje vgrajeni spletni strežnik. Aplikacija se izvaja kot izvršljiva datoteka s pomočjo vgrajenega strežnika. Ogrodje zagotavlja različne začetne pakete za različne HTTP strežnike.
8. Poenostavljeno testiranje: Hitrost QA testiranja se lahko poveča z uporabo intuitivnih privzetih nastavitev za enotska in integracijska testiranja.

2.1.3 Prednosti in slabosti ogrodja

Razvijalci imajo veliko razlogov, da izberejo Spring Boot za izgradnjo mikrostoritev za mobilne in spletne aplikacije. V nadaljevanju navajamo prednosti in slabosti uporabe Spring Boota (AdServio, 2022).

Prednosti (AdServio, 2022):

1. Dobro delovanje z več vgrajenimi servletnimi (razred, ki razširja zmogljivosti strežnikov, ki gostijo aplikacije, razvite z uporabo Java Servlet API) kontejnerji.

2. Zaganjanje aplikacije (Bootstraping) prihrani pomnilniški prostor. Spring Boot uporablja zaganjalnik Boot za prevajanje izvorne programske opreme. Ta tehnika zaganjanja omogoča uporabnikom, da prihranijo spomin na svojih napravah in da se aplikacije hitro naložijo in zaženejo.
3. Zmanjšanje ponavljajoče se kode. Vgrajena podatkovna baza in vgrajeni strežnik (Tomcat) v Spring Bootu zmanjšata ali odpravita ponavljajočo se kodo, ki jo običajno potrebujemo za nastavitev aplikacije. Brez velike količine ponavljajoče se kode lahko razvojne ekipe skrajšajo svoj čas razvoja in cikle posodabljanja.
4. Ni potrebna konfiguracija v XML. Razvijalci lahko izberejo, ali bodo uporabljali anotacije ali konfiguracije v XML.
5. Datoteke WAR niso potrebne. Spring Boot lahko uporablja datoteke WAR (spletne aplikacijske datoteke), vendar te niso potrebne. Spring Boot se lahko namesto na WAR datoteke zanaša na datoteke JAR.
6. Velika skupnost uporabnikov. Spring Boot obsega veliko skupnost uporabnikov, ki radi delijo svoje vpoglede, projekte in nudijo pomoč.

Slabosti (AdServio, 2022):

1. velike datoteke za distribucijo,
2. dolga čakalna doba za zamenjavo zastarelih sistemov s Spring Boot aplikacijami,
3. nesposobnost gradnje velikih, monolitnih aplikacij,
4. potreba po zanašanju na Springov ekosistem namesto uporabe drugih orodij.

2.2 Podatkovne baze

V nadaljevanju podajamo dve različni definiciji podatkovnih baz, ki po našem mnenju dobro definirata njihovo namembnost in funkcionalnost.

»Podatkovna baza je informacija, ki je vzpostavljena za enostaven dostop, upravljanje in posodabljanje. Računalniške podatkovne baze običajno hranijo skupine podatkovnih zapisov ali datotek, ki vsebujejo informacije, kot so prodajne transakcije, podatki o strankah, finančni podatki in informacije o izdelkih. Podatkovne baze se uporabljajo za shranjevanje, vzdrževanje in dostop do vseh vrst podatkov. Zbirajo informacije o ljudeh, krajih ali stvareh. Te informacije

so zbrane na enem mestu, tako da jih je mogoče opazovati in analizirati. Podatkovne baze lahko razumemo kot organizirano zbirko informacij« (Lutkevich, Hughes in Gillis 2023).

»Podatkovna baza je organizirana zbirka strukturiranih informacij ali podatkov, običajno shranjenih elektronsko v računalniškem sistemu. Podatkovno bazo običajno nadzoruje sistem za upravljanje podatkovnih baz (DBMS). Podatki in DBMS, skupaj z aplikacijami, ki so z njimi povezane, tvorijo podatkovni sistem oziroma podatkovno bazo. Podatki v najpogostejših vrstah podatkovnih baz, ki so danes v uporabi, so običajno modelirani v vrstah in stolpcih v seriji tabel, da bi bilo obdelovanje in poizvedovanje podatkov učinkovito. Podatki se nato lahko enostavno dostopajo, upravljajo, spreminjajo, posodabljajo, nadzorujejo in organizirajo. Večina podatkovnih baz uporablja strukturirani poizvedovalni jezik (SQL) za pisanje in poizvedovanje podatkov« (Oracle, 2024).

SQL je programski jezik, ki ga skoraj vse relacijske podatkovne baze uporabljajo za poizvedovanje, manipulacijo ter določanje podatkov, pa tudi za zagotavljanje nadzora dostopa. Razvit je bil v sedemdesetih letih prejšnjega stoletja, prvotno pri podjetju IBM, vendar je bil Oracle glavni prispevek, kar je privedlo do standardizacije SQL ANSI. Kasneje so ga različna podjetja, kot so IBM, Oracle in Microsoft, razširila s številnimi dodatki (Oracle, 2024).

Sistem za upravljanje podatkovnih baz (DBMS) omogoča ustvarjanje, upravljanje in vzdrževanje podatkovnih baz. Omogoča tudi ustvarjanje, branje, posodabljanje in brisanje podatkov v podatkovni bazi. DBMS zagotavlja fizično in logično neodvisnost od podatkov. To pomeni, da uporabniki in aplikacije ne potrebujejo poznati fizične ali logične lokacije podatkov v bazi. DBMS tudi omejuje in nadzoruje dostop do podatkovne baze ter omogoča različne poglede na isto shemo podatkovne baze več uporabnikom. Poleg tega DBMS prinaša tudi dodatne funkcionalnosti, kot so upravljanje transakcij, zagotavljanje varnosti podatkov, izvajanje poizvedb ter optimizacija učinkovitosti delovanja podatkovne baze (Lutkevich, Hughes in Gillis 2023).

Zelo pomembna značilnost podatkovnih baz je njihova zmogljivost obdelave podatkov. Obstaja več dejavnikov, ki lahko vplivajo na zmogljivost baze podatkov (Dragonfly, 2024):

- Strojna oprema: Hitrost in zmogljivost strojne opreme (CPU, pomnilnik, sistem za shranjevanje) lahko bistveno vplivata na zmogljivost baze podatkov.
- Dizajn baze podatkov: torej kako so strukturirane tabele in indeksi. Neučinkovita struktura tabel, slabo indeksirane baze podatkov lahko povzročijo počasno izvajanje poizvedb.

- Optimizacija poizvedb: Slabo zasnovane poizvedbe lahko pomembno vplivajo na hitrost izvajanja poizvedb in po nepotrebnem porabljajo resurse.
- Sočasnost: V kolikor imamo večjo število uporabnikov ali procesov, ki dostopajo do baze podatkov, to poveča obremenjenost CPU in poveča se tveganje za nastanek morebitnih ozkih grl.
- Upravljanje delovne obremenitve: Sistemi za upravljanje delovne obremenitve pomagajo učinkovito določiti prednostne naloge za optimizacijo zmogljivosti.
- Omrežna latenca: Če je strežnik baze podatkov gostovan na daljavo, lahko omrežna latenca vpliva na zmogljivost baze podatkov.

2.2.1 Vrste podatkovnih baz

Obstaja veliko različnih vrst podatkovnih baz. Najboljša podatkovna baza za določeno organizacijo je odvisna od tega, kako namerava organizacija uporabiti podatke. V nadaljevanju predstavljamo različne tipe podatkovnih baz (Oracle, 2024):

1. Relacijske baze podatkov: Predmeti v relacijski podatkovni bazi so organizirani v niz tabel s stolpci in vrsticami. Relacijske baze zagotavljajo najučinkovitejši način dostopa do strukturiranih informacij.
2. Objektno usmerjene podatkovne baze: Informacije v objektno usmerjeni podatkovni bazi so predstavljene v obliki objektov.
3. Porazdeljene podatkovne baze: Porazdeljena podatkovna baza sestoji iz dveh ali več datotek, ki se nahajajo na različnih mestih. Podatkovna baza se lahko shranjuje na več računalnikih, ki se nahajajo na isti fizični lokaciji, ali je razpršena po različnih omrežjih.
4. Podatkovna skladišča: Kot osrednje skladišče podatkov je podatkovno skladišče vrsta podatkovne baze, ki je posebej zasnovana za hitre poizvedbe in analizo.
5. Podatkovne baze NoSQL: Podatkovna baza NoSQL (nerelacijska podatkovna baza) omogoča shranjevanje in manipuliranje nestrukturiranih in polstrukturiranih podatkov. Podatkovne baze NoSQL pridobivajo na priljubljenosti.
6. Grafične podatkovne baze: Grafična podatkovna baza hrani podatke v smislu entitet in povezav med entitetami.

7. OLTP podatkovne baze: OLTP podatkovna baza je hitra, analitična podatkovna baza, zasnovana za veliko število transakcij, izvedenih od več uporabnikov.

2.2.2 Sistem za upravljanje podatkovnih baz (DBMS)

Sistema za upravljanje podatkovnih baz (DBMS - Data Base Management System) ima naslednje funkcije (Priyali, 2024):

1. Organiziranje podatkov: Podatki so organizirani v skladu s specifikacijami jezika za definicijo podatkov.
2. Integracija podatkov: Podatki so povezani ali medsebojno povezani na ravni elementa, imenovanega podatkovno polje, zaradi česar se lahko med izvajanjem določenega aplikacijskega programa sestavljajo v različnih kombinacijah.
3. Ločevanje podatkov: Sistem za upravljanje podatkovnih baz deluje kot filter med aplikacijskimi programi in njihovimi povezanimi podatki.
4. Nadzor podatkov: Sistem za upravljanje podatkovnih baz nadzira, kako in kje so podatki fizično shranjeni. Pri pridobivanju podatkov locira in vrne zahtevane elemente podatkov programom. Poleg tega ločuje logičen opis in odnos podatkov od načina, kako so podatki fizično shranjeni. Podatkovna baza ostane varna in nedotaknjena, čeprav jo različni programi, ki opisujejo podatke na različne načine in ki so lahko napisani v različnih programskih jezikih, obdelujejo.
5. Pridobivanje podatkov: Do shranjenih podatkov lahko dostopamo preko sistema za upravljanje podatkovnih baz (DBMS).
6. Varovanje podatkov: Sistem za upravljanje podatkovnih baz varuje vsebino podatkovne baze kot tudi razmerja med podatkovnimi elementi. Omogoča zaščito pred dostopom nepooblaščenih uporabnikov, fizično škodo, odpovedjo operacijskega sistema, sočasnim posodabljanjem ter določenimi prekinitvami, ki jih sproži gostiteljski program.

Povzamemo lahko, da je sistem za upravljanje podatkovnih baz nepogrešljiv sestavni del sodobne računalniške infrastrukture, ki omogoča organizacijam učinkovito shranjevanje, upravljanje in izkoriščanje podatkov za širok spekter aplikacij.

2.2.3 Oracle podatkovna baza

2.2.3.1 Značilnosti Oracle podatkovne baze

Oracle Database je eden najbolj priljubljenih in pogosto uporabljenih relacijskih sistemov za upravljanje podatkovnih baz na svetu. Vodilna podjetja iz različnih sektorjev zaupajo temu sistemu za učinkovito shranjevanje, organiziranje in upravljanje svojih podatkov. Razvila ga je družba Oracle Corporation leta 1979 in ta sistem je postal njihov paradni izdelek, kar je podjetju pomagalo, da se je uveljavilo kot eden najmočnejših razvijalcev poslovnih programov za različne operacijske sisteme. Poleg Oracle Database ima isto podjetje v lasti tudi MySQL, še en priljubljen relacijski sistem za upravljanje podatkovnih baz, ki je brezplačen in odprtokoden, ter programski jezik Java. Oracle je priljubljen zaradi svoje zmožnosti obdelave velikih količin podatkov. Ponuja podporo za različne administrativne operacije in transakcije ter se ponaša z impresivnimi zmogljivostmi SQL poizvedb. Te lastnosti so prispevale k priljubljenosti Oracle Database zlasti med podjetji. Kljub temu da so se pojavili in uspeli tudi drugi konkurenti, je Oracle uspel ohraniti svoj položaj vodilnega relacijskega sistema za upravljanje podatkovnih baz več kot 30 let in še vedno prevladuje na trgu (Devart, 2024).

V nadaljevanju navajamo glavne značilnosti Oracle podatkovne baze (SolarWinds, 2024):

1. Prenosljivost: Oracle Database je visoko prenosljiva programska oprema, ki deluje na različnih operacijskih sistemih in podpira številne aplikacije, vključno z obdelavo transakcij, analitiko in poslovno inteligenco. Zahvaljujoč svojemu naprednemu omrežnemu skladu omogoča enostavno povezovanje aplikacij z različnih platform (Linux, Windows).
2. Razširljivost in zmogljivost: Oracle Database s svojimi lastnostmi, kot so predpomnilnik podatkov v pomnilniku (IMDB), gruče pravih aplikacij (RAC) in napredno stiskanje podatkov, zagotavlja visoko zmogljivost in nizko zakasnitev v realnem času.
3. Visoka razpoložljivost: Oracle Database vključuje zmogljive funkcije in orodja, ki IT ekipam pomagajo pri maksimiranju razpoložljivosti njihovih ključnih podatkovnih baz. Med temi orodji so Oracle RAC, Data Guard, Oracle Multitenant in Oracle Sharding. Podjetja lahko ta orodja uporabljajo v različnih kombinacijah za optimizacijo razpoložljivosti podatkov v visoko zmogljivih računalniških okoljih.
4. Varnost: Z uporabo orodij in tehnologij za varnost podatkov Oracle, kot so Database Vault, Label Security in Data Safe, lahko organizacije dokazujejo skladnost in zmanjšujejo tveganje kršitev podatkov. IT osebje lahko omogoči prikrivanje podatkov, spremljanje

dejavnosti in nadzor privilegiranih dostopov, s čimer zaščiti relacijsko podatkovno bazo Oracle pred morebitnimi grožnjami.

5. Varnostno kopiranje in obnovitev: Oracle podatkovna baza ima izjemne sposobnosti varnostnega kopiranja in obnavljanja podatkov, kar IT ekipam omogoča hitro obnovitev ključnih informacij po morebitni napaki.

2.2.3.2 Prednosti in slabosti Oracle podatkovne baze

V nadaljevanju navajamo prednosti uporabe Oracle podatkovne baze (Devart, 2024):

1. Globalna priljubljenost na različnih področjih: Oracle podatkovna baza je priljubljena po vsem svetu in se uporablja na različnih področjih, saj ponuja širok nabor funkcionalnosti, ki ustrezajo potrebam različnih industrij in organizacij. Uporablja se v poslovni analitiki in poročanju, finančnem sektorju, telekomunikacijski industriji, e-trgovini, zdravstvu in mnogih drugih panogah.
2. Združljivost z različnimi platformami in aplikacijami: Oracle podatkovna baza je znana po svoji visoki stopnji združljivosti z različnimi platformami in aplikacijami, kar je ena ključnih značilnosti, ki prispevajo k njeni priljubljenosti in razširjenosti. Na voljo je za večino vodilnih operacijskih sistemov, vključno z Oracle Linuxom, Microsoft Windowsom, UNIX-om in Linuxom. Združljiva je s široko paleto programskih jezikov in razvojnih okolij, kot so Java, Python, C#, PHP, Ruby in mnogi drugi. Poleg tega deluje z večino vodilnih aplikacijskih strežnikov, vključno z Oracle WebLogic Serverjem, Apache Tomcatom, IBM WebSphere in drugimi. Oracle podatkovna baza podpira različne podatkovne formate, vključno z relacijskimi tabelami, XML, JSON, geografskimi podatki, slikami, zvoki in videi.
3. Razpoložljivost različnih izdaj baz podatkov z različnimi funkcionalnostmi za različne potrebe: Najpomembnejše izdaje Oracle podatkovnih baz, ki jih je smotrno omeniti, so Oracle Database Standard Edition (SE), Oracle Database Enterprise Edition (EE), Oracle Database Express Edition (XE) in Oracle Database Cloud Service.
4. Avtomatizacija storitev prek oblaka Oracle: Oracle Cloud ponuja številne možnosti za avtomatizacijo storitev, kar organizacijam omogoča optimizacijo procesov, povečanje učinkovitosti in zmanjšanje časa ter stroškov ročnega dela.

5. Obilje varnostnih funkcij: Oracle podatkovna baza ponuja obsežen nabor varnostnih funkcij, ki organizacijam omogočajo zaščito njihovih podatkov pred nepooblaščenim dostopom, manipulacijo in izgubo. Med najpomembnejšimi varnostnimi funkcijami so nadzor dostopa, varnostna avtentifikacija, šifriranje podatkov, upravljanje dostopa do aplikacij, revizijski dnevniki, varne rešitve za izvajanje in nadzorovanje dostopa do zunanjih virov.
6. Obsežna skupnost razvijalcev: Skupnost razvijalcev je ključen element ekosistema vsake tehnološke platforme ali produkta, vključno z Oracle podatkovno bazo. Oracle se ponaša z veliko in aktivno skupnostjo razvijalcev, ki prispeva k razvoju, inovacijam ter izmenjavi znanja v zvezi z Oracle tehnologijami. Ključni vidiki te skupnosti vključujejo forume, konference, programe usposabljanja in certificiranja ter prispevke k odprtokodnemu razvoju.
7. Strokovna podpora: Oracle zagotavlja obsežno strokovno podporo za svojo podatkovno bazo, kar strankam omogoča maksimalno izkoriščanje svoje naložbe v Oracle tehnologije. To vključuje celodnevno tehnično podporo, hitre odzive na zahteve strank, nenehno posodabljanje programske opreme, optimizacijo zmogljivosti Oracle podatkovne baze ter usposabljanje in izobraževanje uporabnikov.

V nadaljevanju navajamo slabosti uporabe Oracle podatkovne baze (Devart, 2024):

1. Visoki stroški plačljivih licenc: Ena od pogostih slabosti uporabe Oracle podatkovne baze so visoki stroški plačljivih licenc. Oracle je znan po tem, da zaračunava visoke cene za svoje licenčne modele, kar lahko predstavlja finančno breme za organizacije, še posebej za manjša podjetja ali projekte z omejenim proračunom. Posledično to lahko vodi v višje stroške infrastrukture in obratovanja ter otežuje pridobivanje novih strank ali projektov. Poleg tega lahko stroški licenciranja vplivajo tudi na donosnost naložbe in celotno ekonomsko uspešnost projekta.
2. Oracle podatkovna baza zahteva zmogljivo strojno opremo, ki lahko podpira zahtevne obremenitve ter zagotavlja visoko razpoložljivost, zmogljivost in zanesljivost. To lahko vključuje nakup dragih strežnikov, shramb podatkov, omrežne opreme in drugih komponent, kar lahko predstavlja velik finančni strošek za organizacije. Poleg tega so potrebni tudi stroški vzdrževanja, nadgradenj, servisiranja in upravljanja te strojne opreme, kar dodatno obremenjuje proračun in vire podjetja.

3. Zahtevnost znanja (poznavanje SQL-a in upravljanja z bazami podatkov): Oracle podatkovna baza je kompleksen sistem, ki zahteva dobro poznavanje SQL (Structured Query Language) ter konceptov upravljanja z bazami podatkov, kot so ustvarjanje tabel, upravljanje indeksov, upravljanje transakcij, varnostne politike in podobno.

2.2.4 PostgreSQL podatkovna baza

2.2.4.1 Značilnosti PostgreSQL podatkovne baze

PostgreSQL je močen, odprtokoden objektno-relacijski sistem za upravljanje z bazami podatkov, ki uporablja in razširja jezik SQL ter vsebuje številne funkcije za varno shranjevanje in obdelavo najbolj zapletenih obremenitev podatkov. Korenine PostgreSQL segajo v leto 1986 kot del projekta Postgres na Univerzi v Kaliforniji v Berkeleyju in že več kot 35 let se aktivno razvija na osnovni platformi. PostgreSQL si je pridobil močan ugled zaradi preizkušene arhitekture, zanesljivosti, integritete podatkov, robustnega nabora funkcij, razširljivosti ter predanosti odprtokodne skupnosti za programsko opremo, ki dosledno zagotavlja zmogljive in inovativne rešitve. Deluje na vseh glavnih operacijskih sistemih in je skladen z ACID-om od leta 2001. PostgreSQL je postal odprtokodna relacijska podatkovna baza izbire za mnoge posameznike in organizacije zaradi svoje zanesljivosti in naprednih funkcij (Postgresql, 2024).

V nadaljevanju navajamo glavne značilnosti PostgreSQL baze (Quest, 2024):

1. PostgreSQL je objektno-relacijski sistem za upravljanje z bazami podatkov (ORDBMS): Podpira tradicionalne relacijske značilnosti, kot so tabele, ključi in SQL poizvedbe, obenem pa tudi objektno usmerjene značilnosti, kot so shranjeni postopki, funkcije in kompleksni tipi podatkov. Omogoča boljše modeliranje podatkov in večjo fleksibilnost pri oblikovanju ter upravljanju kompleksnih baz podatkov.
2. Tuje podatkovne zavijalke: Tuje podatkovne zavijalke v PostgreSQL omogočajo izvajanje SQL poizvedb v drugih jezikih poleg standardnega SQL-a. Najpogostejša tuja podatkovna zavijalka je PL/pgSQL, ki je jezik za shranjevanje postopkov in je podoben PL/SQL-u v Oracle.
3. Skladnost s standardom ACID: PostgreSQL je popolnoma skladen s standardom ACID, kar pomeni, da zagotavlja štiri ključne lastnosti transakcij – atomskost, doslednost, izolacijo in trajnost.

4. Razširljivost: PostgreSQL ponuja visoko razširljivost, ki omogoča prilagodljivo in učinkovito rast podatkovne baze glede na potrebe organizacije.
5. Visoka zmogljivost: PostgreSQL baza je znana po visoki zmogljivosti, še posebej pri obdelavi velikih količin podatkov in kompleksnih poizvedb.
6. Skladnost s standardom SQL: Postgres je skladen s standardom SQL, kar zagotavlja enostavno uporabo za razvijalce in analitike, ki so seznanjeni s SQL.
7. Upravljanje sočasnosti: Postgres ponuja nadzor sočasnosti več različic, kar omogoča hkratni dostop do istih podatkov s strani več transakcij brez konfliktov.
8. Napredno indeksiranje: Postgres podpira različne vrste indeksov, vključno z B-drevesi, Hash, GiST, SP-GiST in GIN.
9. Replikacija in visoka razpoložljivost: Postgres podpira tako sinhrono kot asinhrono replikacijo ter zagotavlja več orodij za dosego visoke razpoložljivosti.
10. Napredna varnost: Postgres ponuja nadzor dostopa na podlagi vlog, šifriranje podatkov in varnost povezav.
11. Podatkovna skladiščenje in poslovna inteligenca: PostgreSQL je dobro prilagojen za naloge podatkovnega skladiščenja in poslovne inteligence, saj podpira napredno analitiko in vizualizacijo podatkov.
12. Skupnost in razvoj: PostgreSQL je projekt z odprto kodo, ki ga vodi velika in aktivna skupnost, ki prispeva k razvoju in vzdrževanju podatkovne baze.

2.2.4.2 Prednosti in slabosti PostgreSQL podatkovne baze

V nadaljevanju navajamo prednosti uporabe PostgreSQL podatkovne baze (Quest, 2024):

1. Stroškovna učinkovitost: PostgreSQL podatkovna baza je odprtokodna programska oprema, kar pomeni, da nima licenčnih stroškov in ima nižje skupne stroške lastništva v primerjavi z lastniškimi bazami podatkov. Ta brezplačna narava odprtokodne licence omogoča organizacijam, da izkoristijo številne prednosti PostgreSQL-a brez potrebe po plačilu licenčnih pristojbin, kar lahko znatno zmanjša skupne stroške lastništva in omogoči večjo prilagodljivost pri proračuniranju in načrtovanju IT infrastrukture.

2. Visoka razširljivost: PostgreSQL podatkovna baza ponuja horizontalno razširljivost z bralnimi replikami in bazenom povezav, kar jo naredi primerno za aplikacije z visokim prometom in veliko sočasnostjo. Ta značilnost omogoča, da se obremenitev enakomerno razporedi med več replik in povezav, kar omogoča boljšo zmogljivost in zanesljivost za aplikacije z intenzivnim prometom in visoko stopnjo sočasnosti.
3. Razširljivost: PostgreSQL podatkovna baza je izjemno prilagodljiva in jo je mogoče enostavno prilagoditi, da ustreza posebnim potrebam. Prav tako podpira široko paleto operacijskih sistemov, strojne opreme in programskih jezikov, kar omogoča organizacijam, da izberejo in prilagodijo okolje, ki najbolje ustreza njihovim zahtevam in infrastrukturnim potrebam.
4. Močna skupnost: PostgreSQL podatkovna baza ima veliko in aktivno skupnost uporabnikov in razvijalcev, kar olajša iskanje podpore in virov.
5. Napredne funkcije: PostgreSQL podatkovna baza ima veliko naprednih funkcij, kot je vgrajena podpora za JSON, polno besedilno iskanje in boljšo zmogljivost za kompleksne poizvedbe.
6. Prilagojenost oblaku: PostgreSQL podatkovna baza ima arhitekturo, ki je bolj prijazna oblaku in je bolj primerna za sodobne oblakom osnovane razporeditve.
7. Licenciranje odprtokodnosti: Odprtokodna licenca PostgreSQL podatkovne baze omogoča večjo prilagodljivost in svobodo glede spreminjanja in distribuiranja programske opreme.
8. Visoka zmogljivost: PostgreSQL podatkovna baza je dobro znana po svoji visoki zmogljivosti, kar ji omogoča obdelavo velikih količin podatkov.

V nadaljevanju navajamo slabosti uporabe PostgreSQL podatkovne baze (Quest, 2024):

1. Zapletenost: PostgreSQL podatkovna baza lahko za nove uporabnike predstavlja kompleksnost pri nastavitvi in uporabi. Z obsežnim naborom funkcij in možnosti je lahko za začetnike izziv, saj lahko obilje možnosti predstavlja preobremenitev. Pri uvajanju novega sistema je ključno zagotoviti učinkovito izobraževanje in podporo, da bi novi uporabniki lahko kar najbolje izkoristili potencial, ki ga ponuja PostgreSQL.
2. Razširljivost: Čeprav se PostgreSQL podatkovna baza lahko dobro razširi, je lahko izziv pri upravljanju v aplikacijah velikega obsega. Skrbniki podatkovnih baz morajo imeti izkušnje z naprednimi nalogami upravljanja z bazami podatkov, kot so replikacija, gručenje in razdeljevanje. Zahtevnejše naloge, kot so nadzor obremenitve, optimizacija zmogljivosti ter

izvajanje varnostnih in vzdrževalnih postopkov, zahtevajo temeljito znanje in izkušnje, ki so ključnega pomena za učinkovito upravljanje PostgreSQL podatkovne baze v okoljih velikega obsega.

3. Zmogljivost: PostgreSQL je lahko počasnejši v primerjavi z nekaterimi drugimi bazami podatkov. Kljub temu ima veliko funkcij in možnosti konfiguracije, ki jih je mogoče prilagoditi za izboljšanje zmogljivosti. S pravilno konfiguracijo in optimizacijo lahko skrbniki podatkovnih baz bistveno izboljšajo delovanje PostgreSQL, kar omogoča boljšo uspešnost in odzivnost pri obravnavi zahtevnih obremenitev.
4. Dokumentacija: Dokumentacija za PostgreSQL podatkovno bazo lahko predstavlja težavo pri navigaciji, še posebej za uporabnike, ki niso seznanjeni s terminologijo podatkovnih baz. Začetniki se lahko znajdejo v težavah pri iskanju in razumevanju informacij, saj lahko kompleksna terminologija predstavlja oviro. Pomembno je, da je dokumentacija jasna, dobro strukturirana in opremljena s primeri, ki olajšajo razumevanje in uporabo PostgreSQL-a tudi manj izkušenim uporabnikom.
5. Pomanjkanje vgrajenih orodij: V primerjavi z nekaterimi drugimi bazami podatkov PostgreSQL podatkovna baza nima vgrajenih orodij za varnostne kopije, spremljanje in upravljanje. Kljub temu pa obstaja veliko orodij tretjih strank, ki lahko zagotavljajo te funkcije. To pomeni, da skrbniki podatkovnih baz lahko izbirajo med številnimi zunanjimi rešitvami, ki ustrezajo njihovim potrebam glede varnostnih kopij, spremljanja in upravljanja, s čimer lahko dosežejo visoko raven prilagodljivosti in učinkovitosti v upravljanju PostgreSQL podatkovne baze.
6. Migracija: Migracija iz drugih baz podatkov v PostgreSQL podatkovni bazi lahko predstavlja izziv, še posebej, če gre za velike količine podatkov.
7. Stroški: Čeprav je PostgreSQL podatkovna baza odprtokodna in brezplačna za uporabo, nekatera orodja tretjih strank in storitve podpore prinašajo stroške.

2.2.5 Primerjava podatkovnih baz

Glavna razlika med sistemoma za upravljanje z bazami podatkov PostgreSQL in Oracle je v njihovem licenčnem modelu. PostgreSQL je odprtokodna baza podatkov, ki je na voljo brezplačno in jo lahko prosto uporabljajo vsi, medtem ko je Oracle zaprta baza podatkov, za katero je potrebno plačati licenčnino za uporabo. PostgreSQL je razvit in vzdrževan s strani

skupnosti prostovoljnih razvijalcev po vsem svetu, medtem ko je Oracle izdelek komercialnega podjetja Oracle. Obe bazi uporabljata osnovne koncepte relacijskih baz podatkov, kot so sheme, tabelni prostori in indeksi. Kljub temu se razlikujeta v funkcionalnostih, ki jih ponujata. Na primer, PostgreSQL ponuja napredne možnosti replikacije in podpore, medtem ko Oracle ponuja širok nabor orodij in funkcij, ki so prilagojene potrebam večjih podjetij in organizacij (IPSpecialistoff, 2022).

V nadaljevanju navajamo nekatere poglavitne značilnosti podatkovnih baz in kako se razlikujeta med sabo Oracle in PostgreSQL bazi glede na te značilnosti (IPSpecialistoff, 2022):

1. Funkcionalnost:

- PostgreSQL baza: a) visoka razpoložljivost, b) skladna z ACID-om.
- Oracle baza: a) visoka razpoložljivost; b) več transakcij na sekundo; c) bolj funkcionalna kot PostgreSQL baza, vendar dodatne funkcije tudi stanejo občutno več; č) skladna z ACID-om.

2. Razširljivost:

- PostgreSQL baza: a) zaradi svoje narave odprtokodnosti je bolj razširljiva; b) podatkovne baze lahko obvladujejo katerokoli količino podatkov; c) brezplačno širjenje je mogoče s skladiščenjem na osnovi gruč; č) ohranjanje celovitosti med operacijami razširljivosti z uporabo datotek WAL, omejenih na 16 MB.
- Oracle baza: a) operacije razširljivosti zahtevajo več naložb v infrastrukturo, saj standardna izdaja omogoča samo štiri vtičnice, medtem ko podjetniška izdaja zagotavlja več; b) dnevniki ponovnega izvajanja pomagajo zagotoviti celovitost podatkov.

3. Varnost:

- PostgreSQL baza: a) omogoča vloge in dedne vloge, s katerimi lahko razvijalci nastavijo dovoljenja; b) podpira domači SSL, kar pomaga pri šifriranju komunikacije strežnika; c) zagotavlja dodatne nadzorne mehanizme dostopa prek SE-PostgreSQL, ki temeljijo na varnostni politiki SELinux.
- Oracle baza: a) varnostne funkcije, ki so bolj robustne kot pri PostgreSQL-u; b) za dostop do naprednih varnostnih funkcij so potrebne dražje izdaje; c) odporna v primeru varnostnih revizij, zaščite podatkov, revizij in spremljanja; č) zagotavlja odlične rešitve za izolacijo med vtičnimi podatkovnimi bazami in neodvisnim upravljanjem s šifriranjem ključev.

4. Podpora:

- PostgreSQL baza: a) aktivna skupnost, ki zagotavlja brezplačno spletno pomoč prek blogov, elektronskih sporočil, kode in drugih kanalov; b) ni nujne številke za tehnično podporo; c) najem razvijalcev iz skupnosti PostgreSQL za premium podporo je manj stroškovno učinkovit kot najem primerljivega specialista za Oracle; č) na voljo so tudi ponudniki podpore tretjih oseb (EnterpriseDB in 2nd QuadrantL).
- Oracle baza: a) draga pomoč; b) velika podjetja morajo najeti Oracleove svetovalce ali se zanašati na Oracleovo podporo (predstavlja do 25 % licenčnih pristojbin).

5. Združljivost in replikacija:

- PostgreSQL baza: a) streaming replikacija zagotavlja visoko razpoložljivost; b) zaradi replikacije glavni-podrejeni razvijalci koristijo brezhibno zmogljivost med varnostnimi kopijami, dodelitvijo nalog in gručenjem; c) pomoč pri okvirju ORM; č) podpora za knjižnice JDBC, ODBC, OLEDB in .Net; d) podpira širši nabor vmesnikov API kot Oracle, kar ga naredi bolj združljivega z mnogimi aplikacijami, dodatki in SQL okolji.
- Oracle baza: a) DataGuard zagotavlja visoko razpoložljivost; b) replikacija glavnih strežnikov; c) pomoč pri okvirju ORM; d) podpora za knjižnice JDBC, ODBC, OLEDB in .Net; č) podpora za API-je je manj obsežna kot pri PostgreSQL-u.

6. Visoka razpoložljivost:

- PostgreSQL baza: a) PgPool v PostgreSQL Enterprise Edition zagotavlja funkcionalnosti, podobne Oracle Real Application Clusters; b) dinamično dodajanje vozlišč z uporabo možnosti horizontalne razširljivosti; c) PgPool ni funkcija PostgreSQL-a in za dosego podobne funkcionalnosti Real Application Clusters zahteva veliko orodij za gručno strojno opremo.
- Oracle baza: a) Oracle Real Application Clusters omogočajo, da se podatkovne baze delijo med strežniki; b) ko ena podatkovna baza odpove, se lahko ta podatkovna baza izvaja na preostalih podatkovnih bazah, da zagotovi neprekinjeno upravljanje delovnega toka; c) Real Application Cluster je že vnaprej nameščena funkcija.

7. Varnostno kopiranje in obnovitev:

- PostgreSQL baza: Postopek obnovitve podatkov je preprost, saj nadomesti mape, podmape in povezane datoteke WAL.

- Oracle baza: a) postopki obnavljanja podatkov lahko postanejo preveč zapleteni, b) RMAN (upravitelj varnostnega kopiranja in obnovitve, ki je na voljo za baze podatkov Oracle) ponuja preprosto in učinkovito rešitev za varnostno kopiranje podatkovne baze.

8. Skupni stroški lastništva:

- PostgreSQL baza: Zaradi odprtokodne narave PostgreSQL-a ni stroškov, povezanih z njegovim pridobivanjem ali s komplementarno podporo strankam. PostgreSQL je odprtokodna podatkovna baza, zato se lahko uporabljajo vse njegove zmogljivosti brezplačno.
- Oracle baza: Za podatkovno bazo Oracle so stroški pridobitve in podpore za izdelek znatni, poleg tega se plača dodatno za vsako dodatno funkcijo, ki jo potrebujemo, pri čemer ima vsaka visoke stroške. Torej je skupni strošek lastništva podatkovne baze Oracle visok.

9. Obvladovanje velike količine podatkov:

- PostgreSQL baza: Na delovnih postajah z velikimi količinami pomnilnika podatkovna baza PostgreSQL poveča produktivnost za 10 do 30 strani z učinkovitim obvladovanjem ogromnih količin podatkov.
- Oracle baza: Glede na podobne pogoje je podatkovna baza Oracle Enterprise Edition učinkovitejša pri obvladovanju velike količine podatkov kot PostgreSQL.

Zaključimo lahko, da sta PostgreSQL in Oracle podatkovni bazi približno enakovredni glede zmogljivosti, učinkovitosti in združljivosti, vendar imata vsaka svoje prednosti. Oracle vodi pri varnosti, replikaciji in razpoložljivosti, medtem ko PostgreSQL ponuja boljšo združljivost z API-ji, manj drago podporo in bolj robustno razširljivost. Izbira med podatkovnima bazama temelji na prioritetah podjetja ter zahtevah za določene funkcionalnosti. PostgreSQL je dobra izbira, če se potrebuje enostavna podatkovna baza, ki jo je mogoče prilagoditi za operacije in ima nizke skupne stroške lastništva. Po drugi strani pa Oracle ponuja robustno funkcionalnost, zlasti če sta visoka razpoložljivost in brezhibna replikacija med obsežnimi transakcijami ključnega pomena za poslovanje (IPSpecialistoff, 2022).

2.3 Docker

2.3.1 Značilnosti Dockerja

Docker je odprtokodna platforma, ki razvijalcem omogoča izdelavo, uvajanje, izvajanje, posodabljanje in upravljanje kontejnerjev – standardiziranih, izvršljivih komponent, ki združujejo izvorno kodo aplikacije s knjižnicami operacijskega sistema (OS) in z odvisnostmi, potrebnimi za izvajanje te kode v kateremkoli okolju. Omogoča razvijalcem, da aplikacije zapakirajo v lahke, prenosljive kontejnerje, ki se lahko enostavno premikajo med različnimi okolji, ne da bi bilo potrebno vsakič ponovno konfigurirati okolje (IBM, 2024).

Docker je platforma, ki omogoča pakiranje, dostavo in izvajanje kontejnerjev. Tehnologija kontejnerjev deluje prek operacijskega sistema, pri čemer vsak kontejner pakira aplikacijsko storitev ali funkcijo skupaj z vsemi knjižnicami, s konfiguracijskimi datotekami, z odvisnostmi in drugimi potrebnimi deli in s parametri za njihovo delovanje. Vsak kontejner deli storitve enega osnovnega operacijskega sistema (OS). Docker slike vsebujejo vse odvisnosti, potrebne za izvajanje kode znotraj kontejnerja, zato kontejnerji, ki se premikajo med Docker okolji z istim OS, delujejo brez sprememb. Docker izkorišča izolacijo virov v jedru operacijskega sistema za poganjanje več kontejnerjev na istem OS. To je drugače kot pri virtualnih strojih (VM), kjer celoten OS sestavlja izvršljiva koda na vrhu abstraktne plasti fizičnih strojnih virov (Hashemi-Pour, Bigelow in Courtemanche, 2023).

Docker paketira aplikacijo in vse njene odvisnosti v virtualni kontejner, ki je zmožen delovanja na kateremkoli strežniku Linux. Zato jih imenujemo kontejnerji. Vsebujejo vse potrebne odvisnosti v enem samem kosu programske opreme. Lahko si predstavljamo kontejnerje kot tehnologijo, ki omogoča izolacijo določenih procesov jedra in jih prepriča, da mislijo, da tečejo na popolnoma novem računalniku. Za razliko od virtualnih strojev, ki imajo vsak svoj kompleten operacijski sistem, kontejnerji delijo jedro operacijskega sistema, medtem ko imajo naložene le svoje različne binarne datoteke in knjižnice (Santos, 2020).

Docker ponuja (IBM, 2024):

1. Izboljšano in brezhibno prenosljivost kontejnerjev: Docker kontejnerji tečejo brez sprememb v kateremkoli namiznem, podatkovnem centru ali oblaku.

2. Avtomatizirano ustvarjanje kontejnerjev: Docker lahko samodejno zgradi kontejner na podlagi izvorne kode aplikacije.
3. Različice kontejnerjev: Docker lahko sledi različicam kontejnerske slike, se povrne na prejšnje različice in sledi, kdo je zgradil različico, in kako.
4. Ponovna uporaba kontejnerjev: Obstoječi kontejnerji se lahko uporabljajo kot osnovne slike – praktično kot predloge za gradnjo novih kontejnerjev.
5. Razvijalci lahko poganjajo Docker kontejnerje na kateremkoli operacijskem sistemu.

Razvijalci se pogosto zatekajo k uporabi Dockerja za namestitev in uporabo podatkovnih baz znotraj kontejnerjev, saj omogoča izboljšano varnost z izolacijo podatkovne baze od gostiteljskega sistema ter zagotavlja natančen nadzor dostopa. Uporaba Dockerja za namestitev podatkovnih baz poenostavlja postopek in zagotavlja doslednost. Poleg tega izboljšuje prenosljivost in ponuja robusten okvir za upravljanje podatkovnih baz v sodobnih razvojnih in produkcijskih okoljih. S pomočjo Dockerja lahko razvijalci enostavno ustvarijo, reproducirajo in delijo okolja za podatkovne baze, kar prispeva k večji učinkovitosti in zanesljivosti razvojnega procesa (DatabaseStar, 2023).

Razlogi za uporabo Dockerja za namestitev in uporabo podatkovne baze so (DatabaseStar, 2023):

1. Hitra namestitev: S pomočjo Dockerja je možno zelo hitro postaviti novo podatkovno bazo v primerjavi z namestitvijo podatkovne baze na operacijskem sistemu.
2. Različne različice: Enostavneje je imeti različne različice iste podatkovne baze, ki tečejo v različnih kontejnerjih, če se morajo preizkusiti različne funkcije.
3. Deluje na več operacijskih sistemih: Nekatere podatkovne baze (kot so SQL Server in Oracle) niso na voljo za Mac. V kolikor želimo inštalirati te podatkovne baze na Mac, je uporaba Dockerja odličen način za namestitev SQL Serverja ali Oracle podatkovne baze.

Slaba stran uporabe Dockeja za podatkovne baze je, da se podatki ne ohranjajo. Za razliko od virtualnega stroja se kontejnerji lahko ustavijo in zaženejo brez shranjevanja podatkov (DatabaseStar, 2023).

2.3.2 Prednosti in slabosti Dockerja

Poleg učinkovitega razvoja aplikacij v kontejnerjih Docker prinaša še naslednje prednosti (Lutkevich, Hughes in Gillis 2023):

1. visoka stopnja prenosljivosti, saj lahko uporabniki registrirajo in delijo kontejnerje na različnih gostiteljih;
2. nižja poraba virov;
3. hitrejše uvajanje v primerjavi z virtualnimi stroji (VM).

Pri uporabi Dockerja pa se lahko pojavijo izzivi, ki predstavljajo njegove slabosti (Lutkevich, Hughes in Gillis 2023):

1. Število kontejnerjev v podjetju je lahko težko učinkovito upravljati.
2. Uporaba kontejnerjev se razvija od granularnega virtualnega gostovanja k orkestraciji aplikacijskih komponent in virov. Posledično postaja distribucija in medsebojna povezanost komponentiziranih aplikacij, ki lahko vključujejo na stotine začasnih kontejnerjev, velik izziv.

2.4 Big Data

2.4.1 Značilnosti Big Data

Masovni podatki nenehno izboljšujejo procese in politike ter omogočajo ustvarjanje izdelkov, storitev in izkušenj, ki so popolnoma prilagojeni potrebam strank. Zaradi raznolikosti in kompleksnosti masovnih podatkov se običajno presegajo zmogljivosti tradicionalnih podatkovnih baz za zajemanje, upravljanje in obdelavo. Zato so sodobni pristopi, kot je uporaba oblačnih tehnologij in naprednih analitičnih orodij, postali ključni za učinkovito obvladovanje in izkoriščanje teh obsežnih podatkovnih tokov. Masovni podatki lahko prihajajo od kjerkoli ali iz česarkoli na svetu, kar lahko spremljamo digitalno. Vremenski sateliti, prometne kamere, trendi na socialnih medijih in druge podatkovne vire je mogoče zajemati in analizirati za povečanje odpornosti in konkurenčnosti podjetij (SAP, 2024). Znotraj masovnih podatkov poznamo tudi koncept neskončnega toka podatkov. Ta koncept se osredotoča na neprekinjen tok podatkov, ki neprestano prihajajo v sistem in se obdelujejo v realnem času. Podatki v tem

neskončnem toku se lahko nanašajo na neprekinjene tokove podatkov iz senzorjev, strežnikov aplikacij, logov, transakcij in drugih virov. Takšen neskončni tok podatkov zahteva posebno obravnavo, saj se podatki obdelujejo, še preden so shranjeni v celoti. Sistemi, ki obdelujejo takšne podatke, morajo biti sposobni hitre analize in odločanja v realnem času. Ta dinamičen pristop omogoča organizacijam, da izkoristijo podatke takoj, ko ti prispejo, kar omogoča hitrejše in bolj prilagodljive odločitve v realnem času (Wickramasinghe, 2023).

Podatki so lahko najdragocenejše premoženje podjetja. Uporaba masovnih podatkov za odkrivanje vpogledov lahko pomaga razumeti področja, ki vplivajo na poslovanje, od tržnih razmer in nakupnih navad strank do poslovnih procesov. Tukaj je nekaj primerov masovnih podatkov, ki pomagajo preoblikovati organizacije v različnih industrijah (GoogleCloud, 2024):

1. Sledenje vedenju potrošnikov in nakupnim navadam za zagotavljanje hiperpersonaliziranih priporočil za maloprodajne izdelke, prilagojenih posameznim strankam.
2. Spremljanje plačilnih vzorcev in njihova analiza glede na zgodovinsko dejavnost strank za zaznavanje prevar v realnem času.
3. Združevanje podatkov in informacij iz vsake faze poti pošiljke z lokalnimi prometnimi vpogledi za pomoč operaterjem flote pri optimizaciji dostave.
4. Uporaba tehnologij, ki jih poganja umetna inteligenca, kot je obdelava naravnega jezika, za analizo nestrukturiranih medicinskih podatkov (kot so raziskovalna poročila, klinični zapisi in laboratorijski rezultati) za pridobivanje novih vpogledov za izboljšanje razvoja zdravljenja in izboljšanje oskrbe pacientov.
5. Uporaba slikovnih podatkov iz kamer in senzorjev ter GPS podatkov za zaznavanje lukenj in izboljšanje vzdrževanja cest v mestih.
6. Analiziranje javnih zbirk podatkov satelitskih slik in geoprostorskih zbirk podatkov za vizualizacijo, spremljanje, merjenje in napovedovanje socialnih in okoljskih vplivov delovanja dobavne verige.

To je le nekaj načinov, kako organizacije uporabljajo masovne podatke, da postanejo bolj podatkovno usmerjene, s čimer se lahko bolje prilagodijo potrebam in pričakovanjem svojih strank ter sveta okoli njih.

Definicije masovnih podatkov se lahko nekoliko razlikujejo, vendar bodo vedno opisane v smislu obsega, hitrosti in raznolikosti. Te značilnosti masovnih podatkov so pogosto imenovane "3 V-ji masovnih podatkov" (SAS, 2024):

1. Obseg: Organizacije zbirajo podatke iz različnih virov, vključno s transakcijami, pametnimi (IoT) napravami, z industrijsko opremo, videi, s slikami, z zvokom, s socialnimi mediji in z drugimi. V preteklosti bi bilo shranjevanje vseh teh podatkov predrago, vendar so cenejše rešitve za shranjevanje z uporabo podatkovnih jezer in oblaka olajšale to breme.
2. Hitrost: Z rastjo interneta podatki pritekajo v podjetja z neprimerljivo hitrostjo in jih je treba obdelati pravočasno. Senzorji in pametni števci povečujejo potrebo po obvladovanju teh tokov podatkov v realnem času.
3. Raznolikost: Podatki prihajajo v vseh vrstah formatov. Od strukturiranih številčnih podatkov v tradicionalnih podatkovnih bazah, do nestrukturiranih besedilnih dokumentov, e-poštnih sporočil, videov, avdio posnetkov, podatkov o borznih tečajih in finančnih transakcijah.

Tipi masovnih podatkov (SAP, 2024):

1. Strukturirani podatki: Ta vrsta podatkov je najlažja za organiziranje in iskanje. Excelova preglednica, s svojo postavitvijo predhodno določenih stolpcev in vrstic, je dober način, kako si predstavljati strukturirane podatke. Njihovi sestavni deli so enostavno kategorizirani, kar omogoča oblikovalcem in skrbnikom podatkovnih baz, da določijo preproste algoritme za iskanje in analizo. Strukturirani podatki so običajno shranjeni v relacijskih bazah podatkov, kjer so organizirani v tabelah z jasno določenimi stolpci in vrsticami. Tradicionalno so podatkovne baze uporabljale programski jezik, imenovan Strukturirani poizvedovalni jezik (SQL), da bi upravljale strukturirane podatke. Ta organizacija omogoča učinkovito uporabo SQL poizvedb za iskanje, filtriranje, urejanje in analizo podatkov.
2. Nestrukturirani podatki: Ta kategorija podatkov lahko vključuje stvari, kot so objave na družbenih omrežjih, avdio datoteke, slike in odprti komentarji strank. Namesto preglednic ali relacijskih podatkovnih baz se nestrukturirani podatki običajno shranjujejo v podatkovnih jezerih, skladiščih podatkov in podatkovnih bazah NoSQL. Ta vrsta podatkov se ne more enostavno zajeti v standardnih relacijskih podatkovnih bazah v obliki vrstic in stolpcev. Tradicionalno so podjetja, ki so želela iskati, upravljati ali analizirati velike količine nestrukturiranih podatkov, morala uporabljati zamudne ročne postopke. Strukturiranje in analiza teh podatkov zahtevata napredne tehnike, kot so naravno jezikovno procesiranje (NLP) in strojno učenje.

3. Polstrukturirani podatki: Kot že ime pove, so polstrukturirani podatki hibrid med strukturiranimi in nestrukturiranimi podatki. E-pošta je dober primer, saj vsebuje nestrukturirane podatke v besedilu sporočila, pa tudi organizacijske lastnosti, kot so pošiljatelj, prejemnik, zadeva in datum.

Masovni podatki prinašajo nove vpoglede, ki odpirajo nove priložnosti in poslovne modele.

Učinkovita uporaba masovnih podatkov zahteva tri glavne ukrepe (Oracle, 2024):

1. Integracija: Za analizo velikih podatkovnih nizov v obsegu terabajtov ali celo petabajtov so potrebne nove strategije in tehnologije. Masovni podatki združujejo podatke iz številnih različnih virov in aplikacij, vendar tradicionalni mehanizmi za integracijo podatkov, kot je pridobivanje, transformacija in nalaganje, običajno niso kos tej nalogi. Med integracijo se morajo uvoziti podatki, moramo jih obdelati in zagotoviti, da so oblikovani in na voljo v obliki, s katero lahko začnejo delati poslovni analitiki.
2. Upravljanje: Masovni podatki zahtevajo učinkovite rešitve za shranjevanje. Rešitev za shranjevanje lahko zajema hrambo v oblaku, na lokaciji ali kombinacijo obeh. Mnogi izberejo rešitev za shranjevanje glede na trenutno lokacijo svojih podatkov. Oblak postopoma pridobiva priljubljenost, saj podpira trenutne računalniške zahteve in omogoča, da se po potrebi povečujejo viri.
3. Analiza: Naložba v masovne podatke se obrestuje, ko jih analiziramo in na podlagi pridobljenih podatkov ukrepamo. Vizualna analiza različnih podatkovnih nizov omogoča pridobitev nove jasnosti ter nadaljnje raziskovanje podatkov in odkrivanje novih spoznanj. Ugotovitve lahko delimo z drugimi, medtem ko se podatkovni modeli gradijo z uporabo strojnega učenja in umetne inteligence. Na ta način pridobljeni podatki se nato uporabijo za različne namene.

2.4.2 Prednosti in slabosti uporabe Big Data

V nadaljevanju predstavljamo prednosti in izzive uporabe masovnih podatkov.

Prednosti uporabe masovnih podatkov (GoogleCloud, 2024):

1. Izboljšano odločanje: Masovni podatki so ključni element za organizacije, ki želijo svoje delovanje usmerjati na podlagi podatkov. Ko podjetja uspešno upravljajo in analizirajo

svoje masovne podatke, lahko odkrijejo vzorce in pridobijo vpoglede, ki izboljšujejo ter vodijo k boljšim operativnim in strateškim odločitvam.

2. Povečana agilnost in inovativnost: Masovni podatki omogočajo zbiranje in obdelavo podatkov v realnem času ter analizo, kar podjetjem omogoča hitro prilagajanje in pridobivanje konkurenčne prednosti. Ti vpogledi lahko usmerjajo in pospešijo načrtovanje, proizvodnjo ter uvedbo novih izdelkov, funkcij in posodobitev. Podatki v realnem času omogočajo organizacijam, da se hitro odzovejo na spremembe na trgu in potrebe strank, kar povečuje njihovo sposobnost prilagajanja in agilnosti.
3. Boljše izkušnje strank: Kombinacija in analiza strukturiranih virov podatkov skupaj z nestrukturiranimi omogoča boljše vpoglede za razumevanje potrošnikov, personalizacijo in načine optimizacije izkušenj, kar omogoča organizacijam, da bolje zadovoljijo potrebe in pričakovanja potrošnikov. Z združevanjem teh podatkov lahko podjetja bolje razumejo potrebe svojih strank, prilagodijo svoje izdelke in storitve ter izboljšajo uporabniško izkušnjo.
4. Neprekinjena inteligenca: Masovni podatki omogočajo integracijo avtomatiziranega neprekinjenega pretoka podatkov v realnem času z napredno analitiko podatkov, kar omogoča neprestano zbiranje podatkov, odkrivanje novih vpogledov in identifikacijo novih priložnosti za rast ter ustvarjanje vrednosti. Ta kombinacija tehnologij omogoča organizacijam, da dinamično sledijo spremembam na trgu in hitro reagirajo na nove priložnosti.
5. Bolj učinkovito delovanje: Z uporabo orodij in zmogljivosti analitike masovnih podatkov se lahko podatki obdelajo hitreje in generirajo se vpogledi, ki pomagajo določiti področja, kjer se lahko zmanjšujejo stroški, prihrani čas ter poveča celotna učinkovitost.
6. Izboljšano upravljanje tveganj: S preučevanjem velikih količin podatkov lahko podjetja lažje spremljajo in prepoznavajo vse možne grožnje, kar omogoča boljše ocenjevanje tveganj. To pripomore k poročanju o pomembnih vpogledih, ki prispevajo k razvoju bolj učinkovitih strategij za nadzor in zmanjševanje tveganj. Z naprednimi analitičnimi orodji se lahko dodatno izboljša sposobnost podjetja za hitro odzivanje na nove izzive.

Čeprav imajo masovni podatki številne prednosti, predstavljajo organizacijam tudi nekaj izzivov, s katerimi se morajo spoprijeti pri zbiranju, upravljanju in ukrepanju na tako velikem obsegu podatkov. Najpogosteje poročani izzivi masovnih podatkov vključujejo (GoogleCloud, 2024):

1. Pomanjkanje talentov in veščin na področju podatkov: Znanje in veščine podatkovnih znanstvenikov, analitikov podatkov ter inženirjev podatkov so redki in so med najbolj iskanimi (in najbolje plačanimi) strokovnjaki v IT industriji. Pomanjkanje veščin na področju masovnih podatkov in izkušenj z naprednimi orodji za obdelavo podatkov je eden od glavnih izzivov pri uresničevanju vrednosti iz okolij masovnih podatkov. Zato organizacije intenzivno iščejo strokovnjake s specializiranimi znanji, ki so ključni za uspešno izkoriščanje potenciala masovnih podatkov.
2. Hitrost rasti podatkov: Masovni podatki so po naravi vedno hitro spreminjajoči se in naraščajoči. Brez trdnih infrastrukturnih temeljev, ki lahko obvladujejo obdelavo, shranjevanje, omrežne zahteve in varnostne potrebe, lahko postane upravljanje zelo težavno. Vzpostavitev ustrezne infrastrukture je ključnega pomena za zagotavljanje stabilnosti, skalabilnosti in zanesljivosti v okoljih masovnih podatkov.
3. Težave s kakovostjo podatkov: Kakovost podatkov neposredno vpliva na kakovost odločanja, analitike podatkov in načrtovalskih strategij. To lahko upočasni poročanje, toda če se ne obravnava, lahko privede do zavajajočih rezultatov in neuporabnih vpogledov. Masovni podatki ne zagotavljajo rezultatov, razen če so podatki točni, relevantni in ustrezno organizirani za analizo. Zagotavljanje kakovosti podatkov je ključnega pomena za zagotavljanje zanesljivih rezultatov in učinkovite analize v okoljih masovnih podatkov.
4. Kršitve skladnosti: Masovni podatki vsebujejo veliko občutljivih podatkov in informacij, zaradi česar je težavno zagotavljati, da obdelava in shranjevanje podatkov ustrezata zahtevam za varstvo podatkov in regulativam, kot so zakoni o lokalizaciji podatkov in zahtevah glede bivanja podatkov. Zagotavljanje skladnosti s predpisi postaja vse pomembnejše, saj nenehno spreminjajoče se regulative na področju varstva podatkov postavljajo pred organizacije nove izzive glede shranjevanja, obdelave in upravljanja velikih podatkovnih zbirk.
5. Kompleksnost integracije: Integracija različnih virov podatkov in omogočanje dostopa do podatkov za poslovne uporabnike je kompleksno, vendar ključno, če se želi doseči kakršnakoli vrednost iz masovnih podatkov. Večina podjetij deluje s podatkovnimi silosi, razdeljenimi po različnih sistemih in aplikacijah po organizaciji.
6. Varnostni pomisleki: Masovni podatki vsebujejo dragocene poslovne informacije, zaradi česar so skladišča masovnih podatkov visoko vredni cilji za napadalce. Ker so ti nabori

podatkov raznoliki in kompleksni, je težje uveljaviti celovite strategije in politike za njihovo zaščito.

2.4.3 Podatkovne baze in Big Data

Masovne podatkovne baze so nerelacijske baze podatkov, ki podatke shranjujejo v formatu, ki niso relacijske tabele, kar pogosto imenujemo NoSQL podatkovne baze. Za razliko od podatkovnega jezera, ki je plast za shranjevanje podatkov katerekoli vrste, lahko velika podatkovna baza vnese strukturo v te podatke in jih naredi poizvedljive, saj je optimizirana za analitiko. Zasnovane so posebej za zbiranje in obdelavo različnih vrst masovnih podatkov, vključno s strukturiranimi, polstrukturiranimi in nestrukturiranimi podatki (Le, 2024).

Prednosti uporabe podatkovnih baz za masovne podatke (Le, 2024):

- Lahko shranjujejo in obdelujejo zapletene podatkovne nize – tehnologije masovnih podatkov lahko upravljajo kombinacijo strukturiranih, polstrukturiranih in nestrukturiranih podatkov.
- Enostavno skaliranje – podatkovne baze masovnih podatkov so bolje opremljene za obvladovanje velikih količin raznolikih podatkov kot relacijske baze podatkov. To je zato, ker je shranjevanje in obdelava podatkov lahko razporejena na več računalnikov. Več ko je dodanih podatkov, več računalnikov je dodanih za obvladovanje naraščajočega povpraševanja.
- Oblačno in robno računalništvo – podjetja in organizacije, ki uporabljajo podatkovne baze za masovne podatke, lahko prenesejo del ali vso obdelavo podatkov v oblak in na rob. Na ta način lahko podjetja in organizacije gradijo, preizkušajo in uvajajo aplikacije na hibridnem ali večoblačnem modelu.

Slabosti uporabe podatkovnih baz za masovne podatke (Le, 2024):

- Pomanjkanje standardizacije – Večina NoSQL baz podatkov uporablja bodisi lastne sheme bodisi sploh ne uporablja shem. Zato lahko razumevanje prednosti in slabosti vsake NoSQL baze podatkov za podjetja, organizacije in razvijalce zahteva veliko časa, kar vodi v veliko truda, porabljenega za predhodno izbiro in integracijo v obstoječi delovni proces.
- Neenakomerna podpora za ACID transakcije – ACID je sistem lastnosti, ki jih SQL baze podatkov uporabljajo za zagotovitev pravilnega obdelovanja spletnih transakcij.

V nadaljevanju predstavljamo seznam štirih najboljših podatkovnih baz in njihovih značilnosti, ki so optimalne za shranjevanje in obdelavo različnih vrst masovnih podatkov (Bekker in Kurskov, 2024):

- AWS DynamoDB: a) podpora za modele podatkov ključ-vrednost in dokumentov; b) transakcije ACID (atomarnost, doslednost, izolacija, trajnost); c) integracije z AWS S3, AWS EMR, Amazon Redshift; d) obdelava podatkov v realnem času z DynamoDB Streams, kriptiranje masovnih podatkov od konca do konca; e) obnovitev do točke v času in obnovitev ter varnostno kopiranje na zahtevo; f) najprimernejša za operativne delovne obremenitve, IoT, družbene medije, igre, aplikacije za e-trgovino.
- Azure Cosmos DB: a) podpora za večmodelno shemo podatkov; b) odprtokodna API-ja za SQL, MongoDB, Cassandra, Gremlin itd.; c) integracija z Azure Synapse Analytics za analizo v realnem času brez potrebe po ETL na operativnih podatkih; d) podpora za transakcije ACID; e) kriptiranje masovnih podatkov (med prenosom in v mirovanju); f) nadzor dostopa; g) najprimernejša za upravljanje operacij, e-trgovina, igre na srečo, aplikacije IoT.
- Amazon Redshift: a) prilagodljiva platforma za upravljanje velikih podatkovnih poizvedb s SQL, vodilna na Gartnerjevem kvadrantu čarovnije za rešitve upravljanja podatkov za analitiko, b) samodejno zagotavljanje infrastrukture, c) Amazon Redshift Spectrum za poizvedovanje masovnih podatkov v jezeru podatkov (Amazon S3), d) kriptiranje masovnih podatkov (med prenosom in v mirovanju), e) omrežna izolacija, f) varnost na ravni vrstic in stolpcev, g) najprimernejša za BI in operativno analitiko poslovnih dogodkov v realnem času.
- Amazon DocumentDB: a) združljivost z MongoDB, b) podpora za ACID transakcije, c) podpora za migracijo (npr. MongoDB baze podatkov v lokalnih okoljih v Amazon DocumentDB) z AWS storitvijo za migracijo podatkovnih baz, d) podpora za dostop na podlagi vlog z vgrajenimi vlogami, e) omrežna izolacija, f) spremljanje in popravilo.

3 IZDELAVA APLIKACIJE

3.1 Analiza in specifikacije aplikacije

Analiza aplikacije:

Potrebe in cilji:

1. Učinkovitost in zmogljivost: Cilj je primerjati čas vnosa podatkov iz CSV datoteke v PostgreSQL in Oracle podatkovni bazi ter čas izvajanja poizvedb. S tem želimo ugotoviti, katera baza podatkov je hitrejša pri vnosu podatkov in katera je hitrejša pri izvajanju določenih poizvedb.
2. Skalabilnost: Oceniti, kako obe podatkovni bazi obravnavata večje količine podatkov in kompleksnejše poizvedbe.
3. Enostavnost uporabe in integracije: Razviti aplikacijo, ki enostavno integrira branje podatkov iz CSV datoteke, vstavljanje v podatkovne baze, izvajanje poizvedb in meritev.

Funkcionalnosti:

1. Uvoz podatkov iz CSV datoteke: Aplikacija mora omogočati branje in obdelavo podatkov iz CSV datoteke.
2. Vstavljanje podatkov v podatkovne baze: Podatke je potrebno pravilno razporediti in vstaviti v PostgreSQL in Oracle bazi.
3. Izvajanje SQL poizvedb: Implementacija osmih različnih poizvedb za testiranje zmogljivosti in odzivnosti obeh podatkovnih baz.
4. Merjenje časov izvajanja: Beleženje in shranjevanje časov izvajanja vnosa podatkov v podatkovni bazi in izvajanja poizvedb za nadaljnjo analizo (potrditev ali zavrnitev zastavljenih hipotez).

Tehnologije in Orodja:

1. Spring Boot: Okvir za razvoj aplikacij v Javi, ki omogoča enostavno konfiguracijo in hitro izdelavo aplikacij.
2. Docker: Orodje za ustvarjanje, nameščanje in upravljanje kontejnerjev, v katerih bosta tekli PostgreSQL in Oracle podatkovni bazi.

3. PostgreSQL in Oracle podatkovni bazi: Relacijski podatkovni bazi, ki služita kot platformi za primerjalno analizo.

Specifikacije aplikacije:

Struktura aplikacije:

1. Controller sloj: inicializacija procesov uvoza podatkov, izvajanja poizvedb in vračanje rezultatov.
2. Service sloj: implementacija poslovne logike, vključno z uvozom podatkov iz CSV datoteke, vstavljanjem podatkov v baze in izvedbo poizvedb.
3. Repository sloj: komunikacija s podatkovnimi bazami preko JDBC Template.
4. Model sloj: definicija entitet in relacij za podatkovne baze.

Uporabniški vmesnik:

Aplikacija nima uporabniškega vmesnika. Rezultati se izpisujejo v terminalu.

Poizvedbe:

V aplikaciji se bodo uporabile različne poizvedbe, ki vključujejo naslednje operacije za testiranje zmogljivosti in odzivnosti podatkovnih baz PostgreSQL in Oracle:

1. SELECT: Izbor vseh polj iz tabele, kar omogoča pregled vseh vnosov v tabeli.
2. INSERT: Vstavljanje novih zapisov v tabelo, kar omogoča testiranje hitrosti vstavljanja podatkov v bazo.
3. GROUP BY: Skupinska poizvedba, ki omogoča analizo podatkov po skupinah.
4. WHERE: Filtriranje podatkov, na primer, da se vrnejo samo vozila določene znamke (npr. TESLA), kar omogoča iskanje specifičnih podatkov.
5. HAVING: Poizvedba, ki vrne nek rezultat glede na postavljen pogoj, kar omogoča naprednejše filtriranje skupinskih rezultatov.
6. ORDER BY: Razvrščanje rezultatov po določenih kriterijih, kot so število vozil, država, znamka in model, kar omogoča urejen pregled podatkov.
7. AGGREGATE FUNCTIONS: Uporaba agregatnih funkcij, kot so SUM in COUNT, za izračunavanje skupnih vrednosti in števila vozil, kar omogoča zbiranje in analizo podatkov.

S temi poizvedbami pokrijemo širok spekter operacij, ki so ključne za analizo učinkovitosti podatkovnih baz. Tako lahko učinkovito merimo in primerjamo odzivnost ter zmogljivost podatkovnih baz PostgreSQL in Oracle.

Merjenje časov:

Za vsako poizvedbo se meri čas od začetka do konca izvajanja, prav tako se meri čas od začetka do konca polnjenja podatkovne baze. Uporabile se bodo metode razreda StopWatch (start(), stop(), getTotalTimeSeconds()), ki je del knjižnice Spring Framework.

Rezultati in Primerjava:

1. Analiza zbranih podatkov: Primerjava časov izvajanja za vsako poizvedbo med PostgreSQL in Oracle podatkovnima bazama ter primerjava časov polnjenja obeh podatkovnih baz s podatki iz CSV datoteke.
2. Grafična predstavitev: Prikaz rezultatov v obliki tabele za boljšo vizualizacijo.
3. Zaključki in priporočila: Na podlagi rezultatov se bodo podali zaključki o učinkovitosti vsake baze, tako bomo odgovorili na zastavljena raziskovalna vprašanja in potrdili ali zavrgli zastavljene hipoteze.

3.2 Razvojno okolje

Za razvoj aplikacije smo uporabili različna orodja in tehnologije, ki so omogočila učinkovito in hitro izdelavo, testiranje ter implementacijo rešitve. Razvojno okolje je bilo sestavljeno iz naslednjih komponent:

1. Spring Boot:

Opis: Spring Boot je priljubljen okvir za razvoj aplikacij v Javi, ki omogoča hitro in enostavno izdelavo samostojnih, produkcijsko pripravljenih aplikacij.

Uporaba: Uporabili smo ga za ustvarjanje osnovne strukture aplikacije, vključno z upravljanjem odvisnosti, konfiguracijo aplikacije in definicijo poslovne logike.

2. Java:

Opis: Programski jezik Java je eden najpogosteje uporabljenih jezikov za objektno programiranje.

Uporaba: Vse poslovne logike, obdelava podatkov in interakcije s podatkovnimi bazami so bile razvite v Javi.

3. ***Maven***:

Opis: Maven je orodje za upravljanje projektov in gradnjo, ki omogoča enostavno upravljanje odvisnosti in avtomatizacijo gradnje projektov.

Uporaba: Uporabili smo ga za upravljanje odvisnosti in avtomatizacijo gradnje projekta.

4. **Docker**:

Opis: Docker je platforma za razvoj, dostavo in zagon aplikacij v kontejnerjih.

Uporaba: Uporabili smo ga za ustvarjanje izoliranih kontejnerjev, v katerih sta bili nameščeni PostgreSQL in Oracle podatkovni bazi, kar je omogočilo enostavno in konsistentno nastavitev razvojnega okolja.

5. **PostgreSQL podatkovna baza**:

Opis: PostgreSQL je zmogljiva odprtokodna relacijska podatkovna baza.

Uporaba: Prva izmed dveh podatkovnih baz, ki smo jo uporabili za testiranje in primerjavo zmogljivosti.

6. **Oracle podatkovna baza**:

Opis: Oracle Database je ena vodilnih komercialnih relacijskih podatkovnih baz.

Uporaba: Druga podatkovna baza, ki smo jo uporabili za testiranje in primerjavo zmogljivosti.

7. **GitHub**

Opis: GitHub je spletna platforma za gostovanje in upravljanje različic kode, ki temelji na sistemu za nadzor različic Git. Omogoča razvijalcem shranjevanje, deljenje in sodelovanje na programskih projektih.

Uporaba: Uporabili smo ga za shranjevanje izvorne kode, sledenje spremembam in sodelovanje.

8. **IntelliJ**

Opis: IntelliJ IDEA je integrirano razvojno okolje (IDE) za razvoj programske opreme, ki ga razvija JetBrains. Podpira številne programske jezike, vključno z Javo, in ponuja napredna orodja za urejanje kode, refaktoring, navigacijo in razhroščevanje.

Uporaba: Uporabili smo ga za razvoj in razhroščevanje pri našem projektu, saj ponuja zmogljivo avtomatsko dopolnjevanje kode, integracijo z orodji za gradnjo in nadzor različic ter bogate funkcionalnosti za povečanje produktivnosti.

3.3 Izdelava aplikacije z ogrodjem Spring Boot

3.3.1 Postavitev projekta

Izdelava aplikacije se je pričela s pripravo okolja, v katerem bomo izdelali aplikacijo. Pričeli smo z inštalacijo IntelliJ razvojnega okolja. Obiskali smo uradno stran JetBrains, poiskali zadnjo različico za prenos in jo inštalirali na računalniku. Kreirali smo svoj račun pri JetBrains in plačali licenco za uporabo IntelliJ – Ultimate verzijo. Po namestitvi programa smo kreirali nov projekt: diplomska_izdelek, določili smo, da gre za Spring Boot projekt, v oknu za ustvarjanje novega projekta smo izbrali Maven kot projektni model in verzijo programskega jezika Java, Java 17. Znotraj projekta smo kreirali banner »Gregor Presker – Dispozicija«.

Nadaljevali smo z inštalacijo Dockerja in s kreiranjem Docker compose datoteke. Docker smo inštalirali na način, da smo obiskali uradno spletno stran Docker Hub in prenesli inštalacijske datoteke za program Docker. Po uspešni inštalaciji smo v terminalu preverili, ali Docker deluje pravilno. To smo naredili na način, da smo v terminalu najprej vpisali ukaz:

```
docker --version
```

Slika 1: Terminal ukaz – Docker version

Vir: Lasten vir

Ukaz nam je vrnil verzijo Dockerja, kar je pomenilo, da je Docker inštaliran na našem računalniku. Nadalje smo preverili, ali pravilno deluje z ukazom:

```
docker run hello-world
```

Slika 2: Terminal ukaz – Docker run

Vir: Lasten vir

Izpisalo se je sporočilo »Hello from Docker!«, kar je pomenilo, da deluje pravilno.

3.3.2 Postavitev in konfiguracija podatkovnih baz

Nadalje smo kreirali Docker compose datoteko znotraj projekta, »docker-compose.yml«.

Slika 3: Docker-compose.yml datoteka

Vir: Lasten vir

Znotraj datoteke docker-compose.yml smo konfigurirali dve bazi podatkov, Postgres in Oracle, vsaka z lastno Docker sliko in konfiguracijo. V nadaljevanju razlagamo konfiguracijo datoteke:

Version: '3.9' : Določa različico Docker Compose datotečnega formata. Verzija 3.9 je ena od najnovejših stabilnih različic, ki ponuja vse funkcionalnosti za upravljanje storitev in omrežij.

Services: Sekcija services določa različne storitve (kontejnerje), ki bodo v tej sestavi. V tem primeru sta dve storitvi: postgres in oracle.

Storitev postgres:

Image: 'postgres:14-alpine': Določa Docker sliko, ki se bo uporabila za storitev postgres. Uporablja se uradna Postgres slika, različica 14, ki temelji na Alpine Linuxu za manjšo velikost slike.

Ports: '5432:5432' pomeni, da se gostiteljski vhod (port) 5432 preslika na vhod 5432 znotraj kontejnerja.

Environment: Okoljske spremenljivke za konfiguracijo Postgres baze podatkov:

POSTGRES_USER=postgres: Nastavi uporabniško ime za Postgres.

POSTGRES_PASSWORD=1234: Nastavi geslo za Postgres.

Volumes: Določa povezave med gostiteljskimi datotekami ali mapami in kontejnerjem.

Storitev oracle:

Image: Določa Docker sliko za Oracle bazo podatkov iz uradnega Oracle kontejnerskega registra. Uporablja se zadnja različica brezplačne Oracle baze podatkov.

Ports: '1521:1521' pomeni, da se gostiteljski vhod (port) 1521 preslika na vhod 1521 znotraj kontejnerja.

Volumes: Določa prostorske volumne, ki se lahko uporabljajo za shranjevanje podatkov kontejnerjev. Ti volumni omogočajo trajno shranjevanje podatkov, tudi če so kontejnerji uničeni.

Po kreiranju datoteke smo želeli vzpostaviti podatkovni bazi v Docker kontejnerjih in preveriti, ali podatkovni bazi delujeta. To smo naredili na način, da smo v terminalu odprli mesto, kjer se nahaja datoteka docker-compose.yml in dali ukaz:

Slika 4: Ukaz terminal – Docker, compose-up

Vir: Lasten vir

To je kreiralo kontejnerje, znotraj katerih sta se naložili podatkovni bazi.

Nato smo preverili stanje zagnanih storitev z uporabo ukaza:

```
docker-compose ps
```

Slika 5: Terminal ukaz – Docker, compose ps

Vir: Lasten vir

Ta ukaz nam je prikazal sezname vseh storitev, ki tečejo znotraj Docker Compose projekta, skupaj s podrobnostmi o njihovem stanju.

Nadalje smo kreirali testne tabele v obeh podatkovnih bazah, jih napolnili s podatki in izvedli poizvedbo »SELECT * FROM test;«, ki nam je vrnila podatke iz obeh podatkovnih baz. Tako smo preverili, da podatkovni bazi delujeta pravilno.

Nadaljevali smo s konfiguriranjem povezav do baz v konfiguracijskih datotekah aplikacije. To smo naredili tako, da smo ustvarili znotraj našega projekta novo mapo, imenovano configuration. Znotraj te mape smo ustvarili dve novi Java datoteki, OracleDatabase configuration in PostgreSQL configuration, v katerih smo konfigurirali povezave do baz. Primer konfiguracije v OracleDatabase configuration datoteki:

```java
package org.example.diplomska_idelek.configuration;

import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

@Configuration  Gregor +1
public class OracleDatabaseConfiguration {

    @Value("${oracle.url}")
    private String url;
    @Value("${oracle.username}")
    private String username;
    @Value("${oracle.password}")
    private String password;

    @Bean  Gregor
    public DataSource oracleDataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        dataSource.setUrl(url);
        dataSource.setUsername(username);
        dataSource.setPassword(password);
        return dataSource;
    }

    @Bean  Gregor
    public JdbcTemplate oracleJdbcTemplate(@Qualifier("oracleDataSource") DataSource dataSource) {
        return new JdbcTemplate(dataSource);
    }
}
```

Slika 6: Kofiguracija podatkovnih baz

Vir: Lasten vir

Znotraj našega projekta smo znotraj datoteke »resources« kreirali mapo csv, v katero smo shranili datoteko Electric_Vehicle_Population_Data.csv, ki predstavlja naše masovne podatke – izsek neskončnega toka podatkov. In iz katere vzamemo podatke ter napolnimo z njimi naši podatkovni bazi.

3.3.3 Implementacija podatkovnih modelov

Nato smo implementirali dva podatkovna modela, ki smo ju uporabljali, to sta ElectricVehicle in ElectricVehicleGeneric. V nadaljevanju ju predstavljamo.

Razred ElectricVehicle vsebuje več spremenljivk tipa Strig, ki omogočajo shranjevanje podatkov, skupaj z metodami za nastavitev in pridobivanje vrednosti teh spremenljivk.

```
package org.example.diplomska_idelek.model;

import java.util.Objects;

public class ElectricVehicle {  10 usages  ▲ Gregor

    private String vin;  5 usages
    private String county;  5 usages
    private String city;  5 usages
    private String state;  5 usages
    private String postalCode;  5 usages
    private String modelYear;  5 usages
    private String make;  5 usages
    private String model;  5 usages
    private String electricVehicleType;  5 usages
    private String cafvEligibility;  5 usages
    private String electricRange;  5 usages
    private String baseMsrp;  5 usages
    private String legislativeDistrict;  5 usages
    private String dolVehicleId;  5 usages
    private String vehicleLocation;  5 usages
    private String electricUtility;  5 usages
    private String censusTract2020;  5 usages

    public String getVin() {  no usages  ▲ Gregor
        return vin;
    }

    public void setVin(String vin) {  1 usage  ▲ Gregor
        this.vin = vin;
    }

    public String getCounty() {  no usages  ▲ Gregor
        return county;
    }

    public void setCounty(String county) {  1 usage  ▲ Gregor
        this.county = county;
    }
```

Slika 7: Podatkovni model – ElectricVehicle

Vir: Lasten vir

Razred vsebuje metode (getterje in setterje) za vsako spremenljivko, ki omogočajo nastavitev in pridobivanje vrednosti posameznih spremenljivk.

Getterji: metode za pridobivanje vrednosti spremenljivk (npr. getVin, getCounty).

Setterji: metode za nastavitev vrednosti spremenljivk (npr. setVin, setCounty).

Nadalje sta še dodani dve metodi: equals in hashCode.

Metoda **equals** preverja, ali sta dva objekta ElectricVehicle enaka. To se zgodi, če so vrednosti vseh spremenljivk obeh objektov enake.

```
@Override  Gregor *
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    ElectricVehicle that = (ElectricVehicle) o;
    return Objects.equals(vin, that.vin) && Objects.equals(county, that.county) &&
            Objects.equals(city, that.city) && Objects.equals(state, that.state) &&
            Objects.equals(postalCode, that.postalCode) && Objects.equals(modelYear, that.modelYear)
            && Objects.equals(make, that.make) && Objects.equals(model, that.model) &&
            Objects.equals(electricVehicleType, that.electricVehicleType) &&
            Objects.equals(cafvEligibility, that.cafvEligibility) && Objects.equals(electricRange,
            that.electricRange) && Objects.equals(baseMsrp, that.baseMsrp)
            && Objects.equals(legislativeDistrict, that.legislativeDistrict) &&
            Objects.equals(dolVehicleId, that.dolVehicleId) && Objects.equals(vehicleLocation,
            that.vehicleLocation) && Objects.equals(electricUtility,
            that.electricUtility) && Objects.equals(censusTract2020, that.censusTract2020);
}
```

Slika 8: Podatkovni model – ElectricVehicle, metoda equals()

Vir: Lasten vir

Metoda **hashCode** generira hash-kodo za objekt ElectricVehicle na podlagi njegovih spremenljivk. To omogoča učinkovito uporabo objektov v strukturah podatkov, ki temeljijo na hash-kodah.

```
@Override  Gregor *
public int hashCode() {
    return Objects.hash(vin, county, city, state, postalCode, modelYear, make, model,
            electricVehicleType, cafvEligibility, electricRange, baseMsrp, legislativeDistrict,
            dolVehicleId, vehicleLocation, electricUtility, censusTract2020);
}
```

Slika 9: Podatkovni model – ElectricVehicle, metoda hashCode()

Vir: Lasten vir

Razred ElectricVehicle je podatkovni model, ki omogoča shranjevanje in manipulacijo podatkov. Vsebuje vnaprej določene spremenljivke, ki ustrezajo stolpcem v podatkovni bazi, kar omogoča enostavno preslikavo podatkov iz podatkovne baze v objekt tega razreda. Ta razred je bil uporabljen v mapperju za poizvedbo SELECT_ALL.

Razred ElectricVehicleGeneric je preprosta predstavitev generičnega podatkovnega modela, ki vsebuje pet spremenljivk tipa String. Omogoča fleksibilno shranjevanje in manipulacijo podatkov brez vnaprejšnje določitve specifičnih spremenljivk. Razred ElectricVehicleGeneric je bil uporabljen v mapperjih za poizvedbe, kot so GROUP_BY, GROUP_BY_MODEL, MODEL_PER_STATE, ELECTRIC_UTILITY_PROVIDER, YEAR_MODEL_MAKE, TESLA_MODEL_CITY_NUM in STATE_COUNTY_MAKE_MODEL_NUM. Ti mapperji preslikavajo samo določene atribute iz podatkovnih baz, namesto da bi zajemali vse podatke. S tem razredom se lahko enostavno preslika izbrana podmnožica podatkov iz podatkovnih baz glede na potrebe poizvedbe.

```
package org.example.diplomska_idelek.model;

public class ElectricVehicleGeneric {  55 usages  Gregor

    private String field1;  2 usages
    private String field2;  2 usages
    private String field3;  2 usages
    private String field4;  2 usages
    private String field5;  2 usages

    public String getField1() {  2 usages  Gregor
        return field1;
    }

    public void setField1(String field1) {  8 usages  Gregor
        this.field1 = field1;
    }

    public String getField2() {  no usages  Gregor
        return field2;
    }
```

Slika 10: Podatkovni model – ElectricVehicleGeneric

Vir: Lasten vir

Razred vsebuje metode (getterje in setterje), ki omogočajo nastavitev in pridobivanje vrednosti posameznih spremenljivk.

Getterji: metode za pridobivanje vrednosti spremenljivk (npr. getField1, getField2, getField3).

Setterji: metode za nastavitev vrednosti spremenljivk (npr. setField1, setField2, setField3).

Razred ElectricVehicleGeneric je preprost, a fleksibilen model podatkov, ki omogoča shranjevanje različnih podatkov brez vnaprejšnje specifikacije spremenljivk. Zaradi generične narave je uporaben v primerih, kjer struktura podatkov ni fiksna ali je potrebna prilagodljivost.

3.3.4 Določitev poizvedb

Potem ko smo definirali podatkovne modele, smo nadaljevali z določitvijo vseh potrebnih poizvedb v razredu Queries. V tem razredu smo definirali spremenljivke, ki vsebujejo SQL poizvedbe, uporabljene za pridobivanje podatkov iz podatkovnih baz ter za analizo njihove učinkovitosti. V nadaljevanju predstavljamo te spremenljivke:

1. public static final String SELECT_ALL

```
public static final String SELECT_ALL = "SELECT "
        ALL_FIELDS +
        FROM_TABLE_NAME;
```

Slika 11: Poizvedba1 – SELECT_ALL

Vir: Lasten vir

Spremenljivka vsebuje SQL poizvedbo za pridobivanje vseh zapisov iz tabele electric_vehicle. Združuje spremenljivki ALL_FIELDS in FROM_TABLE_NAME, da ustvari popoln SQL stavek, ki pridobi vse atribute za vsak zapis v tabeli.

```
private static final String ALL_FIELDS = "vin, county, city, state, postal_code, model_year, make, model, electric_vehicle_type, " +
        "cafv_eligibility, electric_range, base_msrp, legislative_district, dol_vehicle_id, vehicle_location, electric_utility, " +
        "census_tract_2020 ";
private static final String FROM_TABLE_NAME = "from electric_vehicle"; 1 usage
```

Slika 12: Poizvedba1 – ALL_FIELDS, FROM_TABLE_NAME

Vir: Lasten vir

2. public static final String GROUP_BY

```
public static final String GROUP_BY = "SELECT SUM(VEHICLE_COUNT) AS VEHICLE_SUM FROM ( SELECT STATE, COUNT(1) " +
        "AS VEHICLE_COUNT FROM ELECTRIC_VEHICLE GROUP BY STATE) TAB1";
```

Slika 13: Poizvedba 2 – GROUP_BY

Vir: Lasten vir

Spremenljivka vsebuje SQL poizvedbo, ki vsebuje tudi notranjo poizvedbo. Notranja poizvedba nam vrne število električnih vozil po posamezni državi. Zunanja poizvedba pa uporabi rezultat notranje poizvedbe, sešteje vse vrednosti iz stolpca VEHICLE_COUNT, kar vrne skupno število električnih vozil v vseh državah.

3. public static final String GROUP_BY_MODEL

```
public static final String GROUP_BY_MODEL = "SELECT model_year, COUNT(*) AS num_vehicles\n" +
        "FROM electric_vehicle\n" +
        "GROUP BY model_year";
```

Slika 14: Poizvedba 3 – GROUP_BY_MODEL

Vir: Lasten vir

Spremenljivka vsebuje SQL poizvedbo, ki združuje podatke iz tabele electric_vehicle glede na leto modela (model_year) in vrne število vozil, ki so bila izdelana v vsakem letu.

4. public static final String MODEL_PER_STATE

```
public static final String MODEL_PER_STATE = "SELECT state, make, count(MODEL) AS model\n" +
        "FROM electric_vehicle\n" +
        "WHERE 1 = 1\n" +
        "GROUP BY state, make, model\n" +
        "HAVING count(MODEL) > 5\n" +
        "order by state DESC, MAKE DESC, MODEL DESC";
```

Slika 15: Poizvedba 4 – MODEL_PER_STATE

Vir: Lasten vir

Spremenljivka vsebuje SQL poizvedbo, ki združuje podatke iz tabele electric_vehicle glede na državo (state), proizvajalca (make) in model (model). Poizvedba vrne število vozil za vsako kombinacijo države, proizvajalca in modela, vendar prikazuje le tiste kombinacije, kjer je več kot 5 vozil. Rezultate razvrsti po državi, proizvajalcu in modelu v padajočem vrstnem redu.

5. public static final String ELECTRIC_UTILITY_PROVIDER

```
public static final String ELECTRIC_UTILITY_PROVIDER = "SELECT make, model, electric_utility, COUNT(*) AS num_cars\n" +
        "            FROM electric_vehicle\n" +
        "            GROUP BY make, model, electric_utility order by make, num_cars DESC";
```

Slika 16: Poizvedba 5 – ELECTRIC_UTILITY_PROVIDER

Vir: Lasten vir

Spremenljivka vsebuje SQL poizvedbo, ki združuje podatke iz tabele electric_vehicle glede na proizvajalca (make), model (model) in ponudnika električne energije (electric_utility). Poizvedba vrne število vozil za vsako kombinacijo proizvajalca, modela in ponudnika električne energije. Rezultate razvrsti po proizvajalcu v naraščajočem vrstnem redu glede na število vozil.

6. public static final String YEAR_MODEL_MAKE

```
public static final String YEAR_MODEL_MAKE = "SELECT make, model, model_year, COUNT(*) AS num_vehicles FROM electric_vehicle " +
        "GROUP BY make, model, model_year order by make, model, num_vehicles DESC";
```

Slika 17: Poizvedba 6 – YEAR_MODEL_MAKE

Vir: Lasten vir

Spremenljivka vsebuje SQL poizvedbo, ki izbere podatke o proizvajalcu (make), modelu (model) in letu izdelave modela (model_year) električnih vozil iz tabele electric_vehicle. Poizvedba vrne število električnih vozil za vsakega proizvajalca in model v določenem letu ter rezultate uredi po proizvajalcu, modelu in številu vozil.

7. public static final String TESLA_MODEL_CITY_NUM

```
public static final String TESLA_MODEL_CITY_NUM = "SELECT make, model, city, COUNT(*) AS num_cars FROM electric_vehicle " +
        "where make ='TESLA' GROUP BY make, model, city order by num_cars DESC , city DESC";
```

Slika 18: Poizvedba 7 – TESLA_MODEL_CITY_NUM

Vir: Lasten vir

Spremenljivka vsebuje SQL poizvedbo, ki vrne število Teslinih vozil za vsak model v različnih mestih, pri čemer so rezultati urejeni najprej po številu vozil v vsakem mestu in nato po abecedi mest v padajočem vrstnem redu.

8. public static final String STATE_COUNTY_MAKE_MODEL_NUM

```
public static final String STATE_COUNTY_MAKE_MODEL_NUM = "SELECT state, COUNTY, make, model, COUNT(*) " +  2 usages
        "AS num_cars FROM electric_vehicle GROUP BY state, county, make, model order by state DESC, COUNTY DESC , " +
        "make DESC, model DESC, num_cars DESC";
```

Slika 19: Poizvedba 8 – STATE_COUNTY_MAKE_MODEL_NUM

Vir: Lasten vir

Spremenljivka vsebuje SQL poizvedbo, ki vrne število električnih vozil glede na kombinacijo države, okrožja, proizvajalca in modela, pri čemer so rezultati urejeni po državi, okrožju, proizvajalcu, modelu in številu vozil, vse v padajočem vrstnem redu.

3.3.5 Implementacija mapperjev

Nadaljevali smo z implementacijo vseh potrebnih mapperjev za poizvedbe v razredu Queries. Izdelali smo razrede (mapperje): AvgElectricRangeVehiclesByStateMapper, ElectricUtilityProviderMapper, ElectricVehicleGroupByMapper, ElectricVehicleGroupByModelMapper, ElectricVehicleMapper, NumberVehiclesMakeModelYearMapper, StateCountyMakeModelNumMapper, TeslaModelCityNumMapper.

Zaradi velikega števila razredov, predstavljamo razred: NumberVehiclesMakeModelYearMapper, s katerim pojasnjujemo namen uporabe mapperjev in njihovo strukturo.

```
package org.example.diplomska_idelek.mapper;

import org.example.diplomska_idelek.model.ElectricVehicleGeneric;
import org.springframework.jdbc.core.RowMapper;

import java.sql.ResultSet;
import java.sql.SQLException;

public class NumberVehiclesMakeModelYearMapper implements RowMapper<ElectricVehicleGeneric> {  2 usages  Gregor
    @Override  Gregor
    public ElectricVehicleGeneric mapRow(ResultSet rs, int rowNum) throws SQLException {
        ElectricVehicleGeneric ev = new ElectricVehicleGeneric();
        ev.setField1(rs.getString( columnLabel: "make"));
        ev.setField2(rs.getString( columnLabel: "model"));
        ev.setField3(rs.getString( columnLabel: "model_year"));
        return ev;
    }
}
```

Slika 20: Razred NumberVehiclesMakeModelYearMapper

Vir: Lasten vir

NumberVehiclesMakeModelYearMapper implementira vmesnik RowMapper<ElectricVehicleGeneric> iz knjižnice Spring JDBC. Namen tega razreda je preslikava vsake vrstice rezultata SQL poizvedbe v objekt ElectricVehicleGeneric. Ta razred je specifično zasnovan za poizvedbo YEAR_MODEL_MAKE, ki pridobiva podatke o proizvajalcu(make), modelu (model) in letu izdelave modela (model_year). Metoda mapRow je edina metoda, ki se implementira, ko uporabljamo vmesnik RowMapper. Ta metoda se pokliče za vsako vrstico rezultata poizvedbe in pretvori podatke iz ResultSet v objekt ElectricVehicleGeneric.

Najprej se ustvari nov objekt ElectricVehicleGeneric, nato sledi nastavljanje vrednosti spremenljivk objekta ElectricVehicleGeneric z vrednostmi iz ResultSet:

- vrednost spremenljivke field 1 se nastavi z vrednostjo stolpca make iz ResultSet,
- vrednost spremenljivke field 2 se nastavi z vrednostjo stolpca model iz ResultSet,
- vrednost spremenljivke field 3 se nastavi z vrednostjo stolpca model_year iz ResultSet.

Po tem, ko so vse vrednosti nastavljene, metoda vrne objekt ElectricVehicleGeneric.

Vsi ostali razredi so narejeni po istem postopku. Izjema je razred ElectricVehicleMapper, v katerem se ustvari nov objekt ElectricVehicle, ostali razredi ustvarijo objekt iz ElectricVehicleGeneric. Nastavljanje števila spremenljivk in njihovih vrednosti se razlikuje od razreda do razreda, odvisno od podatkov, ki jih potrebujejo poizvedbe, za katere se ti mapperji sploh ustvarijo. Vsak mapper je specifično zasnovan glede na poizvedbo, ki se nahaja v razredu Queries. V nadaljevanju navajamo še preostale mapperje in poizvedbe, za katere so bili zasnovani.

Razred AvgElectricRangeVehiclesByStateMapper je zasnovan za poizvedbo MODEL_PER_STATE.

Razred ElectricUtilityProviderMapper je zasnovan za poizvedbo ELECTRIC_UTILITY_PROVIDER.

Razred ElectricVehicleGroupByMapper je zasnovan za poizvedbo GROUP_BY.

Razred ElectricVehicleGroupByModelMapper je zasnovan za poizvedbo GROUP_BY_MODEL.

Razred ElectricVehicleMapper je zasnovan za poizvedbo SELECT_ALL.

Razred StateCountyMakeModelNumMapper je zasnovan za poizvedbo STATE_COUNTY_MAKE_MODEL_NUM.

Razred TeslaModelCityNumMapper je zasnovan za poizvedbo TESLA_MODEL_CITY_NUM.

3.3.6 Generiranje poizvedb

Na koncu smo še v razredu »CsvLoader« implementirali logiko polnjenja podatkovnih baz iz CSV datoteke ter izvajanje različnih poizvedb nad temi podatki, pri čemer meri čas polnjenja podatkovnih baz, izvajanja poizvedb na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov.

V nadaljevanju podrobneje pojasnjujemo posamezne dele kode.

Na začetku smo uvozili potrebne knjižnice.

```
package org.example.diplomska_idelek;

import jakarta.annotation.PostConstruct;
import org.example.diplomska_idelek.common.Queries;
import org.example.diplomska_idelek.mapper.*;
import org.example.diplomska_idelek.model.ElectricVehicle;
import org.example.diplomska_idelek.model.ElectricVehicleGeneric;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import org.springframework.util.StopWatch;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.List;
```

Slika 21: Razred CsvLoader – knjižnice

Vir: Lasten vir

Nato smo kreirali razred CsvLoader. Znotraj razreda smo najprej ustvarili logger za sledenje dogodkom v programu, definirali dva JdbcTemplate objekta za izvajanje SQL poizvedb na podatkovnih bazah PostgreSQL in Oracle ter inicializirali konstanti za identifikacijo obeh podatkovnih baz.

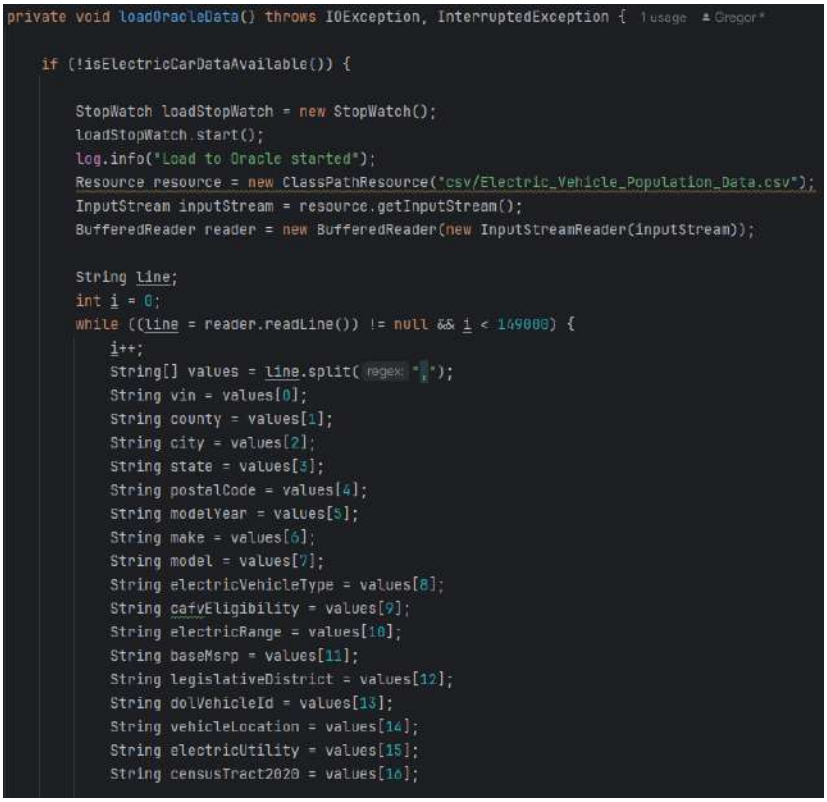
Slika 22: Razred CsvLoader – logger, JdbcTemplate objekt

Vir: Lasten vir

Nadalje smo kreirali metodo loadCsv(). Metoda izvrši v določenem zaporedju naslednje metode: loadOracleData(), loadPostgresData(), measureSelectAllStatement(), measureGroupBy(), numElectricVehiclesByModel(), averageElectricRangeVehiclesByState(), electricUtilityProvider(), numberVehiclesMakeModelYear(), teslaModelCityNum(), stateCountyMakeModelNum().

V nadaljevanju pojasnjujemo namen in način delovanja vsake metode.

1. loadOracleData():



Slika 23: Razred CsvLoader – loadOracleData(), 1. del

Vir: Lasten vir

```
            oracleJdbcTemplate.update(Queries.INSERT_STATEMENT,
                    vin, county, city, state, postalCode, modelYear, make, model, electricVehicleType,
                    cafvEligibility, electricRange, baseMsrp, legislativeDistrict, dolVehicleId,
                    vehicleLocation, electricUtility, censusTract2020);
            Thread.sleep( millis: 20);
        }
        loadStopWatch.stop();
        log.info("Load to oracle is finished in {} seconds", loadStopWatch.getTotalTimeSeconds());
        reader.close();
        inputStream.close();
    } else {
        log.info("Data already exists in table. I will not load!");
    }

}
```

Slika 24: Razred CsvLoader – loadOracleData(), 2. del

Vir: Lasten vir

Metoda loadOracleData() preverja, ali so podatki o električnih vozilih že na voljo v podatkovni bazi. Če podatki še niso na voljo, začne z nalaganjem. Uporablja metode razreda StopWatch, da meri čas nalaganja. Nato prebere podatke iz datoteke Electric_Vehicle_Population_Data.csv vrstico po vrstico in razdeli vsako vrstico na polja, pri čemer uporablja vejico kot ločilo. Nato uporabi JdbcTemplate, da izvede SQL poizvedbo INSERT_STATEMENT, ki vstavi prebrane podatke v podatkovno bazo Oracle. Na koncu izpiše sporočilo o končanem nalaganju in zapre odprte tokove podatkov. V primeru, da so podatki že na voljo v tabeli, izpiše sporočilo, da so podatki že v podatkovni bazi in da se ne bodo ponovno nalagali.

2. loadPostgresData():

```
private void loadPostgresData() throws IOException, InterruptedException {  1 usage   Gregor *

    if (!isElectricCarDataAvailable()) {

        StopWatch loadStopWatch = new StopWatch();
        loadStopWatch.start();
        log.info("Load to Postgres started");
        Resource resource = new ClassPathResource("csv/Electric_Vehicle_Population_Data.csv");
        InputStream inputStream = resource.getInputStream();
        BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

        String line;
        int i = 0;
        while ((line = reader.readLine()) != null && i < 149000) {
            i++;
            String[] values = line.split( regex: ",");
            String vin = values[0];
            String county = values[1];
            String city = values[2];
            String state = values[3];
            String postalCode = values[4];
            String modelYear = values[5];
            String make = values[6];
            String model = values[7];
            String electricVehicleType = values[8];
            String cafvEligibility = values[9];
            String electricRange = values[10];
            String baseMsrp = values[11];
            String legislativeDistrict = values[12];
            String dolVehicleId = values[13];
            String vehicleLocation = values[14];
            String electricUtility = values[15];
            String censusTract2020 = values[16];
```

Slika 25: Razred CsvLoader – loadPostgresData(), 1. del

Vir: Lasten vir

```
            postgresJdbcTemplate.update(Queries.INSERT_STATEMENT,
                    vin, county, city, state, postalCode, modelYear, make, model, electricVehicleType,
                    cafvEligibility, electricRange, baseMsrp, legislativeDistrict, dolVehicleId,
                    vehicleLocation, electricUtility, censusTract2020);
            Thread.sleep( millis: 20);
        }
        loadStopWatch.stop();
        log.info("Load to postgres is finished in {} seconds", loadStopWatch.getTotalTimeSeconds());
        reader.close();
        inputStream.close();
    } else {
        log.info("Data already exists in table. I will not load!");
    }

}
```

Slika 26: Razred CsvLoader – loadPostgresData(), 2. del

Vir: Lasten vir

Metoda loadPostgresData() preverja, ali so podatki o električnih vozilih že na voljo v podatkovni bazi. Če podatki še niso na voljo, začne z nalaganjem. Uporablja metode razreda StopWatch, da meri čas nalaganja. Nato prebere podatke iz datoteke Electric_Vehicle_Population_Data.csv vrstico po vrstico in razdeli vsako vrstico na polja, pri čemer uporablja vejico kot ločilo. Nato uporabi JdbcTemplate, da izvede SQL poizvedbo INSERT_STATEMENT, ki vstavi prebrane podatke v podatkovno bazo PostgreSQL. Na koncu izpiše sporočilo o končanem nalaganju in zapre odprte tokove podatkov. V primeru, da so podatki že na voljo v tabeli, izpiše sporočilo, da so podatki že v podatkovni bazi in da se ne bodo ponovno nalagali.

3. measureSelectAllStatement()

```
private void measureSelectAllStatement() {  1 usage   ▲ Gregor *

    StopWatch postgresStopWatch = initStopWatch();
    postgresStopWatch.start();
    log.info("Select all statement started on {} database!", POSTGRES);
    List<ElectricVehicle> postgreData = postgresJdbcTemplate.query(Queries.SELECT_ALL, new ElectricVehicleMapper());

    postgresStopWatch.stop();
    log.info("Select all statement is finished for {} database with row size of {} and it took {} miliseconds!",
            POSTGRES, postgreData.size(), postgresStopWatch.getTotalTimeMillis());

    StopWatch oracleStopWatch = initStopWatch();
    oracleStopWatch.start();
    log.info("Select all statement started on {} database!", ORACLE);
    List<ElectricVehicle> oracleData = oracleJdbcTemplate.query(Queries.SELECT_ALL, new ElectricVehicleMapper());
    oracleStopWatch.stop();
    log.info("Select all statement is finished for {} database with row size of {} and it took {} miliseconds!",
            ORACLE, oracleData.size(), oracleStopWatch.getTotalTimeMillis());
}
```

Slika 27: Razred CsvLoader – measureSelectAllStatement()

Vir: Lasten vir

Metoda measureSelectAllStatement() meri čas izvajanja poizvedbe SELECT_ALL na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov. Najprej inicializira StopWatch objekt, da meri čas izvajanja. Nato začne meriti čas za PostgreSQL in izvede poizvedbo s pomočjo JdbcTemplate, pri čemer uporabi ElectricVehicleMapper za preslikavo rezultatov. Ko sta poizvedba in preslikava končani, ustavi merjenje časa in izpiše sporočilo o končanem izvajanju, vključno s številom vrstic v rezultatu in časom, ki sta ga poizvedba in preslikava potrebovali v milisekundah. Enak postopek se ponovi tudi za Oracle bazo.

4. measureGroupBy()

```
private void measureGroupBy() {  1 usage   Gregor*

    StopWatch postgresStopWatch = initStopWatch();
    postgresStopWatch.start();
    log.info("Group by statement started on {} database!", POSTGRES);
    List<ElectricVehicleGeneric> postgreData = postgresJdbcTemplate.query(Queries.GROUP_BY, new ElectricVehicleGroupByMapper());

    postgresStopWatch.stop();
    log.info("Group by statement is finished for {} database with row size of {} and it took {} miliseconds!",
            POSTGRES, postgreData.get(0).getField1(), postgresStopWatch.getTotalTimeMillis());

    StopWatch oracleStopWatch = initStopWatch();
    oracleStopWatch.start();
    log.info("Group by statement started on {} database!", ORACLE);
    List<ElectricVehicleGeneric> oracleData = oracleJdbcTemplate.query(Queries.GROUP_BY, new ElectricVehicleGroupByMapper());
    oracleStopWatch.stop();
    log.info("Group by statement is finished for {} database with row size of {} and it took {} miliseconds!",
            ORACLE, oracleData.get(0).getField1(), oracleStopWatch.getTotalTimeMillis());
}
```

Slika 28: Razred CsvLoader – measureGroupBy()

Vir: Lasten vir

Metoda measureGroupBy() meri čas izvajanja poizvedbe GROUP_BY na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov. Najprej inicializira StopWatch objekt za merjenje časa. Nato začne meriti čas za PostgreSQL in izvede poizvedbo s pomočjo JdbcTemplate, pri čemer uporabi ElectricVehicleGroupByMapper za preslikavo rezultatov. Ko sta poizvedba in preslikava končani, ustavi merjenje časa in izpiše sporočilo o končanem izvajanju, vključno s številom vrstic v rezultatu in časom, ki sta ga poizvedba in preslikava potrebovali v milisekundah. Enak postopek se ponovi tudi za Oracle bazo.

5. numElectricVehiclesByModel()

```
private void numElectricVehiclesByModel() {  1 usage   ± Gregor *

    StopWatch postgresStopWatch = initStopWatch();
    postgresStopWatch.start();
    log.info("Number of Electric vehicles by model statement started on {} database!", POSTGRES);
    List<ElectricVehicleGeneric> postgreData = postgresJdbcTemplate.query(Queries.GROUP_BY_MODEL, new ElectricVehicleGroupByModelMapper());

    postgresStopWatch.stop();
    log.info("Number of Electric vehicles by model statement is finished for {} database with row size of {} and it took {} miliseconds!",
            POSTGRES, postgreData.size(), postgresStopWatch.getTotalTimeMillis());

    StopWatch oracleStopWatch = initStopWatch();
    oracleStopWatch.start();
    log.info("Number of Electric vehicles by model statement started on {} database!", ORACLE);
    List<ElectricVehicleGeneric> oracleData = oracleJdbcTemplate.query(Queries.GROUP_BY_MODEL, new ElectricVehicleGroupByModelMapper());
    oracleStopWatch.stop();
    log.info("Number of Electric vehicles by model statement is finished for {} database with row size of {} and it took {} miliseconds!",
            ORACLE, oracleData.size(), oracleStopWatch.getTotalTimeMillis());
}
```

Slika 29: Razred CsvLoader – numElectricVehiclesByModel()

Vir: Lasten vir

Metoda numElectricVehiclesByModel() meri čas izvajanja poizvedbe GROUP_BY_MODEL na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov. Najprej inicializira StopWatch objekt za merjenje časa. Nato začne meriti čas za PostgreSQL in izvede poizvedbo s pomočjo JdbcTemplate, pri čemer uporabi ElectricVehicleGroupByModelMapper za preslikavo rezultatov. Ko sta poizvedba in preslikava končani, ustavi merjenje časa in izpiše sporočilo o končanem izvajanju, vključno s številom vrstic v rezultatu in časom, ki sta ga poizvedba in preslikava potrebovali v milisekundah. Enak postopek se ponovi tudi za Oracle bazo.

6. averageElectricRangeVehiclesByState()

```
private void averageElectricRangeVehiclesByState() {  1 usage   ± Gregor *

    StopWatch postgresStopWatch = initStopWatch();
    postgresStopWatch.start();
    log.info("Average electric range of vehicles by state statement started on {} database!", POSTGRES);
    List<ElectricVehicleGeneric> postgreData = postgresJdbcTemplate.query(Queries.MODEL_PER_STATE, new AvgElectricRangeVehiclesByStateMapper());

    postgresStopWatch.stop();
    log.info("Average electric range of vehicles by state statement is finished for {} database with row size of {} and it took {} miliseconds!",
            POSTGRES, postgreData.size(), postgresStopWatch.getTotalTimeMillis());

    StopWatch oracleStopWatch = initStopWatch();
    oracleStopWatch.start();
    log.info("Average electric range of vehicles by state statement started on {} database!", ORACLE);
    List<ElectricVehicleGeneric> oracleData = oracleJdbcTemplate.query(Queries.MODEL_PER_STATE, new AvgElectricRangeVehiclesByStateMapper());
    oracleStopWatch.stop();
    log.info("Average electric range of vehicles by state statement is finished for {} database with row size of {} and it took {} miliseconds!",
            ORACLE, oracleData.size(), oracleStopWatch.getTotalTimeMillis());
}
```

Slika 30: Razred CsvLoader – averageElectricRangeVehiclesByState()

Vir: Lasten vir

Metoda averageElectricRangeVehiclesByState() meri čas izvajanja poizvedbe MODEL_PER_STATE na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov. Najprej je ustvarjen objekt StopWatch za merjenje časa. Nato se čas začne meriti za PostgreSQL, izvede se poizvedba z uporabo JdbcTemplate, pri čemer se za preslikavo rezultatov uporablja AvgElectricRangeVehiclesByStateMapper. Ko sta poizvedba in preslikava končani, ustavi merjenje časa in izpiše sporočilo o končanem izvajanju, vključno s številom vrstic v rezultatu in časom, ki sta ga poizvedba in preslikava potrebovali v milisekundah. Enak postopek se ponovi tudi za Oracle bazo.

7. electricUtilityProvider()

```
private void electricUtilityProvider() {  1 usage   ± Gregor*

    StopWatch postgresStopWatch = initStopWatch();
    postgresStopWatch.start();
    log.info("Electric Utility Provider statement started on {} database!", POSTGRES);
    List<ElectricVehicleGeneric> postgreData = postgresJdbcTemplate.query(Queries.ELECTRIC_UTILITY_PROVIDER,
            new ElectricUtilityProviderMapper());

    postgresStopWatch.stop();
    log.info("Electric Utility Provider statement is finished for {} database with row size of {} and it took {} miliseconds!",
            POSTGRES, postgreData.size(), postgresStopWatch.getTotalTimeMillis());

    StopWatch oracleStopWatch = initStopWatch();
    oracleStopWatch.start();
    log.info("Electric Utility Provider statement started on {} database!", ORACLE);
    List<ElectricVehicleGeneric> oracleData = oracleJdbcTemplate.query(Queries.ELECTRIC_UTILITY_PROVIDER,
            new ElectricUtilityProviderMapper());
    oracleStopWatch.stop();
    log.info("Electric Utility Provider vehicles by state statement is finished for {} database with row size of {} and it took {} miliseconds!",
            ORACLE, oracleData.size(), oracleStopWatch.getTotalTimeMillis());
}
```

Slika 31: Razred CsvLoader – electricUtilityProvider()

Vir: Lasten vir

Metoda electricUtilityProvider() meri čas izvajanja poizvedbe ELECTRIC_UTILITY_PROVIDER na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov. Najprej inicializira StopWatch objekt za merjenje časa. Nato začne meriti čas za PostgreSQL in izvede poizvedbo s pomočjo JdbcTemplate, pri čemer uporabi ElectricUtilityProviderMapper za preslikavo rezultatov. Ko sta poizvedba in preslikava končani, ustavi merjenje časa in izpiše sporočilo o končanem izvajanju, vključno s številom vrstic v rezultatu in časom, ki sta ga poizvedba in preslikava potrebovali v milisekundah. Enak postopek se ponovi tudi za Oracle bazo.

8. numberVehiclesMakeModelYear()

```
private void numberVehiclesMakeModelYear() {  1 usage   ± Gregor

    StopWatch postgresStopWatch = initStopWatch();
    postgresStopWatch.start();
    log.info("Number of vehicles per company, per model, per model year statement started on {} database!", POSTGRES);
    List<ElectricVehicleGeneric> postgreData = postgresJdbcTemplate.query(Queries.YEAR_MODEL_MAKE, new NumberVehiclesMakeModelYearMapper());

    postgresStopWatch.stop();
    log.info("Number of vehicles per company, per model, per model year statement is finished for {} database with row size of {} and it took " +
            "{} miliseconds!", POSTGRES, postgreData.size(), postgresStopWatch.getTotalTimeMillis());

    StopWatch oracleStopWatch = initStopWatch();
    oracleStopWatch.start();
    log.info("RNumber of vehicles per company, per model, per model year statement started on {} database!", ORACLE);
    List<ElectricVehicleGeneric> oracleData = oracleJdbcTemplate.query(Queries.YEAR_MODEL_MAKE, new NumberVehiclesMakeModelYearMapper());
    oracleStopWatch.stop();
    log.info("Number of vehicles per company, per model, per model year statement is finished for {} database with row size of {} and it took" +
            " {} miliseconds!", ORACLE, oracleData.size(), oracleStopWatch.getTotalTimeMillis());
}
```

Slika 32: Razred CsvLoader – numberVehiclesMakeModelYear()

Vir: Lasten vir

Metoda numberVehiclesMakeModelYear() meri čas izvajanja poizvedbe YEAR_MODEL_MAKE na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov. Najprej inicializira StopWatch objekt za merjenje časa. Nato začne meriti čas za PostgreSQL in izvede poizvedbo s pomočjo JdbcTemplate, pri čemer uporabi NumberVehiclesMakeModelYearMapper za preslikavo rezultatov. Ko sta poizvedba in preslikava končani, ustavi merjenje časa in izpiše sporočilo o končanem izvajanju, vključno s številom vrstic v rezultatu in časom, ki sta ga poizvedba in preslikava potrebovali v milisekundah. Enak postopek se ponovi tudi za Oracle bazo.

9. teslaModelCityNum()

```
private void teslaModelCityNum() {  1 usage   ± Gregor*

    StopWatch postgresStopWatch = initStopWatch();
    postgresStopWatch.start();
    log.info("Number of Tesla vehicles per model per city statement started on {} database!", POSTGRES);
    List<ElectricVehicleGeneric> postgreData = postgresJdbcTemplate.query(Queries.TESLA_MODEL_CITY_NUM, new TeslaModelCityNumMapper());

    postgresStopWatch.stop();
    log.info("Number of Tesla vehicles per model per city statement is finished for {} database with row size of {} and it took {} " +
            "miliseconds!", POSTGRES, postgreData.size(), postgresStopWatch.getTotalTimeMillis());

    StopWatch oracleStopWatch = initStopWatch();
    oracleStopWatch.start();
    log.info("Number of Tesla vehicles per model per city statement started on {} database!", ORACLE);
    List<ElectricVehicleGeneric> oracleData = oracleJdbcTemplate.query(Queries.TESLA_MODEL_CITY_NUM, new TeslaModelCityNumMapper());
    oracleStopWatch.stop();
    log.info("Number of Tesla vehicles per model per city statement is finished for {} database with row size of {} and it took {} " +
            "miliseconds!", ORACLE, oracleData.size(), oracleStopWatch.getTotalTimeMillis());
}
```

Slika 33: Razred CsvLoader – teslaModelCityNum()

Vir: Lasten vir

Metoda teslaModelCityNum() meri čas izvajanja poizvedbe TESLA_MODEL_CITY_NUM na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov. Najprej inicializira StopWatch objekt za merjenje časa. Nato začne meriti čas za PostgreSQL in izvede poizvedbo s pomočjo JdbcTemplate, pri čemer uporabi TeslaModelCityNumMapper za preslikavo rezultatov. Ko sta poizvedba in preslikava končani, ustavi merjenje časa in izpiše sporočilo o končanem izvajanju, vključno s številom vrstic v rezultatu in časom, ki sta ga poizvedba in preslikava potrebovali v milisekundah. Enak postopek se ponovi tudi za Oracle bazo.

10. stateCountyMakeModelNum()

```
private void stateCountyMakeModelNum() {  1 usage   ▲ Gregor *

    StopWatch postgresStopWatch = initStopWatch();
    postgresStopWatch.start();
    log.info("Number of vehicles by State, by county, by company and its models statement started on {} database!", POSTGRES);
    List<ElectricVehicleGeneric> postgreData = postgresJdbcTemplate.query(Queries.STATE_COUNTY_MAKE_MODEL_NUM, new StateCountyMakeModelNumMapper());

    postgresStopWatch.stop();
    log.info("Number of vehicles by State, by county, by company and its models statement is finished for {} database with row size of {} and it took {} " +
            "miliseconds!", POSTGRES, postgreData.size(), postgresStopWatch.getTotalTimeMillis());

    StopWatch oracleStopWatch = initStopWatch();
    oracleStopWatch.start();
    log.info("Number of vehicles by State, by county, by company and its models statement started on {} database!", ORACLE);
    List<ElectricVehicleGeneric> oracleData = oracleJdbcTemplate.query(Queries.STATE_COUNTY_MAKE_MODEL_NUM, new StateCountyMakeModelNumMapper());
    oracleStopWatch.stop();
    log.info("Number of vehicles by State, by county, by company and its models statement is finished for {} database with row size of {} and it took {} " +
            "miliseconds!", ORACLE, oracleData.size(), oracleStopWatch.getTotalTimeMillis());
}
```

Slika 34: Razred CsvLoader – stateCountyMakeModelNum()

Vir: Lasten vir

Metoda stateCountyMakeModelNum() meri čas izvajanja poizvedbe STATE_COUNTY_MAKE_MODEL_NUM na obeh podatkovnih bazah in čas, ki je potreben za preslikavo rezultatov. Najprej inicializira StopWatch objekt za merjenje časa. Nato začne meriti čas za PostgreSQL in izvede poizvedbo s pomočjo JdbcTemplate, pri čemer uporabi StateCountyMakeModelNumMapper za preslikavo rezultatov. Ko sta poizvedba in preslikava končani, ustavi merjenje časa in izpiše sporočilo o končanem izvajanju, vključno s številom vrstic v rezultatu in časom, ki sta ga poizvedba in preslikava potrebovali v milisekundah. Enak postopek se ponovi tudi za Oracle bazo.

Na koncu razreda CsvLoader še imamo metodi: private boolean isElectricCarDataAvailable() in private StopWatch initStopWatch(). V nadaljevanju podajamo njun opis.

1. private boolean isElectricCarDataAvailable()

```
private boolean isElectricCarDataAvailable() {  1 usage   Gregor
    Integer rowCount = postgresJdbcTemplate.queryForObject(Queries.CONTROL_QUERY, Integer.class);
    return rowCount > 0;
}
```

Slika 35: Razred CsvLoader – isElectricCarDataAvailable()

Vir: Lasten vir

Ta metoda preverja, ali so podatki o električnih avtomobilih na voljo v podatkovni bazi. Izvaja poizvedbo CONTROL_QUERY. Rezultat te poizvedbe, ki predstavlja število vrstic, shranimo v spremenljivko rowCount. Če je rowCount večji od 0, pomeni, da obstajajo podatki, metoda vrne true, sicer vrne false. Ta metoda se uporablja za odločanje, ali naj se podatki naložijo v podatkovni bazi ali ne.

2. private StopWatch initStopWatch()

```
private StopWatch initStopWatch(){
    return new StopWatch();
}
```

Slika 36: Razred CsvLoader – initStopWatch()

Vir: Lasten vir

Ta metoda ustvari in vrne novo instanco razreda StopWatch iz knjižnice Spring Framework. To je pomožna metoda, ki se uporablja za inicializacijo merilnika časa pred izvajanjem določenih nalog, kot je merjenje časa izvajanja določenih poizvedb ali operacij.

3.4 Izdelava podatkovnega modela v podatkovnih bazah Oracle in PostgreSQL

V procesu izdelave podatkovnega modela za podatkovni bazi Oracle in PostgreSQL smo sledili naslednjim korakom:

1. **Analiza strukture podatkov:** Najprej smo temeljito pregledali strukturo podatkov v CSV datoteki, ki je vsebovala informacije o električnih vozilih. Preučili smo vrstice in stolpce ter določili, kako najbolje organizirati podatke v tabelah.
2. **Načrtovanje tabel:** Na podlagi analize strukture podatkov v CSV datoteki smo načrtovali tabele za obe podatkovni bazi. Obe morata biti identični po zgradbi (stolpci) glede na tabelo v CSV datoteki.
3. **Definicija stolpcev:** Za vsako tabelo smo določili stolpce, ki so ustrezali poljem v CSV datoteki.
4. **Ustvarjanje tabel in prenos podatkov:** Načrtovane tabele smo ustvarili pred zagonom aplikacije v obeh podatkovnih bazah Oracle in PostgreSQL. Nato se je ob zagonu aplikacije izvedel prenos podatkov iz CSV datoteke v tabele v obeh podatkovnih bazah.
5. **Preverjanje, prilagajanje in testiranje:** Po ustvarjanju tabel smo preverili, ali ustrezajo našim pričakovanjem in zastavljenemu podatkovnemu modelu. V kolikor bi bilo potrebno, bi tabele tudi ustrezno prilagodili. Testiranje podatkovnega modela smo izvedli na način, da smo se prepričali, da so podatki pravilno prenešeni in dostopni za uporabo v obeh podatkovnih bazah. Preverili smo tudi rezultate zastavljenih poizvedb v aplikaciji in jih primerjali s pričakovanimi rezultati. Ti so se morali ujemati.

3.5 Ugotovitve

Razvoj aplikacije je potekal na način, da smo najprej naredili analizo aplikacije, ki jo bomo izdelali, nato pa določili njene specifikacije.

Glavne cilje, ki smo si jih z aplikacijo zastavili so bili: a) narediti aplikacijo, ki bo omogočala primerjavo časa polnjenja podatkovnih baz PostgreSQL in Oracle iz CSV datoteke in za primerjavo časa izvajanja poizvedb; b) narediti oceno podatkovnih baz, kako obravnavata večje količine podatkov in kompleksnejše poizvedbe; c) razviti aplikacijo, ki bo omogočala uvoz podatkov iz CSV datoteke v bazi in izvajanje poizvedb.

Funkcionalnosti aplikacije so obsegale: a) uvoz podatkov iz CSV datoteke in njihovo vstavljanje v Oracle in PostgreSQL podatkovni bazi, b) izvajanje SQL poizvedb, c) merjenje časa izvajanja poizvedb in prenosa podatkov v podatkovni bazi.

Tehnologije in orodja, ki smo jih uporabili za izdelavo aplikacije so: Spring Boot, Java, Maven, GitHub, IntelliJ, Docker, PostgreSQL in Oracle podatkovni bazi.

V sklopu specifikacij smo določili strukturo aplikacije, določili smo, da uporabniški vmesnik ne bo potreben, saj bomo rezultate aplikacije izpisovali v terminalu. Določili smo operacije, ki jih bomo uporabljali pri poizvedbah, kot so SELECT, INSERT, GROUP BY, WHERE, HAVING, ORDER BY in AGREGATE FUNCTIONS (SUM, COUNT). Določili smo tudi metode, ki so bile uporabljene za merjenje časa, potrebnega za izvedbo poizvedbe in polnjenja podatkovnih baz. Določili smo, da bomo rezultate aplikacije analizirali in grafično prikazali, na podlagi tega pa prišli do zaključkov, s katerimi bomo odgovorili na zastavljena raziskovalna vprašanja in potrdili ali ovrgli zastavljene hipoteze.

Izdelava aplikacije z ogrodjem Spring Boot je potekala na sledeč način: a) postavitev projekta (priprava okolja za razvoj), b) postavitev in konfiguracija podatkovnih baz, c) implementacija podatkovnih modelov, d) določitev poizvedb, e) implementacija Mapperjev za poizvedbe, f) generiranje poizvedb.

Izdelava podatkovnega modela v podatkovnih bazah Oracle in PostgreSQL je potekala na sledeč način: a) analiza strukture podatkov v CSV datoteki, b) načrtovanje tabel za PostgreSQL in Oracle podatkovni bazi, c) definiranje stolpcev (tipi podatkov), d) ustvarjanje tabel in prenos podatkov, e) preverjanje, prilagajanje in testiranje.

4 MERITVE

Meritve smo opravili s pomočjo Spring Boot aplikacije, ki smo jo izdelali v ta namen. Ob zagonu je aplikacija najprej preverila, ali sta podatkovni bazi že napolnjeni. Ker naši podatkovni bazi še nista vsebovali podatkov, je aplikacija prenesla podatke iz CSV datoteke v obe podatkovni bazi. Aplikacija je napolnila obe podatkovni bazi na način, da je vsako vrstico iz CSV datoteke najprej prenesla v Oracle bazo, nato pa še v PostgreSQL bazo. Skupno je bilo v obe podatkovni bazi prenešenih sto devetinštirideset tisoč vrstic. Aplikacija je izmerila hitrost polnjenja obeh podatkovnih baz.

Po napolnjenih podatkovnih bazah je aplikacija pričela z meritvami hitrosti opravljenih poizvedb.

Rezultate teh meritev predstavljamo v nadaljevanju.

4.1 Oracle baza

V nadaljevanju podajamo rezultate meritev Oracle podatkovne baze, najprej meritve za polnjenje podatkovne baze, nato meritve za vsako poizvedbo posebej. Poizvedbe smo podrobno opisali v podpoglavju 3.3.4.

Polnjenje podatkovne baze:

Rezultat meritve:

```
Load to Oracle started
Load to oracle is finished in 6438.373126945 seconds
```

Slika 37: Rezultat meritve – polnjenje baze, Oracle

Vir: Lasten vir

Hitrost vnosa podatkov iz CSV datoteke v Oracle podatkovno bazo je znašala 6438,37 sekunde oziroma 1 ura, 47 minut in 17 sekund.

1. **Poizvedba:**

public static final String SELECT_ALL

Rezultat meritve:

```
Select all statement started on ORACLE database!
Select all statement is finished for ORACLE database with row size of 149000 and it took 1220 miliseconds!
```

Slika 38: Rezultat meritve – poizvedba 1, Oracle

Vir: Lasten vir

Oracle baza je izvršila poizvedbo v 1220 milisekundah.

2. **Poizvedba:**

public static final String GROUP_BY

Rezultat meritve:

```
Group by statement started on ORACLE database!
Group by statement is finished for ORACLE database with row size of 149000 and it took 45 miliseconds!
```

Slika 39: Rezultat meritve – poizvedba 2, Oracle

Vir: Lasten vir

Oracle baza je izvršila poizvedbo v 45 milisekundah.

3. **Poizvedba:**

public static final String GROUP_BY_MODEL

Rezultat meritve:

```
Number of Electric vehicles by model statement started on ORACLE database!
Number of Electric vehicles by model statement is finished for ORACLE database with row size of 21 and it took 44 miliseconds!
```

Slika 40: Rezultat meritve – poizvedba 3, Oracle

Vir: Lasten vir

Oracle baza je izvršila poizvedbo v 44 milisekundah.

4. Poizvedba:

public static final String MODEL_PER_STATE

Rezultat meritve:

```
Average electric range of vehicles by state statement started on ORACLE database!
Average electric range of vehicles by state statement is finished for ORACLE database with row size of 129 and it took 52 miliseconds!
```

Slika 41: Rezultat meritve – poizvedba 4, Oracle

Vir: Lasten vir

Oracle baza je izvršila poizvedbo v 52 milisekundah.

5. Poizvedba:

public static final String ELECTRIC_UTILITY_PROVIDER

Rezultat meritve:

```
Electric Utility Provider statement started on ORACLE database!
Electric Utility Provider vehicles by state statement is finished for ORACLE database with row size of 3086 and it took 111 miliseconds!
```

Slika 42: Rezultat meritve – poizvedba 5, Oracle

Vir: Lasten vir

Oracle baza je izvršila poizvedbo v 111 milisekundah.

6. Poizvedba:

public static final String YEAR_MODEL_MAKE

Rezultat meritve:

```
Number of vehicles per company, per model, per model year statement started on ORACLE database!
Number of vehicles per company, per model, per model year statement is finished for ORACLE database with row size of 477 and it took 55 miliseconds!
```

Slika 43: Rezultat meritve – poizvedba 6, Oracle

Vir: Lasten vir

Oracle baza je izvršila poizvedbo v 55 milisekundah.

7. Poizvedba:

public static final String TESLA_MODEL_CITY_NUM

Rezultat meritve:

```
Number of Tesla vehicles per model per city statement started on ORACLE database!
Number of Tesla vehicles per model per city statement is finished for ORACLE database with row size of 1257 and it took 57 miliseconds!
```

Slika 44: Rezultat meritve – poizvedba 7, Oracle

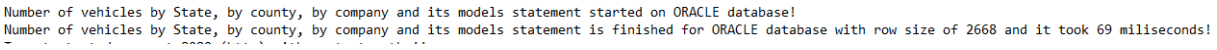
Vir: Lasten vir

Oracle baza je izvršila poizvedbo v 57 milisekundah.

8. Poizvedba:

public static final String STATE_COUNTY_MAKE_MODEL_NUM

Rezultat meritve:



Slika 45: Rezultat meritve – poizvedba 8, Oracle

Vir: Lasten vir

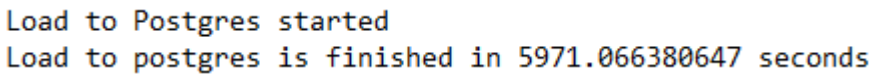
Oracle baza je izvršila poizvedbo v 69 milisekundah.

4.2 PostgreSQL baza

V nadaljevanju podajamo rezultate meritev PostgreSQL podatkovne baze, najprej meritve za polnjenje podatkovne baze, nato meritve za vsako poizvedbo posebej.

Polnjenje podatkovne baze:

Rezultat meritve:



Slika 46: Rezultat meritve – polnjenje baze, PostgreSQL

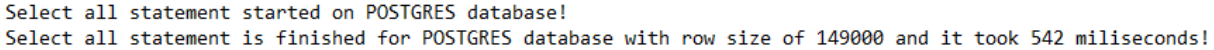
Vir: Lasten vir

Hitrost vnosa podatkov iz CSV datoteke v PostgreSQL podatkovno bazo je znašala 5971,07 sekunde oziroma 1 ura, 39 minut in 31 sekund.

1. Poizvedba:

public static final String SELECT_ALL

Rezultat meritve:



Slika 47: Rezultat meritve – poizvedba 1, PostgreSQL

Vir: Lasten vir

PostgreSQL baza je izvršila poizvedbo v 542 milisekundah.

2. Poizvedba:

public static final String GROUP_BY

Rezultat meritve:

```
Group by statement started on POSTGRES database!
Group by statement is finished for POSTGRES database with row size of 149000 and it took 29 miliseconds!
```

Slika 48: Rezultat meritve – poizvedba 2, PostgreSQL

Vir: Lasten vir

PostgreSQL baza je izvršila poizvedbo v 29 milisekundah.

3. Poizvedba:

public static final String GROUP_BY_MODEL

Rezultat meritve:

```
Number of Electric vehicles by model statement started on POSTGRES database!
Number of Electric vehicles by model statement is finished for POSTGRES database with row size of 21 and it took 28 miliseconds!
```

Slika 49: Rezultat meritve – poizvedba 3, PostgreSQL

Vir: Lasten vir

PostgreSQL baza je izvršila poizvedbo v 28 milisekundah**.**

4. Poizvedba:

public static final String MODEL_PER_STATE

Rezultat meritve:

```
Average electric range of vehicles by state statement started on POSTGRES database!
Average electric range of vehicles by state statement is finished for POSTGRES database with row size of 129 and it took 42 miliseconds!
```

Slika 50: Rezultat meritve – poizvedba 4, PostgreSQL

Vir: Lasten vir

PostgreSQL baza je izvršila poizvedbo v 42 milisekundah.

5. Poizvedba:

public static final String ELECTRIC_UTILITY_PROVIDER

Rezultat meritve:

```
Electric Utility Provider statement started on POSTGRES database!
Electric Utility Provider statement is finished for POSTGRES database with row size of 3086 and it took 54 miliseconds!
```

Slika 51: Rezultat meritve – poizvedba 5, PostgreSQL

Vir: Lasten vir

PostgreSQL baza je izvršila poizvedbo v 54 milisekundah.

6. Poizvedba:

public static final String YEAR_MODEL_MAKE

Rezultat meritve:

```
Number of vehicles per company, per model, per model year statement started on POSTGRES database!
Number of vehicles per company, per model, per model year statement is finished for POSTGRES database with row size of 477 and it took 42 miliseconds!
```

Slika 52: Rezultat meritve – poizvedba 6, PostgreSQL

Vir: Lasten vir

PostgreSQL baza je izvršila poizvedbo v 42 milisekundah.

7. Poizvedba:

public static final String TESLA_MODEL_CITY_NUM

Rezultat meritve:

```
Number of Tesla vehicles per model per city statement started on POSTGRES database!
Number of Tesla vehicles per model per city statement is finished for POSTGRES database with row size of 1257 and it took 36 miliseconds!
```

Slika 53: Rezultat meritve – poizvedba 7, PostgreSQL

Vir: Lasten vir

PostgreSQL baza je izvršila poizvedbo v 36 milisekundah.

8. Poizvedba:

public static final String STATE_COUNTY_MAKE_MODEL_NUM

Rezultat meritve:

```
Number of vehicles by State, by county, by company and its models statement started on POSTGRES database!
Number of vehicles by State, by county, by company and its models statement is finished for POSTGRES database with row size of 2668 and it took 58 miliseconds!
```

Slika 54: Rezultat meritve – poizvedba 8, PostgreSQL

Vir: Lasten vir

PostgreSQL baza je izvršila poizvedbo v 58 milisekundah

4.3 Ugotovitve

4.3.1 Preveritev hipoteze H1

V začetku smo napolnili podatkovni bazi s podatki iz CSV datoteke. Ta je štela sto devetinštirideset tisoč vrstic. Za namene te diplomske naloge je ta datoteka simulirala našo zbirko masovnih podatkov – izsek neskončnega toka podatkov, da bi lahko primerjali zmogljivost podatkovnih baz Oracle in PostgreSQL. Vendar pa, ali lahko rečemo, da lahko s CSV datoteko dejansko simuliramo masovne podatke – neskončen vir podatkov? Iz predelane teorije vemo, da se neskončen vir podatkov nanaša na neprekinjen tok raznolikih podatkov, ki neprestano prihajajo v sistem in se obdelujejo v realnem času. Datoteke CSV so statični viri podatkov, kjer so podatki shranjeni v obliki tabele z določenim številom vrstic. Datoteke CSV ne morejo posnemati dinamike in obsega, ki ju prinašajo neskončni tokovi podatkov v okoljih masovnih podatkov.

Na podlagi tega bi lahko sprejeli sklep, da hipotezo H1: Z datotekami csv lahko simuliramo Big Data – neskončen tok podatkov **ne potrdimo**.

4.3.2 Preveritev hipoteze H2

Aplikacija je napolnila podatkovni bazi s podatki iz CSV datoteke tako, da je najprej napolnila Oracle podatkovno bazo, nato pa še PostgreSQL bazo, vrstico po vrstico, vse do zadnje vrstice v CSV datoteki. Izmerila je hitrost polnjenja podatkovnih baz. V uvodu diplomske naloge smo predpostavili, da prenos podatkov iz CSV datoteke v podatkovni bazi Oracle in PostgreSQL simulira vnos izseka neskončnega toka podatkov v podatkovni bazi, v vnaprej določenem obsegu. Na podlagi opravljenih meritev tega izseka rezultate posplošujemo na celoten vnos neskončnega toka podatkov.

Oracle podatkovna baza je potrebovala za prenos podatkov 1 uro, 47 minut in 17 sekund, PostgreSQL podatkovna baza je potrebovala 1 uro, 39 minut in 31 sekund.

Na podlagi tega bi lahko sprejeli sklep, da hipotezo H2: Hitrost vnosa neskončnega toka podatkov pri Oracle je precej večja kot pri PostgreSQL **ne potrdimo**.

4.3.3 Preveritev hipoteze H3

Glavni namen te naloge je bil ugotoviti, katera podatkovna baza, Oracle ali PostgreSQL, je zmogljivejša pri obdelavi velike količine podatkov. To smo želeli doseči z opravljenimi meritvami.

Spodaj prilagam tabelo rezultatov merjenja hitrosti izvršitve poizvedb podatkovnih baz Oracle in PostgreSQL.

Tabela 1: Rezultati merjenja hitrosti izvršitve poizvedb

Poizvdba št.	Meritve Oracle baze v milisekundah	Meritve PostgreSQL baze v milisekundah
1	1220	542
2	45	29
3	44	28
4	52	42
5	111	54
6	55	42
7	57	36
8	69	58

Vir: Lasten vir

Iz rezultatov lahko razberemo, da več podatkov kot sta morali podatkovni bazi obdelati glede na količino in njihovo raznolikost ter kompleksnejše kot so bile poizvedbe, več časa sta potrebovali za izvedbo poizvedb, pri čemer je bila baza PostgreSQL opazno zmogljivejša (zlasti pri poizvedbi 1). Na podlagi teorije, ki smo jo predelali, vemo, da na hitrost obdelave podatkov podatkovnih baz vplivajo dejavniki, kot so zmogljivost strojne opreme, konfiguracija in optimizacija podatkovnih baz, optimizacija poizvedb itd. Kot smo navedli v uvodu diplomske naloge, sta bili obe podatkovni bazi optimalno konfigurirani, enako optimizirani (ena tabela, brez uporabe indeksov), za zagon aplikacije pa smo uporabili samo en računalnik. Obe podatkovni bazi sta imeli tako enake pogoje pri merjenju hitrosti izvedbe poizvedb. Sklepamo lahko torej, da so na njune rezultate lahko vplivale samo značilnosti podatkov, ki so se obdelovali, in kompleksnost poizvedb.

Na podlagi tega bi lahko sprejeli dva sklepa:

Prvič, da hipotezo H3: Razlika v zmogljivosti med bazama podatkov PostgreSQL in Oracle pri obdelavi večjih količin podatkov v Spring Boot aplikaciji je odvisna samo od specifičnih značilnosti podatkov, ki se obdelujejo, in kompleksnosti poizvedb **potrdimo.**

Drugič, da je PostgreSQL (verzija: 14) podatkovna baza zmogljivejša pri obdelavi velikih količin podatkov v primerjavi z Oracle (izdaja: Express Edition (XE), verzija: 21c) podatkovno bazo.

4.3.4 Preveritev hipoteze H4

Na podlagi predelane teorije sklepamo, da sta lahko Oracle in PostgreSQL bazi del arhitekture za obdelavo masovnih podatkov, vendar se jih običajno ne uporablja za neposredno shranjevanje in obdelavo velikega obsega podatkov, kot se to lahko počne s specializiranimi tehnologijami za masovne podatke. Namesto tega se lahko Oracle in PostgreSQL uporabljata za nekatere specifične naloge ali kot del celotne rešitve za obdelavo masovnih podatkov. Za masovne podatke so bolj primerne naslednje podatkovne baze: a) AWS DynamoDB, b) Azure Cosmos DB, c) Amazon Redshift in d) Amazon DocumentDB. Za obdelavo neskončnega toka podatkov pa sta najboljši AWS DynamoDB in Azure Cosmos DB podatkovni bazi.

Na podlagi tega bi lahko sprejeli sklep, da hipotezo H4: Za neskončni tok podatkov obstajajo bolj primerne podatkovne baze **potrdimo.**

5 SKLEP

Java Spring Boot je ogrodje, ki razvoj spletnih aplikacij in mikrostoritev s pomočjo Spring Frameworka naredi hitrejše in enostavnejše. Spring Boot izhaja iz Spring Frameworka in je njegov podaljšek. Spring Boot poenostavlja proces ustvarjanja aplikacij, tako da odstrani veliko odvečne kode in konfiguracije, ki tradicionalno spremlja razvoj poslovnih Java aplikacij. S Spring Bootom se lahko gradijo samostojne Spring aplikacije, ki so pripravljene za takojšnje izvajanje, s potrebo po zelo malo ali nič konfiguracije (IBM, 2024). Ekosistem Spring Boot ponuja več funkcij, s katerimi je razvoj osredotočen na poslovno logiko in hiter razvoj aplikacij. Fleksibilnost in prilagodljivost ogrodja Spring pomaga ustvariti razširljive in vzdržljive sodobne aplikacije (Geeksforgeeks, 2024). Spring Boot zmanjšuje čas razvijanja aplikacij in povečuje produktivnost vseh vključenih v razvoj. Spring Boot je namenjen Java razvijalcem.

Podatkovna baza je organizirana zbirka strukturiranih informacij ali podatkov, običajno shranjenih elektronsko v računalniškem sistemu. Podatkovno bazo običajno nadzoruje sistem za upravljanje podatkovnih baz (DBMS). Podatki in DBMS, skupaj z aplikacijami, ki so z njimi povezane, tvorijo podatkovni sistem, pogosto skrajšan na podatkovna baza. Večina podatkovnih baz uporablja strukturirani poizvedovalni jezik (SQL) za pisanje in poizvedovanje podatkov, ki ga uporabljajo skoraj vse relacijske podatkovne baze (Oracle, 2024).

Poznamo več vrst podatkovnih baz (Oracle, 2024): a) relacijske baze podatkov, b) objektno usmerjene podatkovne baze, c) porazdeljene podatkovne baze, d) podatkovna skladišča, e) podatkovne baze NoSQL, f) grafične podatkovne baze, g) OLTP podatkovne baze.

Glavne funkcije sistema za upravljanje podatkovnih baz (DBMS) (Priyali, 2024): a) organiziranje podatkov, b) integracija podatkov, c) ločevanje podatkov, d) nadzor podatkov, e) pridobivanje podatkov, f) varovanje podatkov.

Oracle podatkovna baza je najbolj priljubljen in široko uporabljen relacijski sistem za upravljanje z bazami podatkov na svetu, ki ga zaupajo in uporabljajo vodilna podjetja iz različnih industrij za najbolj učinkovito shranjevanje, organiziranje in upravljanje podatkov. Oracle je znan in priljubljen zaradi svoje sposobnosti obdelave ogromnih količin podatkov (Devart, 2024).

PostgreSQL je močen, odprtokoden objektno-relacijski sistem za upravljanje z bazami podatkov, ki uporablja in razširja jezik SQL ter vsebuje številne funkcije za varno shranjevanje in razširjanje najbolj zapletenih obremenitev podatkov (Postgresql, 2024).

PostgreSQL in Oracle podatkovni bazi sta približno enakovredni glede zmogljivosti, učinkovitosti in združljivosti. Oracle vodi pri varnosti, replikaciji in razpoložljivosti, medtem ko PostgreSQL ponuja boljšo združljivost z API-ji, manj drago podporo in bolj robustno razširljivost. Izbira podatkovnih baz temelji na prioritetah podjetja (Postgresql, 2024).

Docker je odprtokodna platforma, ki razvijalcem omogoča izdelavo, uvajanje, izvajanje, posodabljanje in upravljanje kontejnerjev (IBM, 2024). Docker paketira v kotejnerje aplikacijo in vse njene odvisnosti ter lahko deluje na kateremkoli strežniku Linux (Santos, 2020). Razvijalci se pogosto poslužujejo uporabe Dockerja za namestitev in uporabo podatkovnih baz znotraj kontejnerjev. Na splošno Docker zagotavlja robusten okvir za upravljanje podatkovnih baz. Uporaba Dockerja za namestitev podatkovnih baz poenostavlja postopek, zagotavlja doslednost, izboljšuje prenosljivost in zagotavlja robusten okvir za upravljanje podatkovnih baz v sodobnih razvojnih in produkcijskih okoljih (DatabaseStar, 2023).

Masovne podatke opredeljujemo kot "masovne" ne le zaradi njihovega obsega, ampak zaradi njihove raznolikosti in kompleksnosti. Običajno presegajo zmogljivosti tradicionalnih podatkovnih baz za zajemanje, upravljanje in obdelavo. Masovni podatki lahko prihajajo od kjerkoli ali iz česarkoli na svetu, kar lahko spremljamo digitalno. Organizacije uporabljajo Big Data ali masovne podatke za sprejemanje odločitev, izboljšanje procesov in politik ter ustvarjanje izdelkov, storitev in izkušenj, osredotočenih na stranke (SAP, 2024). Uporaba masovnih podatkov za odkrivanje vpogledov lahko pomaga razumeti področja, ki vplivajo na poslovanje, od tržnih razmer in nakupnih navad strank do poslovnih procesov (GoogleCloud, 2024).

Tipi masovnih podatkov so (SAP, 2024): a) strukturirani podatki, b) nestrukturirani podatki, c) polstrukturirani podatki. Za upravljanje in obdelavo masovnih podatkov so najprimernejše nerelacijske baze podatkov. Pogosto jih imenujejo NoSQL podatkovne baze. Zasnovane so posebej za zbiranje in obdelavo različnih vrst masovnih podatkov, vključno s strukturiranimi podatki, polstrukturiranimi podatki in nestrukturiranimi podatki (Le, 2024). Med najboljšimi podatkovnimi bazami za obdelavo velikih količin podatkov so (Bakker in Kurskov, 2024): a) AWS DynamoDB, b) Azure Cosmos DB, c) Amazon Redshift, d) Amazon DocumentDB.

Na podlagi prej naštetih teoretičnih razlag smo zasnovali raziskovalna vprašanja, hipoteze, cilje in raziskovalne metode. Za pridobitev potrebnih podatkov, s katerimi bi lahko odgovorili na zastavljena raziskovalna vprašanja in potrdili oziroma ovrgli zastavljene hipoteze, smo predelali relevantno teorijo in izdelali aplikacijo z ogrodjem Spring Boot.

Pri izbiri raziskovalnih metod, smo preučili njihovo ustreznost za doseganje zastavljenih ciljev diplomskega dela. Pri izbiri tehnologij in virov, pa smo upoštevali njihovo dostopnost, stroškovni in etični vidik.

Pri razmisleku o stroških, dostopnosti sredstev in etičnih izzivih izdelave diplomskega dela in njenega izdelka je pomembno izpostaviti, da so bile uporabljene tehnologije večinoma brezplačne, kar je znatno zmanjšalo finančne obremenitve. Orodja kot so Docker, PostgreSQL, Oracle (izdaja: Express Edition (XE)) in Spring Boot, so dostopna brezplačno. Edini finančni strošek, ki je nastal, je bil za uporabo IntelliJ Ultimate različice, ki je znašal 20,62 EUR za pridobitev mesečne licence, kar je majhna naložba glede na napredne funkcionalnosti, ki jih to orodje prinaša. Dodatno smo uporabili brezplačne vire v obliki elektronskih knjig, spletnih dokumentov in spletnih strani, kar je še dodatno prispevalo k finančni učinkovitosti raziskave.

Kar se tiče dostopnosti sredstev, je pomembno izpostaviti, da so vsa uporabljena orodja, tehnologije in viri enostavno dostopni preko spleta.

V smislu etičnih izzivov lahko izpostavimo, da smo pri uporabi virov in tehnologij namenili posebno pozornost pri varovanju pravic do intelektualne lastnine, pravilni uporabi licenc in spoštovanju pogojev uporabe.

Raziskovalne metode, uporabljene v diplomskem delu, so bile skrbno izbrane za doseganje ciljev diplomskega dela. V teoretičnem delu je bil deskriptivni pristop ustrezen za predstavitev osnovnih pojmov in teoretičnih izhodišč, saj omogoča jasno in strukturirano predstavitev teorije. Metoda analize literature in virov je bila prav tako primerna, saj nam je omogočila podrobno analizo obstoječe teorije in potrditev ali zavrnitev hipotez H1 in H4.

V empiričnem delu je bila metoda zbiranja podatkov ključnega pomena, saj je omogočila pridobitev konkretnih rezultatov iz aplikacije, ki meri hitrost vnosa podatkov in izvrševanja poizvedb v podatkovnih bazah Oracle in PostgreSQL. Empirične meritve so zagotovile trdne podatke, na katerih smo lahko temeljili oceno zmogljivosti teh baz in potrdili ali zavrnili hipotezo H2. Meritve hitrosti so bile ključne za doseganje empiričnih ciljev, saj so omogočile natančno primerjavo zmogljivosti podatkovnih baz Oracle in PostgreSQL.

Logično sklepanje, ki smo ga uporabili za preverjanje hipoteze H3, je bilo prav tako primerno, saj omogoča povezovanje znanih dejstev o pripravi aplikacije z rezultati, ki smo jih pridobili. Na podlagi teh ugotovitev smo lahko potrdili ali zavrnili hipotezo H3.

V sklepnem delu je bila uporaba metode sinteze ustrezna, saj je omogočila sklepanje o celovitih ugotovitvah. Skupna uporaba teoretičnih in empiričnih metod v kombinaciji z logičnim sklepanjem in sintezo je zagotovila celovit pristop k raziskavi in omogočila dosego vseh ciljev diplomskega dela.

Alternativne metodologije, ki bi jih lahko uporabili v diplomski nalogi, vključujejo iterativni razvoj, ki je primeren za raziskave v informatiki, zlasti pri razvoju programske opreme ali sistemov. Namesto klasičnega eksperimentalnega pristopa bi lahko uporabili iterativno metodo, kjer bi se posamezni deli aplikacije razvijali in testirali v več ciklih, kar omogoča sprotno prilagajanje glede na vmesne rezultate. Prav tako bi lahko uporabili metode strojnega učenja za analizo večjih količin podatkov, kar bi omogočilo bolj poglobljeno analizo zmogljivosti podatkovnih baz. Lahko bi uporabili tudi metodo zbiranja podatkov s pomočjo anket, s katerimi bi pridobili mnenja uporabnikov preučevanih tehnologij in na podlagi njihovih izkušenj prišli do zaključkov o zmogljivosti teh tehnologij.

Tekom izvajanja raziskave smo si pridobili številne izkušnje in znanja. Raziskava je pokazala, da je natančno načrtovanje eksperimentov in pravilna izbira orodij ključnega pomena za uspeh raziskave. Uporaba brezplačnih orodij, kot so Docker, PostgreSQL, Oracle Express Edition in Spring Boot, je omogočila stroškovno učinkovito izvedbo raziskave brez večjih finančnih vložkov. Prav tako smo se naučili, da je potrebna previdnost pri obravnavi etičnih vprašanj, kot so varovanje podatkov in zagotavljanje skladnosti z licenčnimi pogoji uporabljenih orodij. Izsledki raziskave so nam dali vpogled v praktične izzive dela z različnimi podatkovnimi bazami, kjer so se pokazale razlike v zmogljivosti med Oracle in PostgreSQL pri obdelavi večjih količin podatkov. Hkrati smo pridobili izkušnje z učinkovito uporabo merilnih metod za spremljanje hitrosti vnosa podatkov in poizvedb, kar bo v prihodnjih projektih še posebej uporabno pri optimizaciji podobnih sistemov. Izkušnje, ki smo jih pridobili tekom raziskave so nas tudi naučile, da je kombinacija različnih metodologij pogosto najboljša pot do celovitih rezultatov in bolj poglobljenih analiz.

Po pridobitvi potrebnih podatkov s pomočjo izdelane aplikacije in predelane teorije smo prišli do naslednjih zaključkov: **a)** Datoteke CSV ne morejo posnemati dinamike in obsega, ki ju prinašajo neskončni tokovi podatkov v Big Data okoljih. Na podlagi tega hipotezo **H1:** Z datotekami csv lahko simuliramo Big Data – neskončen tok podatkov **nismo potrdili**. **b)** Hitrost vnosa neskončnega toka podatkov, glede na meritve izseka neskončnega toka podatkov, je pri PostgreSQL (verzija: 14) podatkovni bazi večja, kot je pri Oracle (izdaja: Express Edition (XE), verzija: 21c) podatkovni bazi. Na podlagi tega hipotezo **H2:** Hitrost vnosa neskončnega toka

podatkov pri Oracle je precej večja kot pri PostgreSQL **nismo potrdili**. **c)** Razlika v zmogljivosti med bazama podatkov PostgreSQL in Oracle pri obdelavi večjih količin podatkov v Spring Boot aplikaciji je bila v našem primeru ob danih enakih pogojih odvisna samo od specifičnih značilnosti podatkov, ki so se obdelovali, in kompleksnosti poizvedb. Na podlagi tega hipotezo **H3:** Razlika v zmogljivosti med bazama podatkov PostgreSQL in Oracle pri obdelavi večjih količin podatkov v Spring Boot aplikaciji je odvisna samo od specifičnih značilnosti podatkov, ki se obdelujejo, in kompleksnosti poizvedb **smo potrdili**. **č)** PostgreSQL (verzija: 14) podatkovna baza je zmogljivejša pri obdelavi velikih količin podatkov v primerjavi z Oracle (izdaja: Express Edition (XE), verzija: 21c) podatkovno bazo. **d)** Za obdelavo neskončnega toka podatkov obstajajo bolj primerne podatkovne baze v primerjavi z Oracle ali PostgreSQL podatkovno bazo. Na primer bazi, kot sta AWS DynamoDB in Azure Cosmos DB. Na podlagi tega hipotezo **H4:** Za neskončni tok podatkov obstajajo bolj primerne podatkovne baze **smo potrdili**.

Na podlagi naših zaključkov bi tako pravni kot fizični osebi (v nadaljevanju uporabnik), glede na različne možne scenarije, podali naslednja priporočila pri izbiri podatkovne baze za obdelavo velikih količin podatkov: **a)** V kolikor bi svetovali uporabniku, ki potrebuje podatkovno bazo za obdelavo velikih količin podatkov in ga zanimata zgolj PostgreSQL (verzija: 14) in Oracle (izdaja: Express Edition (XE), verzija: 21c) podatkovni bazi, edino merilo pri izbiri pa je hitrost obdelave podatkov, bi mu priporočili uporabo PostgreSQL podatkovne baze. **b)** V kolikor bi svetovali uporabniku, ki potrebuje podatkovno bazo za obdelavo velikih količin podatkov, in ga zanimata zgolj PostgreSQL (verzija: 14) in Oracle podatkovni bazi (izdaja: Express Edition (XE), verzija: 21c), merilo pri izbiri pa je zraven hitrosti obdelave podatkov tudi stroškovni vidik, bi mu, v kolikor nima velikih finančnih sredstev, priporočili uporabo PostgreSQL podatkovne baze. PostgreSQL se je izkazala za zmogljivejšo v primerjavi z Oracle podatkovno bazo, prav tako je brezplačna za uporabo. Obstaja tudi močna skupnost, ki nudi podporo pri reševanju težav. V kolikor pa gre za uporabnika, ki ima na razpolago veliko finančnih sredstev, bi mu kljub nekoliko slabšemu rezultatu pri meritvah priporočili uporabo Oracle podatkovne baze. Oracle podatkovna baza ponuja širši nabor funkcij kot PostgreSQL, kar omogoča nove funkcionalnosti in rešitve. Varnostne funkcije so bolj robustne kot pri PostgreSQL, poleg tega pa omogoča avtomatizacijo storitev preko oblaka Oracle. Obstaja velika skupnost, ki nudi podporo pri reševanju težav z bazo podatkov, Oracle podjetje pa nudi tudi strokovno in celodnevno tehnično podporo, neprestano posodabljanje programske opreme ter usposabljanje in izobraževanje uporabnikov. Dodatno bi uporabnika še seznanili z izdajo: Oracle Enterprise

Edition. Oracle Enterprise Edition podatkovna baza bi naj bila na podlagi predelane teorije zmogljivejša pri obdelavi velikih količin podatkov v primerjavi s PostgreSQL podatkovno bazo. **c)** V kolikor pa bi svetovali uporabniku, ki potrebuje podatkovno bazo za obdelavo velikih količin podatkov in nima nobenih preferenc, bi mu priporočili uporabo specializiranih podatkovnih baz za obdelavo velikih količin podatkov, na primer AWS DynamoDB ali Azure Cosmos DB podatkovni bazi. Te baze so zasnovane posebej za zbiranje in obdelavo različnih vrst masovnih podatkov, vključno s strukturiranimi, polstrukturiranimi in nestrukturiranimi podatki. Omogočajo tudi obdelavo podatkov v realnem času, kriptiranje podatkov ter nadzirajo dostop do baze.

6 VIRI IN LITERATURA

AdServio (18. 4. 2022). *AdServio*. Pridobljeno iz AdServio: https://www.adservio.fr/post/advantages-of-Spring Boot

Aws (18. 5 2024). *Aws*. Pridobljeno iz Aws: https://aws.amazon.com/docker/

Bekker, A. in Kurskov, D. (27. 5. 2024). *ScienceSoft*. Pridobljeno iz scnsoft: https://www.scnsoft.com/analytics/big-data/databases

DatabaseStar (10. 6. 2023). *DatabaseStar*. Pridobljeno iz DatabaseStar: https://www.databasestar.com/database-docker/

Devart (15. 5. 2024). *Devart*. Pridobljeno iz Devart: https://www.devart.com/dbforge/oracle/all-about-oracle-database/

Dragonfly (9. 6. 2024). *Dragonfly*. Pridobljeno iz Dragonfly: https://www.dragonflydb.io/faq/what-factors-affect-database-performance

Geeksforgeeks (5. 1. 2024). *Geeksforgeeks*. Pridobljeno iz Geeksforgeeks: https://www.geeksforgeeks.org/Spring Boot-ecosystem/

Geeksforgeeks (1. 5. 2024). *Geeksforgeeks*. Pridobljeno iz Geeksforgeeks: https://www.geeksforgeeks.org/introduction-to-java/

GoogleCloud (18. 5. 2024). *GoogleCloud*. Pridobljeno iz GoogleCloud: https://cloud.google.com/learn/what-is-big-data

HackerNoon (13. 5. 2024). *HackerNoon*. Pridobljeno iz HackerNoon: https://hackernoon.com/replacing-default-tomcat-server-with-jetty-or-undertow-in-Spring Boot-3-a-guide

Hashemi-Pour, C., Bigelow, S. J. in Courtemanche, M. (30. 9. 2023). *TechTarget*. Pridobljeno iz TechTarget: https://www.techtarget.com/searchitoperations/definition/Docker

IBM (13. 5. 2024). *IBM*. Pridobljeno iz IBM: https://www.ibm.com/topics/java-Spring Boot

IBM (18. 5. 2024). *IBM*. Pridobljeno iz IBM: https://www.ibm.com/topics/docker

IPSpecialistoff (29. 12. 2022). *Linkedin*. Pridobljeno iz Linkedin: https://www.linkedin.com/pulse/difference-between-oracle-postgresql-ipspecialistofficial/

JavaTPoint (14. 5. 2024). *JavaTPoint*. Pridobljeno iz JavaTPoint: https://www.javatpoint.com/java-spring-pros-and-cons

Le, V. (8. 7. 2024). *Orient*. Pridobljeno iz orientsoftware: https://www.orientsoftware.com/blog/big-data-databases/

Lutkevich, B., Gillis, A. S. in Hughes, A. (1. 2. 2023). *TechTarget*. Pridobljeno iz TechTarget: https://www.techtarget.com/searchdatamanagement/definition/database

Medium (21. 10. 2023). *Medium*. Pridobljeno iz Medium: https://medium.com/@TechiesSpot/exploring-java-Spring Boot-features-advantages-and-implementation-1a48e0bd507f

Oracle (13. 5. 2024). *Oracle*. Pridobljeno iz Oracle: https://www.oracle.com/database/what-is-database/

Oracle (13. 6. 2024). *Oracle*. Pridobljeno iz Oracle: https://docs.oracle.com/en/java/javase/11/security/java-security-overview1.html#GUID-69EE84E6-E0BD-48B2-B3F2-200D9A5FCF93

Postgresql (15. 5. 2024). *Postgresql*. Pridobljeno iz Postgresql: https://www.postgresql.org/about/

Priyali, S. (14. 5. 2024). *Yourarticlelibrary*. Pridobljeno iz Yourarticlelibrary: https://www.yourarticlelibrary.com/management/dbms/top-6-functions-of-data-base-management-system/70365

Quest (15. 5. 2024). *Quest*. Pridobljeno iz Quest: https://www.quest.com/learn/what-is-postgresql.aspx

Ravikiran, A. S. (24. 2. 2023). *SiplmiLearn*. Pridobljeno iz SiplmiLearn: https://www.simplilearn.com/tutorials/java-tutorial/java-api

Reddy, K. S. P. (2017). *Beginning Spring Boot 2: Applications and Microservices with the Spring Framework.* New York: Apress. Pridobljeno s https://books.google.si/books?hl=sl&lr=&id=9pY3DwAAQBAJ&oi=fnd&pg=PR3&dq=spring+boot+framework+&ots=b4kjdvauWr&sig=v9owvN49xG5duzJu4UjiEa_pFW0&redir_esc=y#v=onepage&q=spring%20boot%20framew

Santos, L. (9. 12. 2020). *FreeCodeCamp*. Pridobljeno iz FreeCodeCamp: https://www.freecodecamp.org/news/what-is-docker-used-for-a-docker-container-tutorial-for-beginners/

SAP (18. 5. 2024). *SAP*. Pridobljeno iz SAP: https://www.sap.com/slovenia/products/technology-platform/what-is-big-data.html

SAS (18. 5. 2024). *SAS*. Pridobljeno iz SAS: https://www.sas.com/en_us/insights/big-data/what-is-big-data.html

SolarWinds. (15. 5. 2024). *SolarWinds*. Pridobljeno iz SolarWinds: https://www.solarwinds.com/resources/it-glossary/oracle-database

Staver, A. (4. 7. 2021). *Bamboo agile*. Pridobljeno iz Bamboo agile: https://bambooagile.eu/insights/pros-and-cons-of-using-Spring Boot

Wickramasinghe, S. (7. 4. 2023). *Splunk.* Pridobljeno iz splunk.com: https://www.splunk.com/en_us/blog/learn/data-streaming.html