OPTIMIZACIJA SQL UPITA NA PRIMJERU AWS SERVISA

Diković, Jasmina

VELEUČILIŠTE U RIJECI

Jasmina Diković

OPTIMIZACIJA SQL UPITA NA PRIMJERU AWS SERVISA

Završni rad

Rijeka, 2022.

VELEUČILIŠTE U RIJECI

Poslovni odjel

Preddiplomski stručni studij Informatika

OPTIMIZACIJA SQL UPITA NA PRIMJERU AWS SERVISA

Završni rad

MENTOR

dr. sc. Ida Panev, viši predavač

STUDENT

Jasmina Diković

MBS: 2422000134/19

Rijeka, lipanj 2022.

SAŽETAK

U ovom završnom radu opisani su i prikazani na praktičnom primjeru različiti postupci koji se mogu provesti u svrhu optimizacije SQL upita, a prilikom korištenja usluga u oblaku za rad s *big data* podacima. U prvom dijelu rada prikazana su najvažnija obilježja *big data* podataka, opisan pojam računarstva u oblaku i određeni servisi koje pruža AWS – Amazon Web Services (Amazon S3, AWS Glue, Amazon Athena). Zatim su razložena obilježja važna za planiranje i provođenje upita nad velikom količinom podataka. Opisani postupci uključuju odabir formata i provedbu kompresije podataka, ali i razne druge mogućnosti značajne za rad u *cloud* okruženju, poput particioniranja podataka i planiranja strukture SQL upita. U drugom dijelu rada prikazan je praktični primjer na odabranom setu podataka, koristeći odgovarajuće Amazon Web Services usluge. Nad podacima različitih obilježja proveden je niz SQL upita kako bi se usporedile performanse u odnosu na primijenjene postupke i obilježja upita. Na temelju navedenih postupaka i prikupljenih rezultata, zaključujemo kako postoji veći broj mogućnosti putem kojih se može optimizirati provedba SQL upita u *cloud* servisima. Važnost ovih postupaka sastoji se u njihovom doprinosu učinkovitom i isplativom korištenju usluga u oblaku prilikom rada s *big data* podacima.

Ključne riječi: SQL, optimizacija, *big data*, računarstvo u oblaku, Amazon Web Services

SADRŽAJ

1. Uvod

U suvremenom poslovanju sve veći udio uspjeha organizacija ovisi o kvaliteti informacija na temelju kojih se donose odluke. Količina novih i procesiranih informacija svakim danom sve se više povećava, što znači da se postavljaju i sve veći zahtjevi na kapacitet resursa kojima se te informacije procesiraju i analiziraju. Važnu ulogu i potencijal u upravljanju podacima i njihovoj obradi ima računarstvo u oblaku, budući da korisnicima pruža resurse i mogućnosti dostupne bez nužnog posjedovanja infrastrukture za obradu velike količine podataka. Budući da je to model poslovanja koji se uobičajeno temelji na plaćanju po potrošnji, u korištenju usluga računarstva u oblaku vrlo je bitno poznavati načine kako efikasno koristiti *cloud* resurse, s obzirom na to da postoji niz postupaka kojima se mogu poboljšati performanse ovih usluga i smanjiti troškovi korištenja. Analiza podataka u bazama podataka često se provodi koristeći SQL upite, koji uključuju vrlo širok raspon mogućnosti i postupaka te značajan prostor za optimizaciju u prethodno spomenute svrhe: bolje performanse postupaka, smanjenje troškova i kraće vrijeme potrebno za provođenje analitičkih aktivnosti. Optimizacija podataka i SQL upita tako predstavlja moćno sredstvo u efikasnom i isplativom korištenju *cloud* usluga za analizu velikih količina podataka, i zato je vrijedno razmotriti na koje sve načine se analiza podataka koristeći SQL upite može optimizirati. U prvom dijelu ovog rada prikazat će se najbitnija obilježja *big data* skupova podataka, opisati pojam računarstva u oblaku i određeni *cloud* servisi koje pruža Amazon Web Services (AWS), a zatim razložiti SQL obilježja važna za *big data* analitiku te postupci kojima se provođenje upita prema bazi podataka može poboljšati. U drugom dijelu rada bit će prikazan praktični dio koji uključuje pripremu jednog seta podataka za analizu u odabranim AWS servisima, a nad kojim će se provesti različiti SQL upiti kako bi se usporedile performanse u odnosu na primijenjene postupke i obilježja upita.

2. Big data: pojam i obilježja

Pojam *big data* postaje često spominjana pojava koja poprima sve veće razmjere i utjecaj u poslovnom svijetu, ali i raznim životnim područjima. *Big data* može se definirati kao skup velikih i kompleksnih podataka, koje zbog svoje veličine ili strukture nije moguće procesirati tradicionalnim alatima i softverima za obradu podataka (Dumbill, 2012). Vrijednost *big data* podataka za organizacije može se promatrati kroz analitičku uporabu ili doprinos novim proizvodima, budući da zaključci iz velike količine podataka mogu dati uvide kojima organizacije inače ne bi imale pristup. Izvori ovakvih podataka mogu biti različiti sustavi poput društvenih mreža, web poslužitelja, senzora, uređaja, financijskog tržišta, dokumenata i raznih drugih kanala kojima se prikupljaju podaci o cijelom nizu ljudskih aktivnosti i djelovanja. Jedan od izvora koji doprinosi brzom rastu količine registriranih podataka je internet stvari (IoT, engl. *Internet of Things*), omogućavajući povezivanje fizičkih događaja i njihovog digitalnog zapisa kroz mrežu senzora različitih uređaja putem kojih se podaci prikupljaju. Procjenjuje se kako je do 2019. godine količina podataka stvorenih putem IoT mreža premašila 13 zetabajta i kako bi se do 2025. godine ta brojka mogla upeterostručiti (Statista Research Department, 2022).

Big data povezuje se s više oblika praktične primjene, poput prediktivne analitike, rudarenja podataka, predviđanja ili optimizacije podataka. Iako su ove funkcije po svojoj prirodi vrlo raznolike, zapravo im je zajedničko obilježje upotreba u svrhu donošenja poslovnih odluka koje se temelje na podacima. U odnosu na ovu svrhu, kroz vrijeme je postojala različita metodologija kojom se definira pojam *big data*, a koja se može predočiti kroz atribute dodijeljene ovakvim skupovima podataka koji se često izražavaju kroz broj „V“ svojstava. Jedna od prvih definicija, koja se naziva i Gartnerova 3V interpretacija, sagledava tri dimenzije koje određuju *big data* podatke: količinu podataka – *volume*, brzinu nastanka i korištenja podataka – *velocity* i raznolikost strukture ili formata podataka – *variety* (Buyya, Calheiros i Dastjerdi, 2016). Isti autori opisuju kako je s vremenom ova definicija proširivana na još veći broj „V“ svojstava, pa je tako primjerice IBM dodao svojstvo vjerodostojnosti (engl. *veracity*) čime je nastala 4V definicija. Postoje i definicije dodatno proširene na 5V sa svojstvom vrijednosti podataka (engl. *value*) ili 6V sa svojstvima kompleksnosti varijabli (engl. *variability*) te cjelovitosti sagledavanja podataka (engl. *visibility*), pa sve do 10V ovisno o svojstvima koja se uzimaju u definiciju.

U odnosu na prethodno navedena obilježja, rješenja putem kojih je moguće koristiti i upravljati *big data* podacima mogu se opisati kroz tri kategorije: rješenja u obliku softvera, računalnih uređaja ili rješenja u oblaku (Dumbill, 2012). Odabir rješenja ovisi o više faktora, poput lokacije podataka, privatnosti, regulativa, dostupnih ljudskih resursa ili projektnih zahtjeva, a u odnosu na navedeno moguća su i hibridna rješenja. Zbog prednosti koje pruža, sve je veći udio organizacija koje za *big data* potrebe počinju koristiti usluge računarstva u oblaku (Statista Research Department, 2022). Kako je za obradu velikih količina podataka potreban i odgovarajući računalni te skladišni kapacitet, korištenje rješenja u oblaku znači veće mogućnosti i za manje organizacije koje vlastitom infrastrukturom ili drugom vrstom kapaciteta inače ne bi mogle podržati analitičke zahtjeve *big data* podataka. S obzirom na raznovrsnost *cloud* usluga u pogledu raspona usluga, modela poslovanja, dostupnih resursa i načina plaćanja te zastupljenosti, podatak o rastućem trendu računarstva u oblaku čini se kao logičan ishod njegovog razvoja.

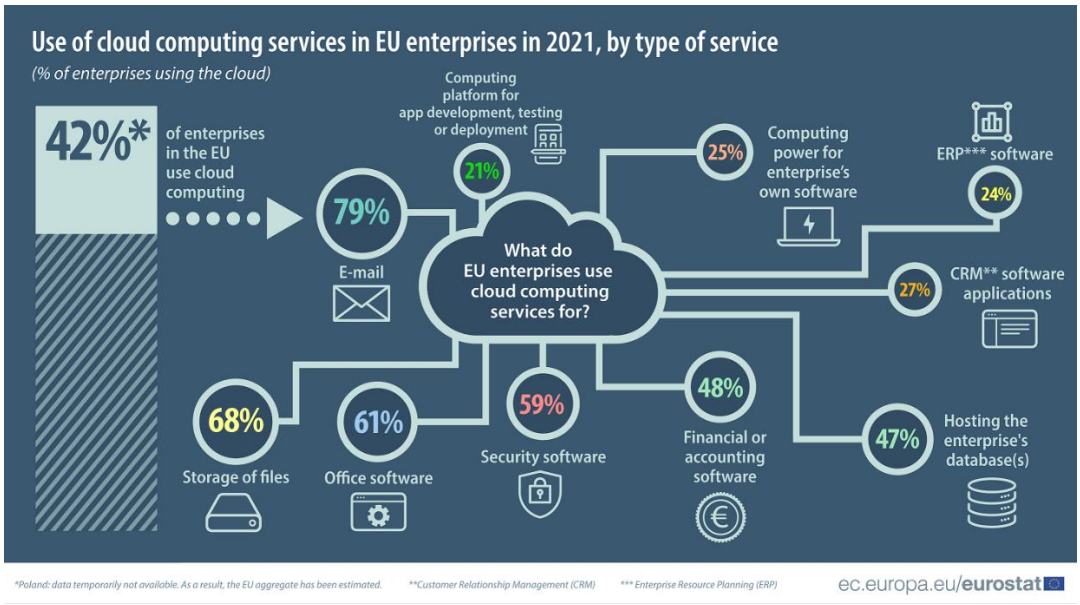
3. Računarstvo u oblaku

Računarstvo u oblaku (engl. *cloud computing*) širok je pojam koji može obuhvaćati različite usluge za različite namjene. Općenito se može definirati kao model i pristup skupu računalnih resursa u domeni informatičkih djelatnosti koji se isporučuju kao usluga koristeći internetske tehnologije (Gartner Glossary, 2022). Većina modela računarstva u oblaku ne zahtijeva posjedovanje fizičkih resursa poput hardvera i infrastrukture na strani korisnika, a usluge su korisnicima dostupne na zahtjev te se uobičajeno plaćaju prema potrošnji. Računarstvo u oblaku rastući je trend u poslovanju organizacija i upotrebi kod privatnih korisnika, što ne iznenađuje ako uzmemo u obzir njegove prednosti koje, osim spomenute činjenice da korisnici ne ovise o dostupnosti vlastite infrastrukture, uključuju i smanjene troškove održavanja sustava, fleksibilniji način rada, dostupnost velikih podatkovnih kapaciteta i snažnih računalnih resursa (Sander, 2021). Zbog širine mogućih usluga, računarstvo u oblaku nalazi svoju primjenu u različitim djelatnostima i u različite svrhe. Uz skladištenje sigurnosnih kopija ili oporavak podataka, koji već spadaju u uobičajenu namjenu, sve je veći udio *cloud* usluga koje se koriste za testiranje i razvoj aplikacija, sigurnost, upravljanje velikim setovima podataka (*big data*), procesiranje podataka i postavljanje okruženja za internet stvari te primjenu u specifičnim područjima poput umjetne inteligencije, strojnog učenja, robotike, napredne analitike i drugo (PCSC, 2018).

Budući da usluge računarstva u oblaku uključuju širok spektar mogućnosti, mogu se podijeliti prema više kriterija, a među uobičajene spadaju oblici i modeli pružanja usluga. Najčešći modeli su Iaas (engl. *Infrastructure-as-a-Service*), PaaS (engl. *Platform-as-a-Service*) i SaaS (engl. *Software-as-a-Service*), ali se zadnjih godina razvijaju i različiti novi modeli pružanja *cloud* usluga. IaaS označava model u kojemu se korisniku isporučuje infrastruktura potrebna za poslovanje u oblaku, uključujući poslužitelje, mrežu, operativne sustave i pohranu putem virtualizacije (Krmpotić, 2020). Drugi spomenuti model, PaaS, odnosi se na usluge platforme u oblaku, što korisniku omogućuje korištenje hardvera, softvera i infrastrukture u svrhu razvoja, izvođenja i upravljanja aplikacijama, bez potrebe za izgradnjom i održavanjem takve platforme na lokaciji korisnika (IBM Cloud, 2020). Model SaaS u smislu mogućnosti koje uključuje označava upravljanje svim resursima korisnika, od mreže, spremišta i servera do podataka i same aplikacije (Krmpotić, 2020), a može se najjednostavnije opisati kao usluga aplikacije u oblaku. Slika 1

prikazuje Eurostat podatke o zastupljenosti pojedinih usluga među europskim poduzećima, iz koje je vidljivo kako u organizacijama koje koriste usluge računarstva u oblaku najveći udio još uvijek čine uporaba e-mail servisa (u 79% organizacija), skladištenje podataka (68%), softveri za uredsko poslovanje (61%) i sigurnost (59%). Ipak, vidljivo je kako se pojavljuju i različita druga područja (korištenje PaaS i SaaS rješenja) kroz koja organizacije povećavaju udio uporabe *cloud* usluga.

Slika 1. Prikaz zastupljenosti usluga računarstva u oblaku u EU organizacijama



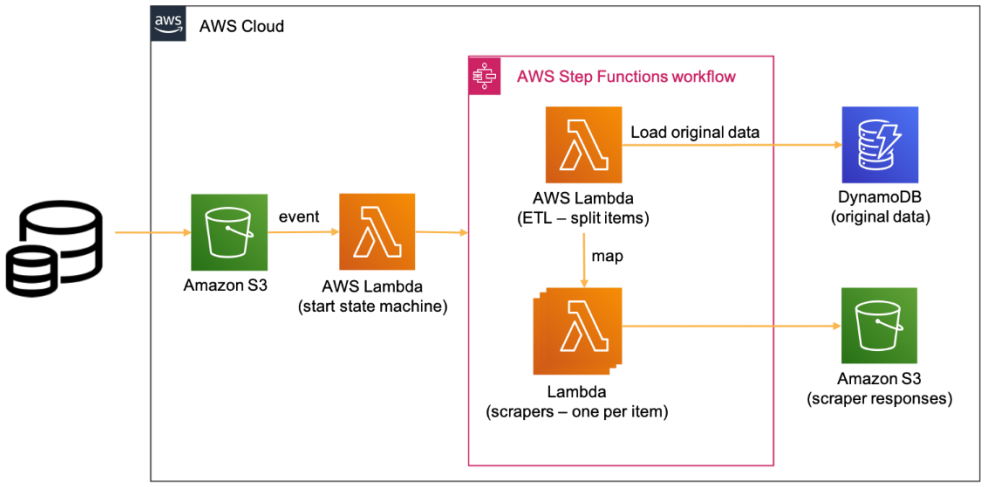
Izvor: Eurostat, 2021

Na tržištu računarstva u oblaku za poslovne korisnike, izdvajaju se dvije tvrtke koje predstavljaju većinu tržišta: AWS i Microsoft Azure. Iako su prisutni i drugi pružatelji usluga kao što su Google Cloud, Alibaba i IBM Cloud, njihovi su udjeli znatno manji u odnosu na dva vodeća konkurenta. Zastupljenost AWS-a proizlazi iz činjenice da uključuje širok raspon usluga, poput na primjer skladištenja podataka, analitičkih alata, mrežnih usluga, razvojnih alata, sigurnosti, poslovnih aplikacija i raznih drugih – AWS uključuje više od 200 pojedinačnih usluga u domeni računarstva u oblaku (Amazon Web Services, 2022 f). S druge strane, Microsoft Azure stekao je velik broj korisnika jer nudi kombinacije s uslugama poput MS Teams i Office 365 platforme, koje su popularne za organizacijske potrebe (Vailshery, 2022). S obzirom na to kako su za postupke provedene u sklopu ovog završnog rada korišteni AWS servisi, u nastavku su detaljnije razmotrena obilježja AWS pružatelja i pojedinih uporabljenih usluga.

4. Amazon Web Services

Amazon Web Services (AWS) označava platformu računarstva u oblaku koju pruža Amazon, a koja uključuje niz različitih usluga iz IaaS, PaaS i SaaS modela (Gillis, 2020). Ove usluge dostupne su kroz pojedinačne servise koji se mogu konfigurirati prema potrebama korisnika, a obuhvaćaju različite kategorije resursa: obradu i skladištenje podataka, mrežne kapacitete, sigurnost, *big data* analitiku, umjetnu inteligenciju i razne druge. Amazon kroz AWS trenutno posluje u 190 zemalja diljem svijeta i koristi globalnu infrastrukturu koja se dalje razvija s rastućim brojem klijenata. Infrastruktura koju koristi organizirana je na temelju tzv. regija i zona dostupnosti (engl. *Regions and Availability Zones*), a trenutno se koristi više od 80 zona dostupnosti i 25 geografskih regija u svijetu. Regija predstavlja fizičku lokaciju u svijetu s više mogućih zona dostupnosti, dok se zona dostupnosti sastoji od jednog ili više podatkovnih centara koji pružaju određeni skup resursa. Infrastruktura u regijama organizirana je na način da su regije međusobno nezavisne, čime se umanjuje rizik nedostupnosti i nestabilnosti usluge (Amazon Web Services, 2022 a). Servisi putem kojih su dostupni resursi mogu se, ovisno o namjeni, koristiti kao zasebne usluge ili koristiti kao niz usluga povezanih jedinstvenom arhitekturom poput primjera prikazanog na slici 2.

Slika 2. Primjer AWS arhitekture za prikupljanje podataka u rudarenju podataka



Izvor: Rotem, 2021

U ovom primjeru arhitektura počinje sa S3 servisom. Pohrana izvornog objekta označava događaj koji priziva prvu Lambda funkciju, a koja koristi originalni set iz S3 kao ulazne podatke. Slijedi Step funkcija koja orkestrira proces pripreme podataka kroz dvije Lambda funkcije, od kojih svaka provodi zadane operacije nad podacima i sprema ih na odredište (DynamoDB ili S3). Povezivanjem ovih usluga u arhitekturu postiže se odvijanje niza aktivnosti kroz različite servise koji služe za prikupljanje podataka i obradu potrebnu prije daljnje uporabe.

Pojedinačni servisi mogu se integrirati na raznolike načine ovisno o ciljevima koje korisnik želi postići, a neki od servisa su primjerice (Amazon Web Services, 2022 f):

Analitika: Amazon Athena, EMR, Kinesis, Redshift, Data Pipeline, Glue, Lake Formation;

Razvoj aplikacija i računarstvo: Amazon EC2, App Runner, Elastic Beanstalk, Lambda;

Baze podataka: Amazon Aurora, RDS, Redshift, DynamoDB;

Razvojni alati: Amazon CodeBuild, CodeCommit, CodeDeploy, CodePipeline, Cloud Development Kit (CDK);

Skladištenje i pohrana podataka: Amazon S3 (Simple Storage Service), Backup, Elastic Disaster Recovery;

Sigurnost: AWS Identity and Access Management (IAM), Cognito, Single Sign-On.

4.1. Amazon S3 (Simple Storage Service)

Servis Amazon S3 omogućava pohranu objekata koji se mogu koristiti za različite namjene, poput stvaranja podatkovnog jezera, pohrane podataka za web ili mobilne aplikacije, sigurnosne kopije podataka, oporavak od katastrofe, IoT uređaje ili *big data* analitiku (Amazon Web Services, 2022 b). Podatke koji se nalaze u S3 moguće je prema potrebi organizirati, optimizirati i konfigurirati im postavke pristupa ovisno o zahtjevima korisnika. Unutar samog servisa moguće je razlikovati nekoliko klasa pohrane, ovisno o namjeni podataka. Primjerice, za podatke koji se ne koriste učestalo, moguće je koristiti S3 Glacier kao jednu mogućnost koja smanjuje troškove. U upravljanju podacima moguće je na primjer definirati vrijeme zadržavanja podataka do brisanja (postaviti *lifecycle policy*), zaštititi podatke od izmjene i brisanja ili nad njima provoditi grupne akcije (engl. *batch operations*) poput kopiranja ili oporavka velikih skupina objekata. Podaci se skladište na način da se organiziraju u spremnike (engl. *bucket*) kao objekti, pri čemu objekt može

biti bilo kakva datoteka (tekstualna datoteka, slika, video, itd.) ili metapodaci o datotekama. Objekte je moguće premještati, otvarati i preuzimati. Svaki objekt unutar spremnika mora imati svoj jedinstven naziv koji služi kao ključ, odnosno jedinstveni identifikator objekta u spremniku. Ažuriranje podataka na pojedinom ključu poštuje svojstvo atomičnosti: primjerice, ako su objektu istovremeno upućeni PUT i GET zahtjev[1] različitih dretvi, podaci koji se vrate kao odgovor neće biti parcijalni ili narušeni, već se vraća ili stari set podataka ili novo ažurirano stanje.

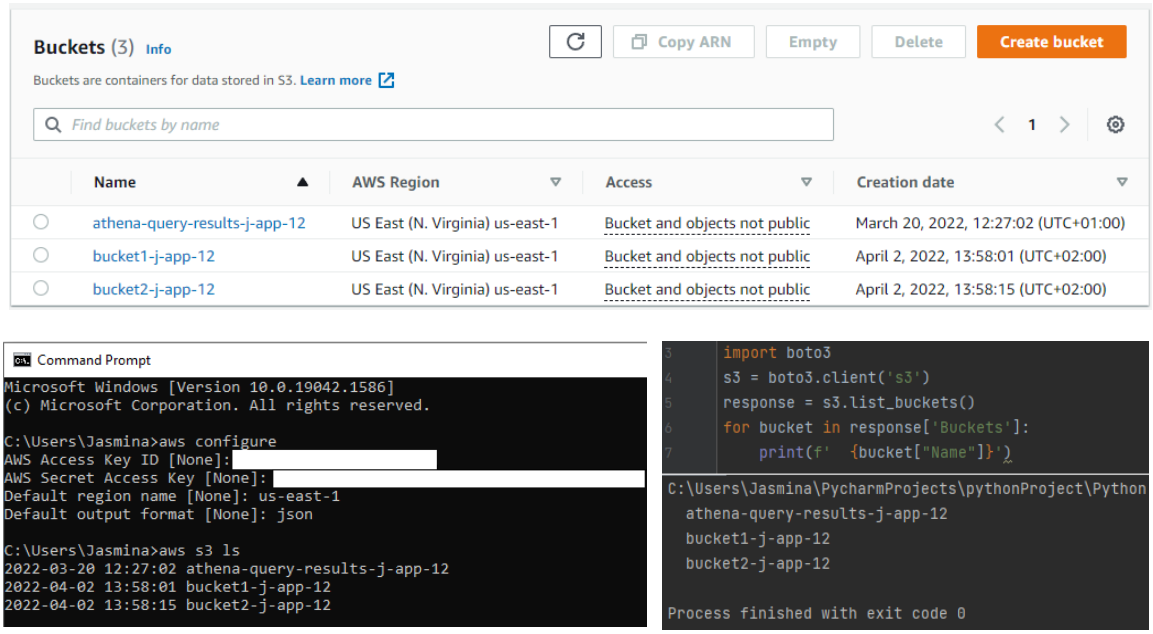
Podatke je u S3 moguće unijeti učitavanjem s lokalnog računala (putem konzole ako su datoteke veličine do 160 GB, u suprotnom koristeći AWS CLI, SDK ili S3 REST API[2]), ili putem drugih servisa koji nude takve mogućnosti. Bitno je spomenuti kako postoje i određena ograničenja po pitanju veličine objekta: maksimalna veličina pojedinog objekta je 5 TB, dok je gornja granica za veličinu objekta u jednom PUT zahtjevu 5 GB. Nakon učitavanja podataka u spremnik(e), podatke je moguće koristiti s drugim AWS servisima. Spremnicima i objektima koji se nalaze u S3 moguće je pristupiti na više načina: putem konzole (web sučelja servisa), sučelja komandne linije (CLI) ili razvojnih alata (SDK). Na slici 3 prikazani su ovi načini: u gornjem dijelu slike vidljiv je pregled S3 spremnika (njih ukupno tri) u AWS konzoli. U donjem dijelu slike (lijevo) vidljiv je pregled dobiven koristeći AWS CLI i naredbu za dobivanje popisa spremnika. Desno od toga na istoj slici nalazi se popis svih spremnika dobiven pomoću AWS SDK za Python (Boto 3) koristeći PyCharm[3]. Kod kreiranja spremnika i učitavanja objekata, važno je posvetiti određeno vrijeme planiranju organizacije i veličine objekata. Jedan od razloga za to je cjenik za S3 koji uključuje količinu skladištenih podataka, ali i akcije prijenosa objekata u spremnik i izvan njega. Broj i veličina objekata također mogu utjecati na performanse drugih servisa koji koriste podatke iz S3 spremnika.

[1] PUT je zahtjev kojime se mogu zapisati novi podaci ili preko postojećih podataka prepisati nova verzija (zahtijeva *write* dozvolu), dok se GET zahtjevi koriste za dohvaćanje podataka u nekoj aplikaciji ili servisu (zahtijevaju *read* dozvolu) (Amazon Web Services, 2022 b)

[2] AWS CLI (*Command Line Interface)* je alat koji omogućava interakciju s AWS servisima koristeći naredbe kroz sučelje komandne linije. AWS SDK (*Software Development Kit*) uključuje skup programskih alata, biblioteka i dijelova koda za različite programske jezike, putem kojih korisnik može programski pristupati servisima. S3 REST API (*Application Programming Interface*) odnosi se na HTTP sučelje prema S3, a omogućava kreiranje, dohvaćanje ili brisanje S3 spremnika i objekata (Amazon Web Services, 2022 b).

[3] PyCharm je integrirano razvojno okruženje (IDE - *Integrated Development Environment*) namijenjeno za Python programski jezik.

Slika 3. Različiti načini pristupa S3 servisu – AWS konzola, CLI, SDK



Izvor: Autor, travanj 2022.

4.2. Amazon Athena

Amazon Athena je interaktivni servis za postavljanje SQL upita putem kojega se podaci mogu analizirati izravno iz Amazon S3 servisa (Amazon Web Services, 2022 c). Putem konzole, moguće je unijeti postavke lokacije podataka u S3 i koristeći standardni SQL provoditi upite nad podacima. Athena je tzv. bezposlužiteljski (engl. *serverless*) servis, što znači da za njezino korištenje nije potrebno postavljati ili održavati nikakvu infrastrukturu. Budući da ima svojstvo automatskog skaliranja, moguće je provoditi više paralelnih upita istovremeno, a korištenje se naplaćuje prema ukupnom broju upita i količini skeniranih podataka. Prilikom razmatranja ovakvog servisa korisno je imati na umu kako postoje različiti servisi koji služe za rad s podacima (npr. Amazon Redshift kao skladište podataka ili Amazon EMR za procesiranje podataka), a svaki od njih namijenjen je primjeni za različite potrebe i slučajeve korištenja. Athena je primijenjiva za analizu nestrukturanih, polustrukturiranih ili strukturiranih podataka pohranjenih u S3. Ti podaci mogu biti u tekstualnom formatu, primjerice CSV ili JSON, ili u kolumnarnom formatu kao što su Apache Parquet ili ORC. Podržava rad s podacima koji su komprimirani u formatu Snappy, ZLib, LZO i

GZIP (Amazon Web Services, 2022 d). U slučajevima kada je potrebna vizualizacija podataka, Athena se može integrirati s drugim servisom, Amazon QuickSight koji to omogućava. Athena se također može koristiti za generiranje izvještaja, primjenu alata poslovne inteligencije ili SQL klijente koji se povezuju koristeći JDBC ili ODBC driver[4]. Jedan od servisa s kojim se može integrirati je i AWS Glue (Data Catalog), koji omogućava rad s metapodacima S3 podataka (poput sheme i tipova podataka). Pomoću njih, moguće je kreirati tablice i postavljati SQL upite u Atheni na temelju središnjeg kataloga metapodataka, a koji se dodatno mogu koristiti s integriranim mogućnostima AWS Glue servisa za analizu podataka i ETL (engl. *extract-transform-load*)[5].

Athena je u pogledu zahtjeva jednostavan servis, budući da je dovoljno definirati tablicu koja sadrži metapodatke i time su oni spremni za postavljanje SQL upita. Tablice i baze podataka prikazane u Atheni zapravo su spremnici definicija metapodataka koji sačinjavaju shemu izvornih podataka. Svaki set podataka treba u Atheni postojati kao tablica, a metapodaci u tablici usmjeravaju Athenu na S3 lokaciju gdje se podaci nalaze i određuju strukturu podataka (npr. nazivi stupaca, tipovi podataka, ime tablice). Baza podataka u Atheni predstavlja skupinu logički grupiranih tablica i također sadrži samo metapodatke i shemu određenog seta podataka. Kad se tablice ili baze kreiraju ručno Athena koristi HiveQL jezik za definiranje podataka (DDL[6]), dok se kod upita nad već kreiranom tablicom u pozadini koristi Presto i ANSI SQL. Tablice se mogu stvoriti ručno ili automatski, a u oba slučaja one se kreiraju kroz AWS Glue Data Catalog (Amazon Web Services, 2022 c).

[4] JDBC (*Java Database Connectivity*) i ODBC (*Open Database Connectivity)* općenito predstavljaju softverske komponente koje omogućavaju povezivanje aplikacija s bazom podataka. Koriste se na način da ih korisnik preuzima, instalira i postavlja konfiguraciju, a kod AWS servisa postoji i mogućnost korištenja JDBC sa ili bez SDK (Amazon Web Services, 2022 c).

[5] ETL, skraćenica od *extract-transform-load*, u prijevodu označava ekstrakciju, transformaciju i učitavanje podataka. ETL sustav dohvaća podatke iz izvorišnih sustava, primijenjuje standarde za kvalitetu i konzistentnost podataka, prilagođava podatke na način da se različiti izvori podataka mogu zajednički koristiti, a sve s ciljem dostavljanja podataka u formatu namijenjenom upotrebi od strane krajnjih korisnika (Vukelić, 2014).

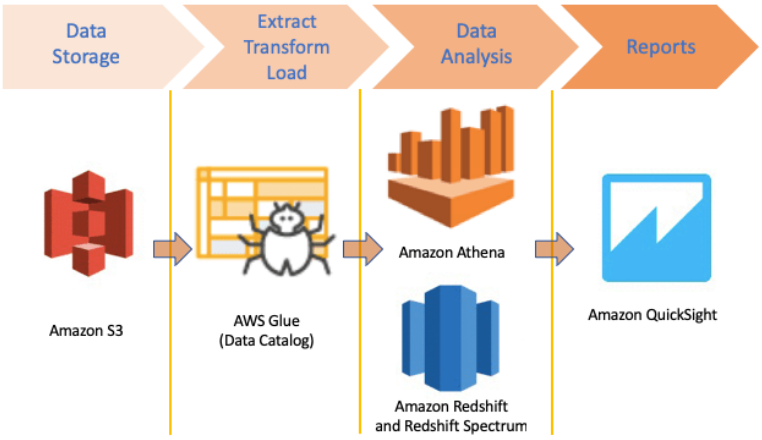
[6] DDL (*Data Definition Language*) je skupina SQL pristupnog jezika čija sintaksa služi za stvaranje i izmjenu elemenata baze podataka i njezinih objekata (npr. tablica), kroz naredbe CREATE, ALTER, DROP (Leskovar, 2014).

4.3. AWS Glue

AWS Glue je servis koji nudi mogućnost potpunog upravljanja ETL procesima, a putem kojega se podaci mogu kategorizirati, čistiti, izmijeniti ili kretati između različitih skladišta i tokova podataka (Amazon Web Services, 2022 e).

Ovaj servis je isto kao i ranije opisani AWS servisi bezposlužiteljski (engl. *serverless*), što znači da korisnik ne treba postavljati i održavati infrastrukturu za njegovo korištenje. Primarno je namijenjen za rad s polustrukturiranim podacima, a služi za otkrivanje podataka, transformaciju i pripremu podataka za pretraživanje i postavljanje upita, kroz njihovu organizaciju, čišćenje, validiranje i formatiranje u skladištu ili jezeru podataka (Amazon Web Services, 2022 e). Jedna od brojnih mogućnosti koje nudi u radu s drugim servisima je uporaba u slučaju kada se podaci nalaze u S3 servisu, budući da može napraviti „katalog" S3 podataka i time ih učiniti dostupnima za postavljanje upita kroz druge servise (npr. Amazon Athena ili Redshift). AWS Glue servis zasniva se na interakciji većeg broja komponenti pomoću kojih se provode koraci ETL procesa. Jedan od važnijih je ranije spomenuti središnji repozitorij metapodataka, AWS Glue Data Catalog, dok *job* predstavlja poslovnu logiku potrebnu za provođenje ETL akcija. Kroz AWS Glue, korisnik može upravljati *job* resursima koji koriste Data Catalog definiciju tablica, a sastoje se od programskih skripti koje izvode transformaciju podataka. Ove skripte koriste Python ili Scala programski jezik. *Job* se može pokrenuti ručno ili postavljanjem okidača (engl. *trigger*) koji se pokreću po određenom rasporedu ili kao rezultat zadanog događaja u sustavu. *Crawler* je komponenta koja se može predočiti kao program kojime se uspostavlja veza sa spremištem podataka (izvornim ili odredišnim podacima), a koji služi za utvrđivanje sheme podataka i stvaranje tablice metapodataka u AWS Glue Catalogu. U slikovnim i shematskim prikazima AWS arhitekture *crawler* se najčešće prikazuje simbolom pauka, kao što je to slučaj na slici 4. Ova slika prikazuje jedan primjer primjene Glue servisa u ETL procesu. Koristeći izvorne podatke koji se nalaze u S3, *crawler* bilježi shemu podataka u Data Catalogu. Podaci se mogu pretvoriti u drugi format koristeći Glue *job*, koje *crawler* zatim može učiniti dostupnima u S3. Podaci su na taj način spremni za fazu analize, u kojoj se podacima pristupa kroz odgovarajuće servise (kao što su prikazani Athena ili Redshift), ovisno o potrebi i ciljevima.

Slika 4. Prikaz primjene ETL procesa koristeći AWS Glue



Izvor: Chandrabose, 2020

U Glue sustavu, skup povezanih resursa koji čine tijek procesa može se predočiti kroz tzv. *workflow*, koji upravlja provođenjem i nadzorom povezanih *job* i *crawler* komponenti (Amazon Web Services, 2022 e). Nakon što se nad podacima provedu potrebne akcije u AWS Glue servisu, podaci su spremni za korisnika koji ih može dohvatiti i analizirati u drugim servisima, najčešće koristeći SQL upite i njihove mogućnosti.

5. SQL upitni jezik: obilježja, mogućnosti i optimizacija upita

SQL (engl. *Structured Query Language*) do danas je postao ključni jezik za rad s bazama podataka, a predstavlja strukturirani upitni jezik, odnosno jezik za postavljanje upita nad bazama podataka (Mujadžević, 2016). Primijenjuje se kroz veći broj sustava za upravljanje bazama podataka, kao što su primjerice Microsoft SQL Server, MySQL, PostgreSQL ili Oracle. Iako postoje standardi za SQL jezik (npr. ANSI, *American National Standards Institute* ili ISO, *International Organization for Standardization*), svaki sustav može imati različite varijante i nadogradnje osnovnog SQL jezika. Zadnja verzija standarda u trenutku pisanja ovog rada je SQL:2016 (puni naziv ISO/IEC 9075:2016, Information technology — Database languages — SQL), a objavljen je krajem 2016. godine (Michels i sur., 2018). Osnovno korištenje SQL-a može se predočiti kroz naredbe pristupnog jezika za čitanje i pisanje podataka u bazi, a koje se mogu svrstati u nekoliko najznačajnijih kategorija (Leskovar, 2014):

DDL (engl. *Data Definition Language*) – naredbe za rad s bazom i objektima: CREATE, ALTER, DROP;
DML (engl. *Data Manipulation Language*) – naredbe za upravljanje podacima: INSERT, UPDATE, DELETE;
DQL (engl. *Data Query Language*) – naredbe za postavljanje upita: SELECT.
Osim navedenih, u literaturi je moguće sresti i proširene kategorije jezika, koje uključuju primjerice DD (engl. *Data Dictionary*), DCL (engl. *Data Control Language*) ili DTL (engl. *Data Transfer Language*). U korištenju upita za izvršavanje željenih zadataka, a posebice kod korištenja DQL i DML naredbi, važnu ulogu osim naredbi imaju i SQL funkcije, budući da omogućavaju širi raspon operacija nad podacima.

5.1. SQL naredbe, funkcije i izrazi

S obzirom na povijest SQL-a, njegovu učestalu primjenu, popularnost i razvoj, pregled svih njegovih obilježja i mogućnosti bio bi zahtjevan zadatak, budući da bi takav pregled bio izrazito opsežan. Zbog toga, za potrebe ovog rada, naglasak će se staviti na pojedine skupine naredbi i funkcija koje će se primijeniti u drugom dijelu rada ili koje imaju određeni značaj u SQL primjeni na *big data* podacima:

1) Upiti. Za dohvaćanje podataka iz jedne ili više tablica, SQL koristi niz klauzula koje se postavljaju određenim redoslijedom kako bi formirale valjan upit. Općenito se redoslijed osnovnih klauzula može prikazati na sljedeći način: SELECT – FROM – WHERE – GROUP BY – HAVING – ORDER BY.
Naredbom SELECT dohvaća se jedan ili više stupaca, FROM označava tablicu iz koje se podaci uzimaju, dok WHERE služi za postavljanje uvjeta kojim se dohvaća samo dio podataka. Ovaj osnovni slijed može se nadopuniti različitim naredbama, među kojima su češće korištene GROUP BY za grupiranje podataka po nekom stupcu, HAVING za uključivanje grupiranih podataka koji zadovoljavaju neki uvjet i ORDER BY za sortiranje podataka željenim redoslijedom (Horvatinović, 2020).

2) Funkcije nad retcima (*scalar/single row functions*). Ova skupina funkcija obuhvaća postupke koji se provode na razini retka tablice i vraćaju rezultat za svaki redak, a najčešće skupine su (Oracle, 2017):

a) numeričke funkcije poput POWER(), ROUND(), ABS();
b) funkcije za rad s tekstom (*string* funkcije) poput UPPER(), LOWER(), LENGTH(), TRIM();
c) datumske i vremenske funkcije poput NOW(), DATE().

3) Agregatne funkcije. Ova skupina funkcija služi za dobivanje jednog podatka na temelju više redaka, odnosno za agregiranje i vraća samo jednu vrijednost stupca nad kojim se primijenjuje (Mujadžević, 2016). Primjeri ovih funkcija su dobivanje sume nekog stupca – SUM(), broja vrijednosti u stupcu ili tablici – COUNT(), računanje prosjeka – AVG(), najmanje vrijednosti – MIN() ili najveće vrijednosti – MAX().

4) Funkcije za rad s podacima u prozoru (*window* funkcije). Ova šira skupina predstavlja napredne funkcije za upite koji koriste rangiranje, ugniježđene, kumulativne i „pomične" agregacije u prozoru, koji se može opisati kao tranzijentni skup n-torki pomoću kojeg se definiraju particija ili okvir (FER, 2018). Particija predstavlja fiksnu komponentu koja može sadržavati okvire, a definira se pomoću PARTITION BY u OVER() dijelu naredbe. U slučaju kada se ne definira, cijela relacija je zapravo jedna particija. Okvir se također definira u OVER () dijelu, ali se može definirati navođenjem različitih naredbi: ORDER BY, ROWS, RANGE. Na slici 5 prikazan je primjer korištenja particije i okvira: upit koji se nalazi lijevo stvara najprije particiju `sifPred`, čime particiji pripadaju sve n-torke s jednakom vrijednosti `sifPred`. Unutar svake od njih stvara se nova particija, `datumRok`, kojoj zatim pripadaju sve n-torke s jednakom vrijednosti `datumRok`. Desno na slici 5 nalazi se primjer u kojemu se okvir definira pomoću ORDER BY naredbe, a u kojemu sve n-torke s jednakom `sifPred` pripadaju istom okviru (i particiji jer je u ovom slučaju sve jedna particija).

Slika 5. Primjer particije i okvira u funkcijama za rad s podacima u prozoru

```
SELECT sifPred
     , sifStud
     , ocjena
     , AVG(ocjena) OVER (PARTITION BY sifPred, datumRok)
  FROM ispit
```

```
SELECT sifPred, sifStud, datumRok
     , ocjena
     , AVG(ocjena) OVER (ORDER BY sifPred)
  FROM ispit
ORDER BY sifPred, datumRok, sifStud;
```

Izvor: FER, 2018

U slučaju kada se okvir definira pomoću ROWS ili RANGE, granice okvira definiraju se pomoću dodatnih naredbi. Kod ROWS, to su različite kombinacije BETWEEN (UNBOUNDED) PRECEDING, (UNBOUNDED) FOLLOWING i CURRENT ROW, kojima se određuje koje n-torke se uzimaju u okvir.

Pojednostavljeno, prozor se može predočiti kao podskup redaka u tablici koji se sagledavaju kao određena cjelina, particija djeluje kao svojevrsno grupiranje, a okvir kao način prikaza redaka u grupi.

Window funkcije mogu se svrstati u podkategorije, iako u literaturi postoji više podjela tako da se ovisno o autoru mogu sresti u različitim podjelama koje se preklapaju. U slučajevima kada je nad podacima potrebno provesti postupke rangiranja, postoje četiri različite funkcije: ROW_NUMBER(), RANK(), DENSE_RANK() i NTILE(). U sintaksi pojedinih od ovih funkcija mogu se primijeniti i argumenti PARTITION BY, ORDER BY ili number_of_groups, ovisno o tome na koji način se rangiranje želi provesti (nad cijelim setom, po određenim grupama, redoslijedu). Prve tri funkcije su relativno slične, ali se razlikuju po tome na koji način se dodjeljuje rang (bez ponavljanja ranga kod istih vrijednosti, dijeljenjem ranga, od koje vrijednosti se nastavljaju rangovi), dok zadnja funkcija NTILE () služi za svrstavanje redaka u željeni broj grupa kojima se dodjeljuje rang prema odabranom kriteriju (Nguyen, 2022). Druga podskupina *window* funkcija može se izdvojiti kao analitičke funkcije, u kojoj se primjerice nalaze LEAD(), LAG(), FIRST_VALUE() i LAST_VALUE(). Ove funkcije služe za određivanje vrijednosti u retku temeljem pozicije retka (Kaggle, 2022), pri čemu su LAG() i LEAD() funkcije koje vraćaju vrijednost atributa koja prethodi ili slijedi nakon tekuće vrijednosti u particiji, dok FIRST_VALUE() i LAST_VALUE() vraćaju prvu ili zadnju vrijednost atributa u okviru.

S obzirom na to kako u radu s velikom količinom podataka upiti mogu postati kompleksni, jedan od načina kako se mogu pojednostaviti i učiniti čitljivijima je korištenje CTE (engl. *Common Table Expressions*) izraza. CTE je „blok“ u upitu koji daje privremeni set rezultata, postoji samo kao dio većeg upita, a njegovi rezultati ne spremaju se kao zasebna vrijednost već se koriste u tijeku istog upita. U tijeku jednog upita može se koristiti veći broj CTE izraza, koji započinje izrazom WITH i ima sljedeću strukturu (Welch, 2021):

```
WITH expression_name_1 AS
(CTE query definition 1)
SELECT expression_A, expression_B, ...
FROM expression_name_1;
```

Uz prethodno navedene funkcije i izraze koji predstavljaju samo dio mogućnosti, u pripremi za korištenje SQL-a može se primijeniti cijeli niz postupaka za poboljšanje učinkovitosti. Pojam optimizacije upita ili poboljšanja performansi široka je tema, zato što obuhvaća različita područja putem kojih se postupci optimizacije mogu provoditi, sve od hardvera, mrežnih postavki, memorije, alata, okruženja pa do samih podataka i primjene postupaka na razini konkretnih upita. S obzirom na ciljeve ovog rada, naglasak u nastavku je na podacima i upitima.

5.2. Postupci u optimizaciji podataka i SQL upita

Rad s *big data* podacima donosi određene izazove na razini ulaza i izlaza pohranjenih podataka iz više razloga: osim same količine podataka, prikupljanje podataka iz mnoštva izvora također znači kako podaci ne moraju biti strukturirani, već mogu biti polustrukturirani ili nestrukturirani i pristizati u različitim formatima. Vrijeme potrebno za skeniranje podataka također je važan faktor, a izravno je povezano s veličinom seta podataka koji se skenira u upitima i načinom na koji su dohvaćeni podaci organizirani u spremištima. U svrhu pripreme podataka i upita za što efikasniju uporabu u fazi analize, razmotren je skup postupaka kojima se podaci mogu transformirati i postupaka kojima se upiti mogu unaprijediti.

5.2.1. Format podataka

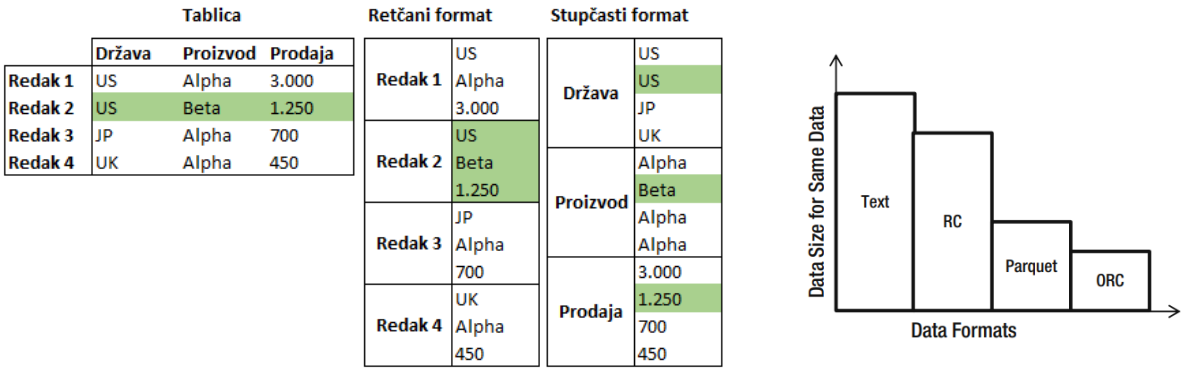
Za odabir formata podataka moguće je voditi se različitim ciljevima koji se žele postići i obilježjima radnih zahtjeva. Oni primjerice mogu uključivati dostupno okruženje (i poznavanje podržanih formata), predviđanje hoće li se struktura podataka mijenjati s vremenom, očekivanu količinu podataka, broj i vrstu zahtjeva za čitanje/pisanje podataka ili potrebu za izvozom baze u drugo okruženje (Pal, 2016). Odabir optimalnog formata može izravno utjecati na performanse pri postavljanju upita, budući da format može odrediti veličinu seta podataka, odnosno broj zahtjeva za čitanjem/pisanjem i količinu podataka koje je potrebno procesirati. Svaki format ima svoja obilježja u smislu prednosti i nedostataka, tako da je za odabir bitno razmotriti svrhu koju format treba postići. Jedna od ključnih podjela formata podataka razlikuje pohranu u retcima (engl. *row data storage*) i pohranu u stupcima (engl. *columnar data storage*), a primjer usporedbe prikazan je na slici 6 (lijevo). Vidljivo je kako pohrana u retcima predstavlja „tradicionalan“ način skladištenja,

u kojemu se podaci pohranjuju slijedno kao niz slogova s podacima retka (redak 1, 2, 3, 4). Za razliku od toga, u stupčastom formatu pohranjuju se podaci u stupcima (u ovom primjeru tri stupca: država, proizvod, prodaja) na susjednim memorijskim lokacijama. U smislu primjene, stupčasti formati primijenjiviji su kod zahtjevnijih analitičkih upita za čitanjem podataka, dok su retčani formati bolji odabir za česte transakcijske upite koji trebaju zapisati podatke (Woodie, 2018).

Najčešći *big data* formati koji spremaju podatke retčano su CSV (engl. *Comma-Separated Values*), JSON (engl. *JavaScript Object Notation*) i AVRO. CSV i JSON su tekstualni formati čitljivi čovjeku, ali s druge strane imaju određena ograničenja (npr. značajnije izmjene sheme, mogućnosti kompresije) i općenito mogu biti manje efikasni za skladištenje podataka (Pal, 2016). AVRO je format otvorenog koda namijenjen za Apache Hadoop, softversku knjižnicu i okvir za skladištenje i procesiranje velikih setova podataka (IBM, 2022). Primijenjiv je za razmjenu *big data* podataka između programa u različitim programskim jezicima i obilježava ga kompaktno i efikasno skladištenje podataka. Ova efikasnost proizlazi iz skladištenja podataka u binarnom formatu, dok definiciju podataka pohranjuje u JSON formatu koji se jednostavno čita i interpretira. Ključna prednost mu je dobra podrška u slučaju izmjene sheme podataka tijekom vremena, tzv. „evolucije sheme“ (IBM, 2022), a omogućava preimenovanje, dodavanje, brisanje i izmjenu tipova podataka definiranjem nove sheme (Pal, 2016). Osim dosad navedenih, među češćim *big data* formatima nalaze se Parquet i ORC (engl. *Optimized Row Columnar*). Iako dijele neka obilježja koja ima i AVRO (podaci su strojno čitljivi, skladištenje skalabilno, procesiranje upita može biti paralelno, uključuju definiciju sheme), najveća razlika proizlazi iz načina kako pohranjuju podatke (Woodie, 2018). Parquet i ORC su stupčasti formati, koji zbog takvog načina pohrane smanjuju opseg zahtjeva i količinu podataka koja se učitava s diska. Ovo obilježje prikazano je i na slici 6 (desno), iz koje je primjetno kako se najveća veličina seta podataka pojavljuje u tekstualnim retčanim formatima, dok se korištenjem stupčastih formata veličina podataka smanjuje. Oba stupčasta formata dijele obilježja poput dobrih mogućnosti kompresije i efikasnog izvođenja upita, dok se razlikuju prema načinu strukturiranja podataka i prilagođenosti za određene tehnologije, npr. Hive ili Spark sustav (Pal, 2016). Na temelju navedenog, vidljivo je kako prilikom odabira formata podataka treba razmotriti veći broj faktora. Ako se podaci izvoze za učitavanje u drugu bazu, CSV bi mogao biti prihvatljiv format. AVRO je bolji izbor ako će se shema mijenjati s vremenom, iako će performanse upita biti slabije nego ako se koristi Parquet ili ORC. S druge

strane, kreiranje dokumenata u stupčastom formatu može trajati duže i dokumenti se ne mogu jednostavno ažurirati, tako da odabir formata svakako treba uskladiti s namjenom samih podataka.

Slika 6. Način pohrane podataka u retčanom i stupčastom formatu (lijevo) i odnos veličine seta podataka i različitih formata podataka (desno)



Izvor: prilagođeno prema Getz, 2014; Pal, 2016

5.2.2. Kompresija podataka

Kompresija podataka još je jedna tehnika kojom je moguće smanjiti zahtjeve na ulaz i izlaz podataka, a obuhvaća različite algoritme. Postupak kompresije podataka može se definirati kao proces prilikom kojega se smanjuje fizički prostor potreban za pohranu podataka, korištenjem određenih metoda za zapis podataka (Farena, 2020). Prilikom korištenja resursa kao što je to slučaj kod računarstva u oblaku, kod kompresije je poželjno da podržava mogućnost dijeljenih datoteka (engl. *splittable files*). Ova mogućnost znači da se dijelovi podataka mogu procesirati na različitim računalnim čvorovima koji su neovisni o drugim čvorovima u klasteru, što može povećati brzinu i propusnost obrade (Pal, 2016). U slučaju da algoritam kompresije ne podržava dijeljenje, druga mogućnost s kojom kompresija ubrzava procesiranje je optimiziranje veličine datoteka (Hocanin, 2017). Na AWS Athena primjeru, ako su datoteke male (okvirno manje od 128 MB), procesiranje se može usporiti zbog vremena potrebnog za otvaranje svih S3 objekata, čitanja metapodataka objekata, čitanja rječnika kompresije i drugo. S druge strane, prevelike datoteke otežavaju paralelno procesiranje čime se proces usporava.

Primjena kompresije u pravilu znači balansiranje između procesorskih računalnih kapaciteta koji su potrebni i zahtjeva za pohranom podataka. Algoritmi kao što su Bzip, Gzip ili ZLib

zahtijevaju manje prostora za pohranu, ali također su sporiji, odnosno procesorska provedba kompresije traje duže. Nasuprot tome, algoritmi Snappy, LZO ili LZ4 imaju bržu procesorsku obradu, ali rezultiraju većim troškovima pohrane (Pal, 2016). Svaki algoritam ima različita obilježja i ne podržavaju ga nužno svi formati podataka, a neki od češćih prikazani su u tablici 1.

Tablica 1. Obilježja algoritama kompresije i podrška formata podataka

Algoritam	***Splittable* mogućnost?**	**Vrijeme kompresije**	**Veličina nakon kompresije**	**Omjer kompresije**[7]	**Avro**	**ORC**	**Parquet**	**TSV, CSV, JSON**
GZIP	Ne	Srednje	Mala	Visok	Ne	Ne	Da	Da
BZIP2	Da	Sporo	Mala	Vrlo visok	Samo čitanje	Ne	Ne	Da
LZO	Da	Brzo	Srednja	Nizak	Ne	Ne	Da	Samo čitanje
Snappy	Ne	Vrlo brzo	Velika	Nizak	Samo čitanje	Da	Da	Da

Izvor: Pal, 2016; Hocanin, 2017; Amazon Web Services, 2022 g

Iz pregleda obilježja u tablici 1 vidljivo je kako primjerice algoritmi GZIP i BZIP2 imaju najviše omjere kompresije od prikazanih, ali s druge strane također zahtijevaju duže vrijeme za procesiranje. LZO i Snappy su brži algoritmi koji rezultiraju manjim stupnjem kompresije, ali zato je primjerice Snappy podržan u najvećem broju formata od spomenutih algoritama. Na temelju navedenog može se zaključiti kako primijenjivost kompresije, kao i kod formata podataka, ovisi o poznavanju važnih obilježja i svrhe uporabe.

5.2.3. Particioniranje i bucketing

Postupci kao što su particioniranje, *bucketing* ili indeksiranje koriste se u svrhu smanjenja latencije upita. Particioniranje je postupak koji omogućava podjelu podataka tablice po jednom ili više stupaca, temeljem vrijednosti particioniranih stupaca (Pal, 2016). Time se podaci odvajaju u zasebne datoteke ili direktorije prema vrijednostima stupaca. Prilikom provođenja standardnog upita skeniraju se svi podaci u tablici, što može usporiti izvedbu zbog količine podataka koja se skenira. Kada su podaci particionirani, stupac koji predstavlja particiju može se uključiti u WHERE

[7] Omjer kompresije: omjer između veličine datoteke bez kompresije i veličine nakon kompresije. Što je rezultat viši, to je stupanj kompresije bolji. U pravilu, što je stupanj kompresije viši, to su veći procesorski zahtjevi (Hocanin, 2017)

uvjet upita, čime se skeniraju samo podaci koji pripadaju toj particiji, odnosno toj datoteci/direktoriju. Na taj način isključuju se svi podaci koji ne zadovoljavaju uvjet iz upita, čime se smanjuje vrijeme i trošak skeniranja podataka. Pritom, kako bi se prednosti particioniranja potpuno iskoristile, preporučljivo je izbjegavati stvaranje prevelikog broja particija (jer one stvaraju i toliki broj datoteka prema odabranim stupcima), a u slučaju da upit ne koristi WHERE uvjet, particioniranje neće činiti razliku. Zbog toga je particioniranje postupak koji je najbolje provoditi s ograničenim brojem stupaca. Dodatno, idealan slučaj za particioniranje su stupci koji imaju srodne vrijednosti ograničenog raspona, primjerice podaci o odjelima u organizaciji, država ili godine. U tom kontekstu važna je kardinalnost podataka u stupcu (jedinstvenost vrijednosti podataka stupca), na način da je particioniranje primijenjivije kod manje kardinalnosti gdje stupac ima ograničen broj jedinstvenih vrijednosti (Amazon Web Services, 2022 h).

Bucketing je postupak kojime se postiže strukturiranje podataka stupaca u dijelove kojima je lakše upravljati, a sastoji se od stvaranja segmenata koji sadrže određeni dio seta podataka (Pal, 2016). Za razliku od particioniranja, ovaj postupak najprimijenjiviji je kada se provodi nad stupcem koji ima visoku kardinalnost (velik broj jedinstvenih vrijednosti) i ujednačenu raspršenost vrijednosti u setu (Amazon Web Services, 2022 h). Primjerice, stupac u kojemu se nalaze identifikatori ili vremenske oznake događaja (engl. *timestamp*) predstavljaju ovakav tip podataka: očekivano je kako će možda i svaki slog relacije imati različitu vrijednost, a vrijednosti takvih zapisa prilično se ujednačeno raspoređuju u setu podataka. Takav stupac najvjerojatnije neće imati *null* vrijednosti, vrijednosti se mogu podijeliti u velik broj dijelova, a svaki dio bi trebao imati približno jednaku količinu podataka. Ovi primjeri predstavljaju idealan slučaj za *bucketing*, budući da je za ovaj postupak najbolje odabrati stupac koji ima visoku kardinalnost podataka, nema puno *null* vrijednosti, čiji se podaci mogu podijeliti u jednolike skupine, a skupine sadrže sličnu količinu podataka (Amazon Web Services, 2022 h). S obzirom na to da se skupine formiraju temeljem predvidljivih, do određene mjere kontinuiranih vrijednosti, moguće je skenirati samo skupinu u kojoj se nalazi određena vrijednost. Time se smanjuje količina podataka koju je potrebno skenirati, što posljedično znači i bržu provedbu upita.

5.2.4. Indeksiranje

U slučaju kada je potrebno pretraživati veliku količinu podataka, jedna od metoda za ubrzavanje upita je indeksiranje. Indeks se može definirati kao struktura u bazi podataka koja sadrži podatke o raspodjeli vrijednosti pojedinog stupca ili stupaca u tablici, a zapisanih na način koji ubrzava pretraživanje (Mujadžević, 2016). Svrha indeksa je brže pretraživanje često korištenih podataka u tablici, pri čemu postoje različite vrste indeksa s drukčijim algoritmima prikladnima ovisno o vrsti upita (Horvatinović, 2020). Ipak, „tradicionalno“ indeksiranje može biti izazovan postupak nad *big data* podacima, posebice pri korištenju usluga računarstva u oblaku koje imaju svoje specifičnosti pohrane i zbog drugih postupaka optimizacije podataka koji se mogu primijeniti. Stvaranje indeksa znači i formiranje nove tablice, tablice indeksa, koja se treba ažurirati usporedno s promjenama podataka. Međutim, *cloud* servisi za pohranu mogu imati određena ograničenja u mogućnostima ažuriranja podataka, primjerice ako podržavaju samo dodavanje novih podataka na kraj datoteke ili imaju posebna pravila ažuriranja zapisa (Pal, 2016).

Korištenje indeksa može ubrzati upite u određenim situacijama, ali zahtijeva i određenu pažnju u njihovom korištenju. Primjerice, indeksi se trebaju ažurirati prilikom upisa novih podataka u tablicu, što može značiti veće zahtjeve za resursima u sustavima koji zapisuju velike količine podataka. Također, indekse bi trebalo kreirati samo kada će se zaista koristiti jer inače nepotrebno zauzimaju memorijski prostor (Amazon Web Services, 2022 i).

U AWS okruženju postoji i postupak indeksiranja na način da se koriste particijski indeksi kroz AWS Glue Data Catalog. Kada nad podacima postoji velik broj particija, može se definirati indeks pomoću kojega se dohvaća podskup particija umjesto da se koristi cijeli skup postojećeg broja particija (Amazon Web Services, 2022 j). Stvaranjem indeksa zapravo se definira manji popis već postojećih particija u nekoj tablici, a popis može biti činiti bilo koja permutacija postojećih particija. Uzmimo za primjer tablicu s četiri particije: država, kategorija, godina, mjesec. Mogući indeksi u ovoj tablici su: (država, kategorija, godina, mjesec), (država, kategorija, godina), (država, kategorija), (država), (kategorija, država, godina, mjesec), itd. Data Catalog će konkatenirati vrijednosti particija onim redoslijedom kojim su bile navedene prilikom stvaranja indeksa. Pritom, postoje određena ograničenja u stvaranju indeksa, poput podržanih tipova podataka, operatora dostupnih u postavljanju upita koji koriste indekse i maksimalnog broja indeksa po tablici. Ipak, u

slučaju kada broj particija u tablici znatno naraste (npr. na stotine ili tisuće particija), ovaj postupak može biti koristan kao tehnika za poboljšanje performansi upita (Amazon Web Services, 2022 j).

5.2.5. Struktura upita i odabir naredbi

SQL upiti koji su ne samo funkcionalni, već i prilagođeni da budu efikasniji (engl. *query tuning*) mogu znatno poboljšati performanse izvedbe. Jedan od ključnih elemenata pritom je skraćivanje putanje potrebne bazi za dohvaćanje potrebnih podataka, a rezultati postupaka optimizacije uključuju brže vrijeme provedbe upita, smanjenje potrebnih procesorskih resursa i manji broj ulazno-izlaznih operacija (Mehrotra, 2016). Neki od postupaka koji se mogu primijeniti kroz strukturu i odabir naredbi opisani su u nastavku.

1) Izbjegavanje SELECT * klauzule. Budući da se ovom klauzulom dohvaćaju svi stupci tablice, uporaba ovog izraza može usporiti izvođenje upita (Mehrotra, 2016). Iako korisniku može biti jednostavnija za korištenje, poželjno je kod svakog upita razmotriti koji stupci su zapravo potrebni i dohvatiti samo njih.

2) Podupiti i JOIN naredbe. Podupiti se odnose na slučaj kada se unutar jednog (glavnog) upita poziva drugi upit. Mogu se postavljati na način da ne postoji veza između vanjskog upita i podupita ili stvaranjem veza čime se dobiva korelirani upit, kao što je prikazano u sljedećim primjerima (Mujadžević, 2016). Niže prikazani upit (lijevo) uz ime i prezime predavača ispisuje i ukupan broj održanih tečajeva od strane svih predavača, a budući da se ne stvara veza u svakom retku ispisat će broj ukupno održanih tečajeva za sve predavače. U primjeru s desne strane podupit će se izvršiti više puta; po jednom za svaki redak, što može znatno usporiti izvršavanje upita ako vanjski upit vraća velik broj redaka. Zbog toga, poželjno je izbjegavati korelirane upite jer mogu znatno utjecati na performanse upita.

```
SELECT Ime,
       Prezime,
       (SELECT COUNT(*) FROM Odrzavanje)
                              AS Ukupno
FROM Predavac;
```

```
SELECT Ime,
       Prezime,
       (SELECT COUNT(*) FROM Odrzavanje
        WHERE PredavacId = Predavac.Id)
                              AS Odrzao
FROM Predavac;
```

Umjesto ovakvih koreliranih podupita može se koristiti postupak *inline* pogleda koji podupit smješta u druge naredbe upita, kako bi se smanjilo opterećenje koje nastaje izvršavanjem podupita na razini svakog retka tablice (Mehrotra, 2016). U slučaju primjene podupita izvedba se može

poboljšati i na način da se umjesto njega upotrijebi JOIN naredba, ako pruža iste funkcionalnosti kao podupit (Schwartz, Zaitsev i Tkachenko, 2012).

JOIN spojevi tablica mogu se izvoditi koristeći više vrsta spojeva ovisno o potrebnom ishodu, a čije specifičnosti su cijela zasebna tema. Međutim, neke od općih smjernica korištenja JOIN naredbi u kontekstu optimizacije upita uključuju (Schwartz, Zaitsev i Tkachenko, 2012):

a) Redoslijed tablica. Određivanje redoslijeda kojim se tablice spajaju u pojedinom JOIN spoju može značajno utjecati na performanse upita (zbog primjerice broja redaka ili pretraživanja indeksa);

b) Odabir vrste spoja. Ovisno o željenom ishodu rezultata, važno je razmotriti mogu li se zahtjevi za spajanjem optimizirati korištenjem druge vrste spoja (npr. unutarnjeg spoja umjesto vanjskog), sužavanjem potrebnog broja tablica i načina spajanja te poznavanjem obilježja (npr. indeksa) atributa kojima se tablice spajaju.

Jedan od Presto postupaka koji može smanjiti zahtjeve JOIN spoja je i navođenje veće tablice na lijevoj strani spoja (odnosno manje tablice na desnoj strani), čime se može smanjiti memorijsko zauzeće i povećati brzina izvedbe upita (Hocanin, 2017).

3) GROUP BY naredba. Ova naredba koja se koristi za grupiranje podataka prema određenim stupcima može dovesti do većih zahtjeva resursa u slučaju da se ne koristi optimalno. Neki od postupaka koji mogu poboljšati njezino korištenje odnose se na ograničavanje broja stupaca u SELECT naredbi, čime se smanjuju memorijski zahtjevi za skladištenje podataka o redovima koji se ovom naredbom grupiraju (Hocanin, 2017). Pritom, poželjno je stupce navoditi redoslijedom od veće prema manjoj kardinalnosti, odnosno stupce navoditi redom počevši od onog s najvećim brojem jedinstvenih vrijednosti.

Također, prilikom korištenja JOIN naredbi, zahtjevi GROUP BY postupka mogu se smanjiti korištenjem stupaca iz samo jedne tablice (Schwartz, Zaitsev i Tkachenko, 2012).

4) ORDER BY naredba. Ova naredba koja služi za sortiranje podataka uzlazno ili silazno izvršava se nad svim retcima upita za koji je definirana, zbog čega može predstavljati značajan memorijski zahtjev kod većeg skupa podataka (Hocanin, 2017). Budući da je namjena ove naredbe sortiranje, uobičajeno je da korisnik sortira podatke jer želi zahvatiti najviše ili najniže vrijednosti nekog atributa. U tim slučajevima, postoje efikasnije naredbe koje mogu zamijeniti ili suziti ORDER BY sortiranje ograničavanjem broja podataka.

5) Ograničavanje broja redaka i vrijednosti. U slučajevima kada je potrebno prikazati samo određeni broj redaka, moguće je koristiti naredbe LIMIT ili TOP (Mujadžević, 2016). Primjerice, ako korisnik želi prikazati samo podskup podataka od 500 redaka, može to postići navođenjem LIMIT 500. Naredba se može koristiti i s prethodno spomenutom ORDER BY, čime se dohvaća zadani broj sortiranih redaka s najvišim ili najnižim vrijednostima navedenog atributa. Naredba TOP navodi se u SELECT izrazu i dohvaća zadani broj prvih redaka tablice s vrijednostima odabranih atributa bez sortiranja, a za sortiranje se također može koristiti u kombinaciji s ORDER BY. Opća struktura upita s ovim naredbama je:

```
SELECT Stupac1, Stupac2, Stupac3 FROM Tablica LIMIT N;
SELECT TOP N Stupac1, Stupac2, Stupac3 FROM Tablica;
```

6) Korištenje LIKE klauzule. Budući da tekstualne vrijednosti mogu sadržavati različite varijante sadržaja (velika i mala slova, različiti unosi), nerijetko se događa da korisnik ima potrebu pretražiti vrijednost nekog stupca tako da se ona samo djelomično preklapa sa zadanim znakovnim nizom. U tom slučaju, kao znak koji zamjenjuje druge znakove pri pretraživanju koristi se % (Mujadžević, 2016). Međutim, korištenje ovog postupka u pretraživanju može biti zahtjevno sa strane potrebnih resursa (a time i vremena izvođenja), posebice kada je potrebno uključiti više LIKE uvjeta u pretragu. Koristeći Presto, u ovom slučaju višestruki LIKE uvjeti mogu se zamijeniti funkcijom *regexp_like*, koja zbog drukčije metode pretraživanja obrasca zahtijeva manje resursa. U primjeru niže, s lijeve strane prikazan je upit koji u LIKE klauzuli sadrži više uvjeta koristeći OR logički operator. Ovaj upit može se optimizirati prilagodbom na način da koristi navedenu funkciju, čime se dobiva upit desno (Medium, 2021).

```
SELECT c1
FROM pin.trips
WHERE user_id LIKE '%usr1%'
   OR user_id LIKE '%usr2%'
   OR user_id LIKE '%usr3%'
   OR user_id LIKE '%usr4%';
```

```
SELECT c1
FROM pin.trips
WHERE regexp_like
(user_id,'usr1|usr2|usr3|usr4');
```

Osim navedenih postupaka, na temu *query tuning* mogućnosti mogao bi se navesti još cijeli niz opcija kojima se mogu poboljšati rezultati upita. Do sada opisane mogućnosti predstavljaju neke od češće primijenjivanih postupaka, ali i elemente koji će se primijeniti ili su relevantni za sljedeći dio ovog rada.

6. Rezultati optimizacije podataka i SQL upita na praktičnom primjeru u AWS servisima

Podaci korišteni za postupke u nastavku preuzeti su iz javno dostupnih Kaggle setova podataka (Davis, 2021). Korišteni set uključuje podatke o COVID-19 slučajevima, socioekonomskim obilježjima i vremenskim prilikama u Sjedinjenim Američkim Državama tijekom 2020. godine. Podaci su preuzeti kao .csv dokument veličine 1.3 GB, a sastoje se od 227 stupaca različitih tipova podataka i ukupno 790331 redaka. Namjena ovog seta podataka izvorno je bila pružiti kombinaciju zdravstvenih, socioekonomskih i vremenskih podataka na razini saveznih država, a u svrhu analize putem koje bi se identificirale ranjive skupine i potrebe u zajednicama s visokim COVID rizikom. U ovom radu, navedeni set podataka korišten je za kreiranje četiri relacijske tablice s različitim podacima u AWS servisima, a nad kojima su provedeni SQL upiti i prikupljeni rezultati provedbe.

6.1. Kreiranje tablica i obilježja podataka

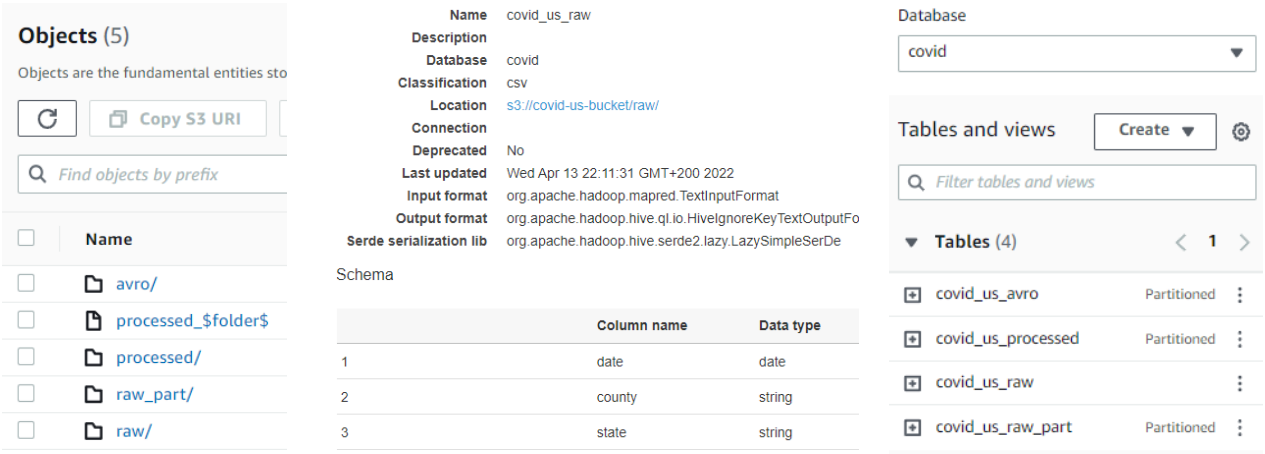
Izvorni podaci u csv formatu najprije su putem konzole učitani kao objekt u S3 spremnik pod prefiks[8] *raw*, koji naznačava da su u njemu spremljeni sirovi podaci. Na slici 7 (lijevo) prikazani su objekti S3 spremnika s prefiksima nakon kreiranja svih tablica, među kojima se u *raw* nalazi i originalni .csv dokument od 1.3 GB. U AWS Glue servisu zatim je kreiran prvi *crawler* kojemu je postavljena putanja na S3 *raw* prefiks, kako bi mogao koristiti metapodatke ovih .csv podataka. Shema podataka vidljiva je u Glue Data Catalogu u tablici covid_us_raw, kojoj je dio podataka prikazan na slici 7 (sredina). Iz ove slike vidljiva su neka svojstva koja se pohranjuju u Data Catalogu, poput formata, naziva baze i tablice, stupaca i tipova podataka u stupcima. Stvaranjem tablice u Glue Data Catalogu ovi podaci postaju dostupni i u Amazon Athena servisu. Na slici 7 (desno) nalazi se prikaz Athena baze *covid* nakon kreiranja svih tablica, koji uključuje i ovu prvu tablicu kreiranu iz originalnih .csv podataka pod istim nazivom kao u Data Catalogu.

Koristeći originalne podatke (sada prikazane kao tablica covid_us_raw), kreirana je sljedeća tablica (covid_us_raw_part). Također je u tekstualnom formatu, ali je ona kreirana kroz Athenu

[8] Prefiks označava naziv koncepta mape u S3 spremniku, putem kojega se objekti mogu grupirati koristeći zajednički naziv za skupinu objekata (Amazon Web Services, 2022 b)

korištenjem CTAS upita uz particioniranje po stupcu *state* i kompresiju koristeći GZIP algoritam. CTAS označava vrstu upita koji koristi sintaksu CREATE TABLE AS SELECT, a kojim se može kreirati nova tablica iz rezultata SELECT naredbe određenog upita (u ovom slučaju, iz *raw* podataka). Budući da je u CTAS upitu zadan i parametar kojim se podaci particioniraju po stupcu *state*, u njihovom S3 prefiksu (raw_part) kreirale su se 54 „mape" – particije. Nazivi particija odgovaraju saveznim državama (state=Alabama/, state=Alaska/, itd.), a svaka particija sadrži dokumente s podacima istoimene države.

Slika 7. Prikaz S3 spremnika s različitim prefiksima (lijevo), tablica covid_us_raw u Glue Data Catalogu (sredina) i prikaz covid baze s četiri tablice u Atheni (desno)



Izvor: Autor, travanj 2022.

Treća tablica (covid_us_avro) također je kreirana korištenjem CTAS upita nad izvornim podacima, ali se razlikuje od prethodne po tome što je zadan format AVRO. Nad podacima u ovoj tablici nije provedena kompresija, a particionirani su po istom stupcu kao i prethodna tablica (stupac *state*). Time su i u S3 prefiksu *avro* stvorene 54 particije po saveznim državama.

Zadnja tablica (covid_us_processed) kreirana je korištenjem AWS Glue resursa koji su uključivali *job* i drugi *crawler*. Najprije je kreiran *job* putem kojega su izvorni .csv podaci iz S3 *raw* prefiksa pretvoreni u komprimirani parquet format, particionirani po stupcu *state* i pohranjeni u novi S3 prefiks *processed*. Nakon izvršenja *job* postupka, kreiran je drugi *crawler* putem kojega je na temelju prethodno procesiranih podataka stvorena tablica covid_us_processed u Glue Data Catalogu. Veličina pojedinih setova podataka u različitim tablicama prikazana je kasnije u

rezultatima prema skeniranim podacima, a sažeti prikaz obilježja pojedinih prethodno opisanih tablica i podataka koje sadrže nalazi se u tablici 2.

Tablica 2. Pregled obilježja četiri kreirane tablice

	Postupak	**Format**	**Kompresija**	**Particije**
covid_us_raw	AWS Glue	CSV	-	-
covid_us_avro	CTAS	AVRO	-	stupac *state*
covid_us_raw_part	CTAS	CSV	GZIP	stupac *state*
covid_us_processed	AWS Glue	PARQUET	GZIP	stupac *state*

Izvor: Autor, travanj 2022.

Budući da korišteni set izvornih podataka po veličini nije bio zahtjevan za obradu, dvije tablice (covid_us_avro i covid_us_raw_part) kreirane su koristeći CTAS upit u Atheni. U ovom slučaju je ovakav postupak bilo moguće primijeniti jer nisu bili potrebni značajniji resursi, ali u kontekstu *big data* podataka postoji mogućnost da, zbog trajanja postupka i ograničenja CTAS upita/Athena servisa, ne bi bilo moguće tablice kreirati na taj način. U slučajevima kada su zahtjevi za resursima veći, mogu se koristiti AWS Glue mogućnosti (a po potrebi i dodatni servisi, ovisno o opsegu zahtjeva).

6.2. Korišteni SQL upiti

Za prikupljanje rezultata ukupno je korišteno osam SQL upita koji se razlikuju prema namjeni, strukturi i količini podataka koje skeniraju (izrada autora). Prva dva upita primijenjena su u svrhu usporedbe rezultata ovisno o formatu, kompresiji i particioniranju podataka, dok su preostali upiti inačice u nekoliko primjera temeljem kojih su dobiveni pokazatelji za usporedbu upita ovisno o stupnju optimiziranosti.

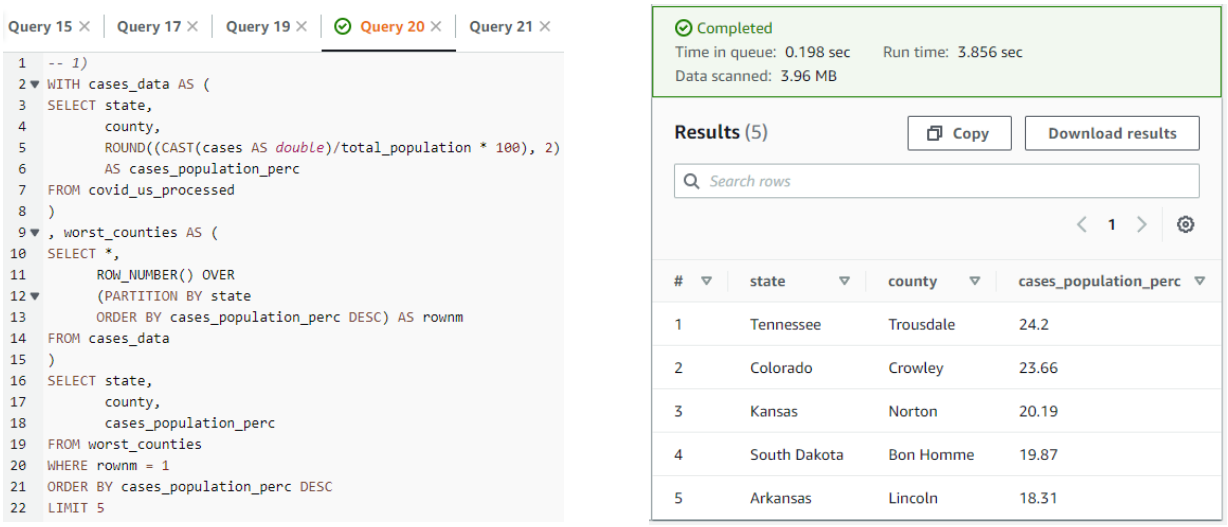
Prvi upit (Prilog 1, upit 1) primijenjen je u svrhu prikaza rezultata (količine skeniranih podataka i vremena izvođenja upita) nad cijelim setovima podataka, u tablicama koje se razlikuju po formatu i postupku kompresije podataka. Upitom se dobivaju podaci o prvih pet saveznih država sa svojim vodećim okrugom po postotku zabilježenih COVID-19 slučajeva u odnosu na ukupnu populaciju okruga. Drugi upit (Prilog 1, upit 2) korišten je u svrhu dobivanja rezultata iz kojih je vidljiv učinak particioniranja, putem WHERE uvjeta koji uključuje odabrane savezne države. Poput prvog upita, također je proveden nad svim tablicama. Ovim upitom dobivaju se podaci koji prikazuju najveći

postotak cijepljenih stanovnika u pet prvih država i njihovih vodećih okruga po zabilježenom broju slučajeva u odnosu na populaciju, počevši od najmanje procijepljenosti. Treći upit (Prilog 1, upit 3) ima dvije srodne inačice koje se razlikuju po stupnju optimiziranosti, ali vraćaju iste rezultate: prikazuju podatke zabilježene tijekom ljetnih mjeseci 2020. S obzirom na to kako su u ovom slučaju uspoređene dvije inačice istog upita, provedeni su nad istom tablicom (parquet podaci). Preostali upiti (Prilog 1, upit 4) predstavljaju četiri srodne verzije istog upita koje se razlikuju po tome što uključuju ili isključuju pojedine naredbe (ORDER BY, LIMIT, WHERE). Poput prethodnog skupa upita, također su zbog usporedbe razlika između inačica provedeni nad istim (parquet) podacima. Upitima se dobivaju vremenski podaci na razini saveznih država o tome kada je u 2020. godini zabilježen prvi i zadnji COVID-19 slučaj, koliki je raspon dana u kojemu su bilježeni slučajevi, i koji je zadnji dostupan omjer slučajeva u populaciji na zadnji datum podataka.

6.3. Rezultati provedenih SQL upita

Korišteni upiti provedeni su u Athena servisu. Svaki upit pokrenut je ukupno pet puta, pri čemu je za svaku izvedbu zabilježeno vrijeme izvođenja (engl. *run time*) i količina skeniranih podataka (engl. *data scanned*). Na slici 8 prikazan je izgled sučelja za uređivanje upita (engl. *query editor*) i rezultati uspješno izvršenog upita, uključujući i način prikaza parametara izvršenog upita u Atheni.

Slika 8. Prikaz Athena sučelja za uređivanje upita (*query editor*; lijevo) i rezultata izvršenog upita (desno)

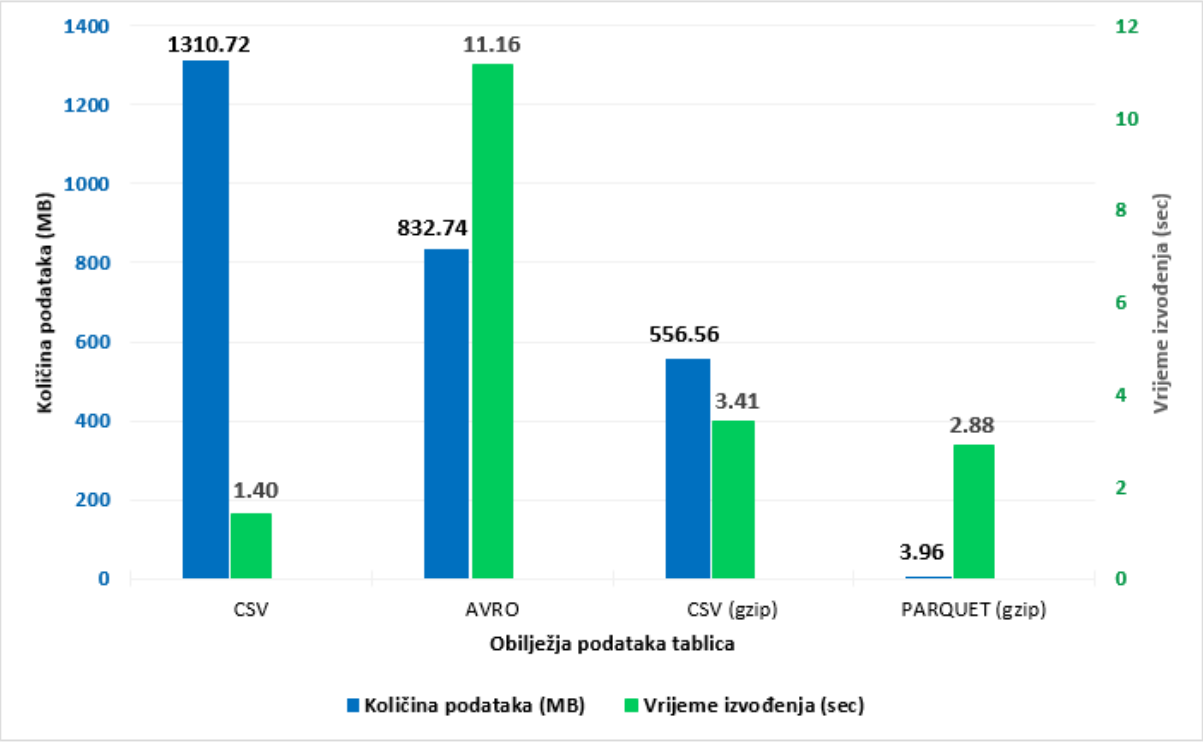


Izvor: Autor, travanj 2022.

Količina skeniranih podataka u istom upitu nije promjenjiva, ali vrijeme izvedbe jest, zbog čega se kao rezultat vremena izvođenja upita računao prosjek pet izvedbi. U rezultatima svih osam upita prikazano je prosječno vrijeme dobiveno na ovaj način. Dobiveni rezultati prikazani su u tri skupine: 1) rezultati prvog upita za usporedbu formata podataka i postupaka kompresije, 2) rezultati drugog upita koji prikazuju učinak particioniranja i 3) usporedba performansi u odnosu na stupanj optimizacije, koristeći treći upit s dvije inačice (Prilog 1, upit 3) i zadnji upit s četiri inačice (Prilog 1, upit 4).

1) Usporedba formata podataka i postupaka kompresije. Za ovu usporedbu korišteni su rezultati dobiveni prvim upitom (Prilog 1, upit 1), a prikazani su na grafikonu 1. Ovaj upit skenira cijeli set podataka u tablicama. Stupci koji su u grafikonu prikazani u skupinama nazvanima prema formatu i algoritmu kompresije odnose se na četiri kreirane tablice koje sadrže podatke s navedenim obilježjima (CSV = covid_us_raw, AVRO = covid_us_avro, CSV gzip = covid_us_raw_part, PARQUET gzip = covid_us_processed).

Grafikon 1. Prikaz performansi prvog SQL upita nad podacima četiri različite tablice



Izvor: Autor, travanj 2022.

Iz grafikona 1 vidljivo je kako je najkraće vrijeme izvedbe u prosjeku zabilježeno u provedbi upita kod CVS formata bez kompresije (nad tablicom s izvornim podacima), ali istovremeno je u ovoj tablici zabilježena najveća količina skeniranih podataka (1310.72 MB). U usporedbi s ovom tablicom, provedbom istog upita nad AVRO podacima bez kompresije trajanje izvedbe upita bilo je znatno duže – gotovo 8 puta više vremena, ali je količina skeniranih podataka bila manja (832.74 MB). Oba formata (CSV i AVRO) pohranjuju podatke u retčanom obliku, ali AVRO se pokazao boljim za čitanje podataka. Upitom se skenira manja količina podataka pa je povoljniji kod korištenja Athene, iako je po pitanju vremena izvedbe pokazao manju efikasnost. Ovi rezultati u skladu su primjerice s odnosom koji je ispitivao Radečić (2021) uspoređujući performanse CSV i AVRO formata, u kojemu je također pronašao kako AVRO zahtijeva duže vrijeme od CSV-a kod čitanja podataka (engl. *read time*) te reducira veličinu podataka. Međutim, zanimljivo je kako u ovom odnosu nije ustanovio isti obrazac kod zapisivanja podataka (engl. *write time*), kod kojega je AVRO bio za 40% brži nego CSV. S obzirom na navedeno, AVRO pokazuje određene prednosti nad CSV-om u slučaju primjene u upitima koji čitaju podatke, ali nije bio najefikasniji format.

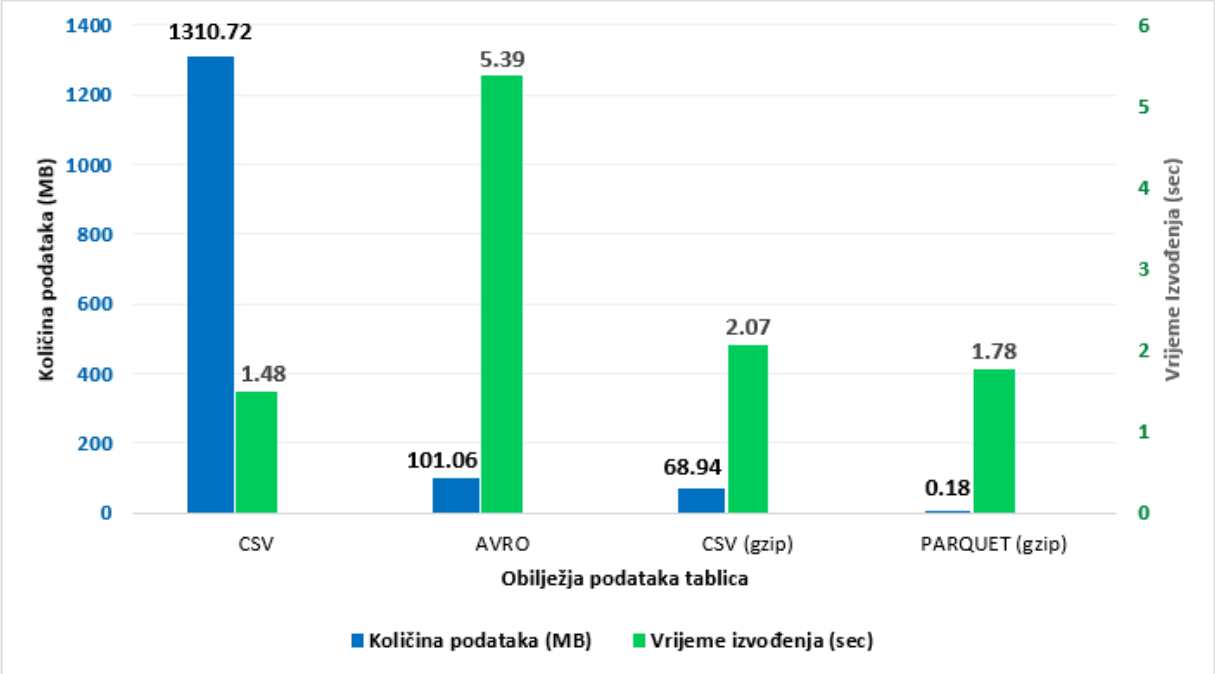
Zadnja skupina na grafikonu 1 prikazuje rezultate provedbe upita nad tablicom s parquet podacima. Budući da su podaci u ovoj tablici komprimirani, mogu se usporediti s kategorijom CSV (gzip) koja također sadrži komprimirane podatke, ali u CSV formatu. Vidljivo je kako se provedbom istog upita nad parquet podacima količina skeniranih podataka izrazito smanjila (na 3.96 MB). U odnosu na izvorne podatke, na parquet podacima skenirana je čak 330 puta manja količina podataka, dok je izvedba upita bila približno 15% brža u odnosu na komprimirani CSV.

U odnosu na izvorne CSV podatke bez kompresije, nad komprimiranim podacima istog formata upitom je skenirano 58% manje podataka (556.56 MB u odnosu na početnih 1310.72 MB). Iz ovog odnosa vidljiv je značajan doprinos gzip algoritma kompresije, a najveća razlika primjetna je u slučaju parquet formata koji prema osnovnim postavkama primijenjuje gzip kompresiju. Općenito se u dobivenim rezultatima parquet pokazao najefikasnijim formatom za pohranu i skeniranje podataka. Budući da količina skeniranih podataka u Atheni izravno utječe na troškove upita, korištenje parquet podataka u ovom slučaju je i najpovoljnija opcija. Ako uzmemo u obzir da prema Athena cjeniku trošak skeniranja 1TB podataka po upitu iznosi 5.00 USD (američkih dolara), to bi na primjer značilo da 1000 upita koji skeniraju po 1.28 GB iznosi ukupno 6.25 USD (AWS Pricing Calculator, 2022). U usporedbi s time, 1000 upita koji skeniraju po 3.96 MB iznosilo bi manje od

0.05 USD. Iako ovi iznosi nisu sami po sebi visoki, na većoj skali *big data* podataka planiranje odabira formata i postupaka nad podacima može imati značajan utjecaj na komponente troškova.

2) Particioniranje. Rezultati ovog postupka prikazani su na grafikonu 2, a predstavljaju vrijednosti dobivene provedbom drugog upita (Prilog 1, upit 2). Kao i kod rezultata prvog upita, grafikon prikazuje količinu skeniranih podataka i prosječno vrijeme izvođenja upita nad četiri tablice koje sadrže različite tipove podataka. U ovom slučaju, upit je strukturiran na način da u WHERE uvjetu koristi stupac *state*, a koji u pojedinim tablicama predstavlja atribut po kojemu su podaci particionirani.

Grafikon 2. Prikaz performansi drugog SQL upita nad podacima četiri različite tablice



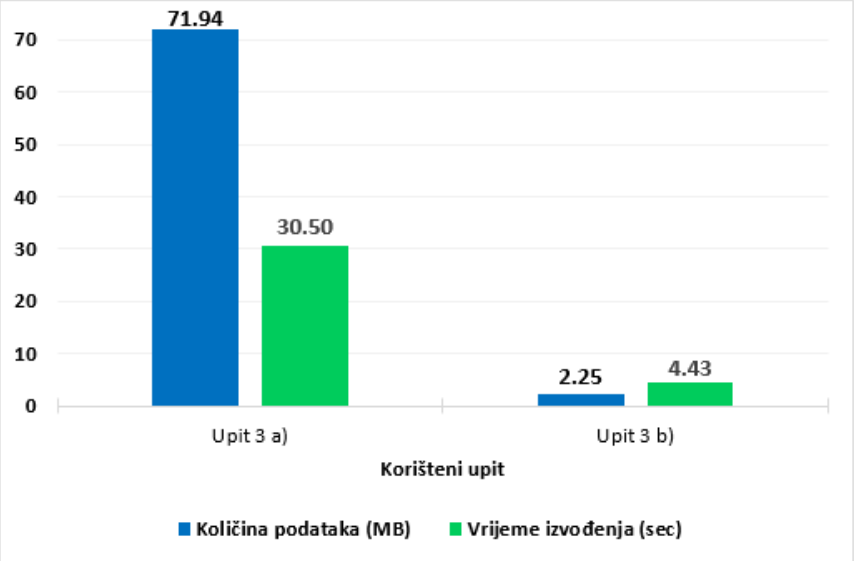
Izvor: Autor, travanj 2022.

Tablica s izvornim podacima (CSV) nije particionirana, zbog čega se neovisno o WHERE uvjetu prilikom provedbe SQL upita skenira cijeli set podataka (1310.72 MB), jednako kao kod provedbe prvog upita na grafikonu 1. Podaci u svim ostalim tablicama particionirani su po stupcu *state*. U usporedbi s izvornim podacima, nad drugom tablicom koja sadrži CSV podatke (CSV gzip) drugi upit skenira značajno manje podataka (68.94 MB). Ako usporedimo ovaj rezultat s

istom tablicom i njezinim vrijednostima prvog upita koji se izvršio nad cijelim setom komprimiranih podataka CSV tablice (grafikon 1), vidljivo je kako se korištenjem particija količina skeniranih podataka znatno smanjila (nad cijelom tablicom skenira se 556.56 MB). Isti odnos vidljiv je i kod podataka ostalih tablica: iz grafikona 2 primjetno je kako u odnosu na izvorne CSV podatke, sve ostale particionirane tablice skeniraju znatno manje podataka. Ovaj odnos vidljiv je i iz usporedbe vrijednosti za iste tablice na grafikonu 1 i grafikonu 2: u drugom upitu s WHERE uvjetom koji uključuje pet saveznih država, tablica s AVRO podacima skenira 88% manje podataka (101.06 MB naspram 832.74 MB), dok se količina skeniranih podataka parquet formata smanjila za 95% (0.18 MB u odnosu na 3.96 MB). U slučaju da tablice nisu bile particionirane po stupcu *state*, ponovno bi se i u ovom slučaju skenirao cijeli set podataka, kao što je to slučaj kod tablice s izvornim CSV podacima. Kako se korištenjem particija smanjila količina skeniranih podataka, tako se smanjilo i prosječno vrijeme izvedbe upita.

3) Usporedba performansi u odnosu na stupanj optimizacije upita. Na grafikonu 3 prikazani su rezultati dobiveni provedbom dvije inačice trećeg SQL upita (Prilog 1, upit 3) nad podacima u tablici covid_us_processed. Format podataka u ovom slučaju nije presudan za usporedbu upita, pa je odabrana tablica koja ima parquet podatke budući da je ona najpovoljnija za provedbu upita.

Grafikon 3. Prikaz performansi inačica trećeg SQL upita na parquet podacima



Izvor: Autor, travanj 2022.

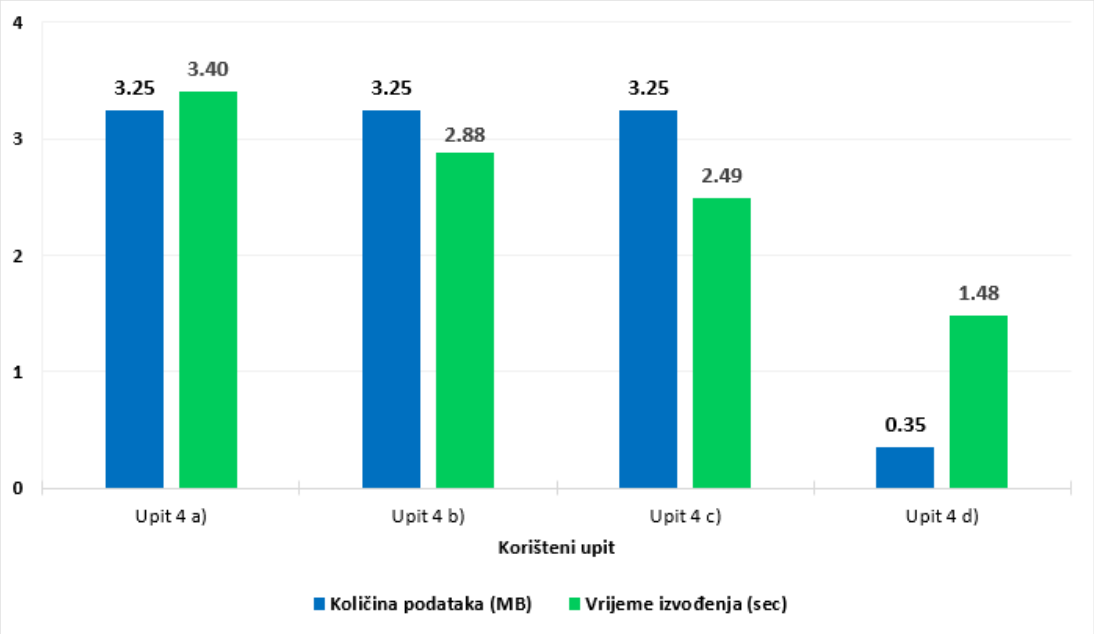
U prvoj inačici upita (Prilog 1, upit 3a) naredbom SELECT odabiru se svi stupci (SELECT *). WHERE uvjetom dohvaćaju se podaci samo za određene mjesece, navođenjem niza LIKE klauzula s logičkim operatorom OR. U drugoj inačici upita (Prilog 1, upit 3b) uvedene su dvije izmjene s ciljem bolje optimizacije upita, a uključuju odabir samo potrebnih stupaca (njih ukupno 7) i zamjenu niza LIKE klauzula funkcijom *regexp_like*. Rezultati dobiveni provedbom inačice 3a pokazuju kako se ovim (neoptimiziranim) upitom na parquet formatu skenira 71.94 MB podataka. Za provedbu ovog upita, prosječno je bilo potrebno 30.5 sekundi. U usporedbi s njim, drugi (optimizirani) upit skenira znatno manje podataka (2.25 MB), zbog čega je i vrijeme provedbe upita gotovo 7 puta brže. Budući da tablica nad kojom su provedeni ovi upiti ima 227 stupaca, postupak odabira svih stupaca (SELECT *), iako jednostavniji, rijetko je opravdan. Analitički upiti u pravilu se i primijenjuju u svrhu dohvaćanja specifičnog skupa podataka koji daju odgovor na određeno pitanje, za što je dovoljan ciljani skup stupaca. Najveći udio smanjenja količine skeniranih podataka u ovoj usporedbi odnosi se upravo na odabir nekolicine stupaca koji daju potrebne podatke, uz što se mogu primijeniti i drugi postupci za dodatno poboljšanje performansi.

Rezultati prikazani na grafikonu 4 odnose se na posljednji primjer u kojemu su uspoređene inačice upita koje se razlikuju po određenim naredbama putem kojih se postiže manji ili veći stupanj optimiziranosti. Navedeni rezultati dobiveni su provedbom četiri inačice SQL upita (Prilog 1, upit 4) nad parquet podacima u tablici covid_us_processed. Razlike među inačicama su sljedeće:
4 a) uključuje ORDER BY naredbu;
4 b) uključuje ORDER BY i LIMIT naredbe;
4 c) izostavlja ORDER BY i LIMIT naredbe;
4 d) izostavlja ORDER BY i LIMIT naredbe te uključuje WHERE uvjet za dohvaćanje podataka tri odabrane savezne države.

Prve tri inačice upita dohvaćaju iste podatke, a razlikuju se prema naredbi za sortiranje i/ili limitiranje podataka na kraju upita. Iz grafikona 4 vidljivo je kako skeniraju jednaku količinu podataka (3.25 MB), dok se prosječno vrijeme izvođenja razlikuje. Najduže vrijeme izvođenja (3.40 sec) zabilježeno je u inačici 4a (Prilog 1, upit 4a), koja koristi ORDER BY naredbu nad cijelim setom podataka, iz čega je vidljivo kako već samo ova naredba može usporiti izvođenje upita. U usporedbi s ovom inačicom, u 4b (Prilog 1, upit 4b) dodatno je uključena naredba LIMIT

kojom se ograničava dohvaćanje 10 sortiranih redaka (silaznim redoslijedom zadanim u naredbi ORDER BY). Dodavanjem LIMIT naredbe na postojeći upit vrijeme izvođenja upita smanjilo se za 15% (2.88 u odnosu na 3.40 sekundi). U trećoj inačici (Prilog 1, upit 4c) izostavljena je naredba za sortiranje, čime se vrijeme izvođenja dodatno smanjilo na prosječno 2.49 sekunde. Najveća razlika u inačicama prisutna je u zadnjoj varijanti upita (Prilog 1, upit 4d), u kojoj se ne primijenjuje sortiranje svih podataka, već se iz podataka uzimaju samo države za čije rezultate smo zainteresirani (u ovom slučaju promatrane su tri najveće savezne države po ukupnoj populaciji: Kalifornija, Teksas i Florida). Ograničavanjem broja podataka kroz stupac *state*, po kojemu su podaci particionirani, upit skenira znatno manju količinu podataka (0.35 MB u odnosu na prijašnjih 3.25 MB), a time je i vrijeme izvođenja upita kraće.

Grafikon 4. Prikaz performansi inačica četvrtog SQL upita na parquet podacima



Izvor: Autor, travanj 2022.

Iz ovog primjera vidljivo je kako se uključivanjem samo jedne dodatne naredbe (ORDER BY) vrijeme izvođenja upita može usporiti. U slučajevima kada je ta naredba neophodna nema nužno prostora za optimizaciju po pitanju sortiranja, međutim kod ovog slučaja (ovisno o inačici) upit zbog grupiranja rezultata vraća najviše 54 retka, jedan za svaku državu. Toliki broj redaka može se pregledati i koristeći mogućnost sortiranja koju Athena pruža u samom sučelju, ako je takva

akcija potrebna. Dodatno, u situacijama u kojima je potrebno sortiranje ono se najčešće provodi kako bi se sagledao skup najviših ili najnižih vrijednosti u rezultatima. Zbog toga, korisno je prilikom sortiranja razmotriti opciju limitiranja rezultata na onaj podskup koji promatramo, što se u ovom primjeru (Prilog 1, upit 4b) pokazalo korisnim za smanjenje vremena izvedbe upita. Poput primjera na grafikonu 3, najkorisniji postupak za optimizaciju upita i u ovom slučaju bilo je limitiranje količine dohvaćenih podataka u samom upitu, na način da se dohvaćaju samo potrebni podaci. Ako iz rezultata koje vraća upit trebamo samo dio redaka (primjerice samo neke države ili dane), poželjno je strukturu upita planirati na način da dohvaća samo potrebne stupce ili koristi uvjete, kako se ne bi nepotrebno obrađivali i podaci koji se na kraju neće upotrijebiti. Osim toga, u slučajevima u kojima se zbog analitičkih zahtjeva može predvidjeti način na koji će se podaci učestalo pregledavati, korisno je razmotriti dodatne postupke poput prikazanog particioniranja koje može značajno utjecati na performanse, i u konačnici trošak resursa.

7. Zaključak

Glavni cilj ovog rada bio je prikazati, opisati i istražiti na praktičnom primjeru različita obilježja i postupke koji se mogu provesti nad podacima u svrhu optimizacije SQL upita u AWS servisima. Rastući trend korištenja *big data* skupova podataka dovodi do potrebe za upravljanjem podacima koje karakteriziraju velika količina, brzina nastanka i korištenja te raznolikost strukture i formata. Budući da obrada ovakvih podataka može zahtijevati jače računalne resurse ili veću fleksibilnost rada, zbog svojih prednosti računarstvo u oblaku postaje sve zastupljeniji model u poslovanju. Amazon Web Services trenutno je jedan od najvećih pružatelja usluga računarstva u oblaku, a pruža usluge dostupne putem više od 200 pojedinačnih *cloud* servisa i resursa. Za provedbu primjera u ovom radu, od ključnih AWS servisa korišteni su Amazon S3, AWS Glue i Amazon Athena.

Za dohvaćanje podataka i postavljanje upita nad bazama podataka uobičajeno se koristi SQL upitni jezik, koji zbog svojih brojnih mogućnosti nudi i značajan prostor za primjenu različitih postupaka nad podacima i upitima u svrhu veće efikasnosti. U ključne postupke mogu se svrstati odabir formata i provedba kompresije podataka, ali i razne druge mogućnosti značajne za rad u *cloud* okruženju, poput particioniranja podataka i planiranja strukture SQL upita.

Koristeći javno dostupan set s COVID-19 podacima u SAD-u tijekom 2020., u praktičnom dijelu rada su putem AWS servisa kreirane četiri tablice različitih obilježja. Glavna obilježja po kojima se razlikuju obuhvaćaju format podataka (csv/avro/parquet), odabir kompresije (bez kompresije/ gzip) i primjenu postupka particioniranja (da/ne). Rezultati su prikupljeni na način da je nad ovim podacima proveden veći broj SQL upita, a zatim su uspoređene dobivene vrijednosti. Po pitanju formata podataka, najučinkovitijim formatom pokazao se stupčasti parquet, dok se provođenje postupka kompresije općenito pokazalo korisnim za smanjenje količine skeniranih podataka. Prilikom provedbe upita koji omogućava korištenje particija, postupak particioniranja pokazao je prednosti u svim primijenjenim slučajevima. Sljedeći dio rezultata uključivao je postupke optimizacije strukture SQL upita i odabira naredbi, uspoređene različitim inačicama srodnih upita. Neki od postupaka koji su primijenjeni, a koji su pokazali da se njima može poboljšati izvedba upita, uključuju planiranje i dohvat samo potrebnih stupaca, primjenu odgovarajućih funkcija, pragmatično korištenje sortiranja, primjenu ograničenja broja rezultata i

prilagodbu upita za uporabu particija. Temeljem opisanih postupaka i dobivenih rezultata, može se zaključiti kako postoji veći broj mogućnosti putem kojih se može optimizirati obrada podataka SQL upitima u *cloud* servisima, a koji su važni jer doprinose učinkovitom i isplativom korištenju usluga u oblaku za rad s *big data* podacima.

Popis pokrata

API = Application Programming Interface

AWS = Amazon Web Services

CLI = Command Line Interface

CSV = Comma-Separated Values

CTAS = Create Table As Select

CTE = Common Table Expressions

DCL = Data Control Language

DD = Data Dictionary

DDL = Data Definition Language

DML = Data Manipulation Language

DTL = Data Transfer Language

DQL = Data Query Language

ETL = Extract, Transform, Load

IaaS = Infrastructure as a Service

IoT = Internet of Things

JDBC = Java Database Connectivity

JSON = JavaScript Object Notation

ODBC = Open Database Connectivity

ORC = Optimized Row Columnar

PaaS = Platform as a Service

SaaS = Software as a Service

SDK = Software Development Kit

SQL = Structured Query Language

Popis literature

Amazon Web Services, Overview of Amazon Web Services – AWS Whitepaper, 2022 a, https://docs.aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf (23.03.2022.)

Amazon Web Services, Amazon Simple Storage Service – User Guide, 2022 b, https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-userguide.pdf (23.03.2022.)

Amazon Web Services, Amazon Athena – User Guide, 2022 c, https://docs.aws.amazon.com/athena/latest/ug/athena-ug.pdf (24.03.2022.)

Amazon Web Services, Amazon Athena FAQs, 2022 d, https://aws.amazon.com/athena/faqs/#When_to_use_Athena_vs_other_big_data_services (24.03.2022.)

Amazon Web Services, AWS Glue Developer Guide, 2022 e, https://docs.aws.amazon.com/glue/latest/dg/glue-dg.pdf (26.03.2022.)

Amazon Web Services, What is AWS, 2022 f, https://aws.amazon.com/what-is-aws/ (20.03.2022.)

Amazon Web Services, Athena Compression Support, 2022 g, https://docs.aws.amazon.com/athena/latest/ug/compression-formats.html (06.04.2022.)

Amazon Web Services, Bucketing vs Partitioning, 2022 h, https://docs.aws.amazon.com/athena/latest/ug/bucketing-vs-partitioning.html (09.04.2022.)

Amazon Web Services, Managing Indexes, 2022 i, https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SQLtoNoSQL.Indexes.html (09.04.2022.)

Amazon Web Services, Working with Partition Indexes, 2022 j, https://docs.aws.amazon.com/glue/latest/dg/partition-indexes.html (09.04.2022.)

AWS Pricing Calculator, https://calculator.aws/#/createCalculator/Athena (23.04.2022.)

Buyya, R., Calheiros, R. N., Dastjerdi, A. V., Big Data Principles and Paradigms, Elsevier Inc., Cambridge, MA, 2016.

Chandrabose, V., AWS – Transform Data Using AWS Glue and Amazon Athena, 2020, https://dev.to/iamvigneshc/aws-transform-data-using-aws-glue-and-amazon-athena-2a7l (26.03.2022.)

Davis, J., US Counties: COVID19 + Weather + Socio/Health data, 2021, https://www.kaggle.com/datasets/johnjdavisiv/us-counties-covid19-weather-sociohealth-data?select=US_counties_COVID19_health_weather_data.csv (13.04.2022.)

Dumbill, E., Planning for Big Data, O'Reilly Media Inc., Sebastopol, CA, 2012.

Eurostat, Cloud computing used by 42% of enterprises, 2021, https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20211209-2 (20.03.2022.)

Farena, N., Kompresija podataka, diplomski rad, Sveučilište J. J. Strossmayera u Osijeku, Osijek, 2020.

FER, Napredni modeli i baze podataka – predavanja, 2018, https://www.fer.unizg.hr/_download/repository/2._Napredni_SQL[5].pdf (02.04.2022.)

Gartner Glossary, Cloud Computing, 2022, https://www.gartner.com/en/information-technology/glossary/cloud-computing (20.03.2022.)

Getz, A., Column And Row Based Database Storage, 2014, https://bi-insider.com/business-intelligence/column-and-row-based-database-storage/ (04.04.2022.)

Gillis, A. S., Amazon Web Services (AWS), 2020, https://www.techtarget.com/searchaws/definition/Amazon-Web-Services (23.03.2022.)

Hocanin, M., Top 10 Performance Tuning Tips for Amazon Athena, 2017, https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/ (06.04.2022.)

Horvatinović, A., Sustav za upravljanje bazama podataka PostgreSQL 12, završni rad, Fakultet organizacije i informatike, Varaždin, 2020.

IBM, Apache Avro, 2022, https://www.ibm.com/topics/avro (04.04.2022.)

IBM Cloud, Top 7 Most Common Uses of Cloud Computing, 2020, https://www.ibm.com/cloud/blog/top-7-most-common-uses-of-cloud-computing (20.03.2022.)

Kaggle, Analytic Functions, 2022, https://www.kaggle.com/code/alexisbcook/analytic-functions/tutorial (02.04.2022.)

Krmpotić, G., SaaS VS PaaS VS IaaS, 2020, www.gorankrmpotic.eu/saas-vs-paas-vs-iaas/ (20.03.2022.)

Leskovar, N., Oracle SQL Developer, završni rad, Fakultet organizacije i informatike, Varaždin, 2014.

Medium, 10 Tips For Presto Query Performance Optimization, 2021, https://medium.com/@seals_xyz/10-tips-for-presto-query-performance-optimization-37fa0b7c6dc3 (10.04.2022.)

Mehrotra, S., SQL Performance Tuning, 2016, https://www.globallogic.com/il/wp-content/uploads/2016/02/SQL-Performance-Tuning.pdf (10.04.2022.)

Michels, J., Hare, K., Kulkarni, K., Zuzarte, C., Liu, Z. H., Hammerschmidt, B., Zemke, F., The New and Improved SQL:2016 Standard, ACM SIGMOD Record, 47(2), 2018, str. 51-60.

Mujadžević, E., Uvod u SQL – priručnik za polaznike, Sveučilišni računski centar, Zagreb, 2016.

Nguyen, C., Distinguish 4 Ranking Functions in SQL, 2022, https://towardsdatascience.com/distinguish-4-ranking-functions-in-sql-37db99107c05 (02.04.2022.)

Oracle, Single-Row Functions, 2017, https://docs.oracle.com/database/121/SQLRF/functions002.htm#SQLRF51178 (02.04.2022.)

Pal, S., SQL on Big Data: Technology, Architecture and Innovation, Apress, Wilmington, MA, 2016.

PCSC, Public Cloud Services Comparison, 2018, https://comparecloud.in/ (20.03.2022.)

Radečić, D., CSV Files for Storage? Absolutely Not. Use Apache Avro Instead, 2021, https://towardsdatascience.com/csv-files-for-storage-absolutely-not-use-apache-avro-instead-7b7296149326 (23.04.2022.)

Rotem, U., Serverless Architecture for a Structured Data Mining Solution, 2021, https://aws.amazon.com/blogs/architecture/serverless-architecture-for-a-structured-data-mining-solution/ (23.03.2022.)

Sander, R., 6 Ways Cloud Computing is Changing Business Today, 2021, www.smallbusinessbonfire.com/cloud-computing-changing-business-today/ (20.03.2022.)

Schwartz, B., Zaitsev, P., Tkachenko, V., High Performance MySQL, O'Reilly Media Inc., Sebastopol, CA, 2012.

Statista Research Department, Big data – Statistics & Facts, 2022, https://www.statista.com/topics/1464/big-data/#dossierKeyfigures (26.03.2022.)

Vailshery, L. S., Organizations' use of cloud vendors worldwide 2021 by vendor, 2022, www.statista.com/statistics/1224552/organization-usecloud-provider-global (20.03.2022.)

Vukelić, D., Izrada ETL alata za transformaciju podataka iz polustrukturiranih izvora, diplomski rad, Fakultet organizacije i informatike, Varaždin, 2014.

Woodie, A., Big Data File Formats Demystified, 2018, https://www.datanami.com/2018/05/16/big-data-file-formats-demystified/ (04.04.2022.)

Popis slika

Popis tablica

Popis grafikona

Prilog 1. SQL upiti

Upit 1. Prvih pet saveznih država s vodećim okrugom po % zabilježenih slučajeva u populaciji

```
WITH cases_data AS (
SELECT state,
       county,
       ROUND((CAST(cases AS double)/
       total_population * 100), 2)
       AS cases_population_perc
FROM covid_us_*)
, worst_counties AS (
SELECT *,
      ROW_NUMBER() OVER
      (PARTITION BY state ORDER BY
cases_population_perc DESC) AS rownm
FROM cases_data)
SELECT state,
       county,
       cases_population_perc
FROM worst_counties
WHERE rownm = 1
ORDER BY cases_population_perc DESC
LIMIT 5
```

Rezultat provedbe upita 1:

#	state	county	cases_population_perc
1	Tennessee	Trousdale	24.2
2	Colorado	Crowley	23.66
3	Kansas	Norton	20.19
4	South Dakota	Bon Homme	19.87
5	Arkansas	Lincoln	18.31

Obrazloženje upita 1:

Prvi korak upita je CTE `cases_data` koji započinje ključnom riječi `WITH`, a služi za dohvaćanje potrebnih stupaca iz pojedine tablice nad kojom se upit provodi. Osim stupaca `state` i `county` koji se izravno dohvaćaju, pokazatelj koji se koristi je postotak COVID-19 slučajeva u populaciji, a računa se na način da se udio slučajeva (količnik) pomnoži sa 100 i zaokruži na dvije decimale. To se postiže naredbom `ROUND` i njezinim parametrom 2 (broj decimala). Budući da su stupci `cases` i `total_population` u tablicama tipa *bigint*, jedan od stupaca (`cases`) pomoću naredbe `CAST AS double` pretvoren je u decimalnu vrijednost, kako bi se mogao izračunati postotak na više decimala. Stupcu koji će sadržavati vrijednost izračuna ovog postotka dodijeljen je naziv (*alias*) `cases_population_perc`. Sljedeći korak je drugi CTE `worst_counties`, koji najprije dohvaća sve stupce iz prvog podupita `cases_data`, a zatim primijenjuje `ROW_NUMBER()` funkciju za rad s podacima u prozoru. Ovom funkcijom podaci se privremeno (u prozoru) particioniraju po kolumni `state`, a zatim se unutar svake particije sortiraju silazno po stupcu `cases_population_perc`. Funkcija svakom retku dodjeljuje redni broj, a na taj način najvećem postotku u svakoj državi dodjeljuje se broj 1 u stupcu `rownm`. Budući da podaci tablice na razini retka prikazuju vrijednosti okruga, redak koji dobije broj 1 istovremeno predstavlja postotak vodećeg okruga svake savezne države. U zadnjem koraku, iz rezultata prethodnog CTE dohvaćaju se stupci `state`, `county` i `cases_population_perc`, a putem `WHERE` uvjeta dohvat je ograničen samo na retke kojima je u stupcu `rownm` dodijeljena vrijednost 1 (vodeći okrug po državi). Dohvaćeni podaci zatim se sortiraju silaznim redom, a budući da nas zanima postotak slučajeva oboljelih u odnosu na populaciju samo u vodećih pet država i njihovih okruga, na kraju upita postavlja se `LIMIT 5`.

Upit 2. Postotak cijepljenih stanovnika u prvih pet saveznih država s vodećim okrugom po zabilježenim slučajevima u populaciji

```
SELECT state,
       county,
       MAX(percent_vaccinated)
       AS perc_vaccinated
FROM covid_us_*
WHERE (state = 'Tennessee' OR
       state = 'Colorado' OR
       state = 'Kansas' OR
       state = 'South Dakota' OR
       state = 'Arkansas')
AND state||county IN ('TennesseeTrousdale',
'ColoradoCrowley', 'KansasNorton',
'South DakotaBon Homme','ArkansasLincoln')
GROUP BY state, county
ORDER BY perc_vaccinated
```

Rezultat provedbe upita 2:

#	state	county	perc_vaccinated
1	South Dakota	Bon Homme	33.0
2	Colorado	Crowley	39.0
3	Kansas	Norton	39.0
4	Arkansas	Lincoln	42.0
5	Tennessee	Trousdale	58.0

Obrazloženje upita 2:

Rezultati upita 1 pokazuju prvih pet država s vodećim okrugom po zabilježenim slučajevima u populaciji okruga. Upit 2 dohvaća podatke samo za ove države i okruge iz pojedine tablice nad kojom se upit provodi. Osim stupaca `state` i `county`, dohvaća najveći postotak cijepljenih stanovnika po okrugu koristeći agregatnu funkciju `MAX`. U uvjetu `WHERE` navode se savezne države čiji se podaci žele dohvatiti, na način da se nižu i povezuju logičkim operatorom `OR`. Postavljen je i dodatan uvjet nakon logičkog operatora `AND`, u kojem se operatorom konkatenacije || spajaju željene vrijednosti stupaca `state` i `county`. Spajanjem ovih dviju vrijednosti dobiva se jedinstvena kombinacija države i okruga. Ovaj dodatni uvjet naveden je budući da više saveznih država može imati iste nazive okruga. Primjerice, sve navedene države imaju okrug Lincoln, ali zanima nas samo Lincoln u saveznoj državi Arkansas. Budući da se ovom konkatenacijom dobiju potrebne vrijednosti država i okruga, prvi dio uvjeta `WHERE` na razini podataka može djelovati kao višak (upit bi vratio iste rezultate i bez navođenja `state` niza u `WHERE` uvjetu). Ipak, on je u ovom slučaju važan jer su podaci particionirani po ovom stupcu, a provedbom upita koristeći particije količina skeniranih podataka se smanjuje. Iz `WHERE` uvjeta su vidljiva i dva srodna načina navođenja stupaca u slučaju da se dohvaća veći broj vrijednosti: povezivanjem višestrukih vrijednosti operatorom `OR` ili nizanjem vrijednosti pomoću `IN ()`. U smislu optimizacije, zadnja naredba kojom se podaci uzlazno sortiraju (`ORDER BY`) mogla bi se i izostaviti, budući da rezultati upita predstavljaju manji skup vrijednosti za samo pet država i njihovih okruga.

Upit 3 (dvije inačice). Podaci o oboljenjima tijekom ljetnih mjeseci 2020.

```sql
-- a)
SELECT *
FROM covid_us_processed
WHERE CAST(date AS varchar) LIKE '%-06-%' OR
      CAST(date AS varchar) LIKE '%-07-%' OR
      CAST(date AS varchar) LIKE '%-08-%'
ORDER BY state, county, date
```

```sql
-- b)
SELECT date,
       state,
       county,
       cases,
       deaths,
       percent_vaccinated,
       percent_uninsured
FROM covid_us_processed
WHERE regexp_like (CAST(date AS varchar),
'-06-|-07-|-08-')
ORDER BY state, county, date
```

Obrazloženje upita 3a:

Ova inačica upita dohvaća sve stupce iz tablice covid_us_processed. Budući da nas zanimaju ljetni mjeseci (lipanj, srpanj i kolovoz), u `WHERE` uvjetu se navodi željeni segment datuma putem niza `LIKE` klauzula povezanih logičkim operatorom `OR`. Budući da `LIKE` uspoređuje izraz s tekstualnom vrijednošću, stupac `date` pretvara se u tip `varchar` putem `CAST` naredbe. Kako vrijednost datuma ima format yyyy-mm-dd, znakovni niz kojim se osigurava dohvat vrijednosti mjeseca uključuje srednji dio vrijednosti, zajedno sa ' - ' znakovima (`-mm-`). Znak '%' zamjenjuje bilo koje druge znakove, što u ovom slučaju znači da se dohvaćaju podaci mjeseca neovisno o tome koja godina i dan su zabilježeni u datumu (u ovom setu podataka to je samo 2020., ali ovim zapisom zanemaruje se godina općenito). Dohvaćeni podaci zatim se sortiraju uzlaznim redom, najprije po državi, pa okrugu i zatim datumu.

Obrazloženje upita 3b:

U ovoj inačici upita dohvaćaju se samo potrebni stupci, njih ukupno sedam. Umjesto niza `LIKE` klauzula, `WHERE` uvjet postavljen je koristeći `regexp_like` funkciju. Ova funkcija traži navedene vrijednosti u zadanom stupcu, a operator | (*pipe*) ima isto značenje kao operator `OR`. Za razliku od `LIKE`, ne traži podudaranje cijelog navedenog izraza s vrijednošću stupca, već provjerava sadrži li stupac u nekom dijelu navedeni izraz. Kombinacijom uzoraka bilo bi moguće detaljnije zadati na koji način se izraz traži u vrijednosti stupca (npr. na početku, kraju, prema broj ponavljanja itd).

Dio rezultata provedbe upita 3b (od ukupno 289977 redaka):

#	date	state	county	cases	deaths	percent_vaccinated	percent_uninsured
1	2020-06-01	Alabama	Autauga	234	5	41.0	8.7216859511
2	2020-06-02	Alabama	Autauga	240	5	41.0	8.7216859511
3	2020-06-03	Alabama	Autauga	240	5	41.0	8.7216859511
4	2020-06-04	Alabama	Autauga	242	5	41.0	8.7216859511

Upit 4 (četiri inačice). Vremenski podaci saveznih država: datum prvog i zadnjeg slučaja, raspon dana bilježenih slučajeva, zadnji dostupan omjer (%) slučajeva u populaciji na zadnji datum podataka

```
-- a)
WITH cases_data AS (
SELECT state
     , cases
     , date
     , total_population
     , FIRST_VALUE(date) OVER (PARTITION BY
state ORDER BY date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS first_date
     , LAST_VALUE(date) OVER
(PARTITION BY state ORDER BY date ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING)
AS last_date
FROM covid_us_processed)
SELECT state
     , first_date
     , last_date
     , ABS(DATE_DIFF ('day',
from_iso8601_date(last_date),
from_iso8601_date(first_date))) AS dates_range
     , ROUND(CAST(SUM(cases) AS
double)/SUM(total_population)*100, 2) AS
latest_cases_population_perc
FROM cases_data
WHERE date = last_date
GROUP BY state, first_date, last_date
ORDER BY dates_range DESC
```

```
-- b)
WITH cases_data AS (
SELECT state
     , cases
     , date
     , total_population
     , FIRST_VALUE(date) OVER (PARTITION BY
state ORDER BY date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS first_date
     , LAST_VALUE(date) OVER
(PARTITION BY state ORDER BY date ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING)
AS last_date
FROM covid_us_processed)
SELECT state
     , first_date
     , last_date
     , ABS(DATE_DIFF ('day',
from_iso8601_date(last_date),
from_iso8601_date(first_date))) AS dates_range
     , ROUND(CAST(SUM(cases) AS
double)/SUM(total_population)*100, 2) AS
latest_cases_population_perc
FROM cases_data
WHERE date = last_date
GROUP BY state, first_date, last_date
ORDER BY dates_range DESC
LIMIT 10
```

```
-- c)
WITH cases_data AS (
SELECT state
     , cases
     , date
     , total_population
     , FIRST_VALUE(date) OVER (PARTITION BY
state ORDER BY date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS first_date
     , LAST_VALUE(date) OVER
(PARTITION BY state ORDER BY date ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING) AS
last_date
FROM covid_us_processed)
SELECT state
     , first_date
     , last_date
     , ABS(DATE_DIFF ('day',
from_iso8601_date(last_date),
from_iso8601_date(first_date))) AS dates_range
     , ROUND(CAST(SUM(cases) AS
double)/SUM(total_population)*100, 2) AS
latest_cases_population_perc
FROM cases_data
WHERE date = last_date
GROUP BY state, first_date, last_date
```

```
-- d)
WITH cases_data AS (
SELECT state
     , cases
     , date
     , total_population
     , FIRST_VALUE(date) OVER (PARTITION BY
state ORDER BY date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS first_date
     , LAST_VALUE(date) OVER
(PARTITION BY state ORDER BY date ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING) AS
last_date
FROM covid_us_processed
WHERE state IN ('California','Texas','Florida'))
SELECT state
     , first_date
     , last_date
     , ABS(DATE_DIFF ('day',
from_iso8601_date(last_date),
from_iso8601_date(first_date))) AS dates_range
     , ROUND(CAST(SUM(cases) AS
double)/SUM(total_population)*100, 2) AS
latest_cases_population_perc
FROM cases_data
WHERE date = last_date
GROUP BY state, first_date, last_date
```

Obrazloženje upita 4a:

Prvi korak upita je CTE `cases_data`, koji dohvaća željene stupce i dvije dodatne vrijednosti (`first_date`, `last_date`) dobivene koristeći funkcije za rad s podacima u prozoru. Funkcija `FIRST_VALUE()` vraća prvu vrijednost u retku prozora, nakon što se podaci particioniraju po stupcu `state` i sortiraju uzlazno po datumu. Particioniranje u ovom slučaju omogućava dobivanje podatka o prvom datumu za svaku državu zasebno. Funkcijom `LAST_VALUE()` na jednak način dobiva se zadnji datum, a ove dvije funkcije međusobno se razlikuju u klauzulama koje definiraju način na koji se okvir promatranih redaka pomiče u prozoru. U sljedećem koraku dohvaćaju se stupci iz prethodnog CTE, zajedno s dvije nove vrijednosti koje se izračunavaju: `dates_range` koji predstavlja raspon dana od prvog do zadnjeg zabilježenog slučaja, i `latest_cases_population_perc` kojim se dobiva zadnji postotak oboljelih u odnosu na populaciju na razini savezne države. Za vrijednost `dates_range` koristi se više postupaka. Kako su `last_date` i `first_date` vrijednosti tipa `varchar`, kako bi se nad njima mogla primijeniti datumska funkcija `DATE_DIFF`, najprije se primijenjuje funkcija `from_iso8601_date`. Ova funkcija pretvara *string* tip podatka u datum. Koristeći `DATE_DIFF` zatim se računa razlika između dvije datumske vrijednosti (zadnji dan – prvi dan), a razlika se izražava u jedinici dana (`'day'`). Zadnje, kako bi se osiguralo da razlika datuma uvijek bude pozitivna vrijednost, primijenjuje se numerička funkcija `ABS` za dobivanje apsolutne vrijednosti raspona. Zadnja vrijednost, `latest_cases_population_ perc`, računa se na sličan način kao `cases_population_perc` iz upita 1. Razlika se ovdje sastoji u tome da se na stupcima `cases` i `total_population` računa zbroj putem agregatne funkcije `SUM`, budući da su u retcima izraženi podaci okruga, a zanima nas stanje na razini savezne države. U `WHERE` klauzuli postavlja se uvjet za uzimanje podataka na zadnji datum, nakon čega se podaci grupiraju, pa sortiraju po `dates_range`.

Obrazloženje upita 4b:

U upit 4a dodana je naredba `LIMIT`, rezultat je 10 država s najdužim periodom bilježenih slučajeva.

Obrazloženje upita 4c:

Iz upita 4a izostavljena je naredba `ORDER BY`.

Obrazloženje upita 4d:

U upit 4c dodana je `WHERE` klauzula sa stupcom `state`, budući da se promatraju podaci samo tri najmnogoljudnije savezne države: Kalifornije, Teksasa i Floride.

Dio rezultata provedbe upita 4a (od ukupno 54 retka):

#	state	first_date	last_date	dates_range	latest_cases_population_perc
1	Washington	2020-01-21	2020-12-04	318	2.58
2	Illinois	2020-01-24	2020-12-04	315	6.0
3	California	2020-01-25	2020-12-04	314	3.41
4	Arizona	2020-01-26	2020-12-04	313	5.28