UNIVERZITET U NIŠU  
ELEKTRONSKI FAKULTET  
KATEDRA ZA RAČUNARSTVO

**PLANETSCALE KAO PRIMER DATABASE-AS-A-SERVICE REŠENJA**

Student:  
Dimitrije Stamenković 1495

**Sadržaj:**

[Uvod 2](#_Toc992499559)

[Istorija 2](#_Toc602170122)

[PlanetScale i Vitess 2](#_Toc1471828929)

[Cena 4](#_Toc1378779638)

[Tok razvoja kod PlanetScale-a 4](#_Toc417991889)

[PlanetScale i grane (branching) 4](#_Toc607399807)

[Regioni 10](#_Toc652308509)

[Strani ključ i ograničenja 11](#_Toc1043327373)

[Strategije za očuvanje integriteta referenciranja 14](#_Toc2053568336)

[Backup 16](#_Toc1211110967)

[Migracija baze podataka 18](#_Toc138101940)

[Literatura 20](#_Toc689498407)

# Uvod

PlanetScale predstavlja MySQL kompatabilnu serverless bazu podataka. Planetscale pruža horizontalni sharding, ne blokirajuće izmene šeme baze podataka i još puno funkcija o kojima će biti reči u nastavku rada.



Slika 1: PlanetScale logo

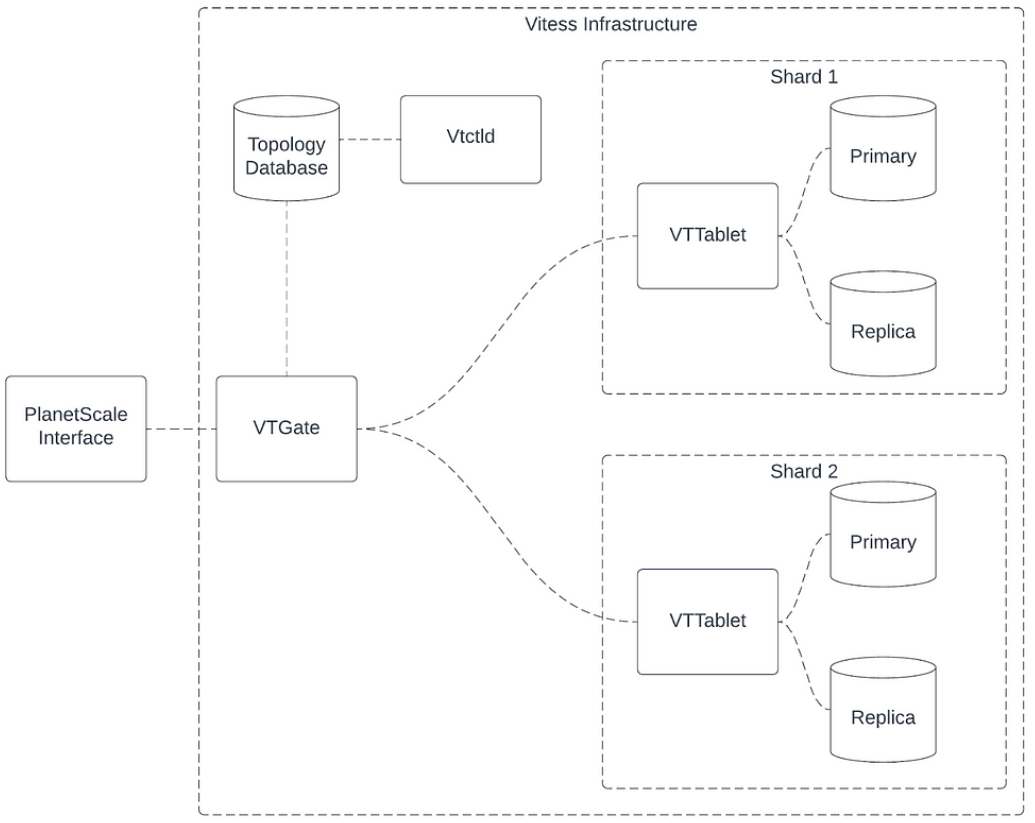
## Istorija

PlanetScale je iza kulisa pokrenuta preko Vitess open source baze podataka. Vitess baza podataka je osmišljena u YoutTube-u 2010. godine kako bi rešila problem skaliranja MySQL baze podataka koju je u to vreme koristio Youtube. Vitess ubrzo nakon toga postaje open-source baza podataka kao i deo CNCF pojekta *(Cloud Native Computing Foundation)* koju i dan danas koriste mnoge kompanije poput Slack-a, GitHub i drugih kako bi lako skalirali svoje baze.

Vitess od osnivanja predstavlja izuzetnu bazu podataka, ali sa druge strane ona je kao baza veoma komplikovana za implementaciju i održavanje bez velikog tima. Jedan od kreatora Vitessa u želji da bazu učini dostupnu svima, osniva svoj projekat pod imenom PlanetScale koji kao što smo pomenui ima Vitess kao temlj. Samim tim svaka baza pokrenuta u PlanetScalu bila ona plaćena ili besplatna pokreće Vitess ispod haube.  
  
Cilj PlanetScale-a jeste smanjiti gap između aplikacije i baze podataka tako da programeri više ne gledaju na ta dva kao odvojene celine gde bi samo odabrani ljudi mogli razvijati bazu. Umesto toga teži se opet smanjiti taj gap gde bi svi mogli dali doprinos oko razvijanja baze podataka. Bitna stvar je i pouzdanost odnosno vreme kada baza ne radi i gubljenje podataka, ti pojmovi kod PlanetScale-a predstavljaju relikt prošlosti. PlanetScale se skalira bez problema, bez obzira na obim podataka.

## PlanetScale i Vitess

Kada se kreira baza podataka u PlanetScale-u, ustvari se kreira ceo *Vitess* cluster. U tom klasteru nalaze se sve komponente koje Vitess zahteva kao na primer: *VTTablets* i *VTGate.* Konekcija prema bazi se ostvaruje preko stingova konekcije. Ispod vidimo arhitekturu *Vitess* klastera.



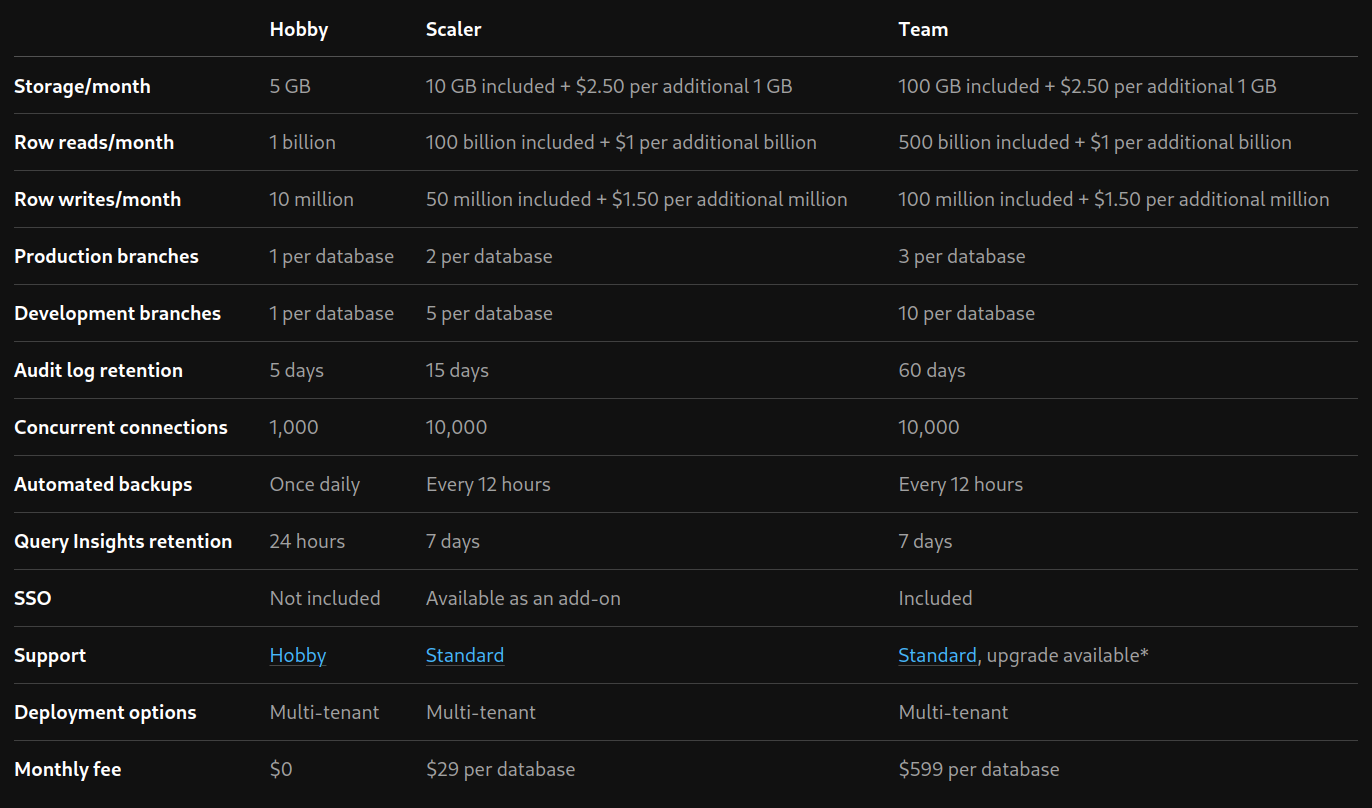
Slika 2: Arhitektura *Vitess* klaster

Sa slike vidimo da klaster sadrži *VTGate, Topology Database, vtcld* instancu i dva šarda od kojih svaki sadrži *VTTablet* i dve baze podataka. *VTGate* predstavlja jednostavan proxy koji usmerava MySQL upita na odgovarajuću bazu podataka čime se smanjuje gap između PlanetScale-a i šardova.   
  
Benifit šardova jeste mogućnost imanja više replika ( primarnih i sekundarnih), koji omogućavaju bolju geografsku distribuiranost uz mogućnost podešavanja hardvera na osnovu opterećenja. Na primer, upisi mogu ići na primarnim instancama dok čitanja mogu biti sa sekundarnih replika. PlanetScale ovu mogućnost pruža kao ugrađenu (out of box) dovoljno je samo uključiti istu kod podešavanja baze.

PlanetScale u teoriji može opslužiti neograničen broj konekcija, to se postiže udruživanjem (pooling) konekcija čime se smanjuje i opterećenje baza. Uz to Vitess komponente su napisane u programskom jeziku programskom *GO*, koga odlikuju velika brzina u vidu konkurentnosti izršenja operacija (goroutine).  
  
Kod date arhitekture vidimo da jedino *VTTablet* komponenta ima direktnu vezu sa MySQL bazom, njega možemo smatrati api-jem prema bazi odnosno sidecar-om ukoliko koristimo kubernetes terminologiju. On dobija upite preko gRPC protokola prevodi i prosleđuje iste MySQL bazi. Osim toga upravlja i brojem replika, izvršava backup, prekidate upite koje traju dugo, beležiti stanje baze i ostalo.  
  
*VTGate* rutira saobraćaj prema bazi odnosno vraća rezulat izvršenja nazad. Klijent se konektuje na *VTGate* komponentu kao da se radi o jednoj instanci MySQL baze, iako se u pozadini može raditi o više instanci.

## Cena

PlanetScale kao servis koji pruža usluge korišćenja baze podataka u osnivi nudi četiri različita plana: Hobby, Scaler, Team i Enterprice. Svi paketi se naplaćuju po bazi podataka, a ukupna cena zavisiće o korišćenju samih baza. Hobby plan je bestplan i uključuje 5gb namenjih za podatke, 1 milijardu čitanja podataka (mesečno), 10 miliona upisa u bazu (mesečno) i dve grane (main i dev) namenjene praćenju verzija šeme baze podataka. Scaler plan košta 29 dolara mesečno po bazi podataka, a uključuje sve stavke iz Hobby plana, uz dodatnu memoriju i više upisa/čitanja. Cena se povećava za svaki dodatni gigabajt memorije i milijardu čitanja odnosno milion upisa. Scaler plan takođe uključuje dve grane za produkciju i pet za razvoj, ali i opciju za SSO. Team plan nudi veću memoriju i više čitanja/upisa. Kao i više grana za versionisanje. Enterprise plan nudi prilagođene cene i pogodnosti za organizacije koje imaju posebne zahteve. Kao što možemo videti na slici ispod hobby paket jako puno nudi ako uzmemo u obzir da je besplatan.



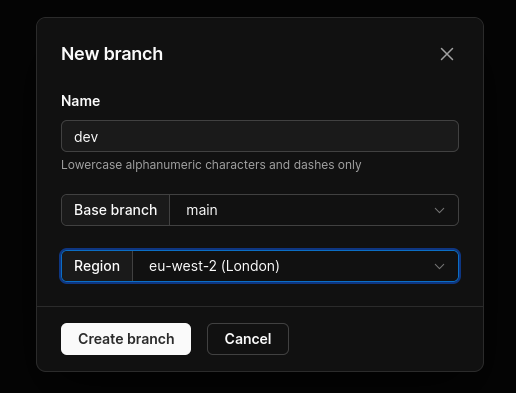
Slika 3: PlanetScale paketi i cene

# Tok razvoja kod PlanetScale-a

Obzirom da PlanetScale ima opciju granjanja tok razvoja šeme baze podataka je nešto drugačiji.

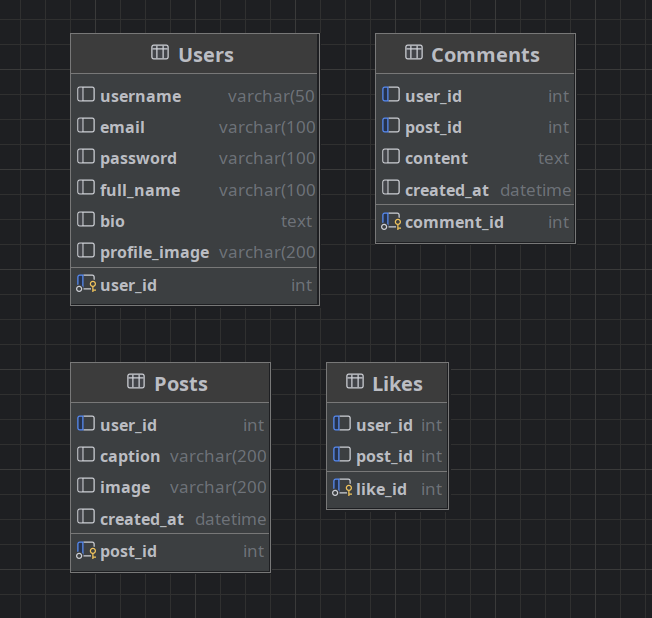
## PlanetScale i grane (branching)

Grananje je funkcija koja nam omogućava da kreiramo izolovane kopije šeme produkcijske baze podataka koje bi služile kao izolovano okruženje za razvoj. Ova funkcija pruža fleksibilnost prilikom razvoja aplikacije i omogućava nam eksperimentisanje sa promenama bez uticaja na produkcijsko okruženje. PlanetScale u besplatnom paketu pruža dve vrste grana baze podataka: razvojno (dev) i produkcijsko (main). Razvojne (dev) grane pružaju izolovane kopije šeme produkcijske baze podataka gde možemo praviti promene, eksperimentisati ili pokretati CI. Produkcijske grane su visoko dostupne grane koje uglavnom uključuju dodatne replike. Takođe postoji opcija uključivanja sigurnih migracija, što omogućava neblokirajuće promene šeme što dalje štiti bazu podataka od slučajnih promena šeme.Kada se kreira novu grana iz produkcijske grane, razvojna grana će imati istu šemu kao i produkcijska, ali neće kopirati nikakve podatke iz produkcijske baze podataka. PlanetScale predlaže da se razvojne grane popune lažnim podacima. Kada se radi promovisanje grana u produkcijsku, uvek možemo kreirati nove grane (izolovane kopije šeme produkcijske baze podataka) iz produkcijske za nastavak razvoja.



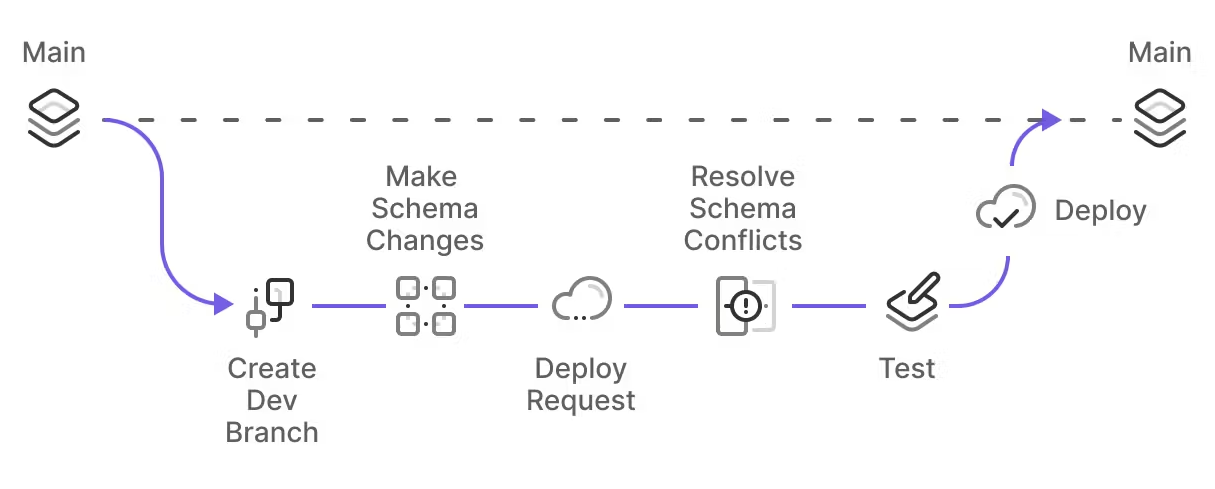
Slika 4: Kreiranje nove dev grane iz već postojeće main

Pre samog primera korišćenja grana postavićemo šemu podataka. Dijagram šeme baze podataka dat je na slici ispod. Radi se o šemi baze podataka Instagram aplikacije u kojoj imam četiri tabele: Users, Comments, Posts i Likes. Primećujemo da svaka od tabela poseduje samo *primary key*, ali ne i *foreign key* preko koga se inače ostvaruju relacije. Razlozi za tim biće objašnjeni u sekciji vezanoj za *foreign key* i očuvanje integriteta baze podataka.



Slika 5: Šema baze podataka

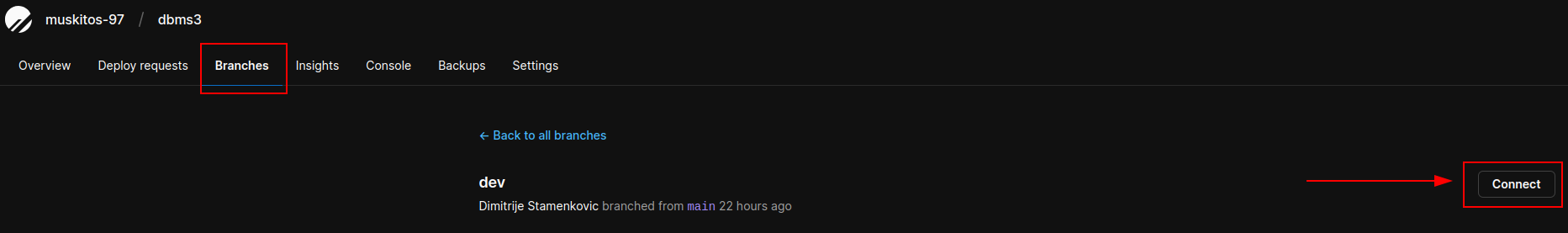
Nakon inicijalnog postavljanja baze koje smo izvršili u grani ***main*** možemo postupkom datim na slici 4 kreirati novu granu ***dev*** koja će u startu imati identičnu šemu poput prethodno kreirane u **main** grani.



Slika 6: Proces menjanja šeme putem dev grane

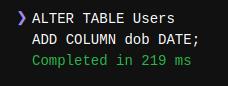
Prvi korak u procesu datom na slici iznad (Slika 6) jeste kreiraranje nove grane ***dev*** što je opisano na slici 4.

Nakon toga slobodni smo izvršiti sve planirane promene, kako se radi o dev grani originalna šema baze podataka neće biti ugrožena.  
  
Kao primer izmene dodaćemo dodatni atribut nekoj od tabela u našoj bazi podataka. To činimo konekcijom na ***dev*** granu, najlakši način zato jeste web ui konzola pre koje ćemo se i mi konektovati. Iz gornjeg menija biramo opciju ***branches*** a potom i opciju ***connect*** koja nas povezuje sa bazom. Konekcija se mogla ostvariti i preko opcije ***console*** iz gornjeg menija.

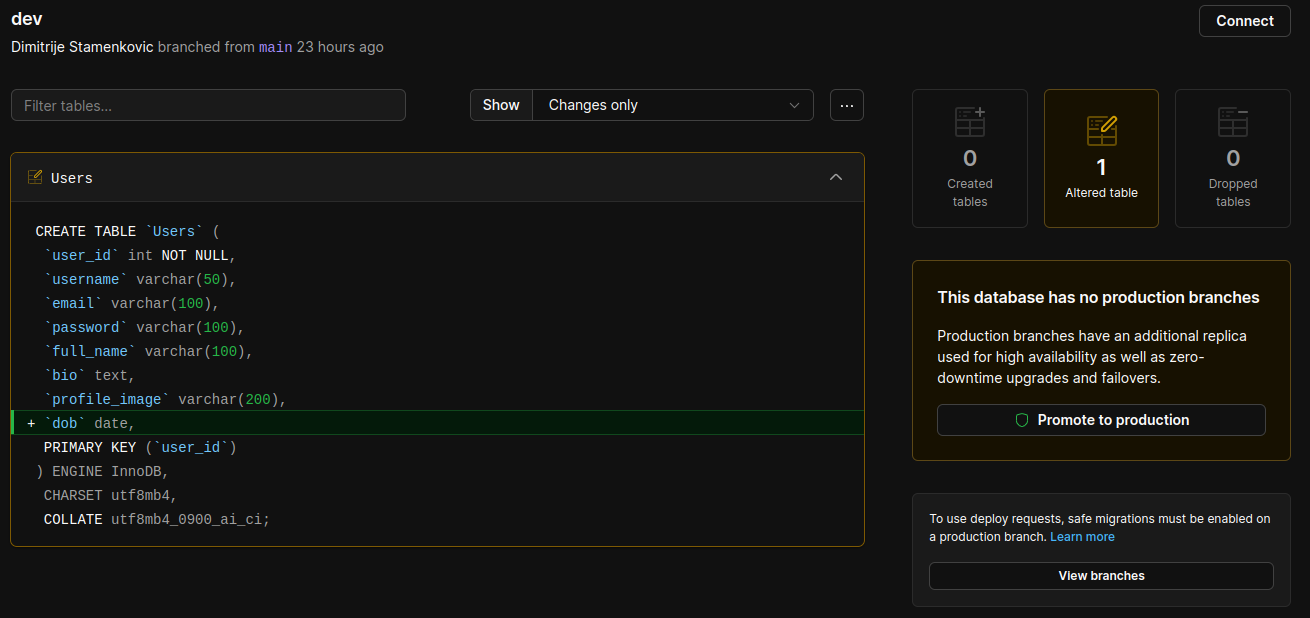


Slika 7: Konekcija ka bazi preko web ui-a

Izmenićemo Users tabelu dob (date of birth) atribut. To radimo komandom datom na slici ispod.

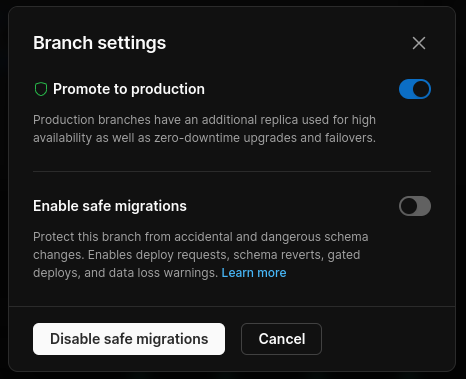
  
Slika 8: Izmena nad tabelom Users

Ukoliko se sada vratimo na ***branches*** opciju iz gornjeg menija primetićemo da je promena prepoznata. Na slici ispod vidimo da je linija sa našom izmenom dodata ali i sa desne strane da je samo jedna tabela promenjena.



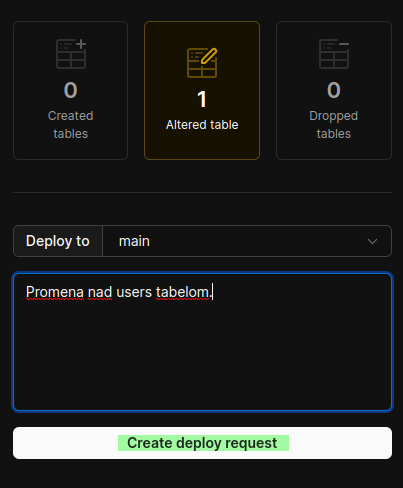
Slika 9: Uočena izmena

Sledeći korak sa slike 5 je ***deploy request*** korak gde šaljemo zahtev za spajanje sa main granom. Pre toga potrebno je aktivirati opciju ***Promote to production***, time kreiramo granu namenjenu produkciji koja je zasebna grana. Prilikom promocije, biramo main granu koju želimo kao granu iz koje povlačimo šemu. Osim toga potrebno je i uključiti Enable safe migrations opciju unutar produkcije grane čime obezbeđujemo da produkcijska grana bude zaštićena od slučajnih promena, ovim omogućavamo sledeće opcije: *deploy requests*, *schema reverts*, *gated deploys* i opomene na gubitak podataka.

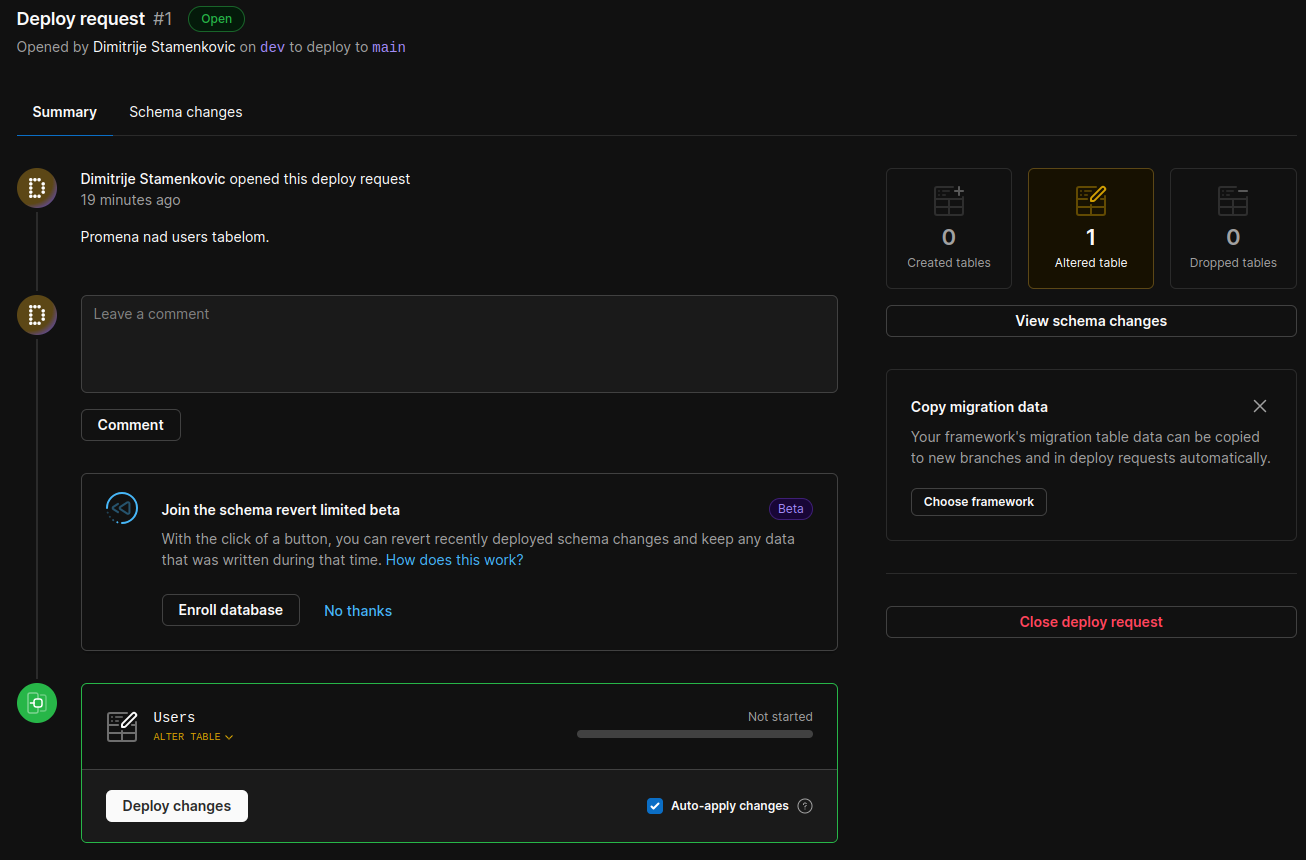


Slika 10: Uključivanje enable safe migrations opcije

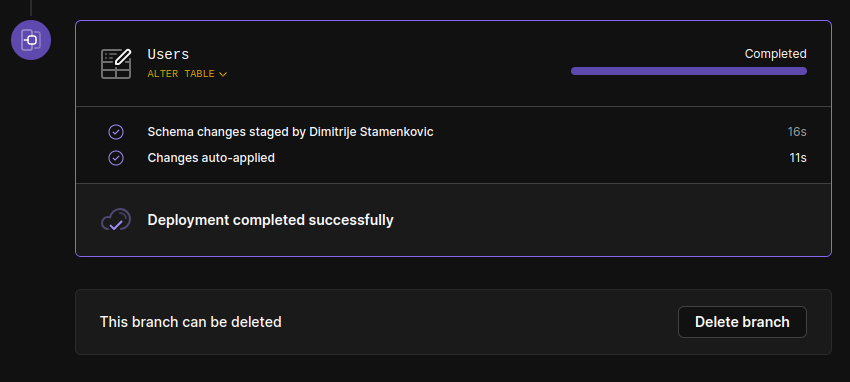
Povratkom na prozor sa slike 9, vidimo da sada imamo opciju da napravimo deploy request uz opis izmene. Ukoliko smo sigurni da želimo da promovišemo izmene koje smo prethodno napravili klikom na dugme ***Create deploy request*** kreiramo zahtev za izmenu na main grani.

  
Slika 11: Deploy request

Kako bismo potvrdili Deploy request iz gornjeg menija sada biramo opciju ***Deploy requests****,* tu vidimo sve otvorene deploy requeste pod tim misli se na zahteve koji su i dalje na čekanju i nisu sjedinjeni sa main granom (slika 11).

Slika 12: Prethodno kreirani Deploy request

Sam zahtev ne mora odmah biti prihvaćen odnosno odbijen jer postoji opcija da komentarisanja gde neko od iskusnijih kolega može proveriti izmene i dati komentar šta još treba promeniti. Ukoliko je sa druge strane pak sve uredu klikom na dugme ***Deploy changes*** spajamo dev i main granu čime se okončava deploy request.



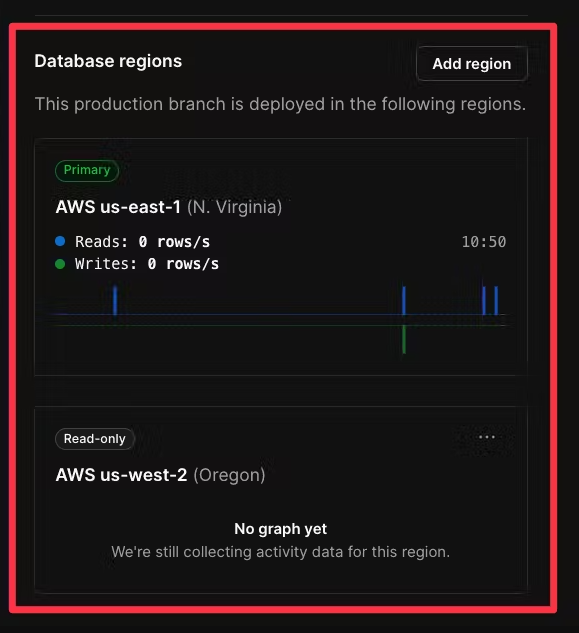
Slika 13: Uspešno izvšen deploy request

Važno je napomenuti da neće uvek biti zahtev uspešno izvršen, nalik na git i github može doći i do konflikta u šemama, tada je potrebno razrešiti prvo konflikte pa tek onda izvršiti deploy. Ukoliko sada pogledamo šemu baze podataka, konretno izmenjenu tabelu vidimo da je zaista došlo do izmene. Na kraju možemo i izbrisati granu nakon zahteva. Poželjno je imati granu po requestu, ali obzirom da je u besplatnom paketu dostupna samo jedna dev grana potrebno je improvizovati.

Sem kriranja grana namenjenih šemi baze podataka uskoro će biti dostupno kreirati i grane namenjene podacima. Na primer, dev i main grana bi imale skroz različite podatke namenjene svojim okruženjima. Ova funkcija je u beta statusu, dakle i dalje je razvoju.

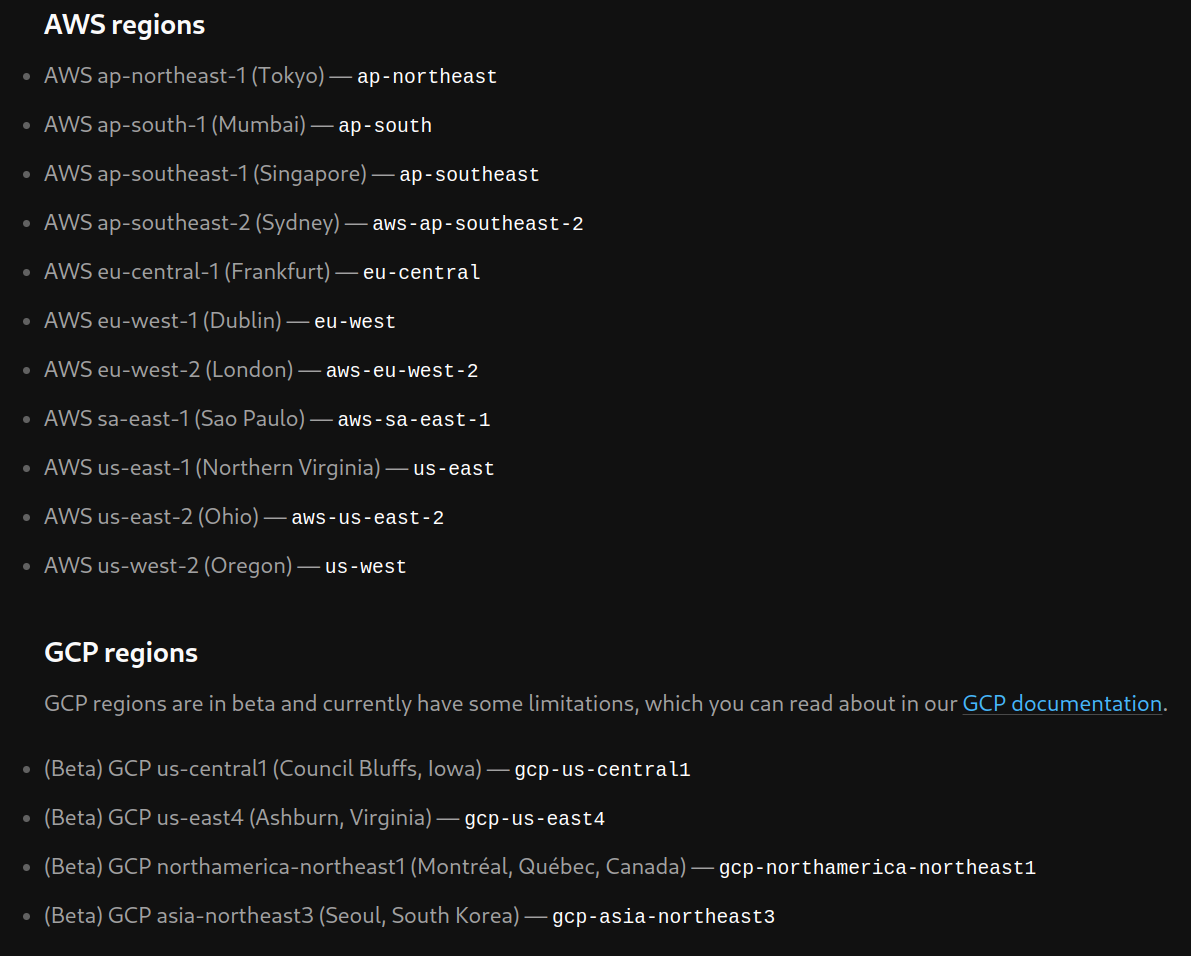
## Regioni

PlanetScale u ponudi ima više lokacija na koje možemo postaviti baze. Osim toga možemo izabrati posebnu lokaciju za svaku od grana. Po pravilu poželjno je uvek birati lokaciju što bliže serverima na kojima se nalazi aplikacija. Produkcijska grana ima i opciju ***read-only*** regiona. *Read-only* regioni predstavljaju replike produkcijske baze na različitim lokacijama obično geografski udaljenim. U hobby paketu koji trenutno koristimo dostupna je samo jedna lokacija, tj. nema read-only replika. Svakako, dodavanje dodatne *read-only* replike nije teško potrebno je izabrati produkcijsku granu (kod nas main), i izabrati opciju ***add region***. Replikacija se obavlja po asihronoj strategiji, dakle promene se prvo izvršavaju na primarnoj bazi ( primarni region). Onda se update propagira ostalim read-only replikama, vreme potrebno za to označeno je kao replikacioni lag koji možemo dobiti izvršavanjem ***SELECT max\_repl\_lag()*** komande u terminalu. Ta funkcija će vratiti maksimalno vreme potrebno da se ažuriranje postavi na nekoj od replika.



Slika 14: Dodavanje dodatnog regiona za read-only repliku

PlanetScale u ponudi trenutno nudi servere sa AWS-a ondosno Googl-ovog GCP-a. Na slici ispod vidimo sve trenutno dostupne servere. Ukoliko se javi potreba za enterprise zahtevima i specifičnim lokacijama, kupac se može obratiti osoblju PlanetScale-a kako bi došli do rešenja.



Slika 15: Dostupni regioni

## Strani ključ i ograničenja

Na slici 5, sa dijagrama naše šeme generisane u DataGrip aplikaciji vidimo da ne postoje veze između tabela. Razlog tome je što u PlanetScale-u nije dozvoljeno korišćenje stranog ključa *(foreign key)* za referenciranje klučeva iz drugih tabela. Zbog čega je to tako i kako PlanetScale rešava pitanje integriteta baze podataka biće objašnjeno u nastavku ove celine.

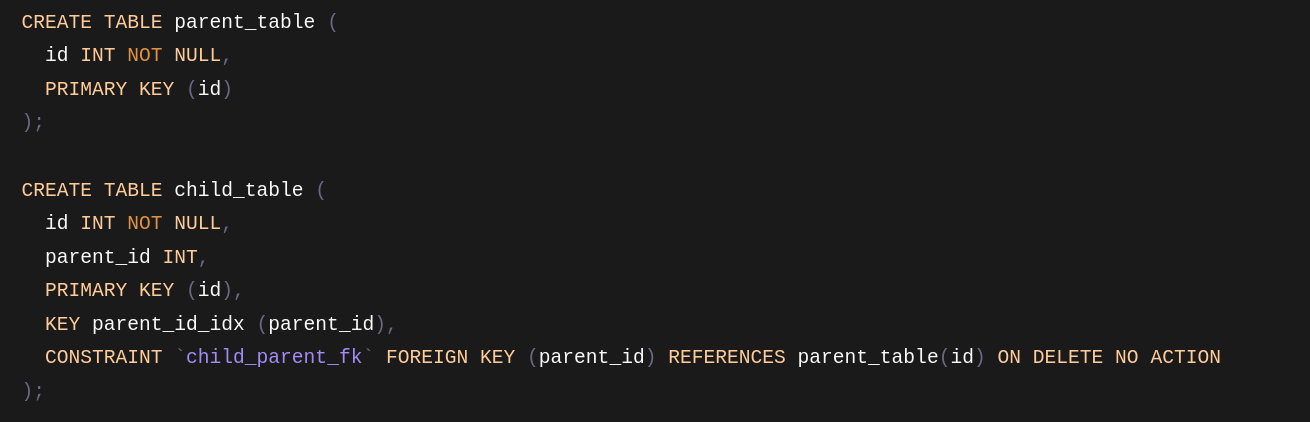
Strani ključ predstavlja povezanost atributa među dveju tabela, možemo reći roditelj-dete relacija između tabela. Atribut u tabeli roditelja može biti referenciran od strane jednog ili više atribute u tabeli deteta. Strani ključ uglavnom definiše i korišćenje join-ova kod pisanja sql naredbi. Tabela može referencirati i samu sebe u specijalnim slučajevima.

Strani ključ *(foreign key)* je sastavni deo skoro svakog sql dbms-a, i predstavlja implementaciju koja forsira integritet među relacijama. Naime, strani ključ osigurava da dete tabela može referencirati tabelu roditelja samo ukoliko atribut koji je referenciran postoji unutar te tabele (roditelja). To ograničenje doprinosi i da ne postoje “orphaned rows” odnosno slučajevi kada atribut referencira atribut iz druge tabele koji ne postoji. Online DDL *(Data Definition Language)* dolazi kao rešenje problema migracija šema baze podataka koje su po pravilu bile blokirajuće. Izvršavanje proste komande **ALTER TABLE** u produkcijiznačilo bi zaključavanje tabele, veliki load na primarnoj instanci a potom i veliki lag prilikom ažuriranja replika.

PlanetScale ne koristi foreign key ograničenja zbog:

* Način na koji su ona implementirana u MySQL-u *(InnoDB storage engine)* nije zadovoljavajuć, u konfliktu je sa Online DDL-om.
* Foreign key je limitiran po pitanju opsega (scope), pa je tako moguće isti upotrebljavati samo u opsegu jedne instance. Rastom količine podataka brzyo se dolazi u situaciju da je potreban sharding ili particionisanje. Nemoguće je održati ograničenja jednom kada se počne sa više database servera.

Stav PlanetScacele-a je da Online DDL ima mnogo više prednosti kao što su *branching, non blocking schema changes, sharding, unlimited scalling* i mnoge druge poprilično nadmašuju ograničenja koja foreign key pruža. Drugim rečima PlanetScale forsira integritet na aplikativnom nivou umesto na database nivou čime pruža čitav spektar novih pogodnosti.



Slika 16: Tradicionalna upotreba foreign key-a

Na slici iznad vidimo poznatu šemu sa prisustvom foreign key-a. Foreign key ograničenja mogu uticati na brisanje vrste (ON DELETE), update vrsta (ON UPDATE). U nastavku više čemo obratiti pažnju na ON DELETE zbog većeg uticaja.

ON DELETE CASCADE

Radi se o tipu koji koji pokušava da obriše što više može. Ukoliko bi obrisali jednu vrstu iz roditeljske tabele, bilo koja vrsta iz tabele potomka biće isto tako obrisana u istoj transakciji. Ova operacija izvršava se rekurzivno za sve potomke roditelja, kao i za potomke potomka.  
  
On delete casace predstavlja riskantnu operaciju koja je i na resursima teška. Sa željom da obrišemo jednu vrstu lako bismo se mogli naći u situaciji da smo obrisali stotine ili hiljade vrsti u više tabela. Iako se čini kao jednostavna transakcija ispada kao veoma opasna i riskantna operacija koja uključuje zaključavanja implikaciju na replikacioni lag i mnoge druge stvari. Svakako najveći minus ove operacije jeste mogući odnosno nekontrolisani gubitak podataka. Ukoliko bi programer izvršio prosti DELETE FROM parent\_table WHERE id=3 ili REPLACE INTO upit koji u pozadini zapravo izvršava implicitni DELETE, shvatamo da ovo potencijalno vodi u ogroman gubitak podataka. Zbog ovoga se izbegava korišćenje ove operacija i zbog toga se stavlja samo NO ACTION.

ON DELETE SET NULL

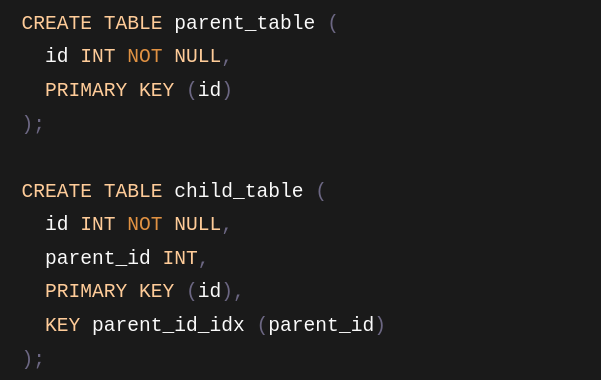
Brisanje iz roditeljske tabele (DELETE FROM parent\_table WHERE id=3) uz konfiguraciju (ON DELETE SET NULL) će u tabeli potomka postaviti null za reference prema tom roditelju. Ovaj pristup eventualno vodi do pojave vrste siročića (orphaned rows), što u suštini nije ništa bolje od neimanja ograničenja stranog ključa uopšte.  
  
Jedina prednost ovog pristupa je jednostavnost otkrivanja vrsta koje nigde ne pokazuju (orphaned rows) jer će samo te vrste imati NULL za vrednost kolona koje pokazuju kao tabeli roditelja.

Slično CASCADE, brisanje jedne vrste može potencijalno izmeniti mnogo vrsta u tabelama potomka što dalje uzrokuje veliki broj transakcija, zaključavanja i replikacioni lag.

ON DELETE NO ACTION

Ovde se radi o možda i najboljoj karakteristici ograničenja stranog ključa. Sada DELETE operacije nad vrstom u tabeli roditelja neće uspeti ukoliko postoji referenca na tu vrstu od strane potomaka. Kako bismo obrisali tu vrstu, potrebno je prvo obrisati sve referencirajuće vrste iz potomaka pa tek onda obrisati vrstu iz tabele roditlja. Isto važi i ukoliko potomci potomaka imaju referencu i te vrste moraju biti obrisane.

Ova karakteristika ( ili nedostatak nje) tera aplikaciju da ista ima veću kontrolu nad podacima. Aplikacija programirana da radi sa ON DELETE NO ACTION neće imati problema sa skaliranjem zato što od početka morati znati koje tabele referenciraju koje. DELETE/UPDATE operacije će od starta znati pravilni redosled iteracija u cilju očivanja integriteta referenci.

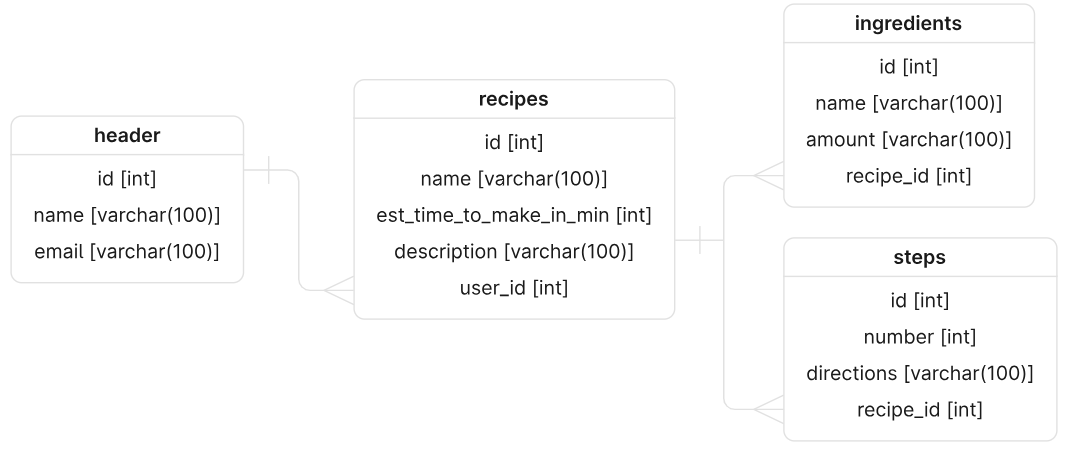


Slika 17: Šema bez upotrebe foreign key-a

Na slici iznad vidimo tabelu šeme sa slike , ali bez korišćenja stranog ključa. Obzirom da ON DELETE NO ACTION ograničenja sada nema, potrebno je nekako očuvati integritet referenciranja. Recimo da ga je aplikacija programirana sa ON DELETE NO ACTION ponašanjem u vidu. Bez tog ograničenja, ukoliko aplikacija izvrši DELETE FROM parent\_table WHERE id=3, aplikacija će prvo obrisati vrstu sa referencom a potom i vrstu u tabeli roditelja jer i sa ON DELETE NO ACTION opcijom morali bi sami voditi računa o referencama. Dakle sa jedne strane gubimo sigurnost koja se ogleda u tome da se da striktno ne možemo obrisati vrstu a da pre toga ne obrišemo sve vrste koje imaju referencu na istu, s druge strane i sami možemo projektovati aplikaciju u skladu sa očuvanjem integriteta referenciranja. Aplikacija koja se od starta prilagodi na šemu baze podataka bez ograničenja nema problema sa delete i update operacijama i nakon skaliranja. Data je sloboda u redosledu izvršenja operacija, nije uvek potrebno da istoj transakciji obrišemo sve vrste koje referenciraju našu vrstu u roditeljskoj tabeli. Aplikacija može odložiti to brisanje.

### Strategije za očuvanje integriteta referenciranja

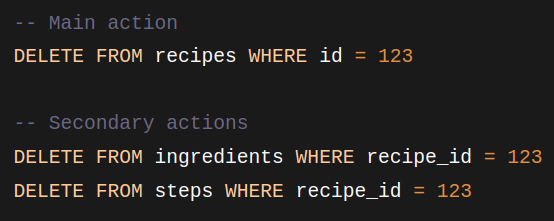
Obzirom da PlanetScale nema podršku za strani ključ, kaskadne akcije moraju se ručno (u kodu) implementirati umesto uobičajenog da baza podataka radi to za nas. U ovom poglavlju obradićemo najbolje strategije za očuvanje integriteta. Za primer uzećemo šemu podataka namenjenu receptima za kuvanje koju možemo videti na slici ispod.



Slika 18: Šema baze podataka o receptima za kuvanje

Inline operacije

Prva metoda jeste konkretno specificirati operacije koje trebaju biti izvršene kako bi nadomestile kaskadne operacije. Na primer, brisanje recepta preko API-ja, možemo modifikovati route handler tako da obavlja i sekundarne operacije (kaskadne operacije).

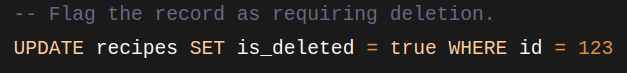


Slika 19: Primer inline operacija

Prednost ovog prilaza jeste konkretnost i konciznost, dok je mana da ako se radi u slučajevima gde je potrebno brisati dosta toga može se desiti zagušenje gde bi korisnici čekali malo duže na izmene.

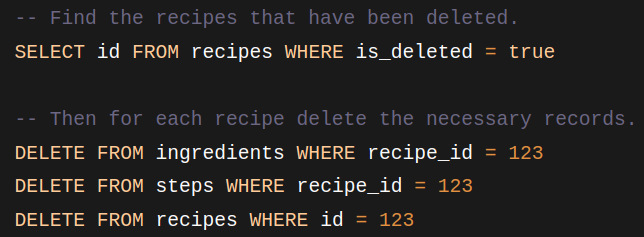
[Scheduled jobs](https://planetscale.com/docs/learn/strategies-for-maintaining-referential-integrity" \l "scheduled-jobs)

Drugi prilaz problemu jeste da baza podataka biva skenirana periodično sa ciljem pronalaženja da li neka kaskadna akcija treba da se izvrši. Na primer, mogli bismo postavili cronjob koji bi pronalazio vrste koje su obrisane (soft-deleted), a potom i izvršio brisanje referenci i na kraju obrisao soft obrisanu vrstu. Soft delete predstavlja kolonu u šemi.



Slika 20: Soft delete

Scheduled task (cronjob) bi izgledao kao na slici ispod, dakle uzeli bismo prvo id od vrsta koje su obeležene za brisanje (is\_deleted = true), nakon toga bismo obrisali vrste iz drugih tabela koje ukazuju na tu vrstu da bi na kraju obrisali i sam recept.

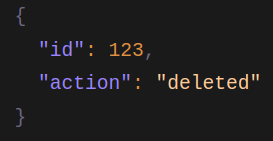


Slika 21: Primer periodične skripte

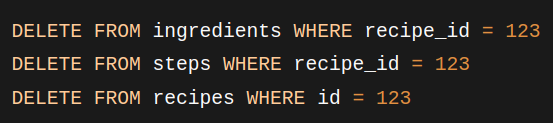
Glavni benefit ovog prilaza je brzina, obzirom da u samoj implementaciji api-ja radimo samo update korisnik odmah može videti rezultate ažuriranja bez čekanja da prođu sve sekundarne (kaskadne) operacije. S druge strane negativna stvar jeste dodatni *overhead* jer mora da postoji dodatni sistem koji će pratiti update promene i delovati u skladu sa tim. Drugi minus jeste što ovom strategijom protekne neko vreme u kojoj baza podataka i dalje ima kaskadne reference koje nigde ne pokazuju.

Asihroni update korišćenjem redova

Poslednja i uglavnom najbolja strategija je korišćenje nekog od message brokera tipa RabbitMQ ili AWS SQS. Aplikacija bi u tom slučaju poslala poruku u red tako da se kaskadne operacije mogo obavljati asihrono. Na primer recimo da je RabbitMQ podešen da prima poruke kada je update primarne tabele gotov recimo brisanje recepta (DELETE FROM recipes WHERE id = 123). Po dolasku poruke okida se funkcija koja čisti kaskadne tabele (Slika ) (može biti i serverless funkcija poput lambde kod AWS-a).



Slika 22: Izgled poruke koja stiže u red

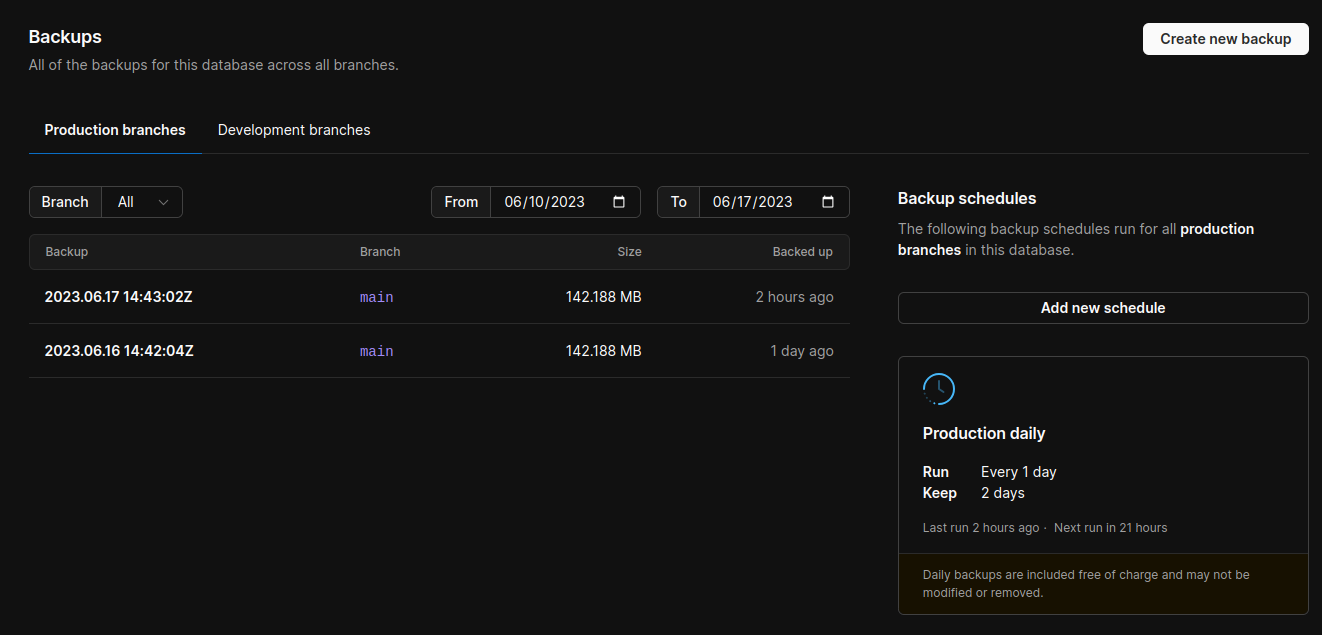


Slika 23: Funkcija koja čitsti posle update/delete operacije

Ovaj prilaz pruža najviše benefita. Kaskadne operacije su sada asihrone, tako da ne mora da se čeka nakon izmene primarne tabele. Lako je skalabilno rešenje zato što je praktično veoma mnogo funkcija moguće pokrenuti ukoliko je opterećenje veliko. Ovim se clean-up operacije izvršavaju u skoro real-time vremenu, najviše za onoliko vremene koliko je potrebno da poruka stigne do odredišta (serverless funkcija ulgavnom). Ključni minus ovog prilaza je dodatna kompleksnost uvođenjem message brokera i serverless funkcija.

# Backup

U besplatnom Hobby planu backup baze radi se jednom dnevno, dok se kod Scaler i Teams paketa radi na 12 sati odnosno po dogovoru kod Enterpise paketa. Kako bi videli rezervne kopije baze dovoljno pozicionirate se u Backups tab u gornjem meniju.

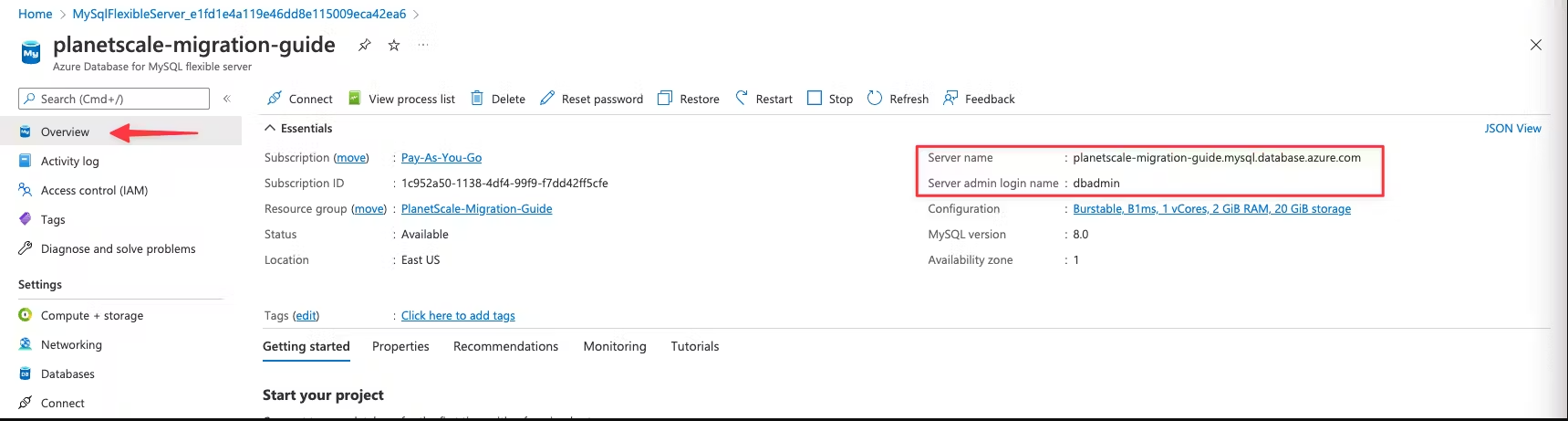


Slika 24: Backup stranica

Na slici iznad vidimo backup stranicu. U njoj vidimo dva taba onaj namenjen produkcijskoj grani i onaj namenjen development grani. Dalje vidimo da se backup izvršava jednom dnevno i je vreme zadržavanja kopije 2 dana. Kao što vidimo sa slike do sada su napravljene 2 kopije naše baze. Moguće je i ručno napraviti kopiju kliko na duge Create new backup dugme u gornjem desnom uglu, tu je moguće izabrati i vreme zadržavanja kopije uz oprez da se ta kopija dodatno naplaćuje.

# Migracija baze podataka

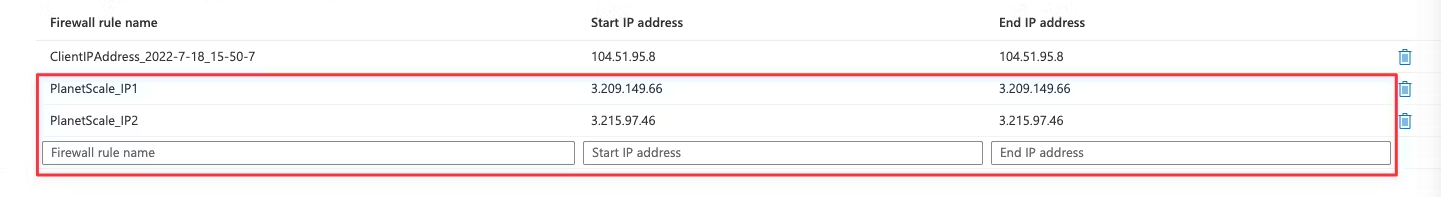
Realna situacija je i potreba za migracijom baze sa nekog drugog cloud servisa na PlanetScale. Trenutno podržani cloud provajderi su: AWS RDS, Azure, DigitalOcean, GCP CloudSQL, Maria DB. U nastavku dat je postupak migracije sa Azure servisa.



Slika 25: MySQL baza na Azure servisu

Ključne stvari su ime servira i admin ime, dok je admin lozinka ista ona koja se specificira pri kreiranju baze.

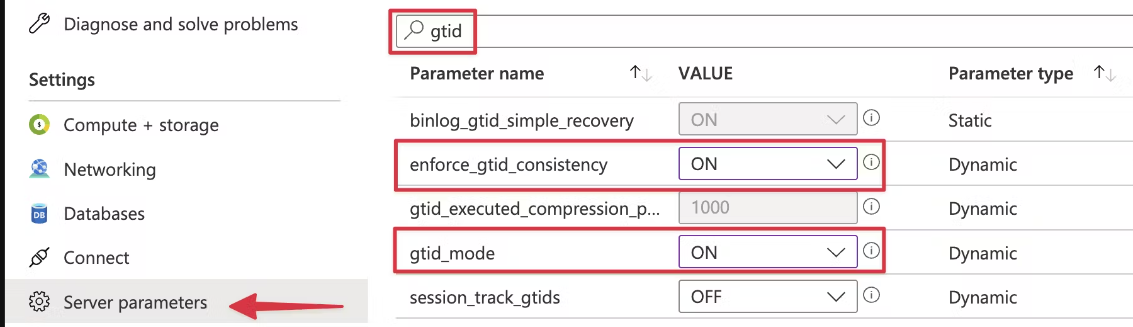
Prvi korak koji je potrebno uraditi jeste konfigurisati firewall tako da server Azure-a i PlanetScale-a mogu komunicirati. Zavisno od izbora mesta lokacije servera kod PlanetScale razlikuju se i IP adrese koje trebaju biti dozovljene, iz tog razloga najbolje je proveriti u samoj konfiguraciji servera.



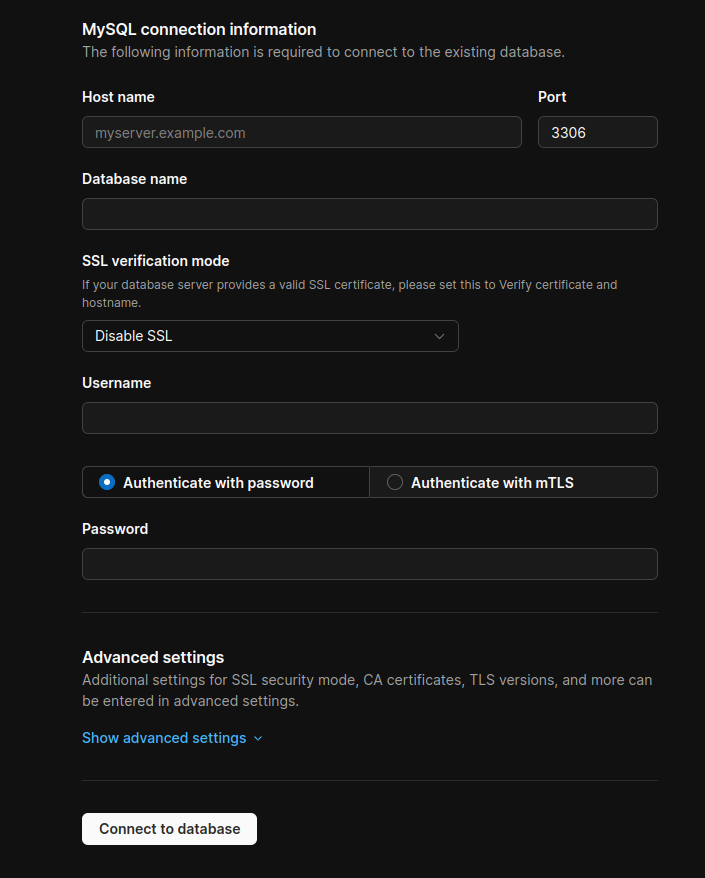
Slika 26: Konfiguracija firewall-a na Azure

Zadnja stvar koju je potrebno uraditi na Azure-u, jeste podesiti enforce\_gtid\_consistency i gtid\_mode na ON.

Na slici iznad dozvolili smo ip adrese na kojima se nalazi baza pokrenuta na PlanetScale-u

.

Slika 27: Konfiguracija servera



Slika 28: Import baze

Sada možemo referencirate bazu sa Azure-a, nakon unosa specifikacije potrebno je kliknuti na dugme *Connect to database* čime ćemo odraditi migraciju sa Azure-a na PlanetScale.

Zaključak

# Literatura

<https://planetscale.com/docs/concepts/data-branching>

<https://vitess.io/blog/2021-06-15-online-ddl-why-no-fk/>

<https://planetscale.com/docs/learn/operating-without-foreign-key-constraints>

https://www.yld.io/blog/planet-scale-and-vitess-a-technical-deepdive/