

Fakultet tehničkih nauka  
Univerzitet u Novom Sadu

Seminarski rad

***Trino***

Student: Mentor:

Petar Trifunović E2 4/2021 Prof. dr Vladimir Dimitrieski

# Sadžaj

**No table of contents entries found.**

# Uvod

U ovom radu dat je opis principa rada *Trino* tehnologije, propraćen praktičnim primerima. U nastavku će biti dat opis slučajeva korišćenja ove tehnolgije, kao i detaljan pregled osnovnih *Trino* koncepata. Nakon toga, fokus rada biće usmeren ka konkretnim načinima na koje je moguće koristiti *Trino*, iz ugla programera, u svrhu implementacije različitih aplikacija koje rade sa velikim količinama podataka.

## Naziv – *Trino* vs *Presto*

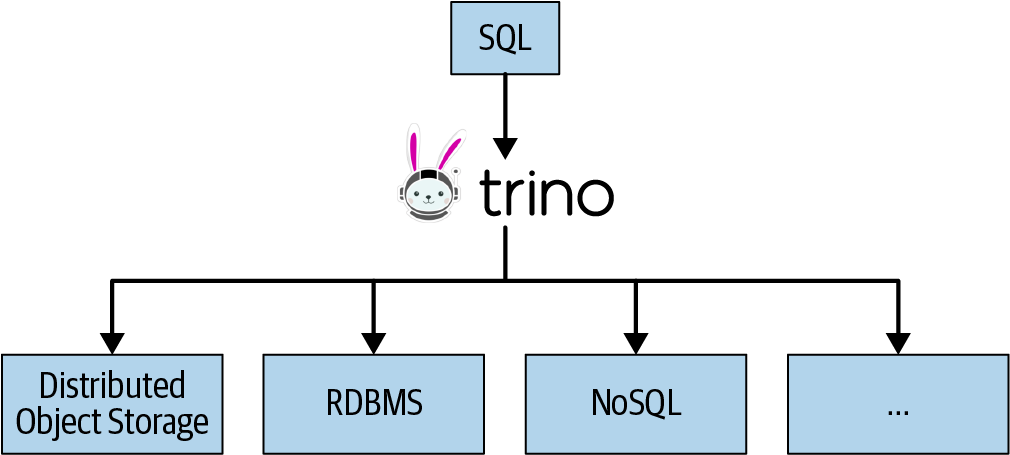
Pre početka opisivanja *Trino* tehnologije, važno je osvrnuti se na kratko na istoriju ovog softvera. 2012. godine, grupa od četvorice zaposlenih u *Facebook*-u - Martin Tavareso, Dein Sandstrom, Dejvid Filips i Erik Hvang, razvili su tehnologiju pod nazivom *Presto*. Njih četvorica zalagali su se, između ostalog, za to da *Presto* bude *open source* projekat, sa željom da se široka zajednica programera iz celog sveta uključi i doprinese razvoju ovog projekta.

2018. godine, *Facebook* je odlučio da u razvoj *Presto*-a unutar same kompanije uključi radnike koji sa *Presto* tehnologijom do tada nisu imali nikakvog iskustva, i to bez prethodnog konsultovanja sa četvoricom osnivača. Ovakav potez doveo je do toga da njih četvorica napuste *Facebook* i da nastave sa samostalnim razvojem tehnologije, proširivši joj ime u *PrestoSQL*. Ipak, i *Facebook* je nastavio sa razvojem, tako da je *Presto* nastavio da se razvija u dve nezavisne grane. *Facebook* je nakon toga pravno zaštitio naziv *Presto*, i svoju granu tehnologije nazvao *PrestoDB*. Četvorica osnivača su iz pravnih razloga bili nakon toga primorani da tehnologiju preimenuju u *Trino*.

U znak poštovanja prema četvorici pokretača razvoja, u nastavku će se za ovu tehnologiju koristiti isključivo naziv *Trino*, nevezano za to da li se radi o periodu pre ili nakon cepanja na dve grane razvoja.

# Princip rada i osnovni koncepti *Trino* tehnologije

Kako je navedeno u zvaničnoj dokumentaciji [[1](#Tri)]: *„Trino je distribuirani obrađivač SQL upita, dizajniran za izvršavanje upita nad velikim skupovima podataka distribuiranim na jednom ili više heterogenih izvora podataka... Trino je (takođe) dizajniran da podrži skladišta podataka i analitiku“*. *Trino* jeste nastao kao alternativa do tada postojećim metodama za pretragu *Hadoop* fajl sistema (*HDFS*) korišćenjem jezika visokog nivoa (*Apache Hive, Apache Pig*), imajući u vidu da su te metode bile nedovoljno brze za potrebe *Facebook*-a. Ipak, *Trino* je razvijen kako bi podržao rad sa raznovrsnim izvorima podataka, a ne samo sa *HDFS*-om, nudeći mogućnost jedinstvenog pristupa tim izvorima korišćenjem *SQL*-a, kao što je prikazano na slici 2.0. U narednim poglavljima dat je pregled načina na koji *Trino* ovo omogućava.



*Slika 2.0 – Pristup različitim izvorima podataka pomoću SQL-a; preuzeto iz* [[2](#Ful21)]

Zvanična dokumentacija [[1](#Tri)], naučni rad *Presto: SQL on Everything* [[3](#Set19)], kao i knjiga koja predstavlja vodič za *Trino* [[2](#Ful21)] daju kvalitetan opis najvažnijih pojmova za razumevanje rada *Trino* tehnologije, tako da su naredna poglavlja pisana pretežno na osnovu ovih izvora.

## Tipovi servera

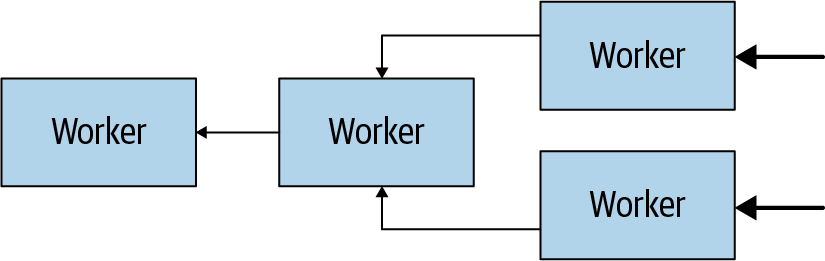
*Trino* klaster sastoji se od servera koordinatora (eng. *coordinators*) i servera radnika (eng. *workers*). U klasteru se uvek mora nalaziti jedan koordinator, uz jednog ili više radnika, a moguće je da jedna *Trino* instanca igra ulogu i koordinatora i radnika (najčešće u razvojnoj fazi).

### Koordinator

Koordinator predstavlja „glavni“ *Trino* server. On je predstavlja sponu klijenta i *Trino* klastera, s obzirom na to da klijenti upravo ovaj čvor klastera kontaktiraju kada izdaju *SQL* iskaz[[1]](#footnote-1). Koordinator prihvata iskaz od klijenta, prevodi ga u upit, stvara plan izvršavanja i raspoređuje posao po čvorovima radnicima. Takođe, na kraju, koordinator je taj koji prihvata rezultate od radnika i prezentuje ih klijentu, ili ih smešta u bafer. Koordinator je u stanju da podeli izvršavanje na faze i da prati stanje izlaznog bafera, tako da, kada klijent pročita određenu porciju rezultata iz bafera, koordinator može sam da pokrene izvršavanje naredne faze.

### Radnici

Serveri radnici jesu oni koji izvršavaju poslove dodeljene od strane koordinatora, i koji vrše eventualne obrade podataka. Radnici su uređeni u stablo (slika 2.1), tako da oni na najnižem nivou, odnosno listovi stabla, vrše obradu nad podacima koji dolaze iz različitih izvora (*HDFS*, relacione baze, *NoSQL* baze...). Čvorovi na višim nivoima stabla rade sa podacima koji nastaju kao rezultat obrade čvorova na nivoima ispod njih. Radnici znaju šta treba izvršiti na osnovu zadataka koje im prethodno dodeli koordinator.



*Slika 2.3 – Primer hijerarhijskog uređenja čvorova radnika; preuzeto iz* [[2](#Ful21)]

## Apstrakcija i pristup izvorima podataka

Kako bi se omogućio jedinstven pogled na raznovrsne izvore podataka, *Trino* za njih kreira apstrakciju i iznad njih postavlja pogodan interfejs u vidu *API*-ja.

### Konektori

Konektor predstavlja komponentu apstrakcije koja povezuje *Trino* sa izvorom podataka, nezavisno od tehnologije kojom je taj izvor predstavljen (*HDFS*, relaciona baza, *NoSQL* baza...). Kako je navedeno u poglavlju 4 vodiča za *Trino* [[2](#Ful21)]: *„Dokle god je moguće podatke predstaviti u vidu tabela, kolona i vrsta korišćenjem tipova podataka dostupnih u Trino-u, moguće je kreirati konektor...“*

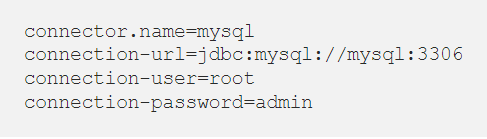
Dakle, ukoliko je moguće podatke iz izvora logički predstaviti u obliku koji *Trino* zahteva, moguće je nad njime kreirati apstrakciju u vidu konektora koji će čvorovima radnicima omogućiti da iz izvora pribave podatke. Kreiranje konektora podrazumeva implementiranje *Trino SPI-*ja[[2]](#footnote-2). Kako je *SPI* interfejs, *Trino* može pozivati metode iz interfejsa bez obzira na implementaciju, što implementaciju raspoređivanja i izvršavanja poslova čini nezavisnom od samog izvora podataka.

*Trino* ima određen broj ugrađenih konektora [[4](#TriConcepts)]. Mnoge ekipe programera razvijale su druge konektore po potrebi, nezavisno od *Trino* razvojnog tima, tako da danas postoji veliki broj konektora, pa *Trino* može pristupiti zaista širokom skupu izvora podataka. Spisak i detalji o postojećim konektorima dostupan je u zvaničnoj dokumentaciji [[5](#TriConnectors)].

### Katalozi

*Trino* katalog je komponenta čija je namena da opiše i definiše izvore podataka koje će *Trino* klaster koristiti. Jedan katalog predstavljen je pomoću konfiguracionog fajla koji sadrži važne informacije, a kao najvažniju sadrži naziv konektora koji će se koristiti za pristup podacima iz izvora na koji se taj katalog odnosi.

Primer jednog konfiguracionog fajla kataloga dat je na slici 2.2. U pitanju je katalog koji opisuje *MySQL* bazu kao izvor podataka. Najvažnija linija u ovom fajlu jeste prva, koja definiše tip konektora, a samim tim i tip izvora podataka, i omogućava *Trino*-u da, pri radu sa ovim izvorom, koristi odgovarajuću implementaciju *SPI*-ja. Ostale linije u fajlu predstavljaju vrednosti koje su karakteristične za sam *MySQL*.



*Slika 2.2 – Primer kataloga za MySQL bazu; preuzeto iz* [[6](#Med)]

Konfiguracioni fajlovi kataloga imaju *.properties* ekstenziju. Naziv kataloga definisan je onime što prethodi ovoj ekstenziji (npr. ukoliko je fajl sa slike 2.2 nazvan *mysql.properties*, katalog će u *Trino* klasteru nositi naziv *mysql*). Naziv kataloga je važan za jedinstvenu identifikaciju *Trino* tabela. Svaka tabela definisana je katalogom, šemom kojoj pripada i na kraju imenom tabele. Primer iz [[4](#TriConcepts)] objašnjava da je *hive.test\_data.test* jedinstveni naziv tabele *test* iz šeme *test\_data* iz kataloga *hive* (dakle, kataloga definisanog fajlom *hive.properties*). Više o konceptima šeme i tabele sledi u narednim poglavljima.

Kako naziv konfiguracionog fajla može biti proizvoljno izabran i nije ni u kakvoj vezi sa nazivom konektora na koji se katalog odnosi, moguće je kreirati više kataloga za rad sa različitim instancama istog tipa izvora podataka (npr. više kataloga za rad sa različitim instancama *MySQL-*a, *Hive*-a...).

### Šeme

Šema predstavlja skup određenog broja tabela iz konkretnog izvora podataka. Svaka šema mora da pripada katalogu, tako da naziv kataloga i naziv šeme jedinstveno određuju skup tabela. Koncept šeme postoji već u relacionim bazama, a i u nekim drugim tehnologijama za čuvanje podataka, kao što je *Hive*. Preslikavanje koncepta šeme u ovakvim izvorima podataka na *Trino* šemu je trivijalno. Kreiranje šema u izvorima gde taj koncept ne postoji zavisi od implementacije konektora i načina na koji programeri koji konektor razvijaju odluče da organizuju podatke, koji su prethodno već organizovani u tabele.

### Tabele

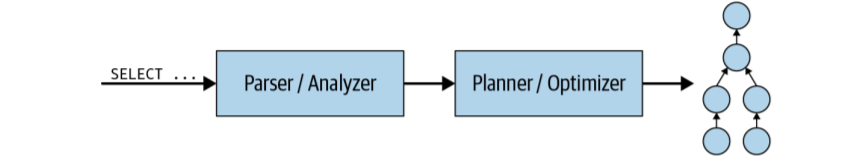
Tabele u *Trino*-u predstavljaju isti koncept kao i tabele u klasičnim relacionim bazama – vrste podeljene na polja definisana kolonama tabele. Kolone i ovde imaju naziv i tip. Kao i kada je šema u pitanju, preslikavanje između tabele iz izvora podataka i tabele iz *Trino*-a zavisi od implementacije konektora, a može biti trivijalno ukoliko su podaci u samom izvoru prirodno organizovani u tabele.

## Model izvršavanja upita

Kao i sa izvorima podataka, *Trino* definiše i nekoliko apstraktnih pojmova koji omogućavaju bolje razumevanje i veću kontrolu nad izvršavanjem upita.

### Plan upita

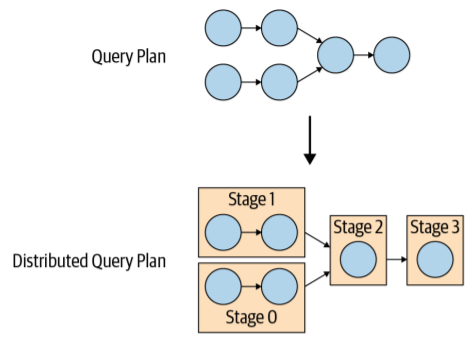
Klijent svoje upite izdaje koordinatorskom čvoru u formi tekstualnog iskaza (razlika između *iskaza* i *upita* opisana je u fusnoti u poglavlju 2.1.1). Koordinator zatim parsira i analizira taj tekst, pa se tako obrađena forma iskaza propušta kroz koordinatorsku komponentu za planiranje i optimizaciju upita (više o detaljima planiranja i optimizacije upita može se naći u *Trino* vodiču [[2](#Ful21)], u poglavlju 4). Kao rezultat svih ovih akcija, dobija se *plan upita*, što je posebna *Trino* struktura koja na visokom nivou, bez previše detalja, daje uvid u niz koraka koje je potrebno izvršiti kako bi se realizovalo ono što je klijent zahtevao svojim *SQL* iskazom. Na slici 2.3 prikazan je proces transformacije iskaza u plan izvršavanja. Struktura nalik na stablo na slici predstavlja plan upita.



*Slika 2.3 – Proces transformacije SQL iskaza u plan izvršavanja; preuzeto iz* [[2](#Ful21)]

### Faze izvršavanja i fragmenti plana

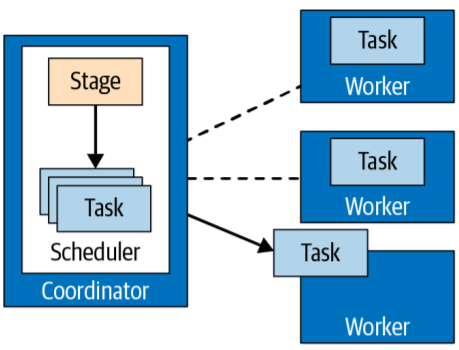
Nakon što je kreiran plan upita kao jedinstvena celina, on se prevodi u distribuiran oblik podelom na *fragmente plana*. Fragment plana u toku izvršavanja naziva se *faza izvršavanja* (eng. *stage*). Podela na fragmente dovodi do nastanka stabla zavisnosti u kojem su čvorovi fragmenti, odnosno faze (slika 2.4). Broj faza zavisi od kompleksnosti početno iskaza, odnosno upita koji se od njega dobija. Zavisnosti među fazama utiču na mogućnost paralelizacije izvršavanja. Ipak, kako bi se maksimalno iskoristilo postojanje više radnika u klasteru i sprovela maksimalna moguća paralelizacija izvršenja, faze nisu te koje se dodeljuju cele na izvršenje jednom radniku. Naime, faze se dalje dele na nezavisne jedinice izvršenja, odnosno *zadatke*.



*Slika 2.4 – Preslikavanje osnovnog plana upita na faze izvršenja; preuzeto iz* [[2](#Ful21)]

### Zadaci

Zadaci (eng. *task*) su jedinice izvršenja od kojih se sastoje faze. Svaka faza sastoji se od jednog ili više zadataka koji obrađuju deo podataka. Koordinator je zadužen za podelu zadataka iz jedne faze čvorovima radnicima, kako je prikazano na slici 2.5. Zadaci se mogu izvršavati u paraleli čime se smanjuje ukupno vreme potrebno za izvršenje jedne faze.

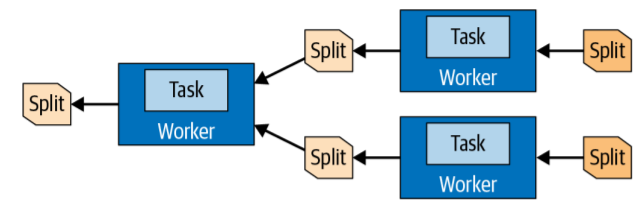


*Slika 2.5 – Podela zadataka jedne faze po čvorovima radnicima; preuzeto iz* [[2](#Ful21)]

### *Split*-ovi[[3]](#footnote-3)

*Split* predstavlja onaj deo čitavog skupa podataka koji obrađuje jedan zadatak. To je zapravo podskup podataka za koji je u određenom trenutku zadužen jedan radnik (ali jedan radnik može u trenutku raditi sa više *split*-ova). Izgled *split*-a i način na koji se on kreira od čitavog skupa podataka zavisi od izvora podataka sa kojim se radi, kao i od implementacije konektora. U [[2](#Ful21)] se kao primer navodi *Hive* kod kojeg je *split* definisan putanjom do fajla na koji se odnosi, kao i *offset*-om[[4]](#footnote-4) i dužinom kojima je određeno koji deo fajla se preslikava na dati *split*.

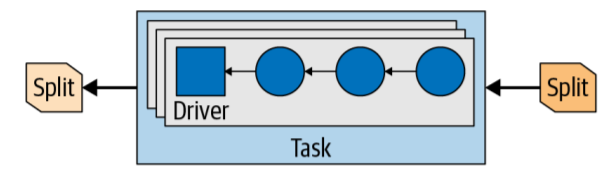
U pogledu toga odakle pribavljaju *split*-ove, postoji razlika između zadataka koji pripadaju fazama koji su listovi u stablu zavisnosti između faza, i onih faza koje se nalaze na višim nivoima. Faze u listovima, tačnije njihovi zadaci, pribavljaju podatke svog *split*-a iz izvora podataka na način koji definiše konektor. Za viši nivo, izvor *split*-a jeste izlaz iz zadataka sa nivoa ispod. Zadaci na najdubljem nivou su oni koji se prvi izvršavaju, pribavljaju podatke u vidu *split*-ova iz izvora, i u svoj izlazni bafer smeštaju nove *split*-ove koji sadrže podatke nastale kao rezultat obrade izvornog *split*-a, a koji se takođe nalaze u tabelarnoj formi. Rezultujući *split*-ovi smeštaju se u izlazni bafer, gde postaju dostupni za ponovnu obradu. Zadaci iz faze na sledećem nivou pribavljaju sada podatke kao *split*-ove iz izlaznih bafera zadataka na prethodnom nivou, i ponovo rezultate obrade smeštaju u bafer. Na kraju, koordinator može preuzeti konačan rezultat iz izlaznih bafera zadataka na najvišem nivou. Na slici 2.6 prikazan je tok podataka kroz zadatke, počev od izvornih *split*-ova. Na desnom kraju prikazane strukture nalaze se *split*-ovi iz originalnog izvora podataka.



*Slika 2.6 – Tok podataka kroz zadatke počev od izvornih split-ova; preuzeto iz* [[2](#Ful21)]

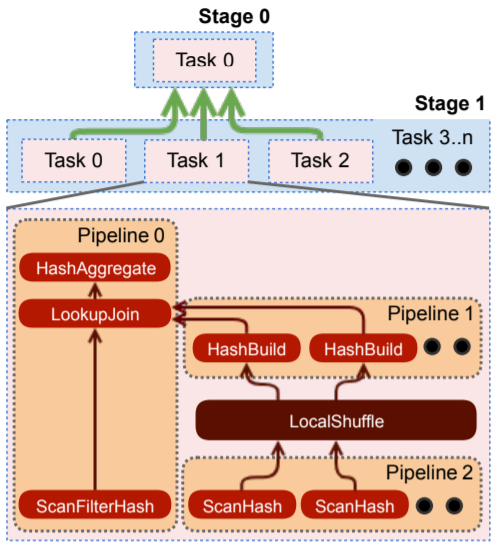
### Drajveri i operatori

Zadaci obrađuju *split*-ove tako što nad njima izvrše operacije definisane **operatorima**, kao što su prolazak kroz tabele radi čitanja, filtriranje, grupisanje, agregacije, spojevi... Dakle, u pitanju su operacije već poznate u *SQL*-u. Operatori se grupišu u sekvence od kojih svaka izvršava neki deo čitavog zadatka. Svaka instanca sekvence operatora predstavlja jedan **drajver** (eng. *driver*). U jednom zadatku koji radi sa jednim *split*-om moguće je imati više sekvenci operatora, odnosno više drajvera, kao što je prikazano na slici 2.7. Stoga, drajver predstavlja jedinicu paralelizma najnižeg nivoa. Zadaci i *split*-ovi definisali su globalni paralelizam, na nivou klastera, u pogledu raspoređivanja posla na čvorove radnike koji ih mogu izvršavati nezavisno, odnosno u paraleli. Drajveri, sa druge strane, predstavljaju osnovnu jedinicu paralelizma lokalnog tipa, na jednom čvoru. Drajver ima jedan ulaz i jedan izlaz. Optimizator izvršenja upita je u stanju da, nakon što je određena sekvenca operatora koja je neophodna za izvršenje zadatka, tu sekvencu ako je moguće podeli na više manjih, odnosno da kreira više drajvera, koji se mogu izvršavati u paraleli.



*Slika 2.7 – Paralelni drajveri u jednom zadatku; preuzeto iz* [[2](#Ful21)]

Ponekad, iako se nakon analize zadatka kreira više drajvera, nije moguće sve njih izvršavati u paraleli. Dobar primer, dat u [[3](#Set19)], jeste heš-spoj[[5]](#footnote-5). Na slici 2.8 prikazana je podela posla na drajvere (na slici označeni sa *pipeline*). Zadatak 1 radi sa dva *split*-a, i mora da pročita podatke iz oba (bilo da dolaze iz osnovnih izvora podataka, ili iz izlaznih bafera zadataka na nižem nivou). Drajver 0 (odnosno *pipeline* 0) predstavlja deo algoritma heš-spoja koji pretražuje heš tabelu (eng. *probe* deo) i radi sa levom stranom spoja, a drajveri 1 i 2 deo za kreiranje heš tabele (eng. *build* deo) u koju se smešta desna strana spoja[[6]](#footnote-6). Kako je u ovom slučaju za *build* fazu neophodno pročitati podatke iz *split*-a i kreirati tabelu, *build* deo je podeljen na dva drajvera, jedan koji učitava podatke (drajver 2) i drugi koji kreira memorijsku strukturu heš tabele (drajver 1). Ova dva drajvera su nezavisna i mogu se izvršavati u paraleli u posebnim nitima, s tim što je posle neophodno podatke iz *split*-a smestiti u tabelu, što je prikazano *LocalShuffle* fazom. Iako je drajver 0 izdvojen kao posebna sekvenca operatora, on za vreme dok se izvršavaju drajveri *build* faze može samo da učita svoj *split* (označeno sa *ScanFilterHash*), pa mora da čeka na kraj kreiranja heš tabele. Svakako, ovo je samo jedan specifičan slučaj, i ne treba na osnovu njega steći utisak da je postojanje drajvera besmisleno, naprotiv, drajveri znatno ubrzavaju i poboljšavaju rad *Trino* klastera.



*Slika 2.8 – Podela izvršavanja heš-spoja na drajvere; preuzeto iz* [[3](#Set19)]

# Praktični primeri primene *Trino*-a

# Bibliography

x

|  |  |
| --- | --- |
| [1] | Trino. [Online]. <https://trinodb.github.io/docs.trino.io/current/> |
| [2] | M. Fuller, M. Traverso, and M. Moser, *Trino: The Definitive Guide*.: O’Reilly Media, 2021. |
| [3] | R. Sethi et al., "Presto: SQL on Everything," *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019. |

x

1. U *Trino* dokumentaciji [[1](#Tri)] pravi se jazna razlika između iskaza (eng. *statement*), što jeste tekst zapisan u *SQL-*u i upita (eng. *query*) predstavljenog planom izvršavanja (eng. *query plan*). Kako se u pozadini mogu naći različiti izvori podataka, razlika između ova dva pojma je mnogo veća u *Trino*-u nego u relacionim bazama podataka. [↑](#footnote-ref-1)
2. *Trino Service Provider Interface*; implementiranjem metoda iz ovog interfejsa tako da odgovaraju konkretnom izvori podataka dobija se *API* putem koga *Trino* može komunicirati sa izvorom. [↑](#footnote-ref-2)
3. U nedostatku adekvatnog prevoda na srpski jezik, zadržan je engleski naziv. [↑](#footnote-ref-3)
4. Udaljenost od početka fajla. [↑](#footnote-ref-4)
5. Eng. *hash join*; više o ovome na [[7](#Wik)] [↑](#footnote-ref-5)
6. Strane spoja u konkretnom primeru izabrane su tako da odgovaraju slici, tako da levi deo prikazanog zadatka radi sa levom, a desni sa desnom stranom. U realnosti, za *build* stranu heš-spoja najčešće se uzima ona strana spoja koja ima manje redova. [↑](#footnote-ref-6)