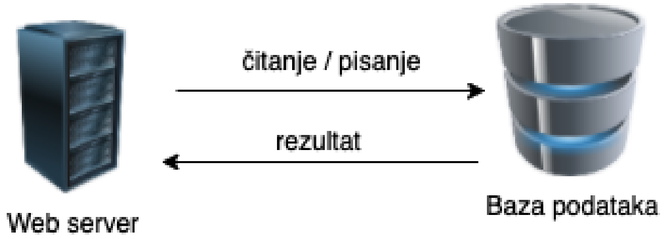
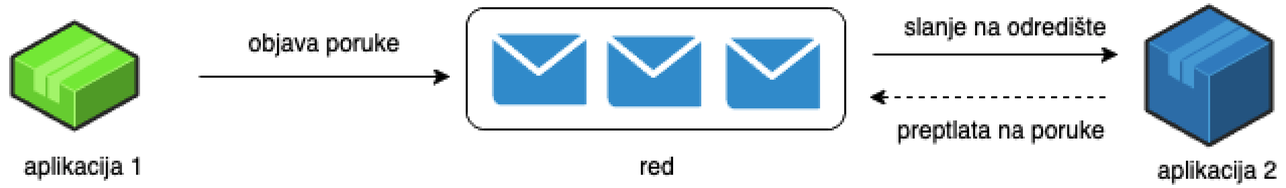
KOMUNIKACIJA SA BAZOM PODATAKA

SCENARIJI TOKOVA PODATAKA

TRI NAJČEŠĆA SCENARIJA

- Direktna komunikacija klijent <-> server kroz poziv servisa
- Serverska aplikacija <-> Baza podataka
- Asinhrona komunikacija razmenom poruka preko reda poruka (message queue)

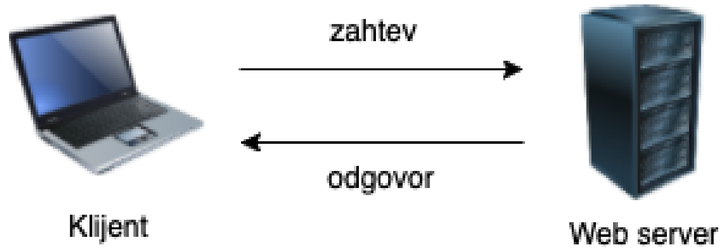


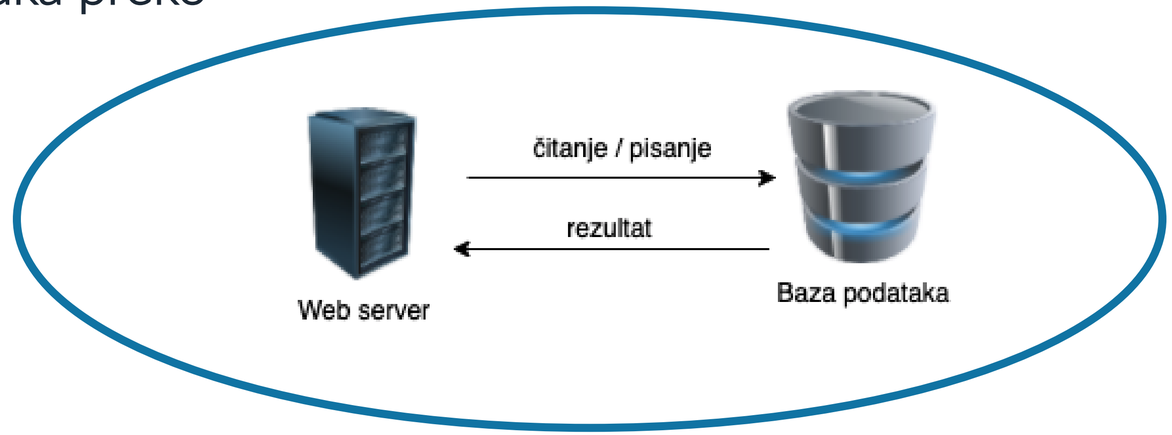


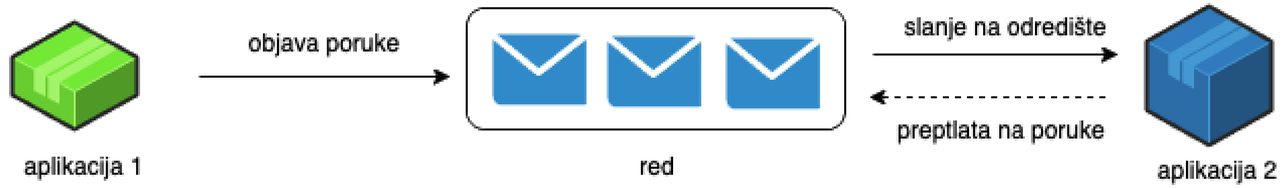
SCENARIJI TOKOVA PODATAKA

TRI NAJČEŠĆA SCENARIJA

- Direktna komunikacija klijent <-> server kroz poziv servisa
- Serverska aplikacija <-> Baza podataka
- Asinhrona komunikacija razmenom poruka preko reda poruka (message queue)

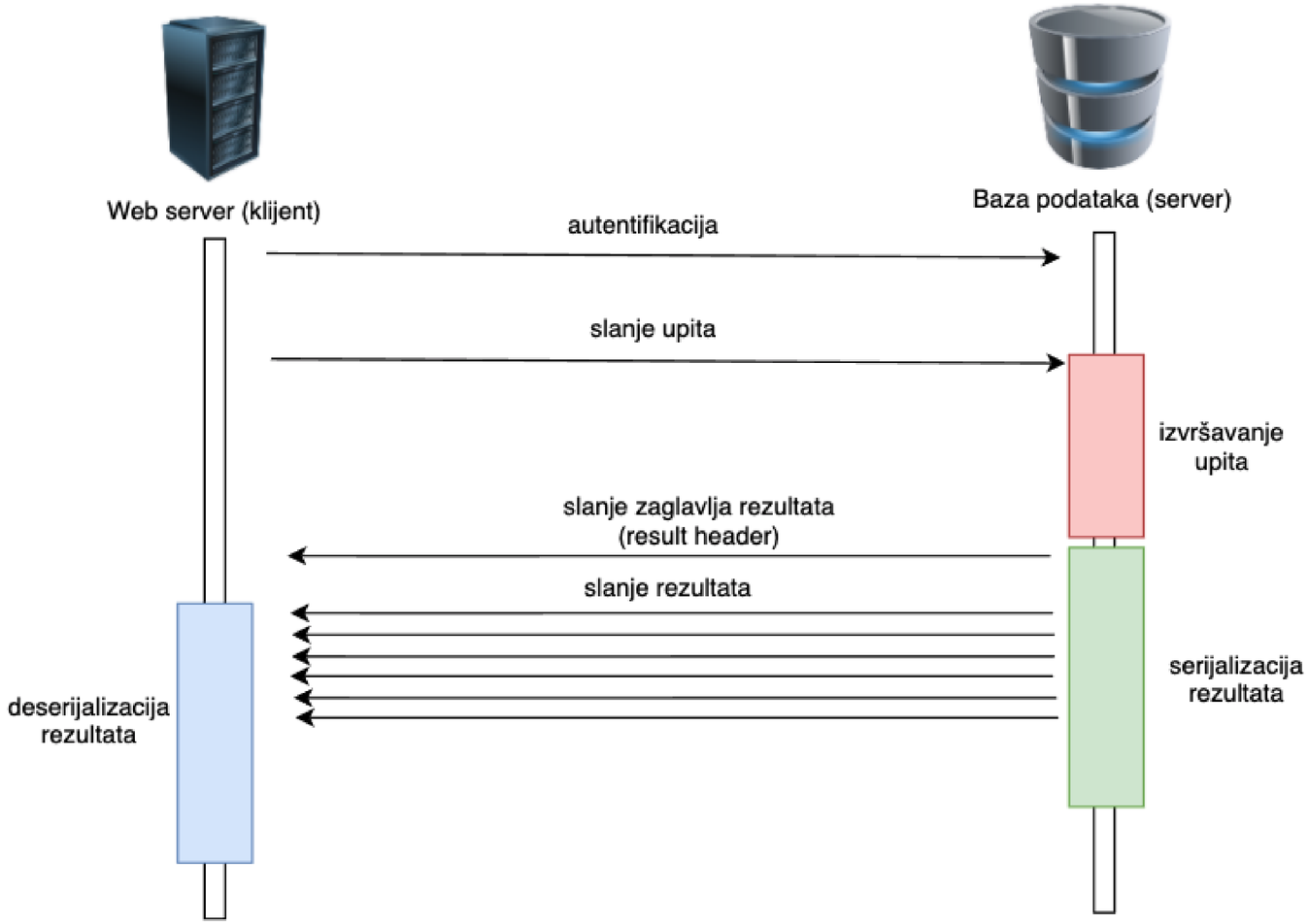






KOMUNIKACIJA SA BAZOM PODATAKA

- **APLIKACIJE PRISTUPAJU BAZI PODATAKA (SUBP, DBMS) KROZ API**



KOMUNIKACIJA SA BAZOM PODATAKA

- **APLIKACIJE PRISTUPAJU BAZI PODATAKA (SUBP, DBMS) KROZ API**
  - Direktan pristup (specifičan za konkretan DBMS)
  - Open Database Connectivity (ODBC)
  - Java Database Connectivity (JDBC)

OPEN DATABASE CONNECTIVITY

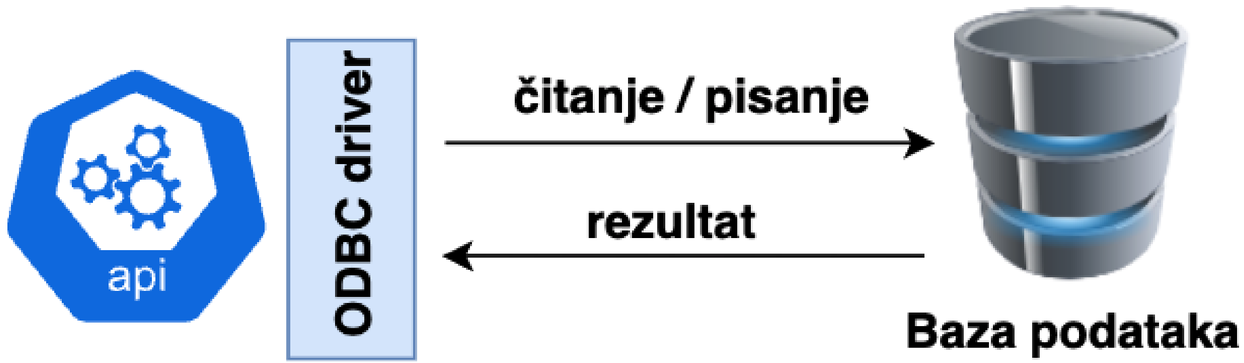
ŠTA JE ODBC?

- Predstavlja standardni API za pristup DBMS
- Dizajniran da bude nezavisan od DBMS i OS
- Microsoft i Simba Technologies su ga razvili ranih 1990ih
- Svaki ozbiljniji DBMS ima ODBC implementaciju

OPEN DATABASE CONNECTIVITY

ŠTA JE ODBC?

- ODBC se bazira na „device driver“[1] modelu
- Drajver enkapsulira logiku koja je potrebna za konverziju standardnog skupa komandi u DBMS specifične pozive

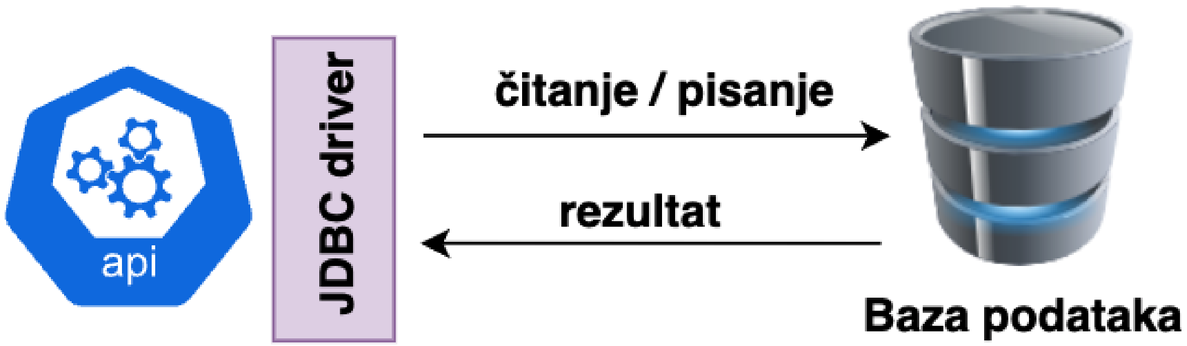


1 Driver model https://www.kernel.org/doc/html/latest/driver-api/driver-model/index.html

JAVA DATABASE CONNECTIVITY

ŠTA JE JDBC?

- Sun Microsystems ga je razvio 1997.
- Pruža standardni API za konekciju Java programa sa DBMS
- Možemo ga posmatrati kao verziju ODBC pisanu u Javi umesto u C programskom jeziku



JAVA DATABASE CONNECTIVITY

PRISTUPI ZA IMPLEMENTACIJU

- JDBC-ODBC Bridge (Type 1)
    - Konvertuje JDBC poziv metode u ODBC poziv funkcije (od JDK 1.8 nije podržan pristup)
- Native API Driver (Type 2)
    - Konvertuje JDBC poziv metode u nativni poziv DBMS API (dodatne .dll ili .so datoteke)
- Network Protocol Driver (Type 3)
    - Drajver se konektuje na *middleware* koji konvertuje JDBC poziv u DBMS specifični poziv
- Database Protocol Driver/Thin Driver (Type 4)
    - Najbrži pristup jer je čista Java implementacija koja konvertuje JDBC poziv u DBMS specifični

JAVA DATABASE CONNECTIVITY

PRISTUPI ZA IMPLEMENTACIJU

Tip drajvera	Naziv	Način rada	Zavisan od platforme	Potrebne native biblioteke	Prednosti	Mane
Type 1	JDBC–ODBC Bridge	JDBC poziv → ODBC poziv → DBMS	Da	Da (ODBC)	Jednostavan, ranije široko podržan	Spor, zastareo, uklonjen iz JDK 1.8
Type 2	Native API Driver	JDBC poziv → native DB API (DLL/SO) → DBMS	Da	Da	Dobre performanse, koristi native API	Težak deploy, nepodržan za Web, platformski zavisan
Type 3	Network Protocol Driver	JDBC poziv → middleware → DBMS	Ne (Java)	Ne	Visoka portabilnost, fleksibilan, centralizovan pristup kroz middleware	Middleware server je dodatni sloj, može biti sporiji
Type 4	Thin Driver (Database Protocol Driver)	JDBC poziv → direktan DBMS protokol	Ne (čista Java)	Ne	Najbrži, najportabilniji, najčešće korišćen, lak za deploy	Specifičan za svaki DBMS (poseban JAR po bazi)

PROTOKOLI ZA RAD ZA BAZOM

PRISTUPI ZA IMPLEMENTACIJU

- Svi veći proizvođači DBMS implementiraju svoje mrežne protokole preko TCP/IP
- Noviji DBMS implementiraju neke od open-source DBMS mrežnih protokola
- To im omogućava da koriste postojeće drajvere bez potrebe da razvijaju i održavaju svoje

PROTOKOLI ZA RAD ZA BAZOM

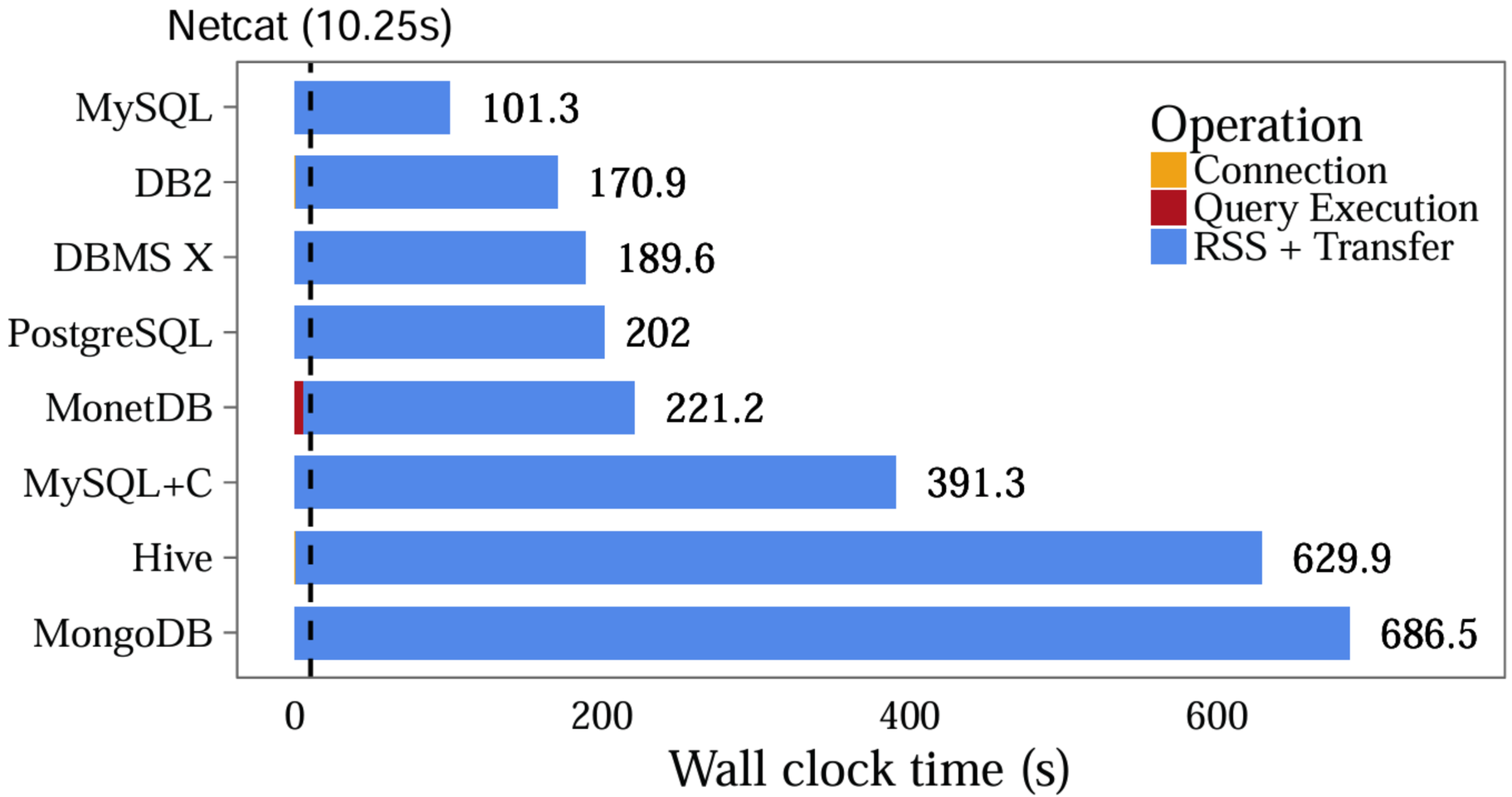
Slika preuzeta sa https://15721.courses.cs.cmu.edu/spring2020/slides/11-networking.pdf

PROTOKOLI ZA RAD ZA BAZOM

PRISTUPI ZA OPTIMIZACIJU

- Row Layout problem
  - ODBC/JDBC su API-ji orijentisani ka redovima iz tabela gde server pakuje torke u poruke jednu po jednu, red po red i klijent deserijalizuje podatke red po red
  - Potencijalno rešenje – slati podatke u vektorima (u *batch*-u, *column layout*)
- Kompresija podataka pre slanja klijentu (Snappy, Zstd, Zlib, ...)
- Serijalizacija podataka
  - Binarni format (protobuf, flatbuffers, Thrift, ...)
  - Tekstualni format
- Upravljanje stringovima
  - Null termination – dodavanje ”\0” na kraj
  - Prefiks dužine - dodavanje dužine stringa na početku bajtova koji se prenose
  - Fiksna dužina - za sve stringove se radi *padding* da se dostigne maksimalna veličina podatka

PROTOKOLI ZA RAD ZA BAZOM



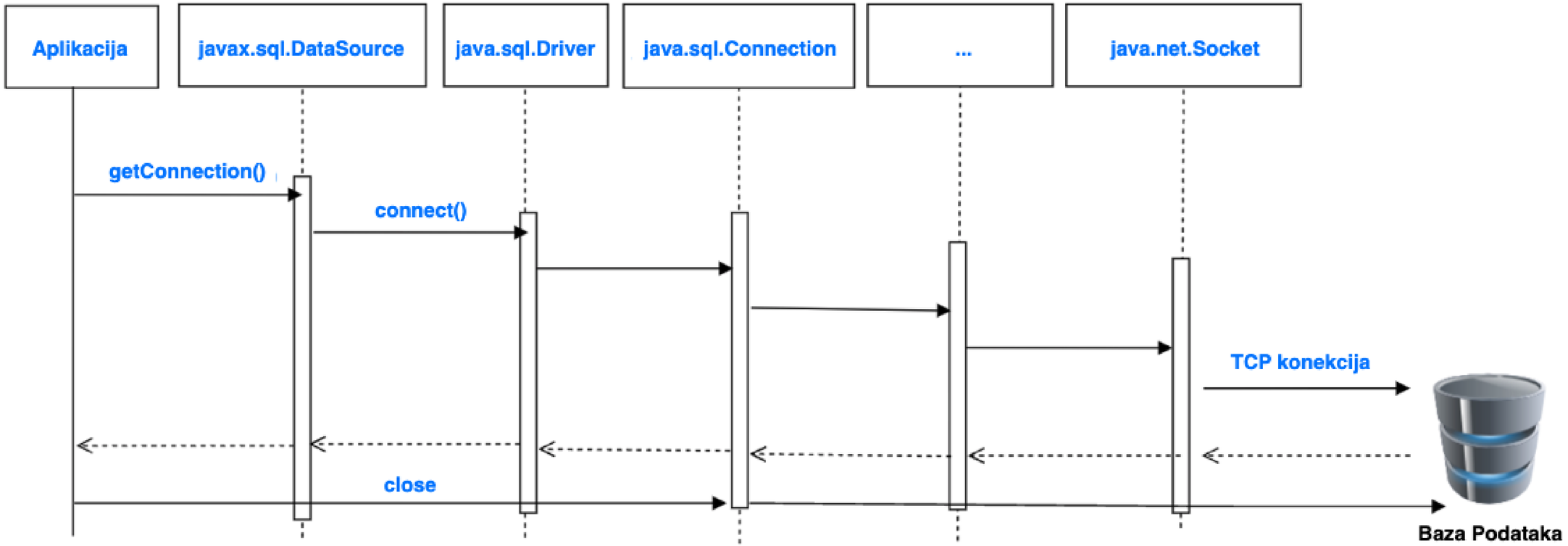
- Vreme da se izvrši **`SELECT * FROM lineitem`**[1] preko ODBC konektora iz različitih baza podataka gde su i klijent i server (baza) na istoj mašini
- Isprekidana linija je vreme za koje **`netcat`** prebaci preko soketa istu količinu podataka u CSV formatu

Slika preuzeta sa https://hannes.muehleisen.org/publications/p852-muehleisen.pdf

[1] TPC Benchmark H lineitem tabela (60M redova, 7.2GB veličina CSV datoteke) - https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

KONEKCIJA SA BAZOM PODATAKA

KAKO SE OSTVARUJE KONEKCIJA SA BAZOM?



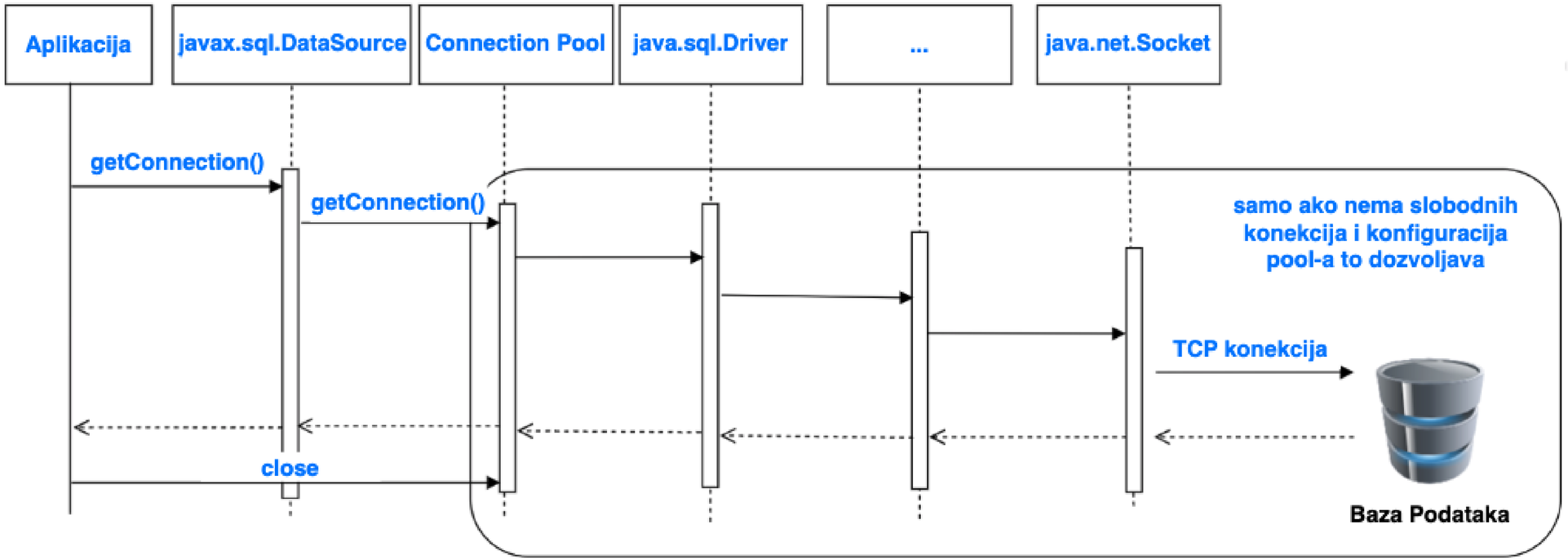
DataSource https://docs.oracle.com/javase/8/docs/api/javax/sql/DataSource.html

Primeri raw-jdbc-example, servlet-jdbc-example

CONNECTION POOL

KAKO SE SMANJUJE I/O OVERHEAD IZMEĐU APLIKACIJE I DBMS?

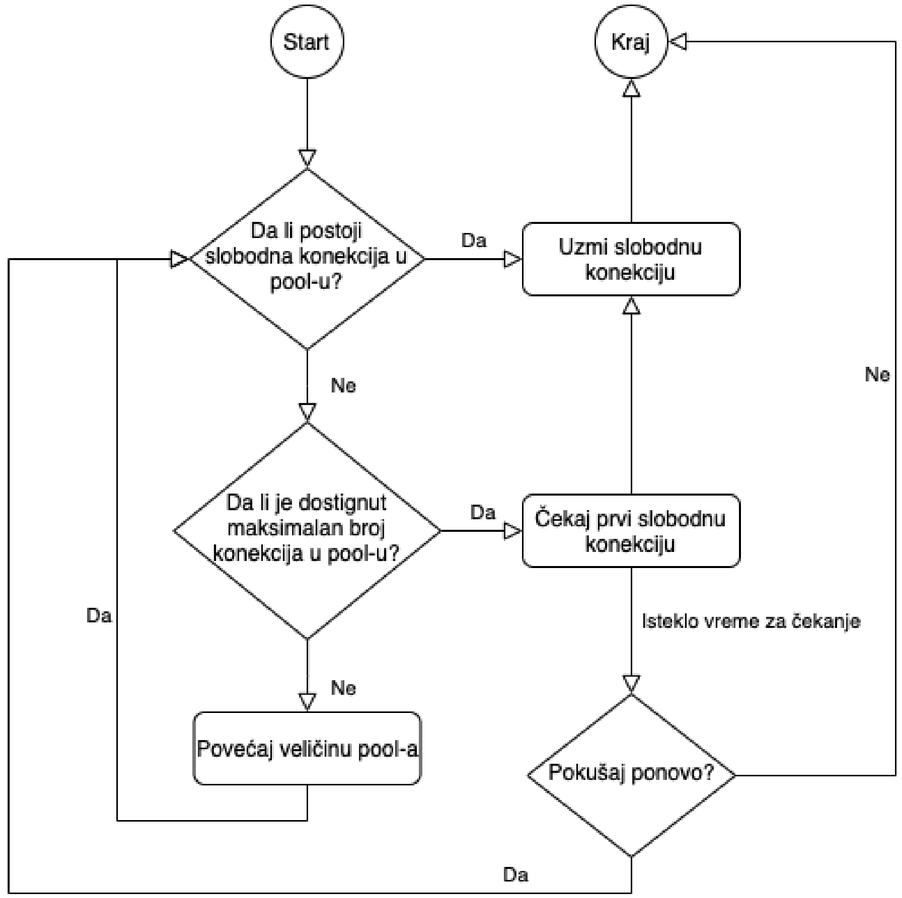
- Korišćenjem connection pool-a koji predstavlja keš za konekcije koje se mogu iznova koristiti bez potrebe kreiranja novih



Primer connection-pool-example

CONNECTION POOL

KAKO IZGLEDA LOGIKA PRIBAVLJANJA KONEKCIJE?



CONNECTION POOL

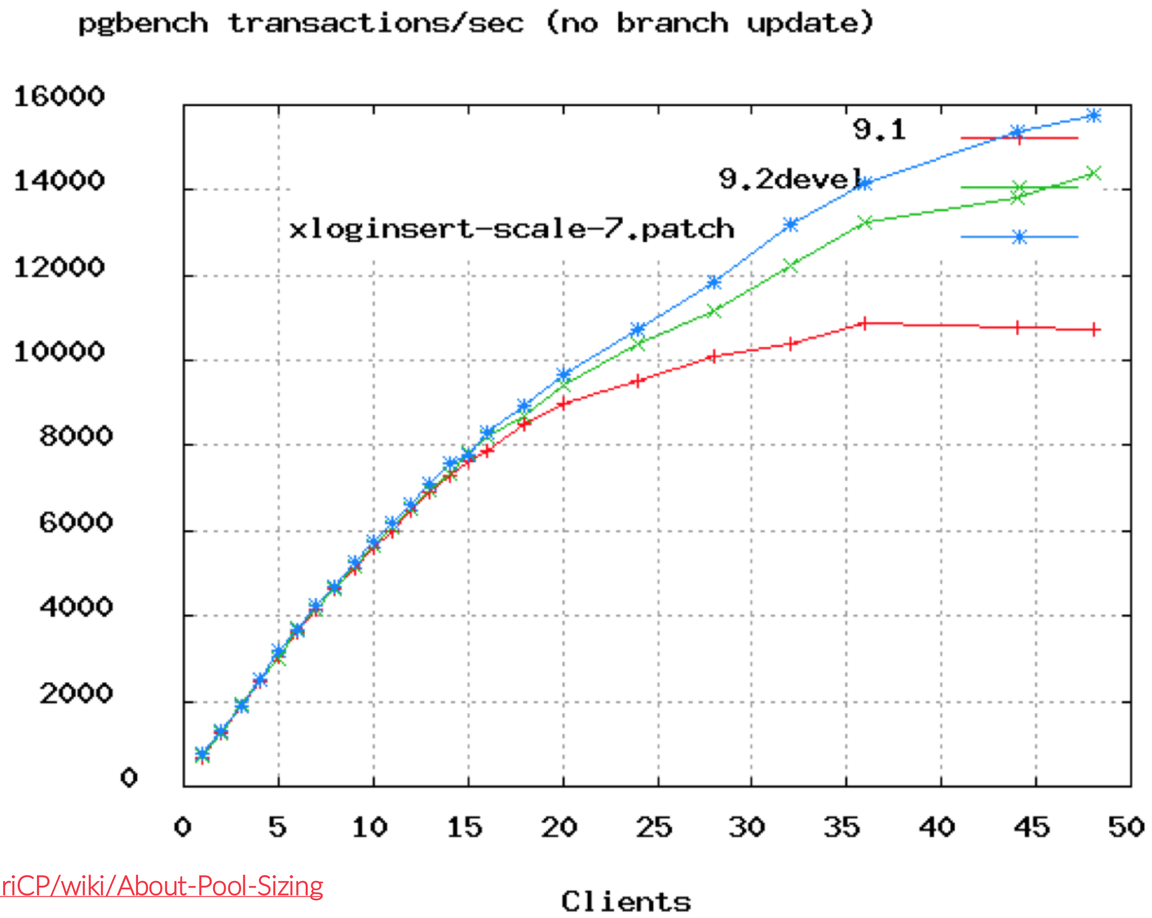
ŠTA SU USKA GRLA RADA SA BAZOM PODATAKA?[1]

- CPU
- Disk
- Mrežni protok
- Memorija (pravi nekoliko redova veličine manju razliku u odnosu na prethodna 3)

[1] https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing

CONNECTION POOL

ŠTA SU USKA GRLA RADA SA BAZOM PODATAKA?



Slika preuzeta sa https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing

CONNECTION POOL

POSTOJI LI IDEALNA FORMULA ZA BROJ KONEKCIJA?

connections = ((core_count * 2) + effective_spindle_count)

*“Formula koja se prilično dobro držala u mnogim benčmarcima godinama jeste da za optimalnu propusnost broj aktivnih konekcija treba da bude negde blizu ((broj_jezgara * 2) + efektivni_broj_hdd_osovina). Broj jezgara ne bi trebalo da uključuje HT niti, čak i ako je hyperthreading omogućen. Efektivni broj HDD osovina je nula ako je aktivni skup podataka u potpunosti keširan i približava se stvarnom broju osovina kako stopa pogodaka keša pada. … Do sada nije bilo nikakvih analiza koliko dobro formula funkcioniše sa SSD diskovima.”*[1]

[1] https://wiki.postgresql.org/wiki/Number_Of_Database_Connections

CONNECTION POOL

POPULARNE IMPLEMENTACIJE

- Hikari-CP[1]
- C3P0[2]
- Tomcat-JDBC[3]
- Apache common DBCP[4]

1 https://github.com/brettwooldridge/HikariCP

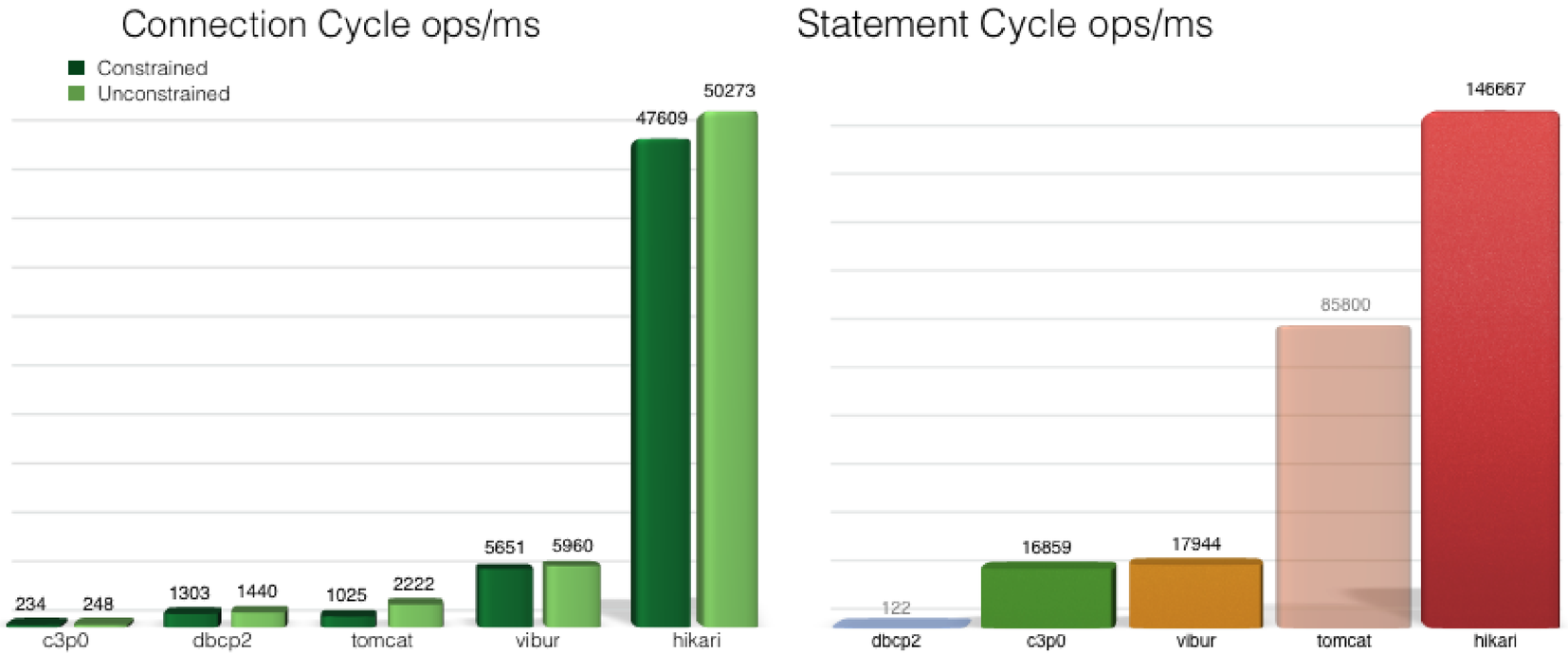
2 https://www.mchange.com/projects/c3p0/

3 http://tomcat.apache.org/tomcat-7.0-doc/jdbc-pool.html

4 http://commons.apache.org/proper/commons-dbcp/

CONNECTION POOL

KAKO SE POREDE POPULARNE IMPLEMENTACIJE?



- Connection Cycle predstavlja jednu konekciju od **`DataSource.getConnection()`** do **`Connection.close()`**
- Statement Cycle predstavlja jednu operaciju **`Connection.prepareStatement()`**, **`Statement.execute()`**, **`Statement.close()`**

Slika preuzeta sa https://github.com/brettwooldridge/HikariCP

SPIKE DEMAND

WELCOME TO THE JUNGLE[1]

- Spike demand predstavlja scenario u kome aplikacija u kratkom vremenu dobije ogroman nalet zahteva
- Stotine ili hiljade niti odjednom pokušava da uzme konekciju iz connection pool-a
- Ovo je najteži scenario za svaki pool, jer se tada vidi kako se ponaša kada *ubrzano ponestane slobodnih konekcija*

1 https://github.com/brettwooldridge/HikariCP/blob/dev/documents/Welcome-To-The-Jungle.md

SPIKE DEMAND

KAKO SE HIKARICP PONAŠA?

- HikariCP ima kontrolisan, asinhroni algoritam za širenje pool-a:
  - Ako pool ostane bez slobodnih konekcija pokrene ograničen broj operacija za dodavanje konekcije, ali ne beskonačno
  - Višak zahteva za dodavanje konekcije se odbacuje jer je već u toku jedno dodavanje
  - Pool raste postepeno, tačno koliko treba
  - Broj aktivnih konekcija se brzo stabilizuje
  - Broj blokiranih/čekajućih niti brzo pada ka nuli
  - Ne pravi „stampedo“ prema bazi

SPIKE DEMAND

KAKO SE NAIVNI POOL PONAŠA?

- Naivni (loše implementirani) pool obično radi sledeće:
  - Svaki put kada nema slobodne konekcije, odmah pokuša da otvori novu
  - Ako 100 niti ostane bez konekcije onda se napravi 100 pokušaja otvaranja nove konekcije
  - Server baze podataka dobija stampedo zahteva za kreiranje konekcija
  - Pool se previše proširi, ponekad i 2–3x iznad realne potrebe
  - Broj konekcija osciluje (skače gore-dole)
  - Veliki broj niti ostaje blokiran duže vreme

REFERENCE

- PRIMERI PO UZORU NA https://github.com/mbranko/isa19/tree/master/06-pooling
- PAVLO A., CMU. ADVANCED DATABASE SYSTEMS – NETWORKING https://15721.courses.cs.cmu.edu/spring2020/slides/11-networking.pdf
- RAASVELDT M., MUHLEISEN H., DON'T HOLD MY DATA HOSTAGE – A CASE FOR CLIENT PROTOCOL REDESIGN https://15721.courses.cs.cmu.edu/spring2020/papers/11-networking/p1022-muehleisen.pdf
- MIHALCHEA V. THE ANATOMY OF CONNECTION POOLING https://vladmihalcea.com/the-anatomy-of-connection-pooling/
- HIKARI-CP. ABOUT POOL SIZING https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing

KOJA SU VAŠA PITANJA?