

*Virtualne Java dretve
i Oracle "virtualne transakcije"
- idealan par?*

Zlatko Sirotić, univ. spec. inf.
ISTRA TECH d.o.o., Pula

C**A****S****E** konferencija **2026**
09.03. - 10.03.2026., Rijeka

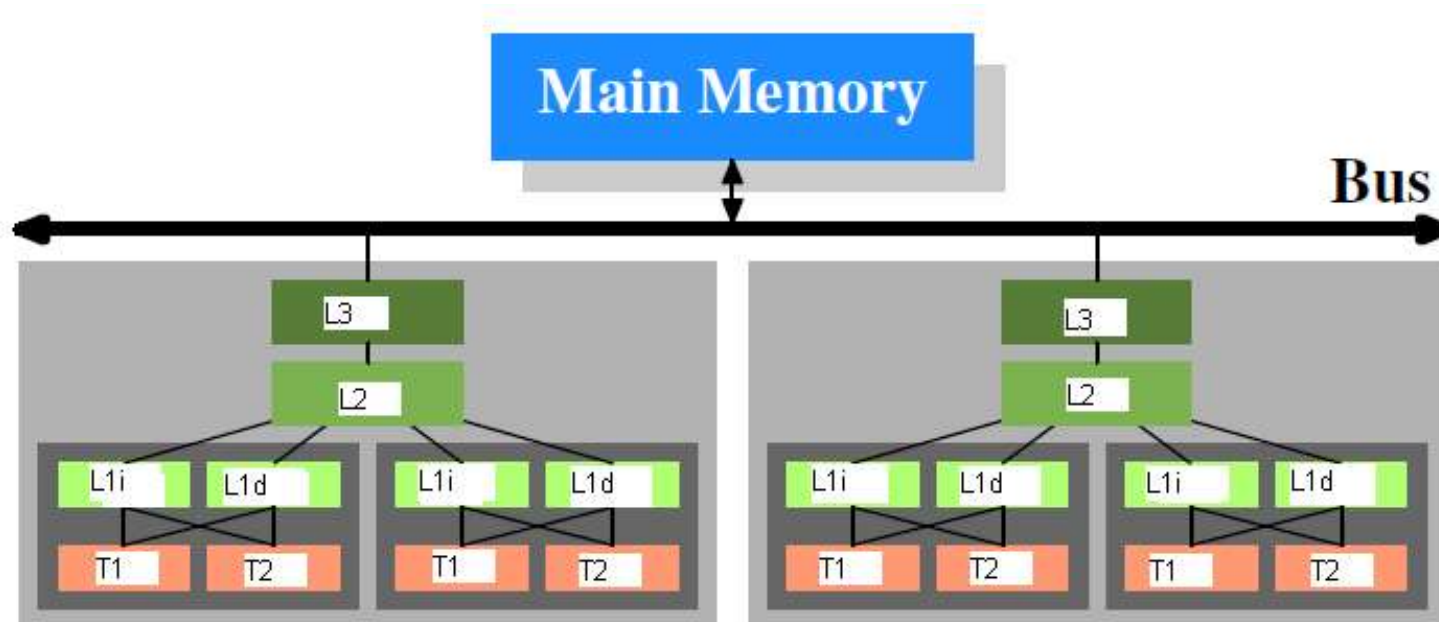
Teme

- ❑ Par slajdova s
 - CASE 2012 (CASE 24)
Utjecaj razvoja mikroprocesora na programiranje
 - CASE 2017 (CASE 29)
Paralelno programiranje u Javi
- ❑ Java virtualne dretve
- ❑ Oracle transakcije bez sesija (Sessionless Transactions)
 - "virtualne transakcije" (naš naziv)
- ❑ Da li nedostaje i "treći igrač"?

CASE 2012 ili (izvorni naziv) CASE 24

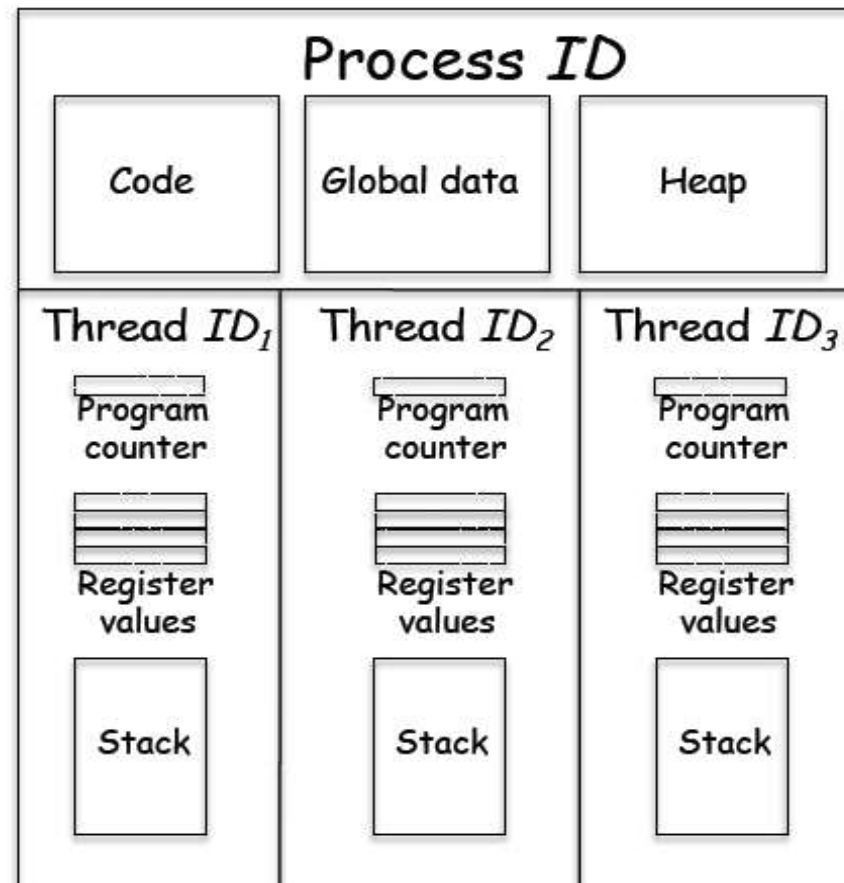
(Utjecaj razvoja mikroprocesora na konkurentno programiranje)

- ❑ Primjer sustava s dva dvojezgrejna mikroprocesorska čipa.
- ❑ Svaka jezgra podržava dvije hardverske dretve, koje dijele cache prve razine (L1i = instrukcijski cache, L1d = data cache). Jezgre dijele cache druge i treće razine (L2 i L3).
- ❑ Ukupno sustav ima 4 jezgre i 8 hardverskih dretvi:



CASE 2012

- ❑ Dretve OS-a dijele globalnu memoriju (programski kod i globalne podatke) i gomilu (heap) procesa OS-a, ali imaju vlastiti stog (stack) i kontekst dretve:



CASE 2017 ili CASE 29

(Paralelno programiranje u Javi)

Java 5/6 Executor, Java 7 ForkJoin, Java 8 Stream

PrimesExecutor	10000000	1	1000	Time (seconds) taken is	8.69
PrimesExecutor	10000000	2	1000	Time (seconds) taken is	4.57
PrimesExecutor	10000000	4	1000	Time (seconds) taken is	2.51
PrimesExecutor	10000000	8	1000	Time (seconds) taken is	1.76
PrimesExecutor	10000000	16	1000	Time (seconds) taken is	1.72

PrimesForkJoin	10000000	1	100	Time (seconds) taken is	8.89
PrimesForkJoin	10000000	2	100	Time (seconds) taken is	4.64
PrimesForkJoin	10000000	4	100	Time (seconds) taken is	2.59
PrimesForkJoin	10000000	8	100	Time (seconds) taken is	1.85
PrimesForkJoin	10000000	16	100	Time (seconds) taken is	1.79

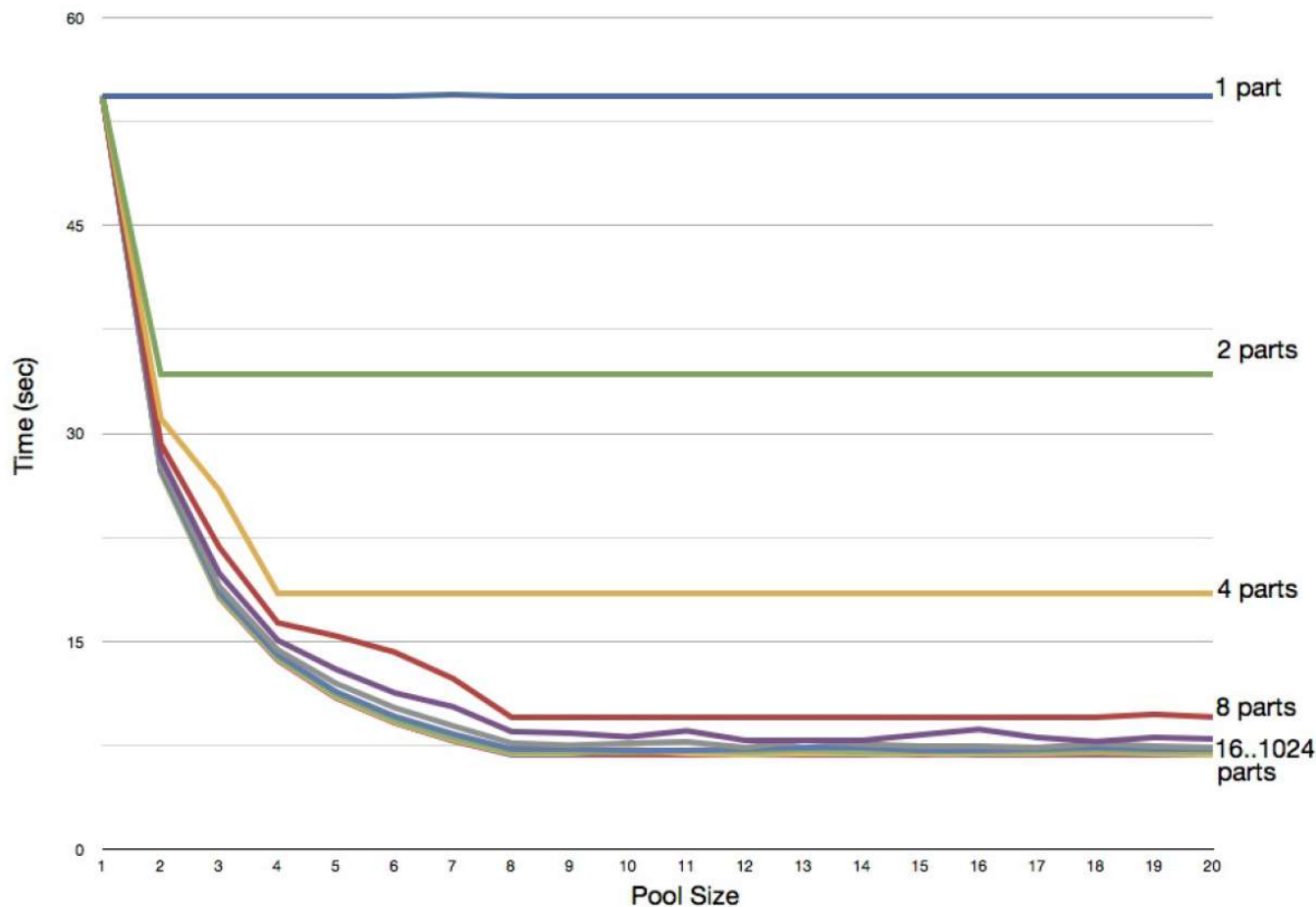
PrimesStream	10000000	0		Time (seconds) taken is	8.92
PrimesStream	10000000	1		Time (seconds) taken is	1.92

... Number of primes under 10000000 is 664579

CASE 2017

Java 5 Executor

- ❑ Brojanje prim brojeva na μP sa 8 HW dretvi (4 jezgre * 2)
 - prikaz efekta mijenjanja broja Java dretvi (PoolSize, os x)
 - i broja podzadataka (različite krivulje):



Java virtualne dretve

- ❑ Java je od početka imala Java dretve, koje se temelje na OS dretvama. Svaki Java program ima barem jednu dretvu, onu koja izvršava metodu main().
- ❑ Rad s više Java dretvi može poslužiti za (barem) dvije različite namjene:
 - **paralelizacija** (jednog) programa, kako bi program iskoristio sve hardverske procesore (hardverske dretve ili jezgre)
 - **konkurentno programiranje**, kod kojeg svaki korisnik za rješavanje svog zahtjeva dobiva Java dretvu (najčešće iz Java thread poola).

Java virtualne dretve

- ❑ Rad s Java dretvama nije jednostavan (paralelno i konkurentno programiranje uglavnom nije jednostavno) ali nije niti strašno komplicirano (barem od Java 5 dalje).
- ❑ Ali, Java dretve, budući da su temeljene na OS dretvama, imaju dvije značajne mane:
 - **utrošak vremena zbog context switch-a**
 - **broj OS dretvi je ograničen**; danas ih može biti nekoliko tisuća, ali nekoliko milijuna je obično previše.

Java virtualne dretve

- ❑ Java asinkrono programiranje (uključujući i reaktivno programiranje) rješava te probleme, ali asinkroni programi imaju sljedeće velike mane:
 - teški su za pisanje / čitanje / testiranje;
 - debug je praktički nemoguć;
 - profiliranje je praktički nemoguće.

Java virtualne dretve

- ❑ Java je u 9. mjesecu 2023. u (dugotrajnoj) verziji 21 dobila i Java virtualne dretve (prije verzije 21, to je bilo prototipno rješenje).
- ❑ Dok su Java ne-virtualne (platformske) dretve povezane 1:1 s dretvama operacijskog sustava, više Java virtualnih dretvi može se izvoditi na jednoj platformskoj dretvi, tj. na jednoj dretvi operacijskog sustava. Time je broj Java virtualnih dretvi praktički neograničen.
- ❑ Java virtualne dretve omogućavaju pisanje programskog koda koji je:
 - efikasan kao i asinkroni kod,
 - ali je čitljiviji, puno lakši za debugiranje
 - i daje jasnije greške (exception) u odnosu na asinkroni kod.

Java virtualne dretve

- ❑ Java dokumentacija objašnjava za što virtualne dretve (ni)su namijenjene:

"Use virtual threads in high-throughput concurrent applications, especially those that consist of a great number of concurrent tasks that spend much of their time waiting. ...Virtual threads are not faster threads; they do not run code any faster than platform threads. They exist to provide scale (higher throughput), not speed (lower latency)."

- ❑ Analizirali smo **jesu li virtualne dretve (značajno) sporije** od klasičnih (ne-virtualnih) dretvi kod paralelnih programa.

Java virtualne dretve

- ❑ Uzeli smo tri naša stara programa koji koriste ne-virtualne dretve na tri različita načina (Executors, ForkJoin framework, Streams) i napravili program koji koristi virtualne dretve tako da smo samo malo mijenjali stari program s Executors.
- ❑ Usporedili smo njihovu brzinu izvršavanja na dva računala
 - Intel i7 čip 7. generacije, ima 8 HW dretvi = $4 \text{ jezgre} * 2$
 - Intel i7 čipom 13. generacije, ima 20 HW dretvi = $6 * 2$ (jake jezgre, imaju Hyper-Threading) + 8 (bez HT)
- ❑ Virtualne dretve se interno temelje na ForkJoin frameworku, koji ima default broj Java dretvi, tako da kod tog novog programa korisnik ne unosi broj dretvi (isto kao i kod starih programa koji koriste ForkJoin framework i Streams).
- ❑ Primijetimo da ne bi imalo smisla raditi s virtualnim dretvama kod ForkJoin i Streams varijanti (Streams se temelji na ForkJoin), jer bismo onda imali "ForkJoin iznad ForkJoin".

Java virtualne dretve

- ❑ Glavna razlika u programima:
- ❑ Stari program, s ne-virtualnim dretvama

```
final ExecutorService executorPool =  
    Executors.newFixedThreadPool(poolSize);
```

- ❑ Novi program, s virtualnim dretvama

```
final ExecutorService executorPool =  
    Executors.newVirtualThreadPerTaskExecutor();
```

Java virtualne dretve

Broj Java dretvi	Executors vrijeme (s) na Intel i7 7700HQ / 13700H	Executors s Java virtualnim dretvama	ForkJoin	Streams
1	7.52 / 2.01	-	7.72 / 2.20	7.20 / 2.01
2	4.03 / 1.01	-	4.09 / 1.12	-
4	2.20 / 0.51	-	2.25 / 0.58	-
8	1.54 / 0.31	1.56 / -	1.56 / 0.36	1.63 / -
12	1.54 / 0.27	-	1.60 / 0.31	-
16	1.54 / 0.23	-	1.60 / 0.28	-
20	1.54 / 0.22	- / 0.23	1.62 / 0.28	- / 0.22
32	1.54 / 0.22	-	1.64 / 0.29	-

Oracle "virtualne transakcije"

- ❑ Oracle je u bazi 26ai (staro ime: 23ai) uveo "transakcije bez sesija" (Sessionless Transactions), koje smo ovdje nazvali "virtualne transakcije", kako bi naziv bio sličan virtualnim Java dretvama. Tu mogućnost ima i SE edicija Oracle baze.
- ❑ Standardno, upravljanje transakcijom zahtijeva da resursi veze i sesije budu vezani za transakciju tijekom cijelog njezina životnog ciklusa.
- ❑ Stoga se sesija/veza može osloboditi tek nakon što je transakcija završila. To često rezultira nedovoljnim korištenjem sesija/veza.
- ❑ Kod transakcija bez sesija, nakon što pokrenemo transakciju, imamo fleksibilnost obustave i nastavka transakcije tijekom njezina životnog ciklusa. Sesija/veza može se vratiti u skup slobodnih sesija/veza i mogu je ponovno koristiti druge transakcije.

Oracle "virtualne transakcije"

- ❑ Transakcije bez sesija omogućuju aplikacijama da obustave i nastavu transakcije među sesijama/vezama (jedna instanca ili RAC) bez potrebe za vanjskim upraviteljem transakcija i bez potrebe da aplikacija koordinira protokole potvrđivanja i oporavka.
- ❑ Baza podataka upravlja životnim ciklusom transakcija, uključujući potvrđivanje i oporavak.
- ❑ Performanse i propusnost aplikacije imaju koristi od smanjene latencije potvrđivanja, jer je potrebno manje kružnih putovanja klijent-poslužitelj.
- ❑ Budući da vanjska koordinacija nije potrebna, korištenje transakcija bez sesija rezultira znatno pojednostavljenom infrastrukturom srednjeg sloja ili sloja aplikacije i značajno smanjuje vrijeme zastoja u usporedbi s vanjskom koordinacijom transakcija (kao što je s XA).

Oracle "virtualne transakcije"

- ❑ Npr. pretpostavimo da korisnik pokreće transakciju kroz sesiju 1, nastavlja kroz sesiju 2 i završava kroz sesiju 3:
- ❑ Session 1
 - **Start transaction** // Returns a transaction identifier
 - Insert row ('Smith', 800)
 - **Suspend the transaction** and Release session
- ❑ Session 2
 - **Resume the transaction** using the transaction identifier
 - Insert row ('Allen', 1600)
 - **Suspend the transaction** and Release session
- ❑ Session 3
 - **Resume the transaction** using the transaction identifier
 - Select inserted rows // Returns rows for Smith and Allen
 - Commit

Oracle "virtualne transakcije"

- ❑ Odsječak JDBC koda koji starta transakciju, radi DML, pa suspendira transakciju (čime otpušta sesiju):

```
final byte[] gtrid = oc.startTransaction() ;

try (PreparedStatement p = c.prepareStatement
    ("insert into orders(product_id,quantity,total_price)
     values(?, ?, ?)")
)
{ p.setInt(1,1);
  p.setInt(2,3);
  p.setBigDecimal(3, new BigDecimal(19.90).multiply
    (new BigDecimal(3)));
  p.executeUpdate();
}

oc.suspendTransactionImmediately() ;
```

Oracle "virtualne transakcije"

- ❑ Odsječak JDBC koda koji ponovno pokreće transakciju (u drugoj sesiji), opcionalno radi neki DML, pa COMMIT:

```
System.out.print("Please provide a global trans.ID: ");
System.out.flush();
byte[] buffer = System.in.readNBytes(32);
try (Connection c = Oracle.dataSource.getConnection())
{
    c.setAutoCommit(false);
    final OracleConnection oc =
        c.unwrap(OracleConnection.class);
    final byte[] gtrid =
        HexFormat.of().parseHex(new String(buffer));
    oc.resumeTransaction(gtrid);
    ... DML
    c.commit();
    System.out.println("Transaction committed!");
}
```

Oracle "virtualne transakcije"

- ❑ Odsječak PL/SQL koda koji starta transakciju, radi neki DML, pa suspendira transakciju:

```
declare
    gtrid VARCHAR2(128);
begin
    gtrid := DBMS_TRANSACTION.START_TRANSACTION (
        XID => UTL_RAW.CAST_TO_RAW('my_test1'),
        transaction_type =>
            DBMS_TRANSACTION.TRANSACTION_TYPE_SESSIONLESS,
        timeout => 2000,
        flag => DBMS_TRANSACTION.TRANSACTION_NEW
    );
    dbms_output.put_line('GTRID is: ' || gtrid);
end;
... DML
execute DBMS_TRANSACTION.SUSPEND_TRANSACTION;
```

Oracle "virtualne transakcije"

- ❑ Odsječak PL/SQL koda koji ponovno pokreće transakciju:

```
declare
    gtrid VARCHAR2(128);
begin
    gtrid := DBMS_TRANSACTION.START_TRANSACTION (
        xid => UTL_RAW.CAST_TO_RAW('my_test1'),
        transaction_type =>
            DBMS_TRANSACTION.TRANSACTION_TYPE_SESSIONLESS,
        flag => DBMS_TRANSACTION.TRANSACTION_RESUME
    );
    dbms_output.put_line('Resumed GTRID: '||gtrid);
end;
```

Java virtualne dretve + "virtualne transakcije"

- ❑ Što se (potencijalno) mijenja na razini modela izvršavanja?
- ❑ Prije (klasični model), u tipičnoj web aplikaciji:
 - 1 HTTP request → 1 platformska dretva
 - 1 HTTP request → 1 DB konekcija iz connection poola
 - 1 HTTP request → 1 transakcija
 - transakcija mora završiti (COMMIT/ROLLBACK) prije slanja HTTP response

Java virtualne dretve + "virtualne transakcije"

- ❑ Virtualne dretve omogućuju:
 - jednostavan sekvencijalni kod
 - bez reactive framework kompleksnosti
 - bez callback komplikacija
- ❑ Sessionless transakcije omogućuju:
 - pauziranje transakcije bez držanja konekcije
 - dugotrajne poslovne tokove
 - razdvajanje HTTP requesta i DB transakcijskog konteksta
- ❑ Zajedno:
 - možemo imati milijune logičkih tokova (virtualne dretve)
 - bez potrebe za milijun DB konekcija
 - bez reactive paradigme
- ❑ **To je arhitekturno nova situacija!**

Java virtualne dretve + "virtualne transakcije"

- ❑ Primjeri gdje bi to moglo biti jako korisno:
 - dugotrajni poslovni workflow
 - mikroservisi - bez saga
 - dramatično veći concurrency uz manji connection pool

- ❑ Ali, dok ova kombinacija može dramatično promijeniti orkestraciju dugotrajnih procesa, ona neće omogućiti da se transakcije protežu preko "korisničkog vremena za razmišljanje" (User Think Time).

- ❑ Razlog: zaključavanje redaka.
Iako Sessionless Transaction oslobađa sesiju (RAM i TCP socket na bazi), ona NE oslobađa zaključane retke u tablicama.

Java virtualne dretve + "virtualne transakcije"

- ❑ Ako web aplikacija dopusti da transakcija traje kroz više HTTP requesta, npr. korisnik dobije formu, a transakcija je pauzirana dok on ne klikne "Save", nakon 5 minuta:

- te retke (rows) baza ima zaključane 5 minuta
- niti jedan drugi proces ih ne može mijenjati
- to može voditi do loših performansi baze (lock contention) i povećanja vjerojatnosti za deadlock (istina, Oracle baza sama detektira deadlock i omogućava da se on lako programski riješi).

"Treći igrač" u Oracle 26ai

- ❑ Oracle baza 26ai je uvela i ovo: Lock-Free Reservation, tj. stupce koji se mogu rezervirati (reservable columns). Nažalost, tu mogućnost ima samo skuplja EE edicija.
- ❑ **Facilitates multiple concurrent updates on a numeric column without traditional locking mechanisms.**
- ❑ Transactions can concurrently add or subtract from the same **row's reservable column** without waiting for others to commit.
- ❑ Updates proceed based on specified conditions for the reservable column.
- ❑ **Reservable column updates do not lock rows, allowing concurrent updates to non-reservable columns of the same row.**

"Treći igrač" u Oracle 26ai

- ❑ Neka (sadašnja) ograničenja kod Lock-Free Reservation, za stupce koji se mogu rezervirati (reservable columns):
- ❑ A reservable column can be specified for Oracle numeric data type columns only.
- ❑ A reservable column cannot be a Primary Key or an identity column (or virtual column) because the reservable column is an aggregate type.
- ❑ A user table can have at most ten reservable columns.
- ❑ User tables that have reservable columns must have a Primary Key.
- ❑ Indexes are not supported on reservable columns.
- ❑ Composite reservable columns are not allowed.

"Treći igrač" u Oracle 26ai

❑ Primjer PL/SQL koda bez reservable columns:

```
CREATE TABLE account (  
    id NUMBER PRIMARY KEY,  
    name VARCHAR2(10),  
    balance NUMBER CONSTRAINT min_bal CHECK (balance >= 50));  
  
DECLARE current NUMBER;  
BEGIN  
    SELECT balance INTO current -- Read and Lock account balance  
        FROM account  
        WHERE id = 12345 FOR UPDATE;  
    IF current >= 75 THEN  
        PurchaseItem(); -- Sufficient funds: Perform item purchase  
        UPDATE account -- Debit account balance and commit  
            SET balance = balance - 25  
            WHERE id = 12345;  
        COMMIT;  
    ELSE  
        ROLLBACK; -- Insufficient funds, so cancel  
    END IF;  
END;
```

"Treći igrač" u Oracle 26ai

❑ Primjer PL/SQL koda s reservable columns:

```
CREATE TABLE account (  
    id NUMBER PRIMARY KEY,  
    name VARCHAR2(10),  
    balance NUMBER RESERVABLE CONSTRAINT min_bal CHECK (balance >= 50));  
  
BEGIN  
    -- Reserve 25 from account balance  
    UPDATE account  
        SET balance = balance - 25  
        WHERE ID = 12345;  
    -- If reservation succeeds perform item purchase  
    PurchaseItem();  
    -- The commit finalizes the balance update  
    -- This gets the account row lock  
    COMMIT;  
EXCEPTION  
    -- This indicates that the reservation failed  
    WHEN Check_Constraint_Violated THEN  
        ROLLBACK;  
END;
```

"Treći igrač" u Oracle 26ai

- ❑ S ovim "trećim igračem", vjerujemo da je moguća arhitektura koja je neblokirajuća na svim slojevima:
 - "Aplikacijski sloj" (Java Virtual Threads):
Java dretva ne troši OS dretvu dok čeka vanjski servis
 - "Mrežni sloj" (Sessionless Tx): konekcija prema bazi se ne blokira (ne troši TCP socket/pool) dok aplikacija čeka
 - "Podatkovni sloj" (Lock-Free Reservation):
mijenjani redak u bazi ne sprečava druge transakcije.

"Treći igrač" u Oracle 26ai

❑ Primjer poslovnog toka (Workflow) koji postaje moguć:

- HTTP Request (npr. kada korisnik klikne "Kupi")
- Start Tx: Počinje transakcija
- Lock-Free Reservation: Rezerviraj proizvod (npr. UPDATE inventory SET qty = qty – 1 s uključenom opcijom lock-free); redak NIJE zaključan za druge kupce
- Suspend Tx: Vрати konekciju u pool
- External Call: Virtualna dretva zove Payment Gateway (što traje relativno dugo)
- Resume Tx: Uzmi novu konekciju, nastavi transakciju
- Baza sada provjerava: "Je li stanje skladišta još uvijek dovoljno nakon svih rezervacija koje su se u međuvremenu dogodile?"; ako da, Success (i COMMIT)

Literatura (dio)

- ❑ Boyarsky, J., Selikoff, S. (2015): OCP: Oracle Certified Professional Java SE 8 Programmer II Study Guide, Sybex
- ❑ Göetz B. and others (2006): Java Concurrency in Practice, Addison-Wesley
- ❑ P-Y. Saumont (2017): Functional Programming in Java, Manning
- ❑ Oracle (2024): Java Platform SE Core Libraries (Release 21), <https://docs.oracle.com/en/java/javase/21/core/>
- ❑ Oracle (2025): Oracle® AI Database – Database Development Guide - 26ai, <https://docs.oracle.com/en/database/oracle/oracle-database/26/adfns/index.html>
- ❑ Paumard, J. (2024): Are Virtual Threads Going to Make Reactive Programming Irrelevant?, <https://2024.javazone.no/program/>