Imutabilne i blockchain tablice u Oracle 19c

Zlatko Sirotić, univ.spec.inf.

ISTRA TECH d.o.o., Pula

ISTRA TECH d.o.o. & me

- ISTRA TECH je novo ime (od 2015.) poduzeća **Istra informatički inženjering**, osnovanog 1990. godine.
- Radim na informatičkim poslovima od 1984. godine.
- Oracle softverske alate (baza, Designer CASE, Forms 4GL, Reports, Java) koristim oko 25 godina.
- Objavljivao sam stručne radove na kongresima / konferencijama HrOUG, JavaCro, CASE, KOM, "Hotelska kuća", te u časopisima "Mreža", "InfoTrend" i "UT".
- Neka moja programska rješenja objavljivana su na web stranicama firmi Oracle i Quest.
- Vanjski sam suradnik na Fakultetu informatike Pula (FIPU) od početka rada (2011./2012.).

HrOUG / JavaCro & me

- Prvi put predavač na HrOUG 2002.
  Sudjelovao sam 18 puta (ne uključujući ovu godinu)
  i održao 24 predavanja.

- Uz 24 prezentacije, za HrOUG sam napisao 15 radova,
  ukupno preko 300 stranica teksta.

- Prvi put predavač na JavaCro 2014.
  Sudjelovao sam 4 puta (ne uključujući ovu godinu)
  i održao 4 predavanja.
  Ove godine održao sam predavanje:
  Java i Prolog: Ima neka (polu)tajna veza

HrOUG predavanja iz prošlog desetljeća

- **2011. a) Konkurentno programiranje – Oracle baza, Java, Eiffel (najbolje ocijenjeno predavanje 16. konferencije)**
- 2011. b) Kriptografija u Oracle bazi
- 2012. a) Ima neka loša veza - priča o in-doubt distribuiranim transakcijama
- 2012. b) Visoka konkurentnost na JVM-u
- **2013. Transakcije i Oracle - baza, Forms, ADF (63 stranice – od tada sam odustao od pisanja radova :)**
- 2014. Nasljeđivanje je dobro, naročito višestruko - Eiffel, C++, Scala, Java 8
- 2015.a) Povratak u Prolog - verzija 1
- 2015 b) Kada Oracle naredba nije serijabilna?
- 2018. Testiranje konkurentnih transakcija
- 2019. Funkcijska paradigma i baze podataka

HrOUG predavanja iz ovog desetljeća

2021.

- ARM arhitektura i Oracle

Izazov 1 – nestašica čipova - više nije!

Izazov 2 – Nvidia želi kupiti Arm - nije im uspjelo!

Izazov 3 – Relaxed memory consistency model

Pitao sam: "A gdje je Oracle baza na ARM arhitekturi?"

Evo je: Oracle Database 19c for LINUX ARM (aarch64)

- Jedu li krave travu?

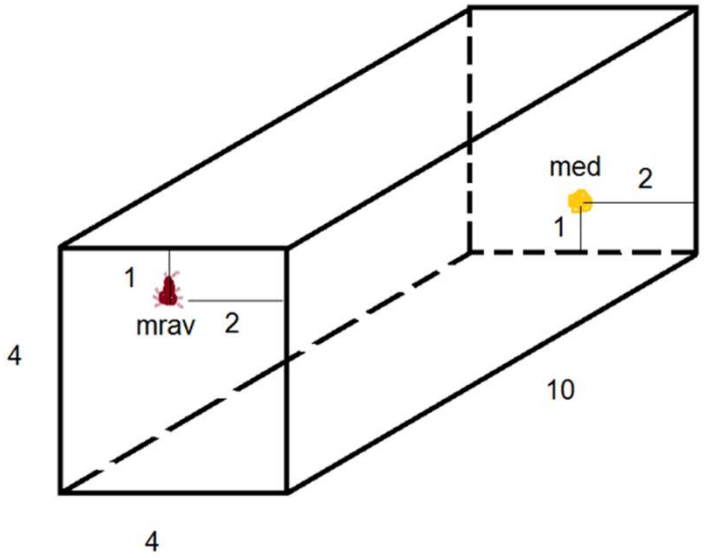
2022.

- Kako debagirati (engl. debugging) Forms programe

- JDBC i Oracle baza

Mrav i med na prizmi (i valjku)

- Na HrOUG 2015. prvi put sam spomenuo Mrav i med na prizmi (verzija 0).



- Na HrOUG 2019. sam najavio prezentaciju verzije 1.

- Na stranici http://www.istratech.hr/mrav-i-med-na-prizmi-i-valjku-verzija-2/ nalazi se najnovija verzija Mrav i med na prizmi - verzija 2

Uvod

- Zadnjih nekoliko godina, **funkcijsko programiranje** stječe veliku popularnost u odnosu na imperativno programiranje (napomena: objektno-orijentirano programiranje je također imperativno). Npr. 2014. godine je i Java jezik dobio neke značajne funkcijske mogućnosti (verzija Java 8).
- Funkcijsko programiranje nije novo. **Prvi funkcijski jezik Lisp nastao je davne 1958. godine**, godinu dana nakon jezika Fortran i godinu dana prije jezika COBOL.
- Funkcijsko programiranje, kao i logičko programiranje, po nečemu je vrlo slično programiranju u SQL-u - **visoko je deklarativno**.
- U prezentaciji ćemo se osvrnuti na funkcijsko programiranje, dati par ideja za proširenje jezika PL/SQL nekim funkcijskim mogućnostima, te vidjeti kako su u Oracle 19c uvedene blockchain tablice i imutabilne tablice.

Teme

- Chomskyjeva hijerarhija jezika
- Lambda račun i funkcijsko programiranje
- Funkcijski jezik Haskell
- Objektno-funkcijski jezik Scala
  i funkcijska proširenja u jeziku Java
- Ideje za uvođenje nekih funkcijskih osobina
  u Oracle PL/SQL
- Razmišljanje o "funkcijskom radu" s podacima
  u bazi podataka (te ideje nisu nove, zagovarao ih je još 80-tih Jim Gray, znanstvenik na području baza podataka i transakcijskih sustava, dobitnik Turingove nagrade 1998. godine, ali su u zadnje vrijeme ponovno "moderne")
- Nove mogućnosti Oracle DBMS-a 19c (od verzije 19.10/11)
  – blockchain tablice i imutabilne tablice
  (plus nove mogućnosti u Oracle 23c).

Chomskyjeva hijerarhija jezika

- Kako kaže biolog i matematičar M.Nowak, (ljudski) jezik je najvažnija invencija prirode nakon što su se prije oko 600 milijuna godina pojavili prvi razvijeni višestanični organizmi. (Prije toga važna je bila pojava prokariota, prije oko 3500 milijuna godina, i eukariota, prije oko 1500 milijuna godina.)
- Slavni američki lingvist i filozof (ali i neumorni politički aktivist) **Noam Chomsky** još je 50-ih godina prošlog stoljeća napravio poznatu klasifikaciju jezika (ljudskih i formalnih).
- Chomskyjeva je hijerarhija jezika postala važna u računarstvu, naročito u konstrukciji jezičnih procesora i teoriji automata.
- **Ta je klasifikacija dala vezu između određenih jezika, gramatika koje ih opisuju i generiraju nizove znakova u tim jezicima, te (apstraktnih) automata koji prihvaćaju rečenice u tim jezicima.**

Hijerarhija (formalnih) jezika / gramatika / automata

- Chomsky je izvorno dao tipove 0, 1, 2, 3 (kasnije su nađena tri međutipa, koja nemaju brojeve).
- Tip 0 predstavlja najjači jezik, gramatiku i automat. Dakle, **Turingov stroj** je najjači automat. Nijedno računalo ne može riješiti problem koji ne može riješiti Turingov stroj.

Teorija automata: formalni jezici i formalne gramatike			
Chomskyjeva hijerarhija	Gramatike	Jezici	Minimalni automat
Tip 0	Neograničenih produkcija	Rekurzivno prebrojiv	Turingov stroj
n/a	(nema uobičajenog imena)	Rekurzivni	Odlučitelj
Tip 1	Kontekstno ovisna	Kontekstno ovisni	Linearno ograničen
n/a	Indeksirana	Indeksirani	Ugniježđenog stoga
Tip 2	Kontekstno neovisna	Kontekstno neovisni	Nedeterministički potisni
n/a	Deterministička kontekstno neovisna	Deterministički kontekstno neovisni	Deterministički potisni
Tip 3	Regularna	Regularni	Konačni
Svaka kategorija jezika ili gramatika je pravi podskup nadređene kategorije.			

Lambda račun

- Lambda izrazi (ili lambda funkcije), imaju teoretsko porijeklo u **lambda računu** (lambda calculus), kojega je sredinom 30-ih godina prošlog stoljeća kreirao **Alonzo Church**.
- Poznata je **Church-Turingova hipoteza** (inače, **Alan Turing** je kod Alonza Churcha radio doktorat), koja se ne može matematički dokazati, već se smatra intuitivno prihvatljivom. Ona (pojednostavljeno, te u današnjoj terminologiji) kaže da sve što se efektivno može izračunati, može se izračunati pomoću lambda računa ili **Turingovog stroja**.
- 50-ih godina se formalno dokazalo da postoje i neki drugi sustavi koji su njima ekvivalentni: Gödelove rekurzivne funkcije, Postov sustav, Markovljevi algoritmi i dr.
- **Bez obzira na teoretsku ekvivalentnost, Turingovi strojevi su po svom ponašanju bliži imperativnoj programskoj paradigmi, dok je lambda račun teoretska osnova za funkcijske programske jezike.**

Funkcijsko programiranje

- Programske jezike moglo bi se grubo podijeliti na **imperativne** (objektne i ne-objektne) i **deklarativne**.
- Kod imperativnih jezika naglasak je: KAKO nešto napraviti, a kod deklarativnih: ŠTO treba napraviti.
- Deklarativni jezici su najčešće funkcijski jezici ili logički jezici (npr. Prolog), ali i SQL (koji nije niti funkcijski, niti logički).
- Kako je već rečeno, prvi funkcijski jezik je Lisp (i još se koristi). Danas je najpoznatiji **Haskell**.
- U zadnjih nekoliko godina se **povećao interes za funkcijske jezike,** naročito zbog toga što se drži da su funkcijski jezici bolji za konkurentno i paralelno programiranje.

Osobine funkcijskih jezika

- Funkcije višeg reda (higher-order functions)
- Leksičko zatvaranje (lexical closure)
- Podudaranje (sparivanje) uzorka (pattern matching)
- Jednokratno pridruživanje (single assignment)
- Lijena evaluacija (lazy evaluation)
- Zaključivanje o tipovima (type inference)
- Eliminacija repnog poziva (tail call elimination, TCE)
- List comprehension: kompaktan i ekspresivan način definiranja listi i rada s listama, kao osnovnih podatkovnih struktura funkcijskog programa
- Monadički efekti (monadic effects)

Čiste funkcije

- Kod funkcijskog programiranja, najviše se govori o tome da bismo trebali (što više) koristiti **čiste funkcije** (pure functions), a s time je povezan i izraz **referencijalna transparentnost** (referential transparency).
- Čiste funkcije:
  - za iste argumente uvijek daju istu povratnu vrijednost
  - nemaju vanjske efekte; vanjski efekti su npr. mijenjanje neke globalne varijable (vidljive izvan funkcije), rad s bazom podataka, čitanje s tipkovnice, pisanje na ekran …
- Jasno je da je nemoguće napraviti koristan softver koji bi se sastojao isključivo od čistih funkcija. Cilj je da većina funkcija bude čista, a da "nečiste radnje" stavimo u posebne funkcije.
- Napomena: poznato je da u Oracle SQL upitu možemo koristiti funkcije, ali one moraju zadovoljavati neke uvjete, minimalno da ne koriste DML i ne mijenjaju pakirane varijable.

Haskell – kratka povijest

- Peter Landin je 1960-ih napravio **ISWIM**, prvi čisti funkcijski jezik strogo temeljen na lambda računu, bez naredbi pridruživanja. John Backus je 1970-ih napravio **FP**, funkcijski jezik koji je imao funkcije višeg reda, a Robin Milner je napravio **ML**, prvi moderni funkcijski jezik, koji je uveo zaključivanje o tipovima (type inference) i polimorfične tipove. 1970-ih i 1980-ih David Turner 70-ih je izradio niz lijenih (lazy) funkcijskih jezika, završno sa sustavom **Miranda**.
- 1987. je internacionalni komitet započeo na izradi jezika Haskell. 1990-ih je Phil Wadler kreirao klase tipova (type classes) i monade (monads), što je glavna inovacija koju je donio Haskell.
- **2003. je publiciran Haskell Report**, koji je definirao prvu stabilnu verziju jezika Haskell. Ažurirana verzija publicirana je 2010. (Haskell 2010.). Sljedeća verzija bit će Haskell 2020.

Haskell - definiranje funkcija

- Konvencija je da se prvo navede **tip funkcije** (u drugim jezicima bismo rekli deklaracija ili signatura), a onda **definicija funkcije**:

```
add :: (Int, Int) -> Int
add (x, y) = x + y

zeroto :: Int -> [Int]
zeroto n = [0..n]
```

Haskell - curried functions

- Drugi način prikazivanja funkcija koje imaju dva ili više ulaznih argumenata (n-torku argumenata) su **curryzirane funkcije** (curried functions), koje se tako zovu po prezimenu matematičara koji ih je izmislio - **Haskell Curry**.
- Npr. funkcija add, koja je imala dva argumenta, može se pretvoriti u funkciju s jednim argumentom koja vraća drugu funkciju:

```
add' :: Int -> (Int -> Int)
add' x y = x + y
```

- **Curryzirane funkcije** su fleksibilnije, jer se parcijalnom primjenom argumenata često mogu napraviti druge korisne funkcije. Npr. iz add' možemo ovako dobiti funkciju koja povećava (neki) broj za 1:

```
add' 1 :: Int -> Int
```

Haskell - rekurzivne funkcije

- Funkcijski jezici obilato koriste rekurziju, jer imaju **optimizaciju** kojom (često) mogu izbjeći dešavanje greške prelijevanja stoga (**stack overflow**):

```
reverse :: [a] -> [a]
reverse []     = []
reverse (x:xs) = reverse xs ++ [x]
```

- I neki ne-funkcijski jezici imaju **eliminaciju repnog poziva kod rekurzije** (tail call elimination ili tail call optimisation), tj. onda kada je zadnja operacija u kodu ("na repu koda") poziv funkcije same.
- Tu mogućnost imaju npr. Scala, Kotlin, C++ (svi kompajleri), a nemaju npr. Java, PL/SQL.

Haskell - funkcije višeg reda

- **Funkcije višeg reda** (higher order functions - HOF) su takve funkcije koje kao argument ili/i kao povratnu vrijednost imaju (neku drugu) funkciju.
- Budući da sve **curryzirane funkcije** imaju kao povratnu vrijednost (drugu) funkciju, najčešće se funkcijama višeg reda zovu samo one koje imaju (drugu) funkciju kao argument.

```
map :: (a -> b) -> [a] -> [b]
map f []     = []
map f (x:xs) = f x : map f xs
```

```
> map (+1) [1, 3, 5, 7]
[2,4,6,8]
```

Scala – kratka povijest

- Programski jezik Scala kreirao je **Martin Odersky**, profesor na Ecole Polytechnique Fédérale de Lausanne (EPFL).
- Krajem 80-ih doktorirao je na ETH Zürich kod profesora Niklausa Wirtha (kreatora Pascala i Module-2).
- Nakon toga naročito se **bavio istraživanjima u području funkcijskih jezika**, zajedno sa kolegom Philom Wadlerom (jednim od dva glavna kreatora funkcijskog jezika Haskell).
- Kada je izašla Java, Odersky i Wadler su 1996. napravili jezik **Pizza** nad JVM-om. Na temelju projekta Pizza, napravili su 1997./98. **Generic Java (GJ)**, koji je uveden u Javu 5 (malo ga je nadopunio Gilad Bracha, sa wildcardsima).

Scala - osnovne osobine

- **Scala je čisti objektno-orijentirani jezik** (sa statičkom provjerom tipova). Osim toga, na temelju objektno-orijentiranih mogućnosti izgrađene su i brojne funkcijske mogućnosti, **tako da je Scala i funkcijski jezik (ali nije čisti)**.
- Sa funkcijskim osobinama došle su i neke osobine koje su vrlo pogodne za **konkurentno programiranje**.
- **Scala je izvrstan jezik i za pisanje DSL-ova** (Domain-Specific Language), jezika za specifičnu problemsku domenu.
- No, važno je da se može programirati u Scali bez da se napusti Java, jer se **Java i Scala programski kod mogu jako dobro upotpunjavati**.

Scala - funkcije višeg reda

- Funkcije koje kao argument ili povratnu vrijednost imaju neku drugu funkciju, zovu se funkcije višeg reda (ovdje je to **suma**):

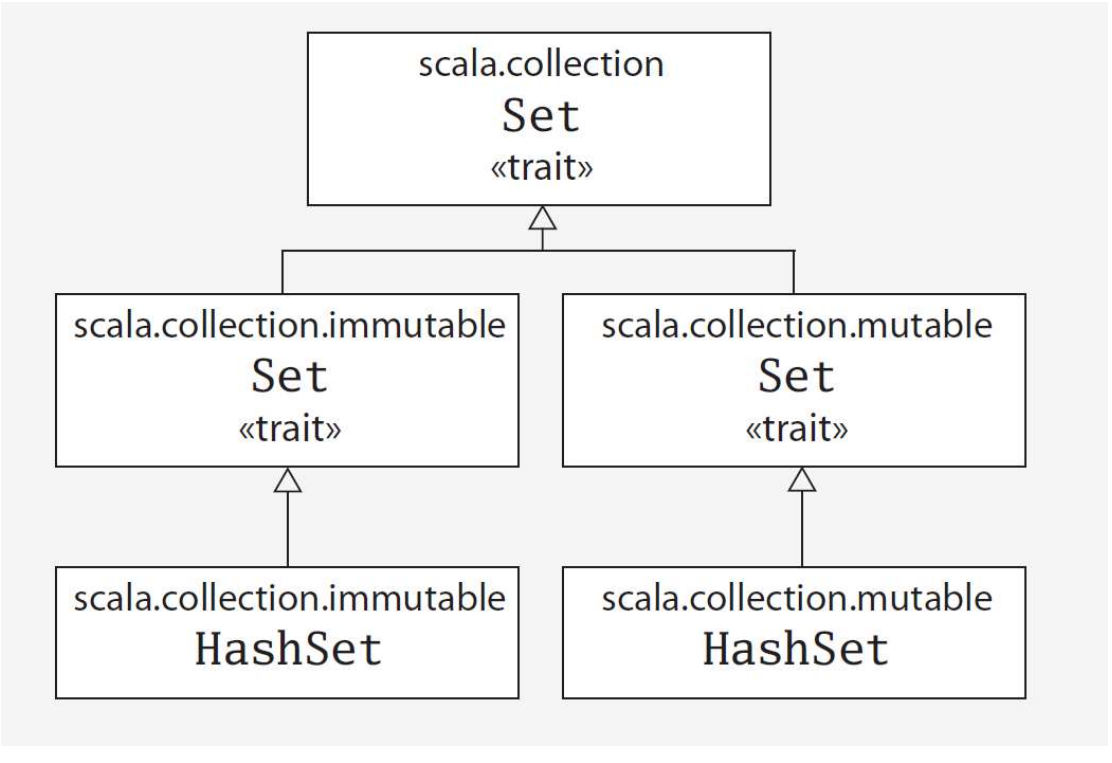
```
def suma(f: Int => Int, a: Int, b: Int): Int =
  if (a > b) 0 else f(a) + suma(f, a + 1, b)
```

- Sada definiramo dvije funkcije i koristimo ih (za punjenje vrijednosti val varijabli) kao 1. parametar funkcije suma:

```
def kvadrat(x: Int): Int = x * x
def dvaNaNtu(x: Int): Int =
  if (x == 0) 1 else 2 * dvaNaNtu(x - 1)
val sumaKvadrata = suma(kvadrat, 1, 5)   // = 55
val sumaDvaNaNtu = suma(dvaNaNtu, 1, 5) // = 62
```

Scala – kolekcije, imutabilne i mutabilne

- Funkcijski jezici poznati su po tome da imaju izvrstan način rada sa kolekcijama, naročito sa listama (list).
- Glavne Scala kolekcije su **List, Set i Map**. List je uređena kolekcija objekata, Set je neuređena kolekcija objekata, a Map je skup parova (ključ, vrijednost).



Scala - lijena evaluacija (lazy evaluation)

- Inicijalizacija vrijednosti odgađa se do trenutka kada se (lijena) vrijednost prvi put koristi:

```
class Radnik (id: Int, ime: String,
  managerId: Int) { // bez lijene evaluacije
  val manager: Radnik = Db.get(managerId)
  val tim: List[Radnik] = Db.tim(id)
}

class Radnik (id: Int, ime: String,
  managerId: Int) { // sa lijenom evaluacijom
  lazy val manager: Radnik = Db.get(managerId)
  lazy val tim: List[Radnik] = Db.tim(id)
}
```

Java - kratka povijest

- Java 1.0 se pojavila 1996. i (u pravo vrijeme!) reklamirana je kao jezik za Internet, čime je odmah stekla ogromnu slavu.
- Počeci Jave sežu u 1992., kada se zvala Oak i bila namijenjena za upravljanje uređajima za kabelsku televiziju i slične uređaje.
- **Sami autori su rekli da je Java = C++--**, tj. da je to pojednostavljeni (u pozitivnom smislu) C++. Nije stoga čudno da Java i C++ imaju sličnu sintaksu. Eiffel sintaksa je inspirirana jezikom ADA 83. Scala je negdje između.
- Međutim, Java nije podskup C++ jezika. Također, iako jednostavniji nego C++, Java nije baš jednostavan jezik.
- Eiffel i Scala su "čisti" OOPL jezici. C++ nije, jer je morao zadržati (potpunu) kompatibilnost sa jezikom C.
  Java je negdje između.

Java 8 - najvažnije nove mogućnosti: lambda izrazi, default metode, Streams

- Na temelju onoga što čitamo i čujemo, mogli bismo zaključiti da je najvažnija nova mogućnost u Javi 8 **lambda izraz** (ili kraće, **lambda**).
- Inače, lambda izraz je (u Javi) naziv za metodu bez imena. U pravilu je ta metoda funkcija, a ne procedura. Zato možemo reći i da **lambda izraz je anonimna funkcija**, koja se može javiti kao parametar (ili povratna vrijednost) druge funkcije (koja je, onda, funkcija višeg reda).
- U Javi 8 pojavile su se i tzv. **default metode** u Java sučeljima (interfaces). One, zapravo, predstavljaju uvođenje **višestrukog nasljeđivanja implementacije** u Javu.
- Međutim, lambda izrazi i default metode su, na neki način, posljedica uvođenja treće važne mogućnosti u Javi 8, a to su **Streamsi**, koji nadograđuju dosadašnje Java kolekcije.

Java 8 - lambda izrazi

- Java 8 lambda (izrazi) temelje se na tzv. **funkcijskim sučeljima** (functional interface), koji su postojali od početka.
- Funkcijska sučelja su ona sučelja koja imaju točno jednu (jednu i samo jednu) apstraktnu funkciju. No, od Jave 8 funkcijska sučelja mogu imati i statičke metode i default metode (koje nisu postojale prije Jave 8).
- Npr. kad Java kompajler naiđe na ovakvu naredbu (lambda izraz je desno od znaka jednakosti; ovo je samo jedna od brojnih varijanti pisanja lambda izraza):

```
StringToIntMapper mapper = (String str) -> str.length();
```

kompajler provjerava da li postoji odgovarajuće sučelje StringToIntMapper, koje ima samo jednu apstraktnu funkciju.

Java 8 - default metode

```
// 1. ako bismo postojeće sučelje Iterable
// htjeli proširiti sa novom metodom forEach,
// morali bismo mijenjati svaku klasu
// koja (direktno ili indirektno) nasljeđuje to sučelje
public interface Iterable<T> {
  public Iterator<T> iterator();
  public void forEach(Consumer<? super T> consumer);
}

// 2. default metode rješavaju taj problem
public interface Iterable<T> {
  public r<T> iterator();
  public default void forEach(Consumer<? super T> consumer) {
    for (T t : this) {
      consumer.accept(t);
    }
  }
}
```

PL/SQL – prijedlog za uvođenje nekih funkcijskih mogućnosti

- Mislimo da bi bilo dobro (a vjerojatno nije nemoguće), da Oracle ugradi u PL/SQL neke funkcijske mogućnosti koje danas imaju svi funkcijski jezici i skoro svi objektni jezici (npr. C++, Eiffel, Java, C#, Scala, Kotlin, Swift):

- **funkcije višeg reda**
  (kod kojih argument može biti druga funkcija)

- **lambda izrazi** (anonimne funkcije)

- **eliminacija (ili optimizacija) repnog poziva** (TCE ili TCO); niti Java još nema

- **mogućnost razlikovanja varijabli** koje se mogu mijenjati samo jednom (val) ili više puta (var)

In-place mutation
- direktno mijenjanje (zajedničkih) podataka

- Jim Gray, jedan od najvećih stručnjaka na području baza podataka (posebno transakcija), napisao je sljedeće

  http://www.hpl.hp.com/techreports/tandem/TR-81.3.pdf.
- **Update in place: a poison apple?**
- When bookkeeping was done with clay tablets or paper and ink, accountants developed some clear rules about good accounting practices.
- One of the cardinal rules is **double-entry bookkeeping** so that calculations are self checking, thereby making them fail-fast.
- A second rule is that one **never alters the books**; if an error is made, it is annotated and a new compensating entry is made in the books. The books are thus a complete history of the transactions of the business…

In-place mutation
- direktno mijenjanje (zajedničkih) podataka

- **Update-in-place strikes many systems designers as a cardinal sin**: it violates traditional accounting practices which have been observed for hundreds of years.

- Napomena: Benedikt Kotruljević (Dubrovnik, oko 1400. - 1468.), talijanski Benedetto Cotrugli, latinski Benedictus de Cotrullis, smatra se izumiteljem **dvojnog knjigovodstva**.

- **Kako možemo primijeniti metode, koje su poznate u knjigovodstvu preko 550 godina, na kolekcije podataka**?
- Naime, kada dvije softverske dretve (ili više njih) mijenja kolekciju podataka na način da se radi in-place mutation, tada:

- nećemo koristiti zaključavanje, pa će doći do nekonzistentnosti
- ili ćemo koristiti zaključavanje, i smanjiti paralelnost rada.

Izbjegavanje in-place mutation

- Rješenje (ne uvijek) je u tome da se izbjegne in-place mutation, tako da se, kao kod dvostrukog knjigovodstva, **ne mijenjaju postojeći podaci, nego se uvijek stvaraju novi.**

- **Umjesto da mijenjamo postojeći element, stvaramo novi. Umjesto da dodajemo element u postojeću kolekciju, stvaramo novu kolekciju s dodanim elementom.**

- Programeri koji su navikli na imperativno programiranje isprva ostanu začuđeni kada čuju za takav način rada.

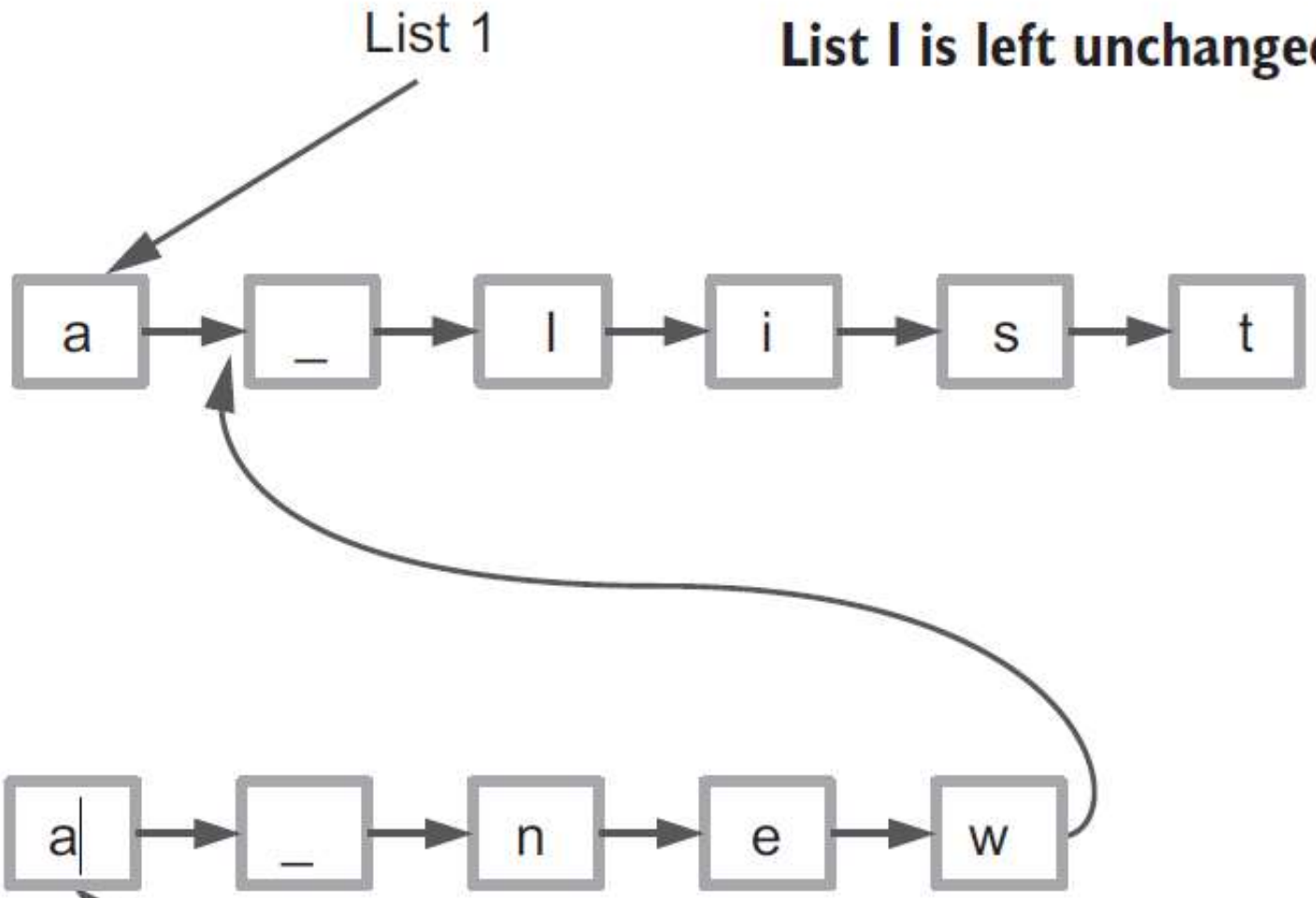
Uobičajene (dvije) primjedbe na izbjegavanje in-place mutation

- Uobičajene (dvije) primjedbe su:
  - ako jedna dretva stvara svoju kopiju podataka, i ne vidi tuđe izmjene, zar to nije loše?
  - zar ovakav način rada ne traži jako velike memorijske resurse, jer svaka dretva radi kopiju svojih podataka?

- Odgovor na prvo pitanje je – **pa dretve i ne bi trebale vidjeti privremene podatke**, koji se ne bi vidjeli u serijskom načinu rada (napomena: kod baza podataka uobičajeno je da transakcija ne vidi rezultate druge transakcije, dok ta druga ne napravi COMMIT).

- Odgovor na drugo pitanje – **najčešće ne treba raditi kopiju svih podataka, nego samo dijela.**

Primjer brisanja 1. elementa i dodavanja 5 novih elemenata u imutabilnu listu



List l is left unchanged.

List 2 is a new list after removing one element and adding five new ones. No copying has occurred.

Da li se s podacima u BP može raditi nešto slično kao sa imutabilnim listama kod programskih jezika?

- Iako to nije u potpunosti isto, postoji nešto što ima puno sličnosti, a to je – **ne koristiti (SQL) UPDATE i DELETE naredbe, nego samo naredbu INSERT**.
- Umjesto da mijenjamo postojeći redak (UPDATE), unesemo novi (INSERT), koji će "zamijeniti" prethodni redak. Naravno, mora postojati stupac, ili više njih, koji će označavati da je to najnoviji redak.
- Umjesto da brišemo postojeći redak (DELETE), unesemo novi, koji će sadržavati informaciju da je taj redak (i njegove prethodne verzije) izbrisan. Na fizičkoj razini ovo nije ništa novo, jer DELETE naredba na fizičkoj razini ne briše redak, već ga označava slobodnim za upis novog retka.

Koje su prednosti i mane takvog pristupa

- Prednost bi bila (i) u tome što bi **podaci bili rjeđe zaključani** (INSERT isto radi zaključavanje, zbog provjere PK / UK).
- Naravno, može se izreći i puno primjedbi, npr.:

- tako bi nam trebalo **puno više prostora** za bazu podataka, jer bi se sve pamtilo; no današnji su diskovi sve veći, a danas neke regulative traže da se pamti sve
- danas skoro sve baze (Oracle od početka) rade tako da **DML naredbe ne zaključavaju podatke za čitanje**, tj. SELECT (Oracle multiversion consistency model rada s transakcijama)
- ako želimo pamtiti sve podatke, **u Oracle bazi možemo koristiti flashback tehnologije** (Flashback Data Archive, Flashback Query ...)
- **tako bi nam bilo jako teško programirati**; no možda se samo treba naviknuti – i funkcijsko programiranje je vrlo neobično onima koji su radili samo imperativno programiranje.

Oracle DBMS 19c (19.10 i 19.11) ima blockchain tablice i imutabilne tablice

- Oracle je od baze 18c (izašla je 2018.) uveo označavanje prema godinama i uveo je pojmove Long Term Release (LTR) i Innovation Release ili non-LTR (napomena: Java 19 ne predstavlja godinu i LTS u Javi znači Long-Term Support).

- Sve baze do 12c, baza 19c i buduća baza 23c su LTR. Baze 18c i 21c su non-LTR.

- Oracle je početkom 2021. uveo **blockchain tables** u bazi 21c, a onda ih je uključio i u verziju 19c, u 19.10 (napomena: 19 je godina, a 10 nije mjesec, nego redni broj).

- Nekoliko mjeseci kasnije, Oracle je u bazu 21c uveo **immutable tables**, a onda ih uključio i u verziju 19c, u 19.11.

Oracle DBMS 19.10 ima blockchain tablice

- Blockchain tablice štite podatke koji bilježe važne radnje, imovinu, entitete i dokumente od neovlaštene izmjene ili brisanja. Blockchain tablice sprečavaju neovlaštene promjene napravljene pomoću baze podataka i otkrivaju neovlaštene promjene koje zaobilaze bazu podataka.
- Blockchain tablice su samo za čitanje (insert-only tablice), i retke organiziraju u više lanaca. Svaki redak u lancu, osim prvog retka, lančano je povezan s prethodnim retkom u lancu pomoću kriptografskog hash-a.
-

```
CREATE BLOCKCHAIN TABLE bank_ledger
  (bank VARCHAR2(128), deposit_date DATE,
   deposit_amount NUMBER)
 NO DROP UNTIL 1000 DAYS IDLE --NO DROP bezuvjetno
 NO DELETE UNTIL 366 DAYS AFTER INSERT --NO DELETE
 HASHING USING "SHA2_512" VERSION "v1";
```

Oracle DBMS 19.11 ima imutabilne tablice

- Imutabilne (nepromjenjive) tablice pružaju zaštitu od neovlaštene izmjene podataka. Imutabilne tablice su tablice samo za čitanje (insert-only) i sprečavaju neovlaštene izmjene podataka od strane insajdera i slučajne izmjene podataka koje su rezultat ljudskih pogrešaka.
- Novi redci mogu se dodati u imutabilnu tablicu, ali se postojeći redci ne mogu mijenjati. Redci postaju zastarjeli nakon zadanog razdoblja zadržavanja i tada se mogu brisati. **Korištenje imutabilnih tablica ne zahtijeva izmjene postojećih aplikacija.**
-

```
CREATE IMMUTABLE TABLE trade_ledger
  (id NUMBER, luser VARCHAR2(40), value NUMBER)
 NO DROP UNTIL 1000 DAYS IDLE
 NO DELETE UNTIL 366 DAYS AFTER INSERT;
```

Imutabilne i blockchain tablice - razlika

Table 20-8 Differences Between Immutable Tables and Blockchain Tables

Immutable Tables	Blockchain Tables
Immutable tables prevent unauthorized changes by rogue or compromised insiders who have access to user credentials.	In addition to preventing unauthorized changes by rogue or compromised insiders, blockchain tables provide the following functionality: • detects unauthorized changes made by bypassing Oracle Database software • detects end user impersonation and insertion of data in a user's name but without their authorization • prevents data tampering and ensures that data was actually inserted in to the table
Rows are not chained together.	Each row, except the first row, is chained to the previous row by using a cryptographic hash. The hash value of a row is computed based on the row data and the hash value of the previous row in the chain. Any modification to a row breaks the chain, thereby indicating that the row was tampered.
Inserting rows does not require additional processing at commit time.	Additional processing time is required, at commit time, to chain rows.

Imutabilne i blockchain tablice - razlika

- Dakle, iako su nastale malo kasnije, Oracle imutabilne tablice su manjih mogućnosti nego blockchain tablice.

- Između ostalog, za razliku od imutabilnih tablica, blockchain tablice otkrivaju neovlaštene promjene koje zaobilaze bazu podataka.

- Međutim, imutabilne tablice imaju i prednosti.

- Jednostavnije su, imaju manje restrikcija za korištenje i ne traže dodatno procesorsko vrijeme za insert podataka, dok je kod blockchain tablica potrebno dodatno vrijeme za ulančavanje blokova (za vrijeme commit procesiranja).

Oracle DBMS 23c – što je novo u blockchain / immutable tables

- Ovi slajdovi napravljeni su na temelju materijala koje je napravio Tim Hall:

https://oracle-base.com/articles/23c/blockchain-table-enhancements-23c

i

https://oracle-base.com/articles/23c/immutable-table-enhancements-23c

Oracle DBMS 23c – blockchain tables

- In Oracle 23c we have the option of using the original "V1" version, which is the default, or the new "V2" version, which supports additional functionality.

- V1 version (default):

```
create blockchain table bct_t1 (
  id              number,
  fruit           varchar2(20),
  quantity        number,
  created_date    date,
  constraint it_t1_pk primary key (id)
)
no drop until 0 days idle
no delete until 16 days after insert
hashing using "SHA2_512";
```

Oracle DBMS 23c – blockchain tables

- V2 version:

```
create blockchain table bct_t2 (
  id              number,
  fruit           varchar2(20),
  quantity        number,
  created_date    date,
  constraint it_t2_pk primary key (id)
)
no drop until 0 days idle
no delete until 16 days after insert
hashing using "SHA2_512" version "v2";
```

- Checking the USER_TAB_COLS view shows us several invisible columns have been added to our column list. Notice the "V2" table has twice the number of hidden columns compared to the "V1" table.

Oracle DBMS 23c – blockchain tables

- In the previous releases blockchain tables only supported up to 32 system generated chains per instance, with rows being assigned to chains at random.
- In Oracle 23c we can create user chains using up to three columns to define the chains. Each unique combination of values represents a separate chain.
- In the following example we use the WITH USER CHAIN clause, resulting in a separate chain for each unique value of the FRUIT column.

Oracle DBMS 23c – blockchain tables


```
create blockchain table bct_uc (
   id              number,
   fruit           varchar2(20),
   quantity        number,
   created_date    date,
   constraint bct_uc_pk primary key (id)
)
no drop until 0 days idle
no delete until 16 days after insert
hashing using "SHA2_512"
with user chain fruit_chain (fruit) version "v2";
```

- The USER_BLOCKCHAIN_TABLE_CHAINS view allows us to display information about user chains associated with a table.

Oracle DBMS 23c – blockchain tables

- In Oracle 23c the maximum idle retention time for the table is controlled by the BLOCKCHAIN_TABLE_RETENTION_THRESHOLD parameter.

```
show parameter blockchain_table_retention_threshold
NAME                                 TYPE         VALUE
------------------------------------ ------------ -----
blockchain_table_retention_threshold integer      16
```

This means we can't set the NO DROP clause to a value longer than 16 days.

- In Oracle 23c we can create flashback data archives (FDA) as block blockchain tables by adding the BLOCKCHAIN keyword to the FLASHBACK ARCHIVE clause. This gives additional assurance the flashback data archive has not been tampered with.
- Many of the issues associated with blockchain tables in the previous releases have been resolved in Oracle 23c.

Oracle DBMS 23c – immutable tables

- In Oracle 23c we have the option of using the original "V1" version, which is the default, or the new "V2" version, which supports additional functionality.

```
create immutable table it_t2 (
   id              number,
   fruit           varchar2(20),
   quantity        number,
   created_date    date,
   constraint it_t2_pk primary key (id)
)
no drop until 0 days idle
no delete until 16 days after insert version "v2";
```

- The hidden columns are the same as those of a blockchain table, but unlike blockchain tables, only the ORABCTAB_CREATION_TIME$ and ORABCTAB_USER_NUMBER$ columns are populated.

Zaključak

- Zadnjih nekoliko godina, funkcijsko programiranje stječe veliku popularnost u odnosu na imperativno programiranje.
- Funkcijsko programiranje, kao i logičko programiranje, po nečemu je vrlo slično programiranju u SQL-u - visoko je deklarativno.
- U prezentaciji smo vrlo kratko prikazali neke osobine čistog funkcijskog jezika Haskell i nekih funkcijskih proširenja u objektnim jezicima Scala i Java.
- Prikazali smo neke funkcijske mogućnosti koje bi Oracle (možda) mogao ugraditi u programski jezik PL/SQL.
- Najvažnije (ali i najteže) je pitanje: možemo li u praksi raditi s bazama podataka bez korištenja naredbi UPDATE i DELETE (samo INSERT)!?
- Oracle DBMS 19c (od verzija 19.10 i 19.11, 2021. godine) ima blockchain tablice (19.10) i imutabilne tablice (19.11).

Literatura (dio)

- Gopalakrishnan, G. (2006): Computation Engineering - Applied Automata Theory and Logic, Springer
- Hutton, G. (2016): Programming in Haskell (2. izdanje), Cambridge University Press
- Meyer, B. (2009): Touch of Class - Learning to Program Well with Objects and Contracts, Springer
- Odersky, M., Spoon, L., Venners, B. (2016): Programming in Scala (3. izdanje), Artima Press
- Saumont, P-Y. (2017): Functional Programming in Java, Manning
- Saumont, P-Y. (2019): The Joy of Kotlin, Manning
- Oracle (2022): Oracle Database Administrator's Guide 19c (priručnik E96348-16, poglavlja 20.17 i 20.18)