TRANSAKCIJE I ORACLE - BAZA, FORMS, ADF

Zlatko Sirotić, univ.spec.inf.
Istra informatički inženjering d.o.o.
Pula

Malo marketinga ☺

iii *maginarna jednadžba*

Istra informatički inženjering

$$iii = i^{i^{i}}$$

Uvod

- Ispravno rukovanje transakcijama u bazama podataka vrlo je bitno za poslovne aplikacije.

- Razumijevanje transakcija postaje sve važnije, jer su one "ušle" u programske jezike, pa i u hardver, u vidu softverskih, hardverskih (npr. Haswell) i hibridnih transakcijskih memorija.

- U radu su prikazane različite teme vezane za upravljanje transakcijama, kroz tri poglavlja:
  - Oracle baza
  - Oracle Forms
  - Oracle ADF

1. Baza

❖ "Standardne" teme:

Osnove arhitekture Oracle DBMS-a (vezano za transakcije)

Transakcije općenito

Transakcije i zaključavanje

Nove mogućnosti u bazi 12c (vezano za transakcije)

❖ Nešto teže teme su:

Distribuirane transakcije

Rješavanje mutiranja okidača baze

1. Baza

❖ Specijalne teme (naša rješenja):

Simulacija COMMIT okidača
pomoću odgođenih deklarativnih integritetnih ograničenja

Simulacija INSERT WAIT naredbe

Simulacija ROLLBACK TO SAVEPOINT naredbe
u okidaču baze

Kako generirati dokumente bez rupa u brojevima

1. Baza - Transaction Guard

- Vezano za transakcije, najvažnije nove mogućnosti u bazi 12c jesu Transaction Guard i Application Continuity (koja se temelji na Transaction Guard mogućnosti).
- Treba naglasiti da obje mogućnosti ima samo Enterprise edicija baze, dok ih Standard edicija nema. Osim toga, Application Continuity traži i Active Data Guard opciju ili Real Application Clusters opciju EE baze.
- Transaction Guard rješava jedan veliki mogući problem u transakcijama. Do sada, kada je klijent baze (to može biti i aplikacijski server, pa čak i pohranjena PL/SQL procedura) poslao bazi COMMIT naredbu, i ako je baš tada došlo do pada veze, klijent je dobio informaciju da je veza pala, ali ne i informaciju da li je COMMIT uspješno napravljen, ili nije.

1. Baza - Transaction Guard

- Klijent nije mogao niti naknadno dobiti tu informaciju (osim, u nekim slučajevima, pomoću relativno složenih aplikacijskih rješenja).
- Zbog toga se moglo desiti da klijent (softver ili korisnik) pokrene dva puta istu transakciju, jer ne zna da je prethodna uspješno završila, ili ne pokrene niti jedanput.
- Transaction Guard zasniva se na tome što se u trenutku COMMIT-a pamti tzv. logical transaction identifier (LTXID), koji se kasnije može koristiti za biranje odgovarajućeg postupka u aplikaciji, kako bi se osiguralo da se neka transakcija neće izvršiti dva puta, a eventualno hoće jedanput.

2. Forms

❖ "Standardne" teme:

Forms - razvoj, varijante, arhitektura

Neka svojstva Forms modula i Forms bloka

Svojstva tipičnih vrsta tekstualnih polja (text item)

Transakcijski i validacijski status Forms objekata

Master-detalj relacije

2. Forms

- Nešto teže teme:

COMMIT i POST Forms procesi

Različiti načini poziva Forms modula

- Specijalne teme (naša rješenja)

Template i library za POST-iranje kod relacije master–detalj

Forms i odgođena deklarativna integritetna ograničenja na bazi

2. Forms - COMMIT i POST procesi

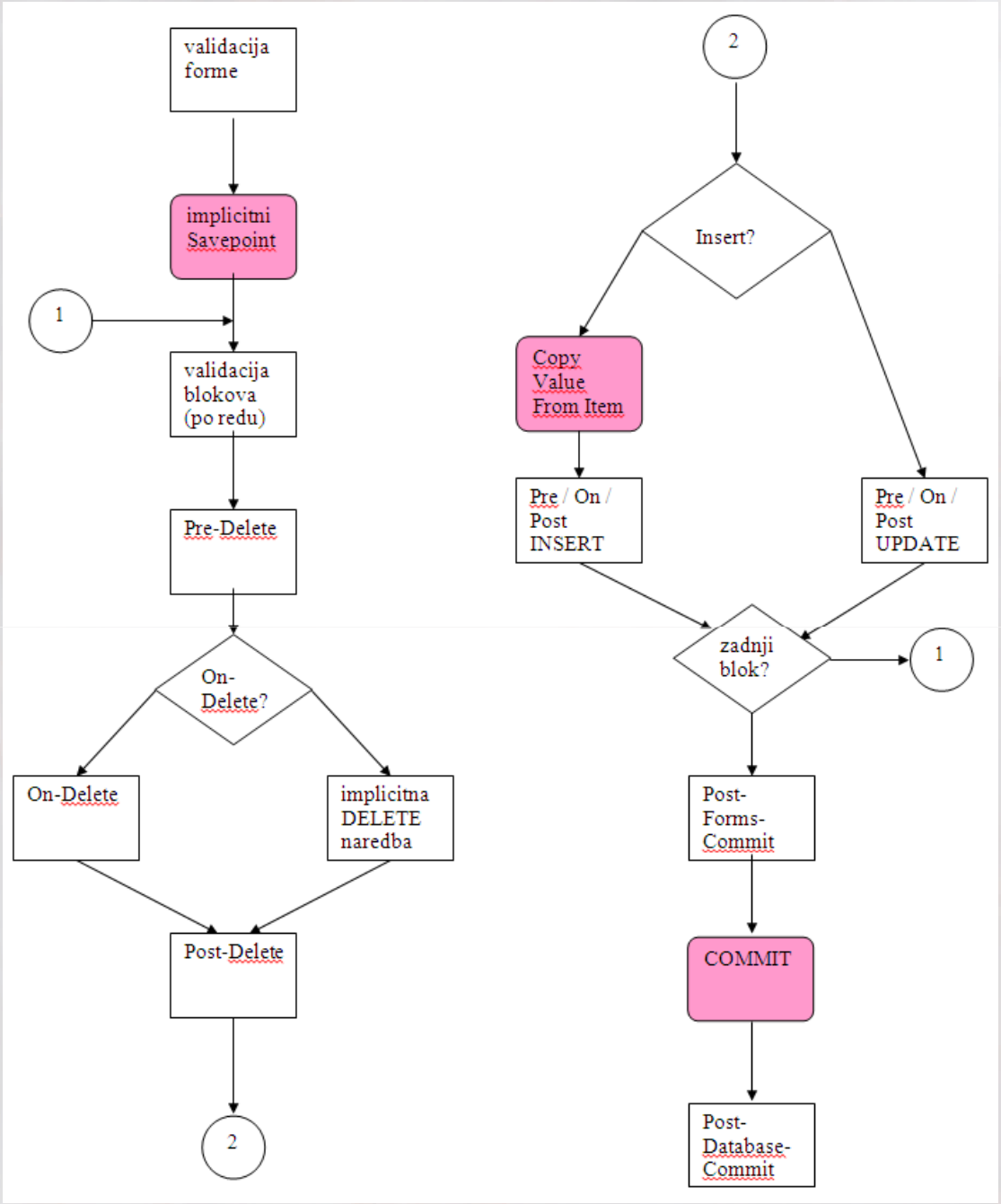
- Među najvažnijim, ali i najsloženijim Forms procesima, jesu COMMIT i POST procesi.
- COMMIT proces može se jednostavno (i točno) prikazati kao:

  COMMIT proces =

  POST proces +

  COMMIT naredba na bazi +

  Forms okidač Post-Database-Commit
- Suština je u tome da se ažuriranje redaka (unos / izmjena / brisanje) prvo radi samo unutar Forms bloka. Kad korisnik odabere gumb Save ili funkcijsku tipku F10, Forms šalje odgovarajuće DML naredbe na bazu, te daje COMMIT na bazu (ako je riječ o COMMIT procesu).


validacija forme
implicitni Savepoint
1
validacija blokova (po redu)
Pre-Delete
On-Delete?
On-Delete
implicitna DELETE naredba
Post-Delete
2
2
Insert?
Copy Value From Item
Pre / On / Post INSERT
Pre / On / Post UPDATE
zadnji blok?
1
Post-Forms-Commit
COMMIT
Post-Database-Commit

2. Forms - master-detalj relacija

- Za povezivanje dva bloka, mastera i detalja, koristi se tzv. Forms relacija (relation), koja pripada master bloku.
- Na temelju definiranih svojstava Forms relacije, Forms Builder automatski generira sljedeće objekte u Forms modulu:

- okidač na razini Forms modula ON-CLEAR-DETAILS;

kako mu ime kaže, on briše iz bloka detalje prethodnog master retka, prije nego ih napuni detaljima novog master retka;

- okidače na razini (master) bloka
  ON-POPULATE-DETAILS
  ON-CHECK-DELETE-MASTER;

- PL/SQL procedure:
  Check_Package_Failure, Clear_All_Master_Details, Query_Master_Details.

2. Forms - master-detalj relacija

- Važno je napomenuti da se prije Pre-Insert okidača automatski izvršava Copy Value From Item potproces. To je važno kod master–detalj blokova gdje master ima PK ili UK stupac (npr. imena ID), koji se puni na bazi, a za njega je vezan vanjski ključ (FK) tablice detalja.
- Zbivanja onda idu ovako: na bazu se uvijek prvo šalje redak master bloka, pri čemu baza generira vrijednost za PK. Ta se vrijednost pomoću Post-Insert okidača na master bloku (ili na temelju postavljenog svojstva DML Returning Value = Yes) puni u master blok.
- Kada (poslije) dođe vrijeme za INSERT detalja, potproces Copy Value From Item puni PK iz master bloka u odgovarajuće polje (vanjskog ključa) bloka detalja.

2. Forms - problem master–detalj–detalj od detalja

- Treba naglasiti jednu veliku manu Forms relacija: nije moguće na standardan način u jednoj transakciji (baze) ostvariti relaciju "master–detalj–detalj od detalja", ili u jednoj transakciji ažurirati više zaglavlja i njihove detalje.
- Razlog je taj što Forms blok detalja može sadržavati retke-detalje samo jednog master retka.
- Za razliku od Formsa, ADF to može bez problema.
- No i Forms to može, ali na neki nestandardan način. Jedan od tih načina (vjerojatno "najčišći") je pomoću POST naredbe.
- Rješenje nije jednostavno, ali suština je jednostavna: kod odabira novog master retka, prethodni (mijenjani) master redak i svi njegovi redovi-detalji POST-iraju se na bazu (dakle, izvršavaju se DML naredbe, ali bez COMMIT-a).

2. Forms - problem master–detalj–detalj od detalja

- Mi smo to rješenje izveli kao Designer template i pripadajući PL/SQL library. Suština rada template-a je sljedeća:

- u WHEN-NEW-RECORD-INSTANCE provjerava se status forme; ako je mijenjana, radi se POST;
- u WHEN-VALIDATE-RECORD mijenja se oznaka valjanosti bloka na formi; oznaka je N(ovi), V(aljan) ili P(romijenjen);
- u ON-CLEAR-DETAILS provjerava se oznaka valjanosti (provjera se ne radi ako je riječ o brisanju mastera) i, ako je N(ovi) ili P(romijenjen), ne dozvoljava prijelaz na novi master;
- gumb "Spremi" postavlja oznaku valjanosti na V(aljan) (ako već nije bila) i radi COMMIT_FORM;
- gumb "Poništi" radi CLEAR_FORM.

3. ADF

❖ "Standardne" teme:

ADF - razvoj i arhitektura

Entity Object

View Object

Application Module

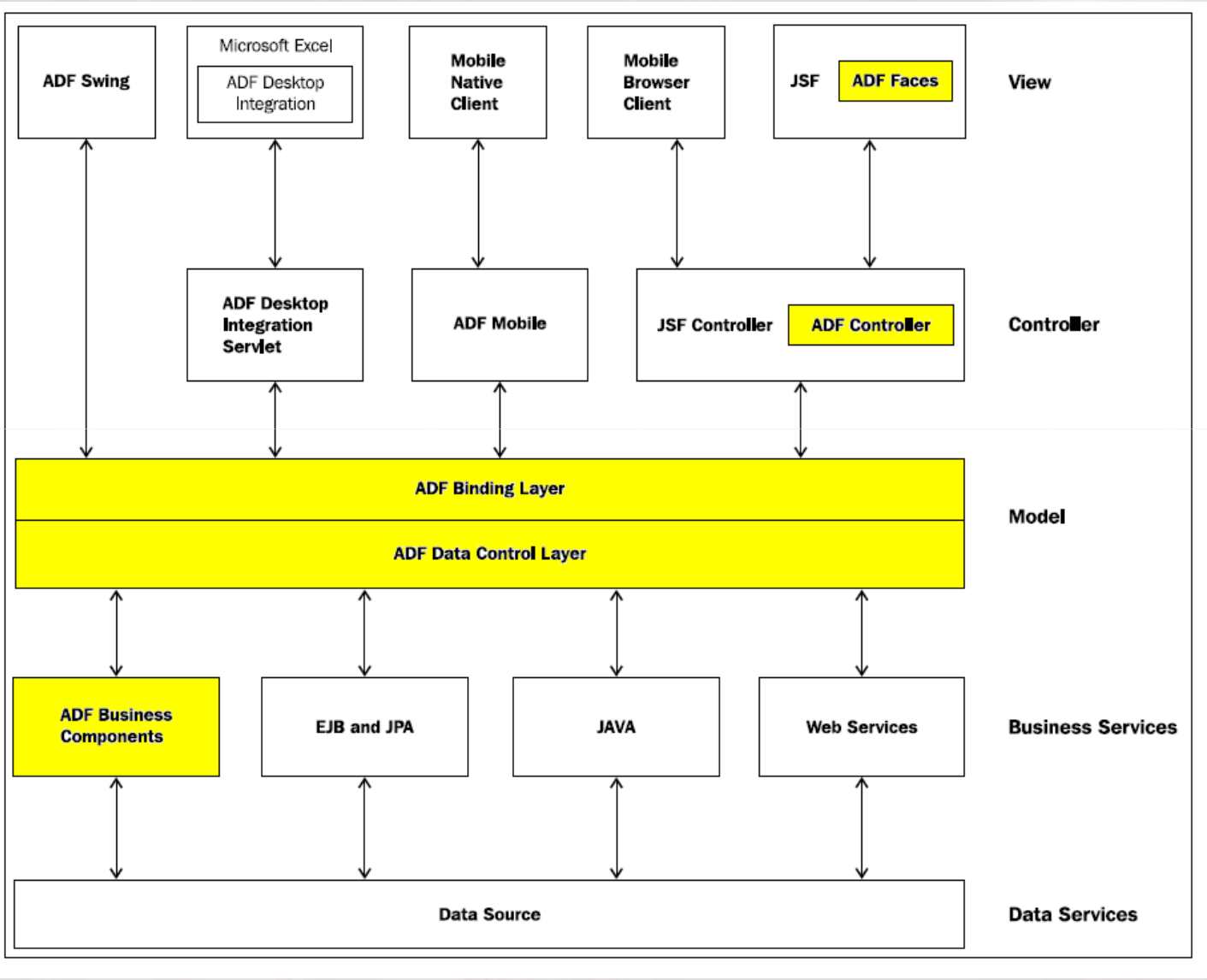
Kako pomiriti HTTP stateless protokol i statefull zahtjeve

❖ Nešto teže teme su:

Application Module Pooling

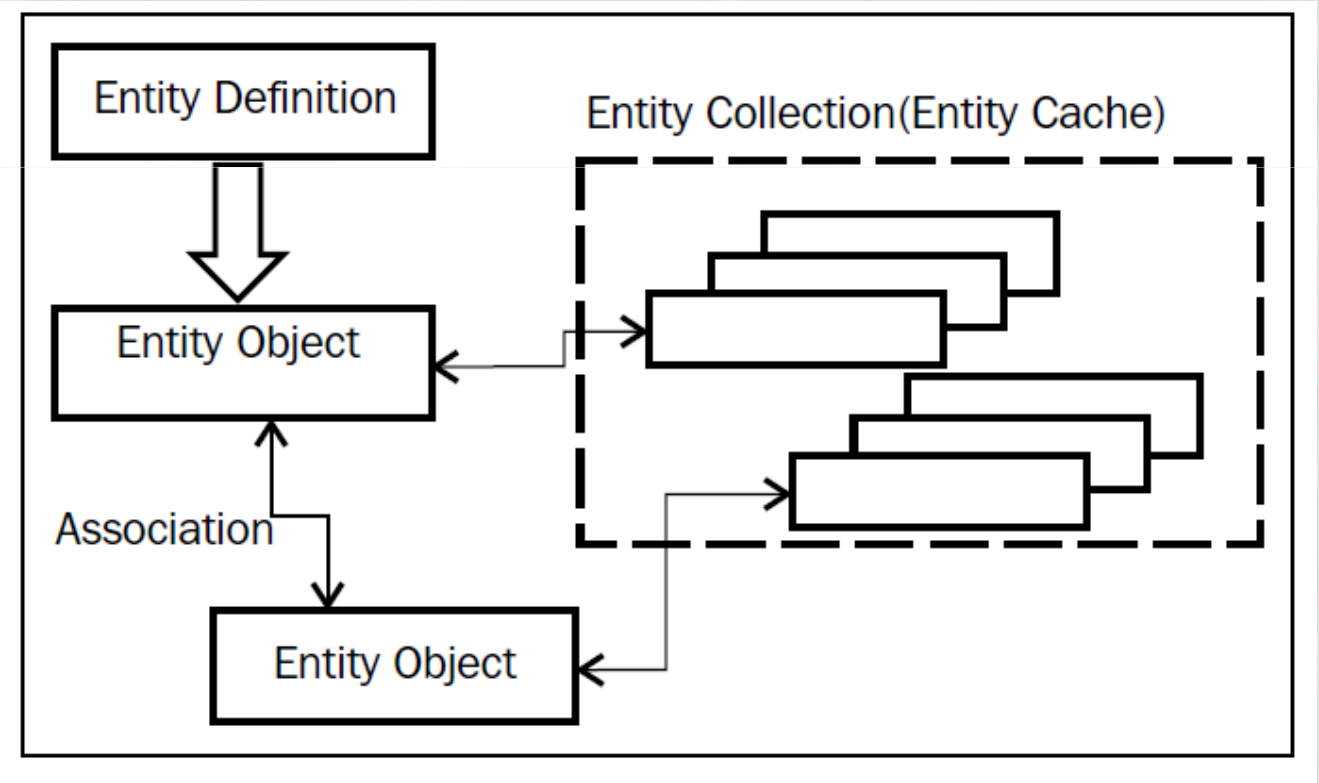
Task Flow i transakcije

3. ADF - arhitektura



3. ADF – Entity Object

- Dvije EO definicije (koje predstavljaju npr. dvije tablice na bazi) mogu biti povezane asocijacijom, koja je najčešće nastala iz FK veze među tablicama na bazi.



3. ADF – EO zaključavanje redaka

- Postavka za određeni AM kroz jbo.locking.mode ili za cijelu aplikaciju u adf-config.xml Locking Mode.
- Moguće vrijednosti:

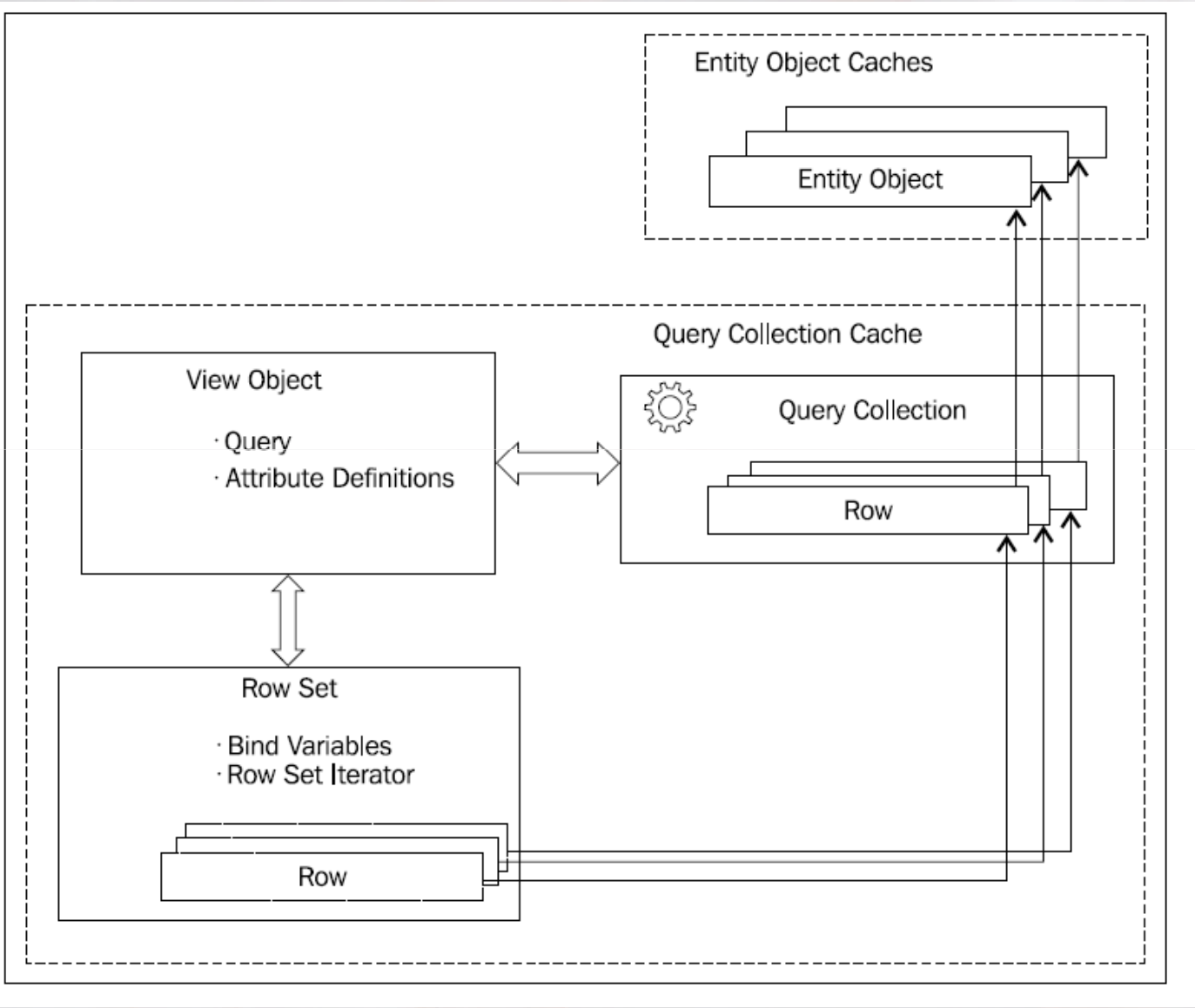
- None: ništa se ne radi;
- Pessimistic: ne preporuča se za web aplikacije, jer čim netko pokuša promijeniti bilo koji podatak retka, redak ostaje zaključan do kraja transakcije;
- Optimistic: optimističko zaključavanje (default); zaključava redak tek na kraju, pa tada provjerava da li su stare vrijednosti polja iz retka jednake onima koje su sada na bazi; može se ubrzati pomoću svojstva Change Indicator;
- Optupdate: slično kao optimističko zaključavanje, ali bez zaključavanja, pa ne pruža istu sigurnost.

3. ADF – kompozitna asocijacija između EO

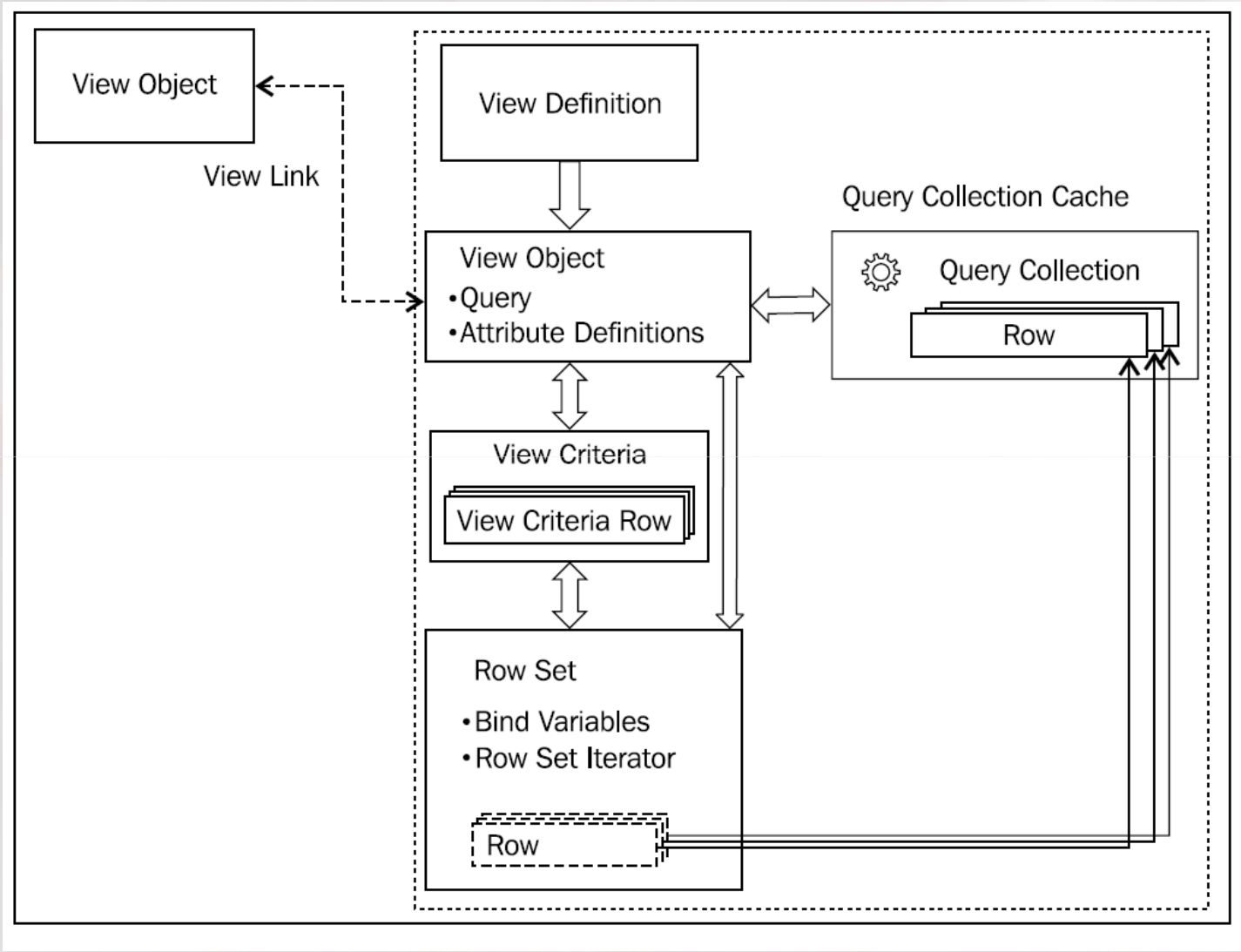
- Kada se u ADF-u označi da je asocijacija između dva EO kompozitna, time se rješava i pravilan redoslijed slanja redaka na bazu (prvo roditelj, pa djeca), što je važno (i) kada se za PK / UK atribut roditelja (npr. ID) postavi tip DBSequence. Kompozitnoj asocijaciji se mogu postaviti različita svojstva, a jedno je Lock Level.
- Lock Level određuje se da li će se zaključati redak roditelja, ako se zaključa bilo koje njegovo dijete; moguće varijante su:

- None: roditelj se neće zaključati;
- Lock Container: zaključat će se prvi (neposredni) roditelj;
- Lock Top-level Container: zaključat će se vršni roditelj, ili onaj u hijerarhiji koji ima isključeno Lock Top-level Container svojstvo.

3. ADF – View Object



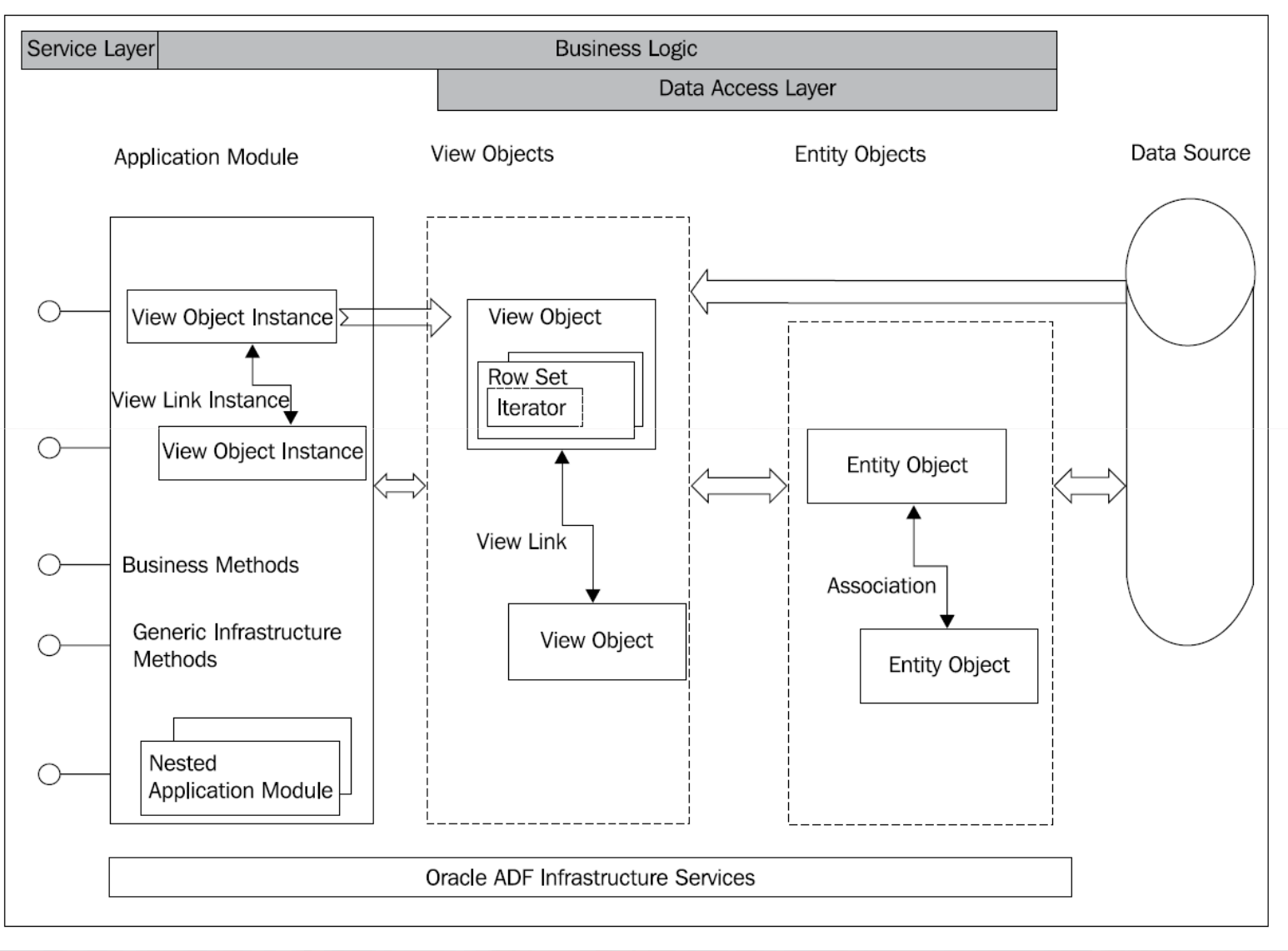
3. ADF – View Link



3. ADF – upit nad VO

- ❖ Kada se izvodi upit (query) nad VO, može se odrediti koji će se izvor podataka koristiti:
- Scan database tables: čita se baza, što je podrazumijevano (default) ponašanje; postavlja se programski sa vo.ViewObject.QUERY_MODE_SCAN_DATABASE_TABLES
- Scan view rows: čita se query collection (kolekcija prvo mora biti napunjena upitom na bazu); postavlja se programski sa vo.ViewObject.QUERY_MODE_SCAN_VIEW_ROWS
- Scan entity cache: čita se entity cache (moguće je samo za VO temeljene nad EO); postavlja se programski sa vo.ViewObject.QUERY_MODE_SCAN_ENTITY_ROWS.
- ❖ Također, VO omogućavaju sortiranje i filtriranje redaka u memoriji.

3. ADF- Application Module



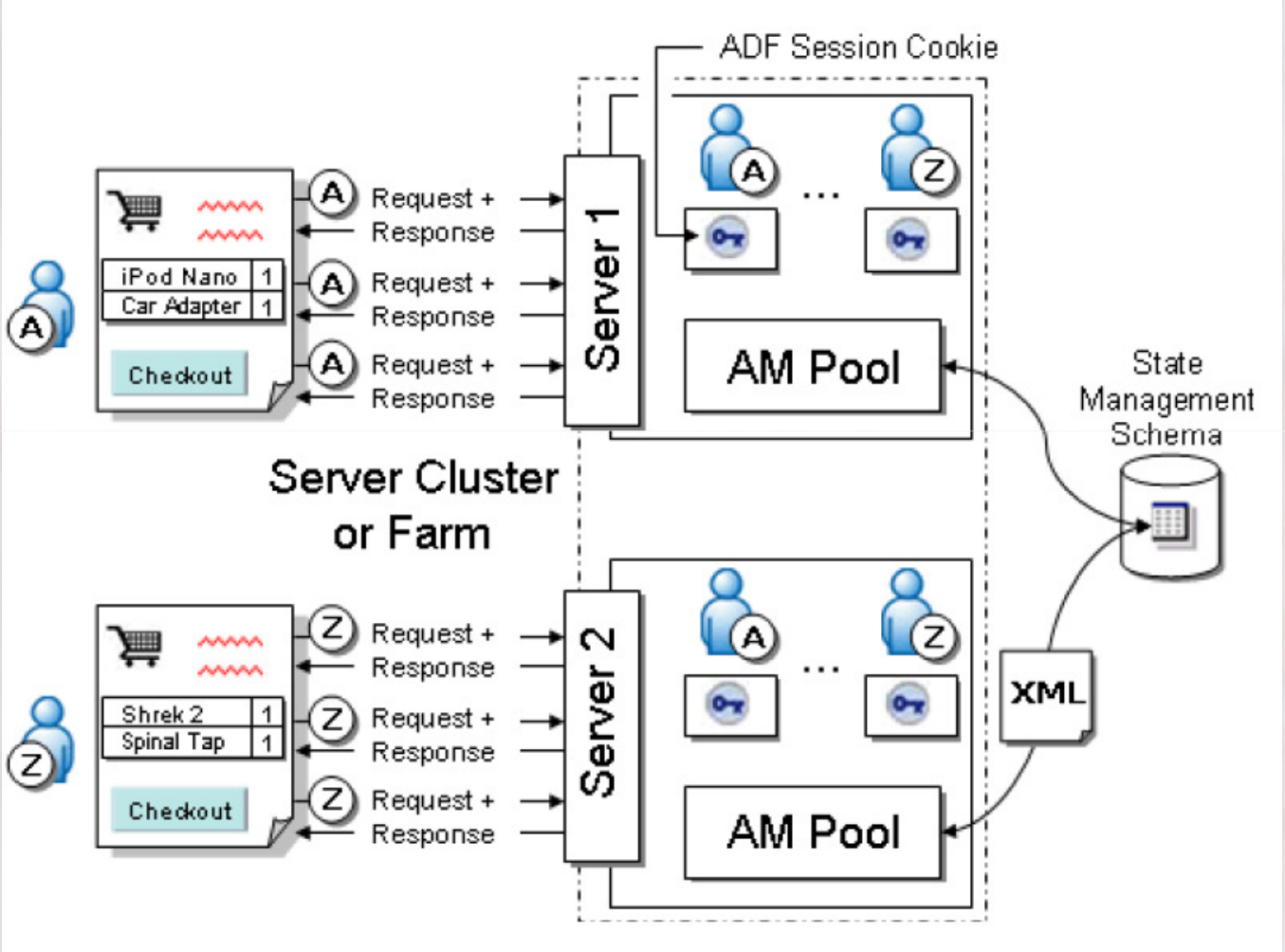
3. ADF – dijeljeni (shared) aplikacijski moduli

- Često je potrebno dijeliti podatke između više korisničkih sesija.
- Korisnička sesija nije isto što i sesija baze, jer se jedna korisnička sesija može realizirati kroz više sesija baze, što je standardno u web aplikacijama.
- Za potrebe dijeljenja podataka, ADF ima tzv. dijeljene (ili zajedničke - shared) AM instance.
- AM instanca može biti dijeljena na razini aplikacije (application level shared application module), pa tada svi korisnici koriste istu AM instancu i vide iste podatke, ili samo na razini korisničke sesije (session level shared AM), gdje AM koje se nalaze u istoj aplikacijskoj sesiji, ali unutar drugog vršnog AM, ne vide podatke iz tako dijeljene AM.

3. ADF – Auto Refresh u VO

- VO ima i svojstvo Auto Refresh. Iako nije usko vezano za dijeljene (shared) AM, ipak se najčešće koristi uz njih.
- To se svojstvo može koristiti na Oracle bazi 10.2 ili većoj, koje podržavaju tzv. Continuous Query Notification (CQN).
- Suština je u tome da baza gura (push) informacije o promjeni redaka prema klijentu (naravno, klijent može biti i aplikacijski server), a klijent na temelju dobivenih informacija izvršava upit i osvježava svoj cache.
- ADF kroz JDBC API prije izvršenja VO upita šalje registraciju tog upita bazi.
- U ADF-u se najčešće tako osvježavaju VO koji se nalazi u AM dijeljenom (shared) na razini aplikacije.

3. ADF – Application Module Pooling



3. ADF – Application Module Pooling

- Kod vraćanja AM instance u AM pool, postoje tri razine otpuštanja (release levels):

- Managed (default): AM pool preferira zadržati istu AM instancu za istog korisnika, ako je to moguće; ako nije moguće, radi se pasivacija AM instance (podaci se smještaju u bazu, rjeđe u datoteku), a kasnije se radi aktivacija (druge) AM instance;
- Unmanaged: nikakvo stanje se ne pamti;
- Reserved: veza 1 : 1 između AM instance i korisnika; podsjeća Forms način rada; nije preporučljiva za web aplikacije, iako je najjednostavnija!

3. ADF – Application Module Pooling

- Kao dio testiranja aplikacije, vrlo je preporučljiva praksa da se AM testiraju sa konfiguracijskim parametrom jbo.ampool.doampooling postavljenim na false.
- Takva postavka zapravo forsira pasivaciju i aktivaciju kod svakog zahtjeva, čime se mogu naći greške koje bi se javile tek u malom broju slučajeva normalnog rada.
- Npr. ako u jednom HTTP request-response paru postavimo varijablu u paketu na neku vrijednost, u drugom HTTP request-response paru (iste aplikacijske sesije) očekivali bismo da se pročita ista vrijednost; no, kod pasivacije to više neće biti tako, jer drugi HTTP par može dobiti drugu AM instancu, što znači i drugu sesiju na bazi.

Zaključak

- Nažalost, greške u rukovanju transakcijama obično se kod testiranja teže uoče, jer često ovise o spletu događaja, broju korisnika koji istovremeno rade i sl.
- Greške u transakcijama mogu uzrokovati npr.:

- pogrešne podatke; primjer: derivirani iznos u zaglavlju dokumenta nije izračunat u istoj transakciji u kojoj su mijenjani podaci iz stavaka dokumenta, na temelju kojih se on računa;
- gubljenje dokumenata; primjer: račun je štampan, COMMIT slijedi iza štampanja, ali transakcija se prekinula i u bazi nema dokumenta, a štampan je;
- pojavu "rupa" u rednim brojevima dokumenata; primjer: za punjenje brojeva dokumenata koristi se sekvenca sa baze, što ne garantira da neće biti rupa ...

Zaključak

- Kako naglašava Tom Kyte, često aplikacijski programeri gledaju na DBMS sustav kao na "crnu kutiju". Nemaju vremena za dublje upoznavanje s mogućnostima konkretnog DBMS sustava, drže da su svi DBMS sustavi isti (ili vrlo slični), žele pisati generički kod (neovisan o DBMS-u) itd.
- Aplikacijski programeri ponekad ne nalaze vremena za upoznavanje problematike rukovanja transakcijama niti unutar alata s kojima rade, npr. Forms i ADF alata, već se pouzdaju u podrazumijevano (default) ponašanje, ponekad i ne znajući kakvo je to ponašanje.
- Istina, često se transakcije i u bazi i u alatima mogu koristiti na "standardan" način, i često je tako i najbolje. No ponekad treba posegnuti za nekim rješenjem koje nije "standardno".