IMA NEKA LOŠA VEZA
(PRIČA O IN-DOUBT DISTRIBUIRANIM TRANSAKCIJAMA)

Zlatko Sirotić
Istra informatički inženjering d.o.o., Pula
e-mail: zlatko.sirotic@iii.hr

SAŽETAK

Najlakše i najbolje je raditi sa centraliziranim bazama podataka. Međutim, ponekad u praksi moramo raditi sa distribuiranim bazama podataka. Tada često koristimo replikaciju, sinkronu ili asinkronu. Sinkrona replikacija predstavlja jednu vrstu distribuirane transakcije, no, distribuirana transakcija nam u distribuiranoj bazi podataka često treba i izvan replikacije. Distribuiranu transakciju podržava dvofazni protokol potvrđivanja transakcije, ili (kraće) dvofazni commit protokol (two phase commit protocol). Taj protokol omogućava da se distribuirana transakcija konzistentno izvrši i kod povremenih padova komunikacijskih veza (ili/i servera koji sudjeluju u distribuiranoj transakciji). Nažalost, ponekad su veze (ili serveri) tako loše da se distribuirana transakcija nađe u nedoumici (in-doubt) i tada moramo ručno intervenirati. U radu se prikazuje kako dolazi do in-doubt transakcije i kako ju riješiti.

The easiest and the best is to work with centralized databases. However, in practice, sometimes we have to work with distributed databases. Then we often use replication, synchronous or asynchronous. A synchronous replication is a kind of distributed transaction; however, we often need a distributed transaction in a distributed database, even outside the replication. A distributed transaction is supported by a two phase commit protocol. This protocol enables a distributed transaction to be consistently performed, even with occasional failure of communication links (and / or servers that take part in that distributed transaction). Unfortunately, sometimes the connections (or servers) are so bad that the distributed transaction is in a dilemma (in-doubt) and then we have to intervene manually. The paper shows how an in-doubt transaction occurs and how it can be solved.

UVOD

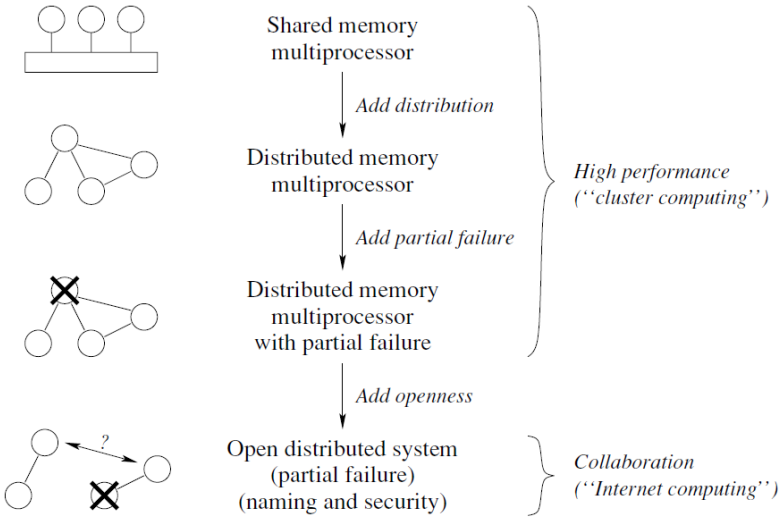
Kada bi to bilo moguće, najbolje bi bilo da ne radimo sa distribuiranim bazama podataka. Tada ne bismo trebali distribuirane transakcije, dvofazni commit protokol, replikaciju podataka. No, ponekad ne možemo izbjeći potrebu za distribuiranim bazama podataka. Moguće je da i kod distribuiranih baza podataka nemamo potrebu za "pravim" distribuiranim transakcijama, u slučaju kada sve transakcije mijenjaju podatke samo u jednoj bazi, a podaci sa drugih baza se u transakciji samo čitaju. Ali, najčešći slučaj je da distribuirana transakcija mijenja podatke u dvije ili više baza, pa tada neminovno imamo potrebu za dvofaznim commit protokolom i tada postoji mogućnost da u toku dvofaznog commit protokola dođe do in-doubt transakcije, zbog pada servera baze ili (češće) zbog problema na vezama.

Rad je podijeljen u šest točaka. U 1. točki prikazujemo podjelu distribuiranih sustava i neke jednostavne protokole na temelju [3]. U 2. točki podsjećamo se na arhitekturu Oracle baze i rad sa transakcijama u centraliziranom slučaju. Distribuiranu bazu podataka ukratko prikazujemo u 3. točki, a replikaciju podataka u 4. točki. U 5. točki prikazujemo distribuiranu transakciju i dvofazni commit protokol. U 6. točki govorimo o in-doubt distribuiranoj transakciji.

1. DISTRIBUIRANI SUSTAVI

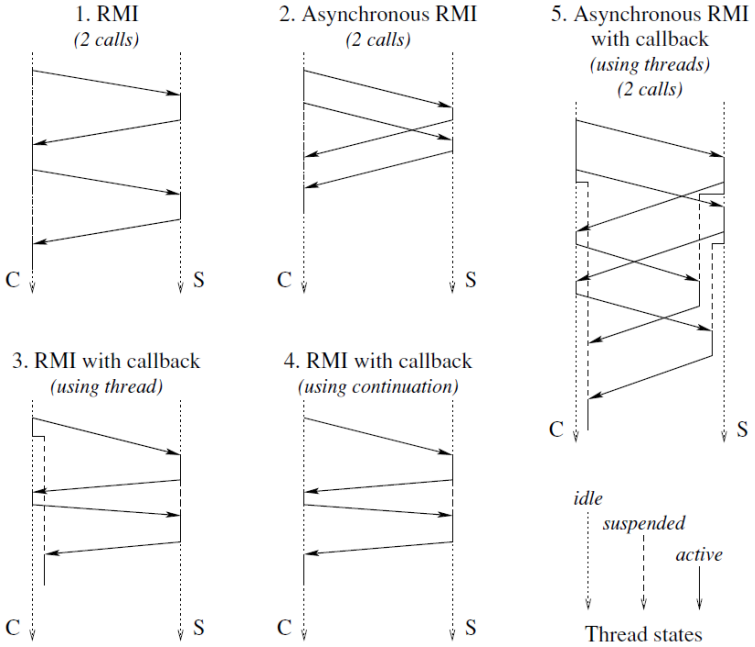
U [3] autori govore o modelima kompjutorskog programiranja, pa prikazuju i distribuirani model. Kod toga navode da se distribuirani sustavi mogu javiti na nekoliko razina. Na najnižoj razini možemo promatrati multiprocesorski sustav jednoga računala, kod kojega više procesora dijeli zajedničku memoriju. Iako ovo na prvi pogled ne djeluje kao distribuirani sustav, on to zapravo jeste, jer se već ovdje javljaju problemi zajednički distribuiranim sustavima – više "agenata" (u ovom slučaju procesora) međusobno komunicira (u ovom slučaju preko djeljive memorije). Sljedeća razina su distribuirani procesori koji imaju distribuiranu memoriju.

U prethodna dva slučaja nije se razmišljalo o padu jednog "agenta", pa se na sljedećoj razini i to uzima u obzir. Autori te tri razine zajednički nazivaju high performance ili "cluster" computing. Na sljedećoj razini je "pravi" distribuirani sustav, koji je otvoren, te isto podložan parcijalnim greškama. Navedeno prikazuje slika 1.:



Slika 1. Jednostavna taksonomija distribuiranih sustava; Izvor: [3]

Na svim navedenim razinama, a za četvrtu razinu to je i najuobičajenije, komunikacija se može izvoditi pomoću slanja poruka. Kod slanja poruka postavlja se jedno važno pitanje – da li poruke slati sinkrono ili asinkrono. Kod sinkronog slanja poruka, pošiljatelj čeka da mu primatelj odgovori na poruku, pa tek onda nastavlja sa radom. Kod asinkronog slanja poruka, pošiljatelj ne čeka na odgovor, već nastavlja sa radom, sve do trenutka kada mu taj odgovor neizostavno treba. Sljedeća slika (2.) prikazuje dijagrame poruka kod nekih jednostavnih protokola (RMI označava Remote Method Invocation):



Slika 2. Dijagrami poruka kod jednostavnih protokola; Izvor: [3]

Kako kažu autori u [3], nažalost, i sinkrona i asinkrona komunikacija imaju svoje dobre i loše strane. Sinkrona komunikacija omogućava rano otkrivanje grešaka, a greške se jednostavnije rješavaju. Asinkrona komunikacija ima bolje performanse od sinkrone, ali otežava rano otkrivanje grešaka i rješavanje grešaka je teže.

2. ARHITEKTURA ORACLE BAZE, TRANSAKCIJE

Arhitekturu Oracle sustava čine dva glavna dijela [2]:

- Baza podataka (BP), koju čine različite vrste datoteka; najvažnije su datoteke podataka, no postoji i desetak drugih vrsta datoteka, od kojih su posebno važne Redo Log datoteke, koje se dijele na Online i Archive; Online Redo Log datoteke služe za oporavak baze u slučaju pada sustava (npr. programske greške ili nestanaka struje), dok Archive Redo Log datoteke služe za uspostavu prijašnjeg stanja, zajedno sa backup datotekama, u slučaju kvara medija (u pravilu – diska);

- Instanca (jedna ili više) baze podataka, koju čine memorijske strukture i procesi u memoriji; od memorijskih struktura, naročito su zanimljivi Block Buffer Cache i Redo Bufer.

Na koji način se radi čitanje (SELECT) i izmjena podataka (INSERT/UPDATE/DELETE) u Oracle sustavu? Klijentski procesi (klijentski proces može biti proces na drugom računalu, proces na istom računalu na kojem se nalazi i Oracle RSUBP, ali može biti i neki proces unutar Oracle RSUBP-a) nikad ne dobivaju podatke direktno sa diska. Svi podaci koji se čitaju sa diska smještaju se u Block Buffer Cache. Također, kada klijentski proces mijenja podatke, ne mijenja ih direktno na disku, već se promjene prvo spremaju u Block Buffer Cache. Zapravo, mijenjanje podataka ne ide tako da se podaci direktno iz Block Buffer Cachea upisuju na disk, već se upis radi preko Redo Buffera. Osim toga, mijenjanje podataka ne ide tako da se podaci odmah upisuju u tablice podataka, već se iz Redo Buffera upisuju u Online Redo Log datoteku

Online Redo Log čine minimalno dvije datoteke (a preporučljivo je da postoje barem tri) koje se koriste u krug – kada se prva napuni, Oracle počinje pisati u drugu, a kada se druga napuni Oracle se vraća na prvu. Kada su podaci zapisani u Online Redo Log može doći do pada sustava (to nije kvar medija) prije nego Oracle te podatke upiše u prave tablice podataka. No, nakon što se Oracle sustav ponovno podigne, pročitat će Online Redo Log i upisati podatke u prave tablice. Naravno, pitanje je da li ti podaci zaista trebaju biti trajno zapisani u tablicu podataka – to ovisi o tome da li su oni commit-irani.

Moglo bi se pomisliti da Oracle sve podatke koji nisu commit-irani drži u memoriji, u Block Buffer Cacheu, a da se tek kod commit-a svi podaci prepisuju na disk. No, to je nemoguće, jer Block Buffer Cache, koliko god bio velik, nikada ne može biti dovoljno velik da svi mijenjani podaci stanu u njega. Stoga se podaci iz Block Buffer Cachea često upisuju u tablice podataka (posredstvom Redo Buffera i Online Redo Loga) prije nego je transakcija commit-irana. Stoga se nakon pada sustava, i nakon što Oracle pročita Online Redo Log i upiše podatke (koji još nisu upisani) u prave tablice, tj. nakon faze koju bismo mogli nazvati REDO fazom, zbiva UNDO faza, u kojoj se oni podaci koji nisu commit-irani brišu iz Oracle tablica podataka. Takav postupak, da se prvo radi REDO, a onda UNDO faza, naziva se ARIES (Algorithms for Recovery and Isolation Exploiting Semantics).

Može se postaviti pitanje – otkuda Oracle uzima stare podatke, koji su mu potrebni da napravi UNDO fazu? Moglo bi se pomisliti da se ti podaci nalaze u Online Redo Log datoteci, tj. da ona čuva sliku redaka kakvi su bili prije promjene (pa bi se ta slika koristila za UNDO) i sliku redaka nakon promjene (pa bi se ta slika koristila za REDO). U stvarnosti Online Redo Log, kako mu i samo ime kaže, sadrži samo sliku za REDO. Podaci potrebni za UNDO nalaze se u jednom posebnom prostoru na disku, koji se naziva UNDO tablespace (uz tablespaceove za tablice podataka). Oracle zapisuje staro stanje redaka u UNDO tablespace uvijek kada radi izmjenu podataka (INSERT / UPDATE / DELETE).

UNDO tablespace, osim što služi za eliminiranje (iz tablica podataka) promjena koje nisu commit-irane, kod oporavka sustava nakon pada, služi i za omogućavanje višekorisničkog rada. Naime, jedan od važnijih zahtjeva na RSUBP sustav je da jedna sesija (baze podataka) ne vidi promjene koje je napravila druga sesija, sve dok ta druga sesija ne napravi COMMIT. No, budući da se često ne-commit-irani podaci moraju spremati na disk, pitanje je otkuda prva sesija može čitati stare podatke (prije promjene). Oni nisu u tablici podataka (tamo su promijenjeni podaci), nisu niti u Online Redo Log datoteci – oni se nalaze u UNDO tablespaceu. Oracle ih iz UNDO tablespace-a čita u Block Buffer Cache, jer klijentski proces sve podatke (pa i stare) čita iz Block Buffer Cachea, a ne direktno sa diska.

Postojanje UNDO tablespacea (ili neke slične strukture u nekom drugom RSUBP sustavu) omogućava da mijenjanje podataka ne utječe na čitanje podataka, tj. mijenjanje podataka ne sprečava da se ti podaci istovremeno i čitaju (zapravo, čitaju se stara stanja tih podataka). U RSUBP sustavima koji nemaju nešto slično kao što je UNDO tablespace, mijenjanje podataka utječe na čitanje, tj. sesija koja mijenja podatke postavlja ključ nad tim mijenjanim redovima, taj ključ onda sprečava druge sesije čak i da čitaju podatke, čime dolazi do znatnog usporavanja korisničkog rada.

Možemo reći da su ovo neke najvažnije osobine Oracle transakcije (u centraliziranom slučaju):

1. Sesija (baze podataka) ne vidi promjene koje je napravila druga sesija, dok druga sesija ne napravi COMMIT (ili ROLLBACK). Međutim, sesije nisu nezavisne, jer zaključavanje redaka u jednoj sesiji utječe na drugu sesiju koja pokušava ažurirati redak koji je zaključala prva sesija.

2. Kada sesija izvršava DML naredbu (INSERT, UPDATE, DELETE), automatski se zaključa redak. Redak se može otključati tek na kraju transakcije (COMMIT ili ROLLBACK), ili pomoću ROLLBACK TO SAVEPOINT. Međutim, redovi koji su otključani sa ROLLBACK TO SAVEPOINT ostaju zaključani za one sesije koje su već prije toga pokušale izvršiti DML na tim redovima. Redovi se mogu zaključati i sa SELECT ... FOR UPDATE. Pritom postoji i opcija NOWAIT / WAIT n (sekundi), npr SELECT ... FOR UPDATE NOWAIT. DML naredbe nemaju opciju NOWAIT - uvijek moraju čekati dok se redak ne otključa. Može se zaključati i cijela tablica, sa LOCK TABLE ... NOWAIT / WAIT n (sekundi).

3. Transakcija može ili u cijelosti uspjeti (COMMIT), ili se u cijelosti poništiti (ROLLBACK). Naravno, ne znači da sve DML radnje unutar transakcije moraju uspjeti, jer one DML radnje koje su uzrokovale grešku, ali je greška obrađena, neće uspjeti.

4. DML naredba može ili u cijelosti uspjeti ili se njen efekat u cijelosti poništava. Pritom se mogu desiti tri tipa grešaka:
- narušen uvjet na tip stupca, npr. ako u NUMBER (2) pokušamo upisati broj 1000;
- narušeno deklarativno integritetno ograničenje (PK, UK, FK, CK, NOT NULL); zapravo provjera deklarativnih ograničenja može se i odgoditi (najkasnije do COMMIT) - tada naredba može uspjeti i ako deklarativna ograničenja nisu zadovoljena;
- narušena proceduralna ograničenja (okidači baze); to je najkompliciraniji slučaj, jer npr. jedna UPDATE naredba može okinuti različite UPDATE okidače, koji mogu pozivati procedure koje dalje rade DML, čime se okidaju drugi okidači.

5. Ako se desila greška na bazi koja nije obrađena, a početak je DML naredba, svi efekti se poništavaju (prethodna točka). Ako se desila greška na bazi koja nije obrađena, a početak je poziv procedure (ili funkcije) sa strane klijenta (npr. Forms ili Java), ili poziv sa strane druge baze (udaljena procedura), svi efekti procedure se poništavaju.

Često se kaže da transakcija treba zadovoljavati ACID svojstva (ACID property):

- Slovo A označava atomarnost (Atomicity);

- C označava konzistentnost (Consistency);

- I označava izoliranost (Isolation);

- D označava trajnost (Durability).

Napomenimo da je konzistentnost transakcije usko povezana sa atomarnošću, ali je za konzistentnost odgovoran programer, a ne baza.

Izoliranost se često zove i serijabilnost (serializability). Misli se na to da bi transakcije trebale uvijek ostaviti efekt kao da se izvršavaju serijski (jedna za drugom), iako se izvršavaju konkurentno (paralelno ili kvazi-paralelno). Da bi se postigla serijabilnost, DBMS sustavi primjenjuju dvofazno zaključavanje (two-phase locking), koje ne treba miješati sa dvofaznim commit protokolom (koji se koristi kod commit-iranja distribuirane transakcije). U fazi širenja (growing phase), transakcija zaključava retke, a poslije ih samo otključava, u fazi stezanja (shrinking phase).

Skoro svi DBMS sustavi koriste specifičnu varijantu dvofaznog zaključavanja ([3]), striktno dvofazno zaključavanje (strict two-phase locking), kod kojeg se faza stezanja radi neposredno prije kraja transakcije, prije commit-iranja (ili rollback-iranja). Ta varijanta eliminira problem kaskadnog abortiranja (cascading abort).

3. DISTRIBUIRANA BAZA PODATAKA

Sustav distribuiranih baza podataka (ili jednostavnije - distribuirana baza podataka) je sustav od dvije ili više baza podataka koje bi aplikacijama (korisničkim programima) trebale izgledati kao jedna jedinstvena baza podataka. Date je u svojoj knjizi [1] postavio "temeljni princip za distribuirane baze podataka", a to je: "Za korisnika, distribuirani sustav (baza podataka) treba izgledati potpuno isto kao ne-distribuirani sustav". Na temelju tog principa, Date u knjizi daje 12 zahtjeva koje distribuirana baza mora zadovoljiti. Činjenica je da je 100%-tno zadovoljenje svih tih zahtjeva gotovo nemoguće, ali ti zahtjevi služe kao ideal prema kojemu bi trebale težiti konkretne implementacije distribuiranih baza podataka.

Ne ulazeći detaljnije u definiranje svakog od ovih zahtjeva, može se reći da ih Oracle baza podataka zadovoljava u velikoj mjeri. U zadovoljavanju tih zahtjeva veliku ulogu igra nepostojanje "pravog" globalnog rječnika podataka u Oracle bazi, jer svaka Oracle baza ima svoj lokalni rječnik podataka. Kad bi postojao "pravi" globalni rječnik podataka koji bi se nalazio samo na jednoj bazi, to bi bilo u neskladu sa 2. zahtjevom. Ako se pak "pravi" globalni rječnik distribuira po svim bazama podataka, to nije u skladu sa 1. zahtjevom.

S druge strane, lokalna baza ipak mora sadržavati neke informacije o bazama sa kojima komunicira. Oracle baza (ali i druge baze) to radi tako da lokalni rječnik sadrži podatke o udaljenim objektima baze.

Za čuvanje informacija o udaljenim objektima, Oracle baza podataka koristi tzv. "database link". Može se reći da je database link objekt baze podataka koji definira jednosmjernu vezu baze podataka na drugu bazu podataka. Postoji više vrsta database linkova. Jedna je podjela na globalne (koji pripadaju bazi) i privatne database linkove (koji pripadaju određenoj shemi baze), a druga podjela dijeli database link-ove prema načinu na koji se korisnik prijavljuje na udaljenu bazu (connected user, fixed user ili current user link). Slijedi primjer definiranja privatnog database linka koji je po drugoj podjeli "fixed user link" (pretpostavimo da smo database link kreirali kao objekt sheme "shemaX" unutar baze "bazaA"):

```
CREATE DATABASE LINK neki_link
  CONNECT TO shemaY IDENTIFIED BY zaporka USING "bazaB";
```

U ovom slučaju ime database linka je "neki_link", a on pokazuje na shemu "shemaY" koja se nalazi na bazi koja je polaznoj bazi (u kojoj definiramo database link) poznata pod imenom "bazaB". Moramo reći da za razliku od "shemaY", što je pravo ime sheme na drugoj bazi, "bazaB" ne mora biti pravo ime druge baze, nego je to tzv. alias (ili service name) tj. ime pod kojim prva baza (bazaA) "poznaje" drugu bazu. Na temelju tog database link-a, korisnik koji je vlasnik sheme "shemaA" (ili neki korisnik koji ima pravo korištenja objekata te sheme) može sada postaviti npr. sljedeći udaljeni upit (remote query):

```
SELECT * FROM neka_tablica@neki_link;
```

Koristeći sinonime (globalne ili privatne) možemo postići da udaljeni upit izgleda kao lokalni upit, tj. nakon kreiranja (u ovom slučaju privatnog) sinonima:

```
CREATE SYNONYM kvazi_lokalna_tablica FOR neka_tablica@neki_link;
```

možemo postaviti sljedeći upit, koji izgleda kao lokalni, iako je udaljeni (to je u skladu sa 4. zahtjevom):

```
SELECT * FROM kvazi_lokalna_tablica;
```

Osim udaljenih upita (remote query), koji čitaju tablice samo sa jedne (udaljene) baze, Oracle podržava i distribuirane upite (7. zahtjev), tj. takve upite u kojima se istovremeno čitaju podaci iz tablica koje se nalaze na dvije ili više lokacija, npr:

```
SELECT * FROM lokalna_tab, udaljena2@baza2, udaljena3@baza3 WHERE ...
```

Osim distribuiranog upita, Oracle podržava i distribuirane DML (INSERT / UPDATE / DELETE) naredbe. Dateov 8. zahtjev govori o upravljanju distribuiranim transakcijama. Distribuirana transakcija je ona transakcija u toku koje se ažuriraju (tj. unose, mijenjanju ili brišu) redovi iz barem dvije tablice koje se nalaze u različitim bazama podataka. Dakle, distribuirana transakcija ne mora nužno sadržavati distribuirane upite ili distribuirane DML naredbe, već može sadržavati npr. dvije nedistribuirane DML naredbe, ali takve da jedna DML naredba ažurira tablicu u jednoj bazi, a druga naredba ažurira tablicu u drugoj bazi (pritom jedna baza može biti lokalna, a druga udaljena, ili obje baze mogu biti udaljene). Suprotno tome, transakcija može sadržavati distribuirane upite ili distribuirane DML naredbe, a da pritom ipak ne bude distribuirana transakcija.

Kao i svaka transakcija, tako i distribuirana transakcija mora zadovoljavati neke uvjete. Transakcija je logička jedinica rada (logical unit of work) koja sadrži jednu ili više SQL naredbi. Transakcija predstavlja jedinstvenu (atomsku) jedinicu rada čiji efekti moraju biti trajno pohranjeni (nakon COMMIT) ili potpuno eliminirani (nakon ROLLBACK) iz baze podataka. Transakcija mora ostaviti podatke u konzistentnom stanju i mora biti nezavisna od drugih transakcija. Kod distribuirane transakcije se navedeni zahtjevi rješavaju kroz tzv. dvofazni protokol potvrde transakcije, ili (kraće) dvofazni commit protokol (two-phase commit protocol), o kojem će detaljnije biti riječ u 5. točki.

4. REPLIKACIJA PODATAKA

Općenito govoreći, replikacija podataka je kopiranje podataka i može postojati neovisno o tome da li postoji / ne postoji distribuirana baza podataka. Npr. mogu se replicirati podaci unutar iste baze, ili (što je puno korisnije) mogu se replicirati podaci između nepovezanih baza (tj. baza koje ne čine distribuiranu bazu podataka). No, kad se govori o replikaciji, najčešće se misli na replikaciju baš unutar distribuirane baze podataka. Zato se može reći da je replikacija proces kreiranja kopija podataka (i metapodataka) i usklađivanja (sinkroniziranja) podataka (i metapodataka) u bazama koje čine distribuiranu bazu podataka. Sve promjene koje se naprave na jednoj bazi (distribuirane baze) moraju se odraziti i na kopije podataka u ostalim bazama (koje imaju te kopije).

Oracle ima više varijanti replikacije. Postoji bazična replikacija ([4]), koja je raspoloživa i u Standard ediciji baze i koja omogućava samo asinkronu replikaciju, sa jednom master bazom. Napredna replikacija ([6]), za razliku od bazične, omogućava i sinkronu replikaciju, multi-master replikaciju i proceduralnu replikaciju. Novija Oracle replikacija je tzv. Oracle Streams [7], a ona je replikacija temeljena na logu.

Naglasimo da Date u [1] tvrdi da je samo sinkrona replikacija "prava" replikacija, jer jedino ona osigurava konzistentnost podataka u bilo kojem trenutku. Asinkrona replikacija je obično brža, ali nauštrb konzistentnosti.

Bazična replikacija uključuje Read-only MV (Materialized View), materijalizirani pogled koji se može samo čitati, i Updateable MV, koji se može i ažurirati. Bazična replikacija podržava replikaciju samo DML, ne i DDL naredbi. Read-only MV (ili samo MV) radi na relativno jednostavnom principu. MV baza (koja sadrži MV) ima database link na drugu bazu (master bazu) i u određenim intervalima čita podatke iz odgovarajuće tablice master baze i tim podacima puni svoj MV (MV baza vuče podatke - pull, umjesto da ih master baza njoj gura - push). Mehanizam se može ubrzati na taj način da master baza uz svoju tablicu sadrži log tablicu koju (sinkrono) puni okidačem baze, pa se na taj način MV ne mora svaki puta puniti "od nule", već se nad MV-om izvršavaju samo ona ažuriranja koja su nastala od prethodnog osvježavanja, tzv. delta podaci (to je tzv. FAST refresh, za razliku od COMPLETE refresh).

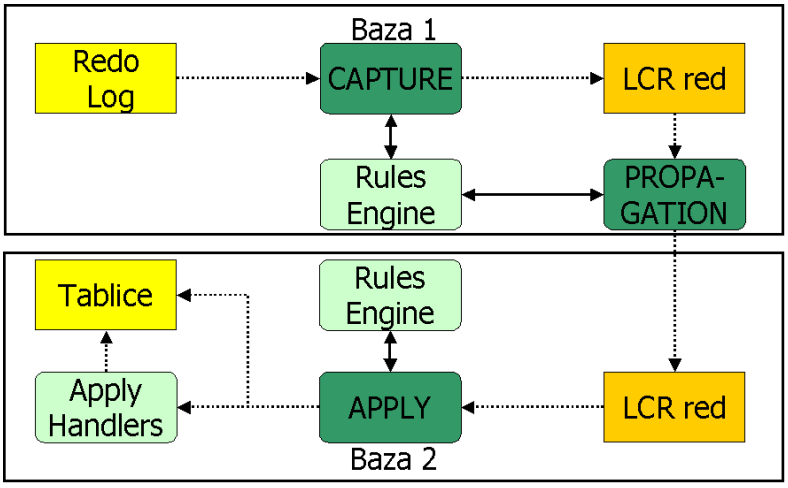
Napredna replikacija, za razliku od bazične, može replicirati i DDL, a ne samo DML naredbe. Postoje dvije varijante. Glavna varijanta je multi-master replikacija, koja je retčana, a druga varijanta je proceduralna replikacija. Obje varijante mogu biti sinkrone ili asinkrone, i obje su dvosmjerne.

Sinkrona multi-master replikacija relativno je jednostavna. Kad jedna master baza izvršava DML naredbu, okida se odgovarajući okidač koji odmah zove udaljenu proceduru na drugoj bazi (ne-odgođeni RPC), a ta udaljena procedura izvršava DML naredbu na udaljenoj master bazi. Budući da se sve odvija u okviru jedne distribuirane transakcije, dvofazni commit protokol osigurava da nema konflikta u podacima. Naravno, problem je u tome što sve baze koje sudjeluju u sinkronoj replikaciji moraju biti raspoložive, inače transakcija ne uspijeva.

Asinkrona multi-master replikacija vrlo je slična UMV replikaciji, tj. onom njenom dijelu u kojem se (asinkrono) repliciraju promjene sa UMV-a na master tablicu. Kad jedna master baza izvršava DML naredbu, okida se odgovarajući okidač, ali on ne poziva udaljenu proceduru odmah, nego stavlja RPC poziv u red (deferred transaction queue) koji se nalazi u prvoj bazi. Podaci (tj. pozivi procedura) iz tog reda prenose se u odgovarajući red na drugoj bazi, čitaju se iz tog reda i onda se izvršavaju procedure zapisane u redu. Kod asinkrone multi-master replikacije mogućnost pojave konflikata veća je nego kod UMV-ova, a konflikti se moraju rješavati na više mjesta (na svim master tablicama). Ako master baza iz nekog razloga (npr. ne radi veza) ne može poslati podatke udaljenoj master bazi, sprema greške u vlastiti deferred error queue, a nakon toga sve ostale promjene idu u taj red, drugim riječima - replikacija se prekida dok DBA ne razriješi problem. Ako master baza uspješno pošalje podatke udaljenoj master bazi, ali ih ona (iz nekog razloga, npr. integritetnog ograničenja) ne može primijeniti, greške se skupljaju u redu (deferred error queue) na udaljenoj bazi.

Niti bazična, niti napredna multi-master replikacija nisu zamišljene za repliciranje velikog broja podataka. Zbog toga je u okviru napredne replikacije Oracle napravio i proceduralnu replikaciju, kod koje se ne repliciraju podaci red po red, već se replicira samo poziv procedure, koja (procedura) na udaljenoj bazi može istovremeno ažurirati više redaka (a ne jedan po jedan). Međutim, kod proceduralne replikacije postoji velika mogućnost pojave konflikata u podacima, a (za razliku od multi-master replikacije) Oracle tu ne pruža programeru nikakve pomoćne mehanizme za rješavanje konflikata.

Oracle Streams (stream = tok) replikacija temelji se na log podacima (log-based replication). Oracle Streams replikacija je isključivo asinkrona, retčana (row-level), dvosmjerna replikacija, a može replicirati i DML i DDL naredbe. Oracle Streams namijenjen je i za punjenje skladišta podataka (Data Warehousing) i za upravljanje redovima poruka (Message Queuing). Oracle Streams se temelje na "nižem sloju", a to je PL/SQL paket DBMS_LOGMNR (i prateći paketi) koji se pojavio već u bazi 8. U bazi 9.2 se zajedno sa Oracle Streams pojavio i novi PL/SQL paket DBMS_RULE (i prateći paketi), koji Streams-ima koristi kao "rules engine", tj. služi za donošenje odluka o prihvaćanju ili odbacivanju redaka podataka u Streams procesima. Streams procesi su Capture, Propagation i Apply proces, tj. procesi za skupljanje, prenošenje i primjenu promjena sa jedne (izvorne) baze na drugu (odredišnu) bazu, što prikazuje slika 3.:



Slika 3. Oracle Streams procesi

Budući da veliki broj naših korisnika ima Standard ediciju Oracle baze, koja nema naprednu replikaciju (koja uključuje i sinkronu replikaciju), a trebala nam je sinkrona replikacija, napravili smo vlastito rješenje sinkrone replikacije. Sljedeća tablica 1. prikazuje različite osobine Oracle varijanti replikacije i naše replikacije:

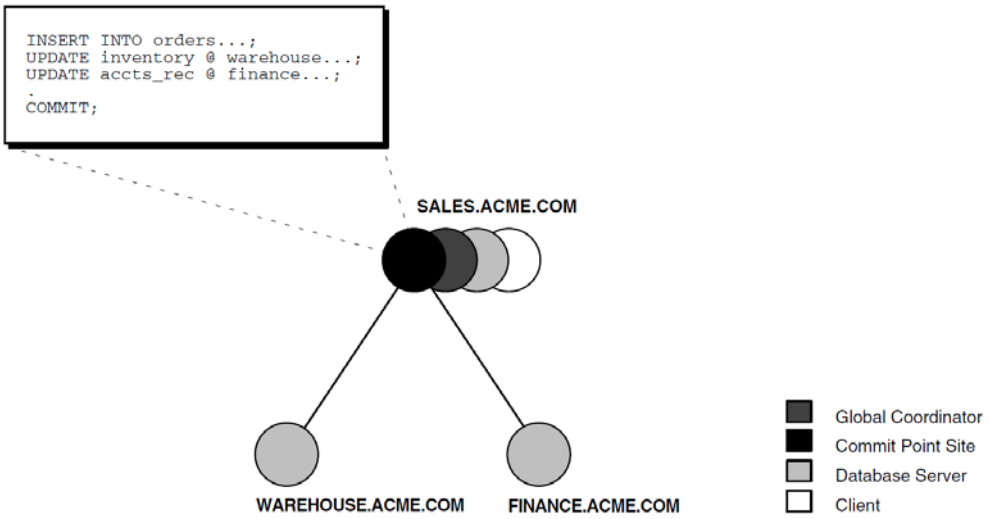
Tablica 1: Osnovne osobine Oracle i naše replikacije

Replikacija / Osobine	MV	UMV	Multi-Master	Proce-dural	Streams	Naša
Transakcijska ili Log bazirana	T	T	T	T	L	T
Sinkrona ili Asinkrona	A	A	S / A	S / A	A	S
Retčana ili Proceduralna	R	R	R	P	R	R
Jednosmjerna ili Dvosmjerna	J	D	D	D	D	J

5. DISTRIBUIRANE TRANSAKCIJE I DVOOFAZNI COMMIT PROTOKOL

Ova točka napravljena je na temelju [5], poglavlje 34.

Pretpostavimo da nam se distribuirana baza podataka sastoji od tri baze, koje izvode distribuiranu transakciju, kako prikazuje sljedeća slika 4.:



Slika 4. Različite uloge baza u distribuiranoj transakciji; Izvor: [5]

Vidljivo je da u prikazanoj distribuiranoj transakciji, koja radi DML na sve tri baze, baze imaju različite uloge, pri čemu jedna baza može imati više uloga. Uz postojeće četiri vrste uloga koje su prikazane na slici, postoji još jedna uloga - lokalni koordinator transakcije. Značenja pojedinih uloga su sljedeća:

- Klijent (client): baza koja referencira podatke na drugim bazama;

- Server baze podataka (database server): bilo koja baza podataka koja sadrži podatke koje druge baze (uključujući samu sebe) referenciraju;

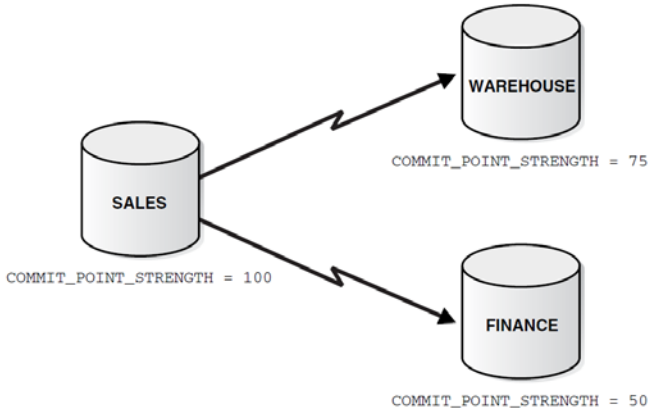
- Lokalni koordinator (local coordirnator): baza koja mora referencirati podatke na drugim bazama da bi završila svoj dio distribuirane transakcije. Svaki koordinator vidi samo svoje neposredne susjede. Lokalni koordinator komunicira sa svojim "podređenim" susjednim bazama, šalje im upite i prima od njih rezultate (ali i informacije o statusu transakcije), pa onda te rezultate šalje "nadređenim" susjednim bazama (koje su upite inicirale). Na prethodnoj slici nema lokalnog koordinatora, odnosno baza SALES je i globalni i lokalni koordinator;

- Globalni koordinator (global coordinator): baza koja je izvorište distribuirane transakcije. Aplikacija koja je pokrenula distribuiranu transakciju direktno je vezana za bazu koja je globalni koordinator. Globalni koordinator radi sljedeće operacije za vrijeme distribuirane transakcije:
  1. Šalje SQL naredbe ili pozive udaljenih procedura na određenu bazu;
  2. Šalje svim bazama sa kojima je direktno povezan, osim commit point site bazi, naredbe da se pripreme;
  3. Ako sve baze odgovore da su se pripremile, šalje commit point site bazi naredbu da izvrši commit;
  4. Ako neka baza odgovori sa abort, šalje svim bazama naredbu da izvrše globalni rollback;

- Commit point site: to je baza koja inicira globalni commit ili globalni rollback, kada dobije takvu instrukciju od globalnog koordinatora. Baza koja će biti commit point site trebala bi biti ona koja čuva najvažnije podatke. Razlikuje se od ostalih baza u distribuiranoj transakciji po sljedeće dvije karakteristike:
  1. Ona nikada ne ulazi u pripravno stanje (prepared state). Za razliku od toga, ostale baze kod greške ostaju u pripremnom stanju, držeći (potrebne) zaključane retke, dok god se ne riješi in-doubt transakcija;
  2. Ona izvršava commit prije svih ostalih baza. Nakon toga je distribuirana transakcija commit-irana, jer se globalni koordinator brine da sve druge baze isto izvrše commit.

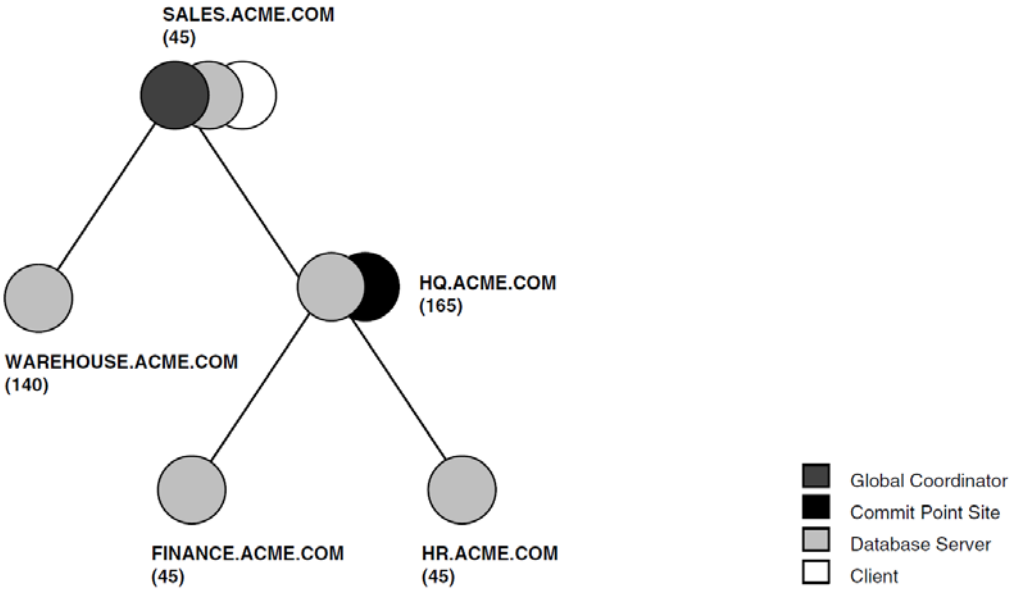
Koja će baza kod distribuirane transakcije biti odabrana kao commit point site, ovisi (i) o parametru baze COMMIT_POINT_STRENGTH. Globalni koordinator na početku pripremne faze (prepare phase) distribuirane transakcije bira između svojih neposrednih susjednih baza (ali samo onih koje su sudjelovale u distribuiranoj transakciji) onu bazu koja ima najveći COMMIT_POINT_STRENGTH. Izabrana baza dalje bira po svojim "podređenim" susjednim bazama (koje su sudjelovale u distribuiranoj tranakciji), opet na temelju COMMIT_POINT_STRENGTH parametra. Kada se izabere baza koja će biti commit point site, globalni koordinator šalje poruku za pripremu svim ostalim bazama, osim njoj.

Sljedeća slika 5.2 prikazuje kako će se odabrati commit point site u našem slučaju sa tri baze. Vidi se da baza SALES, koja je globalni koordinator, ima najveću vrijednost COMMIT_POINT_STRENGTH parametra, pa će ona istovremeno biti i commit point site (kako je bilo i prikazano na slici 5.):



Slika 5. Baza SALES će biti odabrana kao commit point site; Izvor: [5]

Na sljedećoj slici 6. prikazan je složeniji slučaj od prethodnog. Ovdje distribuirana transakcija uključuje pet baza, a kao commit point site odabrana je baza HQ (koja nije globalni koordinator), zato što ona ima najveću vrijednost COMMIT_POINT_STRENGTH parametra. Na slici nije prikazano da je ta baza HQ istovremeno i lokalni koordinator (ima "podređene" susjedne baze FINANCE i HR, koje isto sudjeluju u distribuiranoj transakciji):



Slika 6. Baza HQ će biti odabrana kao commit point site; Izvor: [5]

Nakon objašnjavanja različitih uloga baza koje sudjeluju u distribuiranoj transakciji, može se detaljnije prikazati dvofazni commit protokol. Zapravo, on se u Oracle bazi sastoji od tri faze:

- Faza pripreme (prepare phase);
- Faza potvrde (commit phase);
- Faza zaboravljanja (forget phase).

Naglasimo da se te tri faze rade samo ako distribuirana transakcija završava sa COMMIT. Ako distribuirana transakcija završava sa ROLLBACK, globalni koordinator jednostavno šalje svim bazama da naprave ROLLBACK.

U fazi pripreme (prepare phase), globalni koordinator traži od ostalih baza da se pripreme (lokalni koordinatori u tome pomažu, šaljući poruke svojim neposrednim podređenim susjedima), tj. pređu u pripravno stanje (prepared state). Kako je prije rečeno, to se jedino ne traži od baze koja je izabrana kao commit point site. Baze prelaze u pripravno stanje tako da spreme sve podatke u svoju redo log datoteku, bez obzira da li će kasnije izvršiti COMMIT ili ROLLBACK. Također, u ovoj fazi baze stavljaju distribuirani lokot na sve modificirane tablice – **ovo je krucijalno, jer taj lokot sprečava čak i čitanje podataka (što Oracle baza inače ne radi), pa u slučaju problema sa distribuiranom transakcijom podaci ostaju zaključani i za čitanje!** Nakon što je određena baza javila svom nadređenom (lokalnom ili globalnom koordinatoru), da je pripravna, čeka daljnje upute. Inače, baza može odgovoriti na sljedeća tri načina:

- Prepared: baza javlja da je pripravna. Kako je već rečeno, tada zaključava modificirane tablice čak i za čitanje. Čim je neka baza prešla u pripravno stanje, distribuirana transakcija je u osjetljivom stanju (u [5] na str. 34-10 kaže se da je to in-doubt stanje, ali se na str. 34-11 in-doubt stanjem naziva samo stanje kod greške; zato smo ovdje upotrijebili izraz osjetljivo stanje) i ostaje u njemu dok god se sve promjene ne commit-iraju ili rollback-iraju;

- Read-only: baza javlja da ona nije radila nikakav DML, pa ne treba ići u pripravno stanje;

- Abort: baza javlja da se ne može pripremiti, otključava sve svoje zaključane retke i radi lokalni ROLLBACK.

Ako globalni koordinator od barem jedne baze dobije odgovor Abort, on šalje svim bazama naredbu ROLLBACK. Ako od svih baza dobije odgovor Prepared, globalni koordinator započinje fazu potvrde (commit phase):

- Šalje naredbu commit point site bazi da izvrši COMMIT;

- Commit point site baza izvršava COMMIT i obavještava globalnog koordinatora. U ovom trenutku se smatra da je distribuirana transakcija commit-irana, bez obzira na eventualni neuspjeh kod drugih baza;

- Globalni koordinator i lokalni koordinatori šalju naredbe svim svojim podređenim bazama, tražeći od njih da naprave COMMIT;

- Ostale baze rade lokalni COMMIT, otključavaju svoje zaključane retke i javljaju to svom lokalnom koordinatoru, ili globalnom koordinatoru.

U fazi zaboravljanja (forget phase), globalni koordinator kaže commit point site bazi da zaboravi transakciju, tj. da izbriše stanja o transakciji koja ona vodi kod sebe, te da mu to javi natrag. Globalni koordinator nakon toga briše i kod sebe informacije o distribuiranoj transakciji.

Možemo ovako rekapitulirati zbivanja kod uspješne distribuirane transakcije, koja završava sa COMMIT i kod koje nije bilo nikakvih problema:

1. Klijentska aplikacija šalje DML naredbe (ili/i pozive udaljenih procedura) po distribuiranoj bazi i na kraju završava sa COMMIT;
2. Ovdje počinje prva faza. Globalni koordinator (baza na koju je direktno vezana klijentska aplikacija) određuje koja će baza biti commit point site (u tome pomažu i ostale baze, lokalni koordinatori);
3. Globalni koordinator svim bazama, osim commit point site bazi, šalje naredbu da se pripreme;
4. Sve baze javljaju potvrdan odgovor (Prepared);
5. Ovdje počinje druga faza. Globalni koordinator javlja commit point site bazi da izvrši lokalni COMMIT;
6. Commit point site baza izvršava lokalni COMMIT i obavještava globalnog koordinatora;
7. Globalni i lokalni koordinatori javljaju svim ostalim bazama da naprave lokalni COMMIT;
8. Sve baze rade lokalni COMMIT i obavještavaju koordinatore;
9. Ovdje počinje treća faza. Globalni koordinator obavještava commit point site bazu da zaboravi distribuiranu transakciju;
10. Commit point site baza briše svoje podatke o distribuiranoj transakciji i obavještava globalnog koordinatora;
11. Globalni koordinator briše svoje podatke o distribuiranoj transakciji.

6. IN-DOUBT DISTRIBUIRANE TRANSAKCIJE

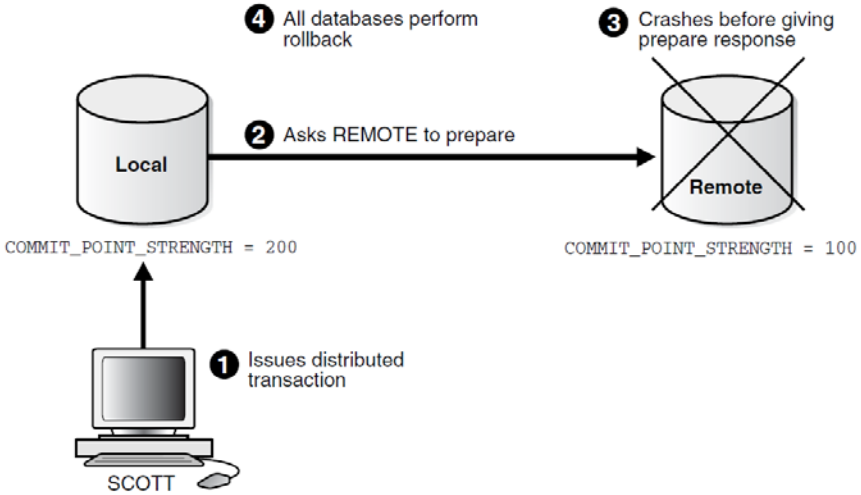
Ova točka napravljena je na temelju [5], poglavlja 34 i 35.

Kako je rečeno u prethodnoj točki, dvofazni commit protokol osigurava da sve baze ili uspješno commit-iraju, ili sve rollback-iraju. U slučaju da se u bilo kojoj fazi desi greška (preciznije, nakon što transakcija postane "osjetljiva", tj. nakon što barem jedna baza završi fazu pripreme), distribuirana transakcija postaje in-doubt. Greške koje se mogu desiti jesu:

- Padne računalo na kojem radi Oracle baza;
- Prekine se veza između dvije ili više baza koje sudjeluju u distribuiranoj transakciji;
- Desi se neki softverski problem.

RECO proces baze najčešće automatski rješava in-doubt transakciju nakon što se podigne računalo, uspostavi veza, odnosno riješi softverski problem. No, dok RECO proces ne riješi problem, ostaju zaključani podaci modificiranih tablica, i to (kako je već rečeno) ne samo za pisanje, već i za čitanje.

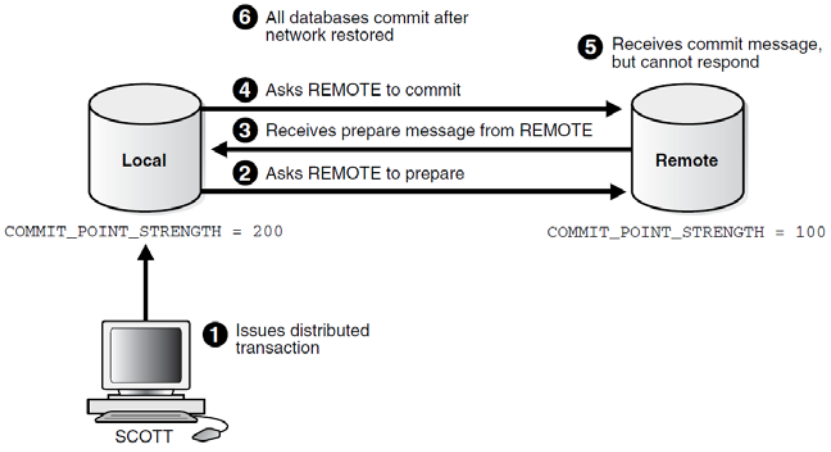
Slijede dva primjera automatskog rješavanja in-doubt transakcije. U prvom primjeru (slika 7.) in-doubt transakcija nastala je u fazi pripreme (prepare phase):



Slika 7. Automatsko rješavanje in-doubt transakcije nastale u fazi pripreme; Izvor: [5]

Primijetimo da ovdje imamo relativno jednostavan slučaj. Kao prvo, imamo samo dvije baze, tako da je baza Remote jedina baza koja drži zaključane podatke za pisanje i čitanje, sve dok se ne razriješi in-doubt transakcija (baza Local, osim što je globalni koordinator, istovremeno je i commit point site baza, pa ne radi pripremnu fazu). Drugačije bi bilo da je broj baza puno veći i da više baza ima zaključane podatke. Kao drugo, ovdje je došlo do automatskog rješavanja in-doubt transakcije, i to (pretpostavljamo) u relativno kratkom periodu, tako da podaci nisu dugo vremena ostali zaključani za pisanje i čitanje. Primjećujemo da je distribuirana transakcija na kraju rollback-irana.

Drugi primjer (slika 8.) prikazuje grešku kod faze potvrde (commit phase):



Slika 8. Automatsko rješavanje in-doubt transakcije nastale u fazi potvrde; Izvor: [5]

U ovom primjeru distribuirana transakcija završila je commit-om. No, i ovdje imamo relativno jednostavan slučaj, iz istih razloga kao što su oni navedeni u prethodnom primjeru.

Najbolje je ostaviti da baza sama razriješi in-doubt transakciju (kao u prethodna dva primjera). No, ako je zadovoljen jedan od ova dva uvjeta, trebali bismo ručno razriješiti in-doubt transakciju:

1. In-doubt transakcija je zaključala kritične podatke ili undo segmente;

2. Pad računala, prekid veze, ili softverski problem ne mogu se riješiti u kratkom vremenu.

Ručno rješavanje in-doubt transakcije može biti kompleksno. Glavni postupak sastoji se od sljedećih koraka:

1. Pronaći identifikacijski broj in-doubt transakcije;

2. Postaviti upit nad sistemskim view-ovima DBA_2PC_PENDING i DBA_2PC_NEIGHBORS, kako bismo odredili da li su baze koje sudjeluju u distribuiranoj transakciji commit-irale;

3. Ako je potrebno, forsirati commit sa COMMIT FORCE, odnosno rollback sa ROLLBACK FORCE naredbom.

Što se tiče pronalaženja identifikacijskog broja in-doubt transakcije, korisnik koji pokreće aplikaciju koja je pokušala izvršiti commit, dobiva jednu od sljedeće tri greške, u kojoj se vidi identifikacijski broj:

```
ORA-02050: transaction ID rolled back,
           some remote dbs may be in-doubt

ORA-02053: transaction ID committed,
           some remote dbs may be in-doubt

ORA-02054: transaction ID in-doubt
```

No, zapravo je veća šansa da će grešku javiti korisnik (ili korisnici) kojima su podaci ostali zaključani ne samo za pisanje, već i za čitanje:

```
ORA-01591: lock held by in-doubt distributed transaction ID
```

Sada kada znamo identifikaciju in-doubt transakcije, možemo analizirati view-ove DBA_2PC_PENDING i DBA_2PC_NEIGHBORS. View DBA_2PC_PENDING prikazuje sve in-doubt distribuirane transakcije. Prazan je ako ih nema, a nakon što se in-doubt transakcije riješe, isto bi se trebao automatski isprazniti (u većini slučajeva). View ima sljedeće stupce:

```
Name                                      Null?    Type
----------------------------------------- -------- --------------
LOCAL_TRAN_ID                             NOT NULL VARCHAR2(22)
GLOBAL_TRAN_ID                                     VARCHAR2(169)
STATE                                     NOT NULL VARCHAR2(16)
MIXED                                              VARCHAR2(3)
ADVICE                                             VARCHAR2(1)
TRAN_COMMENT                                       VARCHAR2(255)
FAIL_TIME                                 NOT NULL DATE
FORCE_TIME                                         DATE
RETRY_TIME                                NOT NULL DATE
OS_USER                                            VARCHAR2(64)
OS_TERMINAL                                        VARCHAR2(255)
HOST                                               VARCHAR2(128)
DB_USER                                            VARCHAR2(30)
COMMIT#                                            VARCHAR2(16)
```

Naročito su zanimljiva dva stupca, STATE i MIXED. Stupac MIXED trebao bi uvijek sadržavati vrijednost NO, osim u slučaju ako smo greškom na nekoj bazi forsirali commit, a na drugoj rollback, pa je distribuirana transakcija ostala u "miksanom" stanju, što baza sama prepoznaje i stavlja vrijednost YES.

Stupac STATE ima sljedeće moguće vrijednosti:

- Collecting: ovu vrijednost normalno sadrže samo baze koje su globalni ili lokalni koordinatori; To znači da te baze prikupljaju informacije sa ostalih servera baze, prije nego odluče da li će se i one pripremiti;

- Prepared: znači da je baza pripremljena (što znači i da je zaključala retke modificiranih podataka i za pisanje i za čitanje), ali mogla je to javiti, ili ne stići javiti svom koordinatoru; baza čeka na poruku da napravi commit;

- Committed: znači da je ova baza commit-irala, ali nisu commit-irale baš sve druge baze (jer inače ne bi bilo in-doubt transakcije);

- Forced commit: znači da je transakciju, koja je prije bila u statusu Prepared, administrator ručno commit-irao sa COMMIT FORCE;

- Forced rollback: znači da je transakciju, koja je prije bila u statusu Prepared, administrator ručno rollback-irao sa ROLLBACK FORCE.

View DBA_2PC_NEIGHBORS prikazuje sve dolazne i odlazne in-doubt transakcije određene baze. Također pokazuje da li je lokalna baza commit point site baza (te transakcije). Prazan je ako nema in-doubt transakcija, a nakon što se in-doubt transakcije riješe, isto bi se trebao automatski isprazniti. View ima sljedeće stupce:

```
Name                                      Null?    Type
----------------------------------------- -------- ---------------
LOCAL_TRAN_ID                                      VARCHAR2(22)
IN_OUT                                             VARCHAR2(3)
DATABASE                                           VARCHAR2(128)
DBUSER_OWNER                                       VARCHAR2(30)
INTERFACE                                          VARCHAR2(1)
DBID                                               VARCHAR2(16)
SESS#                                              NUMBER(38)
BRANCH                                             VARCHAR2(128)
```

Sljedeća tablica (2.) pomaže u donošenju odluke - što napraviti kod ručnog rješavanja in-doubt distribuirane transakcije:

Tablica 2. Što napraviti kod ručnog rješavanja in-doubt transakcije; Izvor: [5]

STATE Column	State of Global Transaction	State of Local Transaction	Normal Action	Alternative Action
Collecting	Rolled back	Rolled back	None	PURGE_LOST_DB_ ENTRY (only if autorecovery cannot resolve transaction)
Committed	Committed	Committed	None	PURGE_LOST_DB_ ENTRY (only if autorecovery cannot resolve transaction)
Prepared	Unknown	Prepared	None	Force commit or rollback
Forced commit	Unknown	Committed	None	PURGE_LOST_DB_ ENTRY (only if autorecovery cannot resolve transaction)
Forced rollback	Unknown	Rolled back	None	PURGE_LOST_DB_ ENTRY (only if autorecovery cannot resolve transaction)
Forced commit	Mixed	Committed	Manually remove inconsistencies then use PURGE_ MIXED	-
Forced rollback	Mixed	Rolled back	Manually remove inconsistencies then use PURGE_ MIXED	-

Možda prethodna tablica izgleda kompleksno, no u praksi je rješavanje relativno jednostavno. Kao prvo, administrator baze bi trebao dobro poznavati cijelu distribuiranu bazu. Dalje, obično se problemi na vezama češće dešavaju sa određenim bazama. Na kraju, najvažnije je da administrator dobro pogleda stanja svih baza koje sadrže in-doubt transakciju (stupac STATE u view-u DBA_2PC_PENDING). Ako je bilo koja baza u stanju Commited, treba raditi COMMIT FORCE. Samo ako su sve baze u stanju Prepared (ili Collecting), treba raditi ROLLBACK FORCE (u našoj praksi je ROLLBACK FORCE puno češći nego COMMIT FORCE).

Kako je već rečeno, view-ovi DBA_2PC_PENDING i DBA_2PC_NEIGHBORS trebali bi se automatski isprazniti (od strane RECO pozadinskog procesa baze) nakon što se riješe in-doubt distribuirane transakcije. Iznimka nastaje kada RECO otkrije forsiranu transakciju koja je u nekonzistentnom stanju u odnosu na druge baze koje sudjeluju u transakciji. U tom slučaju, RECO ostavlja zapis u tablicama i postavlja MIXED stupac u DBA_2PC_PENDING na YES. Te retke ipak možemo brisati, pomoću DBMS_TRANSACTION.PURGE_MIXED procedure.

Ako automatsko rješavanje nije moguće zbog toga što je udaljena baza permanentno bila izgubljena, tada RECO ne može identificirati rekreiranu bazu zato što je ona dobila novi ID kod. U tom slučaju treba koristiti PURGE_LOST_DB_ENTRY proceduru (iz paketa DBMS_TRANSACTION) za brisanje redaka. No, budući da ti redovi ne drže nikakve resurse, nije važno odmah ih brisati – više smetaju kod gledanja ima li in-doubt transakcija.

Kako bi se moglo vježbati ručno rješavanje in-doubt transakcija, Oracle je omogućio da se simuliraju greške koje dovode do in-doubt transakcija. To se radi tako da se umjesto COMMIT naredbe upotrijebi naredba

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-n';
```

pri čemu *n* može imati vrijednost od 1 do 10, a ima efekt koji je prikazan u sljedećoj tablici (3.):

Tablica 3. Efekt vrijednosti *n* kod naredbe COMMIT COMMENT 'ORA-2PC-CRASH-TEST-*n*'; Izvor: [5]

n	Effect
1	Crash commit point after collect
2	Crash non-commit-point site after collect
3	Crash before prepare (non-commit-point site)
4	Crash after prepare (non-commit-point site)
5	Crash commit point site before commit
6	Crash commit point site after commit
7	Crash non-commit-point site before commit
8	Crash non-commit-point site after commit
9	Crash commit point site before forget
10	Crash non-commit-point site before forget

Kako bi se spriječilo RECO proces baze da na brzinu automatski riješi simulirani in-doubt problem, RECO se može isključiti, pomoću:

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

Ponovno uključivanje radi se sa:

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

ZAKLJUČAK

Ponekad nužno moramo raditi sa distribuiranom bazom podataka i distribuiranim transakcijama. Distribuirana transakcija koristi dvofazni commit protokol. Kod njega postoji period u kojem je transakcija osjetljiva na pad (nekog) servera baze ili veze između baza.

Greške koje se mogu desiti jesu:
- Padne računalo na kojem radi Oracle baza;
- Prekine se veza između dvije ili više baza koje sudjeluju u distribuiranoj transakciji;
- Desi se neki softverski problem.

U tom slučaju (ako se desi greška) transakcija postaje in-doubt. U fazi pripreme, baze stavljaju distribuirani lokot na sve modificirane tablice. Taj lokot sprečava čak i čitanje podataka! Ako to traje kratko, nije problem. No, ako transakcija postane in-doubt, može se desiti da duže vrijeme drži zaključane podatke (čak i za čitanje).

Najbolje je ostaviti da baza sama razriješi in-doubt transakciju. No, ako je zadovoljen jedan od ova dva uvjeta, trebali bismo ručno razriješiti in-doubt transakciju:

1. In-doubt transakcija je zaključala kritične podatke ili undo segmente;

2. Pad računala, prekid veze, ili softverski problem ne mogu se riješiti u kratkom vremenu.

Ručno rješavanje in-doubt transakcije može biti kompleksno. Glavni postupak sastoji se od sljedećih koraka:

1. Pronaći identifikacijski broj in-doubt transakcije;

2. Postaviti upit nad sistemskim view-ovima DBA_2PC_PENDING i DBA_2PC_NEIGHBORS, kako bismo odredili da li su baze koje sudjeluju u distribuiranoj transakciji commit-irale;

3. Ako je potrebno, forsirati commit sa COMMIT FORCE (ako je bilo koja baza u stanju Commited), odnosno rollback sa ROLLBACK FORCE (ako su sve baze u stanju Prepared ili Collecting).

LITERATURA

1. Date, C.J. (2004): An introduction to Database Systems (8.izdanje), Addison-Wesley
2. Kyte, T. (2009): Expert Oracle Database Architecture, Apress
3. Van Roy P., Haridi S. (2004): Concepts, Techniques, and Models of Computer Programming, The MIT Press Cambridge, Massachusetts

Oracle priručnici za bazu 11g Release 2 (2010.):

4. 2 Day + Data Replication and Integration Guide
5. Administrators Guide
6. Advanced Replication
7. Streams Replication Administrator's Guide