



Sistemi za upravljanje bazama podataka

Oporavak SQL Servera

Mentor:

doc. dr Aleksandar Stanimirović

Student:

Natalija Mitić, 1046

Sadržaj

Uvod.....	3
Oporavak baze podataka	3
Oporavak od pada transakcije	4
Oporavak od pada sistema	5
Poboljšanje procedure oporavka od pada sistema	8
SQL Server.....	9
Transakcioni log.....	10
Modeli oporavka	14
Write – ahead transakcioni log	15
Kontrolne tačke	17
Oporavak.....	19
Trajanje oporavka	21
Zaključak.....	21
Literatura.....	21

Uvod

Kod DBMS-a, kao i kod drugih sistema može doći do pada usled pada diska, gubitka napajanja, greške u softveru, sabotaze ili vremenskih nepogoda. Prilikom pada, može se desiti da podaci budu izgubljeni. DBMS mora biti u stanju da reaguje na padove, pomoću menadžera oporavka.

Oporavak baze podataka se odnosi na restaurisanje podataka baze podataka u slučaju da dođe do greške u transakciji, u sistemu ili u mediju na kome se nalazi baza podataka. Razmatraju se aktivnosti koje menadžer oporavka preduzima da bi oporavljeni sadržaj baze podataka zadovoljio, privremeno narušene, uslove integriteta baze podataka.

DBMS treba da obezbedi da se pri izvođenju transakcija, sve operacije uspešno izvedu i da efekat bude trajno zapisan u bazu podataka i da stanje baze bude konzistentno. Ne sme dozvoliti izvršenje samo nekih operacija. To znači da se, u slučaju pada transakcije, mora izvršiti oporavak.

U ovom radu je prikazan proces oporavka koji se koristi kod *SQL Servera* prilikom pada transakcija i sistema, kao i pomoćne strukture koje su potrebne za oporavak.

Oporavak baze podataka

Kod transakcija se ili sve radnje uspešno izvrše ili nijedna. Transakcija je osnovna jedinica oporavka, a zadatak menadžera oporavka je da garantuje dva od četiri ACID (atomičnost, konzistentnost, izolacija, postojanost) svojstva – atomičnost i postojanost. Svi efekti transakcija treba da budu trajno zabeleženi u bazi ili nijedan od njih, što znači da menadžer oporavka garantuje da promene od strane transakcija neće biti izgubljene i da transakcije neće biti prekinute.

Kod nemogućnosti uspešnog kompletiranja, poništava se efekat parcijalno izvršenih transakcija. Ako je transakcija uspešno izvršena, ali se desio pad sistema pre nego što su svi efekti upisani u bazu, potrebno je ponovo izvršiti neke delove. Prilikom pada sistema će se obaviti ili poništavanje ili ponovno izvršavanje. Ukoliko dođe do pada medija, potrebno je izvršiti prepisivanje arhivirane kopije baze na ispravan medij i eventualno ponoviti izvršenje transakcija kompletiranih posle poslednjeg arhiviranja, a pre pada medija.

Da bi se oporavak baze podataka mogao uspešno izvršiti, neophodno je da DBMS obezbedi neke dodatne podatke koji će neutralizovati prekinute ili pogrešne transakcije i ponovo uspostaviti bazu podataka nakon pada. Jedan skup takvih podataka se čuva u logu (žurnal datoteci). Ova datoteka pamti istoriju svake transakcije koja je menjala bazu podataka nakon stvaranja poslednje rezervne kopije. Drugi skup podataka se čuva u tzv. arhivskoj memoriji na drugom mediju u koju se povremeno prenosi sadržaj celokupne baze. Dakle, arhivska memorija sadrži rezervnu kopiju baze podataka. Log se koristi za otkaze koji fizički ne oštećuju memorijske jedinice (diskove) na kojima se čuva baza, dok arhivska memorija služi za oporavak posle ovakvih otkaza. Tipična strategija za oporavak baze podataka je:

- Ako su oštećene memorijske jedinice na kojima se čuva baza podataka (npr. pad glava diska), koristi se poslednja arhivska kopija cele baze za njen oporavak. Arhivirana kopija se prepisuje na ispravni medijum, a zatim se koriste informacije iz log datoteke za ponovno izvršenje transakcija, kompletiranih posle poslednjeg arhiviranja, a pre pada medija.
- Kada baza podataka nije fizički oštećena, oporavak se vrši korišćenjem loga, preko posebno definisanih protokola oporavka. U ovom slučaju DBMS treba da vodi računa o sledećim vrstama otkaza:
 1. Softverski ili hardverski otkaz koji ne oštećuje bazu, a u kome se može izgubiti sadržaj centralne memorije (npr. nestanak napajanja)
 2. Otkaz u samoj transakciji, na primer zbog deljenja sa nulom, davanja parametra pogrešnog tipa i slično
 3. Akcije sistema za konkurentnu obradu transakcija, ubijanje transakcije koja je izazvala mrtve petlje ili neserijabilnost izvršenja skupa transakcija i slično

Zadaci komponente za upravljanje oporavkom su:

- periodično prepisivanje cele baze podataka na medij za arhiviranje
- upisivanje slog promene u log datoteku pri svakoj promeni baze podataka
 - tip promene
 - nova i stara vrednost kod ažuriranja
 - nova vrednost pri unosu
 - stara vrednost pri brisanju

Oporavak se vrši u dva glavna koraka. Prvenstveno se preduzimaju akcije tokom normalnog funkcionisanja transakcija, kada treba obezbediti dovoljno informacija za eventualni kasniji oporavak. Zatim se, nakon pada, preduzimaju akcije za oporavak sadržaja baze podataka.

UNDO i REDO operacije

UNDO i REDO su neophodne procedure kojima se poništavaju, odnosno ponovo izvršavaju pojedinačne radnje transakcija kojima se menja baza podataka. Baza podataka se menja operacijama ažuriranja, unošenja ili brisanja podataka. UNDO logika se odnosi na poništavanje, a REDO na ponovno izvršavanje, što je moguće izvršiti na osnovu vrednosti zapamćenih u sistemskom logu. S obzirom na to da se pad sistema može desiti u fazi oporavka od prethodnog pada, može doći do ponovnog poništenja već poništenih operacija ili ponovnog izvršenja već izvršenih radnji. Zato za UNDO i REDO važi svojstvo idempotentnosti, odnosno $UNDO(UNDO(x)) \equiv UNDO(x)$ i $REDO(REDO(x)) \equiv REDO(x)$, za svaku radnju x .

Oporavak od pada transakcije

Izvršenje jedne transakcije može da se završi planirano ili neplanirano. Do planiranog završetka dolazi izvršenjem COMMIT naredbe (uspešno kompletiranje) ili eksplicitne ROLLBACK naredbe (greška za koju postoji programska provera, a program nastavlja sa radom). Neplanirani završetak izvršenja transakcije se dešava kada dođe do greške za koju ne postoji programska provera. To može biti usled pada računara, grešaka u sistemu ili transakciji, detektovanja greške u samoj transakciji, neispravnosti diska itd. Tada se vrši

implicitna (sistemska) ROLLBACK operacija, radnje transakcije se poništavaju i program prekida sa radom.

Izvršavanjem operacije BEGIN TRANSACTION, u log datoteku se upisuje slog početka transakcije. Kada se izvrši COMMIT naredba, efekti transakcije postaju trajni i ne mogu se poništiti procedurom oporavka. U log datoteku se tada upisuje COMMIT slog. Pošto izvršenje COMMIT naredbe ne podrazumeva upisivanje svih ažurnih podataka u bazu, već se čeka da se baferi napune, REDO logika garantuje upis podataka u nekom narednom trenutku.

Operacija ROLLBACK se sastoji od poništavanja učinjenih promena nad bazom. Oporavak se izvršava čitanjem unazad svih slogova iz log datoteke koji pripadaju toj transakciji, do BEGIN TRANSACTION sloga. Za svaki slog, promena se poništava primenom odgovarajuće UNDO operacije. Aktivnost oporavka od pada transakcije ne uključuje REDO logiku.

Oporavak od pada sistema

U slučaju pada sistema, sadržaj unutrašnje memorije je izgubljen. Ponovnim startovanjem sistema, za oporavak se koriste podaci iz log datoteke i poništavaju se efekti transakcija koje su bile u toku u trenutku pada sistema. Ove transakcije se mogu identifikovati čitanjem sistemskog loga unazad, kao transakcije za koje postoji BEGIN TRANSACTION slog, ali ne postoji COMMIT slog. S druge strane, uspele transakcije treba ponovo izvršiti.

Da bi se smanjila količina posla vezana za poništavanje transakcija, u pravilnim vremenskim intervalima, obično posle određenog broja upisanih slogova u log datoteku, se kreiraju tačke pamćenja. Podaci iz log bafera i bafera podataka se fizički upisuju u log datoteku i bazu podataka. U log datoteku se upisuje i slog tačke pamćenja koji sadrži informacije o svim aktivnim transakcijama u trenutku tačke pamćenja, adrese poslednjih slogova tih transakcija u log datoteci i druge informacije o bazi. Adresa sloga tačke pamćenja se upisuje u datoteku ponovnog startovanja (eng. restart file). Fizički upis podataka se vrši u tački pamćenja bez obzira da li su baferi puni ili ne, a upis ažurnih podataka uspešno kompletirane transakcije se garantuje tek u tački pamćenja. Prilikom upisivanja, upis COMMIT sloga i svih ažuriranja transakcije se vrše odvojeno. Protokol upisivanja u log unapred (eng. WAL – Write Ahead Log) obezbeđuje da se obe radnje izvrše. Prvo se odgovarajući slog fizički upisuje u log datoteku, a zatim se podaci upisuju iz bafera podataka u bazu. Na taj način se obezbeđuje da, ako dođe do pada sistema posle upisa COMMIT sloga u log datoteku, a pre upisa podataka iz bafera u bazu, podaci budu restaurirani iz sistemskog loga REDO logikom. Dakle, upis podataka u log slog, pre nego što se stranica sa podacima ažurira obezbeđuje atomičnost, dok forsiranje uspisa svih log slogova pre izvršenja COMMIT operacije obezbeđuje postojanost podataka.

Pri ponovnom startovanju sistema, posle pada, moguće je prema sadržaju log datoteke identifikovati neuspele transakcije (kandidate za poništavanje) i uspele transakcije (kandidate za ponovno izvršavanje). Oporavak baze podataka se tada vrši prema ARIES algoritmu. DBMS-ovi uglavnom koriste ili ovaj algoritam ili neke njegove modifikacije.

Algoritam:

BEGIN

naći slog poslednje tačke pamćenja u log datoteci (iz datoteke ponovnog startovanja);

IF u poslednjoj tački pamćenja nema aktivnih transakcija i slog tačke pamćenja je poslednji slog u log datoteci, oporavak je završen

ELSE

BEGIN

formirati dve prazne liste transakcija – uspele i neuspele;

sve transakcije aktivne u poslednjoj tački pamćenja staviti u listu neuspelih;

čitati redom log datoteku od tačke pamćenja do kraja;

kada se naiđe na slog BEGIN TRANSACTION, dodati transakciju listi neuspelih;

kada se naiđe na COMMIT slog, premestiti tu transakciju u listu uspelih;

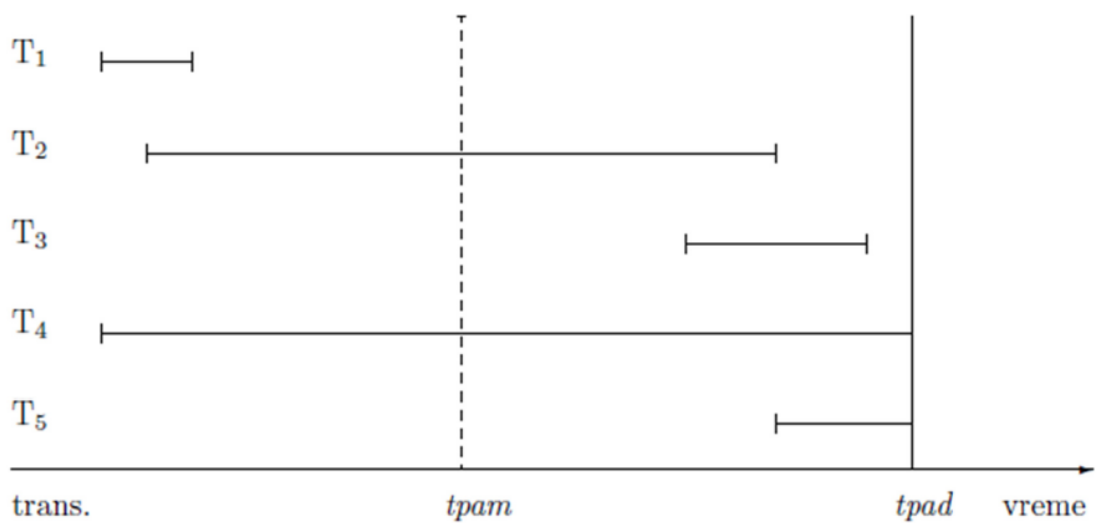
čitati unazad log datoteku i poništiti akcije i efekte neuspelih transakcija;

čitati i log datoteku od poslednje tačke pamćenja unapred i ponovo izvršiti uspele transakcije;

END

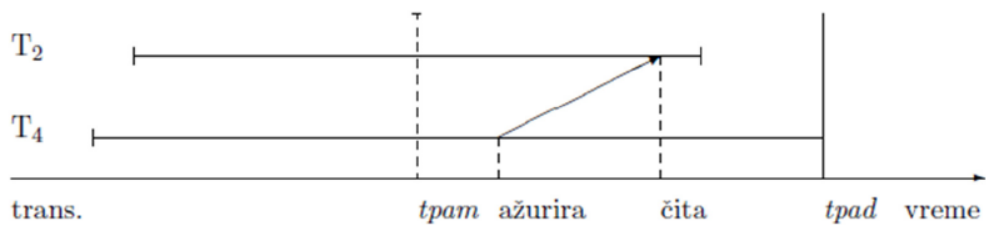
END

Na slici 1 je prikazan algoritam, ilustrovan skupom transakcija, gde transakcija T1 ne ulazi u proces oporavka, a njeni efekti su definitivno potvrđeni u trenutku *tpam*, zajedno sa registrovanjem nove tačke pamćenja. U tački pamćenja *tpam*, T2 i T4 se dodaju u listu neuspelih transakcija, jer su još uvek aktivne. Kada T3 počne sa izvršenjem i ona se pridodaje u tu listu, kao i kasnije T5. Pošto je T2 završena pre pada, prebacuje se u listu uspelih transakcija, a zatim i T3. Dakle, transakcije T4 i T5 se poništavaju, dok se transakcije T2 i T3 obnavljaju.

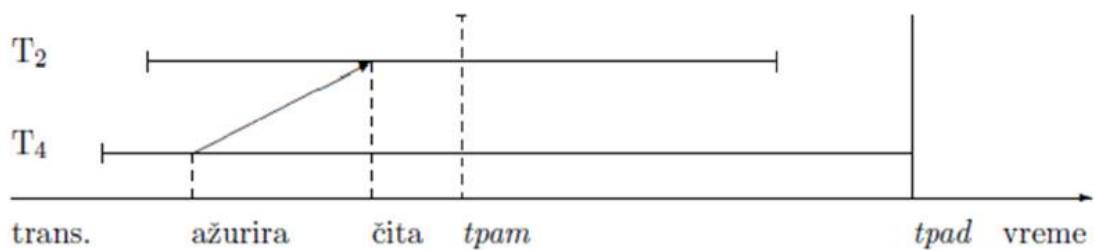


Slika 1 – Primer oporavka od pada sistema

Menadžer oporavka podrazumeva da transakcija oslobađa sva zaključavanja pri izvršenju COMMIT operacije, što je saglasno sa svojstvom izolacije. Ovo rešava problem zavisnosti od poništenog ažuriranja, bilo da se ono dogodilo pre ili posle tačke pamćenja. Dakle, izvršenja na slikama 2 i 3 nisu moguća.



Slika 2 – Zavisnost od poništenog ažuriranja posle tačke pamćenja



Slika 3 – Zavisnost od poništenog ažuriranja pre tačke pamćenja

Poboljšanje procedure oporavka od pada sistema

Prethodno opisan protokol poništava efekte neuspelih transakcija koji možda nisu ni bili ubeleženi u bazu. Takođe, ponovo se izvršavaju radnje uspelih transakcija od tačke pamćenja, čak i ako su efekti tih radnji upisani u bazu pre pada sistema.

Postoji niz poboljšanja postupka oporavka od pada sistema. Moguće je sva upisivanja jedne transakcije u bazu ostaviti za trenutak izvršenja COMMIT operacije te transakcije. Tako se eliminiše potreba za UNDO logikom. Takođe je moguće oslabiti zahtev za izolacijom transakcije, tj. zahtev za dvofaznošću, što nosi opasnost od nelinearizovanog izvršenja.

Jedno od poboljšanja protokola se odnosi na aktivnosti vezane za tačku pamćenja. Moguće je eliminisati fizičko upisivanje bafera podataka u bazu podataka u tački pamćenja. Onda se u fazi oporavka od pada sistema poništavaju samo one radnje neuspelih transakcija čiji su efekti upisani u bazu. Ponovo se izvršavaju samo one radnje uspelih transakcija čiji efekti nisu upisani u bazu. Uvodi se, u vreme upisa sloga u log, jedinstvena oznaka sloga – serijski broj loga (eng. LSN – Log Sequence Number). Ovi brojevi su u rastućem poretku. Uvek kada se ažurirana stranica fizički upisuje u bazu podataka, u stranicu se upisuje i LSN sloga sistemskog loga koji odgovara tom ažuriranju. U sistemskom logu može biti više slogova koji odgovaraju ažuriranju iste stranice baze podataka. Ako je serijski broj upisan u stranicu P veći ili jednak serijskom broju sloga loga, efekat ažuriranja kome odgovara taj slog fizički je upisan u bazu, a ako je manji, efekat nije upisan. Da bi se slog iz baze podataka ažurirao, potrebno je pročitati stranicu iz baze na kojoj se on nalazi. Posle ažuriranja sloga, stranica je (u baferu podataka) spremna za upis u bazu i i dalje nosi serijski broj svog prethodnog upisa u bazu. Pri registrovanju tačke pamćenja se eliminiše fizički upis bafera podataka u bazu, dodaje se upis, u slog tačke pamćenja, vrednosti LSN m najstarije stranice bafera podataka (najmanjeg LSN stranica iz bafera podataka, koje su ažurirane ali još nisu upisane u bazu).

Slog sistemskog loga sa serijskim brojem m odgovara tački u sistemskom logu od koje treba pri oporavku od pada sistema, ponovo izvršavati uspele transakcije. Tačka m prethodi tački pamćenja. Može se desiti da neka transakcija koja je uspešno kompletirana pre tačke pamćenja bude uključena u skup aktivnih (uspelih) transakcija. Na takvu transakciju primeniće se procedura oporavka, što se ne bi dogodilo u prethodnoj varijanti oporavka. To je cena smanjenog broja poništavanja, ponovnog izvršavanja i fizičkog upisa koje obezbeđuje ovaj postupak.

Poboljšani algoritam:

BEGIN

formirati dve prazne liste transakcija – uspele i neuspele;

sve transakcije aktivne u tački m staviti u listu neuspelih;

čitati redom log datoteku od tačke m do kraja;

kada se naiđe na slog BEGIN TRANSACTION, dodati transakciju listi neuspelih;

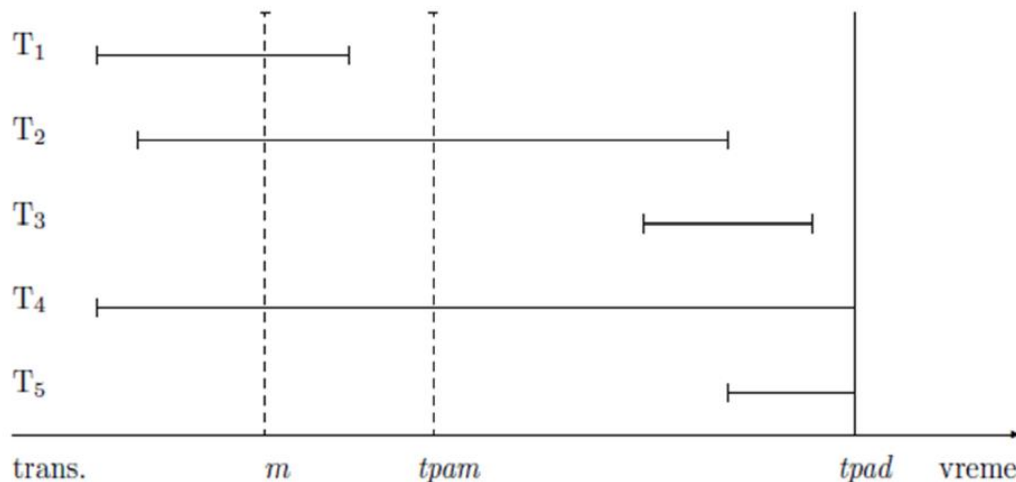
kada se naiđe na COMMIT slog, premestiti tu transakciju u listu uspehlih;

čitati unazad log datoteku i poništiti efekte onih radnji neuspehlih transakcija za koje je $p \geq r$, gde je r LSN tekućeg sloga, a p LSN odgovarajuće stranice baze podataka;

čitati i log datoteku od tačke m unapred i ponovo izvršiti one radnje uspehlih transakcija za koje je $p < r$ (p, r – kao u prethodnom koraku);

END

Procedura oporavka od pada sistema je prikazana na slici 4. U ovom slučaju i transakcija T_1 se svrstava u skup transakcija nad kojima se vrši oporavak, jer je ona bila aktivna u tački m .



Slika 4 – Primer oporavka (poboljšani algoritam)

SQL Server

Proces oporavka baze podataka kod *SQL servera* je zasnovan na algoritmu za oporavak ARIES. Podaci i log se čuvaju odvojeno, koristeći WAL algoritam za beleženje svih operacija u log transakcija pre nego što se izvrše bilo kakve izmene nad odgovarajućim stranicama podataka.

Svaka baza podataka u *SQL Serveru* ima transakcioni log koji beleži sve transakcije i modifikacije baze podataka izvršene od strane svake transakcije. Transakcioni log je kritična komponenta baze podataka. Ako dođe do kvara sistema, on je potreban kako bi baza podataka bila vraćena u ponovo konzistentno stanje.

Pored transakcionog loga, i kontrolne tačke imaju ulogu kod oporavka. *SQL Server* periodično kreira kontrolne tačke koje služe da se od poslednje ovakve tačke može početi primena promena sadržanih u transakcionom logu tokom oporavka.

Transakcioni log

Transakcioni log je fajl koji je sastavni deo svake baze podataka *SQL Servera*. Sadrži log zapise nastale tokom procesa evidentiranja u bazi podataka. Transakcioni log je najvažnija komponenta baze podataka *SQL Servera* kada je u pitanju oporavak od katastrofe, međutim, mora biti nekorumpiran. Nakon svake izmene baze podataka (pojave transakcija), u transakcioni log se upisuje zapis. Sve promene se pišu sekvencijalno.

Održavanje loga je korisno iz više razloga:

- Osigurava da će se, ako se dogodi pad, izvršena transakcija pravilno odraziti u bazi podataka nakon pada
- Obezbeđuje da će se neobrađena transakcija ispravno vratiti i da se neće odraziti u bazi podataka posle pada
- Obezbeđuje mogućnost otkazivanja *in-flight* transakcija i povraćaj svih njenih operacija
- Omogućuje kreiranje sigurnosne kopije loga transakcija tako da se baza podataka može vratiti na osnovu nje
- Podržava funkcije koje se oslanjaju na čitanje loga transakcija, kao što su replikacija, *mirroring* baze podataka i prikupljanje promena podataka

Transakcioni log omogućava oporavak pojedinačnih transakcija, kao i oporavak svih nepotpunih transakcija nakon što se *SQL Server* ponovo pokrene.

Oporavak pojedinačnih transakcija

Ako aplikacija obavlja ROLLBACK operaciju ili ako *Database Engine* prepozna grešku, kao što je gubitak komunikacije sa klijentom, log zapisi se koriste da bi se vratile modifikacije napravljene od strane nekompletne transakcije. Na taj način se obezbeđuje da transakcija ne menja bazu podataka.

Oporavak svih nepotpunih transakcija kada se pokrene SQL Server

Ako server doživi pad, baze podataka mogu se ostaviti u stanju u kojem neke izmene nikada nisu zapisane iz bafera u fajlove podataka, a može doći do nekih izmena načinjenih od strane nepotpunih transakcija u fajlovima podataka. Kada se pokrene instanca *SQL Servera*, ona pokreće oporavak svake baze podataka. Svaka izmena zabeležena u logu koja možda nije zapisana u fajlovima podataka, preusmerava se unapred kako bi se osigurao upis. Svaka nepotpuna transakcija pronađena u logu transakcija vraća se nazad da bi se osiguralo očuvanje integriteta baze podataka.

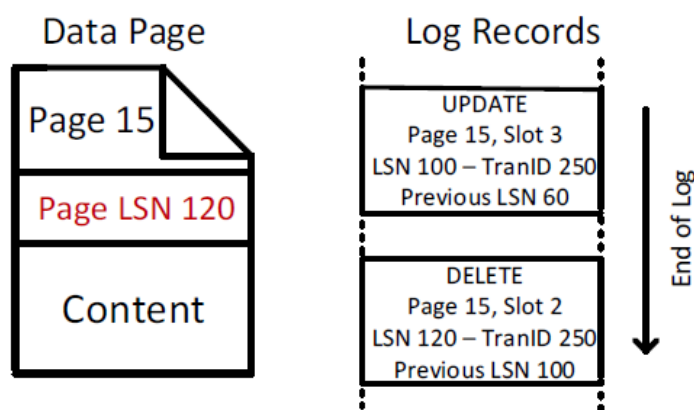
Karakteristike transakcionog loga

Transakcioni log se implementira kao zasebna datoteka ili skup datoteka u bazi podataka. Kešom loga upravlja se odvojeno od keša bafera stranica s podacima, što rezultuje jednostavnim, brzim i robusnim kodom u *SQL Server Database Engine-u*. Format zapisa i stranica loga nije ograničen da prati format stranica sa podacima. Transakcioni log se može implementirati u nekoliko fajlova. Fajlovi se mogu definisati da se automatski prošire,

ukoliko je potrebno. To smanjuje potencijalni gubitak prostora u transakcionom logu, a istovremeno smanjuje administrativne troškove. Mehanizam za ponovnu upotrebu prostora unutar log fajlova je brz i ima minimalan uticaj na protok transakcija.

Transakcioni log pamti svaku transakciju izvršenu nad bazom podataka, osim onih koje se minimalno evidentiraju kao npr. BULK IMPORT ili SELECT INTO. Svi log zapisi su zapisani u log transakcija jedinstveno su identifikovani pomoću njihovog serijskog broja (LSN) koji je inkrementalni identifikator na osnovu fizičke lokacije log zapisa u fajlu loga. Osim detalja o izvršenoj operaciji (npr. brisanju ključa), svaki log zapis, takođe, sadrži podatke o modifikovanoj stranici (ID stranice), transakciji koja je izvršila operaciju (ID transakcije) i LSN prethodnog log zapisa (prethodni LSN) za istu transakciju.

Stranica sa podacima, između ostalog, pamti i LSN log zapisa koji odgovara poslednjoj modifikaciji te stranice. Na slici 5 je dat primer stranice sa podacima i odgovarajućeg log zapisa.



Slika 5 – Primer stranice sa podacima i log zapisa

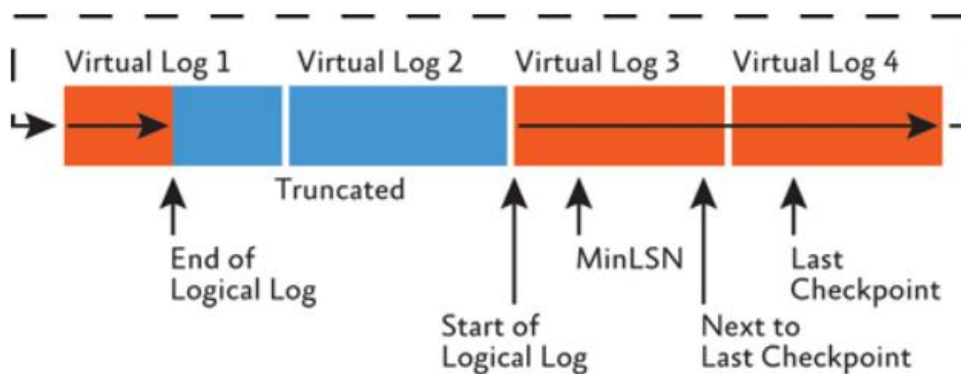
Log je interno podeljen na manje delove – virtualne log fajlove (Virtual Log Files - VLFs). Kad jedan VLF postane pun, zapisivanje se nastavlja s upisom u sledeći VLF dostupan u transakcionom logu. Transakcioni log fajl se može predstaviti u obliku kružne datoteke. Kada evidentiranje dođe do kraja fajla, započinje iz početka, ali samo ako su svi zahtevi ispunjeni i neaktivni delovi su odsečeni, odnosno spremni za ponovnu upotrebu. Da bi se VLF označio kao neaktivni, on ne treba da sadrži nijedan aktivni log povezan sa otvorenom transakcijom, jer se ovi zapisi mogu koristiti za postupak oporavka. Odsecanje loga ne znači da će se veličina loga transakcija smanjiti, samo će taj deo biti označen kao neaktivan da bi se koristio u budućnosti. Proces odsecanja je neophodan za obeležavanje svih neaktivnih delova kako bi se ponovo mogli koristiti i prebrisati.

Zapis u transakcionom logu nije više potreban ako su tačne sve sledeće tvrdnje:

- Transakcija čiji je on deo je izvršila COMMIT operaciju
- Sve stranice baze podataka koje su promenjene su zapisane na disk u tački pamćenja
- Zapis loga nije potreban za izradu sigurnosnih kopija

- Zapis loga nije potreban ni za jednu funkciju koja čita log (kao što je *mirroring* baze podataka ili replikacija)

Logički log je aktivni deo loga transakcija. Redni broj loga (LSN) identifikuje svaku transakciju u logu transakcija. MinLSN je početna tačka najstarije aktivne transakcije u logu transakcija. Log s najnižim LSN-om (MinLSN) određuje početak aktivnog loga koji će možda biti potreban za bilo koju aktivnost baze podataka, a log s najvišim LSN-om određuje kraj tog aktivnog loga. Bilo koji log transakcija sa LSN vrednošću manjom od najmanje LSN smatraće se neaktivnim logom transakcija. Na slici 6 je prikazan log kao kružna datoteka, podeljen na virtualne logove.



Slika 6 – Transakcioni log

Nije moguće funkcionisanje bez transakcionog loga zbog dizajna *SQL Servera* i usklađenosti sa ACID-om. Sve transakcije moraju ispuniti karakteristike koje zadovoljavaju ACID svojstva i njihovo stanje se, zato, mora pratiti.

Postojanje više transakcionih logova je moguće, ali se preporučuje samo u specifičnim situacijama. Dodavanje više transakcionih log fajlova ni na koji način neće poboljšati performanse baze podataka *SQL Servera*. Upisivanje se može izvršiti u samo jednu datoteku, pa paralelne I / O operacije nisu moguće. Postojanje više transakcionih log fajlova se preporučuje samo ako je prvi pun ili na disku nema dovoljno prostora. Svakako, ove probleme treba rešiti ranije i rešavati ih kreiranjem sigurnosnih kopija transakcionih logova i nadgledanjem raspoloživog prostora na disku.

Rast loga

Kao što je već spomenuto, transakcioni log beleži sve izmene baze podataka. Ako je sistem previše zauzet, to samo može prouzrokovati rast transakcionog loga. Ako automatski rast nije omogućen, jednom kada log fajl dostigne navedenu maksimalnu veličinu, prouzrokuje grešku za svaku transakciju koja pogodi bazu podataka sve dok se problem ne reši. Uvek se preporučuje uključivanje funkcije *Autogrow*. Određivanje malog priraštaja rasta log fajla rezultuje većim brojem VLF-ova, što može umanjiti performanse. Povećanje rasta log fajla bi trebalo da bude dovoljno veliko da se izbegne učestalo proširenje.

Faktori koji će uzrokovati rast log fajla mogu uključivati sledeće:

- Neizvršene transakcije
- Operacije indeksa - CREATE INDEX, obnova indeksa itd.
- Nereplikovane transakcije
- Dugotrajne transakcije
- Netačne postavke modela oporavka
- Velike transakcije

Održavanje loga

Održavanje transakcionog loga je važan zadatak u administraciji *SQL Servera*. Nadgledanje se preporučuje svakodnevno ili još češće ako baza podataka ima veliku količinu saobraćaja. Prostor loga transakcija se može nadzirati pomoću naredbe DBCC SQLPREF, kao što je prikazano na slici 7.

	Database Name	Log Size (MB)	Log Space Used (%)	Status
1	master	1.992188	29.41176	0
2	tempdb	0.484375	60.58468	0
3	model	7.992188	6.95259	0
4	msdb	0.7421875	73.94736	0
5	Recipes	7.992188	9.897361	0
6	Biblioteka	135.9922	2.356092	0
7	BibliotekaSQL	7.992188	14.96212	0
8	Ip1	7.992188	10.74658	0
9	SeoskiTurizam-IV4	7.992188	12.20063	0
10	proba	7.992188	7.502444	0

Slika 7 – Logovi različitih baza podataka

Prikazuju se parametri:

- Database Name - Naziv baze podataka za prikazane statistike loga
- Log size (MB) - Trenutna veličina prostora dodeljena logu. Ova vrednost je uvek manja od iznosa prvobitno dodeljenog logu, jer Database Engine zadržava malu količinu prostora na disku za interne heder informacije
- Log Space Used (%) - procenat log fajla trenutno zauzetog sa informacijama transakcionog loga
- Status - Status log fajla

Za transakcioni log treba redovno praviti rezervne kopije kako bi se izbegla operacija automatskog rasta i popunjavanje log fajla. Prostor u logu transakcija može biti odsečen (očišćen) putem izvršenja kreiranja sigurnosne kopije loga, kao na slici 8.


```
BACKUP LOG ACMEDB  
TO DISK = 'C:\ACMEDB.TRN'  
GO
```

Slika 8 – Kreiranje sigurnosne kopije loga

Važno je čuvati rezervne kopije transakcionih logova, jer je to jedan od najvažnijih resursa kada je u pitanju oporavak od katastrofe. Oni nisu potrebni (i dostupni) samo ako se koristi model oporavka *Simple*, ali postoji izloženost gubitku podataka. Većina administratora baza podataka koristi interval za pamćenje od 15 minuta ili čak manje za baze podataka sa velikim saobraćajem. Sigurnosne kopije transakcionih logova su važne, jer se prilikom snimanja označavaju neaktivni VLF-ovi koji se mogu koristiti za upis novih transakcija.

Podaci o VLF-ovima se mogu nadgledati izvršenjem naredbe DBCC LOGINFO (slika 9). Prikazuju se parametri:

- FieldId – Sve vrednosti su 2, što znači da postoji jedan fizički fajl
- FileSize – Veličina VLF-a
- Start Offset – Ofset od početka fajla (Prvi VLF je veličine jedne stranice i sadrži heder informacije)
- FseqNo – Definiše logički redosled korišćenja VLF-ova
- Status – Označava da li se VLF koristi (2) ili ne (0)
- Parity – Parnost koja se interno koristi da se odredi kraj loga u VLF-u
- CreateLSN – Trenutni LSN kada je VLF dodat. Vrednost 0 označava da je VLF dodat pri kreiranju loga



	RecoveryUnitId	Field	FileSize	StartOffset	FSeqNo	Status	Parity	CreateLSN
1	0	2	253952	8192	36	0	64	0
2	0	2	253952	262144	37	2	64	0
3	0	2	253952	516096	0	0	0	0
4	0	2	278528	770048	0	0	0	0

Slika 9 – Podaci o virtualnim logovima

Modeli oporavka

Model oporavka je svojstvo baze podataka koje definiše kako *SQL Server Engine* tretira transakcione logove baze podataka, uključujući

- određivanje kako će se transakcije beležiti u log i sačuvati u log fajl
- vrste *backup* i *restore* operacija koje se mogu izvoditi u bazi podataka
- rešenja za veliku dostupnost ili oporavak od katastrofe koja su podržana u bazi podataka
- tačku oporavka u koju se može vratiti baza podataka

Postoje tri modela oporavka kod *SQL Servera*: *Full*, *Bulk-logged* i *Simple*.

Model oporavka *Simple*

Kod ovog modela se log pamti u log fajlu jako kratko – samo dok je transakcija aktivna. Ovaj model ne zahteva kreiranje sigurnosnih kopija transakcionog loga, jer *SQL Server* obeležava VLF-ove kao neaktivne u trenutku kontrolne tačke. Nakon što se log zapis prebaci iz bafera u log fajl na disku, log se odseca kada se u kontrolnoj tački izvodi COMMIT

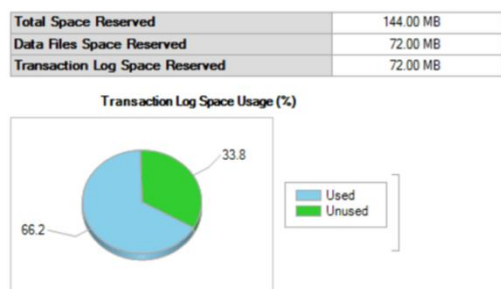
transakcije. Ako ne postoje dugačke transakcije, log neće puno rasti. Kod ovog modela se ne može izvesti oporavak u tački u vremenu. Podaci se mogu vratiti jedino iz prethodno kreiranih sigurnosnih kopija, što znači da je verovatnoća gubitka podataka veća.

Model oporavka Bulk-logged

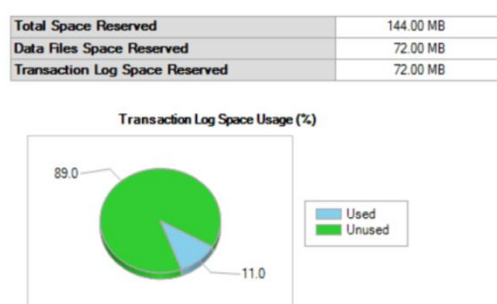
Za neke operacije je potrebno samo minimalno zapisivanje. To znači da se ne beleži svaka transakcija pojedinačno. Pamte se samo informacije potrebne za oporavak transakcija, bez mogućnosti oporavka u tački u vremenu. *SQL Server* obeležava promenjene opsege preko BCM (Bulk Changed Map) stranice. Ovaj model obezbeđuje najbolje performanse i najmanje korišćenje prostora tokom izvođenja velikih *bulk* operacija.

Model oporavka Full

Kod ovog modela *SQL Server Engine* drži transakcione logove u log fajlu kao neaktivne, nakon što izvrši transakciju i izvede operaciju kontrolne tačke. Log neće biti odsečen dok se ne izvrši sigurnosna kopija loga. Dakle, ovaj model podrazumeva da treba izvoditi redovne sigurnosne kopije loga transakcija. Nakon sigurnosnog kopiranja loga *SQL Server* može označiti VLF-ove kao neaktivne i može ih prepisati kasnije. Ako se ne izvede redovne sigurnosne kopije loga transakcija, *SQL Server* ne može da prepíše VLF-ove, jer su svi aktivni, pa će transakcionog log rasti. Rast loga je spor i dovodi do fragmentacije loga (ogromna količina različitih VLF-ova). Još jedna prednost redovnih sigurnosnih kopija je činjenica da je moguće izvršiti oporavak u tački u vremenu, gde se baza podataka nalazila u određeno vreme – vreme kontrolne tačke. Sa odgovarajućom strategijom kreiranja sigurnosnih kopija, ovaj model minimizuje ili čak potpuno sprečava gubljenje podataka. Na slici 10 je prikazano stanje loga nakon izvršenja nekoliko transakcija, dok je na slici 11 stanje nakon kreiranja sigurnosne kopije loga.



Slika 10 - Log pre sigurnosne kopije



Slika 11 – Log posle sigurnosne kopije

Write – ahead transakcioni log

SQL Server koristi algoritam za upisivanje u log unapred (WAL), koji garantuje da se na disk ne zapisuju nikakve izmene podataka pre nego što se pridruženi log zapis ne upiše na disk, kako bi se osiguralo pravilno izvršenje transakcije *SQL Server* održava bafer keš u koji čita stranice podataka kada podaci moraju biti pronađeni. Kada se stranica modifikuje u bafer kešu, ona se ne vraća odmah na disk. Umesto toga, stranica je označena kao prljava (eng. dirty). Stranica podataka može imati više logičkih upisa pre nego što bude fizički zapisana na

disk. Za svaki logički upis, zapis u transakcionom logu je ubačen u bafer keš koji beleži modifikaciju. Log zapis se mora upisati na disk pre nego što pridružena prljava stranica bude uklonjena iz bafer keša i zapisana na disk. Proces kontrolne tačke periodično skenira kako bi pronašao bafere sa stranicama iz određene baze podataka i upisuje sve prljave stranice na disk. Kontrolne tačke štede vreme tokom kasnijeg oporavka tako što stvaraju tačku u kojoj je zagarantovano da su sve prljave stranice na zapisane na disk. *SQL Server* ima logiku koja sprečava prebacivanje (eng. flushing) prljave stranice pre nego što je odgovarajući log zapisan. Zapisi logova se zapisuju na disk kada se baferi prebacuju. To se događa kad god se neka transakcija izvrši ili log baferi postanu puni.

Na primer neka se izvrši transakcija:

```
BEGIN TRANSACTION  
INSERT INTO tbl VALUES (50)  
COMMIT TRANSACTION
```

SQL Server obavlja sledeće aktivnosti kod svake naredbe (neka je broj stranice koja je pogođena 320):

- **BEGIN TRANSACTION** - Zapisuje se u log keš. Međutim, nije potrebno prebacivanje na stabilnu memoriju jer *SQL Server* nije izvršio nikakve fizičke promene.
- **INSERT INTO**
 - Stranica podataka sa brojem 320 preuzima se u keš, ako već nije dostupna
 - Stranica je zaključana i obeležena i obezbeđuju se odgovarajući katanci
 - Insert log zapis se kreira i dodaje u log keš
 - Novi red se dodaje stranici sa podacima
 - Stranica se otključava
 - Log zapisi povezani sa transakcijom ili stranicom u ovom trenutku ne moraju biti obrisani, jer sve promene ostaju u nestabilnom skladištu
- **COMMIT TRANSACTION**
 - Formira se zapis commit loga i log zapisi povezani sa transakcijom moraju biti zapisani u stabilnu memoriju. Transakcija se ne smatra izvršenom dok log zapisi nisu ispravno dodeljeni stabilnoj memoriji.
 - Stranica sa podacima sa brojem 320 ostaje u kešu podataka i ne prebacuje se odmah u stabilnu memoriju. Kad su log zapisi ispravno osigurani, oporavak može ponoviti operaciju, ako je neophodno.
 - Katanci koje drži transakcija se oslobađaju.

Na slici 12 su prikazani log zapisi koji se kreiraju kada se izvrši prethodna transakcija (u ovom slučaju je korišćena i kontrolna tačka).

Operation	Context	Page ID	Slot ID	Locks	Lock Information
1 LOP_BEGIN_XACT	LCX_NULL	NULL	NULL	NULL	NULL
2 LOP_INSERT_ROWS	LCX_HEAP	0001:00000140	1	3	HoBt 72057594041270272.ACQUIRE_LOCK_IX OBJECT: 12:565577053:0 ,ACQUIRE_LOCK_IX PAGE: 12:1:320 ,ACQUIRE_LOCK_X RID: 12:1:320: 1
3 LOP_COUNT_DELTA	LCX_CLUSTERED	0001:0000008f	90	NULL	NULL
4 LOP_COUNT_DELTA	LCX_CLUSTERED	0001:000000a1	20	NULL	NULL
5 LOP_COUNT_DELTA	LCX_CLUSTERED	0001:00000047	73	NULL	NULL
6 LOP_BEGIN_CKPT	LCX_NULL	NULL	NULL	NULL	NULL
7 LOP_XACT_CKPT	LCX_NULL	NULL	NULL	NULL	NULL
8 LOP_XACT_CKPT	LCX_BOOT_PAGE_CKPT	NULL	NULL	NULL	NULL
9 LOP_END_CKPT	LCX_NULL	NULL	NULL	NULL	NULL
10 LOP_COMMIT_XACT	LCX_NULL	NULL	NULL	NULL	NULL

Slika 12 – Log zapisi prilikom izvršenja transakcije

Kontrolne tačke

Zbog performansi, stranice podataka se keširaju u memoriju u *Buffer Pool*. Transakcije ažuriraju podatke samo u memoriji (kreirajući ili modifikujući prljave stranice), a proces pozadinske kontrolne tačke je odgovoran za periodično upisivanje svih prljavih stranica na disk, nakon što se garantuje da su odgovarajući log zapisi takođe upisani na disk da bi se sačuvala semantika WAL-a. To umanjuje aktivni deo loga koji se mora obraditi tokom potpunog oporavka baze podataka.

Kontrolna tačka izvodi sledeće procese u bazi podataka:

- U log upisuje zapis koji označava početak kontrolne tačke
- Pamti informacije, snimljene za kontrolnu tačku, u lancu zapisa logova kontrolnih tačaka.

Jedan deo podataka zabeležen u kontrolnoj tački je LSN prvog zapisa loga koji mora biti prisutan za uspešno vraćanje cele baze podataka (MinLSN). Zapisi o kontrolnim tačkama takođe sadrže listu svih aktivnih transakcija koje su promenile bazu podataka.

- Ako baza podataka koristi model oporavka *Simple*, označava mesto za ponovnu upotrebu prostora koji prethodi MinLSN-u
- Zapisuje sve prljave stranice i log na disk
- U log fajl upisuje zapis koji označava kraj kontrolne tačke
- Zapisuje LSN početka ovog lanca na stranicu za pokretanje baze podataka

Kontrolne tačke se registruju u sledećim situacijama:

- Naredba CHECKPOINT je eksplicitno izvršena
- U bazi se vrši operacija sa minimalnim logovanjem - na primer bulk-copy operacija
- Fajlovi baza podataka su dodati ili uklonjeni korišćenjem ALTER DATABASE
- Instanca *SQL Servera* je zaustavljena naredbom SHUTDOWN ili zaustavljanjem usluge *SQL Servera*. Bilo koja radnja uzrokuje kontrolnu tačku u svakoj bazi podataka u instanci *SQL Servera*
- Instanca *SQL Servera* periodično generiše automatske kontrolne tačke u svakoj bazi podataka da umanje vreme koje će instanci trebati za oporavak baze podataka
- Preuzeta je sigurnosna kopija baze podataka

Dakle, prvi korak je čuvanje log zapisa u baferu i zatim se oni prebacuju na disk u jednoj od sledećih situacija:

- Kada se izvrši commit/rollback transakcije
- Kada log blok dostigne maksimalnu veličinu od 60 KB
- Kada se stranica podataka upisuje na disk – svi log zapisi, uključujući i poslednji koji utiče na ovu stranicu, moraju se upisati na disk bez obzira na to koje transakcije su deo

Prljave stranice su povezane sa transakcionim logom kroz heder stranice. Svaki heder stranice sadrži min_lsn broj koji odražava najnoviji LSN gde podaci te stranice postoje u logu. To daje broj koji određuje koje su stranice zapisane u fajlove podataka i kao takav predstavlja minimalni LSN transakcionog loga koji mora biti aktivan da bi oporavak baze podataka bio omogućen.

Proces kontrolne tačke traži prljave stranice u *buffer pool-u* (svako zaglavlje stranice sadrži minLSN) i ako stranica ima minLSN koji je veći od LSN-a poslednje kontrolne tačke (sačuvana u stranici za pokretanje baze podataka), tada bi stranica trebalo da bude zapisana u fajl i označena kao „čista“. Cilj je smanjiti vreme potrebno za slučaj kada je potreban oporavak - što je novija kontrolna tačka, manje je operacija u logu koje treba izvršiti da bi se kretalo unapred ili vratilo unazad.

Kontrolne tačke se mogu kreirati ručno pomoću naredbe CHECKPOINT ili korišćenjem automatskih kontrolnih tačaka. Automatske kontrolne tačke se automatski kreiraju u pozadini kada broj log zapisa u baferu dostigne procenjeni broj zapisa koje *SQL Server Database Engine* može obraditi u vremenu koje konfiguriše server – interval oporavka. Zato će se kontrolne tačke kreirati češće ako se vrši mnogo modifikacija. Interval između dve tačke zavisi i od modela oporavka. Kod *full* i *bulk-logged* modela, kontrolna tačka se kreira kada broj log zapisa dostigne procenjeni broj zapisa. Kod *Simple* modela kontrolna tačka se kreira ili kada log postane 70% pun ili kada broj log zapisa dostigne procenjeni broj zapisa.

Ako se kreira transakcija i izvrši neka naredba (u ovom slučaju je ubačen 1 red u tabelu), a zatim se izvrši kontrolna tačka pre komitovanja transakcije, log će izgledati kao na slici 13. Dakle, na početku je zapis o započetoj transakciji, zatim u ubacivanju reda, nakon čega sledi ačuriranje broja redova u meta podacima. Nakon toga sledi zapis o početku kontrolne tačke. LOP_XACT_CKPT je generisan zato što postoje aktivne transakcije koji izlistava informacije o tim transakcijama kada se kreira kontrolna tačka, što se koristi prilikom oporavka da bi se znalo od kog dela loga treba čitati informacije. Na kraju se nalazi zapis o kraju kontrolne tačke.

	Current LSN	Operation
1	00000022:00000181:0001	LOP_BEGIN_XACT
2	00000022:00000181:0002	LOP_INSERT_ROWS
3	00000022:00000181:0003	LOP_COUNT_DELTA
4	00000022:00000181:0004	LOP_COUNT_DELTA
5	00000022:00000181:0005	LOP_COUNT_DELTA
6	00000022:00000181:0006	LOP_BEGIN_CKPT
7	00000022:00000184:0001	LOP_XACT_CKPT
8	00000022:00000184:0002	LOP_XACT_CKPT
9	00000022:00000185:0001	LOP_END_CKPT

Slika 13 – Log zapisi kontrone tačke

Oporavak

Oporavak baze podataka je proces koji upotrebljava *SQL Server* kako bi svaka baza podataka započela u transakciono konzistentnom, odnosno čistom stanju. U slučaju greške ili drugog nečistog isključivanja, baze podataka mogu biti ostavljene u stanju u kojem neke modifikacije nikada nisu zapisane iz bafera podataka u fajlove podataka, a može doći do toga da su neke modifikacije nepotpunih transakcija ostale zapisane u fajlovima podataka. Kada se pokrene instanca *SQL Servera*, ona pokreće oporavak svake baze podataka, koja se sastoji od tri faze, na osnovu poslednje kontrolne tačke baze podataka: Faza analize, REDO faza i UNDO faza.

Faza analize

Analizira se transakcioni log kako bi se utvrdilo koja je poslednja kontrolna tačka i kreira se tabela prljavih stranica (Dirty Page Table - DPT) i tabela aktivnih transakcija (Active Transaction Table - ATT). DPT sadrži zapise stranica koje su bile prljave u vreme gašenja baze podataka. ATT sadrži zapise o transakcijama koje su bile aktivne u vreme kada baza podataka nije bila čisto ugašena.

Pošto je postupak kontrolne tačke obuhvatio sve aktivne transakcije i LSN najstarije prljave stranice u vreme kontrolne tačke, faza analize može skenirati log počevši od minimuma od početka poslednje dovršene kontrolne tačke (Checkpoint Begin LSN) i LSN-a najstarije prljave stranice kako bi se rekonstruisali potrebni podaci.

REDO faza

U ovoj fazi se izvršava ponovo svaka promena koja je zabeležena u logu koja možda nije bila zapisana u fajlove podataka u vreme kad je baza podataka bila isključena. Najmanji sekvencijalni broj loga (minLSN), potreban za uspešan oporavak u bazi podataka, nalazi se u DPT-u i označava početak ponovljenih operacija potrebnih nad svim prljavim stranicama. U ovoj fazi, *SQL Server Database Engine* upisuje na disk sve prljave stranice koje pripadaju izvršenim transakcijama.

Pošto je faza analize preračunala LSN najstarije prljave stranice, REDO bi trebalo da obrađuje log samo od ove tačke, jer su sve prethodne ispravke već zapamćene na disk. Prilikom obrade zapisa loga, REDO upoređuje LSN stranice sa LSN-om trenutnog log zapisa i operaciju primenjuje samo ako je LSN stranice niži, što ukazuje da je trenutna slika stranice starija.

Iako je za ponovno obrađivanje ovog manjeg dela loga dovoljno da se stranice podataka dovedu u traženo stanje, REDO *SQL Servera* obrađuje log počevši od početka najstarije aktivne transakcije. To omogućava oporavku da ponovo zatraži sve katance u okviru aktivnih transakcija i učini bazu podataka dostupnom na kraju ove faze radi bolje dostupnosti. Međutim, to uzrokuje da postupak REDO bude proporcionalan veličini najduže aktivne transakcije.

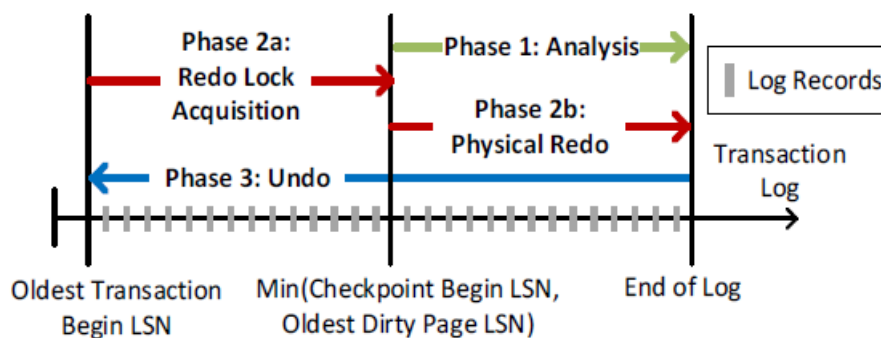
UNDO faza

Ova faza vraća nepotpune transakcije pronađene u ATT-u kako bi se osiguralo očuvanje integriteta baze podataka. Nakon povratka, baza podataka postaje dostupna.

Kako je REDO faza ponovo postavila katance koje zahtevaju ove transakcije, UNDO proces se može izvesti dok je baza podataka dostupna i korisnički upiti biće blokirani samo ako pokušaju da pristupe podacima modifikovanim transakcijama koje čekaju na poništenje.

Za svaku aktivnu transakciju UNDO će skenirati log unazad, počevši od poslednjeg log zapisa koji je generisan ovom transakcijom, i poništiti operaciju koju izvršava svaki log zapis. To čini troškove ove faze srazmernim veličini nezavršenih transakcija. Poništavanje ovih operacija takođe se beleži korišćenjem evidencije log kompenzacije (Compensation Log Records - CLR) da bi se garantovalo da se baza podataka može oporaviti, čak i nakon kvara usred procesa poništavanja. Log zapisi za svaku transakciju mogu se efikasno zaobići korišćenjem prethodnog LSN-a koji je sačuvan u svakom log zapisu. Jednom kada se transakcija izvrši, katanci se oslobađaju.

Na slici 14 je prikazana svaka faza i deo loga koji obrađuje.



Slika 14 – Faze oporavka

Neka se izvrši sledeći kod:

```
BEGIN TRANSACTION
UPDATE tbl
SET col1=REPLICATE ('o',10)
GO
SHUTDOWN WITH NOWAIT
```

Kreirana je in-flight transakcija, jer nije izvršena COMMIT naredba. Zatim je simuliran pad sistema. Nakon restartovanja, pokrenut je proces oporavka, pa će se povlačenjem podataka iz tabele dobiti rezultat kao na slici 15. Dakle, rezultati transakcije su poništeni.

	col1	col2	col3
1	xxxxxxxxxx	bbbbbbbbbb	cccccccccc

Slika 15 – Stanje tabele nakon pada

Trajanje oporavka

Vreme koje je potrebno SQL Server Database Engine-u da obavi REDO i UNDO svih transakcija se naziva vreme oporavka. Vreme se oporavka određuje na osnovu toga koliko je posla urađeno od poslednje kontrolne tačke i koliko je posla urađeno u svim aktivnim transakcijama u trenutku gubitka podataka. SQL Server koristi konfiguracionu opciju koja se naziva interval oporavka (eng. recovery interval) kako bi postavio približni maksimalni broj minuta po bazi podataka koji je potreban SQL Serveru da povрати baze podataka. Ova postavka intervala oporavka kontroliše frekvenciju kontrolne tačke. Ako je interval oporavka postavljen na inicijalna podešavanja i ne postoje dugačke transakcije, oporavak bi trebalo da traje najviše 1 minut. Progres se prati na osnovu log fajla. Na osnovu faze analize, gde se čita log, procenjuje se koliki deo loga će biti pročitán tokom oporavka.

Zaključak

Oporavak baze podataka je jako važna karakteristika svih baza podataka. Kako je primarni cilj baza podataka da pamte podatke, obrađuju ih i obezbeđuju ih korisniku kada su mu potrebni, gubitak bi mogao biti katastrofalan. Zbog toga baze podataka implementiraju algoritme za oporavak. Niz akcija se sprovodi u toku normalnog rada, kako bi vraćanje podataka bilo moguće nakon pada. Na osnovu log zapisa koji se pamte, DBMS može kasnije da ponovo izvrši ili poništi različite akcije kako bi baza ostala u konzistentnom stanju.

Literatura

[1] – SQL Server Transaction Log Architecture and Management Guide, SQL Server documentation

<https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide?view=sql-server-ver15#WAL>

[2] – SQL Server transaction log, SQLShack

<https://www.sqlshack.com/sql-server-transaction-log-part-1-log-structure-write-ahead-logging-wal-algorithm/>

[3] – Understanding recovery performance in SQL Server, SQL Server documentation

[https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms189262\(v=sql.105\)](https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms189262(v=sql.105))

[4] – A beginner's guide to SQL Server transaction logs, SQLShack

<https://www.sqlshack.com/beginners-guide-sql-server-transaction-logs/>

[5] – How do checkpoints work and what gets logged, Paul S. Randal

<https://www.sqlskills.com/blogs/paul/how-do-checkpoints-work-and-what-gets-logged/>

[6] – SQL Server transaction log and recovery models, SQLShack
<https://www.sqlshack.com/sql-server-transaction-log-and-recovery-models/>

[7] - Constant Time Recovery in Azure SQL Database
<https://www.microsoft.com/en-us/research/uploads/prod/2019/06/p700-antonopoulos.pdf>