Mõisted: andmebaasi multikasutus; serialiseeritavus; tupikseis; transaktsioonide rakendamise ekvivalents; lukustamine; kahefaasiline lukustamise protokoll.
Oskus: Kirjeldada multikasutusega seotud probleeme, oma andmebaasi põhjal tuua näide multikasutuse probleemist.

Andmebaasi multikasutus

Olgu meil andmebaasi server, mille teenuseid (üle võrgu või otse terminalide taga) kasutavad samaaegselt mitmed kasutajad. Kuidas juhtida nende koostegevust nii, et andmebaas oleks kogu aeg kooskõlalises seisus? Vastav termin, tähistamaks ala, mis selliste probleemidega tegeleb, on inglise keeles '***concurrency control***', meie nimetame seda multikasutuseks (mitme kasutaja samaaegse kasutuse probleemid). Analoogsed probleemid on tuntud ka operatsioonisüsteemide puhul. Multikasutus on tihedalt seotud andmete taastamise probleemiga.

Üks multikasutuse põhitermineid on transaktsioon (ingl.k. transaction). Arvutis täidetavate operatsioonide jada jagatakse sellisteks lühimateks alamjadadeks, mida vaadatakse tervikuna, s.t. neid enam ei saa jagada ja millede puhul neid käsitatakse täitmise seisukohalt tervikuna, s.t. iga transaktsioon ka lõpetab edukalt või kogu ta täitmine ei osutu võimalikuks – transaktsiooni täitmine ei saa jääda mingisse keskmisse seisu.

Multikasutusega seotud ohud on sõnastatud kolme probleemina.

1. **Muutmise kaotsiminek** - kui kaks transaktsiooni loevad sama kirje, teevad muudatused ja siis kirjutavad tagasi andmebaasi, siis ilma süsteemi spetsiaalvahendeid kasutamata lähevad esimese kirjutaja tulemused kaotsi.
2. **Lõpetamata muudatuse probleem** - kui üks transaktsioon loeb muudetud väärtuse andmebaasist, peale seda teine transaktsioon võtab muudatuse tagasi (rollback), siis esimene kasutab valesid väärtusi.
3. **Kooskõlastamata tegevuse probleem**. Seda probleemi selgitame järgmise näitega.

   Olgu meil andmebaasis vaja teha kaks muudatust.

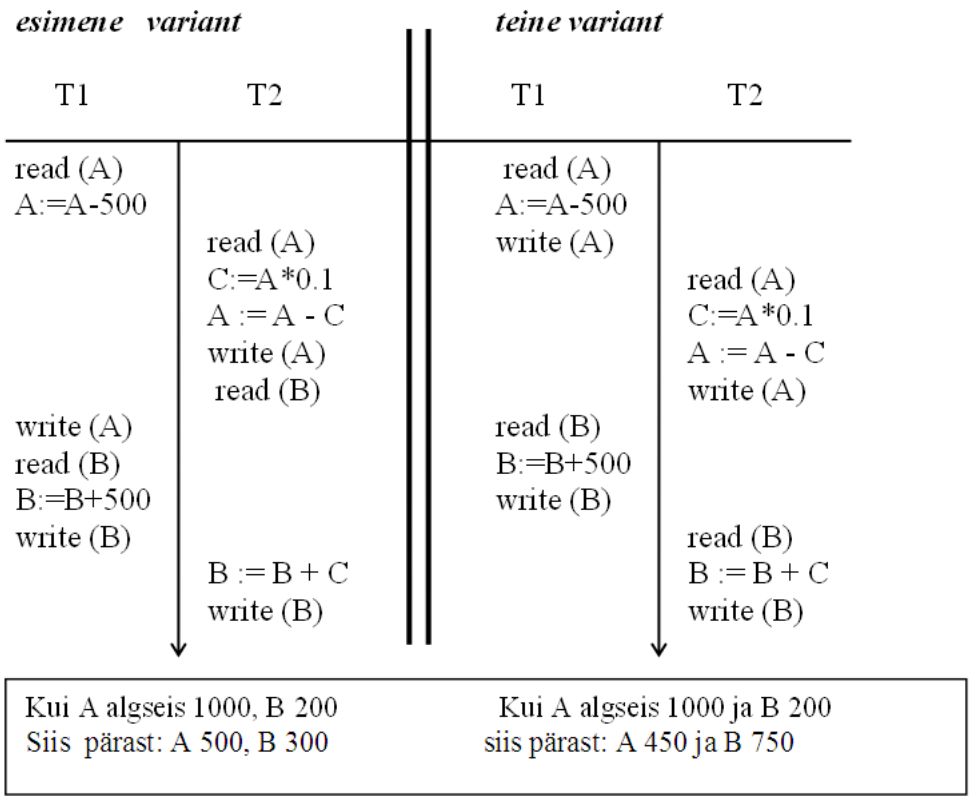
   1. operatsioon: Kanda 500 € arvelt A arvele B.
   2. operatsioon: Kanda 10% arvel A olevast rahast arvele B.

   Seega on vaja teha kaks transaktsiooni: tähistame esimest T1 ja teist T2. Paneme mõlemad algoritmid kirja lihtsas pseudokeeles. NB! Kõik omistamised toimuvad sisemälus, seos andmebaasiga on ainult läbi protseduuride 'read' ja 'write'.

T1	T2
read (A)	read (A)
A:=A-500	C := A*0.1
write (A)	A :=A - C
read (B)	write (A)
B:=B+500	read (B)
write (B)	B :=B + C
	write (B)

On selge, et kui neid transaktsioone rakendada üksteise järel, siis resultaadid sõltuvad nende rakendamise järjekorrast (T1 ja peale selle lõppemist T2; või T2 ja peale selle lõppemist T1), kuid mõlemal juhul jääb A + B muutumatuks. Mis juhtub siis, kui püüda neid operatsioone sooritada samaaegselt, sõltumatuna teisest?

Kaks näidet nende võimalikust samaaegsest sooritamisest.



Kui A algseis 1000, B 200 Siis pärast: A 500, B 300	Kui A algseis 1000 ja B 200 siis pärast: A 450 ja B 750

Kui oletada, et arvel A oli alguses 1000 € ja arvel B 200 €, siis esimesel juhul on pärast transaktsioone A + B 800 €, teisel aga 1200€ . Kuna tegu ainult kahe arve omavaheliste ülekannetega, siis loomulikult ei tohi nende summa muutuda.

Nõue samaaegselt tehtavatele transaktsioonidele: **tulemus peab olema sama, kui nende transaktsioonide eraldi tegemise puhul**. Nõue on tuntud **seriaaliseeritavuse** (ingl.k. *serializability*) nõude nime all. Kui ühe transaktsiooni tulemus ei sõltu teisest, siis neid nimetatakse **vahetatavateks** (ingl.k. *commute*). Üldjuhul on raske öelda, kas kaks transaktsiooni on vahetatavad või mitte (võivad olla läbi kolmandate transaktsioonide kaudselt seotud) ja me ei hakka seda probleemi siin lähemalt uurima.

Vaatame lähemalt aga **serialiseeritavuse** nõuet. Me ütleme, et kaks transaktsioonide rakendamise varianti on **ekvivalentsed**, kui

1. Transaktsioonid, mis osalevad mõlemas variandis on samad;
2. Iga andmeelemendi Q jaoks, kui variandis 1 transaktsioon T_i loeb väärtust Q ning see oli kirjutatud transaktsiooni T_j poolt, siis sama peab kehtima ka variandi 2 jaoks;
3. Kui variandis 1 andmeelement Q viimaseks kirjutajaks oli transaktsioon T_i, siis sama peab kehtima ka variandi 2 jaoks ja nii kõigi andmeelementide jaoks.

Näites variant 1 ei ole ekvivalentne variandiga 2, kuna variandis 2 arve A väärtuse kirjutas T1, variandis 1 aga mitte.

Def. Olgu antud transaktsioonide hulk $\{T_1, ..., T_n\}$, mis osalevad rakendamise variandis S. Me ütleme, et S on **serialiseeritav** kui leidub temaga ekvivalentne rakendamise variant S', mis on seriaalne (ekvivalentne mingi nende transaktsioonide eraldi täitmisega).
Leidub algoritm serialiseeritavuse kontrolliks.

Kõiki eespoolvaadeldud kolme multikasutuse probleemi saab lahendada lukustamise teel. Kasutatakse kaht tüüpi lukke:

a) välistavad lukud e. eksklusiivlukud (ingl.k. *exclusive locks*), mida nimetatakse ka kirjutamislukkudeks (ingl.k. *write locks*) ja
b) kooskasutamise lukud e. lugemislukud (ingl.k. *shared locks* e. *read locks*).

Tüüpiliseks lukustamise objektiks on kirje (lukke saab kasutada ka teistel tasemetel).

Reeglid:

1) Kui transaktsioon T1 lukustab eksklusiivse lukuga kirje p, siis ükski teine transaktsioon seda kirjet kasutada ei saa.
2) Kui transaktsioon T1 paneb lugemisluku kirjele p, siis ükski teine transaktsioon ei saa sama kirjet lukustada eksklusiivse lukuga, aga saab lugeda kooskasutuslukuga.

Andmete kasutamist reguleerivad järgmised reeglid:

1) Iga transaktsioon, mis tahab kirjet p andmebaasist lugeda, peab sellele panema kooskasutusluku .
2) Iga transaktsioon, mis tahab kirjet muuta, peab sellele panema eksklusiivluku.
3) Kui lukustamine ei õnnestu, siis transaktsioon läheb ooteseisundisse - ootab, kuni kirje vabastatakse.
4) Kirjed vabastatakse alles siis, kui transaktsioon on lõppenud (operatsiooni tagasivõtmine e. *rollback*, ei saa enam toimuda).

Tupikseis – ingl.k. *deadlock,* situatsioon kus kaks või enam protsessi sulustavad üksteist. Andmebaasides on see situatsioon, kus kaks või enam transaktsiooni ootavad vastastikku lukkude vabanemist.

Kui eeltoodud näites transaktsioon T1 tahab lukustada kirjed A ja seejärel B, transaktsioon T2 aga tahab need lukustada nii, et enne lukustab B, seejärel A. Kui mõlemad transaktsioonid enam-vähem korraga käima lasta, siis esimene jääb ootama B vabanemist, teine A vabanemist. Ongi tupikseis – kumbki ei saa edasi ja ootavad igavesti

Andmebaasisüsteem peab olema võimeline ära tundma, et tegemist on tupikseisuga ja selle lahendama. Selleks kasutatakse ühe transaktsiooni tagasivõtmist (*rollback*).

Kahefaasiline lukustamise protokoll:

1) mistahes kirje, mida transaktsioon vajab, tuleb lukustada (lukustamisfaas).
2) Kui lukk on vabastatud, ei saa see transaktsioon lukustada enam mitte ühtegi kirjet.

Kui kõik transaktsioonid kasutavad kahefaasilist lukustamise protokolli, siis transaktsioonid on serialiseeritavad.

Andmete granulaarsus

Kõik multikasutuse tehnikad kasutavad mingeid andmeühikuid oma algoritmides. Nendeks ühikuteks võivad olla:

- Andmebaasi kirje või korteež
- Andmeväli
- Andmeplokk kettal
- Kogu fail
- Kogu andmebaas

Mida suurem on andmeühik, seda madalam on samaaegse kasutuse tase. Võrdle näiteks: faili lukustamine ja korteeži lukustamine relatsioonilise mudeli korral. Teisalt – mida väiksem andmeühik, seda enam lukke tekib ja seda keerulisem peab olema nende käsitlemise süsteem, seda rohkem lisaaega lukustamisega tegelemine võtab ja ka rohkem mälu see nõuab.

Kuidas valida õiget granulaarsuse, s.t. andmeühiku taset? Optimaalne andmeühik sõltub sellest, milliseid operatsioone andmetega on vaja teha. Kui transaktsioon käsitleb korraga üksikuid kirjeid, siis on hea andmeühikuks valida korteeži (kirje) tase. Kui aga protsess nõuab suure arvu korteežidega samaaegset tegelemist, võib olla optimaalseks tasemeks terve fail. Tavaliselt kasutavad multikasutuse vahendid ühes süsteemis ühte kindlat andmete granulaarsuse taset, aga leidub ka selliseid süsteeme, kus seda saab valida vastavalt vajadusele.

Kokkuvõtteks:

Andmebaasi juhtimissüsteem peab omama vahendeid mitmesuguse väärkasutuse vastu ja suutma andmeid hoida kooskõlas. Selleks peavad kõik andmebaasisüsteemid omama järgmiste valdkondade vahendeid:

1. Andmebaaside taastamine
2. Andmebaaside multikasutuse juhtimine
3. Andmebaaside turvalisuse kaitse
4. Andmete kooskõlalisuse kontroll

Kirjandus:

1. R. Elmasri, S. B. Navathe Fundamentals of Database Systems. 1989-2003, 4 editions: 1.ed. 1989, 2nd 1994, 3rd 2000, 4th 2003. Chapter: Concurrency Control Techniques.
2. C.J.Date, An introduction to Database Systems, 6th edition.Addition-Wesley, 1995