

Transaktsioonide haldus

Teema 6

Kontekst

- ♦ Andmebaasi kasutavad tüüpiliselt samaaegselt paljud *erinevad kasutajad*.
- ♦ Andmebaasisüsteem peab nende korraldused võimalikult kiiresti täitma, tagades samal ajal, et andmebaas on *korrektses seisundis* ja kasutaja päringutele antakse *korrektsed vastused*.
 - **Hajusates andmebaasides** võidakse neid nõudeid efektiivsuse nimel lõdvendada (vt teema 12).

Transaktsioon e tehing

- ♦ Üldiselt.
 - Infovahetuse ja sellega seotud toimingute jada (näiteks andmebaasi värskendamine), mida käsitletakse ühe tervikuna.
- ♦ SQL-andmebaasis.
 - Järjestatud hulk (üks või rohkem) SQL lauseid:
 - mis täidetakse ühes või mitmes andmebaasis ja
 - mis moodustavad ühe loogilise terviku.
 - Seega: kas kõigi lausete täitmine õnnestub või ükski lause ei mõjuta ühtki SQL skeemi ja SQL-andmebaasis olevaid andmeid.

Piirang

- ♦ Mõnes andmebaasisüsteemis (näiteks Oracle) ei saa andmekäitluskeele (DML) ja andmekirjelduskeele (DDL) laused olla koos ühes transaktsioonis.
 - **NB!** PostgreSQLis saavad andmekäitluskeele (DML) ja andmekirjelduskeele (DDL) laused olla koos ühes transaktsioonis.

Transaktsioonide omadused (ACID)

- ♦ Atomaarsus (**A**tomicity)
 - Transaktsiooni väljakutsuja (klient, rakendus) seisukohalt on transaktsioon loogiline tervik, mis täidetakse kas täielikult või jäetakse täielikult täitmata.
- ♦ Terviklikkus, konsistentsus (**C**onsistency)
 - Transaktsioon viib andmebaasi ühest korrektsest seisundist teise.
- ♦ Isoleeritus (**I**solation)
 - Transaktsioonid ei saa piiluda üksteise vahetulemusi.
- ♦ Püsivus, jätkuvus (**D**urability)
 - Andmebaasisüsteem garanteerib transaktsiooni lõppemise järel selle tehtud muudatuste püsiva salvestamise.

Transaktsioonidest ÕISI näitel

- ♦ Näide – üliõpilane saab samas aines eksamil hinde 0 ja järeleksamil hinde 3.
- ♦ Sama üliõpilase samas aines saadud kahe hinde ÕISI lisamine on *üks loogiline tervik* – üliõpilase tulemuse registreerimine (**atomaarsuse omadus**).
- ♦ Selle tulemus peaks olema kasutajatele nähtav, kui *mõlemad hinded* on lisatud.

Transaktsioonidest ÕISI näitel (2)

- Kui kasutaja näeb hindeid nii, et üks hinne on ÕISis registreeritud ja teist veel ei ole (**pole täidetud isoleerituse omadus**), siis tekitab see asjatut segadust ja meelehärmi.
- Kui hinne on ÕISI kantud, siis ei tohi see sealt kaotsi minna (**püsivuse omadus**).
- Üliõpilasele ei saa panna hinnet -1 või 6 või sisestada ühes aines ühe deklaratsiooni alusel üle kahe hinde (**terviklikkuse omadus**).

7.12.2017

Teema 6

7

Mida ikkagi tähendab terviklikkus?

- Andmed **ei paikne hajusalt**
 - Andmed on kooskõlas jõustatud kitsendustega
- Andmed **paiknevad hajusalt**
 - Andmed on kooskõlas jõustatud kitsendustega +
 - Erinevates arvutivõrgu sõlmedes olevad koopiad on ühesugused
 - Kuna selle tagamisel suur hind (kui koopiad pole kooskõlas, siis andmeid kasutada ei saa), siis on populaarne *eventual consistency* mudel – koopiad sünkroniseeritakse viiteajaga
 - Eventual consistency* tähendab, et andmebaasis olevad andmed on kättesaadavad ka **osaliste võrgukatkestuste** korral

7.12.2017

Teema 6

8

Transaktsioonide omadused (2)

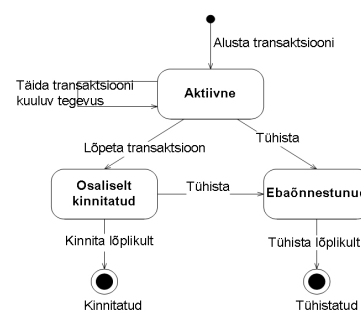
- Kui mitu SQL lauset *ei ole* koondatud üheks transaktsiooniks, peavad ACID omadused kehtima üksiku lause tasemel.
 - INSERT INTO Tabel1(veerg1, veerg2, veerg3) SELECT veerg3, veerg4, veerg5 FROM Tabel2;
 - Täidetakse täielikult või jäetakse täielikult täitmata.
 - Ükski lisatav rida ei tohi minna vastuollu andmebaasis jõustatud kitsendustega.
 - Lisamise tulemused muutuvad teistele kasutajatele nähtavaks alles lause täitmise järel.
 - Lause täitmise järel peab andmebaasisüsteem garanteerima lisatud ridade püsiva salvestamise.

7.12.2017

Teema 6

9

Transaktsiooni seisundid



7.12.2017

Teema 6

10

FAIL Transaktsiooni ebaõnnestumise põhjuseid

- Tühistamise korraldus andmeid kasutavalt programmilt
 - Programm leidis vea
 - Kasutaja andis korralduse
- Andmete vastuolu deklaratiivsete kitsendustega
- Viga trigeri protseduuri täitmisel
 - Viga tekkis mõne trigeri protseduuris täidetud käsu täitmisel
 - Viga tekitati programmeerija korraldusel (RAISE EXCEPTION)

7.12.2017

Teema 6

11

FAIL Transaktsiooni ebaõnnestumise põhjuseid (2)

- Tupik
- Andmeelement on lukustatud ja transaktsioon ei jää selle vabanemist ootama
- SERIALIZABLE isolatsioonitaseme puhul õnnestub konkureerivatest andmemuudatustest esimene
- Hajusa andmebaasi puhul pole mõni server, mille poole pöördutakse, kättesaadav
- Süsteemi viga

7.12.2017

Teema 6

12

SQL laused

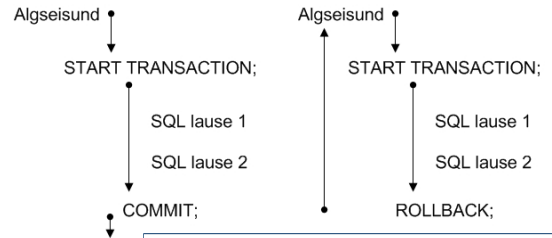
- ♦ START TRANSACTION – alustamine
- ♦ COMMIT – kinnitamine (lõpetamine)
- ♦ ROLLBACK – tühistamine (lõpetamine)
- ♦ SAVEPOINT *nimi* – salvestuspunkti defineerimine
 - Salvestuspunktid võimaldavad emuleerida hierarhilisi transaktsioone
- ♦ ROLLBACK TO SAVEPOINT *nimi* – andmemuudatuste tagasirullimine kuni salvestuspunktini, **kuid mitte transaktsiooni lõpetamine**
- ♦ RELEASE SAVEPOINT *nimi* – salvestuspunkti kustutamine

7.12.2017

Teema 6

13

Transaktsiooni lõpetamine



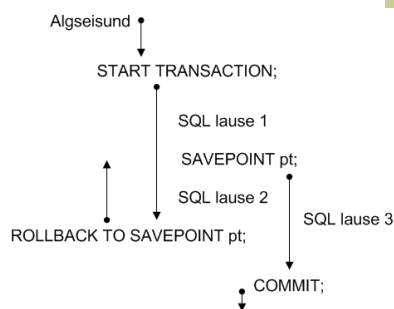
Tagasirullimine võtab rohkem aega kui kinnitamine!

7.12.2017

Teema 6

14

Salvestuspunkt



7.12.2017

Teema 6

15

SQL laused (2)

- ♦ SET CONSTRAINTS – transaktsiooni piires kitsenduste täidetuse kontrolli käitumise muutmine
 - SET CONSTRAINTS ALL IMMEDIATE;
- ♦ transaktsiooni omaduste määramine:
 - SET TRANSACTION – üheks transaktsiooniks
 - SET SESSION CHARACTERISTICS AS TRANSACTION – sessiooniks
 - Tüüp:
 - READ ONLY,
 - READ WRITE.
 - Isolatsiooni tase

7.12.2017

Teema 6

16

Transaktsiooni näide (PostgreSQL)



- ♦ SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;
- ♦ START TRANSACTION;
- ♦ UPDATE bankacct SET balance = balance - 100 WHERE bankacct_id = 82021;
- ♦ UPDATE bankacct SET balance = balance + 100 WHERE bankacct_id = 96814;
- ♦ COMMIT;

7.12.2017

Teema 6

17

PostgreSQL töötab automaatse kinnitamise režiimis

- ♦ Kui ei anta korraldust alustada uut transaktsiooni, siis moodustab iga lause omaette transaktsiooni.
 - INSERT INTO TableA (veerg1, veerg2) SELECT veerg1, veerg FROM TableB;
 - UPDATE TableB SET veerg1=2;

7.12.2017

Teema 6

18

Transaktsiooni näide (PostgreSQL) (2)

- ♦ START TRANSACTION;
- ♦ INSERT INTO Isik(isikukood, eesnimi, perenimi) VALUES ('38101012331', 'Tarmo', 'Tee');
- ♦ INSERT INTO Isik(isikukood, eesnimi, perenimi) VALUES ('38202022556' 'Tiit', 'Tuul');
- ♦ ROLLBACK;
- ♦ Tühistatakse ka käivitunud trigerite tehtud muudatused.

Transaktsiooni näide (PostgreSQL) (3)

- ♦ START TRANSACTION;
- ♦ UPDATE bankacct SET balance = balance - 100 WHERE bankacct_id = 82021;
- ♦ UPDATE bankacct SET balance = balance + 100 WHERE bankacct_id = 96814;
- ♦ SAVEPOINT **kanne_tehtud**;
- ♦ INSERT INTO Log(bankacct_id, comment) VALUES ('82021', 'Raha maha');
- ♦ ROLLBACK TO SAVEPOINT **kanne_tehtud**;
- ♦ COMMIT; /*Andmebaasis UPDATE lausete tulemused, aga mitte INSERT lause tulemus*/

Oracle ei tööta automaatse kinnitamise režiimis

- ♦ **Puudub** eraldi lause transaktsiooni alustamiseks.
- ♦ Esimene andmekäitluskeele lause peale eelmise transaktsiooni lõppu **algatab** sessioonis **uue** transaktsiooni.
- ♦ Iga andmekirjelduskeele lause (CREATE, ALTER, DROP, GRANT, REVOKE) täitmise järel täidab andmebaasisüsteem **automaatselt COMMIT** lause.

Terviklikkust kontrollitakse kohe lause täitmise järel (1)

```
CREATE TABLE bankacct (
  bankacct_id NUMBER(10),
  balance NUMBER(10,2) NOT NULL,
  CONSTRAINT pk_bankacct PRIMARY KEY
    (bankacct_id),
  CONSTRAINT chk_balance CHECK
    (balance >= 0));
```

Kitsendus – jääk ei tohi olla negatiivne

Terviklikkust kontrollitakse kohe lause täitmise järel (2)

- ♦ INSERT INTO bankacct (bankacct_id, balance) VALUES (1, 0);
- ♦ UPDATE bankacct SET balance=-100 WHERE bankacct_id=1;
- ♦ ORA-02290: check constraint (C##TUD1.CHK_BALANCE) violated
- ♦ ROLLBACK;

Terviklikkuse kontrollimise lükkamine transaktsiooni lõppu

```
CREATE TABLE bankacct2 (
  bankacct2_id NUMBER(10),
  balance NUMBER(10,2) NOT NULL,
  CONSTRAINT pk_bankacct2 PRIMARY
    KEY (bankacct2_id) ,
  CONSTRAINT chk_balance2 CHECK
    (balance >= 0) INITIALLY DEFERRED);
```

Terviklikkuse kontrollimise lükkamine transaktsiooni lõppu (2)

- ♦ INSERT INTO bankacct2 (bankacct2_id, balance) VALUES (1, 0);
- ♦ UPDATE bankacct2 SET balance=-100 WHERE bankacct2_id=1; jääk = -100
- ♦ UPDATE bankacct2 SET balance=balance+1000 WHERE bankacct2_id=1; jääk = 900
- ♦ COMMIT; /*Transaktsioon õnnestus*/

7.12.2017

Teema 6

25

Veel PostgreSQL ja Oracle erinevusi

Eeltingimus – tabelis bankacct pole ühtegi rida

- ♦ **Oracle:**

```
INSERT INTO bankacct (bankacct_id, balance) VALUES (1, 0);
INSERT INTO bankacct (bankacct_id, balance) VALUES (2, -1000);
-- ERROR at line 1: ORA-02290: check constraint (C##TUD1.CHK_BALANCE) violated
INSERT INTO bankacct (bankacct_id, balance) VALUES (3, 50000);
COMMIT;
SELECT * FROM bankacct;
-- Tabelis 2 rida
```
- ♦ **PostgreSQL:**

```
START TRANSACTION;
INSERT INTO bankacct (bankacct_id, balance) VALUES (1, 0);
INSERT INTO bankacct (bankacct_id, balance) VALUES (2, -1000);
-- new row for relation "bankacct" violates check constraint "chk_balance"
INSERT INTO bankacct (bankacct_id, balance) VALUES (3, 50000);
-- ERROR: current transaction is aborted, commands ignored until end of transaction block
COMMIT; --Täidetakse ROLLBACK
SELECT * FROM bankacct;
-- Tabelis 0 rida
```

Transaktsiooni algataja peab otsustama, mida erandi korral edasi teha.

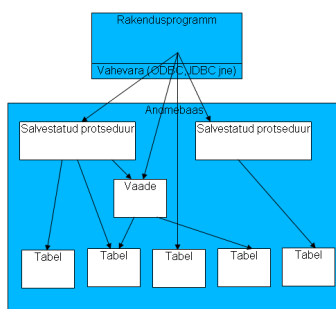
Erandi korral transaktsioon tühistatakse.

7.12.2017

Teema 6

26

Rakenduse arhitektuur



Oracle võimaldab protseduuris COMMIT / ROLLBACK lauset käivitada, PostgreSQL funktsioonis ei ole see võimalik.

7.12.2017

Teema 6

27

Rakenduse arhitektuur (2)

- ♦ Transaktsioonide juhtimisega võiks tegeleda andmebaasi **kasutav programm!**
- ♦ See on **paindlikum**.
 - Erinevates transaktsioonides saab kombineerida erinevate rutiinide väljakutseid.
- ♦ Ebaõnnestumise korral on edasi tehtava tegevuse üle otsustamine lõppkasutajale **lähemal** – lihtsam ja kiirem temalt küsida.

7.12.2017

Teema 6

28



Probleemid

- ♦ Järgnevad probleemid ilmnevad, kui transaktsioonid näevad üksteise kinnitamata vahetulemusi.
- ♦ Öeldakse, et transaktsioonid *trügivad* samade andmete pärast (konkureerivad üksteisega) – nende vahel eksisteerivad *race conditions* (võidusõidu tingimused).

7.12.2017

Teema 6

29

Kaotsi läinud muudatuse probleem

Kui täita kaks transaktsiooni järjest, siis peaks kogus olema **190**

Aeg	Transaktsioon 2	Transaktsioon 1	Kogus mälus asuvas andmeplokis
t1		START TRANSACTION;	100
t2	START TRANSACTION;	oe(kogusx)	100
t3		kogusx=kogusx+100	100
t4	oe(kogusx)		100
t5		kirjuta(kogusx)	200
t6	kogusx=kogusx-10		200
t7	kirjuta(kogusx)	COMMIT;	90
t8	COMMIT;		90

Analoogia – laadite veebi muudatustega töö ja projektikaaslane kirjutab selle üle.

Isolatsioonitasemed

7.12.2017

Teema 6

30



Mis võib tänu sellele juhtuda?

- ♦ How I stole roughly 100 BTC from an exchange and how I could have stolen more!
 - https://www.reddit.com/r/Bitcoin/comments/1wtbiu/how_i_stole_roughly_100_btc_from_an_exchange_and/
 - Rünnak üritas *samaaegselt* teha samalt kontolt väga väikese ja väga suure väljavõtu, lootuses, et väikese väljavõtu järgne kontoseisu väärtus *kirjutab üle* suure väljavõtu järgse kontoseisu väärtuse ning suurt väljavõttu nagu poleks olnudki.

7.12.2017

Teema 6

31

Vahemärkus

- ♦ Veeru pealkiri "*Kogus mälus asuvas andmeplakis*" viitab sellele, et andmebaasisüsteem teeb andmemuudatuse kõigepealt muutmälli loetud plakis ning kirjutab muudetud ploki kettale millalgi hiljem.

7.12.2017

Teema 6

32

Poolikute andmete lugemise probleem

Kui täita kaks transaktsiooni järjest, siis peaks kogus olema **90**

Aeg	Transaktsioon 4	Transaktsioon 3	Kogus mälus asuvas andmeplakis
t1		START TRANSACTION;	100
t2		loe(kogusx)	100
t3		kogusx=kogusx+100	100
t4	START TRANSACTION;	kirjuta(kogusx)	200
t5	loe(kogusx)	...	200
t6	kogusx=kogusx-10	ROLLBACK;	200
t7	kirjuta(kogusx)		190
t8	COMMIT;		190

Analoogia – laadite veebi pooliku töö, mida õppejõud hakkab hindama.

Isolatsioonitasemed

7.12.2017

Teema 6

33

Hägusa lugemise probleem

Aeg	Transaktsioon 3	Transaktsioon 4
t1	START TRANSACTION;	
t2	SELECT col2 FROM t WHERE col=5;	
t3		START TRANSACTION;
t4		UPDATE t SET col2=7 WHERE col=5;
t5		COMMIT;
t6	SELECT col2 FROM t WHERE col=5;	
t7	COMMIT;	

Sama välja väärtuse korduval lugemisel transaktsiooni käigus on tulemuseks erinev väärtus.

7.12.2017

Teema 6

Isolatsioonitasemed

34

Fantoomkirje probleem

Aeg	Transaktsioon 3	Transaktsioon 4
t1	START TRANSACTION;	
t2	SELECT * FROM t WHERE col=5;	
t3		START TRANSACTION;
t4		INSERT INTO t(col) VALUES (5);
t5		UPDATE t SET col=5 WHERE col=6
t6		COMMIT;
t7	SELECT * FROM t WHERE col=5;	
t8	COMMIT;	

Sama päringu korduval täitmisel transaktsiooni käigus on tulemuseks saadav ridade hulk erinev.

7.12.2017

Teema 6

Isolatsioonitasemed

35



Kokkuvõte

- ♦ Transaktsioonid ei tohi näha üksteise vahetulemusi – peab kehtima **isoleerituse** omadus.
- ♦ Mida **suurem** isoleeritus, seda **vähem** eelnimetatud probleeme!

7.12.2017

Teema 6

36

Tööplaan

- ◆ Konkureerivatest transaktsioonides sisalduvatest operatsioonidest kokku pandud operatsioonide jada, mis säilitab operatsioonide järjekorra transaktsioonide sees.

Järjestikulise tööplaani näide

Aeg	Transaktsioon 2	Transaktsioon 1
t1		START TRANSACTION;
t2		loe(kogusx)
t3		kirjuta(kogusx)
t4		loe(kogusy)
t5		kirjuta(kogusy)
t6		COMMIT;
t7	START TRANSACTION;	
t8	loe(kogusx)	
t9	kirjuta(kogusx)	
t10	loe(kogusy)	
t11	kirjuta(kogusy)	
t12	COMMIT;	

Mittejärjestikulise tööplaani näide

Aeg	Transaktsioon 2	Transaktsioon 1
t1		START TRANSACTION;
t2		loe(kogusx)
t3		kirjuta(kogusx)
t4	START TRANSACTION;	
t5	loe(kogusx)	
t6		loe(kogusy)
t7	kirjuta(kogusx)	
t8		kirjuta(kogusy)
t9		COMMIT;
t10	loe(kogusy)	
t11	kirjuta(kogusy)	
t12	COMMIT;	

Mittetaastatava tööplaani näide

Aeg	Transaktsioon 1	Transaktsioon 2
t1	START TRANSACTION;	
t2	loe(kogusx)	
t3	kogusx = kogusx + 100	
t4	kirjuta(kogusx)	
t5		START TRANSACTION;
t6		loe(kogusx)
t7		kogusx = kogusx * 1,1
t8		kirjuta(kogusx)
t9		loe(kogusy)
t10		kogusy = kogusy * 1,1
t11		kirjuta(kogusy)
t12	loe(kogusy)	COMMIT;
t13	kogusx = kogusx - 100	
t14	kirjuta(kogusy)	
t15	ROLLBACK;	

Tb

Ta

Andmebaasi kasutatav mittejärjestikuline tööplaani peab olema taastatav.

Taastatav tööplaani. Iga transaktsioonide paari Ta ja Tb puhul kehtib: kui Ta loeb andmebaasist, mille on Tb eelnevalt kirjutatud, siis Tb peab lõppema enne kui Ta (kui Tb tühistatakse, siis saab ka Ta tühistada).

Serialiseeritav tööplaan

- ◆ **Serialiseeritav** tööplaan, on selline *mittejärjestikuline* tööplaan, mille alusel toimingute teostamine annab sõltumata andmebaasi algseisundist garanteeritult sama tulemuse, kui mingi *järjestikuline* tööplaan.
- ◆ Tööplaani koostab andmebaasisüsteemi komponent – *plaanur*.
- ◆ Plaanuri ülesanne – koostada *serialiseeritav* tööplaan.

Konfliktipõhine serialiseerimine

- ◆ Enamlevinud plaanuri strateegia.
- ◆ Plaanur koostab serialiseeritava tööplaani:
 - iga **konfliktsete** operatsioonide paari op_i ja op_j puhul on nende operatsioonide teostamise järjekord sama, nagu järjestikuses tööplaanis,
 - iga **mittekonfliktsete** operatsioonide paari op_i ja op_m puhul võib plaanur valida, kumb operatsioon täita varem ja kumb hiljem.

Andmebaasid II 2017 © Erki Eessaar

Mittekonfliktsete operatsioonide paarid

- ♦ $r_a(X), r_b(X)$
- ♦ $r_a(X), r_b(Y)$
- ♦ $r_a(X), w_b(Y)$
- ♦ $w_a(X), r_b(Y)$
- ♦ $w_a(X), w_b(Y)$


r – read – lugemisoperatsioon
 w – write – kirjutamisoperatsioon
 a, b – transaktsioonid
 X, Y – andmelemendid

- ♦ Selliste paaride puhul *ei ole* operatsioonide täitmise järjekord oluline.

7.12.2017 Teema 6 43

Andmebaasid II 2017 © Erki Eessaar

Konfliktsete operatsioonide paarid



LET EAT!

- ♦ $r_a(X), w_a(X)$ (sama transaktsiooni sees toimuvad operatsioonid)
- ♦ $w_a(X), w_b(X)$
- ♦ $r_a(X), w_b(X)$
- ♦ $w_a(X), r_b(X)$

Operatsioonid pöörduvad sama andmelemendi poole ja vähemalt üks nendest on kirjutamisoperatsioon.

- ♦ Selliste paaride puhul *on* operatsioonide täitmise järjekord oluline.

7.12.2017 Teema 6 44

Andmebaasid II 2017 © Erki Eessaar

Konfliktipõhine serialiseerimine (2)

- ♦ Kaks tööplaani S1 ja S2 on *konflikti-ekvivalentsed* kui.
 - S1 ja S2 hõlmavad samu transaktsioone (sama operatsioonide järjekorraga transaktsioonide sees).
 - Konfliktsete operatsioonipaare järjekord on tööplaanides S1 ja S2 ühesugune.
- ♦ Tööplan on konfliktipõhiselt serialiseeritav, kui see on konflikti-ekvivalentne mingi järjestikulise tööplaaniga.
- ♦ Eesmärk: koostada konfliktipõhiselt serialiseeritavad tööplaanid.

7.12.2017 Teema 6 45

Andmebaasid II 2017 © Erki Eessaar


Serialiseerimise saavutamise vahendid

- ♦ **Konservatiivne** (pessimistlik) lähenemine (nt lukustamine).
 - Andmebaasisüsteemides enimlevinud meetod!
 - Eeldatakse, et transaktsioonide paralleelsel täitmisel võib tekkida konflikte ja nendega tuleb kohe (mitte tagantjärele) tegeleda.
- ♦ **Optimistlik** lähenemine (nt ajatemplitel ja valideerimisel põhinevad meetodid).

7.12.2017 Teema 6 46

Andmebaasid II 2017 © Erki Eessaar

Lukustamine



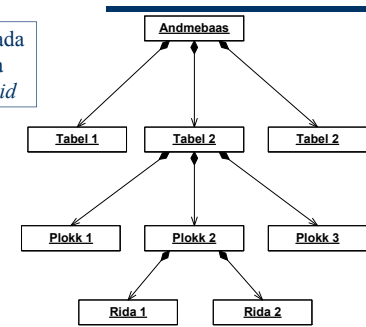
- ♦ Andmelemendi kasutamise blokeerimine.
- ♦ Lukkude tüübid:
 - jagatud lukk e ühislukk – lugemiseks,
 - eksklusiivne lukk e monopollukk – muutmiseks.
- ♦ Teiste transaktsioonide valikud, kui element eksklusiivselt lukustatud:
 - ootamine,
 - tagasirullimine.

7.12.2017 Teema 6 47

Andmebaasid II 2017 © Erki Eessaar

Elementide granulaarsus

Lukustada tuleb ka indekseid



Andmete samaaegse kasutuse võimalus väheneb

Andmete samaaegse kasutuse võimalus suureneb

7.12.2017 Teema 6 48

Lukkude tüübid

- ♦ **Jagatult** lukustatud elemendile (ja selle alam- ning ülelementidele) saab teine transaktsioon panna jagatud luku, aga ei saa panna eksklusiivset lukku.
 - Andmete lugemine blokeerib nende samaaegse muutmise teiste transaktsioonide poolt.
- ♦ **Eksklusiivselt** lukustatud elemendile (ja selle alam- ning ülelementidele) ei saa teine transaktsioon teisi lukke panna.
 - Andmete muutmine blokeerib nende samaaegse lugemise ja muutmise teiste transaktsioonide poolt.
- ♦ **Kavatsuslik lukk** – teistele hierarhia elementidele.

7.12.2017

Teema 6

49

Lukkude ühilduvus

Elemendil oleva luku tüüp	Elemendile küsitava luku tüüp		
		jagatud	eksklusiivne
	jagatud	jah	ei
	eksklusiivne	ei	ei

Eeldus – transaktsioon T_a üritab lukustada elementi, mis on lukustatud transaktsiooni T_b poolt.

7.12.2017

Teema 6

50

DDL (andmekirjelduskeele) lukud

- ♦ Tabeli struktuuri muutmise ajal ei saa seal andmeid lugeda ega muuta (tabel on lukustatud).
- ♦ Tabelis olevate andmete lugemise ja muutmise ajal ei saa muuta selle tabeli struktuuri (tabel on lukustatud).

7.12.2017

Teema 6

51

Kahefaasiline lukustamisprotokoll (2PL)

- ♦ **Teoreem:** Kui kõik transaktsioonid täidavad kahefaasilist lukustamisprotokoll, siis on kõik antud protokoll järgi võimalikud mittejärjestikulised tööplaanid serialiseeritavad.

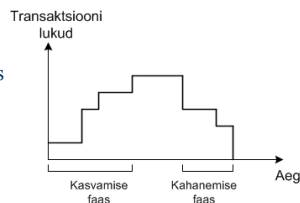
7.12.2017

Teema 6

52

Kahefaasiline lukustamisprotokoll (2PL) (2)

- ♦ Transaktsiooni kaks faasi:
 - kasvamise (lukkude küsimise) faas
 - sellel ajal lukke ei vabastata
 - kahanemise (lukkude vabastamise) faas
 - sellel ajal ei küsita uusi lukke



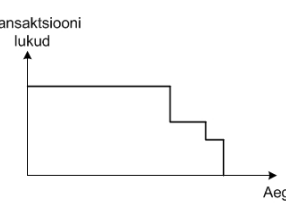
7.12.2017

Teema 6

53

Kahefaasiline lukustamisprotokoll (2PL) – variatsioonid

- ♦ Konservatiivne (staatiline) 2PL (C2PL)
 - Kõik vajalikud lukud küsitakse transaktsiooni algul (enne kõige esimest lugemise või kirjutamise operatsiooni).
 - Transaktsiooni käigus vabastatakse lukke.



7.12.2017

Teema 6

54

Kahefaasiline lukustamisprotokoll (2PL) – variatsioonid (2)

- ♦ Range 2PL (S2PL)
 - Transaktsiooni käigus küsitud *eksklusiivsed* lukud vabastatakse alles transaktsiooni lõpus.
 - *Jagatud* lukke vabastatakse transaktsiooni käigus.

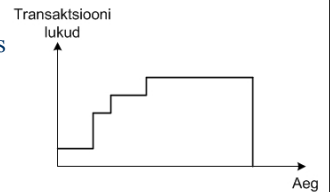
7.12.2017

Teema 6

55

Kahefaasiline lukustamisprotokoll (2PL) – variatsioonid (3)

- ♦ Tugev range 2PL (SS2PL)
 - Transaktsiooni käigus küsitud *jagatud* ja *eksklusiivsed* lukud vabastatakse alles transaktsiooni lõpus.
 - Seda protokollit kasutavad enamik andmebaasisüsteeme.



7.12.2017

Teema 6

56

Lukustamine ja kaotsiläinud muudatuse probleemi stsenaarium

Aeg	Transaktsioon 2	Transaktsioon 1	Kogus mälus asuvas andmeplokis
1		START TRANSACTION.	100
2	START TRANSACTION.	lukusta kogusx jagatult	100
3		loe(kogusx)	100
4	lukusta kogusx jagatult	kogusx=kogusx+100	100
5	loe(kogusx)		100
6	kogusx=kogusx-10	lukusta kogusx eksklusiivselt, et teostada kirjutat(kogusx)	100
7	lukusta kogusx eksklusiivselt et teostada tekib tupik	OOTA	100
8	ROLLBACK: eemalda transaktsiooni poolt kogusx pandud lukud	OOTA	100
9		lukusta kogusx eksklusiivselt kirjutat(kogusx)	100
10		COMMIT: eemalda transaktsiooni poolt kogusx pandud lukud	200

Tupik e ummik e "surnud ringi tekkimine" on patiseis, mis tekib kui kaks või rohkem võistlevat protsessi ootavad üksteise taga, et need mingi ressursi kasutamise lõpetaksid ja ülejäänud protsessid saaksid tööd jätkata.

7.12.2017

Teema 6

57

Lukustamine ja poolikute andmete lugemise stsenaarium.

Aeg	Transaktsioon 4	Transaktsioon 3	Kogus mälus asuvas andmeplokis
1		START TRANSACTION.	100
2		lukusta kogusx jagatult	100
3		loe(kogusx)	100
4	START TRANSACTION.	kogusx=kogusx+100	200
5	lukusta kogusx jagatult, et teostada loe(kogusx)	kirjutat(kogusx)	200
6	OOTA	ROLLBACK	100
7	lukusta kogusx jagatult		100
8	loe(kogusx)		100
9	kogusx=kogusx-10		100
10	lukusta kogusx eksklusiivselt kirjutat(kogusx)		90
11	COMMIT: eemalda transaktsiooni poolt kogusx pandud lukud		90

7.12.2017

Teema 6

58

Lukkude tehniline realiseerimine

- ♦ Luku **eskaleerimine** – suure hulga tabeli ridade lukustamisel lukustatakse automaatselt **kogu tabel**.
- ♦ Riivid.
 - Lühiajalised lukud plokkidele.
 - Süsteem paneb neid plokkide lugemiseks ja kirjutamiseks.
 - Põhjus, miks töökiiruse optimeerimine üritab vähendada operatsioonide täitmiseks vajalikku plokkide lugemiste ja kirjutamiste arvu.



7.12.2017

Teema 6

59

Kus hoida andmeid lukkude kohta?

- ♦ **Tsentraalselt.** Muutmälus eraldiseisev lukustatud elementide nimekiri.
 - Elemendi lukustatuse kindlaks tegemiseks tuleb see läbi vaadata.
 - Kui nimekiri pole kättesaadav, siis süsteem ei saa toimida.
- ♦ **Hajutatult.** Andmed lukustamise kohta on seotud lukustatud elemendiga.
 - Elemendi lukustaja leiab info kiiremini.
 - Parem töökindlus, sest pole keskset nimekirja.

7.12.2017

Teema 6

60

Vajalikud tingimused tupiku tekkimiseks

- ♦ Eksisteerib **ressurss**, mida saab korraga kasutada vaid **üks protsess**.
- ♦ Protsess, mis juba kasutab mingeid ressursse, võib soovida kasutada **veel ressursse**, mida omakorda kasutavad teised protsessid.
- ♦ Protsessilt **ei saa** ressursi **jõuga** ära võtta. Ressursi saab vabastada vaid seda hoidev protsess.
- ♦ Kaks või rohkem protsessi moodustavad "**surunud ringi**" kui iga selles osalev protsess **ootab**, et ringis järgmisena osalev protsess vabastaks mingi ressursi.

7.12.2017

Teema 6

61

Tupik – patiseis



Transaktsioon1	Transaktsioon2	Kirjeldus
START TRANSACTION	START TRANSACTION	
UPDATE rida 64		Kumbki transaktsioon lukustab/muudab erinevad read.
UPDATE rida 83	UPDATE rida 83	
	UPDATE rida 64	Transaktsioon 1 jääb ootama, et transaktsioon 2 vabastaks luku tabeli reale 83.
	Automaatne tagasirullimine	Transaktsioon 2 jääb ootama, et transaktsioon 1 vabastaks luku tabeli reale 64.
COMMIT		Tekkis tupiku olukord ja transaktsioon 2 rulliti tagasi.
		Transaktsioon 1 saab oma töö lõpetada.

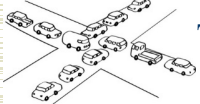
Lõpetamiseks võib kasutada *aegumise meetodit* – transaktsioon võib oodata vaid mingi ajaperioodi.

7.12.2017

Teema 6

62

Tupiku käsitlemine



- ♦ Kuidas ennetada?
- ♦ Kui tupik on siiski tekkinud, siis kuidas lahendada?

7.12.2017

Teema 6

63

Võimalikud meetodid tupiku ennetamiseks

- ♦ **Korraga** saab ressursi (nt andmebaas) kasutada ainult **üks** kasutaja.
- ♦ Ilmselt vastuvõetamatu!
- ♦ Tupiku võimalus on **hind**, mida tuleb ressursside **jagamise** eest maksta.
- ♦ Disainivalikud aitavad tupikute võimalust **vähendada**.

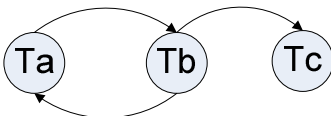
7.12.2017

Teema 6

64

Võimalikud meetodid tupiku ennetamiseks (2)

- ♦ Perioodiliselt genereeritakse transaktsioonide põhjal *suunatud graaf*, kust otsitakse *tsükleid*.
- ♦ Tipud – transaktsioonid.
- ♦ Kahe tipu T_a ja T_b vahel on orienteeritud kaar $T_a \Rightarrow T_b$ sellisel juhul kui T_a ootab, et lukustada andmelement, mis on hetkel lukustatud T_b poolt.



Transaktsioonide graaf, mis sisaldab tsüklit ja viitab tupikule.

Edasi

7.12.2017

Teema 6

65

Võimalikud meetodid tupiku ennetamiseks (3)

- ♦ Transaktsioonil on *ajatempel*
- ♦ Ajatemplid on *täielikult järjestatud*
- ♦ Ajatempel saadakse süsteemi kella või loenduri abil
- ♦ *Olukord*: Transaktsioon T_a lukustab elementi, T_b tahab ka sama elementi lukustada
- ♦ Erinevad algoritmid nagu *Wait-Die* ja *Wound Wait*, mille järgimine aitab tupikuid vältida.

7.12.2017

Teema 6

66

Võimalikud meetodid tupiku ennetamiseks (3)

- ♦ Wait-Die (**Oota-Sure**).
 - Kui T_b on vanem kui T_a , siis jääb T_b ootama (**Oota**)
 - Kui T_b on noorem kui T_a , siis rullitakse T_b tagasi (**Sure**) ja käivitatakse uuesti sama ajatempliga
- ♦ Wound-Wait (**Haava-Oota**).
 - Kui T_b on vanem kui T_a , siis noorem transaktsioon T_a rullitakse tagasi ja käivitatakse uuesti sama ajatempliga (**Haava**)
 - Kui T_b on noorem kui T_a , siis jääb T_b ootama (**Oota**)



Tupiku lõpetamine

Why Me?

- ♦ "Ohvri" valimine.
 - Ohvriks transaktsioon, mis vähem toiminguid teinud
 - Ohvriks transaktsioon, mis vähem aega kestnud
 - Vältida transaktsiooni pidevalt ohvriks jäämist
 - Uuesti käivitamisel kasutada vana ajatemplit – transaktsioon "vananeb" ja selle eelistamise tõenäosus suureneb
- ♦ Kui kaugale tagasirullida?
 - Kogu transaktsioon tagasirullida
 - Rullida tagasi salvestuspunkti
- ♦ Mida "ohver" edasi teeb?
 - Automaatne uuesti käivitamine
 - Tööjärg rakendusele



Transaktsiooni "nälgimine"

- ♦ Ei õnnestu kuidagi elementi lukustada.
- ♦ Võimalik lahendus – prioriteedid ja nende dünaamiline muutmine.
- ♦ Nii *Wound-Wait* kui *Wait-Die* väldivad nälgimist, sest tagasirullitav transaktsioon *vananeb* ja seoses sellega paraneb transaktsiooni jätkamise võimalus.
 - Nende algoritmide probleem – liiga palju tagasirullimisi.

Multiversioon-konkurentsjuhtimine

- ♦ *Multi-version concurrency control (MVCC)*.
- ♦ Samade andmete samaaegne lugemine ja kirjutamine ei lähe omavahel konflikti ja ei blokeeri üksteist.
 - Süsteemides, mis ei toeta, blokeerivad lugemine ja muutmine üksteist.
- ♦ **Hind:** Andmebaasisüsteem peab säilitama ridade või plokkide versioone kuni leidub mõni transaktsioon, mis soovib neid kasutada.

Multiversioon-konkurentsjuhtimine (2)

Ajahetk	Transaktsioon 1	Transaktsioon 2
t1	SELECT Sum(balance) AS summa FROM Account;	
t2		UPDATE Account SET balance=balance-100000 WHERE Account.account_id=23428;
t3	Kontole numbriga 23428 vastav rida loetakse varasemate ridade versioonide seast.	
t4	COMMIT;	
t5		COMMIT;

Süsteem, mis MVCC'd ei toeta, täidaks UPDATE lause alles peale SELECT lauset sisaldava transaktsiooni lõppu.

Multiversioon-konkurentsjuhtimine (3)

- ♦ Kui transaktsiooni T_i käigus tuleb lugeda objekti o , mida transaktsioon T_j parajasti muudab, siis loeb süsteem T_i täitmiseks o muutmiseelse versiooni.
- ♦ Kui transaktsiooni T_i käigus tuleb muuta objekti o , mida transaktsioon T_j parajasti loeb, siis saab T_i muuta o -d ja T_j saab kasutada o muutmiseelset versiooni.

Multiversioon-konkurentsjuhtimine (4)

- ◆ Kui transaktsiooni T_i käigus tuleb muuta objekti o , mida transaktsioon T_j parajasti muudab, siis T_i jääb ootele.
- ◆ Objekti all mõeldakse *loogilist objekti* nagu tabeli rida.
- ◆ Objektide versioonide säilitamine toimub süsteemi poolt automaatselt.

Multiversioon-konkurentsjuhtimine (5)

- ◆ Eelised.
 - Andmete lugemine ei blokeeri nende andmete muutmist.
 - Andmete muutmine ei blokeeri nende andmete lugemist.
- ◆ Probleemid.
 - Versioonide hoidmine nõuab lisaruumi.
 - Andmete lugemine ja muutmine võib muutuda aeglasemaks (sõltub realiseerimisest).
 - Nõuab eritähelepanu andmebaasis loodud rutiinides/trigerites ja andmebaasi kasutavates programmides.

Multiversioon-konkurentsjuhtimine (6)

- ◆ Andmebaasi kasutavad programmid ning andmebaasis loodud protseduurid/funktsioonid/trigerid/meetodid (ja nende kirjutajad) peavad arvestama, et lugemisoperatsioonid ei blokeeri kirjutamisoperatsioone (ja vastupidi).

Multiversioon-konkurentsjuhtimine (7)

- ◆ PostgreSQL ja Oracle realiseerivad selle automaatselt kõigi transaktsioonide korral.
- ◆ MS SQL Serveris on see võimalik sisse lülitada (vaikimisi välja lülitatud).
 - Kui andmebaasisüsteem MVCC-d ei realiseeri, siis andmete lugemine blokeerib nende samade andmete samaaegse muutmise mõne teise transaktsiooni poolt ja vastupidi.

Hierarhilised transaktsioonid



START TRANSACTION (transaktsioon A)
 ...
 START TRANSACTION (transaktsioon B)
 muuda rida X
 COMMIT (transaktsioon B)
 START TRANSACTION (transaktsioon C)
 muuda rida X
 COMMIT (transaktsioon C)
 ...
 ROLLBACK (transaktsioon A)

- ◆ B ja C on vennad.
- ◆ A on B ja C vanem.
- ◆ "Vendi" saab täita **parallelselt**.
- ◆ Alamtransaktsiooni lõppemisel pärib vanem nende lukud.
- ◆ Alamtransaktsiooni kinnitamisel muutuvad tulemused nähtavaks vendadele ja vanemale.
- ◆ Alamtransaktsioonid saavad elemendi eksklusiivselt lukustada kui sellel pole lukku või ainus lukustaja on vanem.
- ◆ A tühistamisel tühistatakse B ja C.

Hierarhilise transaktsiooni emuleerimine salvestuspunktide abil

- ◆ START TRANSACTION;
- ◆ INSERT INTO Dept(deptno, dname) VALUES (100,'Advert');
- ◆ **SAVEPOINT sub_transaction1;**
- ◆ **UPDATE Dept SET dname='Advert' WHERE deptno=100;**
- ◆ **ROLLBACK TO SAVEPOINT sub_transaction1;**
- ◆ COMMIT;
- ◆ Transaktsiooni käigus tühistati alamtransaktsioon.
- ◆ Transaktsiooni tulemusena lisati tabelisse Dept osakond nr 100, mille nimi on "Advert".

Kauakestvad transaktsioonid

- Kui kaua on "liiga kaua" sõltub olukorrast ja võib olla sekunditest päevadeni.
- Kauakestvate transaktsioonide probleemid.
 - Tagasirullimisel tuleb teha väga palju tööd.
 - Transaktsiooni lõpuni kehtvat lukustamist ei saa lubada.
- Kauakestvaid transaktsioone kasutavate süsteemide näideteks on:
 - disainisüsteemid (CASE, CAD),
 - töövoo süsteemid.

Saaga

- Saaga on kogum toiminguid, mis üheskoos moodustavad kauakestva transaktsiooni.
- Saaga koosneb *lühikestest* alamtransaktsioonidest, millest igaühe puhul kasutatakse traditsioonilisi andmete samaaegse kasutamise juhtimise meetmeid (nt lukustamine).
- Saaga kasutamine lõdvendab *isoleerituse* omadust.
- Saaga tagasirullimisel igale alamtransaktsioonile **A** *kompenseeriv transaktsioon* **A⁻¹**.

Stsenaarium – $A + A^{-1}$ ei anna sama tulemust kui A puudumine

Stsenaarium koos kompenseerivate transaktsioonidega	Stsenaarium ilma kompenseerivate transaktsioonideta
Arvel on 2000 EUR . Arve jääk ei tohi olla negatiivne. A : Arvele kantakse 1000 eurot (jääk 3000 EUR) B : Arvelt võetakse 2500 eurot (jääk 500 EUR) A⁻¹ : Arvelt võetakse 1000 eurot. Viga! (jääk 500 EUR)	Arvel on 2000 EUR . Arve jääk ei tohi olla negatiivne. B : Arvelt võetakse 2500 eurot. Operatsioon ebaõnnestub, sest jääk ei tohi olla negatiivne (jääk 2000 EUR)

Isolatsioonitasemed

- Transaktsiooni üks põhiomadus on isoleeritus.
- Mida *kõrgema* isolatsioonitasemega on tegu,
 - seda vähem tekib andmete samaaegse kasutamise probleeme,
 - kuid samas vähenevad andmete samaaegse kasutamise võimalused (ja seega süsteemi jõudlus väheneb).
- SQL standardi järgi vaikimisi – **SERIALIZABLE**.
- Paljudes andmebaasisüsteemides vaikimisi – **READ COMMITTED**.

Isolatsioonitasemed – SQL:2011

isoleeritus

väike

suur

Isolatsioonitase	Poolikute andmete lugemise probleem	Hägusa lugemise probleem	Fantoomkirje probleem
READ UNCOMMITTED – ei tohi andmeid muuta	jah	jah	jah
READ COMMITTED	ei	jah	jah
REPEATABLE READ	ei	ei	jah
SERIALIZABLE	ei	ei	ei

READ COMMITTED isolatsioonitase (Oracle näide) (1)

```
CREATE TABLE Test (nr NUMBER(5));
INSERT INTO Test (nr) VALUES (1);
INSERT INTO Test (nr) VALUES (2);
COMMIT;
```

Ajalahet	Transaktsioon 1	Transaktsioon 2
1	SET TRANSACTION ISOLATION LEVEL READ COMMITTED.	
2	SELECT * FROM Test; (2 rida)	
3	(Päringu täitmisel kontrollitakse, ega andmed pole peale päringu algatamist muutunud. Kuna ei ole, siis loeb andmed tabelile vastavatest andmeplokkidest)	
4	SELECT * FROM Test; (2 rida)	DELETE FROM Test;
5	(Muudatuse teinud transaktsiooni pole veel kinnitatud.)	
6	SELECT * FROM Test; (0 rida)	COMMIT;
7	COMMIT;	
8	SELECT * FROM Test; (0 rida)	

Lause "näeb" enne selle käivitamist kinnitatud andmeid ja samuti teda sisaldava transaktsiooni käigus tehtud muudatusi.

Fantoomkirje probleem – transaktsioonis päringu mitmekordne täitmine andmete põhjal, mida transaktsioon pole muutnud. Tulemuseks on erinev ridade hulk.

Andmebaasid II 2017 © Erki Eessaar

SERIALIZABLE isolatsioonitase (Oracle näide) (2)

INSERT INTO Test (nr) VALUES (1);
INSERT INTO Test (nr) VALUES (2);
COMMIT;

Ajahetk	Transaktsioon 1	Transaktsioon 2
t1	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	
t2	SELECT * FROM Test (2 rida) (Päringu täpsisel kontrollitakse, ega andmed pole peale esimese serialiseeritud transaktsioonis sisaldunud kogu algust muutunud. Kuna ei ole, siis loetakse andmed tabelile vastavatest andmeplökidest)	
t3		DELETE FROM Test;
t4		COMMIT;
t5	SELECT * FROM Test (2 rida) (Päringu täpsisel kontrollitakse, ega andmed pole peale esimese serialiseeritud transaktsioonis sisaldunud kogu algust muutunud. Kuna on, siis loetakse andmed undo/rollback segmentid selle seaduse, nagu need olid transaktsiooni algul)	
t6	COMMIT;	
t7	SELECT * FROM Test (0 rida)	

7.12.2017 Teema 6 85

Lause "näeb" enne seda sisaldava transaktsiooni algust kinnitatud andmeid ja samuti teda sisaldava transaktsiooni käigus tehtud muudatusi.

Tagasi

Andmebaasid II 2017 © Erki Eessaar

READ COMMITTED isolatsioonitase (Oracle näide) (3)

Ajahetk	Transaktsioon 1	Transaktsioon 2
t1	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	
t2		UPDATE Test SET nr=3 WHERE nr=2;
t3	UPDATE Test SET nr=3 WHERE nr=2;	
t4	OOTAB	COMMIT;
t5	Muudab 0 rida.	
t6	COMMIT;	

7.12.2017 Teema 6 86

Andmebaasid II 2017 © Erki Eessaar

SERIALIZABLE isolatsioonitase (Oracle näide) (4)

Ajahetk	Transaktsioon 1	Transaktsioon 2
t1	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	
t2		UPDATE Test SET nr=3 WHERE nr=2;
t3	UPDATE Test SET nr=3 WHERE nr=2; (jaab ootama, sest teine transaktsioon on muutmise eesmärgil sama rea juba lukustanud)	
t4		COMMIT;
t5	ERROR at line 1: ORA-08177: can't serialize access for this transaction (kui transaktsioon töötab serialiseerimise isolatsioonitasemel ja üritab muuta rida, mida tema töö ajal on muudetud, siis transaktsioon ebaõnnestub)	
t6	ROLLBACK;	

7.12.2017 Teema 6 87

Reegel: esimene muutja võidab!
MVCC mittekasutamise korral tuleb kogu tabel lukustada.

Andmebaasid II 2017 © Erki Eessaar

MVCC puhul on võimalik "skew write" probleem

CREATE TABLE A (nr NUMBER(5));
CREATE TABLE B (nr NUMBER(5));

Ajahetk	Transaktsioon 1	Transaktsioon 2
t1	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	
t2		SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
t3	INSERT INTO A SELECT Count(*) FROM B;	
t4		INSERT INTO B SELECT Count(*) FROM A;
t5	COMMIT;	
t6		COMMIT;
t7	SELECT * FROM A; (1 rida - väärtus 0)	
t8		SELECT * FROM B; (1 rida - väärtus 0)

7.12.2017 Teema 6 88

Serialiseeritud tööplani järgi peaks tabelis B olema väärtus 1 ja tabelis A väärtus 0!

Andmebaasid II 2017 © Erki Eessaar

"Skew write" probleemi lahendusi

- Ühine rida, mida kumbki transaktsioon muudab (üks jääks ootama).
- Anda käsk tabeli lukustamiseks, kui andmebaasisüsteem sellist käsku võimaldab kasutada.
 - Tähendab, et andmebaasi programmeerija peab hakkama ise realiseerima lukustamisskeemi – halb!
 - SQL standard ei kirjelda LOCK lauset ja SELECT ... FOR UPDATE lauset.

7.12.2017 Teema 6 89

Andmebaasid II 2017 © Erki Eessaar

"Skew write" probleemi lahendusi (2)

- Andmebaasisüsteemis PostgreSQL 9.1 on uuendusena SERIALIZABLE isolatsioonitase realiseeritud *Serializable Snapshot Isolation (SSI)* algoritmi põhjal.
 - Väldib "skew write" probleemi tekkimist.
 - Kui **kõik** andmebaasis toimuvad transaktsioonid kasutaksid (SSI-põhist) SERIALIZABLE isolatsioonitaset, siis ei oleks vaja ühegi ärireegli jõustamiseks kasutada LOCK, SELECT FOR SHARE või SELECT FOR UPDATE lauset (ilmutatud kujul korraldusi üksikute ridade või terve tabeli lukustamiseks).

7.12.2017 Teema 6 90

Andmebaasid II 2017 © Erki Eessaar

"Skew write" probleemi lahendus – PostgreSQL 10

```
CREATE TABLE A (nr INTEGER);
CREATE TABLE B (nr INTEGER);
```

Ajahetk	Transaktsioon 1	Transaktsioon 2
t1	START TRANSACTION;	
t2	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;	
t3		START TRANSACTION;
t4		SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
t5	INSERT INTO A SELECT Count(*) FROM B;	
t6		INSERT INTO B SELECT Count(*) FROM A;
t7	COMMIT;	
t8	SELECT * FROM A; (1 rida – väärtus 0)	
t9		COMMIT;
		ERROR: could not serialize access due to read/write dependencies among transactions DETAIL: Reason code: Canceled on identification as a pivot, during commit attempt.
t10		SELECT * FROM B; (0 rida)

7.12.2017 Teema 6 91

Andmebaasid II 2017 © Erki Eessaar

Vahekokkuvõte

- PostgreSQL (alates 9.1) ja Oracle andmebaasisüsteemis ei tähenda **SERIALIZABLE** isolatsioonitase ühte ja sama asja (mõelge, kui segadust tekitav on see nt rakenduste programmeerijatele).
 - PostgreSQLis tähendab see **tugevat** isolatsiooni, Oracles **nõrgemat**.
 - Oracle **ei realiseeri** õiget **SERIALIZABLE** isolatsioonitaset SQL standardi mõttes.

7.12.2017 Teema 6 92

Andmebaasid II 2017 © Erki Eessaar

Parimad praktikad seoses lukustamisega

- Kui tabelis T toimuvad samaaegsed sagedased väikese ulatusega andmemuudatused, siis ära käivita samal ajal massiivseid muudatusi (nt **pakktöötlus**), mis hõlmavad kõiki või suurt osa T ridadest.
 - Pakktöötlus** – andmete rühmiti töötlemine ilma inimese sekkumiseta.

7.12.2017 Teema 6 93

Andmebaasid II 2017 © Erki Eessaar

Parimad praktikad seoses lukustamisega (2)

- Ära teosta andmebaasiobjektide (tabelid, indeksid) **kirjelduste muudatusi** ajal, kui andmeid loetakse ja muudetakse.
 - Kirjelduste muudatuste jaoks võidakse kasutajate juurdepääsu andmebaasile piirata.
 - Öösel, nädalavahetuse – võimalikult lühikest aega.
 - Andmebaasisüsteemide arendajad otsivad viise, kuidas andmekäitluse ja andmekirjelduse operatsioonid üksteist võimalikult vähe segaks.

7.12.2017 Teema 6 94

Andmebaasid II 2017 © Erki Eessaar

Muudatuste tegemise akna näide

lisatud: 01.12.2016 16:38
TÄHELEPANU!
Seoses TTÜ struktuurireformiga on ÕIS kinni alates 24.12.2016 kuni 01.01.2017. Sel ajal toimub ÕISI uuele struktuurile üleviimine....

Õppekavad, üliõpilased, õppeained, õppejõud tõstetakse uude struktuuri. Alates 02.01.2017 on kõik uutes struktuurides. Õppejõudude ja üliõpilaste kasutajaõigused ÕISis säilivad. Kõik ülejäänud kasutajaõigused (rollid) tuleb jaanuaris uuesti seadistada vastavalt uue struktuuri juhi taotlusele.

Ilmselt segaksid selleks vajalikud operatsioonid ja igapäevased ÕISI kasutamise operatsioonid üksteist liiga palju – operatsioonid hakkaksid üksteise järgi ootama.

7.12.2017 Teema 6 95

Andmebaasid II 2017 © Erki Eessaar


Parimad praktikad seoses lukustamisega (3)

- Hoia lukke nii **vähe aega** kui võimalik.
 - Jaga mahukas andmete muutmise transaktsioon mitmeks transaktsiooniks.
 - Hoia andmebaasiga mitte seotud töö (näiteks mahukas arvutused, suhtlus lõppkasutajatega) andmebaasi transaktsioonist väljas.
 - Kasuta andmebaasiserveris talletatud rutiine, et vähendada võrguliiklust, mis kulutab aega ja pikendab transaktsiooni.

7.12.2017 Teema 6 96

Andmebaasid II 2017 © Erki Eessaar

Parimad praktikad seoses lukustamisega (4)



- ♦ **Liiga mahukad** transaktsioonid.
 - Lukke hoitakse liiga kaua – blokeerib teisi kasutajaid.
 - Andmete muutmiseelse versiooni hoidmine nõuab palju ressursse.
 - Suurem tõenäosus erandi tekkimiseks, mille tulemusel tuleb transaktsioon tagasirullida.
 - Transaktsiooni tühistamine aeglane.
- ♦ **Liiga väikesed** transaktsioonid.
 - Suhteliselt liiga palju aega kulub COMMIT lause te täitmiseks.

7.12.2017 Teema 6 97

Andmebaasid II 2017 © Erki Eessaar

Parimad praktikad seoses lukustamisega (5)

- ♦ Ärge kasutage **koodiga juhitud** arvujada generaatorit.
 - Tänu kasutatava abitabeli lukustamise vajadusele saab tabelisse korraga andmeid lisada vaid üks kasutaja.
 - Kasutage unikaalsete väärtuste genereerimiseks andmebaasisüsteemi pakutavaid võimalusi (nt arvujada generaatorid).

7.12.2017 Teema 6 98

Andmebaasid II 2017 © Erki Eessaar

Parimad praktikad seoses lukustamisega (6)

- ♦ "Traditsioonilise" (mitte-MVCC) lukustamise meetodi korral blokeerib andmete lugemine samaaegse andmete muutmise ja vastupidi.

7.12.2017 Teema 6 99

Andmebaasid II 2017 © Erki Eessaar

Parimad praktikad seoses lukustamisega (7)

- ♦ Kui andmebaasis samaaegselt **koondandmete leidmise päringud** ja sagedased **väikese ulatusega andmemuudatused/andmete lugemised**, siis hakkavad need operatsioonid üksteist segama.
 - Lahenduseks on luua **andmeait** ja/või **andmevakad**, mis on mõeldud mahukate päringute täitmiseks.
 - Isegi kui operatiivandmete andmebaasi andmebaasisüsteem toetab MVCC-d on mõistlik luua eraldi andmeait ja/või andmevakad.
 - Erinevad operatsioonid.
 - Eeldavad erinevat tüüpi andmebaasi disaini sh indekseid.

7.12.2017 Teema 6 100

Andmebaasid II 2017 © Erki Eessaar

Parimad praktikad seoses lukustamisega (8)

- ♦ Kui tarkvara hakkab kasutama uut andmebaasisüsteemi või sama andmebaasisüsteemi **uut versiooni**, siis ole valmis muutusteks lukustamise põhimõtetes!
 - Näiteks mingit tüüpi operatsioon kasutab teistsuguse "kangusega" lukku kui varem.
 - Tarkvara hakkab mingites (piir?) olukordades teisiti käituma.
 - Testi, testi, testi!

7.12.2017 Teema 6 101

Andmebaasid II 2017 © Erki Eessaar


Parimad praktikad seoses lukustamisega (9)

- ♦ Tupikute võimaluse vähendamiseks pöördu transaktsioonides andmebaasiobjektide (nt tabelid) poole **samas järjekorras**.
- ♦ Valiku koht.
 - Madalam isolatsioonitase – "nõrgemad" lukud, vähem transaktsioonide poolt üksteise blokeerimist, väiksem tupikute tõenäosus.
 - Kõrgem isolatsioonitase – vähem andmete samaaegse kasutamise vigu.

7.12.2017 Teema 6 102

Andmebaasid II 2017 © Erki Eessaar

Lukustamine ja rakendused – pessimistlik lähenemine




- Mitu kasutajat tahavad samal ajal lugeda sama rida, et loetud andmeid muuta.
- Esimene lugemine **lukustab** rea muutmiseks.
- Kuni lukk on peal, jäävad järgmised muutmised ootele (või lükatakse tagasi).
- Kui andmete muutmise transaktsioon lõpeb, siis lukk eemaldatakse ja võidujooks reale järgmise muutmise luku panemiseks jätkub.

7.12.2017 Teema 6 103

Andmebaasid II 2017 © Erki Eessaar

Lukustamine ja rakendused – pessimistlik lähenemine (2)




- Multiversioon konkurentsjuhtimise (nt PostgreSQL, Oracle) korral ei blokeeri "tavaline" SELECT andmete muutmist.
- Tuleb kasutada SELECT ... FOR UPDATE või LOCK TABLE.
 - Lukustada nii palju kui vajalik, nii vähe kui võimalik.

7.12.2017 Teema 6 104

Andmebaasid II 2017 © Erki Eessaar

Lukustamine ja rakendused – optimistlik lähenemine



- Mitu kasutajat loevad samal ajal muutmiseks sama rida.
- Muutja peab muudetud rea tagasikirjutamisel kontrollima, ega rida pole vahepeal muutunud.
 - Kui on, siis muudatus ei õnnestu.
 - Muutja peab meeles, milline oli rida enne muutmist.
 - Esimesel muudatuste salvestajal muudatus õnnestub.

7.12.2017 Teema 6 105

Andmebaasid II 2017 © Erki Eessaar

Lukustamine ja rakendused - kokkuvõte

- Pessimistlik lähenemine
 - konfliktide ennetamine
 - võib tingida tupiku
 - levinud kasutaja arvutis "istuvates" rakendustes
- Optimistlik lähenemine
 - konfliktide tuvastamine ja nendega käigult tegelemine
 - levinud veebirakendustes
 - negatiivne kasutajakogemus – sisestan nõutud andmed ja siis ...

7.12.2017 Teema 6 106

Andmebaasid II 2017 © Erki Eessaar

Optimistlik lähenemine rakenduses – Oracle APEX

1 error has occurred

• Current version of data in database has changed since user initiated update process. current row version identifier = "A6AF6783B2E28665DFB26C7EA2EB0D28" application row version identifier = "945FCA09E7B76BBBC90BF9145DC005B8B"

Kaks kasutajat muutsid Oracle APEXis tehtud veebirakenduse kaudu samal ajal sama rida. Esimene muudatuste salvestus õnnestus. Teine salvestamine ebaõnnestus ja süsteem väljastas veateate.

7.12.2017 Teema 6 107

Andmebaasid II 2017 © Erki Eessaar

Rakenduste arendajad peavad oma tööd kooskõlastama

Rakendus 1 (optimistlik)

- SELECT empno, sal, **row_ver** FROM Emp WHERE empno=7788; /*Loetud rea versioon oli 8*/
- UPDATE Emp SET sal=sal+100, **row_ver=row_ver+1** WHERE empno=7788 **AND row_ver=8**;

Rakendus 2 (pessimistlik)

- START TRANSACTION;
- SELECT empno, sal FROM Emp WHERE empno=7788 **FOR UPDATE**;
- UPDATE Emp SET sal=sal+100 WHERE empno=7788;
- COMMIT;

Rakendus 1 kirjutab rakenduse 2 tehtud muudatuse üle. Rakenduse 1 arvates pole rida vahepeal muutunud.

7.12.2017 Teema 6 108

Pessimistlik lähenemine (Oracle näitel)

- ♦ SELECT empno, ename, sal FROM Emp WHERE empno=10; /* lugemine ei lukusta rida eksklusiivselt – teised transaktsioonid saavad seda muuta*/
- ♦ SELECT empno, ename, sal FROM Emp WHERE empno=10 FOR UPDATE NOWAIT;
 - -- spetsiaalne süntaks päringuga koos rea *eksklusiivseks* lukustamiseks
- ♦ UPDATE Emp SET sal=sal+1000 WHERE empno=10; /* kuna rida oli eksklusiivselt lukustatud, siis on kindel, et keegi pole seda vahepeal muutnud */
- ♦ COMMIT;

7.12.2017

Teema 6

109

Optimistlik lähenemine (Oracle näitel)

- ♦ SELECT empno, ename, sal FROM Emp WHERE empno=10;
- ♦ UPDATE Emp SET sal=sal+1000 WHERE empno=10 AND SAL=1000;
- ♦ /*Kui keegi on vahepeal rida muutnud ja WHERE klauslis olev tingimus enam ei kehti, siis rida ei muudeta*/
- ♦ COMMIT;

7.12.2017

Teema 6

110

Optimistlik lähenemine (Oracle näitel) (2)

- ♦ SELECT ora_rowscn, empno, ename, sal FROM Emp WHERE empno=10;
- ♦ Pseudoveerule ora_rowscn vastavas väljas on sellise transaktsiooni süsteemi muutuste arv (SCN), mis viimati seda rida muutis.
- ♦ Tabel tuleb luua kasutades ROWDEPENDENCIES määrangut.
 - Rea suurus kasvab 6 baiti

7.12.2017

Teema 6

111

Optimistlik lähenemine (Oracle näitel) (3)

- ♦ Andmemuudatuste andmebaasi viimisel tuleb kontrollida, et reaga seotud SCN poleks peale lugemist muutunud.
- ♦ UPDATE Emp SET sal=sal+1000 WHERE empno=10 AND ora_rowscn=:scn_väärtus;

7.12.2017

Teema 6

112

Optimistlik lähenemine (PostgreSQL)

- ♦ PostgreSQLis saab ridade versioonide määramiseks kasutada **süsteemset** (andmebaasisüsteem loob ise ja muudab ise väärtuseid) ja **nähtamatut** (andmete lugemiseks peab otse viitama) veergu *xmin*.
- ♦ Kui muudetakse rida, siis tekib reast uus versioon, millel on eelmisest versioonist suurem *xmin* väärtus.

7.12.2017

Teema 6

113

Oracle

- ♦ Multiversioon-konkurentsjuhtimine (MVCC)
- ♦ Rutiinides võib kinnitada transaktsioone
- ♦ Transaktsioonidega seotud SQL laused:
 - COMMIT
 - ROLLBACK
 - SAVEPOINT
 - ROLLBACK TO SAVEPOINT
 - SET TRANSACTION
 - SET CONSTRAINT
 - ALTER SESSION

Puudub eraldi lause transaktsiooni alustamiseks

7.12.2017

Teema 6

114

Oracle – autonoomsed transaksioonid

```
CREATE TABLE t (msg VARCHAR2(25));

CREATE OR REPLACE PROCEDURE Autonomous_Insert
AS
  pragma autonomous_transaction;
BEGIN
  INSERT INTO t(msg) VALUES ( 'Autonomous Insert' );
  COMMIT;
END;
/
```

7.12.2017

Teema 6

115

Oracle – autonoomsed transaksioonid (2)

```
CREATE OR REPLACE PROCEDURE
  NonAutonomous_Insert
AS
BEGIN
  INSERT INTO t(msg) VALUES (
    'NonAutonomous Insert' );
  COMMIT;
END;
/
```

7.12.2017

Teema 6

116

Oracle – autonoomsed transaksioonid (3)

```
BEGIN
  INSERT INTO t(msg) VALUES ( 'Anonymous
    Block' );
  NonAutonomous_Insert;
  ROLLBACK;
END;
/
SELECT * FROM t;
  ■ Tabelis on kaks rida, sest COMMIT lause protseduuris
    NonAutonomous_Insert kinnitas transaksiooni.
```

7.12.2017

Teema 6

117

Oracle – autonoomsed transaksioonid (4)

```
BEGIN
  INSERT INTO t(msg) VALUES ( 'Anonymous Block' );
  Autonomous_Insert;
  ROLLBACK;
END;
/
SELECT * FROM t;
  ■ Tabelis on üks rida, mis lisati Autonomous_Insert poolt.
  ■ ROLLBACK lause rullis tagasi anonüümses ploki oleva
    INSERT lause.
```

7.12.2017

Teema 6

118

Kus autonoomseid transaksioone kasutada?

- ♦ Erandolukordade registreerimine
- ♦ Kõigi (ka ebaõnnestunud!!) andmemuudatuste registreerimine

7.12.2017

Teema 6

119

Oracle (2)

- ♦ Spetsiaalsed *SQL laused* lukustamiseks (ei kuulu standardisse):
 - SELECT ... FOR UPDATE
 - SELECT * FROM Aine WHERE aine_kood='IDU0120' FOR UPDATE;
 - Leitud read lukustatakse eksklusiivselt.
 - LOCK TABLE
 - LOCK TABLE Aine IN EXCLUSIVE MODE;
 - Saab lukustada ka sektsioone.
 - Pakett DBMS_LOCK
- ♦ Andmeid lukkude kohta hoitakse andmeplokkides.

7.12.2017

Teema 6

120

Oracle (3)

- ♦ Lukud vabastatakse transaktsiooni lõpus.
- ♦ Andmekirjelduskeele ja andmekäitluskeele lauseid ei saa panna kokku ühte transaktsiooni – iga andmekirjelduskeele lause täitmise järel täidetakse andmebaasisüsteemi poolt automaatselt COMMIT lause.

Oracle (4)

- ♦ Alates Oracle 11g (Release 1 ja mõnel juhul Release 2) on märgatavalt suurenenud võimalused teha DDL (andmekirjelduskeele) operatsioone *aktiivselt kasutuses olevates andmebaasides*.
 - Näide: Alates Oracle Database 11g Release 1 saab lisada tabelisse veeru samal ajal kui mõni teine transaktsioon selles tabelis andmeid loeb või muudab.

PostgreSQL

- ♦ Multiversion-konkurentsjuhtimine (MVCC)
- ♦ Funktsioonides ei tohi sisalduda transaktsioonide juhtimise lauseid
- ♦ Transaktsioonidega seotud SQL laused:
 - START TRANSACTION
 - COMMIT
 - ROLLBACK
 - SAVEPOINT
 - ROLLBACK TO SAVEPOINT
 - SET TRANSACTION
 - SET CONSTRAINT
 - SET SESSION

PostgreSQL (2)

- ♦ Spetsiaalsed *SQL laused* lukustamiseks:
 - SELECT ... FOR UPDATE
 - SELECT * FROM Aine WHERE aine_kood='IDU0120' FOR UPDATE NOWAIT;
 - Leitud read lukustatakse eksklusiivselt.
 - LOCK TABLE
 - LOCK TABLE Aine IN ACCESS EXCLUSIVE MODE NOWAIT;
- ♦ Lukud vabastatakse üldjuhul transaktsiooni lõpus.
 - Salvestuspunktini tagasirullimisel vabastatakse tagasirullitud lausete täitmise käigus küsitud lukud.

PostgreSQL (3)

- ♦ Andmekirjelduskeele ja andmekäitluskeele lauseid saab panna kokku ühte transaktsiooni.
- ♦ Saab luua *kitsenduste trigerid*, mis on baastabeliga seotud reataseme AFTER trigerid ja mille protseduuri täitmine on võimalik lükata transaktsiooni lõppu.
 - Mõeldud keerukamate kitsenduste jõustamiseks.



Vigade liigitus ja tõrjumise meetodid

- ♦ **Andmete sisestusviga.**
 - Tabelitega seotud deklaratiivsed kitsendused ja sisestatud andmeid kontrollivad trigerid, mis käivituvad tabelis andmete muutmisel.
- ♦ **Andmekandja rike.** Ühelt või mitmelt kettalt pole võimalik andmeid lugeda.
 - Liiasuse kasutamine:
 - RAID,
 - andmebaasist varukoopiate tegemine ja nende paigutamine primaarsest andmebaasist geograafiliselt eemale.

Andmebaasid II 2017 © Erki Eessaar

Vigade liigitus ja tõrjumise meetodid (2)

- ♦ **Katastroofiline viga.** Kõik ühes geograafilises punktis asuvad andmekandjad hävivad.
 - Andmebaasist varukoopiategemine ja nende paigutamine erinevatesse geograafilistesse punktidesse.

7.12.2017 Teema 6 127

Andmebaasid II 2017 © Erki Eessaar

Vigade liigitus ja tõrjumise meetodid (3)

- ♦ **Süsteemi viga,** mille tulemusel kaob info serveri muutmälust ja läheb kaotsi informatsioon hetkel toimuvate transaktsioonide seisundi kohta.
 - Süsteemi poolne logifailide genereerimine ning nende alusel hiljem andmebaasi taastamine.

7.12.2017 Teema 6 128

Andmebaasid II 2017 © Erki Eessaar

Jätkuvus

- ♦ Transaktsiooni üks põhiomadus on **jätkuvus**.
- ♦ Süsteem peab tagama, et peale transaktsiooni kinnitamist ei lähe selle tehtud muudatused kaotsi (st saavad püsivalt andmebaasis salvestatud).
- ♦ Andmemuudatuste tegemise järjekord.
 - Kõigepealt muutmälus
 - Seejärel stabiilses andmebaasis – kettal salvestatud failides

7.12.2017 Teema 6 129

Andmebaasid II 2017 © Erki Eessaar

Probleem

- ♦ Mida teha, kui peale transaktsiooni kinnitamist toimub **voolukatkestus** ning serveri muutmälust kustuvad transaktsiooni käigus muudetud andmeplokid, mida pole veel kettale kirjutatud?

7.12.2017 Teema 6 130

Andmebaasid II 2017 © Erki Eessaar

Tähistus

- ♦ **undo** – transaktsiooni käigus tehtud muudatused tuleb andmebaasi veaolukorra järgsel taastamisel *tagasirullida*
- ♦ **redo** – transaktsiooni käigus tehtud muudatused tuleb andmebaasi veaolukorra järgsel taastamisel *uuesti teha*
- ♦ **stabiilne andmebaas** – andmefailid kettal

7.12.2017 Teema 6 131

Andmebaasid II 2017 © Erki Eessaar

Andmete salvestamise ja taastamise tehnikad

- ♦ no undo / no redo
 - Andmemuudatused stabiilsesse andmebaasi **kohe peale** transaktsiooni **lõppu**, aga mitte enne.
- ♦ no undo / with redo
 - Andmemuudatused stabiilsesse andmebaasi mingi aeg **peale** transaktsiooni lõppu, aga mitte enne lõppu.
- ♦ with undo / no redo
 - Kõik muutmälus tehtud muudatused **kohe** stabiilsesse andmebaasi.
- ♦ with undo / with redo
 - Muudetud andmeplokke kirjutatakse regulaarselt muutmälust stabiilsesse andmebaasi. Võib nii enne kui pärast transaktsiooni kinnitamist. **Kõige paindlikum lahendus!!**

7.12.2017 Teema 6 132

Tagajärjed

- ♦ **no undo** – Muutmälu peab olema palju, sest kinnitamata muudatused tuleb mälus hoida.
- ♦ **no redo** – Sage plokkide kettale kirjutamine. Sealhulgas ühe ja sama ploki sage kettale kirjutamine.
- ♦ Samas on andmebaasi taastamine lihtsam võrreldes **with undo/with redo** meetodiga.

With undo/wiht redo – transaktsiooni alustamine

- ♦ Transaktsiooni alguse logimine.
- ♦ Andmeploki otsimine ja mällu lugemine.
- ♦ Lukustamine.
- ♦ Andmeploki muutmise, ploki märkimine muudetuks ja muudatuse logi genereerimine.
- ♦ Logikirjete (perioodiline ja sage) salvestamine kettale.

Logikirje võimalik struktuur

- ♦ unikaalne järjekorranumber
- ♦ transaktsiooni identifikaator (SCN – Oracle)
- ♦ muudatuse aeg
- ♦ muudatuse tüüp (insert, update, delete)
- ♦ ploki number, millel asus muudetud rida
- ♦ andmeelemendi väärtus enne muutmist
- ♦ andmeelemendi väärtus peale muutmist
- ♦ viited eelmisele ja järgnevale logikirjele, mis käivad sama transaktsiooni kohta

Write Ahead Logging (WAL)

- ♦ Ükski muudatus ei lähe stabiilsesse andmebaasi **enne**, kui selle kohta käiv logi on kirjutatud kettal asuvasse logifaili (st püsivalt salvestatud).
- ♦ Tagab, et iga stabiilses andmebaasis tehtud muudatuse kohta on olemas ka **logikirje** logifailis.
- ♦ Vajadusel on võimalik see muudatus tühistada (*undo*) või uuesti teha (*redo*).



Kontrollpunkt (*checkpoint*)

- ♦ Andmebaasisüsteem kirjutab kõik muudetud plokid (räpased plokid) muutmälust kettal asuvasse andmefailidesse.
- ♦ Rakendatakse *Write Ahead Logging* (WAL) meetodit.
- ♦ Andmebaasi taastamisel veaolukorrast on vaja käsitleda transaktsioone, mis lõppesid *peale viimast* kontrollpunkti.

Kontrollpunkti läbiviimine

- ♦ Üldiselt hoolitseb selle regulaarse läbiviimise eest andmebaasisüsteem.
- ♦ Eeliskasutaja võib sundida süsteemi jõuga koheselt kontrollpunkti läbi viima.
 - PostgreSQLis tuleb kasutada lauset:
 - CHECKPOINT;
 - Oracles tuleb kasutada lauset:
 - ALTER SYSTEM CHECKPOINT;

Andmebaasid II 2017 © Erki Eessaar

With undo/wiht redo – transaktsiooni lõpetamine

- Transaktsiooni kinnitamine.
 - Andmete kitsendustele vastavuse kontroll (deklaratiivsete kitsenduste puhul, mille täidetuse kontroll on lõkatud transaktsiooni lõpu).
 - Ressursside vabastamine (lukud, kursorid, ajutised tabelid).
 - Transaktsiooni lõpu logi genereerimine ja logi andmete püsisalvestamine.
- Transaktsiooni tühistamine.
 - Ressursside vabastamine (lukud, kursorid, ajutised tabelid).
 - Muudatuste tühistamine logi põhjal.
 - Iga muudatuse tühistamise kohta logi genereerimine.
 - Transaktsiooni lõpu logi genereerimine ja logi andmete püsisalvestamine.
- Tühistamine võtab palju rohkem aega kui kinnitamine!!

7.12.2017 Teema 6 139

Andmebaasid II 2017 © Erki Eessaar

With undo/wiht redo – andmete taastamine

Kõigi muutmäls olevate muudetud plokkide kettale kirjutamine.

T1 on OK (muudatused püsisalvestatud)!

T2 ja T3 tuleb uuesti teha (redo)!

T4 ja T5 tuleb tagasirullida (undo)!

Kontrollpunkt

Viga

T1, T2 ja T3 on lõppenud (kas kinnitamise või tühistamisega)

7.12.2017 Teema 6 140

Andmebaasid II 2017 © Erki Eessaar

With undo/wiht redo – andmete taastamine (2)

- Eelduseks on **regulaarne** varundamine.
- Varundamispoliitika on üks osa **turvapoliitikast!**
 - Andmefailid
 - Logifailid
 - Logifailide grupid (koopid erinevatel ketastel)
- Peale taastamist.
 - Viimase kontrollpunkti ja veaolukorra vahel lõpetatud transaktsioonid tuleb uuesti teha (**Redo**).
 - Vea hetkel aktiivsed transaktsioonid tuleb tagasirullida (**Undo**).

7.12.2017 Teema 6 141

Andmebaasid II 2017 © Erki Eessaar

With undo/wiht redo – andmete taastamise algoritm

- Analüüsi faas.
 - Võitja-transaktsioonide** nimekiri – logifailis kirje **kinnitamise või tühistamise** kohta peale viimast kontrollpunkti.
 - St ka tühistatud transaktsioone vaadeldakse võitjatena.
 - Kaotaja-transaktsioonide** nimekiri – logifailis kirje transaktsiooni alguse kohta, kuid pole kirjet transaktsiooni kinnitamise või tühistamise kohta.
- Muudatuste uuesti tegemise faas.
 - Logikirjete järjekorranumbrite *kasvamise* järjekorras tehakse kõik võitja-transaktsioonide toimingud uuesti.

7.12.2017 Teema 6 142

Andmebaasid II 2017 © Erki Eessaar

With undo/wiht redo – andmete taastamise algoritm (2)

- Muudatuste tühistamise faas.
 - Logikirjete järjekorranumbrite kahanemise järjekorras tühistatakse kõigi kaotaja-transaktsioonide tehtud muudatused.
- Muudetud plokkide salvestamine.
- Logi kirjed, mida pole enam vaja undo ega redo jaoks, võib kustutada.

7.12.2017 Teema 6 143

Andmebaasid II 2017 © Erki Eessaar

Milleks on vaja kontrollpunkti?

- Kontrollpunktis toimub kõigi muutmäls muudetud plokkide kettale kirjutamine.
- Vajalik, et andmebaasi taastamisel, peaks süsteem tühistama ja uuesti tegema **võimalikult vähe** transaktsioone.
- Peale kontrollpunkti ei lähe enam vaja enne seda lõppenud transaktsioonide kohta käivat logi.

7.12.2017 Teema 6 144

Oracle

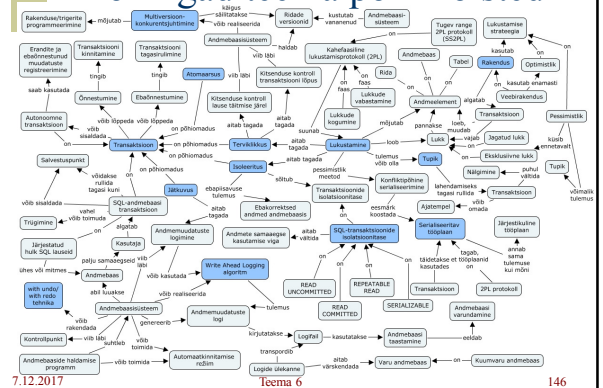
- ♦ Milleks läheb vaja *rollback/undo segmente*?
 - Transaktsiooni tühistamiseks.
 - MVCC tagamiseks.
 - Tagasiulatuvate päringute võimaldamiseks.
- ♦ Milleks läheb vaja *logifaile*?
 - Andmebaasi taastamiseks korrektses seisus.
 - Logigrupp – erinevates asukohtades asuvate logifailide kogum. Neisse kirjutatakse logi paralleelselt.

7.12.2017

Teema 6

145

Mõningad teema põhimõisted



7.12.2017

Teema 6

146