Concurrency Control

Persikirtimų kontrolė

DBVS

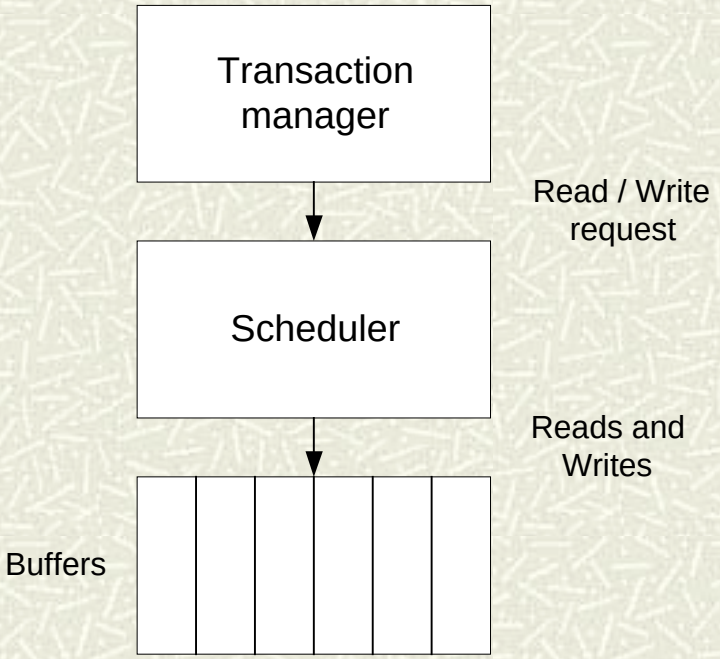
Parengė: Jurgita Dabulytė, doktorantė

Concurrency Control

- Serial and Serializable Schedules
- Conflict – Serializability
- Enforcing Serializability by Locks
- Locking Systems With Several Lock Modes
- An Architecture for a Locking Scheduler
- Managing Hierarchies of Database Elements
- The Tree Protocol
- Concurrency Control by Timestamps
- Concurrency Control by Validation

•DBVS komponentė *scheduler* t.y. funkcija, kontroliuojanti skirtingų tranzakcijų žingsnius.

•Procesas, garantuojantis, kad tranzakcijos išlaikys nuoseklumą veikdamos vienu metu yra vadinamas *concurrency contol*.



Scheduler vaidmuo:

Jis paima read/write užklausas iš tranzakcijų ir arba jas vykdo buferiuose arba atideda.

Pagrindinis klausimas:

Kaip priversti tuo pat metu vykstančias tranzakcijas išlaikyti d.b.b. korektiškumą?

Pagrindiniai reikalavimai:

serializability ir *conflict-serializability*.

Serial and Serializable Schedules

Pagrindinė tema:

Metodų, priversiančių kelias tranzakcijas veikti vienu metu tik tais būdais, kai jos veiks po vieną kažkokiu tai laiko momentu, aptarimas.

Schedules

Schedule yra seka svarbių veiksmų paimtų iš vienos ar kelių tranzakcijų.

Nagrinėjant *concurrency control* pagrindiniai veiksmai read ir write vyksta ne diske, o pagrindinės atminties buferiuose. Taigi d.b. elementas **A**, tranzakcijos **T** patalpintas į buferį, gali būti perskaitytas arba perrašytas kitos tranzakcijos, kuri turi priėjimą prie **A**. Mes ignoruojame INPUT ir OUTPUT veiksmus, taigi tik read ir write yra svarbiausi nagrinėjant *concurrency control* .

Serial Schedules

Schedule bus *serial*, jei jo veiksmai susidės iš visų vienos tranzakcijos veiksmų, po to iš visų kitos tranzakcijos veiksmų ir t.t., nemaišant jų.

T1	T2
READ (A,t)	READ (A,s)
t : = t+100	s : = s*2
WRITE (A,t)	WRITE (A,s)
READ (B,t)	READ (B,s)
t : = t+100	s : = s*2
WRITE (B,t)	WRITE (B,s)

T1	T2	A	B
		25	25
READ (A,t)			
t : = t+100			
WRITE (A,t)		125	
READ (B,t)			
t : = t+100			
WRITE (B,t)			125
	READ (A,s)		
	s : = s*2		
	WRITE (A,s)	250	
	READ (B,s)		
	s : = s*2		
	WRITE (B,s)		250

Serializable Schedules

Tranzakcijų korektiškumo principas teigia: kiekvienas *serial schedule* išlaiko d.b.b. nuoseklumą.

Schedule bus *serializable*, jei jo rezultatas d.b.b. bus toks pats kaip ir *serial schedule*, nepaisant kokia pradinė d.b.b. bus.

T1	T2	A	B
		25	25
READ (A,t)			
t : = t+100			
WRITE (A,t)		125	
	READ (A,s)		
	s : = s*2		
	WRITE (A,s)	250	
READ (B,t)			
t : = t+100			
WRITE (B,t)			125
	READ (B,s)		
	s : = s*2		
	WRITE (B,s)		250

Serializable, bet ne *serial schedule*.

T1	T2	A	B
		25	25
READ (A,t)			
t : = t+100			
WRITE (A,t)		125	
	READ (A,s)		
	s : = s*2		
	WRITE (A,s)	250	
	READ (B,s)		
	s : = s*2		
	WRITE (B,s)		50
READ (B,t)			
t : = t+100			
WRITE (B,t)			150

Nonserializable sheduler.

The Effect of Transaction Semantic

Jau galime pasakyti ar *schedule* yra *serializable* ar ne iš jau išnagrinėtų tranzakcijų vykdomų operacijų smulkmenų.

T1	T2	A	B
		25	25
READ (A,t)			
t : = t+100			
WRITE (A,t)		125	
	READ (A,s)		
	s : = s*1		
	WRITE (A,s)	125	
	READ (B,s)		
	s : = s*1		
	WRITE (B,s)		25
READ (B,t)			
t : = t+100			
WRITE (B,t)			125

Serializable scheduler

A Notation for Transactions and Schedules

Pažymime:

- $\mathbf{r_i(X)}$ – read veiksmas, atliekamas tranzakcijos $\mathbf{T_i}$ d.b. elementui **X**.
- $\mathbf{w_i(X)}$ – write veiksmas, atliekamas tranzakcijos $\mathbf{T_i}$ d.b. elementui **X**.

T1	T2
READ (A,t)	READ (A,s)
t : = t+100	s : = s*2
WRITE (A,t)	WRITE (A,s)
READ (B,t)	READ (B,s)
t : = t+100	s : = s*2
WRITE (B,t)	WRITE (B,s)

T1: $\mathbf{r_1(A)}$; $\mathbf{w_1(A)}$; $\mathbf{r_1(B)}$; $\mathbf{w_1(B)}$;

T2: $\mathbf{r_2(A)}$; $\mathbf{w_2(A)}$; $\mathbf{r_2(B)}$; $\mathbf{w_2(B)}$;

- Veiksmais vadiname išraiškas $\mathbf{r_i(X)}$ ar $\mathbf{w_i(X)}$, kurios reiškia, kad tranzakcija $\mathbf{T_i}$ atlieka read arba write veiksmus su d.b. elementu **X**.
- Tranzakcija $\mathbf{T_i}$ vadiname seką veiksmų su indeksu **i**.
- *Schedule* **S** vadiname transakcijų aibės ***T*** seką veiksmų, kuris kiekvienai tranzakcijai $\mathbf{T_i}$ iš ***T***, bus toks pats, kaip ir seka veiksmų pačioje tranzakcijoje $\mathbf{T_i}$.

Conflict-Serializability

Nagrinėjame pakankamą sąlygą, kuri garantuoja, kad *schedule* yra *serializable*: *conflict-serializability*.

Ji paremta konflikto idėja: poros vienas po kito tvarkaraštyje einančių veiksmų tvarkos pakeitimas, gali pakeisti bent vienos iš tranzakcijų elgseną.

Conflicts

Veiksmų poros, kurios <u>nekonfliktuoja</u>:

•$\mathbf{r_i(X)}$;$\mathbf{r_j(Y)}$ – nėra konfliktas, net jei **X=Y**. Priežastis – nei vienas iš šių veiksmų nekeičia reikšmių.

•$\mathbf{r_i(X)}$;$\mathbf{w_j(Y)}$ - nėra konfliktas, su sąlyga **X≠Y**. Priežastis – ar $\mathbf{T_i}$ pirma perskaitys **X**, o po to $\mathbf{T_j}$ užrašys **Y**, ar atvirkščiai, reikšmes nepasikeis.

•$\mathbf{w_i(X)}$;$\mathbf{r_j(Y)}$ - nėra konfliktas, su sąlyga **X≠Y**.

•$\mathbf{w_i(X)}$;$\mathbf{w_j(Y)}$ - nėra konfliktas tol kol **X≠Y**.

Veiksmų poros, kurios <u>konfliktuoja</u>:

•Du tos pačios tranzakcijos veiksmai: $\mathbf{r_i(X)}$;$\mathbf{w_i(Y)}$ – konfliktas. Priežastis – atskirų tranzakcijų veiksmų eiga yra nekintama ir negali būti pertvarkoma DBVS.

•Du to pačio elemento rašymai su skirtingomis tranzakcijomis: $\mathbf{w_i(X)}$;$\mathbf{w_j(X)}$ – konfliktas. Priežastis - **X** reikšmė pasilieka tokia, kokią suskaičiuoja tranzakcija $\mathbf{T_j}$.

•Skaitymas ir rašymas to paties elemento skirtingomis tranzakcijomis: $\mathbf{r_i(X)}$;$\mathbf{w_i(X)}$ ar $\mathbf{w_i(X)}$;$\mathbf{r_i(X)}$ – konfliktas.

Išvados: Skirtingų tranzakcijų du veiksmai gali būti keičiami vietomis, nebent:

- Jie siejasi su tuo pačiu d.b. elementu.
- Bent vienas iš veiksmų yra rašymo.

Sakysime, kad du tvarkaraščiai yra *conflict – equivalent*, jei vienas iš jų gali būti pervestas į kitą seka nekonfliktinių gretimų veiksmų pakeitimų.

Tvarkaraštį vadinsime *conflict – serializable*, jei jis yra *conflict – equivalent serial* tvarkaraščiui.

Conflict – serializability yra pakankama *serilaizability* sąlyga.

Precedence Graphs and a Test for Conflict – Serializability

Turime tvarkaraštį **S**, siejantį tranzakcijas $\mathbf{T_1}$ ir $\mathbf{T_2}$: sakysime, kad $\mathbf{T_1}$ yra prieš $\mathbf{T_2}$: $\mathbf{T_1 <_S T_2}$, jei veiksmai $\mathbf{A_1}$ iš $\mathbf{T_1}$ ir $\mathbf{A_2}$ iš $\mathbf{T_2}$ tokie, kad:

- Veiksmas $\mathbf{A_1}$ yra prieš $\mathbf{A_2}$ tvarkaraštyje **S**
- Abu $\mathbf{A_1}$ ir $\mathbf{A_2}$ sieja tas pats elementas
- Bent vienas $\mathbf{A_1}$ ar $\mathbf{A_2}$ yra rašymo veiksmas.

Pastebėkime, kad tai yra tos sąlygos, pagal kurias mes negalime keisti $\mathbf{A_1}$ ir $\mathbf{A_2}$ tvarkos. Taigi $\mathbf{A_1}$ bus prieš $\mathbf{A_2}$ bet kokiame tvarkaraštyje, kuris yra *conflict – equivalent* tvarkaraščiui **S**. Jei bent vienas iš tvarkaraščių yra *serial* tai $\mathbf{T_1}$ turi būti prieš $\mathbf{T_2}$.

Visa tai galime vaizduoti protėvių grafu: tvarkaraščio **S** tranzakcijos bus grafo viršūnėmis ($\mathbf{T_i}$-**i**); iš viršūnės **i** į viršūnę **j** briauna bus tada, kai $\mathbf{T_1 <_S T_2}$.

Taisyklė, pagal kurią pasakysime ar **S** yra *conflict – serializable*: sukonstruojame protėvių grafą ir pažiūrime ar jame yra ciklų.

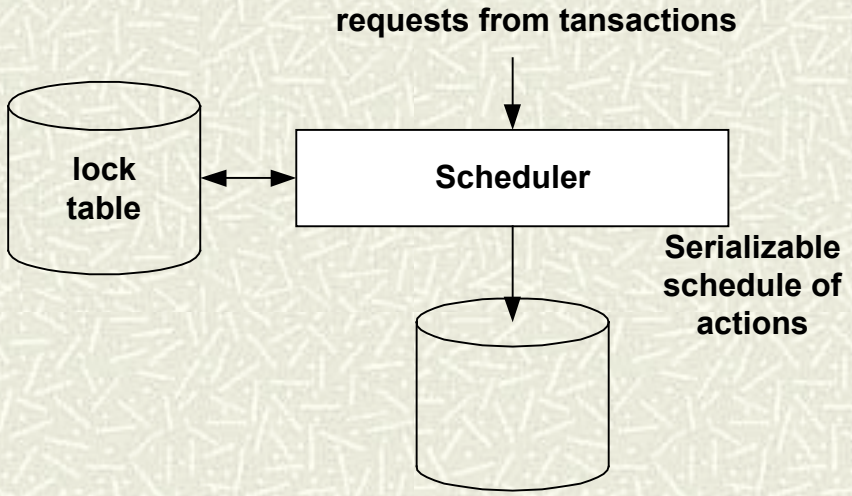
Jei taip – **S** nėra *conflict – serializable*; jei ne – **S** yra *conflict – serializable* ir viršūnių topologinė tvarka yra *conflict – equivalent* **S** tvarkai.

Enforcing Serializability by Locks

Tarkime turime kažkokį tai rinkinį tranzakcijų, kurios atlieka veiksmus kažkokia nustatyta tvarka. Veiksmai formuoja tvarkaraštį, bet vargu ar jis *serializable*. Taigi tvarkaraščio darbas yra neleisti veiksmų, kurie formuoja ne *serializable* tvarkaraštį.

Tranzakcijos naudoja užraktus tam, kad keliom tranzakcijom nebūtų galima dirbti su tuo pačiu d.b. elementu ir tuo būdu mažina ne *serializability* riziką.

Locks



Tvarkaraštis naudoja užraktų lentelę.

Tvarkaraštis leis vykdyti užklausą, jei tas vykdymas nesukels d.b.b. nenuoseklumo.

Užraktai turi būti naudojami dviem prasmėmis: tranzakcijų struktūroms, tvarkaraščių struktūroms.

- Tranzakcijų nuoseklumas. Veiksmai ir užraktai siejami tokiais būdais:
- Tranzakcija gali rašyti arba skaityti elementą tik tada, jei buvo užklausa užraktui ir ji dar nenuimta.
- Jei tranzakcija užrakina elementą, tai vėliau jį turi būtinai atrakinti.

• Tvarkaraščių legalumas. Dvi tranzakcijos negali užrakinti to paties elemento, prieš tai vienai iš jų neatrakinus jo.

Pažymime:

$\mathbf{l_i(X)}$ – tranzakcija $\mathbf{T_i}$ užrakina **X**

$\mathbf{u_i(X)}$ – tranzakcija $\mathbf{T_i}$ atrakina **X**

Tranzakcijų nuoseklumo sąlyga: Jei tik $\mathbf{T_i}$ turi veiksmus $\mathbf{r_i(X)}$ ar $\mathbf{w_i(X)}$, tai prieš tai turi būti veiksmas $\mathbf{l_i(X)}$, o tik po to $\mathbf{u_i(X)}$.

Tvarkaraščių legalumo sąlyga: Jei yra veiksmai $\mathbf{l_i(X)}$ ir $\mathbf{l_j(X)}$, tai tarp jų turi būtinai būti $\mathbf{u_i(X)}$.

The Locking Scheduler

Scheduler darbas, peremtas užraktais, turi patenkinti užklausas tada ir tik tada jei ją suvykdžius gausime legalų tvarkaraštį.

Yra naudojama užraktų lentelė, kaip sąryšis:

Locks (element , transaction),

susidedanti iš porų **(X,T)** tokių, kad tranzakcija **T** turi užraktą **X**-ui.

Two – Phase Locking

Yra žinoma sąlyga, pagal kurią mes galime garantuoti, kad nuoseklių tranzakcijų legalus tvarkaraštis yra *conflict – serializable*. Ši sąlyga yra vadinama *two-phase locking* arba 2PL:

•Kiekvienoje tranzakcijoje visos užraktų užklausos eina prieš atrakinimų užklausas.

Tranzakcijos, kurioms galioja 2PL, vadinamos *two-phase locking* tranzakcijomis.

2PL: pirma fazė – užraktų galiojimas,

antra fazė – užraktų atsisakymas.

Locking Systems with Several Lock Modes

Pagrindinė problema:

Tranzakcija **T** norėdama tik perskaityti, bet neužrašyti elementą **X** turi vis tiek užrakinti **X**. Iš kitos pusės, nėra priežasties, kodėl keletas tranzakcijų negalėtų skaityti **X** tuo pat metu, iki tol kol nedaromas užrašymas.

Todėl yra naudojami skirtingi užraktai:

- *Shared lock* – užraktas skaitymui.
- *Exclusive lock* – užraktas rašymui.

Shared and Exclusive Locks

Užraktas rašymui yra “stipresnis” nei užraktas skaitymui. Taigi kiekvienam d.b. elementui **X** gali būti vienas rašymo užraktas, arba jokio rašymo užrakto, bet keletas skaitymo užraktų.

Pažymime:

$sl_i(X)$ – tranzakcija T_i reikalauja skaitymo užrakto **X**-ui.

$xl_i(X)$ - tranzakcija T_i reikalauja rašymo užrakto **X**-ui.

Trys reikalavimai: tranzakcijų nuoseklumas, 2PL tranzakcijos ir tvarkaraščių legalumas.

- Tranzakcijų nuoseklumas. Negalima atlikti rašymo veiksmo elementui, neturint ant jo rašymo užrakto, taip pat ir su skaitymu. Detaliau, kiekvienoje tranzakcijoje $\mathbf{T_i}$:

  •Skaitymo veiksmas $\mathbf{r_i(X)}$ turi eiti po $\mathbf{sl_i(X)}$ arba $\mathbf{xl_i(X)}$, neįsiterpiant $\mathbf{u_i(X)}$.

  •Rašymo veiksmas $\mathbf{w_i(X)}$ turi eiti po $\mathbf{xl_i(X)}$, neįsiterpiant $\mathbf{u_i(X)}$.

  Visi užrakinimai turi eiti po to paties elemento atrakinimų.

- Tranzakcijų 2PL. Užrakinimas turi eiti prieš atrakinimą. Detaliau, 2PL tranzakcijose, $\mathbf{u_i(X)}$ negali eiti prieš $\mathbf{sl_i(X)}$ arba $\mathbf{xl_i(X)}$.
- Tvarkaraščių legalumas. Elementas gali būti užrakinamas su rašymo užraktu vienos tranzakcijos arba keliomis tranzakcijomis skaitymo užraktais, bet kartu ne. Detaliau:

  •Jei $\mathbf{xl_i(X)}$ yra sąraše, tai po to negali būti $\mathbf{xl_j(X)}$ arba $\mathbf{sl_j(X)}$, kažkokiam **j**, be $\mathbf{u_i(X)}$ įsiterpimo.

  •Jei $\mathbf{sl_i(X)}$ yra sąraše, tai po to negali būti $\mathbf{xl_j(X)}$, $\mathbf{j \neq i}$, be isiterpimo $\mathbf{u_i(X)}$.

Compatibility Matrices

Pristatysime sekančius pastebėjimus, apsprendžiančius užraktų suteikimo politiką, paprastoje skaitymo / rašymo sistemoje.

Suderinamumo matrica tai matrica sudaryta iš eilutės ir stulpelio kiekvienam užraktų tipui. Eilutės yra sutapatinamos su užraktais, kurie jau yra sutekti **X**, o stulpeliai – pareikalautais užraktais.

		Lock requested	
		S	**X**
Lock held in mode	**S**	Yes	No
	X	No	No

Suderinamumo matricos taikymo taisyklė užraktų reikalavimo problemoje:

•Mes galime suteikti **C** tipo užraktą tada ir tik tada, jei kiekvienai eilutei **R**, tokiai, kurioje jau yra **X**-ui suteiktas **R** tipo užraktas, yra “taip” **C** stulpelyje.

Upgrading Locks

Tranzakcija **T**, kuri turi skaitymo užraktą **X**-ui yra “draugiška” su kitomis tranzakcijomis, tol kol jos tik nuskaitinėja tą patį elementą **X**.

Galime domėtis ar ji taip pat bus “draugiška” ir tada, kai **T** norės ne tik nuskaityti, bet ir užrašyti naują **X** reikšmę. Tranzakcija **T** pirma gali **X** užrakinti su skaitymo užraktu, o tik po to jį *upgrade* iki rašymo užrakto.

Update Locks

Kad išvengti aklaviečių problemos yra naudojami trečio tipo užraktai: *update locks*.

Šis užraktas $\mathbf{ul_i(X)}$ leidžia tranzakcijai $\mathbf{T_i}$ tik skaityti **X**, bet ne rašyti. Tačiau tik *update* užraktas gali būti pakeistas į rašymo, į skaitymo – ne.

Naudojame šį užraktą tada, kai ant **X** yra uždėtas skaitymo užraktas, bet jei **X** turi jau *update* užraktą, tai mes uždraudžiame papildomus užraktus: skaitymo, rašymo ar *update*.

	S	**X**	**U**
S	Yes	No	Yes
X	No	No	No
U	No	No	No

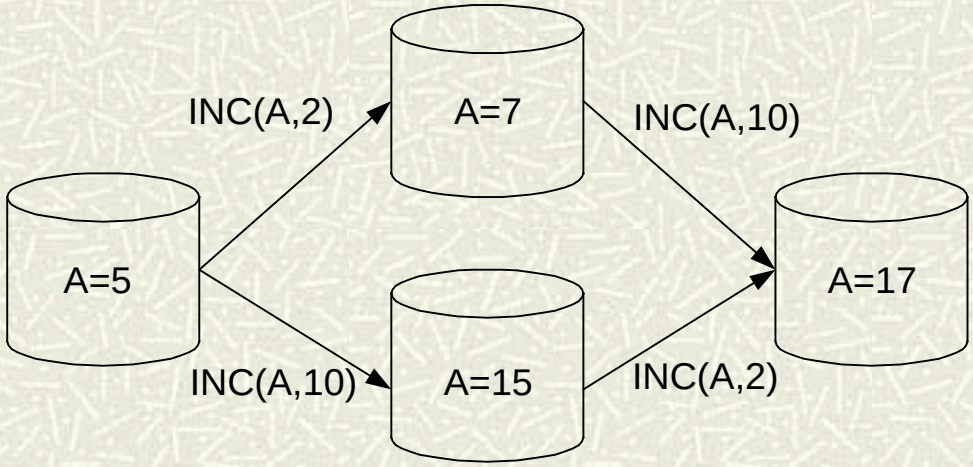
Update užraktas bus kaip *shared* užraktas, kai mes jo užklausiame, ir kaip *exclusive*, kai mes jį jau turime.

Increment Locks

Pavyzdžiai:

•Tranzakcija, kuri perveda iš vienos banko sąskaitos pinigus į kitą sąskaitą.

•Tranzakcija, kuri parduoda lėktuvo bilietus ir mažina laisvų vietų skaičių lėktuve.



Increment veiksmai INC(A,c) komutuoja.

Formaliai **INC(A,c)** reiškia:

READ(A,t) ; t := t + c ; WRITE(A,t) ;

Increment veiksmui mums reikia *increment* užrakto.

Pažymime:

- **$il_i(X)$** – tranzakcija T_i reikalauja *increment* užrakto **X**-ui.
- **$inc_i(X)$** – tranzakcija T_i padidina **X** reikšmę kažkokia konstanta.

Increment veiksmų egzistavimas ir užraktų pareikalavimas reikalauja įvesti tam tikrus pastebėjimus tranzakcijų nuoseklumui, konfliktams ir legaliems tvarkaraščiams.

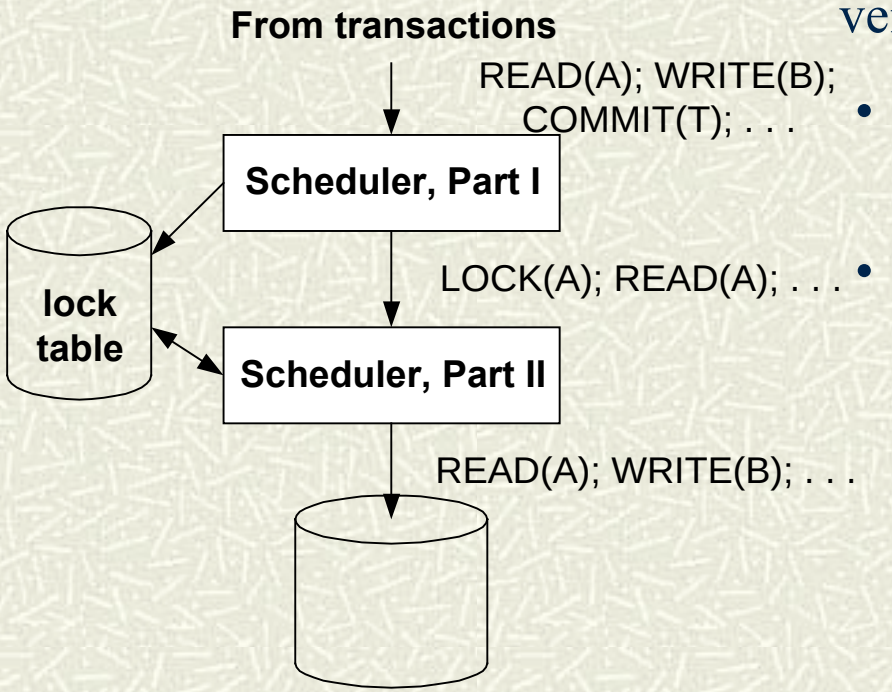
- Nuosekli tranzakcija gali atlikti su **X** *increment* veiksmą, jei ji turi *increment* užraktą ant **X**. *Increment* užraktas neleidžia skaitymo ir rašymo veiksmų.
- Legaliame tvarkaraštyje bet koks kiekis tranzakcijų gali turėti ant **X** *increment* užraktą.
- Veiksmas **$inc_i(X)$** konfliktuoja su **$r_j(X)$** ir **$w_j(X)$**, kai **$j \neq i$**, bet nekonfliktuoja su **$inc_j(X)$**.

	S	**X**	**I**
S	Yes	No	No
X	No	No	No
I	No	No	Yes

An Architecture for a Locking Scheduler

- Tranzakcijos pačios nereikalauja užraktų, joms tai ir nepriklauso. *Scheduler* darbas yra įterpti užrakinimo veiksmus į srautą skaitymo, rašymo ir kt. veiksmų.
- Tranzakcijos nerealizuoja užraktų. *Scheduler* realizuoja užraktus, kai tranzakcijų valdytojas pasako ar tranzakcija nutraukiama ar vykdoma.

A Scheduler That Inserts Lock Actions



Dviejų dalių *scheduler* atlieka tokius veiksmus:

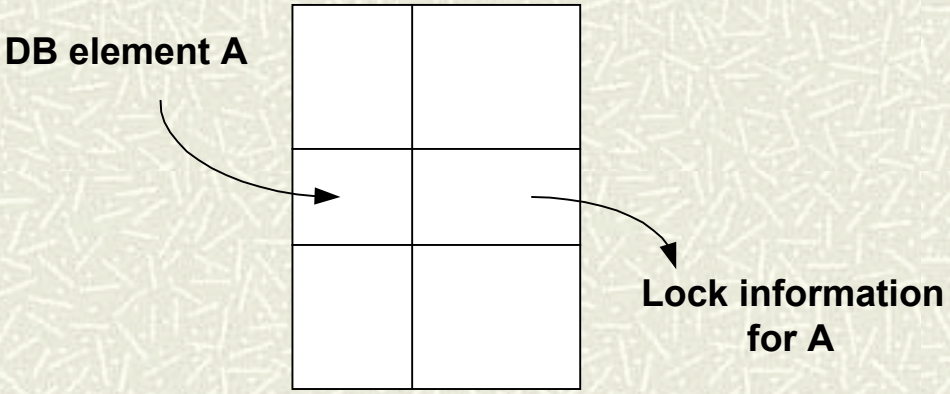
- **Part I** – įterpia užraktų veiksmus prieš visus kitus veiksmus.
- **Part II** – vykdo visus veiksmus. Jei tranzakcija T nesulaikyta tai
  - jei veiksmas yra leidžiamas d.b., tai jis persiunčiamas d.b. ir vykdomas.
  - jei užrakinimo veiksmas yra gaunamas iš **Part II**, tai yra tikrinama užraktų lentelė ir žiūrima ar užraktas yra leidžiamas.

❖Jei taip, užraktų lentelė yra patobulinama, įrašant užraktą kuris ką tik buvo leidžiamas.

❖Jei ne, tai užraktų lentelėje turi būti padarytas įrašas, kad užraktas buvo pareikalautas.

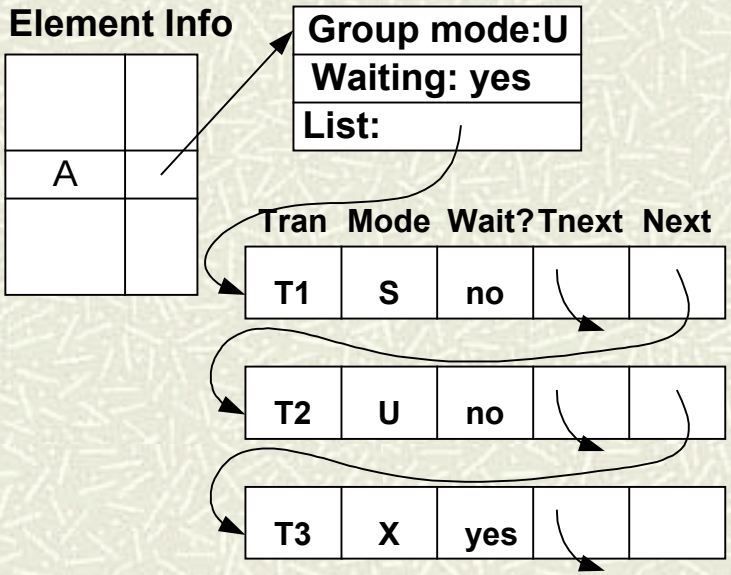
- Kai tranzakcija *commits* arba *aborts*, tranzakcijų tvarkytojas praneša **Part I** ir visi užraktai, kurie buvo uždėti **X**-ui yra realizuojami. Jei kokia tranzakcija laukia šių užraktų, **Part I** praneša **Part II**.
- Kai **Part II** pamato, kad kažkos užraktas yra galimas **X**-ui, ji nutraukia kitų tranzakcijų darbą ir suteikia **X**-ui užraktą.Tranzakcijos, kurioms yra suteikiamas užraktas, gali atlikti tiek veiksmų, kiek jų buvo atidėtų iki tol kol atliks visus veiksmus, arba pasieks kita užrakto užklausą, kuri nėra leidžiama.

The Lock Table



Užrakinimo lentelė - sąryšis tarp d.b. elemento ir informacijos apie jo užrakinimą.

Bet koks elementas, kuris nėra užrakintas neatsiduria šioje lentelėje, taigi lentelės dydis yra proporcingas užrakintų elementų skaičiui, bet ne visų d.b. elementų skaičiui.



Šiame pavyzdyje yra naudojama *shared-exclusive-update* schema.

Elementas **A** yra *tuple* su sekančiomis komponentėmis:

- *Group mode* yra santrauka stipriausių sąlygų tranzakcijai prašant užrakto ant **A**. SXU schemai taisyklė yra paprasta: *group mode*

  ❖**S** reiškia, kad yra uždėti tik skaitymo užraktai.

  ❖**U** reiškia, kad yra vienas *update* užraktas ir vienas ar keli skaitymo užraktai.

  ❖**X** reiškia, kad yra vienas rašymo užraktas ir jokių kitų.

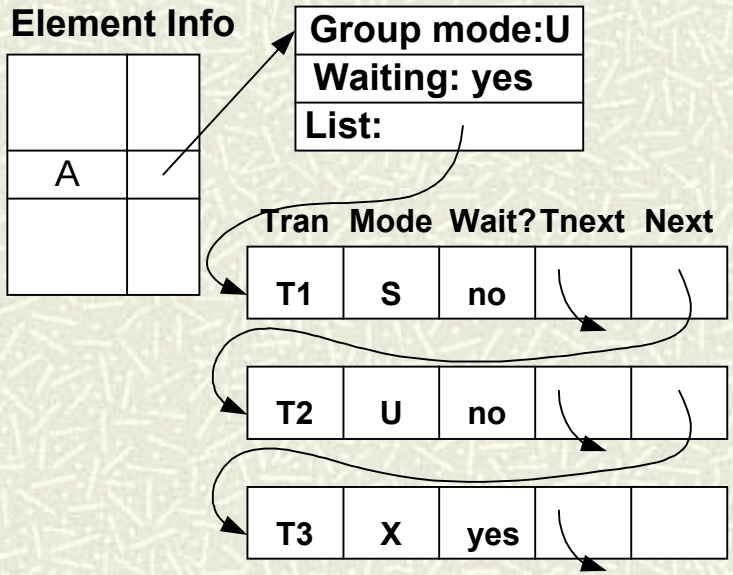
- *Waiting* bitas sako, kad yra bent viena tranzakcija, laukianti užrakto elementui **A**.
- Sąrašas aprašantis visas tranzakcijas, kurios laiko užraktus ant **A** ar dar tik laukia kada galės uždėti užraktus. Kiekvienas sąrašas susideda iš:

  ❖Tranzakcijos, kuri laiko ar laukia užrakto, vardo.

  ❖Užrakto tipo.

  ❖Informacijos ar tranzakcija laukia ar laiko užraktą.

Handling Lock Requests



Tarkime tranzakcija **T** nori uždėti užraktą ant elemento **A**.

Jei užraktų lentėlėje nėra jokio įrašo apie **A**, tai reiškia, kad jokių užraktų **A** neturi, taigi įrašas yra sukuriamas ir užklausa leidžiama.

Jei užraktų lentelėje yra įrašai apie **A** užraktus, tai žiūrime į *group mode* – **U**. Kadangi ant **A** yra *update* užraktas, tai joks kitoks užraktas nėra leidžiamas. Tai reiškia, kad užklausa apie **A** užraktą yra atidedama ir tada matome: Wait? =“yes”.

Tas pats atsitiks ir su group mode **X**, bet su **S** bus kitaip. **S** gali būti pakeistas kitu skaitymo užraktu arba *update* užraktu. Taigi matysime: Wait? =“no” ir pakeisime į kitą užraktą.

Handling Unlocks

Tarkime tranzakcija **T** atrakino **A**. Taigi **T** įrašas iš sąrašo yra ištrinamas, jei užraktas buvo ne toks pats kaip *group mode*, jei toks pats – tai turime rasti naują *group mode*. Jei *group mode* buvo:

- **U** – tai kita *group mode* gali būti **S**, jei tokių užraktų dar yra.
- **X** – tai nėra kitų užraktų.
- **S** – tai reikia patikrinti ar nėra kitų skaitymo užraktų.

Jei *Waiting* reikšmė yra "yes", tai mes turime leisti vieną ar kelis užraktus iš užklaustų užraktų sąrašo.

- First-come-first-served: leidžiamas užraktas, kurio ilgiausiai laukėme, tuo būdu nebus badavimo.
- Pirmenybė skaitymo užraktams:Pirmiausiai leidžiame visus laukiamus skaitymo užraktus, po to vieną *update* ir rašymo, jei kitų jau nebėra.
- Pirmenybė upgrade užraktams:Pirmiausiai leidžiame *ugrade* užraktus.

Managing Hierarchies of Database Elements

Grįžtame prie skirtingų užraktų schemos ir visą dėmesį sutelksime į dvi problemas, kai mūsų duomenys yra medžio struktūros.

- Pirma medžio struktūros rūšis, su kuria susidūrėme yra užrakinamų elementų hierarchija. Kaip suteikti užraktus dideliems ir mažesniems elementams.
- Antra, svarbi *concurency-control* sistemoms yra duomenys, kurie patys konstruoja medį (B –medžio indeksai).

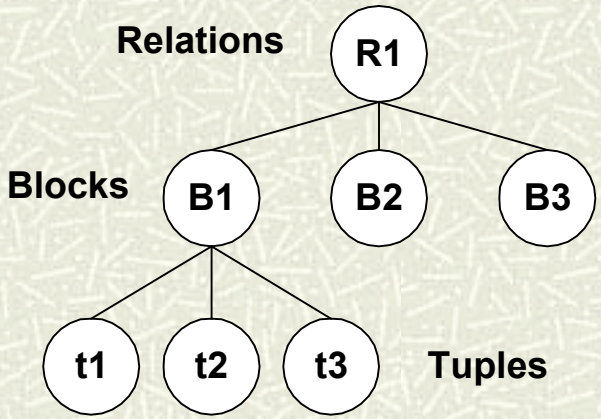
Locks With Multiple Granularity

Skirtingos sistemos naudoja skirtingo dydžio d.b. elementus užrakinimui: puslapiai, blokai, sąryšiai, tuples.

Kai kada naudinga dirbti su didesniais, kai kada su mažesniais. Tam yra įvedamas užraktų smulkinimas.

Warning Locks

Įvedame naują užraktą: "*Warning*", skirtingo grūdėtumo užraktų valdymo problemai spręsti. Šie užraktai yra naudingi hierarchinei struktūrai.



Kaip matome yra trys d.b.elementų lygiai:

- Ryšiai – didžiausi.
- Kiekvienas ryšys sudarytas iš vieno ar kelių blokų ar puslapių.
- Kiekvienas blokas susideda iš vienos ar kelių *tuples*.

Warning protokolas sieja ir įprastus, ir "warning" užraktus.

Nagrinėsim schemą su **S** ir **X** užraktais. Įspėjamieji užraktai bus žymimi su **I** raide pradžioje: **IS** reiškia ketinimą suteikti skaitymo užraktą subelementui.

Warning protokolo taisyklės:

- Turime pradėti nuo hierarchijos šaknies, jei norime suteikti įprastus **S** ar **X** užraktus bet kokiam elementui.
- Jei mes esame prie elemento, kurį norime užrakinti, tai nėra reikalo žiūrėti į tolį. Mes suteikiame elementui užraktą.
- Jei elementas yra hierarchija žemiau, tai mes ant jo viršūnės uždedame "warning".

	IS	IX	S	X
IS	Yes	Yes	Yes	No
IX	Yes	Yes	No	No
S	Yes	No	Yes	No
X	No	No	No	No

Phantoms and Handling Insertions Correctly

Kai tranzakcijos sukuria naują užrakinamo elemento sublelementą, visada yra galimybė suklysti.

Tarkim viena tranzakcija turi surasti visus filmus Disney kino kompanijos, o kita įterpti dar vieną Disney filmą į d.b. Kadangi antroji tranzakcija norėdama įrašyti turi turėti **X** užraktą ant to pačio ryšio, o pirmoji tranzakcija užrakinus ryšį **IS** užraktu, ir jis yra nesuderinamas su **X**, tai antroji tranzakcija turės palaukti kol pirmoji baigsis.

The Tree Protocol

Kalbėsime apie medžių struktūras, kurios yra suformuojamos jungčių struktūromis iš pačių elementų.

D.b. elementai yra atskiri duomenų gabaliukai, bet vienintelis kelias patekti į viršūnę yra per jos protėvius(B-medžiai).

Žinojimas, kad mes turime pereiti konkretų kelią iki elemento, leidžia mums laisvai tvarkyti užraktus, skirtingai nuo *2PL*.

Rules for Access to Tree-Structured Data

Sekantys užraktų apribojimai formuoja medžio protokolą. Tarkime, yra tik vienos rūšies užraktas $\mathbf{l_i(X)}$, bet idėja yra taikoma bet kokiam užraktų tipų rinkiniui. Mes teigiame, kad tranzakcios yra nuoseklios ir sąrašai yra legalūs, bet tranzakcijoms negalioja *2PL*.

- Pirmas tranzakcijos užraktas gali būti ant bet kokios viršūnės.
- Tolesni užraktai gali būti dedami, tik toms viršūnėms, kurių tėvų viršūnės jau turi užraktus.
- Viršūnės turi būti atrakinamos bet kada.
- Tranzakcijos gali iš naujo neužrakinti viršūnės, nors jos ką tik realizavo užraktą, net jei jos tėvo viršūnė užrakinta.

Concurrency Control by Timestamps

Toliau nagrinėsime du kitokius nei užrakinimas metodus, garantuojančius tranzakcijų *serializability*.

- *Timestamping*.Priskirsime kiekvienai tranzakcijai laiko ženklą, reiškiantį kiek laiko reikia, kad būtų atlikti skaitymo ir rašymo veiksmai su kiekvienu d.b.elementu ir palyginsime šias reikšmes, kad garantuotume *serial schedule* pagal tranzakcijų laiko ženklus yra ekvivalenti dabartiniam tranzakcijų tvarkaraščiui.
- *Validation*. Nagrinėsime tranzakcijų laiko ženklus ir kai tranzakcija yra beveik baigiama, tai šis procesas vadinsis tranzakcijų "įtvirtinimas". *Serial* tvarkaraštis, kuris tvarko tranzakcijas pagal jų tvirtinimo laiką turi būti ekvivalentus dabartiniam tvarkaraščiui.

Timestamps

Kad naudoti laiko ženklinimą kaip *concurrency-control* metodą, kiekvienai tranzakcijai **T** *scheduler* turi priskirti skaičių, laiko ženklą **TS(T).**

Yra du laiko ženklų sukūrimo būdai:

- Naudoti sistemos laikrodį, nes *scheduler* nesugebės dirbti taip grietai, kad sukurs laiko ženklus dviem tranzakcijos tuo pat laikrodžio tikstelejimu.
- Įvesti skaitiklį. Tranzakcijos, kurios prasidėjo vėliau turi didesnius laiko ženklus, nei tos kurios prasidėjo anksčiau – svarbi savybė reikalinga laiko ženklų generavimo sistemai.

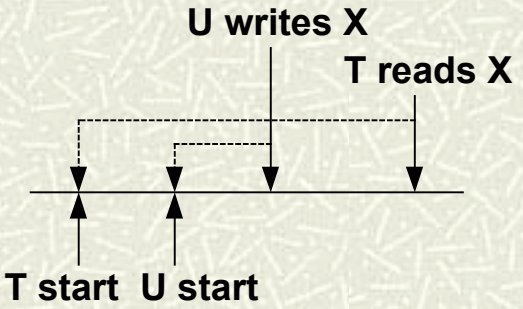
Kad naudoti laiko ženklinimą kaip *concurrency-control* metodą, su kiekvienu d.b.elementu **X** turime turėti 2 laiko ženklus ir papildomą bitą:

- **RT(X), X** skaitymo laikas, didžiausias iš tranzakcijų, kurios skaitė **X**, laiko ženklų.
- **WT(X), X** rašymo laikas, didžiausias iš tranzakcijų, kurios rašė **X**, laiko ženklų.
- **C(X), X** *commit bit*, kuris yra true, jei paskutinė tranzakcija, kuri atliko rašymą yra *committed*. Norima apsidrausti nuo atvejų, kai viena tranzakcija nuskaitinėja duomenis užrašytus kitos tranzakcijos U, ir tada tranzakcija U sustabdoma. Ši problema, kai T padaro "purviną skaitymą" ne *committed* duomenų, gali sukelti d.b.b. nenuoseklumą.

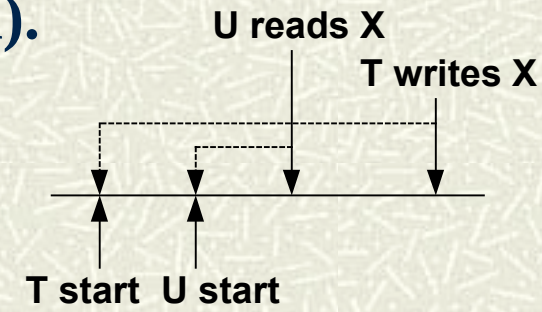
Physically Unrealizable Bechaviors

Du tipai problemų:

- Read too late. **TS(T)<WT(X).**



- Write too late. **WT(X)<TS(T)<RT(X).**



The Rules for Timestamp-Based Scheduling

Scheduler, atsakydamas į tranzakcijos **T** skaitymo ar rašymo užklausą turi pasirinkti:

•Leisti užklausą,

•Nutraukti **T** ir atnaujinti **T** su nauju laiko ženklu, arba

•Atidėti **T** ir tik vėliau nuspręsti nutraukti **T** ar vykdyti užklausą.

Taisyklės yra sekančios:

- Tarkime *scheduler* gauna užklausą $\mathbf{r_T(X)}$.

  ☐Jei **TS(T)>=WT(X),** skaitymas fiziškai realizuojamas.

  •Jei **C(X)** yra true, ledžiama užklausa. Jei **TS(T)>RT(X),** tai **TS(T):=RT(X),** kitu atveju **RT(X)** nesikeičia.

  •Jei **C(X)** false, atidedama **T** iki **C(X)** taps true, arba tranzakcija, kuri rašo **X** bus nutraukta.

☐ Jei **TS(T)<WT(X),** skaitymas fiziškai ne realizuojamas. *Rollback* **T**; nutraukti **T** ir atnaujinti su nauju, didesniu laiko ženklu.

- Tarkime *scheduler* gauna užklausą $\mathbf{w_T(X)}$.

  ☐Jei **TS(T)>=RT(X)** ir **TS(T)>=WT(X),** rašymas fiziškai realizuojamas ir turi buti atlikta.

  - Užrašyti naują **X** reikšmę,
  - **WS(X):=TS(T)** ir
  - **C(X):=false**.

  ☐Jei **TS(T)>=RT(X),** bet **TS(T)<WT(X)**, rašymas yra fiziškai realizuojamas, bet turime vėlesnę **X** reikšmę. Jei **C(X)** yra true, tada prieš tai einantis **X** rašytojas yra *committed*, ir mes ignoruojame **T** rašymą; mes ledžiame **T** tęsti ir nepadaryti jokių d.b. pakeitimų. Kai **C(X)** yra false – turime atidėti **T**.

  ☐Jei **TS(T)<RT(X),** tai rašymas fiziškai nerealizuojamas. *Rollback* **T**.

- Tarkime tvarkaraštis gauna užklausą *commit* **T**. Jis turi surasti visus d.b.elementus **X** užrašytus tranzakcijos **T** ir priskirti **C(X):=true.**
- Tarkime tvarkaraštis gauna užklausą nutraukti **T** arba *rollback* **T**. Tai bet kokia transakcija, kuri laukė **X**-o, kurį užrašė **T**, turi pakartoti rašymą ar skaitymą ir pažiūrėti ar dabar veiksmas yra legalus.

Multiversion Timestamps

Praktikoje naudojama bendra technika tik skaitymo tranzakcijoms *scheduled* laiko ženklų, bet su sudėtiniais variantais, kur rašymas ne perrašo ankstesnės elemento reikšmės, kol visos tranzakcijos, kurioms galbūt bus reikalinga ankstesnė reikšmė dar nebaigtos. Rašymo tranzakcijos yra *scheduled* paprastais užraktais

Timestamps and Locking

Laiko žymymėjimas yra geresnis tose situacijose, kai dauguma tranzakcijų yra tik skaitymo. *High-conflict* situacijose yra geriau naudoti užraktus.

Argumentai šiai *rule-of-thumb*:

•Užrakinimai dažnai atideda tranzakcijas, net susidaro aklavietės, kai kelios tranzakcijos laukia labai ilgai ir tuo būdu viena iš jų turi būti *rollback*.

•Bet jei persikertančios tranzakcijos dažnai skaito ir rašo elementus bendrai, tada *rollbacks* bus dažni, ir duos dar daugiau aklaviečių.

Read-only – vykdomos su *multiversion timestamping'u*.

Read/write – su *2PL*.

Concurrency Control by Validation

Nagrinėjant *validation*, kitaip nei *timestamping*, tvarkaraštis išlaiko įrašus apie aktyvių tranzakcijų veiklą, o ne laiko read ir write laikus kiekvienam d.b.elementui.

Prieš tai kai tranzakcija pradeda rašyti d.b.elementus, ji pereina per "*validation phase*", kur elementų, kuriuos perskaitė, ir kuriuos rašys, rinkinys yra lyginamas su kitų aktyvių tranzakcijų rašymo rinkiniais. Jei tik atsiranda rizika, kad turėsime fiziškai nerealizuojamą elgesį, tai tranzakcija yra *rolled back*.

Architecture of a Validation-Based Scheduler

Tranzakcijos yra vykdomos trimis fazėmis:

- *Read*. Tranzakcija nuskaito visus d.b.elementus iš skaitymo rinkinio.
- *Validate*. Lyginami tranzakcijų skaitymo ir rašymo rinkiniai su kitų tranzakcijų rinkiniais.
- *Write*. Užrašo d.b.elementų reikšmes rašymo rinkiniuose.

Aptarnaujami trys rinkiniai:

- **START**. Tranzakcijų, kurios prasidėjo, bet dar nepasibaigė validacija, rinkinys.
- **VAL**. Tranzakcijų, kurios jau validavosi, bet dar ne užbaigė trečios rašymo fazės, rinkinys.
- **FIN**. Tranzakcijų, kurios jau pabaigė trečią fazę, rinkinys.

Comparision of Three Concurrency-Control Mechanism

Atminties panaudojimas:

✓ *Locks*. Užraktų lentelė užima tiek vietos, kiek yra užrakinamų d.b.elementų.

✓ *Timestamps*. Tiek pat kiek ir užraktai.

✓ *Validation*. Reikalauja vietos kiekvienos aktyvios tranzakcijos laiko ženklams ir read/write rinkiniams, plius dar keletas tranzakcijų, kurios baigėsi, kai aktyviosios tranzakcijos prasidėjo.

Taigi kiekvienas mechanizmas užima tiek vietos, kiek yra aktyvių d.b.elementų tranzakcijų. *Timestamps* ir *validation* – truputį daugiau.