

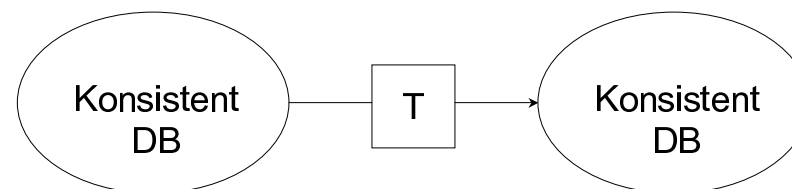
Transaksjonshåndtering og samtidigheitskontroll

Ragnar Normann

Mange lysark er basert på en original laget av
Hector Garcia-Molina

Transaksjoner

En **transaksjon** er en samling aksjoner
som bevarer konsistens i databasen



ACID-egenskapene

- A – Atomicity
 - enten blir hele transaksjonen utført
 - eller så blir ikke noe av den utført
- C – Consistency
 - transaksjoner skal bevare konsistens (dette er en del av definisjonen av begrepet transaksjon)
- I – Isolation
 - transaksjoner skal ikke merke at andre transaksjoner utføres samtidig med dem selv
- D – Durability
 - når transaksjoner er avsluttet, skal effekten av dem være varig og ikke kunne påvirkes av systemfeil

Eksekveringsplaner

- Operasjonene i en transaksjon er **ordnet**
Det vil si at en transaksjon er en liste av operasjoner
- Gitt en mengde transaksjoner $\{T_1, \dots, T_n\}$
En **eksekveringsplan** (schedule) S for $\{T_1, \dots, T_n\}$ er en sekvens (liste) av operasjoner slik at:
 - hvert element i S er en operasjon i en av transaksjonene
 - hver operasjon i hver transaksjon er element i S nøyaktig en gang
 - hvis o_p og o_q er to operasjoner i samme transaksjon T_k , og o_p kommer foran o_q i T_k , så kommer o_p foran o_q i S (Det vil si at S bevarer ordningen i transaksjonene)

Transaksjonseksempel

Integritetsregel: $A = B$

T1: Read(A);	T2: Read(A);
A \leftarrow A+100;	A \leftarrow A \times 2;
Write(A);	Write(A);
Read(B);	Read(B);
B \leftarrow B+100;	B \leftarrow B \times 2;
Write(B);	Write(B);

Eksekveringsplan S_A

		A	B
T1	T2	25	25
Read(A); A \leftarrow A+100;			
Write(A);		125	
Read(B); B \leftarrow B+100;			
Write(B);			125
	Read(A); A \leftarrow A \times 2;		
	Write(A);	250	
	Read(B); B \leftarrow B \times 2;		
	Write(B);		250
		250	250

Eksekveringsplan S_B

		A	B
T1	T2	25	25
	Read(A); A \leftarrow A \times 2;		
	Write(A);	50	
	Read(B); B \leftarrow B \times 2;		
	Write(B);		50
Read(A); A \leftarrow A+100;			
Write(A);			
Read(B); B \leftarrow B+100;		150	
Write(B);			
			150
		150	150

S_A og S_B er **serielle** planer – de tar en transaksjon om gangen

Eksekveringsplan S_C

		A	B
T1	T2	25	25
Read(A); A \leftarrow A+100;			
Write(A);		125	
	Read(A); A \leftarrow A \times 2;		
	Write(A);	250	
Read(B); B \leftarrow B+100;			
Write(B);			125
	Read(B); B \leftarrow B \times 2;		
	Write(B);		250
		250	250

Eksekveringsplan S_D

		A	B
T1	T2	25	25
Read(A); A ← A+100;			
Write(A);		125	
	Read(A); A ← A×2;		
	Write(A);	250	
	Read(B); B ← B×2;		
	Write(B);		50
Read(B); B ← B+100;			
Write(B);			150
		250	150

INF3100 – 2.5.2006 – Ragnar Normann

9

Eksekveringsplan S_E

		A	B
T1	T2'	25	25
Read(A); A ← A+100;			
Write(A);		125	
	Read(A); A ← A×1;		
	Write(A);	125	
	Read(B); B ← B×1;		
	Write(B);		25
Read(B); B ← B+100;			
Write(B);			125
		125	125

INF3100 – 2.5.2006 – Ragnar Normann

10

«Gode» eksekveringsplaner

- Vi vil ha eksekveringsplaner som er «gode»
- Begrepet «god» skal være uavhengig av
 - initialtilstanden
 - transaksjonssemantikken
- Begrepet skal bare være avhengig av lese- og skriveoperasjonene og deres innbyrdes rekkefølge

Eksempel:

$$S_C = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$$

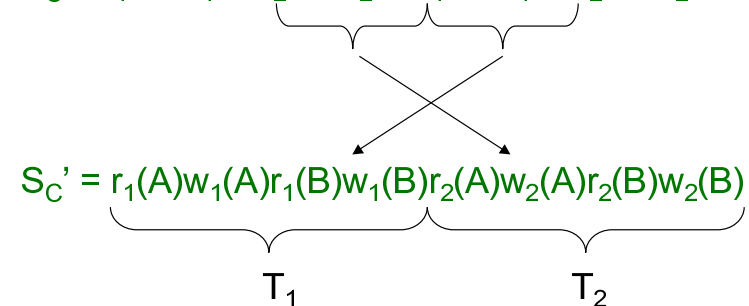
INF3100 – 2.5.2006 – Ragnar Normann

11

«Gode» eksekveringsplaner (forts.)

Eksempel:

$$S_C = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$$



S_C kan rearrangeres til en seriell eksekveringsplan
Vi sier at S_C er **serialiserbar**

INF3100 – 2.5.2006 – Ragnar Normann

12

En «dårlig» eksekveringsplan

La oss så se på plan S_D :

$$S_D = r_1(A)w_1(A)r_2(A)w_2(A)r_2(B)w_2(B)r_1(B)w_1(B)$$

- Legg merke til at i S_D håndterer T_2 dataelementet B før T_1 håndterer det
- Vi sier at T_2 har presedens (over T_1 på B) og skriver det $T_2 \rightarrow T_1$
- Imidlertid håndterer T_1 dataelementet A før T_2 gjør det
- Altså har vi også at $T_1 \rightarrow T_2$ i S_D

En «dårlig» eksekveringsplan (forts.)

$$S_D = r_1(A)w_1(A)r_2(A)w_2(A)r_2(B)w_2(B)r_1(B)w_1(B)$$

- Problemet er at vi både har $T_2 \rightarrow T_1$ og $T_1 \rightarrow T_2$ i S_D
- Denne sykel-avhengigheten gjør at S_D ikke kan omarrangeres til en seriell eksekveringsplan
- Vi sier at S_D er **ikke-serialiserbar**
- Ikke-serialiserbare eksekveringsplaner er «dårlige» eksekveringsplaner, og vi vil ikke tillate dem

Begreper

- **Transaksjon:** En sekvens av $r_i(A)$ og $w_i(B)$ (lese- og skriveoperasjoner)
- **Konflikt:** Det finnes tre typer konflikter:
 - 1) **Lese-skrive-konflikt:** $r_i(A) w_k(A)$ eller $w_i(A) r_k(A)$
 - 2) **Skrive-skrive-konflikt:** $w_i(A) w_k(A)$
 - 3) **Intratransaksjonskonflikt:** $o_i(A) o_i(B)$ (skrive eller lese)
- **Plan:** Kronologisk eksekveringsrekkefølge av transaksjonenes enkeltoperasjoner
- **Seriell plan:** Plan hvor hver transaksjon fullføres før neste transaksjon startes (dvs. ingen fletting av transaksjoner)

Konfliktserialiserbarhet

- To eksekveringsplaner S_1 og S_2 kalles **konflikt-ekvivalente** hvis S_1 kan omformes til S_2 ved en serie ombyttinger av naboooperasjoner som ikke er i konflikt med hverandre
- En eksekveringsplan er **konfliktserialiserbar** hvis den er konfliktekvivalent med en seriell eksekveringsplan
- Litt notasjon:
La S være en eksekveringsplan, og la $p_i(A)$ og $q_k(B)$ være to (vilkårlige) operasjoner i S

Da betyr $p_i(A) <_S q_k(B)$ at $p_i(A)$ gjøres før $q_k(B)$ i S

Presedensgrafer

La S være en eksekveringsplan.

Da defineres **presedensgraften til S , $P(S)$** , slik:

- Noder: Transaksjonene i S
- Kanter: $T_i \rightarrow T_k$ (der $i \neq k$) dersom
 - 1) $p_i(A)$ og $q_k(A)$ er operasjoner i S
 - 2) $p_i(A) <_S q_k(A)$
 - 3) minst en av p_i og q_k er en skriveoperasjon

Oppgave:

- Tegn $P(S)$ for
 $S = w_3(A) w_2(C) r_1(A) w_1(B) r_1(C) w_2(A) r_4(A) w_4(D)$
- Er S serialiserbar?

Presedensgrafer (forts.)

Lemma

S_1 og S_2 er konfliktekvivalente planer $\Rightarrow P(S_1) = P(S_2)$

Bevis:

Anta at $P(S_1) \neq P(S_2)$

Da finnes i og k slik at $T_i \rightarrow T_k$ er kant i S_1 , men ikke i S_2
(eventuelt ved å bytte om S_1 og S_2)

Det betyr at det finnes operasjoner p_i og q_k som er i konflikt over et dataelement A slik at

$$S_1 = \dots p_i(A) \dots q_k(A) \dots$$

$$S_2 = \dots q_k(A) \dots p_i(A) \dots$$

Dette viser at S_1 og S_2 ikke er konfliktekvivalente

q.e.d.

Presedensgrafer (forts.)

Merk!

$P(S_1) = P(S_2) \not\Rightarrow S_1$ og S_2 er konfliktekvivalente

Bevis (Moteksempel):

$$S_1 = w_1(A) r_2(A) w_2(B) r_1(B)$$

$$S_2 = r_2(A) w_1(A) r_1(B) w_2(B)$$

S_1 og S_2 er åpenbart ikke konfliktekvivalente

$P(S_1)$ og $P(S_2)$ har begge de to nodene T_1 og T_2 og de to kantene $T_1 \rightarrow T_2$ og $T_2 \rightarrow T_1$, så $P(S_1) = P(S_2)$

Presedensgrafer (forts.)

TEOREM

$P(S)$ er asyklisk (sykelfri) $\Leftrightarrow S$ er konfliktserialiserbar

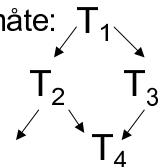
Bevis (\Rightarrow):

Anta at $P(S)$ er asyklisk. Omform S på følgende måte:

- 1) Velg en transaksjon T_1 som ikke har noen inngående kanter i $P(S)$

- 2) Flytt alle operasjonene i T_1 til starten av S

$$S = \dots q_k(B) \dots p_1(A) \dots$$



- 3) Nå har vi $S_1 = \langle \text{operasjonene i } T_1 \rangle \langle \text{resten av } S \rangle$

- 4) Gjenta 1–3 for å serialisere resten av S !

Presedensgrafer (forts.)

TEOREM

$P(S)$ er asyklisk $\Leftrightarrow S$ er konfliktserialiserbar

Bevis (\Leftarrow):

Anta at S er konfliktserialiserbar

Da finnes en seriell plan S_S som er konfliktekvivalent med S

Siden S_S er seriell, er $P(S_S)$ opplagt uten sykler

I følge foregående lemma er $P(S) = P(S_S)$

Følgelig er $P(S)$ asyklisk

q.e.d.

Håndheving av serialiserbarhet og serialiserbarhetsprotokoller

- Metode 1:
Kjør systemet og registrer $P(S)$
På slutten av dagen sjekker vi om $P(S)$ er sykefri, dvs. om alt gikk bra
- Metode 2:
Kontroller på forhånd at eksekveringsplanen aldri kan føre til at det blir sykler i $P(S)$
- Et regelverk som understøtter metode 2, kalles en **serialiseringsprotokoll**

Låseprotokoller

Vi innfører to nye operasjonstyper:

Lock: $l_i(A)$ T_i setter (eksklusiv) lås på A


Unlock: $u_i(A)$ T_i frigir låsen på A

I tillegg må DBMS vedlikeholde en låstabell som viser hvilke dataelementer som er låst av hvilke transaksjoner

De fleste DBMS har en egen «lock-manager»-modul som holder styr på låstabellen

Låseregler i 2PL (2 Phase Locking)

- Regel 1: Velformede transaksjoner:
Hvis T_i gjør en operasjon $p_i(A)$ som er involvert i en konflikt, skal T_i ha gjort $l_i(A)$ før den gjør $p_i(A)$, og den skal gjøre $u_i(A)$ etter at den har gjort $p_i(A)$
Eks.: $T_i: \dots l_i(A) \dots r_i(A) \dots w_i(A) \dots u_i(A) \dots$
- Regel 2: Lovlige eksekveringsplaner:
Eksekveringsplaner kan ikke tillate to transaksjoner å ha lås på samme dataelement samtidig

Eks.: $S: \dots l_i(A) \dots u_i(A) \dots$

ingen $l_k(A)$

Eksekveringsplan S_F

T1	T2
$I_1(A); \text{Read}(A)$	$I_2(A); \text{Read}(A)$
$A \leftarrow A+100; \text{Write}(A); u_1(A)$	$A \leftarrow Ax2; \text{Write}(A); u_2(A)$
$I_1(B); \text{Read}(B)$	$I_2(B); \text{Read}(B)$
$B \leftarrow B+100; \text{Write}(B); u_1(B)$	$B \leftarrow Bx2; \text{Write}(B); u_2(B)$

A	B
25	25
125	
250	
	50
	150
250	150

Låseregler i 2PL (forts.)

- **Regel 3: 2-faselåsing:**
En transaksjon som har utført en unlock-operasjon, har ikke lov til å utføre flere lock-operasjoner

$$T_i = \dots \xleftarrow{\text{ingen } u_i(B)} I_i(A) \dots \xrightarrow{\text{ingen } I_i(B)} u_i(A) \dots$$

- Tiden frem til transaksjonens første unlock-operasjon kalles transaksjonens **voksefase**
- Tiden fra og med transaksjonens første unlock-operasjon kalles transaksjonens **krympefase**

Konfliktregler for lock/unlock

- $I_i(A), I_k(A)$ gir konflikt
- $I_i(A), u_k(A)$ gir konflikt
- Merk at følgende to situasjoner **ikke** gir konflikt:
 - $u_i(A), u_k(A)$
 - $I_i(A), r_k(A)$

Start av krympefasen

- En midlertidig hjelpedefinisjon:
 - $\text{Shrink}(T_i) = \text{Sh}(T_i) =$ første unlock-operasjon T_i gjør
 - **Lemma:** Hvis $T_i \rightarrow T_k$ i S , så er $\text{Sh}(T_i) <_S \text{Sh}(T_k)$
- Bevis:** At $T_i \rightarrow T_k$ betyr at
- $S = \dots p_i(A) \dots q_k(A) \dots$; der p_i og q_k er i konflikt
- Regel 1 sier at $u_i(A)$ må komme etter $p_i(A)$ og $I_k(A)$ før $q_k(A)$
- Regel 2 sier at $I_k(A)$ må komme etter $u_i(A)$. Altså har vi
- $S = \dots p_i(A) \dots u_i(A) \dots I_k(A) \dots q_k(A) \dots$;
- Regel 3 sier at $\text{Sh}(T_i)$ ikke kan komme etter $u_i(A)$ og at $\text{Sh}(T_k)$ må komme etter $I_k(A)$
- Dermed har vi bevist at $\text{Sh}(T_i)$ må komme før $\text{Sh}(T_k)$ i S

q.e.d.

2PL sikrer konfliktserialiserbarhet

TEOREM Hvis en plan S overholder regel 1, 2 og 3, så er S konfliktserialiserbar

Bevis: I følge forrige teorem er det nok å vise at hvis en plan S overholder regel 1, 2 og 3, så er presedensgrafen $P(S)$ sykelfri

Anta derfor (ad absurdum) at $P(S)$ har en sykkel

$$T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$$

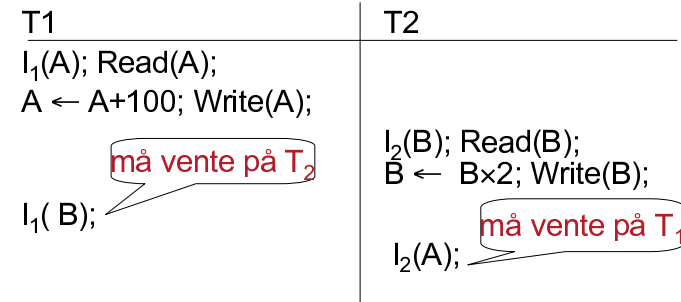
I følge lemmaet er da

$$\text{Sh}(T_1) <_S \text{Sh}(T_2) <_S \dots <_S \text{Sh}(T_n) <_S \text{Sh}(T_1)$$

Men det er umulig, så $P(S)$ er sykelfri

QED

Vranglås



Dette viser at 2PL **ikke** sikrer mot vranglås (deadlock)!

Lese- og skrivelåser

- For å få bedre samtidighet kan vi bruke to låsetyper:
- **Leselås (Shared lock)** som tillater andre transaksjoner å lese dataelementet, men ikke å skrive det
- **Skrivelås (eXclusive lock)** som hverken tillater andre transaksjoner å lese eller skrive dataelementet
- **Notasjon:**
 - $sl_k(A)$ – T_k setter leselås på A
 - $xl_k(A)$ – T_k setter skrive lås på A
 - $u_k(A)$ – T_k sletter sin(e) lås(er) på A
- $sl_k(A)$ blir ikke utført hvis noen annen transaksjon enn T_k har skrive lås på A
- $xl_k(A)$ blir ikke utført hvis en annen transaksjon enn T_k har låst A (det spiller ingen rolle om det er lese- eller skrive lås)

Regler for lese- og skrivelåser

- **Velformede transaksjoner**
Enhver transaksjon T_k må overholde disse tre reglene:
 - En $r_k(A)$ må komme etter en $sl_k(A)$ eller $xl_k(A)$ uten at det er noen $u_k(A)$ mellom dem
 - En $w_k(A)$ må komme etter en $xl_k(A)$ uten at det er noen $u_k(A)$ mellom dem
 - Etter en $sl_k(A)$ eller $xl_k(A)$ må det komme en $u_k(A)$
- **2PL for transaksjoner**
Enhver 2PL-transaksjon T_k må i tillegg overholde:
 - Ingen $sl_k(A)$ eller $xl_k(A)$ kan komme etter en $u_k(B)$ (uavhengig av hva A og B er)

Regler for lese- og skrive-låser (forts.)

- **Lovlige eksekveringsplaner**

Hvert dataelement er enten ulåst, eller det har én skrive-lås, eller det har en eller flere lese-låser

Dette sikres ved at alle planer S følger disse reglene:

- Hvis $xl_i(A)$ forekommer i S, må den etterfølges av en $u_i(A)$ før det kan komme en $xl_k(A)$ eller $sl_k(A)$ med $k \neq i$
 - Hvis $sl_i(A)$ forekommer i S, må den etterfølges av en $u_i(A)$ før det kan komme en $xl_k(A)$ med $k \neq i$
- Merk at det er tillatt for en transaksjon å ha både lese- og skrive-lås på samme dataelement
- Dette gir større fleksibilitet og *kan* bidra til mer samtidighet (dette er situasjonsavhengig)

Konfliktserialiserbarhet av S/X-planer

TEOREM

Hvis en plan S overholder reglene for lese- og skrive-låser på de to forrige lysarkene, så er S konfliktserialiserbar

Bevis:

Nær identisk med beviset for at planer som bare bruker skrive-låser, sikrer konfliktserialiserbarhet

Den eneste forskjellen er at vi får bruk for at hverken

$Is_i(A)$ fulgt av $Is_k(A)$

eller

$Is_i(A)$ fulgt av $u_k(A)$

er en konflikt

Kompatibilitetsmatriser

- Kompatibilitetsmatriser brukes for å lagre regelverket for tildeling av låser når vi benytter flere låstyper
- Matrisene har en rad og en kolonne for hver låstyp
- Kompatibilitetsmatriser tolkes slik:

Hvis T_i ber om å få satt en lås av type K på dataelement A, får den det bare hvis det står 'Ja' i kolonne K i alle rader R i matrisen hvor noen annen T_k har en lås av type R på A
- **Eksempel:** Kompatibilitetsmatrise for S/X-låser

		S	X	(lås T ber om å få på A)
(lås som A har)	S	Ja	Nei	
	X	Nei	Nei	

Oppgradering av låser

- For å gi bedre samtidighet kan vi tillate T å først sette lese-lås og så oppgradere den til skrive-lås ved behov
- **Eksempel:**

T_1	T_2
$sl_1(A); r_1(A);$	$sl_2(A); r_2(A);$ $sl_2(B); r_2(B);$
$sl_1(B); r_1(B);$ $xl_1(B);$ Avslått	$u_2(A); u_2(B);$
$xl_1(B); w_1(B);$ $u_1(A); u_1(B);$	

Oppgradering av låser (forts.)

- En ulempe er at å oppgradere låser gir økt risiko for vranglås

- Eksempel:

T ₁	T ₂
sl ₁ (A); r ₁ (A);	sl ₂ (A); r ₂ (A);
xl ₁ (A); Avslått	xl ₂ (A); Avslått

- Eksemplet illustrerer at protokoller som bruker oppgradering av låser, bare egner seg hvis det er mange flere lese- enn skrivetransaksjoner

Oppdateringslåser

- En oppdateringslås er en leselås som senere skal oppgraderes til en skrive­lås
- Oppdateringslåser betegnes med U (Update lock)
- Kompatibilitetsmatrisen for S/X/U-låser finnes i to varianter (hvor den usymmetriske er mest vanlig):
 - en usymmetrisk som prioriterer skrivetransaksjoner
 - en symmetrisk som prioriterer lesetransaksjoner

Komp.matrise	S	X	U	(ønsket lås)
S	Ja	Nei	Ja	
(holdt lås) X	Nei	Nei	Nei	
U	J/N	Nei	Nei	

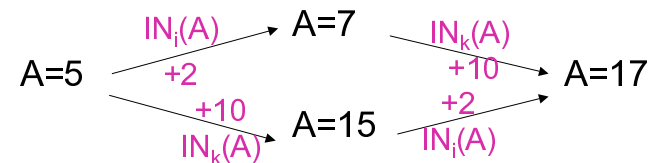
Oppdateringslåser (forts.)

- Planer som gir vranglås på grunn av oppgraderinger av lese- til skrive­låser, gjør det ikke med bruk av oppdateringslåser
(**NB** Det kan være andre årsaker til vranglås!)
- Eksempel (Det som ga vranglås i sted (ark 37)):

T ₁	T ₂
ul ₁ (A); r ₁ (A);	ul ₂ (A); Avslått
xl ₁ (A); w ₁ (A); u ₁ (A);	ul ₂ (A); r ₂ (A);
	xl ₂ (A); w ₂ (A); u ₂ (A);

Inkrementlåser

- Atomisk inkrementoperasjon: IN_i(A)
{Read(A); A ← A+v; Write(A)}
- (**NB** Her gjøres Read og Write bare fra og til hukommelsen, **ikke til disk** !!)
- IN_i(A) og IN_k(A) er ikke i konflikt!



Inkrementlåser (forts.)

- Hensikten er å effektivisere bokholderitransaksjoner
- Inkrementlåser betegnes med I (Increment lock)
- Inkrementlåser er i konflikt med både lese- og skrivelåser, men ikke med andre inkrementlåser
- Her er kompatibilitetsmatrisen for S/X/I-låser:

Komp.matrise		S	X	I	(ønsket lås)
	S	Ja	Nei	Nei	
(holdt lås)	X	Nei	Nei	Nei	
	I	Nei	Nei	Ja	

Låshåndtering (Lock Scheduling)

- I praksis vil ikke noe DBMS la transaksjonene sette eller frigi noen låser selv
- Transaksjonene utfører bare operasjonene read, write, commit og abort, og eventuelt update og increment
- Låsene legges inn i transaksjone og settes og frigis av en egen modul i DBMS kalt **låshåndtereren** (Lock Scheduler (i noen DBMS kalt Lock Manager))
- Låshåndtereren bruker en egen intern datastruktur, **låstabellen**, til å administrere låsene
- Låstabellen er ikke en del av bufferområdet; den er utilgjengelig for transaksjonene

Låshåndtering (forts.)

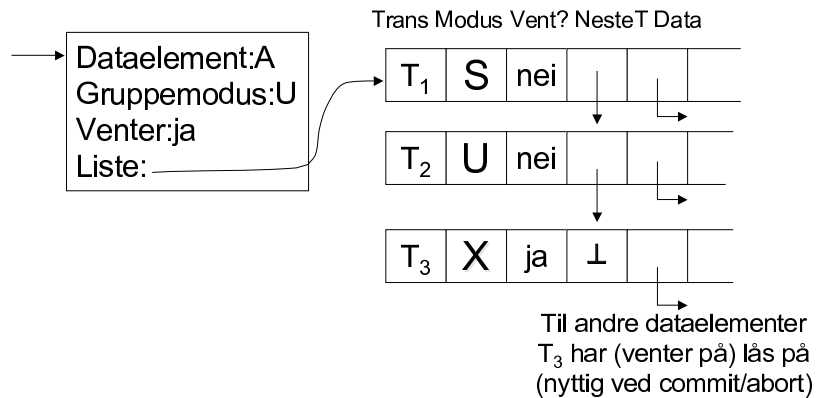
Låshåndtereren består av to deler:

- Del I analyserer hver transaksjon T og legger inn «riktige» låsekrav foran operasjonene i T og legger kravene i låstabellen
Hvilke krav den velger, er avhengig av hvilke låstyper som er tilgjengelige
- Del II kontrollerer om operasjonene og låskravene den mottar fra del I kan utføres
De som ikke kan det, legges i en kø for å vente på at den låsen som hindrer utførelsen blir fjernet (det er en kø for hver lås)
- Når T gjør commit (eller abort), sletter del I alle låser satt av T og varsler del II som sjekker ventekøene for disse låsene og lar de transaksjonene som kan, fortsette

Låstabellen

- Låstabellen er logisk sett en tabell som for hvert dataelement i databasen inneholder all låsinformasjon for dette elementet
- I praksis organiseres låstabellen som en hashtabell med dataelementets adresse som nøkkel
- Ulåste dataelementer er ikke med i låstabellen
- Låstabellen er derfor proporsjonal med antall ønskede og innvilgede låser, og ikke med antall dataelementer
- For hver A i låstabellen lagres følgende informasjon:
 - Gruppemodus (strengeste lås som holdes på A)
 - Et venteflagg som sier om noen venter på å få låse A
 - En liste over de T som har, eller venter på, lås på A

Eksempel på låsinfo for et dataelement A



Granularitet og varsellåser

- Begrepet dataelement er med vilje udefinert
Tre naturlige granulariter på dataelementer er:
 - en relasjon – et naturlig største (låsbar) dataelement
 - en blokk – en relasjon består av en eller flere blokker
 - et tuppel – en blokk kan inneholde ett eller flere tupler
- Forskjellige transaksjoner kan samtidig ha behov for låser på alle disse nivåene
- For å få til det innfører vi varsellåser, IS og IX, som sier at vi har til hensikt å sette henholdsvis en lese- eller skrive-lås lenger ned i hierarkiet

Varsellåser (forts.)

Komp.matrise		IS	IX	S	X (ønsket lås)
(holdt lås)	IS	Ja	Ja	Ja	Nei
	IX	Ja	Ja	Nei	Nei
	S	Ja	Nei	Ja	Nei
	X	Nei	Nei	Nei	Nei

- Eksempel: T vil skrive tuppel A i blokk B i relasjon R
 1. Hvis R hverken har S-lås eller X-lås, setter T IX-lås på R
 2. Fikk T satt IX-lås på R, sjekker den om B har S- eller X-lås
Har ikke B det, setter T IX-lås på B
 3. Fikk T satt IX-lås på B, sjekker den om A har noen lås
Har ikke A det, setter T X-lås på A og kan skrive A
- Merk at hvis en transaksjon T har en skrive-lås på R, kan ingen andre skrive noe tuppel i R før T sletter låsen

Håndtering av fantomtupler

- Eksempel:
- Vi skal summere et felt for alle tuplene i en relasjon R
- Før summeringen setter vi leselås på alle tuplene i R for å sikre oss et konsistent svar
- Under summeringen legger en annen transaksjon inn et nytt tuppel i R, noe som gjør at summen kan bli gal
- Dette er mulig fordi tuppelet ikke eksisterte da vi satte våre leselåser (et slikt tuppel kalles et **fantomtuppel**)
- Løsningen er å sette en IS-lås på relasjonen
Da får ingen legge inn nye tupler før låsen er slettet