DB Øving 5 Gruppe 206

Anhkha Vo Nguyen, Kristoffer Nyvoll og Øyvind J. Schjerven

Oppgave 1

Eksamensregistrering(Studentnr, Eksamennr, Dato, Status)

a) SELECT * FROM Eksamensregistrering;

Her skal vi hverken gjøre noen søk etter attributter eller sortere, hvilket medfører at den mest intuitive løsningen er å bruke en heapfil. Du kan da finne blokken der tabellen er lagret og hente ut den informasjonen du trenger.

b) SELECT * FROM Eksamensregistrering WHERE Eksamennr = 963432;

B+ med Eksamennr som , ettersom dette er enklest å bruke for å hente ut et spesifikt nr.

c) SELECT * FROM Eksamensregistrering WHERE Eksamennr = 963432 ORDER BY Studentnr DESC;

B+ med eksnr og studnr som nøkkel ettersom dette tillater å sortere fortløpende på studnr. B+ vil også gi minst ekstra utregning.

d) INSERT INTO Eksamensregistrering (14556589, 963439, ‘26.11.2018’, “Øvingsopplegg bestått”);

Her skal vi kun sette inn verdier uten noe krav for hvor det spesifikt skal være. Dermed kan vi legge det sist i filen. Siden det heller ikke er noe krav til sortering, kan vi bruke en heapfil til denne operasjonen.

Oppgave 2

NLJ leser rader fra den første tabellen i en loop (enkeltvis) og videresender hver rad en nested loop som prosesserer den neste tabellen som skal joines. Dette gjentas til alle tabellene er joinet. Ettersom NJL kjører rader enkeltvis fra ytre til indre loops, leser den vanligvis tabeller i den indre loopen mange ganger.

S := Student: 47 000 poster i 800 blokker
E := Eksamensregistrering: 500 000 poster i 12 800 blokker
34 blokker er tilgjengelig i buffer. Hvor mange blokker leses ila joinen?

1 - Vi tester først med Student i den ytre løkken:
Vi bruker altså 32 bufferblokker til **S**, 1 til **E** og 1 til output. Får da:
(800/32) * (32 + 12 800), der 12 800 er antall blokker i **E**. Itererer gjennom den ytre løkken 800/32 = 25 ganger. Totalt antall blokker som leses her: **320 800**

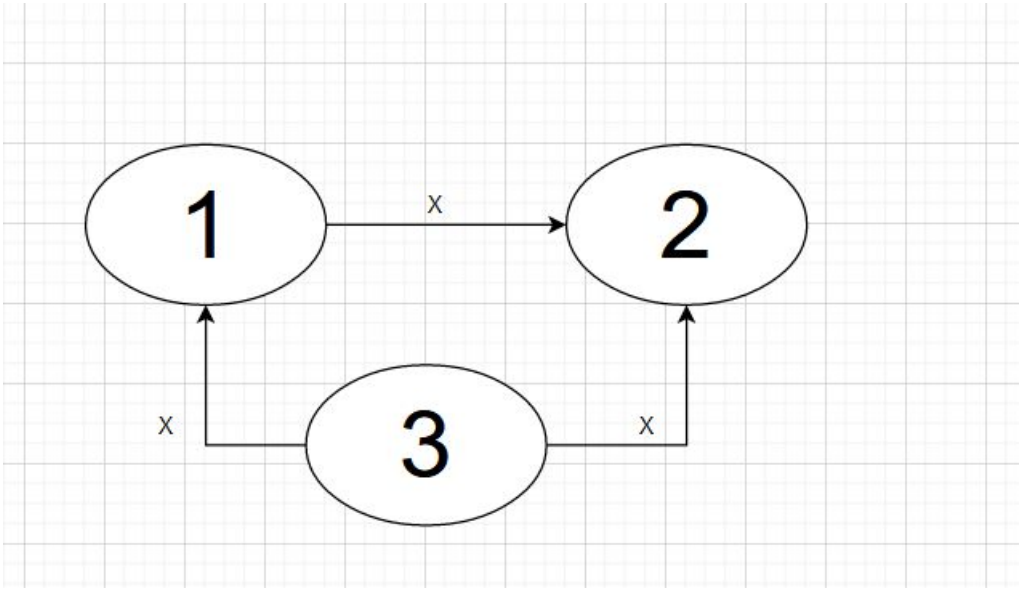
2 - Tester deretter med **E** i den ytre løkken. Får da:
400 * (32 + 800) = **332 800**

<u>DVS: Det leses 320 800 blokker med Student i den ytre løkken og 332 800 med eksamensregistrering i den ytre løkken.</u>

Oppgave 3

a) Støtter deling og samtidig aksess av data. Støtter trygg, pålitelig atomisk aksess av store mengder data
b) Atomisk - Kan ikke bli avbrutt. Enten kjører de til slutten, eller ikke i det hele tatt
   Consistency - Overholder konsistenskrav
   Isolerende - Merker ikke at noen andre kjører samtidig. Operasjonene skal være isolert
   Durability - Er permantente, mistes ikke etter en commit
c) Historie 1 er: Ikke gjennomretbar siden i T2 så leses X, som er en ikke-commita element. Den blir skevet i T1
   Historie 2 er: ACA. X er eneste element som blir lest i H2, og eneste X blir comitta i T1 før den blir lest i T2 og T3.
   Historie 3 er: Strict. Y i T3 blir committa før den blir brukt i T1. X i T2 blir comitta før den blir skrevet i T1.

d) To operasjoner er i konflikt når de tilhører forskjellige transaksjoner, bruke samme dataelement og minst en av operasjonene er en write.
e) HIstorie 4 er konfliktserialiserbar siden presedensgrafen ikke har noen loops



Figur 1: Presedensgrafen til Historie 4

f) En vranglås mellom to transaksjoner oppstår hvis hver av transaksjonene krever en handling/avhenger av den andre, slik at begge blokkeres. En presedensgraf f.eks. vil derfor være sirkulær.

g)

Time	T1	T2	T3
			write_lock(x)
	read_lock(x)		Write(x)
	wait	write_lock(x)	unlock(x)
	read(x)	wait	read_lock(y)
	commit/unlock(x)	wait	read(y)
		read(x)	wait
		write(x)	wait
		commit/unlock(x)	wait
			commit/unlock(y)

Oppgave 4

a) T2 er comitted før krasjen, og er dermed en vinner. T1 og T3 er ikke comitted, og er derfor tapere. Altså skal T2 være permanent og få REDO, mens T1 og T3 skal aborteres.

LSN	Prev_LSN	Transaction	Operation	Page_ID	...
147			End_ckpt		
148	0	T_1	Update	A	
149	0	T_2	Update	B	
150	149	T_2	Commit		
151	148	T_1	Update	A	
152	0	T_3	Update	B	
Krasj!					

Figur 2: Skjermbilde av loggen som viser at T2 er commited

b) Transaksjonstabell og Dirty Page Table (DPT)

Under vises en minimalistisk tabell som får frem tilstand og det sist brukte Log Sequence Number (LSN).

TransaksjonsID	Tilstand	Last LSN
T1	Aktiv	151
T2	Committed	150
T3	Aktiv	152

Figur 3: Transaksjonstabellen til oppgave 4b)

Skitne blokker inneholder all informasjonen som er *staged,* altså ikke commited.

Skitne blokker

A	B
148	149

Figur 4: DPT etter ferdigstilt analysefase