TDT4145 Øving 4

Philip Ditlevsen, Yngve Tryggestad Larsen, Natalie Sørensen Forshaw

April 2021

1 Extendible hashing

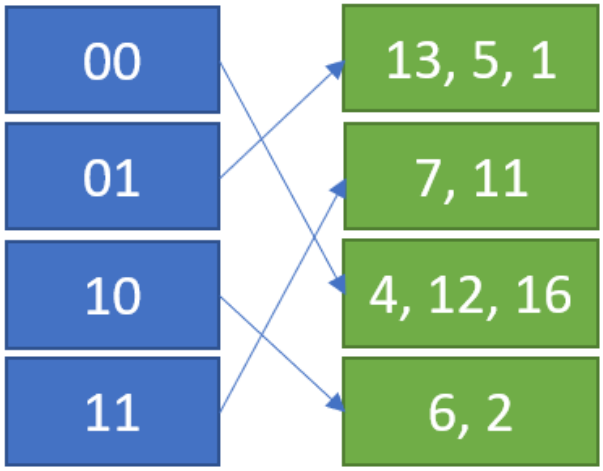

Figure 1: Global dybde 2, lokal dybde 2, og alle nøkler som passer inn i bøttene slik de er oppdelt

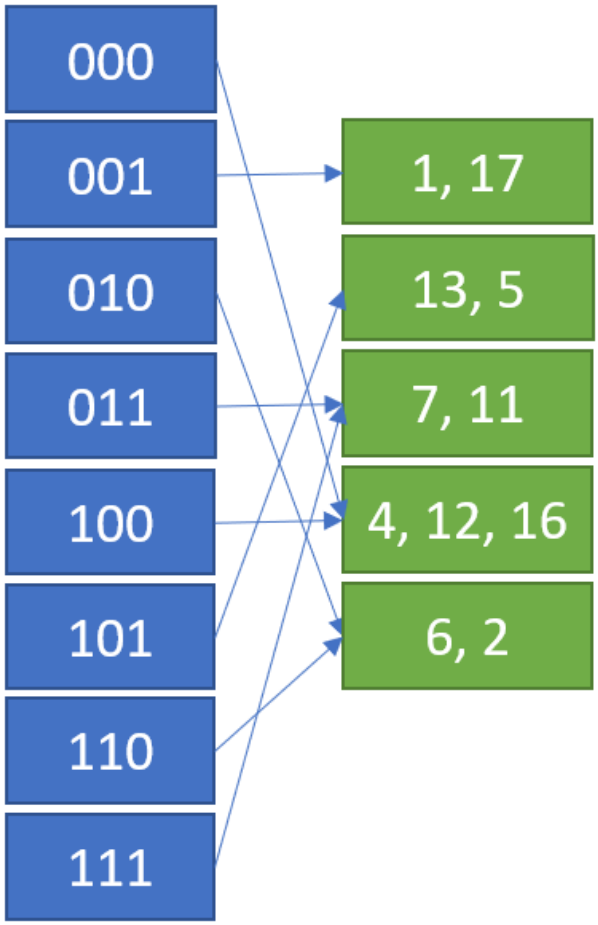

Figure 2: Global dybde 3, og alle nøkler plassert i sine bøtter

2 Konstruksjon av B+-tre

Figure 3: Første 3 nøkler i stigende rekkefølge

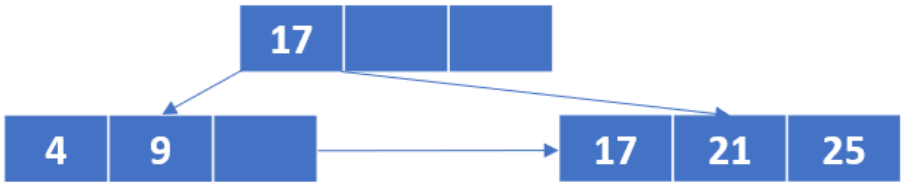

Figure 4: Det er ikke plass til flere nøkler i roten og treet splittes på 17

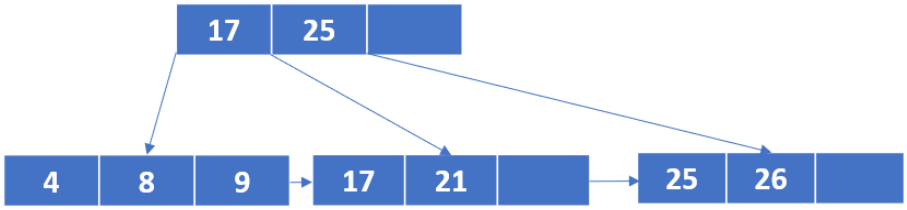

Figure 5: Høyre blokk fylles opp og splittes på 25

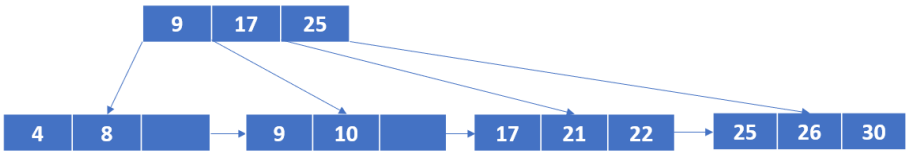

Figure 6: Venstre blokk fylles opp og splittes på 9

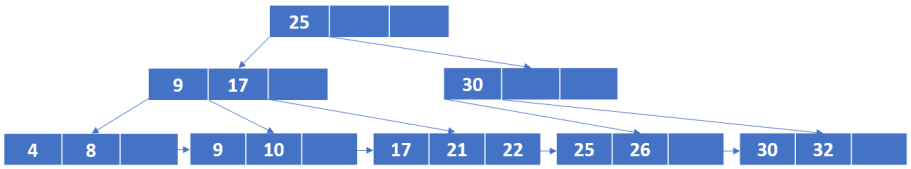

Figure 7: Høyre blokk fylles opp og splittes på 30, men roten er fylt opp og splittes selv på 25

3 Lagring og queries med clustered B+-tre

a Først deler vi antall poster(personer) på 8, siden hver blokk kan inneholde 8 poster da får vi 10 000 % 8 = 1250.
Siden vi har en fyllingsgrad på 2/3, deler vi det vi fikk med 2/3, som gir oss 1250 / 2/3 = 1875 løvnoder.

b 2048 / 8 (PersonID + bloddidentifikator) gir 256
Med fyllgrad (2/3) blir det 170.666 pekere per blokk
Som gir 1875/170.666 = 11.029, som vil si vi trenger 12 blokker
Siden det kun er 12 blokker i andre nivå, vet vi at vi kun trenger 1 blokk på øverste nivå (rotnivå).

c 1: 3 blokker. En for hvert nivå i B+ treet.
2: 1877 blokker (2 + 1875). Først to hakk ned i treet, deretter lese alle løvnodene.
3: 1877 Blokker. Dette blir samme som forrige, siden treet allerede er sortert etter PersonID.
4: 96 blokker. 2 + (1875*5%). To hakk ned, og antall løvnoder ganget med prosenten av de som utfyller kravet.

4 Queries med heap og unclustered B+-tre

1. Her aksesseres 1250 blokker, siden den må scanne hele heapfilen (S1).
2. Her aksesseres rundt 625 blokker, siden equality search i en heap må se igjennom omtrent halvparten av blokkene.
3. Vi må aksessere Heapen en gang for hver person med det etternavnet + lese 3 blokker i B+ treet.
Dette gir oss 3 + antall personer med etternavn "Søkerud"
4. Vi må gå igjennom alle løvnodene i B+ treet.
Dette gir oss 2 + antall løvnoder(300).
Vi må derfor aksessere 302 blokker.
5. Vi må aksessere 6 blokker. Vi må lese 1 og skrive 1 i heapen + lese 3 og skrive 1 i B+ treet.

5 Nested-loop-join

12800 blokker som joines med 800 blokker med 34 blokker i buffer
det brukes 1 blokk som skal holde på resultatet og vi setter 1
blokk for den første tabellen med 32 resterende for den siste
tabellen.

Vi leser Student først og får dermed 25 loops antall blokker som
totalt leses vil da bli 25*(32+12800) = 320 800
Det leses da 320 800 blokker i løpet av joinen.

6 Transaksjoner

a Transaksjoner sørger for at operasjoner blir utført etter hverandre i stedet for samtidig, dermed kan man unngå komplikasjoner om flere brukere utfører handlinger som påvirker hverandre samtidig. Transaksjoner er også veldig nyttig for å oppnå data integritet, for eksempel kan man ved hjelp av transaksjoner hindre større konsekvenser ved et strømbrud. Man kan gå tilbake i transaksjonshistorikken og rette opp i de siste transaksjonene før strømbruddet så ingen data går tapt.

b Atomicity: Denne egenskapen skal forsikre at om en del av en operasjon utføres, så utføres også resten. Ingen halve operasjoner skal utføres. Om en feil oppstår og en del av operasjonen ikke kjører, da skal heller ikke neste del kjøres.

Consistency: Databasen før og etter en operasjon skal følge regelen for databasen for å beholdes konsistens, det er fortsatt lov å være inkonsistent mens operasjonen kjøres, så lenge resultatet følger databasens regler.

Isolation: Denne egenskapen er viktig for systemer som ikke kjører serielt. Isolasjon innebærer at ingen operasjoner skal være avhengige av midlertidige variabler. Om det er snakk om en variabel som oppdateres etter hver operasjon, vil bare 1 av flere parallelle operasjoner som bruker denne variabelen få rett resultat.
I hovedsak handler det om at transaksjoner ikke skal påvirkes av at andre transaksjoner kjører samtidig, slik at resultatet er det samme som om de hadde blitt kjørt serielt.

Durability: Handler om at utførte transaksjoner aldri skal forsvinne Resultatene fra utførte transaksjoner skal alltid lagres slik at det ikke oppstår større konsekvenser om systemet feiler.

c Ingen av historiene er strict eller cascadeless.
H1: Unrecoverable
H2: Unrecoverable
H3: Recoverable

d To operasjoner kan havne i konflikt dersom de hører til ulike transaksjoner, minst en av de er en skriveoperasjon eller om operasjonene utføres på det samme objektet (read eller write).

e

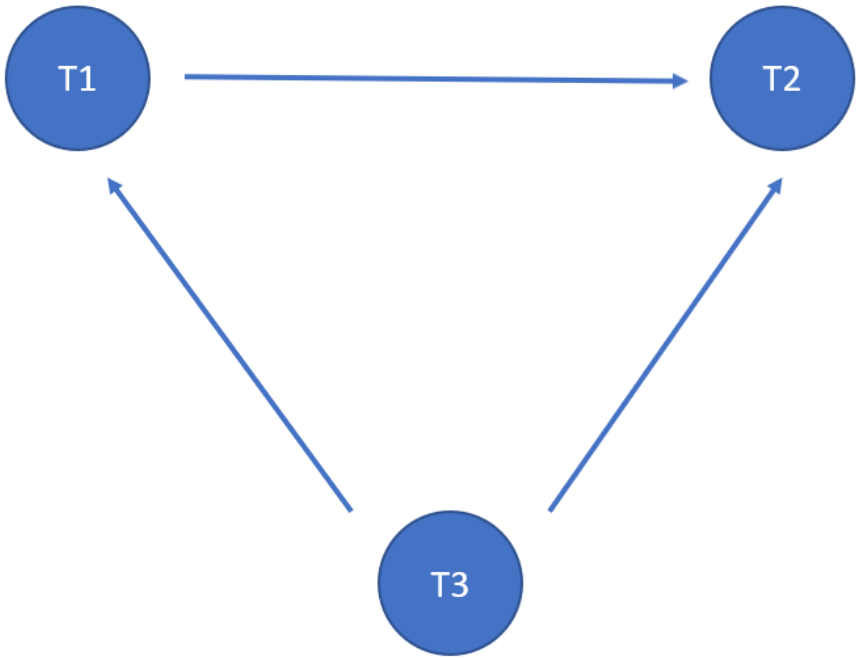

Figure 8: Historien har ingen sykel og er derfor konfliktserialiserbar

f Vi sier vi har en vranglås mellom 2 eller flere transaksjoner om hver operasjon i en transaksjon venter på en hendelse som bare kan forårsakes av en annen operasjon i transaksjonen.

g Se tabell

T1	T2	T3
		wl3(X)
		w3(X)
Venter på X		
	Venter på X	
		rl3(Y)
		r3(Y)
		c3
		unlock(X), unlock(Y)
rl1(X)		
r1(X)		
	rl2(X)	
	r2(X)	
	Venter på X	
c1		
unlock(X)		
	wl2(X)	
	w2(X)	
	c2	
	unlock(x)	

7 Recovery etter krasj med Aries

a I dette tilfellet vil T2 være vinner. t1 og t3 vil være tapere dette fordi de ikke har fått committe enda.

Log tail vil flushes ved hver committ slik at vi er sikker på at hva som er committet lagres persistent til enhver tid. T2 er den eneste transaksjonen som har gjort operasjon commit og vil derfor være den siste transaksjonen som har gått gjennom.

b)

T-table

Transaction	Status	Last LSN
T1	Running	151
T2	Commited	150
T3	Running	152

Figure 9: Transaksjonstabell etter analysefase

DPT

PageID	Recovery LSN
A	148
B	149

Figure 10: ”Dirty page” tabell etter analysefasen