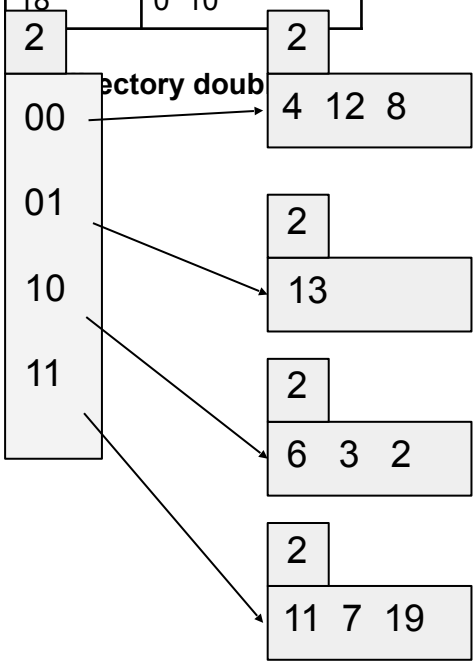
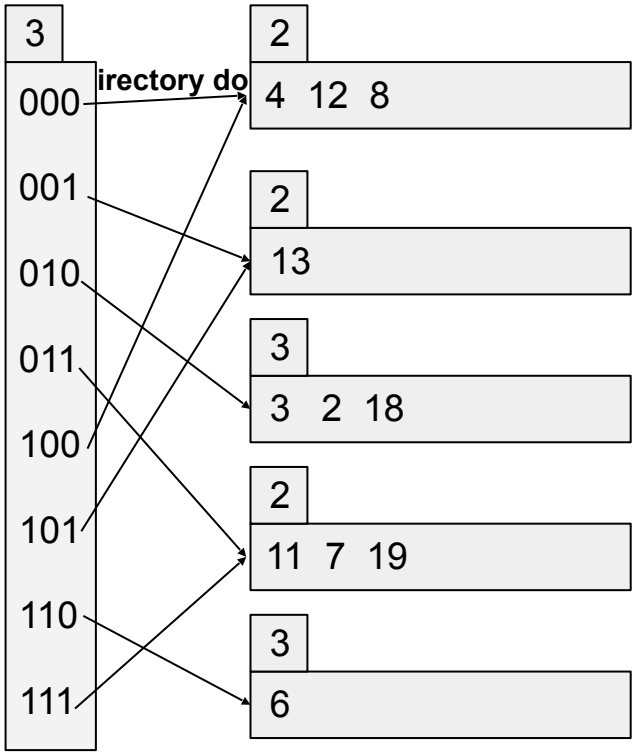
Øving 4

Oppgave 1 - Extendible hashing

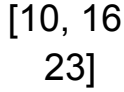
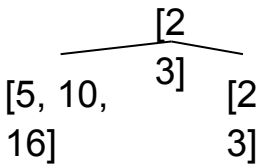
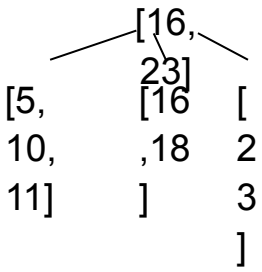
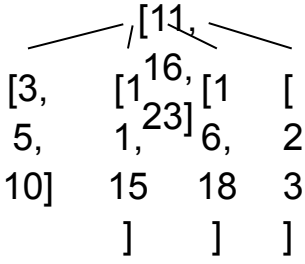
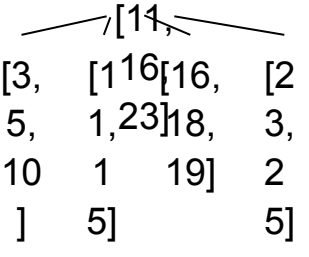
Verditabell i binært (K MOD 4 extended til K MOD 8):

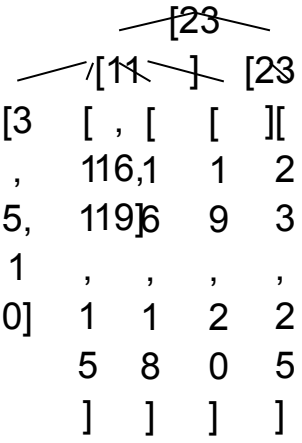
Verdi	K MOD 8
6	1 10
4	1 00
11	0 11
13	1 01
12	1 00
7	1 11
8	0 00
3	0 10
2	0 10
19	0 11
18	0 10





Oppgave 2 - Innsetting i B+-tre

Steg	Tre
1	
2	
3	
4	
5	

6	

Oppgave 3 - Lagring og queries med clustered B+-tre

a) $10\,000/(8*2/3) = 10\,000/5{,}33\ldots 3 = 1\,875\ blocks$

b) Utregning av antall blokker

Nivå	**Blokker**
0 (Løvblokk)	10000/(8*⅔) = **1875**
1 (Intern)	2048/8 = 256 ⅔*256 = 170,66…6 ≈ 170 1875/170 = 11,03 ≈ **12**
2 (Rotblokk)	12/(⅔*256) = 0,07 ≈ **1**

c) Spørringer

i) 3 blokker fordi tre nivå må gjennom for å nå den relevante blokken.

ii) 1 875 + 2 = 1 877 blokker aksesseres fordi vi må ned i treet i tillegg til å lese ned to blokker fra rotblokk.

iii) Samme som over. Treet er allerede clustered og har PersonID (PK) som søkenøkkel.

iv) 5% * 1 875 + 2 = 94 + 2 = 96 blokker, to nivåer ned pluss scan gjennom 5% av blokkene bortover (94)

Oppgave 4 - Queries med heap og unclustered B+-tre

Spørringer:

1. Min: **1 250 blokker**. Skanner hele heapfilen.
   Maks: 2 + 300 + 10 000 = **10 302 blokker**. Søker med B+-treet. Dette er med maksimal “uflaks” ved at alle ingen etterfølgende poster ligger i samme blokk.
2. Min med heapfil: **1 blokk**. Treffer på første blokk i heapfilen.
   Min med B+-tre: 2 + 1 + 1 = 4 blokker. Treffer på første blokk i treet.
   Gj.snitt med heapfil: 1 250 * 50% = **625 blokker**. Antar at vi finner match i midten av søket.
   Gj.snitt med B+-tre: 10 302 * 50% = 5 151 blokker. Antar at vi finner match i midten av søket.

3. 2 + 1 + 1 = **4 blokker**. Lastname er søkenøkkel i treet.
4. 2 + 300 = **302 blokker**. Må gjennom alle løvblokkene i B+-treet.
5. Heap: 1 read + 1 write = 2
   B+-tre: 3 read + 1 write = 4
   2 + 4 = **6 blokker**. Fordi man setter inn posten på slutten av treet + må skrive inn på slutten av heapfilen.

Oppgave 5 - Nested-loop-join

Student:
- 47 000 poster
- 800 blokker

Eksamensregistrering:
- 500 000 poster
- 12 800 blokker

Alternativ 1:

12 800/32 = 400
400 * (32 + 800) = 332 800 blokker

Alternativ 2:

800/32 = 25
25 * (32 + 12 800) = 320 800 blokker ← Går for den laveste

Oppgave 6 - Transaksjoner

a) Vi har transaksjoner for å kunne benytte pipelining i databasen, og dermed kunne utføre langt flere operasjoner samtidig.

b) **A** - Atomiske: enten kjører de fullstendig eller så kjører de ikke
C - Consistency: overholder konsistenskrav
I - Isolation: som er isolert fra hverandre, merker ikke at andre kjører samtidig
D - Durability: er permanente, mistes ikke etter en commit

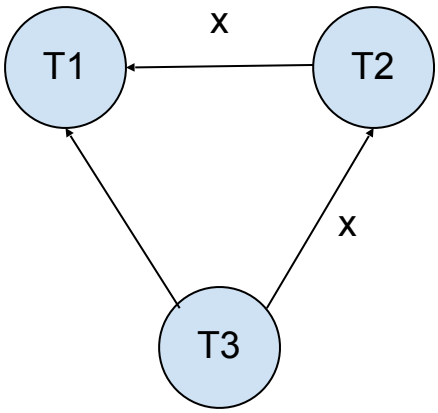
c) H1: w1(Y); r2(X); w1(X); w3(Y); r2(Y); c1; c3; c2
Denne historien har **recoverable** recovery-egenskap.

H2: r2(X); w2(Y); r1(X); w3(X); c2; r1(Y); r3(Y); w1(X); c3; c1
Denne historien har **ACA** recovery-egenskap.

H3: r2(X); w2(X); r3(X); w1(Y); c1; w3(Y); c3; c2
Denne historien er **ikke gjenopprettbar** fordi t2 committer etter t3, og t3 leser den endrede verdien til t2.

d) To operasjoner i en historie er i konflikt hvis de tilhører forskjellige transaksjoner og opererer på samme dataobjekt, og minst en av operasjonene er en skriveoperasjon (write).

e) H4: r1(Y); w3(X); r3(Y); w2(X); r1(X); c1; c2; c3;
Denne er konfliktserialiserbar ettersom det ikke finnes noen sykler:



f) Når vi har en vranglås mellom transaksjoner betyr at disse venter på hverandre på å kunne fortsette, dette skjer hvis for eksempel T1 har låst X, og T2 har låst Y, hvor T1 venter på å kunne bruke Y samtidig som T3 venter på å bruke X. Dermed står disse å venter på hverandre uten å kunne fortsette.

g) H4: r1(Y); w3(X); r3(Y); w2(X); r1(X); c1; c2; c3;

H4	T1	T2	T3
r1(Y)	read_lock(Y)		
	read(Y)		
w3(X)			write_lock(X)
			write(X)
r3(Y)			read_lock(Y)
			read(Y)
w3(X)*		try_lock(X)	
r1(X)*	try_lock(X)	wait	
c1*	wait	wait	
c2*	wait	wait	
c3	wait	wait	commit / unlock
w3(X)**	wait	write_lock(X)	
	wait	write(X)	
r1(X)** c1** c2**	wait	commit / unlock	
r1(X)***	read_lock(X)		
	read(X)		
c1***	commit / unlock		

* try, but has to wait
** second try
*** third try

Oppgave 7 - Recovery etter krasj med ARIES

a) **REDO: T1 og T2** ble comitet etter siste sjekkpunkt og trenger å bli gjort igjen.
UNDO: T3 ble ikke comitet etter siste sjekkpunkt og må derfor reverseres.

b) Transaksjonstabell:

TID	**Last_LSN**	**Status**
T1	173	Committed
T2	170	Committed
T3	174	In progress

DPT:

Page_ID	**LSN**
A	168

B	172