UNIVERSITETET I OSLO
Institutt for informatikk

Rindex med tråder

Masteroppgave

60 studiepoeng

Marius Vikdal

1. mai 2008

Forord

Denne oppgaven er utarbeidet ved Institutt for Informatikk ved Universitetet i Oslo. Oppgaven er en del av det toårige masterprogrammet i informatikk. Den er verdt 60 av de totale 120 studiepoengene som masterprogrammet består av. Oppgaven er veiledet av amanuensis Ragnar Normann.

Arbeidet har vært utfordrende og veldig lærerikt. Utfordringene har spesielt ligget i å sette seg inn i eksisterende kode som er utarbeidet av tre forskjelige studenter, i tre tidligere oppgaver. Da alle har sin måte å gjøre ting på, har det være utfordrende å få oversikten.

Gjennom oppgaven har jeg fått en bredere forståelse av hvordan et databasehåndteringssystem (DBMS) henger sammen og fungerer.

Jeg vil takke Ragnar Normann for god veiledning og oppfølging gjennom hele oppgaven. Jeg vil også takke IFI drift, spesiellt Kristin Skar, for teknisk hjelp med servere og programvare, samboeren min Ina for å holde ut med mine noe ukurante arbeidstider, valpen vår Thea for fine og avslappende turer for å samle tankene, Thomas "kniven" Kristensen for svar på mine utallige kodespørsmål, Roger Bystrøm for hjelp til å finne feil i koden, familie for korrekturlesning og støtte gjennom hele utdannelsen og ikke minst alle andre medstudenter for oppmuntring under arbeidet.

Oslo, 1. mai 2008
Marius Vikdal

Innhold

Figurer

Kapittel 1

Innledning

Rindex er et forskningsprosjekt ved gruppen Objektorientering, modellering og språk ved Institutt for Informatikk (IFI) ved Universitetet i Oslo (UiO).

Rindex står for RAM index og er en plattform laget for å ligge på toppen av allerede eksisterende databasehåndteringssystemer (DBMS).

Rindex har vært jobbet med ved tre tidligere masteroppgaver. De to første ble utført av Tomas Are Haavet og Kjell-Magne Øierud i 2005. Sammen utviklet de Rindex plattformen og la med det grunnlaget for videre utvikling. Denne versjonen er kjent som Rindex 0.1. Den er presentert i fellesdelen av deres masteroppgaver. Tomas Are Haavet videreutviklet plattrformen i sin masteroppgave[1](Rindex 0.2)

Da Rindex 0.1/0.2 ble utviklet, ble det implementert med begrenset funksjonalitet. Det ble også gjort endel valg i oppbygningen av datastrukturen, blant annet at det kun ble implementert støtte for statiske indekser. Dette medførte blant annet at databasen ikke støttet oppdateringer, noe som svært sjelden er optimalt for en database. Rindex støttet også kun et begrenset utvalg av SQL setninger.

Rindex ble videreutviklet i en påfølgende masteroppgave utført av Cecilie Haaland Fritzvold[2]. Her ble det gjort omfattende forandringer i Rindex' datastruktur. Det ble implementert en ny trestruktur for indeksene, kalt CSB+tre[3]. Det ble også implementert støtte for oppdateringer, samt at SQL vokabularet ble utvidet.

Denne versjonen er kjent som Rindex 0.3 og det er denne som danner basis for arbeidet med denne oppgaven.

1.1 Bakgrunn

Bakgrunnen for Rindexprosjektet er at dagens DBMS'er er utviklet på en tid da minne var en begrenset og ikke minst dyr ressurs. Slik er det ikke lenger. I dag er det vanlig at en ordinær hjemmemaskin har 2 - 4GiB minne. Det er heller ikke unormalt at servere har 16GiB eller mer. Når vi i tillegg

vet at det å søke i minne er 1.000.000 ganger raskere enn å søke på disk, kan vi tenke oss at det å ta i bruk minne mer aktivt i DBMS'er vil kunne gi en vesentlig ytelsesøkning.

Tidligere lå hele databasen på disk, og det å søke etter noe innebar mange disk I/O operasjoner. For å bøte på dette har det blitt utviklet DBMS'er som leser hele databasen inn i minne. Et eksempel på et slikt DBMS er SolidDB[4]. Dette gir god ytelse på spørringer, men egner seg dårlig på store databaser.

Et problem med denne typen DBMS er at ved systemfeil, strømbrudd eller omstart av maskinen vil data gå tapt dersom de ikke er skrevet til disk. Minne er ikke persistent[1], i motsetning til disk som har evnen til å lagre data selv når maskinen er slått av.

Rindex er utviklet med tanke på å kombinere dette ved å både ha egenskapene til det raske søket i RAM samtidig som dataene må kunne overleve en systemfeil eller en omstart av systemet. Dette oppnås i Rindex ved at alle sekundærindeksene legges i minne, samtidig som selve dataene og primærindeksene ligger på disk.

1.2 Formål med oppgaven

Ett av problemene med de tidligere versjonene av Rindex, er at man ved oppstart ikke har hatt anledning til å gi kommandoer eller utføre spørringer før hele databasen som ligger i bunn er indeksert. I denne oppstartstiden vil det for en bruker se ut som systemet henger.

Min oppgave har hatt som mål å se på mulige løsninger på dette oppstartsproblemet. Dette innebar videreutvikling av programkoden og testing av hvilken effekt endringene har for oppstartstiden for Rindex, og hvordan det oppfattes for brukeren.

I tillegg har Rindex til nå blitt testet mot en liten database på 550.000 tupler. Denne databasen ligger på Oracle. Parallellt med arbeidet med denne oppgaven ble det satt opp en ny testdatabase som inneholder 28 millioner tupler. Da dette ligger på PostGres vil det også være nødvendig å utvikle en ny frontend slik av vi kan benytte denne databasen ved testing. Vi får da også muligheten til å se på hvordan oppstartstiden til Rindex skalerer med størrelsen på databasen.

[1] Verdiene eksisterer kun så lenge det er strøm på.

Kapittel 2

Beskrivelse av Rindex

Her følger en beskrivelse av Rindex. Beskrivelsen er basert på versjon 0.3, som er utgangspunktet for mitt arbeid.

2.1 Oppbygging

Rindex er bygget opp av en rekke komponenter som til sammen utgjør plattformen. Sentralt i Rindex er biblioteket librindex som består av modulene: Scanner, Parser, Analyzer, Optimizer, IndexManager, Executor, DBSchema og DBFrontend. Her er det definert funksjoner som brukes til komunikasjon mellom Rindex og databasen. I tillegg benyttes tegnstrømmene *"in"*, *"out"* og *"err"*.

Klassen DBSchema bygges opp under oppstart av Rindex. Denne benyttes til å holde på informasjon som navn på relasjoner. For hver relasjon lagres:

- antall attributter og tupler
- antall attributter i primærnøkkelen
- navn og datatype til hvert attributt

I denne klassen finner vi også 4 vektorer som sammen utgjør en intern databasestruktur. Disse bygges opp under oppstart og knytter attributter og relasjoner sammen. Disse vektorene brukes så under oppslag. Mer om bruken av disse i 3.3.2.

For å kommunisere med databasen brukes klassen DBFrontend som har funksjoner for å gjøre spørringer og hente ut verdier av et resultat. Klassen er bygget opp som en abstrakt klasse slik at det skal være enkelt å utvide Rindex til å støtte flere DBMS'er. Idag støtter den kun Oracle, PostGres og CSV[1] filer. Med støtter menes at det er laget en frontendklasse for disse DBMS'ene

[1] Comma Separated Values

Indeksene blir håndtert av klassen IndexManager som tar seg av alle operasjonene på indeksene. Denne klassen kan enkelt byttes ut om man ønsker å teste ut en annen måte å håndtere indeksene på.

Scanner, Parser og Analyzer utgjør kompilatoren i Rindex. Den fungerer slik at scanneren først går gjennom kommandoen som blir gitt og ser etter nøkkelord[2]. Deretter går parseren gjennom spørringen og sjekker SQL syntaksen. Om den godkjenner syntaksen, utfører Analyzer en semantisk sjekk og utarbeider et syntakstre. Dette treet sendes så til optimizeren som oversetter dette til et spørretre, optimaliserer det og genererer en eksekveringsplan. Det lages kun en eksekveringsplan.

Til slutt sendes eksekveringsplanen til Executor som utfører selve spørringen og returnerer et resultatsett.

Rindex benytter seg også av tredjeparts APIet SQLAPI[5] for kommunikasjon mot DBMS. Dette har støtte for de aller mest brukte DBMS'ene. Kommunikasjonen med dette API'et blir gjort fra frontendklassen. Frontenden definerer alle systemspørringene som trengs for å hente den nødvendige metainformasjonen fra databasen. Disse spørringene sendes så til SQLAPI'et via et connection-objekt som utfører selve spørringen og returnerer det til frontenden.

2.2 Interaksjon

Man kan kommunisere med Rindex på to måter. Den første er en interaktiv modus som man starter fra kommandolinje under oppstart av Rindex. Man starter da Rindex med et -i flagg som forteller at man ønsker at Rindex skal lytte til tastaturet og ta kommandoer derfra. Dette benyttes i en test- og debuggingsituasjon. Dette kan i enkelte tilfeller skape problemer. Mer om dette i 3.3.4.

Den andre måten er den som brukes når Rindex kjøres som et DBMS, sammen med Oracle eller PostGres. Man kommuniserer da med Rindex via en kommandotolker som fungerer som en klient-server applikasjon. Kommandotolkeren kommuniserer med Rindex-serveren med en stop-and-wait protokoll som benytter en pipe-fil. Det fungerer på den måten at brukeren (klienten) gir en kommando til serveren (Rindex). Når kommandoen er gitt, må klienten vente på svar fra serveren før den har anledning til å gi en ny kommando.

Både klienten og serveren lytter på denne pipe-filen.

2.3 CSB+ trær

Ved oppstart genererer Rindex sekundærindekser av databasen som lagres i minne som et CSB+ tre. Disse indeksene eksisterer, som tidligere nevnt,

[2]Nøkkelord: Er det en spørring eller en kommando

kun i minne og lagres ikke ved avslutning av systemet.

CSB+ er en spesiell variant av et B+ tre hvor alle nodene er tilpasset cache-størrelsen til systemet som databasen kjører på. Ved å bruke denne datastrukturen oppnår man at man kun behøver å benytte seg av én skrive-/leseoperasjon mellom cache og minne for hver node som skal aksesseres. Rindex får cache-størrelsen gjennom en variabel som blir satt av et lite assembler program som kjører ved oppstart av Rindex. Denne verdien brukes ved generering av nodene i treet.

Et CSB+ tre gir raskere søk i minne enn hva et ordinært B+ tre gir og finnes i tre forskjellige varianter[3]. Dette er vanlige CSB+-trær, segmenterte CSB+-trær og fulle CSB+-trær.

Rindex benytter seg av fulle CSB+-trær og implementasjonen er beskrevet i [2]. Grunnen til dette er at de to førstnevnte er tregere enn B+ på innsetting i databasen. De er tregere fordi de løser splitting av noder på en annen måte enn hva fulle CSB+-trær gjør. Ved en splitting allokerer de ny plass i minne for så å kopiere over alle verdiene til den nye noden. Et fullt CSB+-tre løser dette ved at den, for hver ny node som opprettes, allokerer maksimal plass for en node med en gang. Dette medfører at det ikke er nødvendig med flytting av verdier når en node splittes.

En ulempe med denne varianten er at det blir allokert plass i minne som ikke nødvendigvis blir brukt.

Fordelen med å bruke et CSB+ tre er at det benytter seg av nodegrupper der nodene ligger etter hverandre i minne. Dette gjør at man kun behøver å lagre én peker til noden for så å benytte seg av et offset for å nå de andre nodene i gruppen. Dette gjør at man kan lagre dobbelt så mange nøkler i en intern node, så lenge en peker og en nøkkel tar like mye plass. De interne nodene inneholder:

- Antall nøkler
- Barn-peker
- Liste av nøkler

Bladnodene inneholder:

- Antall nøkkel/data-par
- Søskenpeker
- Liste med nøkler
- Liste med data

Et dataelement i en bladnode er en peker til et sted i minne hvor data ligger.

2.3.1 Søk

Søk i treet utføres på samme måte som i et vanlig B+-tre. Når nøkkelen er funnet, returneres det en iterator. Denne tillater iterering både fremover og bakover i treet og kan for eksempel brukes til å finne alle like verdier (noder på samme nivå), alle verdier som er mindre (noder til venstre for den returnerte noden) eller større verdier (noder som ligger til høyre for den returnerte noden).

Om det finnes flere like nøkler, returneres den første som blir funnet (lengst til venstre i treet). Dersom nøkkelen ikke finnes, returneres den første nøkkelen som er større. Finnes det ingen større, returneres den største i treet (noden lengst til høyre).

Kapittel 3

Oppstartsproblematikk

Et av problemene med de tidligere versjonene av Rindex er at man ved oppstart ikke har anledning til å gi kommandoer eller utføre spørringer før hele databasen som ligger i bunn er indeksert. Dette har blitt nevnt som et anbefalt punkt å se på i både [1] og [2]. Begge sier at valget om å la Rindex indeksere hele databasen før brukerne får tilgang, er gjort for at Rindex ikke skal få variabel ytelse. Med det menes at dersom Rindex kun indekserer de tabellene som det spørres på, så vil den første spørringen ta lenger tid enn de påfølgende. Dette kommer av at Rindex må indeksere relasjonen(e) før første spørring, mens de påfølgende bruker de(n) ferdig indekserte relasjonen(e) med en gang.

Hittil har Rindex blitt testet på et utsnitt av IMDB databasen[1]. Utsnittet består av filmer som har noe med Frankrike å gjøre. Databasen består av ca 550.000 tupler fordelt på 11 tabeller. Denne testdatabasen bruker Rindex ca 40 sekunder på å indeksere. Ved større databaser vil tiden skalere og man vil kunne risikere å måtte vente i mange minutter før man kan benytte seg av systemet. Når det i tillegg vil se ut som systemet henger i den tiden som Rindex bruker på å indeksere relasjonene, er dette ikke en heldig situasjon.

Dette gjelder likevel kun i de tilfellene man starter opp databasen første gang. Normalt vil databasen kjøre kontinuerlig mens brukere kobler seg opp til serveren ved hjelp av klienten[2]. Det er likevel ikke heldig at det tar så lang tid før man kan benytte seg av Rindex de gangene man må starte eller restarte systemet.

3.1 Mulige løsninger

Det er flere mulige løsninger på dette problemet. Vi skal begrense oss til to aktuelle løsninger. En av dem er allerede nevnt og omtalt i de to tidligere

[1] Internet Movie Database: www.imdb.com
[2] client.plx: Kommandotolkeren til Rindex

oppgavene.

3.1.1 Index on demand

En løsning er å la Rindex starte uten å indeksere noen relasjoner. Dette vil gi brukeren Rindex promtet med en gang. Rindex vil så sjekke hvilke tabeller som det blir spurt på og indeksere disse. Dette vil medføre at det vil kunne ta noe tid før svaret på spørringen blir gitt, men ved gjentatte spørringer mot de samme tabellene vil det fra andre gang kun være selve beregningen av resultatet som vil ta tid.

Det negative med denne løsningen er at det i de tilfellene hvor Rindex blir startet uten at det er noen som spør mot det, vil Rindex bare bli stående og ikke gjøre noe. Det ville da vært gunstig å la Rindex indeksere tabeller helt til noen startet en spørring. Dette er for å unngå unødvendig dødtid.

3.1.2 Tråder

En annen løsning er å gi brukeren tilgang til Rindex med en gang etter en omstart/oppstart samt at Rindex indekserer relasjoner i ledig tid. Denne løsningen innbærer bruk av tråder. Ved oppstart opprettes to tråder. Én lytter etter spørringer og én indekserer relasjonene. Dette vil medføre at i de tilfellene hvor serveren blir startet for første gang eller restartet, så vil den begynne å indeksere relasjonene med en gang.

Dersom en bruker starter serveren samtidig med at han ønsker å utføre spørringer mot databasen, så vil han i tillegg få opp Rindex-promptet med en gang og kan gi en spørring eller kommando.

Med tråder vil man fortsatt kunne havne i den situasjon at Rindex ikke har indeksert den tabellen det spørres etter enda. Man risikerer derfor å måtte vente på at Rindex har indeksert de nødvendige tabellene før resultatet av spørringen er beregnet. Dette kan løses ved å prioritere de relasjonene som det spørres på.

3.2 Implementering av tråder

Jeg valgte trådløsningen. Grunnen til dette er at jeg så på dette som den beste løsningen ved at Rindex da kunne starte å indeksere direkte ved oppstart samtidig som den var klar til å ta imot spørringer eller kommandoer. Siden det i tillegg er vanlig at servere har 2 eller flere prosessorkjerner, vil disse to trådene kjøre på hver sin kjerne og på den måten ikke konkurrere om prosessortid.

3.2.1 POSIX tråder

POSIX[6] er et trådbibliotek for C++. Disse trådene har et ferdig implementert bibliotek som man kan inkludere med filen *pthread.h*. Denne lar deg opprette tråder og har i tillegg noen grunnleggende metoder for trådmanipulasjon.

Man oppretter en tråd ved å kalle konstruktøren:

```
int pthread_create(pthread_t * thread,
                        const pthread_attr_t * attr,
                        void * (*start_routine)(void *),
                        void *arg);
```

med følgende parametere:

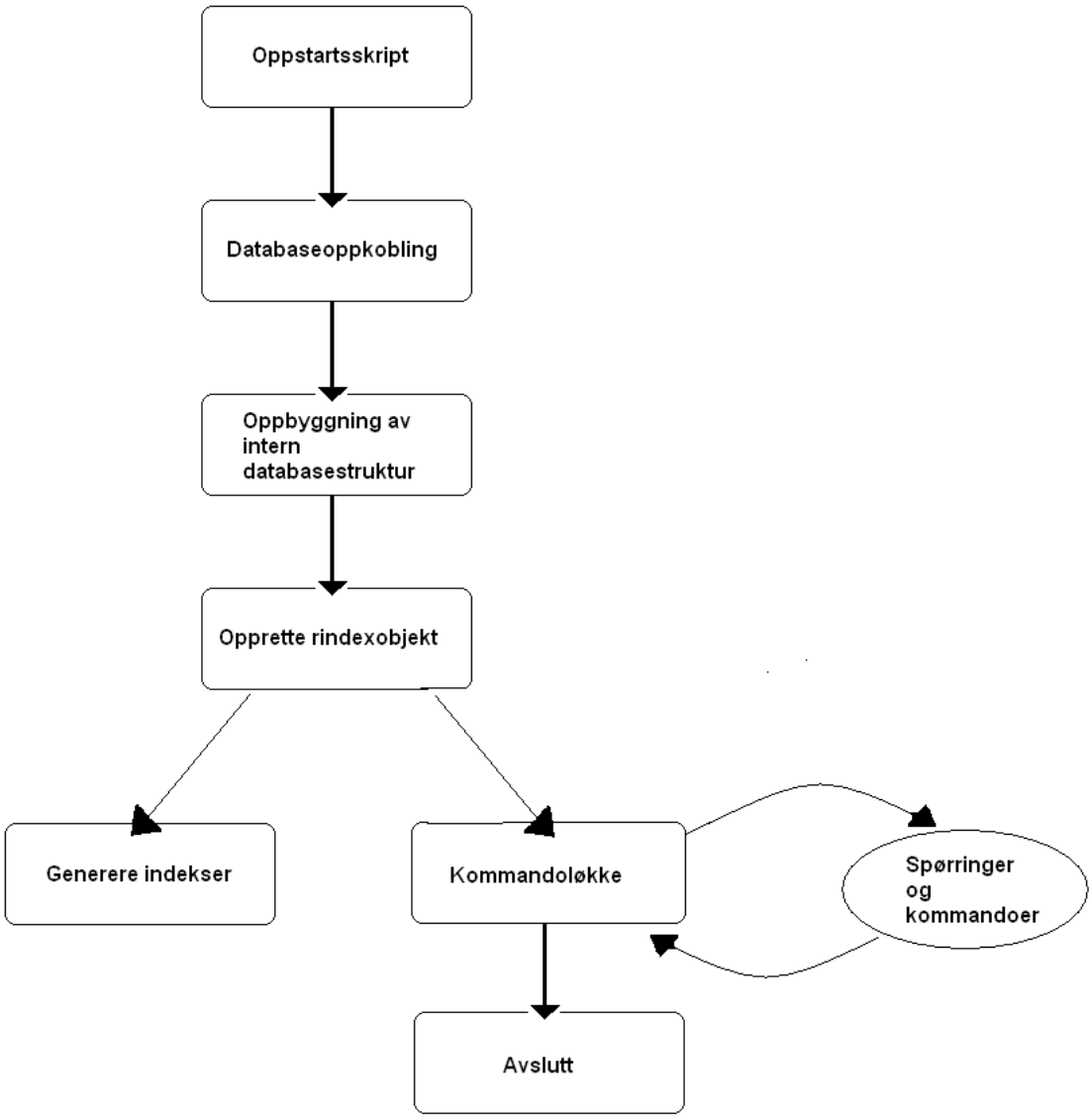
- pthread_t * thread: id'en til tråden som opprettes
- const pthread_attr_t * attr: settes til NULL dersom du ikke vil ta deg av avansert styring, som for eksempel skedulering, selv
- void * (*start_routine)(void *): Peker til funksjonen som tråden skal starte
- void *arg: eventuelle argumenter som den funksjonen tråden skal starte trenger. Dersom flere argumenter er nødvendig, må det sendes med en peker til en struct.

3.2.2 Implementasjon

Under implementasjon av tråder ble det eksperimentert med ulike løsninger. En av de gikk ut på å flytte selve opprettelsen av Rindex-objektet og kommandoløkka ut i egne funksjoner, for så å opprette 2 tråder som kalte hver sin funksjon. Dette fungerte ikke da den opprinnelige tråden avsluttet Rindex. Dette kunne ha blitt unngått ved å implementere synkroniseringslogikk, men istede ble det utviklet en løsning hvor kun opprettelsen av Rindex-objektet ble flyttet ut i en egen funksjon.

Dette viste seg heller ikke å være en god løsning da det er under opprettelse av Rindex-objektet at indeksene blir generert. Siden det er dette som tar tid, ville Rindex kun være istand til å motta spørringer og kommandoer men ikke effektuere dem før opprettelsen av objektet var ferdig.

Løsningen ble derfor å gjøre en endring i konstruktøren til Rindex objektet. Opprinnelig ble indekseringsmetoden kalt fra Rindexkonstruktøren. Ved å fjerne dette kallet, returnerte konstruktøren uten å indeksere. Man fikk da et fungerende Rindex-objekt å benytte i query-løkka. Indekseringsmetoden blir så kalt fra den andre tråden i Rindex-bin.cpp. Denne tråden bygger opp indeksene for Rindex-objektet. En enkel fremstilling av den nye programflyten er vist i figur 3.1.



Figur 3.1: Rindex 0.4 programflyt

3.2.3 Synkronisering

Jeg har valgt å kun opprette én tråd slik at Rindex har én spørre- og én indekseringstråd. Man kunne kanskje opprettet flere tråder, f.eks en for hver relasjon som skal indekseres. Dette har jeg valgt å ikke gjøre da det ikke fungerer med indekseringsalgoritmen, slik den er implementert idag. Mer om dette i 3.3.2.

I tillegg er det svært vanlig at en server har 2 - 4 kjerner. Ved å begrense antall tråder til å ikke overstige antall tilgjengelige kjerner, vil hver tråd trolig bli tildelt hver sin kjerne og dermed ikke bli skedulert bort til fordel for hverandre.

Når man jobber med tråder, er det viktig å passe på om de interakterer med hverandre eller ikke. Dersom de gjør det, må man hindre at de kan ødelegge for hverandre ved å aksessere delte ressurser samtidig.

I Rindex har vi nå to tråder: én som utfør spørringene og én som indekserer. Tråden som indekserer dør når den er ferdig. Den andre, kommandotråden, fortsetter helt til brukeren avslutter eller logger av Rindex. Dette medfører at så lenge den holder på med indekseringen, kan de to trådene jobbe på den samme delte ressursen, den interne databasestrukturen.

De to trådene vil komme i 'kontakt' med hverandre dersom det kommer en spørring før indekseringstråden er ferdig. For å hindre at dette skjer er det blitt implementert en MUTEX[3]. Denne er implementert som en metode, *is_swapping()*, i *DBSchema.cpp*. Denne returnerer verdien av en variabel, *SWAPPING*, som initialiseres til false ved oppstart.

Dersom Rindex mottar en spørring fra kommandotråden før indekseringstråden er ferdig, sjekker den først om den aktuelle relasjonen er ferdig indeksert. Dersom den er det, går den videre med parsing, optimalisering og utførsel av spørringen. Er relasjonen ikke indeksert, *rid_is_indexed()* returnerer false, vet vi at indekseringstråden ikke er ferdig ennå. Vi kaller derfor metoden *"is_swapping()"* som setter *"SWAPPING"* til "true". Dette hindrer indekseringstråden fra å gå videre til neste relasjon før den etterspurte relasjonen har blitt flyttet frem i køen.

Nå kan selve endringen i rekkefølgen begynne. Indekseringstråden kan nå enten være midt i en indeksering eller akkurat ha fullført og blitt blokkert. Vi sjekker først om den etterspurte relasjonen er den neste i køen for å bli indeksert. Dersom den er det, gjøres det ikke noe mer siden det ikke er mulig å prioritere den på noen måte. Dersom den etterspurte relasjonen ikke er den neste i køen, ønsker vi å flytte denne frem slik at den blir den neste som skal indekseres. Dette gjøres ved å kalle på swapmetoden (3.3.3) i DBSchema.cpp.

[3]Mutual exclusion: Sørger for at kun en tråd av gangen har tilgang til den delte ressursen

Dersom indekseringsalgoritmen i mellomtiden har avsluttet indekseringen av relasjonen den jobbet med, vil den nå bli stående og vente på at swappingen er fullført for så å gå videre. Dette er gjort ved å lage en whileløkke som tester på *is_swapping()* for så å sove i 2 millisekunder dersom den returnerer false:

```
/* Check if swapping is in progress */
while(dbschema->is_swapping()){
 sleep(2);
}
```

Jeg har valgt å la tråden sove i 2 millisekunder før den tester igjen. Dette er for å la den andre prosessen kunne gjøre seg ferdig raskest mulig. Et menneske vil uansett ikke merke 2 millisekunders forsinkelse.

3.3 Endringer i koden

Implementeringen av tråder innebar en del endringer i koden til Rindex. For det første så måtte det gjøres endringer i Rindex-bin.cpp. Det er her Rindex startes opp. Her opprettes Rindex objektet som inneholder alle indeksene samt at kommandoløkken kjører. Det er denne løkka som lytter etter spørringer eller kommandoer fra brukeren.

Siden hele programmet "styres" fra denne delen av koden, var det naturlig å implementere trådene her. De delene som måtte skilles fra hverandre, var selve oppbyggningen av Rindex-objektet og lyttingen etter interaksjon fra bruker. Den opprinnelige programflyten (Rindex 0.3) er beskrevet i figur 3.2.

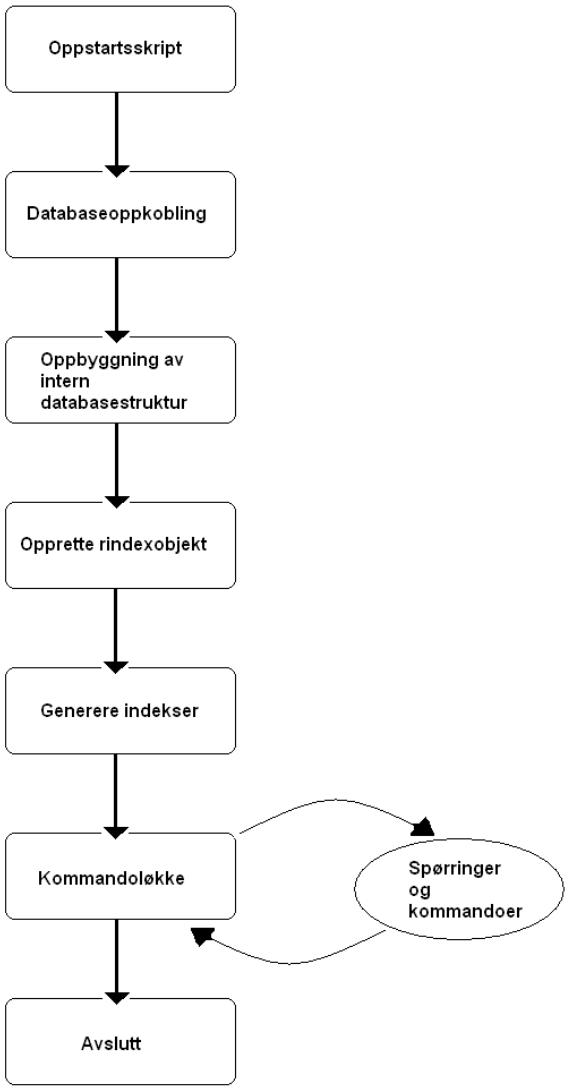
Som vi ser, opprettes først Rindex-objektet og indeksene genereres. Vi kommer så inn i kommandoløkka nederst i figuren. Her vil brukeren forbli helt til han gir kommandoen quit.

Kommandoløkka kjører ved å kalle query()-metoden i Rindexklassen. Denne venter på input fra bruker, parser, kompilerer og eksekverer spørringen ved hjelp av noen hjelpemetoder.

3.3.1 Endringer i relasjonsstrukten

Jeg har gjort en liten endring i relasjonsstrukturen. Det er lagt til et datafelt som sier om relasjonen har blitt indeksert eller ikke.

Dette feltet blir satt til "false" ved oppstart og blir satt til "true" når relasjonen er ferdig indeksert. Dette blir gjort i bunn av indekseringsalgoritmen, før den tester på om en "swap" er initiert.



Figur 3.2: Rindex 0.3 kontrollflyt

3.3.2 Endringer i indekserinsalgoritmen

For å få trådene og swappingen til å fungere som tenkt, var det nødvendig med noen endringer i indekseringsalgoritmen. Som nevnt tidligere, bygger Rindex opp en intern struktur på relasjoner og attributter under oppstart. Dette gjøres i DBSchema.cpp med de 4 vektorene beskrevet i 3.3.3.

Indekseringsalgoritmen, *init_indexes()*, er metoden som bygger opp indeksene for hvert atributt. Denne fungerte opprinnelig slik at den først gikk gjennom alle attributtene for så å gå gjennom alle relasjonene når den bygget opp indeksene. Da dette skjedde før den faktisk begynte å indeksere, kunne det medføre at vi måtte gjøre mange oppdateringer ved en eventuell swap.

Endringene jeg har gjort går ut på at det som tidligere var en relasjonløkke og en attributtløkke, nå er en ytre relasjonsløkke med en intern attributtløkke. Den indre attributtløkka tar kun for seg de attributtene som hører til den aktuelle relasjonen. For hver relasjon går den gjennom de aktuelle attributtene og indekserer disse.

Ved å gjøre dette sørger vi for at attributtene ikke blir plassert i indekseringsvektoren før de faktisk blir indeksert. Det vil derfor kun være nødvendig å bytte om rekkefølgen på de uindekserte relasjonene ved en swap.

For å sørge for at Rindex er klar til å motta kommandoer fra brukeren umiddelbart etter oppstart, er Rindex satt til å indeksere de relasjoene med færrest tupler først. Dette er gjort for at Rindex, så raskt som mulig, skal kunne prioritere de relasjonene som brukeren er interessert i.

3.3.3 Swapmetoden

Når Rindex-objektet opprettes, kjøres det en del spørringer mot den underliggende databasen. Disse spørringene sjekker blant annet:

- antall relasjoner som skal indekseres
- antall attributter for hver relasjon
- antall tupler i hver relasjon
- antall felter som er med i primærnøkkelen for hver relsjon
- hvilke attributter som er med i primærnøkkelen
- hvilke datatyper attributtene har

Dette benyttes til å bygge opp en oversikt over hva som skal indekseres, hvilken rekkefølge det skal indekseres i, og hvilke attributter som høre sammen med hvilken relasjon.

Dette gjøres i konstruktøren til *DBSchema*, som fungerer som komunikasjonsleddet mellom Rindex og databasen. Relasjonene og attributtene

knyttes sammen med interne ID'er og settes inn i 2 vektorer: *rel_id2data* og *attr_id2data*. For å gjøre oppslag i disse vektorene benyttes den interne ID'en som relasjonene og attributtene blir tildelt av Rindex. I tillegg finnes det to vektorer til: *rel_name2data* og *attr_name2data*. Disse fungerer på samme måte, men her kan man bruke navnet på relasjonen for å gjøre oppslag i *rel_name2data* og attributtnavnet for å gjøre oppslag i *attr_name2data*. Disse er satt til å peke på det samme som tilsvarende relasjon i de 2 andre vektorene peker på.

Det som må oppdateres ved en swap er den strukturen som er gjengitt i de tidligere nevnte vektorene. Disse brukes av klassen *"DBSchema"* ved oppslag under en spørring.

Swapmetoden kalles med ID'en til den relasjonen som skal flyttes frem som neste i køen, samt id'en til den relasjonen som er neste i køen. Så blir disse to relasjonene byttet om i *rel_id2data* vektoren, og *rel_name2data* vektorene blir oppdatert til å peke på den nye plasseringen i *rel_id2data* vektoren.

Neste skritt er å oppdatere attributtvektorene slik at disse blir knyttet opp mot riktig relasjonsID. Dette er en litt større operasjon da relasjonene ikke nødvendigvis har like mange attributter.

Et eksempel er at vi skal bytte om relasjonene country og filmparticipation.

```
hansmv=> \d country
   Table "public.country"
 Column  | Type | Modifiers
---------+------+-----------
 country | text |


hansmv=> \d filmparticipation
Table "public.filmparticipation"
  Column  |  Type   | Modifiers
----------+---------+-----------
 partid   | integer |
 personid | integer | not null
 filmid   | integer | not null
 parttype | text    | not null
```

Country består av kun ett attributt mens filmparticipation har fire. Siden alle attributtene skal ligge i sortert rekkefølge etter relasjonsid'er ser vi fort at det å flytte fire attibutter frem til et område i vektoren hvor det bare er en ledig plass, eller motsatt, vil by på problemer. Dette medfører at vi må flytte alle de andre attributtene, om noen, slik at vi lager eller fyller igjen ledig plass i vektoren.

Dette gjøres ved en rekke whileløkker som først henter ut de attributtene som skal flyttes frem og legger disse over i en temporær vektor. Så henter den ut alle attributtene som ligger 'mellom' de to attributtlistene som skal bytte plass. Deretter henter den ut de attributtene som skal flyttes bakover og til slutt de attributtene som er igjen. Helt til slutt flytter den attributtene tilbake i den opprinnelige vektoren.

3.3.4 Utskrift til fil

Implementajonen av tråder gir brukeren Rindex-promptet direkte ved oppstart. Tidligere var det slik at dette først kom etter at Rindex var ferdig med å indeksere databasen.

Under testing kom det frem at dette medførte at Rindex leste inn sin egen utskrift under indeksering. Rindex sin kommandotolker er lagt opp slik at dersom den ikke gjenkjenner kommandoen som blir gitt, så går den ut i fra at det er en spørring og sender denne videre til Rindex. Dette vil i noen tilfeller få Rindex til å avslutte med en feilmelding..

For å komme rundt dette problemet har jeg valgt å la all output gå til en egen fil, output.txt. Denne opprettes ved oppstart, eller fortsettes på dersom den eksisterer fra før. Man kan så bruke kommandoen:

```
tail -f output.txt
```

for å følge med på svarene fra Rindex. Et alternativ til dette kan være å utvide kommandoene som kommandotolkeren kjenner og på den måten filtrere bort kommandoer som ikke er spørringer, og dermed ikke sende disse til Rindex.

Kapittel 4

PostGresfrontend

Rindex har hittil kun støttet Oracle som DBMS i bunn[1]. Vi har også, frem til nå, kun kjørt tester på et utsnitt av IMDB-databasen. Dette utsnittet inneholder kun filmer som har noe med Frankrike å gjøre. Denne basen har 11 relasjoner og ca 550.000 tupler og ligger på en Oracledatabase. IFI har nå gått over til PostGres, og i den forbindelse har det blitt satt opp en ny versjon av IMDB databasen. Denne er vesentlig større og har 28 millioner tupler fordelt på 21 relasjoner[7]. Den største relasjonen, *filmparticipation*, består av 10.8 millioner tupler med 4 attributter. Den minste relasjonen, *country*, har 190 tupler med ett attributt.

For å kunne benytte denne databasen til testing, og samtidig øke test-datagrunnlaget betraktelig, trengtes en ny frontend.

Rindex ble opprinnelig designet slik at det enkelt skulle kunne utvides med flere underliggende DBMS'er. Dette er gjort slik at alle systemspørringene som går mot det underliggende DBMS'et er skilt ut i en egen fil. Dette er gjort fordi ulike DBMS ikke nødvendigvis benytter samme SQL-dialekt. I tillegg er det forskjell på hvordan de ulike DBMS'er bruker systemtabellene sine, og av den grunn må spørringene skrives om for å få ut den nødvendige informasjonen før Rindex kan indeksere databasen.

Valget mellom å bruke Oracle eller PostGres til test må man ta før man kompilerer Rindex. Siden frontendklassene er basert på templates, blir det problemer med inkludering av begge filene i Rindex under kompilering. Man må derfor ta bort linken til den frontenden man ikke ønsker. Det er uansett bare mulig å kjøre mot en underliggende database av gangen.

4.1 Nye metoder

For å få tilgang til PostGres databasen måtte alle systemspørringene, som Rindex trenger for å bygge opp indeksstrukturen, skrives om. Dette innbar

[1] Det støtter også CSV (Comma Separated Values) filer, men dette er ikke et DBMS. Det fungerer kun som en datakilde.

Datatype	Intern Rindextype
_text	TEXT
_int4	INT
_float4	FLOAT
_bpchar	CHAR

Tabell 4.1: Konvertering av PostGresdatatyper til Rindextyper

å sette seg inn i systemtabellene til PostGres som er organisert på en annen måte enn Oracle sine.

4.2 Nye datatyper

PostGres har noen andre datatyper enn hva Oracle har. Med andre datatyper menes at de heter noe annet enn hva de gjør i Oracle.

Tabell 4.1 viser hvilke datatyper fra PostGres som Rindex støtter per i dag. Det kan enkelt legges til flere datatyper, men det er kun disse som benyttes i den aktuelle databasen. Derfor er det kun disse datatypene som er implementert.

PostGres har et begrep, "UNIQUE", som Oracle ikke har. Dette benyttes for å angi at attributtet, eller flere attributter sammen, er unikt. Dette begrepet eksisterer sammen med det mer kjente "PrimaryKey" begrepet. I denne versjonen av IMDB er det kun "UNIQUE", og ikke "PrimaryKey" som er brukt for å angi primærnøkkel. Siden ikke alle feltene som er markert som "UNIQUE" nødvendigvis er en del av primærnøkkelen, må spørringen som henter ut alle primærnøkklene sjekke et internt array som holder kolonnenummeret til de attributtene som er en del av primærnøkkelen.

Kapittel 5

Testing

5.1 Formål og testmetode

I versjon 0.4 har Rindex fått skrevet om sin indekseringsalgoritme til å benytte tråder. I tillegg har Rindex fått en ny frontend mot PostGres. Vi må derfor teste at dette fungerer som det skal.

Vi må teste om trådimplementasjonen medfører at Rindex har fått en negativ utvikling av sin indekseringstid i forhold til hva den hadde tidligere. Med dette menes om den har blitt *vesentlig* tregere.

Dersom det skulle vise seg at Rindex nå bruker mye lengre tid på å bygge opp indeksene ved oppstart, kan det innebære at det ikke vil lønne seg å bruke en slik løsning.

Det må også testes om den nye frontend'en fungerer slik den skal. Dersom den ikke fungerer, vil det ikke det være mulig å kjøre noen tester mot den nye databasen.

5.2 Hva skal testes?

Vi skal se på hvordan oppstartstiden til Rindex 0.4 (flertråd) er i forhold til Rindex 0.3 (enkeltråd).

Vi skal også se på hvordan oppstartstiden til Rindex 0.4 blir påvirket når den mottar en spørring under indeksering.

5.3 Testdata

Rindex 0.3 og 0.4 vil bli kjørt på en dedikert maskin (rindex.ifi.uio.no). Denne maskinen ble satt opp spesielt for Rindex i april i år og benyttes ikke av noen andre.

PostGresdatabasen ligger på en lokal maskin, kurspg.ifi.uio.no, som er en dedikert PostGresserver. Denne ble ikke benyttet av noen andre mens testene pågikk.

Eneste ytre påvirkningen som kan forekomme er cron[1]-jobber på maskinen. Disse får man ikke gjort noe med grunnet manglende rettigheter på maskinen.

Filmdatabasen inneholder 21 relasjoner og totalt 28 millioner tupler. To av disse relasjonene måtte droppes da det ikke var mulig å opprette noen primærnøkkel på grunn av repeterende tupler. Den siste relasjonen, *filmparticipation*, måtte droppes da den er så stor at den ikke får plass i minne. Mer om dette i 5.4.2. Databasen testene er utført på innholder derfor 18 relasjoner med totalt 17 millioner tupler.

5.4 Problemer under testing

Under de første testene av den nye frontenden ble det raskt klart at 2GiB ram ikke er nok for å indeksere den nye testdatabasen. Bare 5 av de 21 relasjonene ble indeksert før minne var fullt. Dette var omtrent 9 millioner av totalt 28 millioner tupler. Dette var overraskende da beregninger på forhånd viste at det burde være nok med 2GiB ram.

Dette førte til at det ble satt opp en ny server for Rindex prosjektet, rindex.ifi.uio.no.

5.4.1 Nye servere

rindex.ifi.uio.no er en maskin dedikert til dette prosjektet. Den kjører en 64bits versjon av RedHat 5 og har 8GiB minne. Maskinen har 2 stk Intel Xeon E5320 prosessorer, som hver har 4 kjerner på 1.86GHz.

Når det først ble satt opp en ny testmaskin, ble det også investert i ekstra ram for å være sikker på å ikke støte på ytterligere minneproblemer.

Overgangen til 64bits-arkitektur viste seg å ikke bli helt problemfri. For det første ville ikke Rindex kompilere på 64bits plattform. Dette lot seg raskt rette da det stort sett var noen forskjeller på g++ kompilatoren mellom RedHat 4 og 5 som skapte problemene.

I tillegg måtte SQLAPI'et rekompileres for å fungere på 64bit. Dette viste seg å være så problematisk at det etter utallige forsøk ble satt opp enda en ny server for prosjektet, rindex2.ifi.uio.no.

rindex2 er en 32bit maskin med 8GB minne. Maskinen kjører RedHat 4. 32bits arkitektur og mer enn 4GiB minne har lenge ikke vært mulig, men Linux har fra 2.6-kjernen støtte for inntil 64GiB minne gjennom PAE[8][2] som øker minneadressene fra 32bit til 36bit.

Det viste seg likevel at selv om operativssystemet er 32bit og støtter mer enn 4GiB minne gjelder forstatt den kjente 4GiB begrensningen per prosess. Rindex klarte overraskende ikke å indeksere hele databasen selv med 4GiB

[1] Prosesser som maskinen selv kjører på fastsatte tider eller med gitte mellomrom

[2] Physical Address Extension

tilgjengelig minne. Mer om dette i 5.4.2. Dette gjorde det enda mer kritisk å få rekompilert SQLAPI'et på 64bits maskinen rindex.

Med god hjelp fra utviklerene av SQLAPI'et via epost, ble rekompileringsproblemet omsider løst, og SQLAPI'et er nå kompilert for 64bit på rindex.ifi.uio.no. Alle testene er utført på denne maskinen.

5.4.2 Minnebruk

Under testene ble det avdekket av Rindex bruker mye mer ram enn forventet. For en enkel tabell med en int og en varchar verdi bruker den ca 200Byte pr tuppel[3], noe som er overraskende mye. For en tabell med 3 verdier bruker den rundt 255Byte pr tuppel. Når vi har relasjoner på mange millioner tupler, medfører det høyt forbruk av ram.

Da den høye rambruken skaper problemer for å få kjørt tester på et fornuftig datagrunnlag ble Rindex kompilert med -g opsjon og kjørt gjennom Valgrind[9][4]. Valgrind starter Rindex og kjører gjennom koden en gang og teller antall minne allokeringer og deallokeringer. Resultatet er en rapport som forteller om eventuelle minnelekasjer og hvor disse er.

Da Valgrind testet Rindex, ble Rindex kjørt mot en enkel tabell, *film*. Dette ble for det første gjort fordi at Valgrind bruker lang tid[5] på å kjøre gjennom programmet, men også fordi de ulike relasjonene består av et ulikt antall tupler og attributter. Ved å begrense oss til en enkelt relasjon håpet vi at eventuelle lekasjer ville vise seg å stemme overrens med antall attributter og tupler. Relasjonen *film* ble valgt fordi den består av både int og text attributter, samt størrelsen på relasjonen: 692.361 tupler.

Rapporten viser at Rindex lider av store minnetap. Valgrind rapporterer at det i løpet av gjennomkjøringen ble gjort 10.427.085 allokeringer, mens det kun ble gjort 6.965.207 deallokeringer. Hovedårsaken til det store minnetapet rapporteres å være indekseringsalgoritmen.

I [1] skrives det på side 88 at Rindex er grundig sjekket for minnelekasjer med nettopp Valgrind. Det skrives at "eneste kjente lekkasje er i SQLAPI-biblioteket som Rindex benytter". Det er derfor grunn til å tro at lekkasjene oppstod under omskrivningen av indekseringsalgoritmen i [2].

Grunnet begrenset tid ble det dessverre ikke tid til å gjøre grundige forsøk for å rette opp i disse lekasjene. Testene er derfor utført under de allerede beskrevne forholdene.

[3] Beregnet ved kommandoen top i linux, og dividert på antall tupler for den indekserte tabellen

[4] Valgrind er et feilsøkingsverktøy spesiellt beregnet på å avsløre minnelekasjer i c++ programmer (http://valgrind.org/)

[5] 10 til 30 ganger lengre tid enn vanlig i følge dokumentasjonen

	Rindex 0.3	**Rindex 0.4**
`Min`	79.72 sekunder	101.18 sekunder
`Median`	79.95 sekunder	102.09 sekunder
`Max`	80.29 sekunder	106.10 sekunder

Tabell 5.1: Resultater av oppstartstider

5.5 Testene

5.5.1 Oppstartstid

Indekseringstiden er den tiden som Rindex bruker på å hente data fra databasen og bygge opp indeksene. Tiden er målt med Rindex' egen tidtakningsfunksjon. Resultatet fra denne er skrevet ut i fila *"output.txt"* som ligger i rotkatalogen til Rindex.

Tiden er målt 20 ganger på både Rindex 0.3 og Rindex 0.4. Testene er utført med følgende kommando:

```
./client.plx -- -m psql -t -u hansmv2 -p h4n5mv2
```

der -t angir at tiden for blant annet indekseringen skal beregnes. Denne kommandoen gis første gang man starter Rindex. Deretter gis kommandoene `stop` og `start` for å utføre målingen en gang til.

Målingene ble utført ved manuellt inntasting av stop og start da Rindex ikke støtter å hente annet enn sqlsetninger fra filer.

Ved måling av tiden med Rindex 0.4 rapporterte Rindex dobbelt så lang tid som det faktisk tok å indeksere. Dette kommer av at Rindex 0.4 har 2 tråder kjørende og tiden beregnes separat for begge trådene og legges sammen. Da den målte tiden kun gjelder for selve indekseringen, og at det kun er under selve indekseringen at det eksisterer 2 tråder, er derfor de rapporterte tidene delt på 2 for å kunne sammenliknes med Rindex 0.3

Resultater

Som vi ser av tabell 5.1, bruker Rindex 0.4 litt lengere tid på å indeksere enn hva Rindex 0.3 gjør. Det er viktig å huske at Rindex 0.4 lar brukere utføre spørringer og kommandoer umiddelbart, mens Rindex 0.3 ikke lar brukeren foreta seg noe før den er ferdig med indekseringen. Jeg vil derfor påstå at de 20 ekstra sekundene Rindex 0.4 bruker er en akseptabel økning i og med at man under hele indekseringen har mulighet til å benytte seg av systemet.

Ved begge tester var det første oppstart som tok lengst tid. Dette kommer trolig av at PostGres bufrer resultatene det beregner, og at det derfor går raskere de påfølgende gangene.

5.5.2 Spørring under indeksering

Med Rindex 0.4 har brukeren anledning til å gi spørringer til Rindex før den er ferdig med å indeksere. Dersom vi antar at en bruker gjør dette har vi tre mulige situasjoner når Rindex mottar en spørring:

- Relasjonen er allerede indeksert, og resultatet kan beregnes direkte
- Relasjonen holder på å bli indeksert, og vi må vente på at den blir ferdig før vi kan beregne resultatet
- Relasjonen må flyttes frem i køen slik at den blir den neste relasjonen som skal indekseres. Vi må da vente på at Rindex gjør seg ferdig med den relasjonen den holder på med, for så å indeksere den etterspurte relasjonen.

Vi skal derfor se på hvordan oppstartstiden påvirkes dersom Rindex mottar en spørring i disse tilfellene.

Gjennomføring

Testene er gjennomført vet at vi legger en spørring i en fil *"filnavn"*, som vi så laster inn i Rindex med kommandoen *load filnavn* samtidig som Rindex indekserer. Dette er gjort for å sikre at de samme spørringene blir kjørt hver gang, på omtrent samme tidspunkt i indekseringen. Alle spørringene i *filnavn* blir utført ved at Rindex leser inn spørringen og returnerer resultatet før den henter neste spørring.

Alle testene ble kjørt 5 ganger. Dersom det var stort sprik på den målte oppstartstiden så ble de kjørt nye 5 ganger, og alle 10 målingene ble sett under ett.

Spørring på ferdig indeksert tabell Spørringen i dette tilfelle er en seleksjon på relasjonen *film*. Denne er valgt fordi den er blandt de første relasjonen som blir indeksert. Dette gir oss tid til å kjøre spørringen(e) en eller flere ganger og så måle utslaget.

Spørring med swap Dette er en sammenslåing av de to siste mulige situasjonen, da disse i prinsippet er helt like. Spørringen i dette tilfelle er en seleksjon på relasjonen *person*. Valget av relasjonen kommer av at det er den nest største relasjonen, og derfor blir indeksert nest sist. Dette gjør at vi har litt tid på å kjøre spørringen en eller flere ganger for å se om vi får noe merkbart utslag. Vi får riktignok kun kjørt spørringen en gang før tabellen er indeksert, for Rindex gir ikke brukeren mulighet til å gi flere spørringer samtidig (se 2.2).

Resultater

Ingen av testene ga noe målbart utslag på oppstartstiden til Rindex. Alle de målte tidene var innenfor de samme verdiene som den allerede målte oppstartstiden var. Dette var egentlig forventet da Rindex har en egen spørretråd og derfor ikke bør påvirke indekseringen. Dette er selvfølgelig avhengig av at maskinen har én CPU[6]-kjerne til hver tråd. Dersom testmaskinen kun hadde hatt én CPU kjerne, ville man trolig kunne se utslaget siden trådene da ville konkurrere om CPU-tid.

Det forekom ett og annet markant avvik under testingen, men disse kan trolig forklares med at det samtidig ble kjørt en cron-jobb på maskinen.

[6]Central Processing Unit: Prosessoren

Kapittel 6

Oppsummering

Denne oppgaven har tatt for seg den lange oppstartstiden til de tidligere versjonene av Rindex. Oppstartstiden er den tiden det tar før brukeren har anledning til å benytte systemet etter oppstart eller restart.

Dette problemet er løst gjennom implementering av tråder. Med tråder har vi gitt brukeren anledning til å gi Rindex kommandoer og spørringer samtidig som Rindex indekserer relasjoner. Tester har vist at det kun er en liten økning i indekseringstiden til Rindex, og at denne økningen i tiden er akseptabel. Det har heller ikke gått utover den totalte oppstartstiden[1] i noen vesentlig grad.

Med trådene har vi også gjort at Rindex vil være tilgjengelig for bruker direkte etter oppstart eller restart av systemet. Trådene gjør det også mulig å få resultatet av spørringene før indekseringen av hele databasen er ferdig.

Før denne oppgaven har Rindex kun vært testet på en liten database. Det har derfor ikke vært mulig å si noe om hvordan Rindex ville yte på en stor database. Vi har sett hvordan ytelsen til Rindex er på en vesentlig større database. Testene som er gjennomført viser at Rindex klarer å indeksere 17 millioner tupler, fordelt på 18 relasjoner på rundt 100 sekunder. Dette må vi kunne si er meget tilfredstillende.

6.1 Videre arbeid

Gjennom arbeidet med oppgaven, har det blitt avdekket sider ved Rindex som danner grunnlag for videre arbeid. Slik som Rindex er idag, har det ikke en fornuftig bruk av minne når det blir kjørt mot en så stor database som vi har testet opp mot. Dette kommer trolig av at Rindex opprinnelig ble designet under forutsetning om at vi har mer enn nok minne. Dette har vi sett at fører til problemer når datamengden blir stor.

[1] Den tiden det tar å indeksere, bygge opp metainformasjon, koble seg opp mot databasen, osv

På bakgrunn av dette vil jeg trekke frem tre punkter for videre arbeid. To av dem er direkte tilknyttet minnebruken, mens det tredje er et punkt som vil gjøre Rindex til et mer komplett DBMS.

6.1.1 Minnelekkasjer

Siden Rindex bruker mye mer minne enn hva beregninger viser at det skal, er dette et viktig punkt å se på i det videre arbeidet. Testene som er utført viser at Rindex bruker opptil dobbelt så mye minne som det burde. Noe av minnebruken kan trolig forklares med de avdekkede minnelekkasjene. Dersom retting av disse medfører en halvering av minnebruken, vil det gjøre Rindex mer egnet på store databaser.

6.1.2 Omskriving av indeksstrukturen

Dersom retting av minnelekkasjene skulle vise seg å ikke medføre de helt store endringene i minnebruken, kan det være aktuellt å se på minnebruken i indeksstrukturen. I [1] ble det eksperimentert med forskjellige varianter av indeksstrukturer og det kan være aktuellt å se på de ulike konklusjonene og valgene som ble gjort der.

6.1.3 Utvide SQL vokabularet

Rindex støtter kun et begrenset utvalg av SQL. Ved å utvide vokabularet vil Rindex være i stand til å utføre større og mer komplekse spørringer. Det ville vært interessant å sett hvordan Rindex yter opp mot PostGres og Oracle på slike spørringer.

Bibliografi

[1] Thomas Are Haavet. Ram-basert indeksering – en effektivitetsstudie. Master's thesis, Institutt for Informatikk, Universitetet i Oslo, 2005.

[2] Cecilie Fritzvold. Databaseindekser i ram, en ytelsesstudie. Master's thesis, Institutt for Informatikk, Universitetet i Oslo, 2006.

[3] Jun Rao and Kenneth A. Ross. Making b+- trees cache conscious in main memory. *SIGMOD 00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pages 475-486*, 2000.

[4] http://en.wikipedia.org/wiki/soliddb.

[5] http://www.sqlapi.com/.

[6] http://www.yolinux.com/tutorials/linuxtutorialposixthreads.html.

[7] http://www.uio.no/studier/emner/matnat/ifi/inf1300/h07/undervisningsmateriale/filmdatabasen-orm-uml.pdf.

[8] http://en.wikipedia.org/wiki/physical_address_extension.

[9] http://valgrind.org/.