

Dokumentbaseret database

Implementering og optimering af en
dokumentbaseret database

Jonas Vittrup Biegel

01/01-1980

Semesterprojekt



PROFESSIONSHØJSKOLEN



Afdeling

Professionshøjskolen UCN

www.ucn.dk

Titel:

Dokumentbaseret database

Tema:

Semesterprojekt

Projektperiode:

Efterårssemstret

Projektgruppe:

Gruppe 9

Deltager:

Jonas Vittrup Biegel

Vejleder:

Brian Hvarregaard

Oplagstal: 1

Sidetal: 0.9 sider af 2083 anslag

Afleveringsdato:

01/01-1980

Resumé

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequale doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguere possit, augeri amplificari non possit. At etiam Athenis, ut e patre audiebam facere et urbane Stoicos irridere, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

Indhold

1. Indledning	1
2. Foranalyse	2
2.1. Database typer	2
2.1.1. Relationel database	2
2.1.2. Ikke-relationel database	2
2.2. Database struktur	3
2.2.1. B-tree	3
2.2.2. B+ Tree	3
2.3. Læsning fra disk	4
2.4. Programmering	4
2.4.1. Memory Leak	4
2.4.2. Use-After-Free	5
2.4.3. Buffer Overflow	6
2.4.4. Data Race	7
2.4.5. Noter	8
3. Kravsspecifikation	9
3.1. Krav til databasen:	9
3.1.1. Nice to have	9
4. Problemformulering	10
5. Metode(?)	11
6. Analyse	12
7. Design	13
8. Implementering	14
9. Test	15
10. Overdragelse (?)	16
11. Konklusion	17
Bibliografi	18

1. Indledning

Indlæning til rapporten

- databaser, datastrukturer etc
- optimering
- Memorysikkerhed

2. Foranalyse

En database er brugt næsten alle steder ude i den rigtige verden i næsten alt software. De bliver brugt til at opbevare en masse data, både informationer om f.eks. kunder men også relevante informationer om de systemer som programmerne holder i gang. Der er mange store database systemer som f.eks. PostgreSQL, SQLite og Microsoft SQL Server. Disse tre har det til fælles at de er relationelle databaser. Der findes dog også ikke-relationelle databaser, såsom MongoDB.

Disse databaseformer vil der nu analyseres nærmere.

2.1. Database typer

Der er to primære typer af database: Relationel database og en ikke-relationel database, også kendt som en NoSQL database (Not only SQL). Disse to har hver deres fordele og ulemper som der vil beskrives nærmere nu.

2.1.1. Relationel database

Relationelle databaser bruger primært det standard sprog for relationelle databaser, SQL. SQL er sprog der blev opfundet i løbet af 1970'erne til at interagere med databaser der indeholdte strukturerede data opbygget på relationel algebra. De bliver brugt i databaser med en tabelstruktur hvor alt data er gemt under et kolonnenavn. Dette gør det nemt at styre hvilke data hører til hvor. [1]

En stor fordel ved relationelle databaser er også at de opfylder ACID. ACID er et sæt af egenskaber som mange relationelle databaser opfylder under deres transaktioner. Dette gør at data kan redigeres og flyttes rundt uden anomalier som ender med at skabe forkert data. Derfor er disse databaser brugt meget i kommercielle programmer hvor der er brug for personstyring etc. [2]

2.1.2. Ikke-relationel database

Ikke-relationelle databaser, modsat relationelle databaser, er ikke tvunget til at være opsat i en tabelstruktur. Disse er databaser der typisk skal holde på meget data som man enten er usikker på størrelsen af, eller som er i en datastruktur som kan ændre hurtigt på sigt. De er også brugt steder hvor det er vigtigt at man kan læse og skrive data i så hurtig en hastighed som muligt. [3]

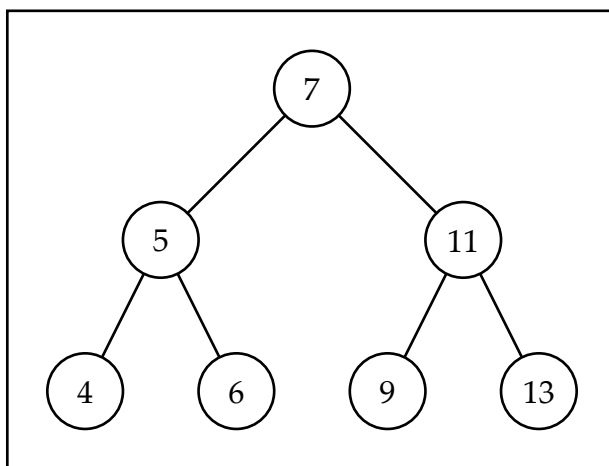
Et eksempel på noget software som bruger en ikke-relationel database er ting som Facebook og Twitter. Disse bruger databasen til at holde styr på ting som brugergenereret indhold, kommentarer og interaktioner. Disse databaser skal kunne skalere meget hurtigt og holde på mange forskellige typer af data, hvilket er derfor de bliver brugt. [3]

2.2. Database struktur

At gemme, læse og skrive så store mængder data kan være svært og kræver en god datastruktur. Disse strukturer er typisk baseret på en binær træstruktur. Den normale binære træstruktur har dog den ulempe at den ikke er særlig skalerbar, og at den i længden kan blive langsom i forhold til andre træstrukturer. Der er derfor blevet udviklet andre former for træstrukturer som er bedre til at gemme på store mængder data, og som skalere bedre i forhold til læse- og skrivehastighed. Disse vil nu beskrives yderligere.

2.2.1. B-tree

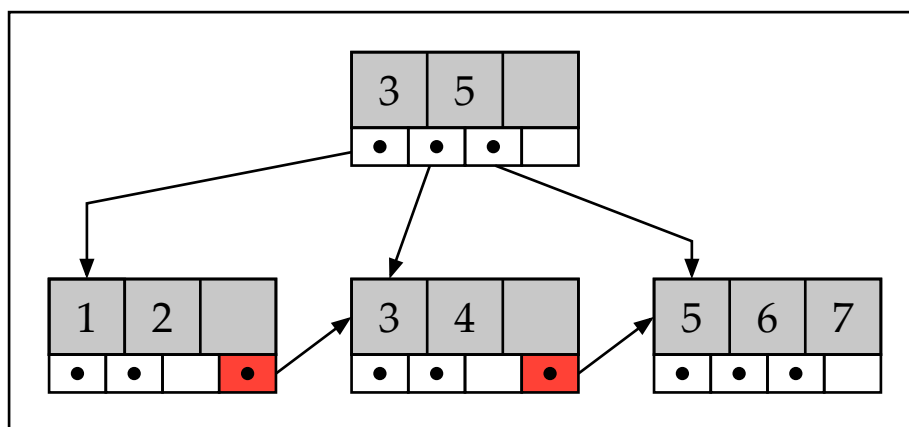
På Figur 2.1 kan et binært træ ses.



Figur 2.1: Binært træ med værdierne 4, 5, 6, 7, 9, 11, 13

2.2.2. B+ Tree

På Figur 2.2 Kan et B+ Tree ses.



Figur 2.2: Et B+ Tree med værdierne fra 1 til 7

Jeg finder information om B+ Tree fra [4].

2.3. Læsning fra disk

For at læse og skrive data fra og til disken, skal det data først ligge i hukommelsen af computeren. Hukommelsen læser disken i én "page" ad gangen, hvilket mindst er 4096 bytes på de fleste styresystemer [5]. Dette betyder at ens kode skal optimeres ved kun at læse de data som der er relevante for den operation der bliver kørt mod databasen, én page ad gangen.

- Side fra bogen hvor de snakker om pages
- Hvorfor pages?

2.4. Programmering

Mange databaser er i dag skrevet i programmeringssproget C. Dette sprog er "low-level", forstået som at man arbejder tæt med hardwaren af computeren. Dette er f.eks. ved at man selv skal allokerer hukommelse dynamisk. Dette gør også at ting kan optimeres rigtig meget, da sproget ikke selv bruger en garbage-collector til at sørge for at der ikke er nogle memory fejl, og ikke håndtere det i runtime.

De mest hyppige memory bugs vil nu beskrives, og noget kode der udløser fejlen vil vises.

2.4.1. Memory Leak

Et memory leak opstår når der bliver allokeret memory uden at det bagefter bliver frigivet. Dette kan gøres ved at lave en pointer og så give den en værdi. Dette vil medføre at det memory er optaget indtil slutningen af programmet.

Koden i Liste 2.1 viser et meget overdrevet eksempel på et memory leak.

```
1 int main() {
2     while(true) {
3         leak_memory(100);
4     }
5
6     return 0;
7 }
8
9 void leak_memory(int i) {
10     int *ptr = malloc(sizeof(int));
11     *ptr = i;
12     return;
13 }
```

Liste 2.1: En funktion i C der allokerer memory kontinuerligt

Hvis man er meget uopmærksom på sine pointers og ikke holder styr på at frigive dem efter de er brugt, ender det ofte i et memory leak. Disse kan ende med at være dyre, da de vil have en virkning på ydeevne, og i værste tilfælde vil crashe programmet.

2.4.2. Use-After-Free

```
1 int main() {
2     int *ptr = malloc(sizeof(int));
3     *ptr = 10;
4     free(ptr);
5     do_something(ptr);
6     return 0;
7 }
8
9 void do_something(int *ptr) {
10     // does something with the pointer ...
11 }
```

Liste 2.2: Et program der bruger memory efter det er frigivet

2.4.3. Buffer Overflow

```
1 #include <stdio.h>
2
3 int main() {
4     char buf[64] // create a buffer that holds 64 characters
5     gets(buf); // get input from user
6
7     return 0;
8 }
```

Liste 2.3: Et program der kan lave et buffer overflow

2.4.4. Data Race

```
1 #include <pthread.h>
2
3 int main() {
4     // define 2 threads
5     pthread_t thread1;
6     pthread_t thread2;
7
8     int shared_value = 0;
9
10    // make the threads do some operation on a shared value
11    pthread_create(&thread1, NULL, foo, &shared_value);
12    pthread_create(&thread2, NULL, bar, &shared_value);
13
14    // wait for threads to finish
15    pthread_join(thread1, NULL);
16    pthread_join(thread2, NULL);
17
18    return 0;
19 }
20
21 void* foo(void* arg) {
22     // read and write continually to and from arg ...
23 }
24
25 void* bar(void* arg) {
26     // read and write something else continually to and from arg ...
27 }
```

Liste 2.4: Et program der viser et data race

2.4.5. Noter

- Databaser
- Problemer i databaser (skrevet i C, C++, sikkerhed i memory)
 - SQL Server (C og C++)
 - PostgreSQL (C og C++)
 - MySQL (C og C++),
 - MongoDB (C, C++, JavaScript og Python(?))
 - SQLite (C)
- Strukturen af databaser måske?

```
1 fn main() {  
2     println!("lol");  
3  
4     for i in 0..100 {  
5         println!("lol {i}")  
6     }  
7 }
```

3. Kravsspecifikation

3.1. Krav til databasen:

- Dokumentbaseret, ikke relationel
- B+ tree for hurtige operationer på databasen
- Memory sikkerhed (sprogvalg = rust)

3.1.1. Nice to have

- API, sprogagnostik

4. Problemformulering

Der kan ud fra foranalysen og kravsspecifikationen udarbejdes følgende problemformulering:

Hvordan kan en dokumentbaseret database implementeres uden memory sårbarheder, og optimeres mest muligt?

Følgende underspørgsmål kan nu opstilles til problemformuleringen:

- Hvilken datastruktur er bedst til en database?
- Hvordan optimere man læsning og skrivning til databasen?
- Hvordan sikrer man sig mod memory sårbarheder?

5. Metode(?)

- Iterativt
- Benchmarking af de forskellige implementeringer
- Først en MVP, derefter optimering herfra

6. Analyse

- Træstrukturer
- Noget med hvordan memory læser disken i pages (4096 bytes ad gangen)

7. Design

Er ikke sikker med design

- Måske opbygningen af structs og modules? Kommer nok til at have et par
- Ved ikke med patterns, det er ikke fordi der er de store patterns i Rust kontra OOP
- Der skal måske tages hensyn til designet af træstrukturen der peger på forskellige børn

8. Implementering

- Vis implementering (kode)
- Vis noget fra modules måske, vis de vigtige dele af koden
 - Træstruktur
 - Skriv til filen i bytes, læs fra filen i bytes ned af strukturen med pointers til index i filen
- Måske et eksempel program der bruger databasen? Kunne være en todo list etc.
- Hvis jeg når så langt, så vis implementeringen af API'en og hvordan den tager generiske parametre, opretter tabeller og indsætter data etc.

9. Test

- Vis tests
 - Unit tests, integration tests, benchmarks

10. Overdragelse (?)

- Ved ikke hvad det her er

11. Konklusion

Konklusion af rapporten

Bibliografi

- [1] Wikipedia, "SQL". [Online]. Tilgængelig hos: <https://en.wikipedia.org/wiki/SQL>
- [2] Wikipedia, "ACID". [Online]. Tilgængelig hos: <https://en.wikipedia.org/wiki/ACID>
- [3] What Is A Database?, "What Is a Non-Relational Database? Understanding Its Key Features". [Online]. Tilgængelig hos: <https://www.whatisdatabase.com/what-is-a-non-relational-database-understanding-its-key-features>
- [4] A. Petrov, *Database Internals: A Deep Dive into How Distributed Data Systems Work*. O'Reilly, 2019.
- [5] Wikipedia, "Page (computer memory)". [Online]. Tilgængelig hos: https://en.wikipedia.org/wiki/Page_%28computer_memory%29