Advanced Database Design and Optimization

Mathias, Tobias & Robert

Explanation of the CAP theory and ACID properties, and how they were applied to the database design and implementation

CAP theorem

CAP dikterer at ethvert distribueret datalager kun kan give to af af følgende 3 garantier:

- C står for consistency. Det vil sige systemet ikke ville være i en illegal state, i forbindelse med transactions. Dette betyder at hvis en transaktion ikke kan fuldføres vil systemet gå tilbage til sidste kendte safe state.
- A står for availability. Dette betyder at en klient på hvilket som helst tidspunkt ville kunne få et respons fra systemet. I praksis betyder dette at hvis dele af systemet går ned, så vil resterende servere tager over indtil de dele som er gået ned bliver funktionsdygtige igen.
- P står for Partition tolerance. Dette betyder at systemet kan kan fortsætte med at operere, på trods af komplikationer, der kan komprimere systemets tilstand. Det kan både være netværksfejl, menneskelige fejl eller naturkatastrofer.

Man vælger kun to ud af de tre muligheder i CAP Theorem. Grunden til dette er, at det ikke er muligt at garantere alle tre ønskelige egenskaber på samme tid i et distribueret system. Som udgangspunkt er relationelle database CA. Dette betyder at den database vi har vagt har et fokus på consistency, altså at dataen i databasen altid er ens på tværs af servere, og availability, så vi sikre os at dataen er tilgængelig på alle tidspunkter

ACID

ACID står for atomicity, consistency, isolation og durability. ACID har stor betydning på datalagrings krav og planlægning. Det skal være med til at sørge for en pålidelig og konsistent processering af transaktioner i et databasesystem. Kort fortalt så handler ACID om transaktioner/commits, hvor at alle relationelle databaser kører efter disse principper.

Atomicity

- Alle ændringer til data der bliver udført skal ske som en enkelt operation. Det vil sige at alle ændringer skal udføres, ellers bliver ingen af dem udført.

Consistency

- Hvis databasen kommer i en ulovlig state, vil processen blive opgivet og vil rulle tilbage til sidste lovlige state
- Data skal være konsistent. Skulle noget data der ikke møder de foruddefinerede værdier komme ind i databasen, vil det resultere i en *consistency error* for datasættet. Database konsistent opnås ved nogle etablerede regler. En transaktion af data kan kun ændre dataen der er defineret af de etablerede regler, heraf constraints, triggers, variabler osv.

Isolation

- Ingen handlinger må påvirke hinanden

Durability

- Sørge for at data bliver gemt når der bliver committed

Challenges

Vi har siddet fast på vores UPDATE_BOOK stored procedure, grundet at vi prøvede at gøre den “smart”, ved at gøre alle parametre i en book optional. Det skal forstås, at intentionen var at vi ikke behøvede at skrive samtlige parametre ind, men kun dem der ønskes at skulle ændres. Vi fik det løst efter en del troubleshooting, og det endte med at køre som ønsket. Denne mindre udfordring gav rig mulighed for at lære nogle nye keywords, og også få bedre styr på SQL syntax. Undervejs i projektet har vi udelukkende lavet pair-programming, så en af vore større og mere irriterende udfordringer, har været at skulle migrere vores database internt mellem os.

Conclusion

Egentlig så har opgaven været meget givende, da vi alle rent praktisk har fået bedre erfaring med brugen af stored procedures, transactions og rollbacks - også med henblik på at styrke atomicity i vores database. Set fra et teoretisk synspunkt har vi fokuseret meget på ACID, og mindre på CAP, da (som også nævnt tidligere) relationelle databaser by default kører efter CA.