1. Lav alle sql script. (done)
   1. Book – flere sprog – flere grouper af årstal… 1000-1500-1600-1700-1800-1900-2000
   2. Book – 30000 bøger
   3. Rating -- Fiktiv ratings for alle bøger dvs. Vi havde 10000 men nu har 30000 bøger. Med user ids (fiktiv tilføj flere user id’er)
   4. User – alle user id’er fra ”book raiting” genere fiktiv names, country language
2. Mysql
   1. Trigger, partion osv.
3. MongoDb
   1. Find en måde at lave log ?
4. Backend
   1. Connect med mysql (X)
   2. Brug de trigger partion osv til at (sql syntax) (LinQ c#) ----(
   3. Connect med MongoDb (X)
   4. Lav log ud fra søgninger
5. Frontend
   1. Lav en enkel side hvor man kan se alle data
   2. Søgning (filter osv tjek exam pdf)

# Book

Behold de eksisterne 10k bøger fra den uploaded csv men tilføj og genere ny baseret på de cloumns vi har valgt: drop de andre cloumns alle columns skal være udfyldt selv de eksisterne 10k f.eks hvis årsal mangler så put random i som findes i forvejen.

Genere 20.000 esktra bøger med random med nye fictive original\_title, title osv men de skal selvfølige være nye i forhold dem som eksister i forvejen men language\_code, original\_publication\_year, average\_rating, ratings\_count og authors skal det være anderledes:

Authors: genbrug kun de eksisternde authors der er med i listen, som random til at udfylde de næste 20.000 fiktive bøgers aurthors.

Id: de første 10k samt de nye 20k. dvs alle. Skal have fra 1- 30000 sådan de unik og AI

Isbn: random

books\_count: random

work\_text\_reviews\_count: random

average\_rating. Lav fiktiv average\_rating som minder om de eksisterne fordel dem ligeligt ud.

ratings\_count. Lav fiktiv ratings\_count som minder om de eksisterne fordel dem ligeligt ud.

language\_code : lav nye language code for de nye 20000 bøger og fordel dem ud med de her tre nye language code: (Dansk,tysk og fransk)

original\_publication\_year: de nye år for de ekstra 20000 må gerne være random men baseret på de år som er brugt i forvejen sådan det er oplagt til partion sener.

Afslut det med at lave en csv file og sql file hvor jeg kan execute nemt dvs, opret tabel hvis den ikke eksister med navnet: Book

Rating af de 30000 bøger fra Book tabellen dvs. brug book id. alle columns skal være udfyldt (book id, user id og rating)  
lav en tabel der minder om denne Rating tabel, lav kun en csv file.

Hver bog skal rating være 50 gang

Den sammen bog må ikke rate af den sammen user.

Fordel random at en user har ratede mellem 10-20 bøger dvs. samme user ikke må lave rate mere end 20 gang.

Random rating skal være mellem 1 til 5

User id skal være unik og random

Den skal indholde alle 1.5mil ratings dvs for alle 30000 bøger. Det vigtigt at en hver bog bliver ratet og at deres id’er er korrekt.

User – alle user id’er fra ”book raiting” genere fiktiv names, country language

Lav User table med de her columns:

User\_id = Alle user id’er fra ”raiting” skal med I denne table

firstName: genere fiktiv first name

lastName: genere fiktiv last name

country: genere fiktiv max 20 lande (incl: england, tyskland, USA, Danmark,)

antal: rating dvs, baseret på de antal gang de har lavet i ”rating table”

Book table

Opret: Pertioner ?

Slet: når en bog bliv slette forsvind alle sine ratings (rating table) Pertioner?

Update: Pertioner? Updatere rating/rating count baseret på users samlet rating / antal gang

Pertioner:

Årstal: lave pertion på hvert 100 år. F.eks. 1400, 1500, osv.

Sprog:

Athour:

Rating table

Opret: Pertioner?

Slet: Pertioner?

Update: Pertioner?

Pertioner:

User\_id:

Book id:

Rating:

User table

Opret: Pertioner?

Slet: Pertioner? når en user bliver slette forsvind alle sine ratings (rating table)

Update: Pertioner? Den skal updatere de antal gang de har lavet rating i rating table.

Pertioner:

Land:

Intro:   
**Boganmeldelsesplatform (Book Reviews App)**

**💡 Kort beskrivelse**

En platform hvor brugere kan søge og anmelde bøger. Det bruges af:

* Slutbrugere: søger bøger og afgiver anmeldelser.
* Admins: overvåger data, kvalitet, segmenterer brugere og justerer indhold.
* System: analyserer data for anbefalinger og performance.

**Funktionelle krav (eksempler)**

* Søge og filtrere bøger på sprog, forfatter, årstal.
* Give anmeldelser (ratings) og opdatere dem.
* Automatisk beregning af gennemsnitlig rating pr. bog.
* Anbefale bøger baseret på brugerens adfærd (cluster/profilering).
* Sletning af bruger fjerner ratings.
* Sletning af bog fjerner dens ratings.
* Se statistik (top-rated books per årstal/land).

**Database Design med Polyglot Persistence**

Brug **to forskellige databaser**:

**1. Relational Database (PostgreSQL / MySQL)**

Til strukturerede data og relationer.

**Tabeller:**

* Books
* Users
* Ratings

**Partitionering (Table Partitioning):**

* Books:
  + **Range partition** på Year (hver 100 år)
  + **List partition** på Language
* Ratings:
  + **Composite partitioning** på User\_id og Book\_id
* Users:
  + **List partition** på Country

**Operations:**

* Ved sletning af Book: triggere sletter ratings (CASCADE)
* Ved sletning af User: triggere sletter ratings (CASCADE)
* Ved opdatering af ratings: triggere eller app logic opdaterer gennemsnit.

**Begrundelse**:  
Relationsdatabaser egner sig til **data-integritet**, transaktioner, joins og partitionering hjælper med **performance og vedligeholdelse**.

**NoSQL Database (MongoDB / Elasticsearch)**

Til fleksibel søgning, aggregering og hurtig adgang.

**Indhold:**

* Duplicerede data om bøger og brugere for hurtig søgning.
* Dokumenter som:

json

KopiérRediger

{

"book\_id": "123",

"title": "Moby Dick",

"language": "English",

"ratings": [

{"user\_id": "u1", "rating": 5},

{"user\_id": "u2", "rating": 4}

]

}

**Brug**:

* Fuzzy search på titel/forfatter.
* Aggregeringer (top-rated per sprog/år).
* Anbefalinger (segmentering, analyser).

**Begrundelse**:  
NoSQL er hurtig og fleksibel til **læsning og aggregering** – passer perfekt til søgning og anbefalinger.

**Dataflow og Integration**

* App skriver til begge databaser.
* Ratings opdateres i RDBMS og synkroniseres til NoSQL (eventuel brug af CDC eller batch).
* Periodisk sync for anbefalinger/statistik.

**🚀 Deployment og Performance**

* PostgreSQL med partitionering og evt. clustering (pgpool, Patroni, eller bare flere noder).
* MongoDB kan sættes op i **replica set** eller **sharded cluster**.
* Kør på localhost med docker-compose eller Kubernetes lokalt.
* Brug Postman til at demonstrere CRUD og queries.

**📊 Visualisering og Analyse**

* Brug Grafana eller MongoDB Charts til at vise aggregerede data.
* Eller et simpelt React-dashboard med API-kald.

**🎯 Afsluttende bemærkninger**

Du kan nu tydeligt vise:

* Hvilke **data** ligger hvor og hvorfor.
* Hvordan **partitionering** forbedrer performance.
* Hvordan **forskel i databasevalg** matcher behov (struktur vs. fleksibilitet).
* Hvordan **datakvalitet og integritet** sikres.
* Hvordan **brugere og admins** interagerer forskelligt med systemet.

**Fordeling mellem MySQL og MongoDB**

| **Database** | **Bruges til** | **Data** | **Begrundelse** |
| --- | --- | --- | --- |
| **MySQL** (relational) | Primær kilde til data og transaktioner | Books, Users, Ratings | ACID, data-integritet, relationer, joins |
| **MongoDB** (document) | Hurtig læsning, søgning og analyse | Kopieret og aggregeret bog- og brugerdata | Fleksibel struktur, hurtige aggregeringer, søgbarhed |

Trigger:

“Vi valgte at bruge TRIGGER i stedet for FOREIGN KEY ON DELETE CASCADE, fordi MySQL ikke tillader det sammen med partitionering. Triggerne sikrer stadig data-integritet automatisk og er udført direkte i databasen.”

Køre bat fil cmd: .\setup\_all.bat

Count i mysql: SELECT COUNT(\*) FROM bookreview.ratings;

**Validation i SQL**

**Sådan gør du det i SQL:**

sql

KopiérRediger

ALTER TABLE ratings

ADD CONSTRAINT chk\_rating\_range CHECK (rating BETWEEN 1 AND 5);

**🧠 Forklaring:**

* ALTER TABLE ratings = vi ændrer tabellen ratings.
* ADD CONSTRAINT chk\_rating\_range = vi giver constrainten et navn (chk\_rating\_range).
* CHECK (rating BETWEEN 1 AND 5) = selve reglen.

**🛠️ I MySQL Workbench (hvis du bruger GUI):**

1. Højreklik på tabellen ratings → **Alter Table**
2. Gå til fanen **"Constraints"** eller **"Check Constraints"** (hvis synlig)
3. Klik på **+** og tilføj:
   * **Name:** chk\_rating\_range
   * **Expression:** rating BETWEEN 1 AND 5
4. Klik **Apply**

**💡 Bonus:**

Du kan tjekke dine eksisterende check constraints med:

sql

KopiérRediger

SELECT table\_name, constraint\_name, check\_clause

FROM information\_schema.check\_constraints

WHERE constraint\_schema = 'bookreview';

SQL Qure:

**✅ Hvad gør søgning hurtig i MySQL (relationsdatabase)?**

| **Tiltag** | **Forklaring** | **Eksempel** |
| --- | --- | --- |
| ✅ **Indexes** | Gør WHERE, JOIN, ORDER BY, LIKE søgninger meget hurtigere | CREATE INDEX idx\_title ON books(title) |
| ✅ **Korrekte datatyper** | MySQL læser hurtigere INT og VARCHAR(50) end f.eks. TEXT | book\_id INT, ikke VARCHAR |
| ✅ **Enkle forespørgsler** | Vælg kun de felter du skal bruge | SELECT title FROM books WHERE language\_code = 'eng' |
| ✅ **Query Planning** | Brug EXPLAIN for at vise brug af indeks | EXPLAIN SELECT ... |
| ✅ **Skema-design** | Del data korrekt i normaliserede tabeller | users, books, ratings fremfor én stor tabel |

**✅ Hvad gør søgning hurtig i MongoDB (dokumentdatabase)?**

| **Tiltag** | **Forklaring** | **Eksempel** |
| --- | --- | --- |
| ✅ **Indexes** | F.eks. db.logs.createIndex({ search: 1 }) | Bruges ved søgefelter som search, user\_id, timestamp |
| ✅ **Fleksibel struktur** | Hurtig adgang til nested data | F.eks. { book\_id: 1, rating: 4, user\_country: "DK" } |
| ✅ **Projection** | Returnér kun de felter du skal bruge | { title: 1, \_id: 0 } |
| ✅ **Filtrering på objektfelter** | Hurtigt, da det er dokumentorienteret | find({ rating: { $gte: 4 } }) |
| ✅ **In-memory storage** (til logs, cache) | MongoDB kan holde hot data i RAM | Hurtigere adgang til ofte brugte søgninger |

“Vi bruger MySQL til stærk struktureret søgning – f.eks. ‘alle bøger med rating > 4’, hvor vi har index på average\_rating. Det sikrer hurtig access via query planner og B-tree indexes. Til ustruktureret logdata og søgehistorik bruger vi MongoDB med index på search, som er ideel til logs og fleksible forespørgsler.”

“Vi bruger LINQ i Entity Framework til at lave optimerede forespørgsler, som fx .Where(), .OrderBy() og .Include(). Det genererer effektiv SQL, som bruger de indexes vi har defineret i databasen – så søgninger forbliver hurtige og skalerbare.”

**1. Tilføj version kolonne (hvis du ikke har gjort det)**

ALTER TABLE books

ADD COLUMN version INT NOT NULL DEFAULT 1;

**2. Reservation med OCC – baseret på books\_count**

Den her del forgår I backend, forklar her lidt mere omkring vores setup.

Forklar hvorfor OCC og ikke PCC.

**MongoDB**

**lave 2 MongoDB collections**:

* booklogs (lejehistorik)
* reviews (kommentarer)

Så har I en fuld, sammenhængende og bestå-venlig løsning, uden at gøre det for komplekst.

Forklar fx i rapporten eller jeres kode:

* Brugeren (user\_id) og bogen (book\_id) er **den røde tråd**
* F.eks.:
  + Ratings i MySQL → detaljeret kommentar i MongoDB
  + Bøger i MySQL → lejehistorik i MongoDB

👉 Du behøver **ikke synkronisere automatisk**, bare vis logisk sammenhæng.

**3. 📄 Dokumentér valget i jeres synopsis**

Skriv kort:

* Hvorfor I valgte MongoDB til fx booklogs og reviews
* Hvad der gemmes hvor
* Hvilke typer queries MongoDB understøtter bedre (ubegrænsede kommentarer, fleksible datoformater, osv.)

**Vis 2-3 MongoDB-queries i jeres rapport eller demo**

// Find alle bøger user\_0 har lejet

db.booklogs.find({ user\_id: "user\_0" })

// Find alle kommentarer til bog\_id 1

db.reviews.find({ book\_id: 1 })

// Find alle anmeldelser med rating over 3

db.reviews.find({ rating: { $gt: 3 } })

API

Bog - MySQL:

Get – all books (with ratings) (: 286ms)

Get – 1 book (with raitings)

Delete – (se om trigger virker – fjern trigger fra database)

User - MySQL:

Delete – (se om trigger virker)

Rating - MySQL:

Post (med I den store function) ()

Delete (auto ved trigger)

Booklog - Mongo:

Get

Post (med I den store function)

Review - Mongo:

Get - all reviews

Get one review

Post (med I den store function)

“

Rent book, rating and review

ved at leje en bog, tjek med bog – mysql ved om den stadige er ledige hvis den er lej den, gøre den til 0 i mysql og gem info i mongoDb.  
ved aflevering af en bog, angiv en review samt rating af bogen og gem dem i hver sit database gøre den til 1 i mysql book og gem info i mongoDb booklog

”

**MySQL database engine.**

I want to group the following database operations together. If one of them fails, roll everything back. If all succeed, commit them

“I will update the book and only finalize the changes if everything goes smoothly — otherwise, undo it all.”

**Add Indexes on the Most-Used Filters**

Run these **once in MySQL** to improve query speed dramatically:

sql

KopiérRediger

CREATE INDEX idx\_books\_title ON books(title);

CREATE INDEX idx\_books\_average\_rating ON books(average\_rating);

CREATE INDEX idx\_books\_language\_code ON books(language\_code);

CREATE INDEX idx\_books\_year ON books(original\_publication\_year);

SHOW INDEX FROM books;