



# Databaseadministration og sikkerhed

## Opgavebeskrivelse

Denne uges opgave handler om at administrere forskellige grader af adgang til en database i en organisation, fx en webstore. Dette aspekt af datamanagement er kritisk, fordi data kan indeholde sensitivt materiale, der ikke skal deles på tværs af alle organisationsniveauer, kunder eller ledere. Noget der måske lyder selvindlysende den dag i dag, men bestemt ikke var det, inden man begyndte at regulere inden for området, eksempelvis som i [EU's GDPR \(General Data Protection Regulation\)](#) fra 2018.

Formålet med opgaven er at give en praktisk og refleksiv forståelse af det ofte indforståede arbejde, der ligger i:

- Hvorfor og hvordan man styrer adgang til forskellige typer data
- Hvordan man separerer mellem kundedata og virksomheds-/organisationsdata
- Hvordan man administrerer forskelle i adgangsbehov mellem brugertyper og organisationsniveauer
- Hvordan man tager højde for sikkerhedsaspekter i databasearkitektur og -vedligeholdelse fra starten

I skal opdele dataene, så forskellige medarbejdere og kunder kan tilgå relevant information om køb. Til det formål skal I også implementere en form for adgangskontrol, som I selv må bestemme, hvordan I løser. Det kan være via login, adgangstokens eller andet. Jeres kode skal udvikles på baggrund af jeres ETL-database fra uge 6 og 7.

---

## Afleveringsformat

Afleveringen til denne uges opgave består af:

- En individuel præsentation på fredag:
  - Varighed: 10-15 minutter



- Der skal være fokus på, hvordan I har håndteret adgangsrettigheder samt hvilke overvejelser I har haft i forbindelse med implementeringen af jeres DB-administration
  - Link til jeres Github-repository indeholdende:
    - Kildekode med kommentarer
    - En README.md-fil, der kort og udførligt beskriver hvordan man kører jeres kode.
- 

### **Casebeskrivelse**

Du er hyret som IT-konsulent for online-cykelhandleren *Bikestore*. De ønsker at øge graden af database-sikkerhed, og har flere grupper af medarbejdere, som skal have forskellige adgangsrettigheder.

---

### **Opgaver og opmærksomhedspunkter**

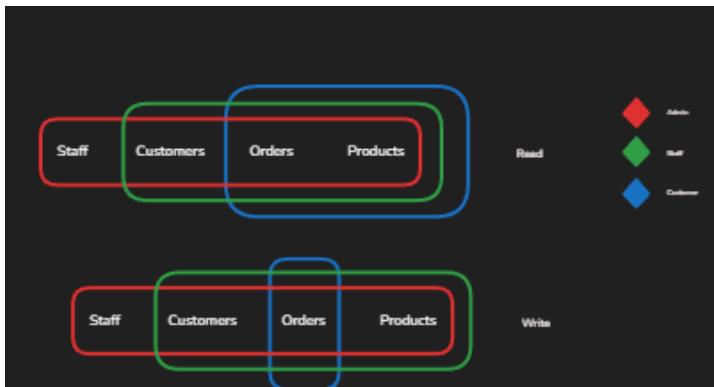
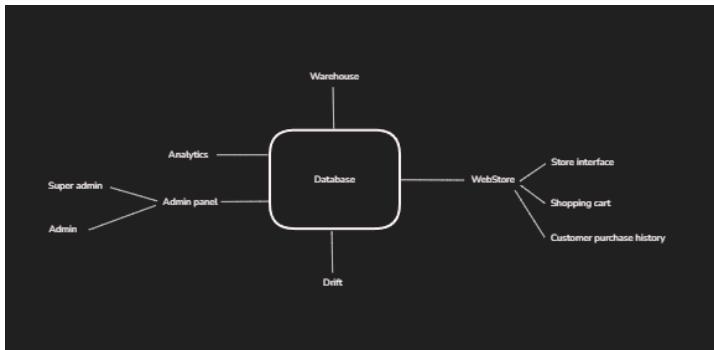
#### **Opgave 1:** Adgangskontrol og brugerroller

Implementér et system, hvor forskellige brugerroller har forskellige rettigheder til dataadgang i jeres Bikestore-database.

- Opdel medarbejderne i grupper, og giv dem forskellige rettigheder. Lav evt. undergrupper.
  - Hvilke adgangsrettigheder skal de forskellige grupper have og hvorfor?
  - Her er nogle eksempler på grupper, der skal have forskellige adgangsrettigheder:
    - Admin
    - Customer
    - Warehouse
    - Analytics
    - Andre?
- Find en måde hvorpå disse roller kan implementeres teknisk.
  - I kan fx. bruge SQL-kommandoer som GRANT, REVOKE, og CREATE ROLE til at styre rettigheder direkte i databasen



- I kan også bygge adgangsstyring ind i et Python- eller applikationslag “rundt om” databasen.
- Måske giver det mening at kombinere begge dele?
- Tænk også over, hvordan I dokumenterer og forklarer jeres tilgang. Hvilken rolle skal have adgang til hvilke tabeller eller felter – og hvorfor? Tag evt. Udgangspunkt i nedenstående diagrammer:



- Brug principper som *Least Privilege* (mindst mulige adgang) og *Separation of Duties* (adskillelse af ansvar) som grundlag for jeres overvejelser.

### ➊ Opgave 2: Sikkerhedsrisici og databaseskyttelse

Databaser er utsatte – både over for eksterne angreb og interne fejl. Derfor skal I tænke over, hvordan I beskytter jeres data mod ændringer, misbrug og utilsigtede hændelser.

- Implementér løsninger der beskytter databasen mod fjendtlige aktører



- Overvej hvordan I undgår angreb som SQL-injection – fx ved at bruge prepared statements eller stored procedures i stedet for at lade brugere sende direkte SQL-kommandoer.
- Andre løsninger?
- Opsæt foranstaltninger der modvirker at medarbejdere eller systemer laver utilsigtede ændringer.
  - I kan overveje at begrænse adgang til skriveoperationer for de fleste brugere, og måske kun lade bestemte roller køre opdateringer via godkendte procedurer.
  - Brug også gerne versionskontrol eller logging til at kunne spore ændringer
- Overvej hvordan I sikrer, at credentials som admin-adgangskoder ikke bliver lækket.
  - Dette kan evt. gøres ved brug af *secrets management*, f.eks. gennem .env-filer, vault-løsninger eller lignende – og del aldrig credentials direkte i jeres kode.
  - Ved at bruge sikker kodepraksis og tydelig rolleadskillelse kan I minimere risikoen for både tekniske fejl og brud på datasikkerheden

### **Opgave 3:** Håndtering af følsomme data og databaseseparation

I en database findes ofte flere typer data med forskellige grader af følsomhed – og derfor bør de også behandles forskelligt

- Sørg for at der skelnes mellem almindelige og følsomme data i jeres database
  - Tænk på tre overordnede datatyper:
    - Kundedata: navn, e-mail, købshistorik
    - Forretningsdata: lagerstatus, ordrebehandling, medarbejderdata
    - Personfølsomme data: helbredsoplysninger, CPR-numre, adgangslogs
- Sørg for at kun relevante personer har adgang til de mest følsomme data
  - Opdel evt. data i forskellige tabeller eller opret views, hvor man kun ser det nødvendige.
  - I kan også implementere adgangslogik i applikationslaget, der validerer, hvem der må tilgå hvad.
- Er der data, der er omfattet af GDPR eller virksomhedens interne fortrolighedsregler?

- Hvis ja, hvordan tager jeres design hensyn til det? Måske kræver noget data kryptering, anonymisering eller særskilt adgangskontrol.
  - Tænk på, hvordan databaseparation kan hjælpe med at undgå utilsigtet adgang – f.eks. ved at adskille kundedata fra interne medarbejderdata og derved ”siloisere” adgangen
- 

### Pensum og Ressourcer

Authentication: <https://frontegg.com/blog/authentication>

Secrets: [https://cheatsheetseries.owasp.org/cheatsheets/Secrets\\_Management\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html)

Login types: <https://uxdesign.cc/building-better-logins-a-ux-and-accessibility-guide-for-developers-9bb356f0a132>