Dimensionel modellering for Nedrivning

Indhold

[1. Indledning 3](#_Toc146192559)

[1.1 Dokumentets formål 3](#_Toc146192560)

[2. Krav 3](#_Toc146192561)

[3. Løsning 3](#_Toc146192562)

[3.1 Teknisk design 3](#_Toc146192563)

[3.1.1 Udtrækning 3](#_Toc146192564)

[3.1.2 Transformation 3](#_Toc146192565)

[3.1.3 Historik logning 3](#_Toc146192566)

# 1. Indledning

I virksomheder spiller korrekt og effektiv håndtering af data en afgørende rolle for at opnå forretningsmæssig succes. Det er derfor særligt vigtigt at implementere et velstruktureret og omfattende rapportlag i forbindelse med data modellering. Dette rapportlag udgør en central komponent i datamanagementstrategien og muliggør en dybere forståelse af virksomhedens performance og træffer bedre informerede beslutninger.

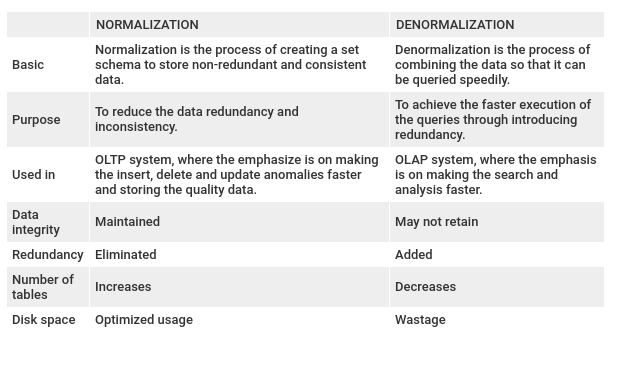
1. **Forretningsindsigt og Analyse**

Forretningsindsigt og Analyse: Et velfungerende rapportlag giver virksomhedens interessenter mulighed for at generere avancerede rapporter og udføre dybdegående analyse af data. Dette skaber en øget forretningsindsigt, hvilket er afgørende for at identificere trends, mønstre og nøgleområder, der kræver opmærksomhed.

2. **Beslutningsstøtte**

Beslutningsstøtte: Rapportlaget fungerer som en afgørende kilde til beslutningsstøtte for data på nedrivning. Det giver ledelsen og andre beslutningstagere mulighed for at trække på aktuelle og nøjagtige data, hvilket letter processen med at træffe strategiske beslutninger. Dette reducerer usikkerhed og øger troværdigheden af de beslutninger, der træffes.

Samlet set er et rapportlag afgørende for at optimere værdien af data modellering på BYGST-data. Det skaber et solidt fundament for datadrevne beslutninger og gør det muligt for en bredere kreds at arbejde med data.



[Database Normalization vs. Denormalization | by Ehsan Maiqani | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/database-normalization-vs-denormalization-a42d211dd891)

# 2. Teknisk design

Overordnet set består det tekniske design af tre dele per miljø, som illustreret i figuren ovenfor:

1. Først trækkes der rådata fra driftssystemet (Atrium) over til extract-databasen hos SIT.
2. Derefter transformeres data og stilles til rådighed i en forarbejdet udgave i staging-databasen hos SIT.
3. Resultatet er en datamodel, hvor entiteterne og kolonnerne har meningsfyldte navne og entiteterne er entydigt relateret til hinanden.

De tre skridt er specificeret nedenfor.

## 2.1 Udtrækning

Følgende tabeller hentes fra Nedrivning databasen, hvor der ikke er teknisk opsatte relationer mellem dem og hvor data foreligger i et normaliseret skema

|  |
| --- |
| **Tabel navn** |
| Adresser |
| Brevproces |
| Faser |
| Intern\_screening |
| Maengdeopgoerelse |
| Miljoeproces |
| Miljoeraadgiverdelaftale |
| Minkavlere |
| Minkavlere\_Minkfarme |
| MinkavlersRaadgiver |
| Minkfarme |
| Minkfarme\_Faser |
| Minkfarme\_Processer |
| Myndighed |
| Nedriver\_Kontaktpersoner |
| Nedrivere |
| Nedrivergrupper |
| Nedrivergrupper\_Nedrivere |
| Nedrivningsproces |
| Nedrivningstilladelse |
| Projektledere |
| TekniskRaadgiver\_Kontaktpersoner |
| TekniskRaadgivere |
| TekniskRaadgivergrupper |

### 2.1.1 Udtrækning

### 2.1.2 Transformation

For alle tabellerne gælder det, at det skal være hurtigere og nemmere for brugeren at arbejde med tabellerne. Data fra enkelte tabeller samles og sammensættes på anden vis end de er i rådata, så forretningsprocesserne overføres korrekt i dimensionelle tabeller.

For hver eneste rapporteringsbehov/måling bliver der ligeledes bygget en fact-tabel, der sammen med dimensions-tabellerne udgør et stjerne-skema.

Dokumentationen for dimensionelle, fact og stjerneskemaet findes i GitHub.

### 2.1.3 Historik logning

Der er blevet brugt Type 2 SCD (Slowly Changing Dimensions) for at bibeholde hele historikken fordelt på pågældende primær- og historiktabel.

Princippet for historiklogning er ens på tværs af entiteterne. Det finder sted ved at der i hver række i de tabellerne eksisterer et ”DW\_ValidFrom” og ”DW\_ValidTo” felt.

Hvis man eksempelvis har tre minkfarme vil data første gang ovenstående job kører komme til at se således ud:

|  |  |  |  |
| --- | --- | --- | --- |
| **CHR** | **Nedrivergruppe\_id** | **DW\_ValidFrom** | **DW\_ValidTo** |
| 93 | 1 | 01-01-2021 | 31-12-9999 |
| 253 | 2 | 01-01-2021 | 31-12-9999 |
| 88784 | 1 | 01-01-2021 | 31-12-9999 |

Lad os sige at en bruger den 10. januar ændrer. Næste gang vil indholdet af tabellen dermed blive:

|  |  |  |  |
| --- | --- | --- | --- |
| **CHR** | **Nedrivergruppe\_id** | **DW\_ValidFrom** | **DW\_ValidTo** |
| 93 | 1 | 01-01-2021 | 31-12-9999 |
| 253 | 2 | 01-01-2021 | 31-12-9999 |
| 88784 | 1 | **01-01-2021** | **10-01-2021** |
| 88784 | 2 | **10-01-2021** | **31-12-9999** |

Ændringer bliver registreret helt ned på klokkeslættet. Det vil sige, at der kan være flere ændringer i løbet af dagen. For, at nå til enighed om resultatet på dagen, er der implementeret en kolonne "DW\_MostRecentOnDay" der rangere ændringerne fra seneste til tidligste på dagen.

Eksempelvis når samme minkfarm har 4 forskellige ændringer på samme dag, så viser DW\_MostRecentOnDay = 1 (Den seneste ændring) DW\_MostRecentOnday = 4 (I dette tilfælde den tidligste ændring)

|  |  |  |  |
| --- | --- | --- | --- |
| **CHR** | **Nedrivergruppe\_id** | **DW\_ValidFrom** | **DW\_ValidTo DW\_MostRecentOnDay** |
| 88784 | 1 | 01-01-2021 01:00:00 | 10-01-2021 01:00:01 **4** |
| 88784 | 2 | 01-01-2021 01:00:01 | 10-01-2021 01:00:02 **3** |
| 88784 | 3 | 01-01-2021 01:00:02 | 10-01-2021 01:00:03  **2** |
| 88784 | 4 | 01-01-2021 01:00:03 | 10-01-2021 01:00:04 **1** |

Fordi vi hovedsagligt er interesseret i den seneste ændring, så logikken lavet således.

Logikken DW\_MostRecentOnDay = 1 for at kunne finde seneste ændring er ens på tværs af tabellerne. Den er bygget op på denne måde, fordi ved at vende logikken på hovedet vil seneste ændring i teorien kunne være 3 eller 4 afhængig af antallet af ændringer på dagen.