# Oplæg

Dette er den første løsningsmodel hvor vi sigter mod målet at opretholde keys som business keys. I denne løsning er der ingen ’\_id’ kolonner og nøgler realiseret ved automatisk inkrementering. Vi anvender straight business keys hvilket giver en del redundans i relaterede tabel, men det er prisen.

I et opfølgende oplæg løser vi opgaven med auto id kolonner.

* Det indledende om dataanalyse er selvfølgelig det samme uanset hvilken model man vælger at realisere.
* Men der er ret stor forskel når det gælder implementeringen, auto nummereringen tilføjer kompleksitet fordi vi når vi fylder tabeller med data, har behov for at kunne slå allerede oprettede rækker og deres id’er op, når vi fylder data med referentiel integritet over auto id kolonner.
* og der er fordele og ulemper ved begge tilgange.

Tendensen bevæger sig mod auto id løsninger, fordi det feks. kan være en forudsætning for at anvende frameworks og kodegeneratorer. I en auto id løsning kan man gennem en constraint stadigt opretholde en forventning om unikke værdier. Det er tit også nemmere at migrer data i auto id databaser.

# Forhistorie

Vi har indhentet data, og vi har etableret følgende databaser

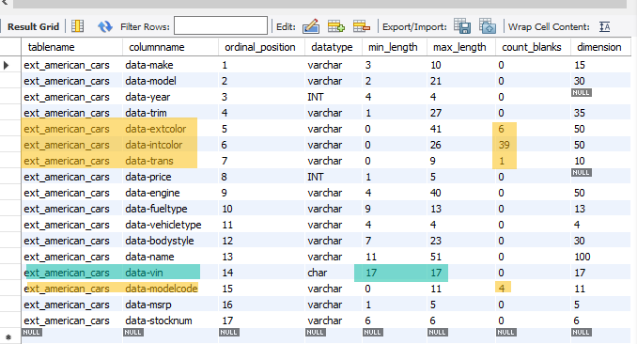
**Cars\_stage:**

En database vi bruger til at transformere data. Indlæsning og derefter typecasting samt hvis der havde været det behov, feks. afledte kolonner. Indlæsningstabeller hedder **’ext**\_<tabelnavn>’, ext for extract. Typede data findes i tabellen **’dat**\_<tabelnavn> ’, dat for data.

Database objekter der anvendes til transformering er navngivet som ´tfm\_<type>\_<tabelnavn> (et view og en stores procedure) pr. tabel. Nogle integrationer kunne godt indeholde flere tabeller, feks. kan man i et MS Excel dokument gemme flere faneblade. Hvert faneblad vil aflede behovet for en tabel. En anden form kunne være en zip fil.

**Cars\_template:**

En database der genererer databaseobjekter i feks. cars\_stage databasen. Man kunne have andre databaser med helt andre typer oplysninger, denne database vil md få tilpasninger kunne anvendes til etablering af integrationer på tværs af databaser, men så skulle databasen nok også have et andet navn…

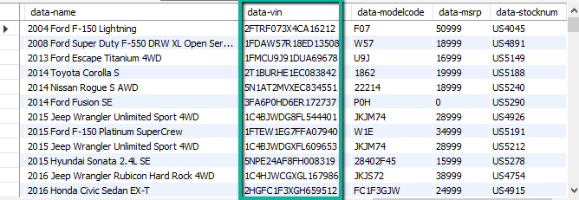
Tabellen template\_meta indeholder meta data om data, som vi fandt dem da vi skrabede bilforhandlerens webside. Der er mange bilmærker og modeller, men næppe alle varianter der findes på markedet. Der er de varianter han havde udstille på sin hjemmeside, da vi skrabede siden.

Ikke desto mindre vælger vi at sætte vores lid til det vi har fundet, måske vælger vi at gøre vores system lidt mere robust, feks. hvis vi forventer et tekstfelt ret sandsynligt kan rumme flere karakterer ind hvad vi lige ser repræsenteret (ved at selectere distinct på kolonnen vi interesserer os for). Af samme grund er vores extract tabel dimensioneret så den kan rumme meget længere tekststrenge. Strategien er at indlæse data fra fysisk fil kompromisløst. Ingen begrænsninger, ingen datatyper der ikke kan tåle bogstaver, ingen nøgler. Smidigheden i vores værktøj som cars\_stage databasen skal betragtes som, giver os muligheder for agilt hurtigt at kunne reetablere integrationen ved at danne nye tfm objekter, der virker efter hensigten efter vores nye viden om data.

# Tilgange til data analyse

**Finde nøgler**

Template\_meta har en kolonne der hedder ’count\_blanks’. Igen med udgangspunkt i de data vi kunne se, tælles der antal forekomster hvor der ikke er oplyst data. Umiddelbart kan vi med det samme afskrive disse kolonner som nøglekandidat.

Til gengæld er kolonnen data-vin interessant, alle forekomster er 17 karakterer lange, og data kunne godt ligne stelnumre. Så med denne kan vi identificere de enkelte forekomster af biler der stod til salg.

Vi kan afprøve vores tese med feks en sql som denne:

select count(distinct `data-vin`), count(`data-vin`)

from cars\_stage.ext\_american\_cars;

Forespørgelsen returnerer antallet af distincte forekomster og antallet af forekomster i det hele taget. Hvis disse tal er ens, er data i kollonnen UNIKKE og det vil opfylde kriteriet for at være en nøgle.

**Bruge din fornuft og kendskab til problemområdet**

Kolonnoverskrifterne bør selvfølgelig på en kort og præcis og forståelig måde beskrive data indholdet. Data i sig selv kan du genkende, hvis du har domæne viden.

**Finde domæner og subdomæner**

Når vi når hertil ved vi godt hvad der står i de enkelte kolonner. Vi ved sikkert også alle at man til næsten ethvert bilmærke kan få flere modeller, så det kan være vores første to tabeller.

Med de resterende kan overvejelserne være hvordan forholdet mellem engine og make/model mon forholder sig. Men det er nok sådan, at man ikke til enhver model kan få enhver motor. En V8 motor i en Ford Mustang kan ikke uden store modifikationer sidde i en Ford Ka. Så lad nu engine står i forhold til model.

Nogle kolonner indeholder data som ikke på nogen måde relaterer til make-model. Transmission feks. er tydeligtvis forhandlerens egen værdilister. De data vi ikke kan relatere lægger vi i datalister, og de vil relatere sig direkte til det enkelte køretøj.

# Etablering af databasen

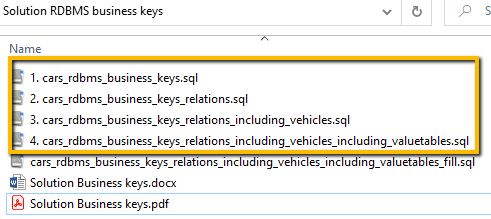
*cars\_rdbms\_business\_keys.sql*

|  |  |
| --- | --- |
| Man kan lægge ud med lavet tabellerne med business entiteter som er nødvendige for at etablere relationen.  Model står i forhold til make, ved at have make kolonnen i model, kan vi bruge make som fremmed nøgle i model. |  |
| ER diagrammet med nøgler uden relationer |  |
| Lav en relation med ikonet nederst til venstre, og se dialog boksen der dukker op. Følg anvisningerne: |  |
| Og du har en en til mange relation mellem make og model |  |

# Next step

|  |  |
| --- | --- |
| Næste step kunne så være at få genereret DDL ud fra modellen med **Forward Engineer**. |  |
| Følg dialog rækkefølgen i Forward Engineer to Database guiden, undervejs kan du se DDL kode som viser hvordan relationen er blevet etableret med DDL SQL  NB i en af dialogerne vælger man om man vil droppe eksisterende objekter. Default var ikke at overskrive, og så bliver databaen selvfølgelig ikke opdateret… |  |

Følg hvordan opbygningen af databasen sker iterativt og vi gemmer hvert Forward Engineer script for hver iteration.

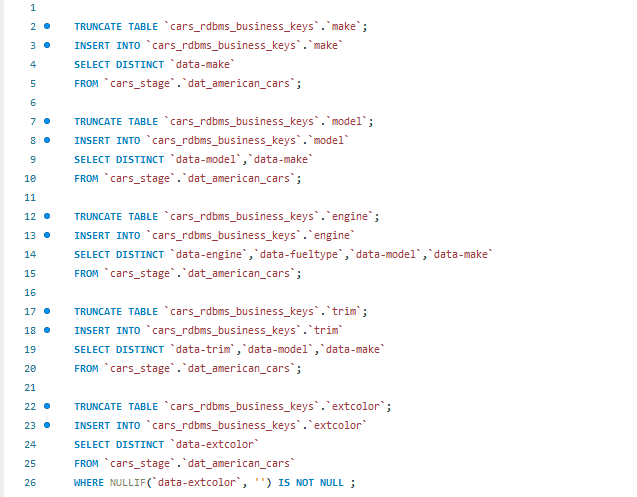


# Iterationer

|  |  |
| --- | --- |
| Måske foretrækker du code first tilgangen eller du foretrækker at arbejde videre i modellen (model first). Du kan bruge modellen og få genereret kode med Forward Engineer, eller du kan arbejde med koden og se resultatet af din indsats i ER diagrammet ved at bruge Reverse Engineer. |  |
| Her inkluderer vi værdilisterne og vores model er komplet |  |

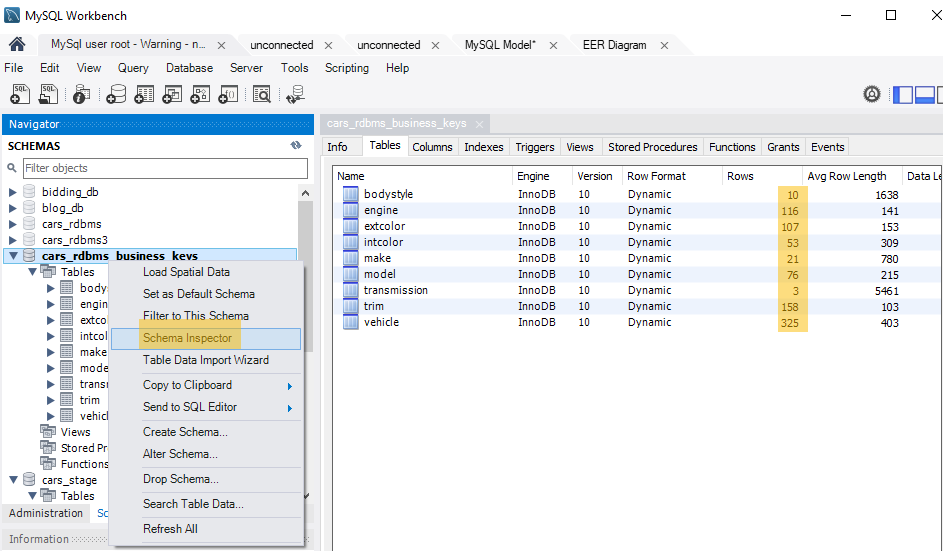
# Data fill

cars\_rdbms\_business\_keys\_relations\_including\_vehicles\_including\_valuetables\_fill.sql

Nu skal vi hælde data i vores database

Fordi vi bruger business keys er fill statements relativt ukompliceret

Med Scema Inspectoren kan du I fanen Tables se rækkeantal for hver tabel. Der skulle jo gerne være rækker i alle tabeller



**Ved som det endelige step, at hælde alle vores rækker fra** cars\_stage.dat\_american\_cars **til vores** vehicle **tabel, og gøre det uden fejl, er beviset for at vores relationer er intakte jf. modellen og scriptet.**

# Opmærksomhedspunkter

* Bemærk at kollonnen data-trans er oversat til transmission
* At DDL for vehicle tabeææen er identisk med DDL for cars\_stage.dat\_american\_cars
* Især at for kolonnerne feks. extcolor, intcolor gælder at de er nullable i vehicles tabellen – empty values accepteres
* At vi slet og ret importerer data til vores vehicle tabel med en plain select på cars\_stage.dat\_american\_cars
* At fill er lige ud af landevejen, fordi vi arbejder med business keys.