# Oplæg

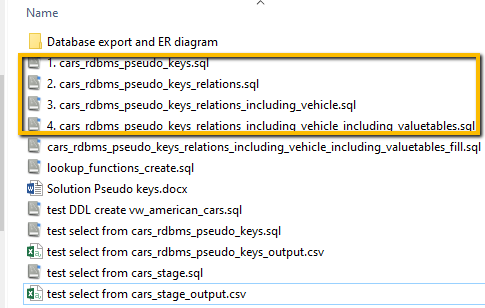
Dette er den anden løsningsmodel hvor vi sigter mod målet at opretholde keys som pseudo keys eller populært kaldet auto id’s. I denne løsning er der ’\_id’ kolonner og nøgler realiseret ved automatisk inkrementering.

# Etablering af databasen

*cars\_rdbms\_business\_keys.sql*

|  |  |
| --- | --- |
| Som ved business keys kan vi lægge ud med at lave tabellerne, denne gang med AUTO\_INCREMENT på ’\_id’ kolonner som udgører primær nøgler og fremmed nøgler i relationerne  Model står i forhold til make, ved at have make\_id kolonnen i model, kan vi bruge make\_id som fremmed nøgle i model.W |  |
| ER diagrammet med nøgler uden relationer  *1. cars\_rdbms\_pseudo\_keys.sql*  Bemærk at vi her vælger at lægge int- og extcolor i en tabel for sig selv, og lader tabellen relatere til make, med en antagelse om at farveskemaet på bilerne fra et givent mærke kan vælges på tværs af modeller. En Ford Ka og en Ford Mustang har dermed samme farveskema. |  |
| Brug modellen eller kode til at lave relationer  *2. cars\_rdbms\_pseudo\_keys\_relations.sql* |  |
| Inkluder vehicle realationer  *3. cars\_rdbms\_pseudo\_keys\_relations\_including\_vehicle.sql*  Bemærk at selv om vi laver auto id på tabellerne, laver vi en undtagelse med vehicle tabellen, der får ’vin’ som primær nøgle. |  |
| Og endeligt inkluder value lists  4. cars\_rdbms\_pseudo\_keys\_relations\_including\_vehicle\_including\_valuetables.sql |  |

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



# Data fill

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

|  |  |
| --- | --- |
| Inserts i realtionelle tabeller sker igen med select distinct, men vi har brug for at joine tabeller for at få fat i de rigtige id’er, der skal indsættes i fremmed nøgle kolonner.  Vi kan ikke bare som før trunkere pga. auto id kollonner. Når vi trunkerer slår vi SET FOREIGN\_KEY\_CHECKS fra |  |
| Og når vi når til vehicle tabellen, møder vi en lidt kompleks opgave, fordi vi for hver række har brug for at slå id’er op i i alt 7 tabeller.  Vi kunne lave en meget stor forespørgelse hvor vi inner joiner alle 7 relaterede tabeller, men det vil være komplekst og ikke særligt vedligeholdelses venligt.  Vi kan lave database funktioner der kan anvendes til opslagene. Database funktioner er objekter, der fungerer ligesom stored procedures, dog med den undtagelse at man inden i en funktion ikke kan opdatere data i tabeller. Funktioner har pr. definition ingen sideeffekter. |  |
| Med de 7 funktioner på plads kan vi indsætte data i vehicles tabellen med en DML som denne |  |

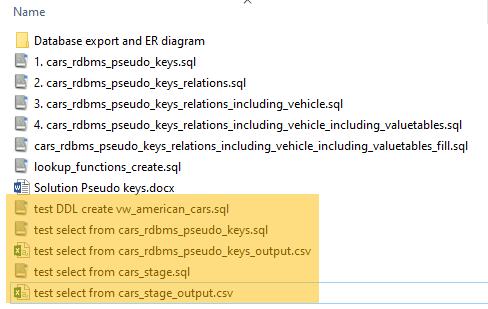
Med Scema Inspectoren kan du I fanen Tables se rækkeantal for hver tabel. Med få forklarlige undtagelser skulle der gerne være samme antal rækker i relations tabellerne uanset om vi ser på databasen cars\_rdbms\_business\_keys eller cars\_rdbms\_pseudo\_keys.

|  |  |
| --- | --- |
| cars\_rdbms\_pseudo\_keys | cars\_rdbms\_business\_keys |
|  |  |

# Test

Vi kan efterkontrollere at outputtet fra de to databaser (næsten) er ens. Der er forskelle vi genre skulle kunne forklare. Øvelsen går ud på at lave et udtræk fra de to databaser og sammenligne data ved at se efter forskelle i de to tekstfiler vi kan generere ifm. Vores undersøgelse

Filerne til denne øvelse er vist her:



|  |  |
| --- | --- |
| Her laver vi et view der joiner vehicles tabellen med alle 7 relaterede tabeller  *test DDL create vw\_american\_cars.sql* |  |
| Vi bruger viewet I forespørgelsen  *test select from cars\_rdbms\_pseudo\_keys.sql,*  Ligeledes laver vi en forespørgelse der henter data direkte fra cars\_stage. I denne forespørgelse håndterer vi tomme felter så resultatet bliver sammenligneligt.  *test select from cars\_stage.sql*  I begge tilfælde har vi en order by på alle kolonner. Vi er nødt til at være eksplicitter med rækkefølgen igen for at resultatet bliver sammenligneligt.  Rent faktisk forholder det sig sådan at en database returnerer rækker arbitrært. Det kan måske se ud som om at rækkefølgen altid er den samme når vi arbejder med data, men det kan man ikke tage for givet. Derfor Order by alle kolonner. |  |
| Vi kører begge forspørgelser og gemmer resultatet i csv-tekstfiler.  *test select from cars\_rdbms\_pseudo\_keys\_output.csv*  *test select from cars\_stage\_output.csv*  Nu kan vi sammenligne indholdet I filerne. Øverst ser vi at headeren er forskellig, det er nemt at forklare. |  |
| Vi ser også andre forskelle, som her hvor trim i den ene database står som ’Lariat’ men i den anden database står som ’LARIAT’.  Forklaringen er den, at når vi fylder data fra cars\_stage databasen, så selecterer vi distinct, og fordi det er arbitrært hvad databasen returnerer, får vi enten LARIAT eller Lariat, men kun en variant, fordi vores collation er ’ci’ case insensitive. |  |

# Opmærksomhedspunkter

* Bemærk at kollonnen data-trans er oversat til transmission
* At vi i DML fill anvender ’UNK’ for rækker hvor vi mangler data.
* At vi nu lader int- og ext color stå i en samletabel (color) og lader den relatere til make.
* At når importerer data til vores vehicle tabel sker det stadigt med opslag på cars\_stage.dat\_american\_cars, men id’er slår vi op i den relationelle database, hvor de er at finde med den nummerering id’erne fik i de første selects vi gør i DML fill scriptet.
* At når databasen returnerer rækker er rækkefølgen arbitrær medmindre vi styrer rækkefølgen feks. gennem en order by.