

Data analyse – Pseudo Keys

Vejen til en RDBMS database

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 udgør 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*

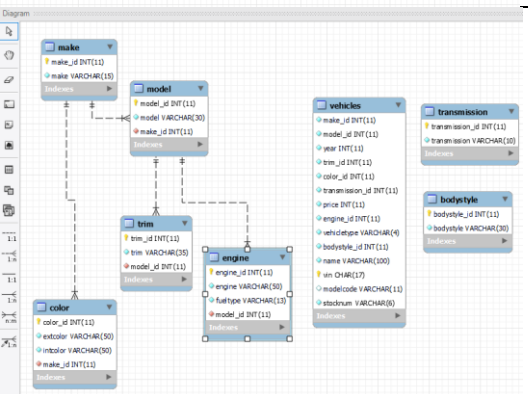
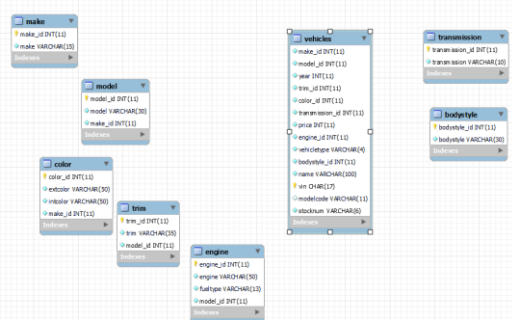
```
CREATE TABLE `engine` (
  `engine_id` INT(11) NOT NULL AUTO_INCREMENT,
  `engine` varchar(50) NOT NULL,
  `fueltype` varchar(15) NOT NULL,
  `model_id` INT(11) NOT NULL,
  PRIMARY KEY (`engine_id`),
  INDEX `fk_engine_model_id` (`model_id` ASC),
  CONSTRAINT `fk_engine_model_id` FOREIGN KEY (`model_id`) REFERENCES `cars_rdbms_pseudo_keys`.`model` (`model_id`)
) ENGINE = InnoDB DEFAULT CHARACTER SET = utf8;

CREATE TABLE `color` (
  `color_id` INT(11) NOT NULL AUTO_INCREMENT,
  `extcolor` varchar(50) NOT NULL,
  `intcolor` varchar(50) NOT NULL,
  `make_id` INT(11) NOT NULL,
  PRIMARY KEY (`color_id`),
  INDEX `fk_color_make_id` (`make_id` ASC),
  CONSTRAINT `fk_color_make_id` FOREIGN KEY (`make_id`) REFERENCES `cars_rdbms_pseudo_keys`.`make` (`make_id`)
) ENGINE = InnoDB DEFAULT CHARACTER SET = utf8;
```

```
DROP DATABASE IF EXISTS `cars_rdbms_pseudo_keys`;
CREATE SCHEMA IF NOT EXISTS `cars_rdbms_pseudo_keys` DEFAULT CHARACTER SET utf8 ;
USE `cars_rdbms_pseudo_keys`;

CREATE TABLE `make` (
  `make_id` INT(11) NOT NULL AUTO_INCREMENT,
  `make` varchar(15) NOT NULL,
  PRIMARY KEY (`make_id`)
) ENGINE = InnoDB DEFAULT CHARACTER SET = utf8;

CREATE TABLE `model` (
  `model_id` INT(11) NOT NULL AUTO_INCREMENT,
  `model` varchar(30) NOT NULL,
  `make_id` INT(11) NOT NULL,
  PRIMARY KEY (`model_id`)
) ENGINE = InnoDB DEFAULT CHARACTER SET = utf8;
```



Data analyse – Pseudo Keys

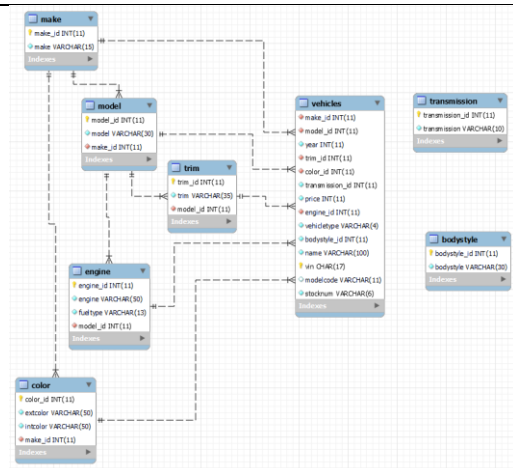
Vejen til en RDBMS database

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.

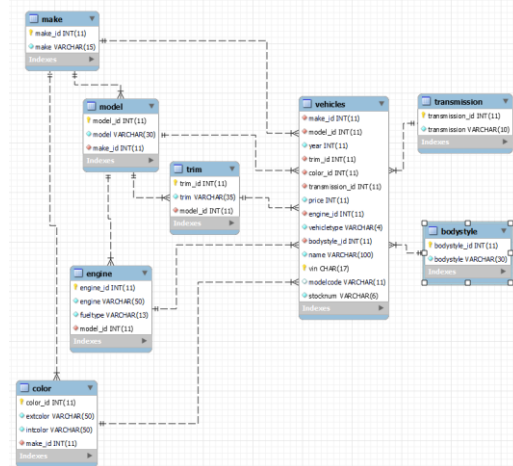
```
`bodystyle_id` int(11) NOT NULL,  
`name` varchar(100) NOT NULL,  
`vin` char(17) NOT NULL,  
`modelcode` varchar(11) DEFAULT NULL,  
`stocknum` varchar(6) NOT NULL,  
PRIMARY KEY (`vin`),  
INDEX `fk_vehicles_make1_idx` (`make_id` ASC) VISIBLE,  
INDEX `fk_vehicles_model1_idx` (`model_id` ASC) VISIBLE,  
INDEX `fk_vehicles_engine1_idx` (`engine_id` ASC) VISIBLE,  
INDEX `fk_vehicles_color1_idx` (`color_id` ASC) VISIBLE,  
CONSTRAINT `fk_vehicles_make1`  
FOREIGN KEY (`make_id`) REFERENCES `cars_rdbms_pseudo_keys`.`make` (`make_id`),  
CONSTRAINT `fk_vehicles_model1`  
FOREIGN KEY (`model_id`) REFERENCES `cars_rdbms_pseudo_keys`.`model` (`model_id`),  
CONSTRAINT `fk_vehicles_trim2`  
FOREIGN KEY (`trim_id`) REFERENCES `cars_rdbms_pseudo_keys`.`trim` (`trim_id`),  
CONSTRAINT `fk_vehicles_engine1`  
FOREIGN KEY (`engine_id`) REFERENCES `cars_rdbms_pseudo_keys`.`engine` (`engine_id`),  
CONSTRAINT `fk_vehicles_color1`  
FOREIGN KEY (`color_id`) REFERENCES `cars_rdbms_pseudo_keys`.`color` (`color_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```



Og endeligt inkluder value lists

4.

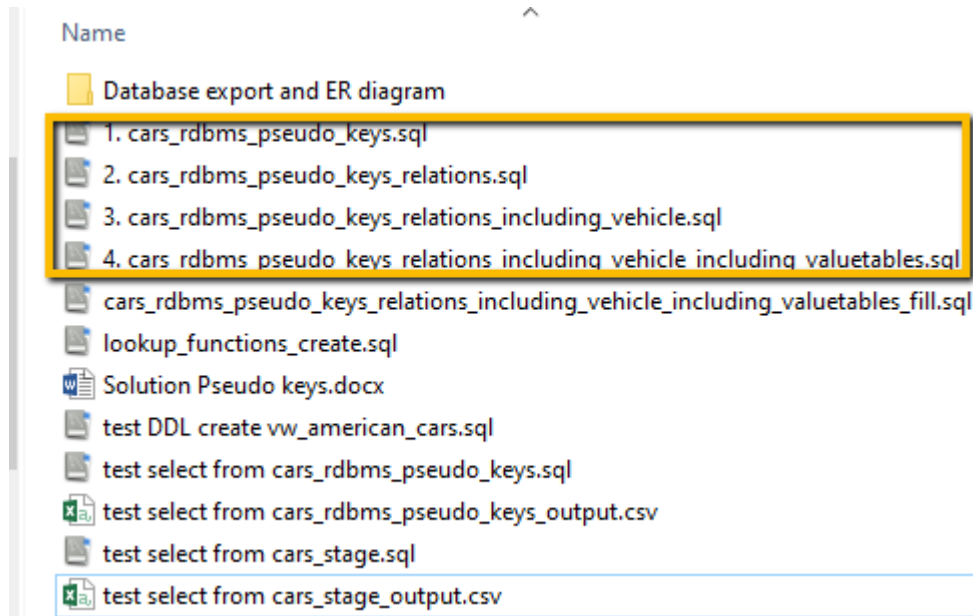
cars_rdbms_pseudo_keys_relations_including_vehicle_including_valuetables.sql



Data analyse – Pseudo Keys

Vejen til en RDBMS database

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 reationelle 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 kolonner. Når vi trunkerer slår vi SET FOREIGN_KEY_CHECKS fra

```
1 • USE `cars_rdbms_pseudo_keys`;
2 • SET FOREIGN_KEY_CHECKS = 0;
3 • TRUNCATE TABLE `make`;
4 • TRUNCATE TABLE `model`;
5 • TRUNCATE TABLE `trim`;
6 • TRUNCATE TABLE `engine`;
7 • TRUNCATE TABLE `color`;
8 • TRUNCATE TABLE `bodystyle`;
9 • TRUNCATE TABLE `transmission`;
10 • TRUNCATE TABLE `vehicles`;
11 • SET FOREIGN_KEY_CHECKS = 1;
12
13 • INSERT INTO `make` (`make`)
14   SELECT DISTINCT (`data-make`)
15   FROM `cars_stage`.`dat_american_cars`;
16
17 • INSERT INTO `model` (`model`,`make_id`)
18   SELECT DISTINCT dat.`data-model`, m.make_id
19   FROM `cars_stage`.`dat_american_cars` AS dat
20   INNER JOIN `make` AS m ON dat.`data-make` = m.make;
21
22 • INSERT INTO `trim` (`trim`,`model_id`)
23   SELECT DISTINCT dat.`data-trim`, mo.model_id
24   FROM `cars_stage`.`dat_american_cars` AS dat
25   INNER JOIN `model` AS mo ON dat.`data-model` = mo.model;
```

Data analyse – Pseudo Keys

Vejen til en RDBMS database

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.

```
1 USE `cars_rdbms_pseudo_keys`;  
2  
3 DROP function IF EXISTS `fn_lookup_make_id`;  
4 DELIMITER $$  
5 CREATE FUNCTION fn_lookup_make_id (make varchar(15))  
6 RETURNS INTEGER  
7 BEGIN  
8 DECLARE id INT;  
9 SELECT make_id INTO id  
10 FROM make  
11 WHERE make.make = make ;  
12 RETURN id;  
13 END$$  
14 DELIMITER ;  
15  
16 DROP function IF EXISTS `fn_lookup_model_id`;  
17 DELIMITER $$  
18 CREATE FUNCTION fn_lookup_model_id (make varchar(15), model varchar(30))  
19 RETURNS INTEGER  
20 BEGIN  
21 DECLARE id INT;  
22 SELECT model_id INTO id  
23 FROM model  
24 INNER JOIN make ON make.make_id = model.make_id  
25 WHERE model.model = model  
26 AND make.make = make;  
27 RETURN id;  
28 END$$
```

Med de 7 funktioner på plads kan vi indsætte data i vehicles tabellen med en DML som denne

```
INSERT INTO `cars_rdbms_pseudo_keys`.`vehicles`  
(  
  'make_id',  
  'model_id',  
  'year',  
  'trim_id',  
  'color_id',  
  'transmission_id',  
  'price',  
  'engine_id',  
  'vehicletype',  
  'bodystyle_id',  
  'vin',  
  'name',  
  'modelcode',  
  'stocknum')  
SELECT  
  fn_lookup_make_id(det.'data-make'),  
  fn_lookup_model_id(det.'data-make', det.'data-model'),  
  det.'data-year',  
  fn_lookup_trim_id(det.'data-make', det.'data-model', det.'data-trim'),  
  fn_lookup_color_id(det.'data-make', det.'data-vehiclecolor', det.'data-vehiclecolor'),  
  fn_lookup_transmission_id(det.'data-trans'),  
  det.'data-price',  
  fn_lookup_engine_id(det.'data-make', det.'data-model', det.'data-engine', det.'data-fueltype'),  
  det.'data-vehicletype',  
  fn_lookup_bodystyle_id(det.'data-bodystyle'),  
  det.'data-vin',  
  det.'data-name',  
  det.'data-modelcode',  
  det.'data-stocknum'  
FROM  
  'cars_stage'.det.american.cars AS det;
```

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.

<

Data analyse – Pseudo Keys

Vejen til en RDBMS database

Test

Vi kan efterkontrollere at outputtet fra de to databaser (næsten) er ens. Der er forskelle vi gerne 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:

Name
Database export and ER diagram
1. cars_rdbms_pseudo_keys.sql
2. cars_rdbms_pseudo_keys_relations.sql
3. cars_rdbms_pseudo_keys_relations_including_vehicle.sql
4. cars_rdbms_pseudo_keys_relations_including_vehicle_including_valuetables.sql
cars_rdbms_pseudo_keys_relations_including_vehicle_including_valuetables_fill.sql
lookup_functions_create.sql
Solution Pseudo keys.docx
test DDL create vw_american_cars.sql
test select from cars_rdbms_pseudo_keys.sql
test select from cars_rdbms_pseudo_keys_output.csv
test select from cars_stage.sql
test select from cars_stage_output.csv

Her laver vi et view der joiner vehicles tabellen med alle 7 relaterede tabeller

test DDL create vw_american_cars.sql

```
CREATE VIEW 'american_cars' AS
SELECT
  'make'.'make',
  'model'.'model',
  'v'.'year',
  'tr'.'tr' as 'tr',
  'color'.'extcolor',
  'color'.'intcolor',
  'transmission'.'transmission',
  'v'.'price',
  'engine'.'engine',
  'engine'.'fueltype',
  'v'.'vehicletype',
  'bodystyle'.'bodystyle',
  'v'.'name',
  'v'.'vin',
  'v'.'modelcode',
  'v'.'stocknum'
FROM
  'vehicles' as v
INNER JOIN 'make' on 'make'.make_id = v.make_id
INNER JOIN 'model' on 'model'.model_id = v.model_id
INNER JOIN 'tr' on 'tr'.tr_id = v.tr_id
INNER JOIN 'color' on 'color'.color_id = v.color_id
INNER JOIN 'transmission' on 'transmission'.transmission_id = v.transmission_id
INNER JOIN 'engine' on 'engine'.engine_id = v.engine_id
INNER JOIN 'bodystyle' on 'bodystyle'.bodystyle_id = v.bodystyle_id
```

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 eksplicit med rækkefølgen igen for at resultatet bliver sammenligneligt.

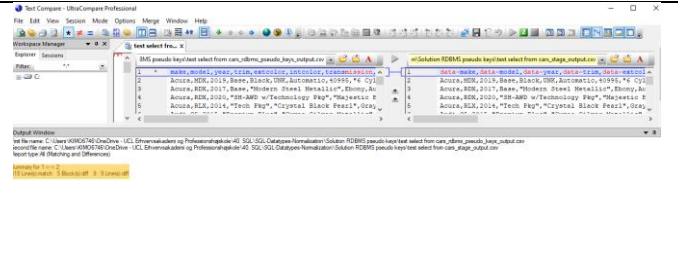
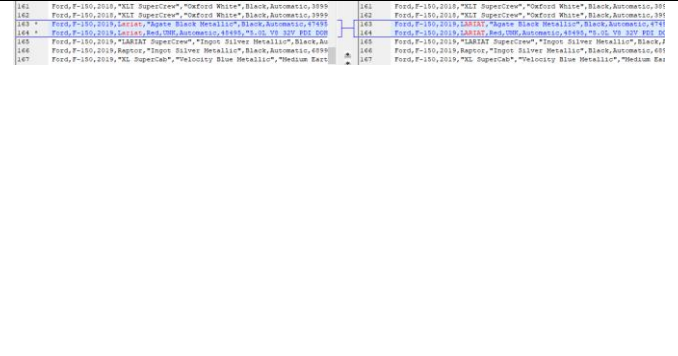
Rent faktisk forholder det sig sådan at en database returnerer rækker arbitrært. Det kan måske se ud som om at rækkefølgen altid er den samme når vi arbejder med data, men det

```
test select from cars_rdbms_pseudo_keys
1 SELECT 'make',
2 'model',
3 'year',
4 'tr',
5 'extcolor',
6 'intcolor',
7 'trans',
8 'price',
9 'engine',
10 'fueltype',
11 'vehicletype',
12 'bodystyle',
13 'name',
14 'vin',
15 'modelcode',
16 'stocknum'
17 FROM 'vw_american_cars'
18 ORDER BY 'make',
19 'model',
20 'year',
21 'tr',
22 'extcolor',
23 'intcolor',
24 'trans',
25 'price',
26 'engine',
27 'fueltype',
28 'vehicletype',
29 'bodystyle',
30 'name',
31 'vin',
32 'modelcode',
33 'stocknum'

test select from cars_stage
1 USE cars_stage;
2 SELECT 'data-make',
3 'data-model',
4 'data-year',
5 'data-tr',
6 IFNULL(nullif('data-extcolor',''), 'UNK') as 'data-extcolor',
7 IFNULL(nullif('data-intcolor',''), 'UNK') as 'data-intcolor',
8 IFNULL(nullif('data-trans',''), 'UNK') as 'data-trans',
9 'data-price',
10 'data-engine',
11 'data-fueltype',
12 'data-vehicletype',
13 'data-bodystyle',
14 'data-name',
15 'data-vin',
16 'data-modelcode',
17 'data-stocknum'
18 FROM 'data-american_cars'
19 ORDER BY 'data-make',
20 'data-model',
21 'data-year',
22 'data-tr',
23 'data-extcolor',
24 'data-intcolor',
25 'data-trans',
26 'data-price',
27 'data-engine',
28 'data-fueltype',
29 'data-vehicletype',
```

Data analyse – Pseudo Keys

Vejen til en RDBMS database

kan man ikke tage for givet. Derfor Order by alle kolonner.	
<p>Vi kører begge forspørgelser og gemmer resultatet i csv-tekstfiler.</p> <pre>test select from cars_rdbms_pseudo_keys_output.csv</pre> <pre>test select from cars_stage_output.csv</pre> <p>Nu kan vi sammenligne indholdet i filerne.</p> <p>Øverst ser vi at headeren er forskellig, det er nemt at forklare.</p>	
<p>Vi ser også andre forskelle, som her hvor trim i den ene database står som 'Lariat' men i den anden database står som 'LARIAT'.</p> <p>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.</p>	

Opmærksomhedspunkter

- Bemærk at kolumnen 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.