

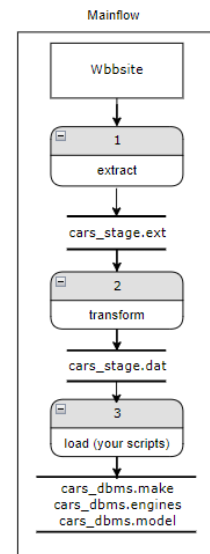
# American Cars Database Case

## Screenscraping data, etl og normalisering

I denne opgave vil vi screenscrape en kilde på nettet og bruge de data vi finder til at lave en serie databaser, der i trin vil behandle de oplysninger vi har fundet. De enkelte trin i opgaven kan springes over, hvis teknikken volder dig kvaler er, er der i filer og beskrivelser der kan bringe dig videre.

Læringsmålene er:

1. Programmering: PHP til screenscraping.
  - a. Screenscraping teknologier i PHP
  - b. No OOP
  - c. Funktionel programmering – no sideeffects.
2. Etablering af staging databaser
  - a. The BI way of doing it.
  - b. Datatype og type konvertering, Completeness.
  - c. Historik og batches.
  - d. Fysisk og Logisk databaselag, hvem arbejder med hvad?
3. Etablering ad RDMS
  - a. Normalformer og relationer
  - b. DDL,DQL,DML



Målet er at lande vores screenscraping data i en RDBMS

**Opgaven kan i sin helhed løses individuelt, i gruppe, eller som en øvelse i pair-programming.**

Opgaven er beskrevet i afsnittet: [Etablering ad RDBMS](#)

Der hører to git repositorier til dette oplæg

<https://github.com/kimo1ucl/Webscraping>:

<https://github.com/kimo1ucl/SQL-Datatypes-Normalization>:

## Contents

Programmering: PHP til screenscraping. ....	2
Etablering af staging databaser .....	2
<b>Opgave:</b> Etablering ad RDBMS.....	3
Instruks: PHP screenscraping .....	4
Instruks: Etablering af stage og template database .....	5
Stage database.....	6
Template database .....	7
Instruks: Anvendelse af template database .....	8
Development Integration: Seed og Edit.....	8
Development Integration: Generate database objects .....	11
Konklusion om replika et BI udviklingsflow .....	13
Afprøvning: Rådata ETL.....	14

# American Cars Database Case

## Screenscraping data, etl og normalisering

### Programmering: PHP til screenscraping.

Screenscraping er noget hvor PHP altid har skinnet! Specifikt til den disciplin har der længe eksisteret frameworks i PHP, som man i en Windows virksomhed måtte kigge langt efter. Vi ser på en interessant artikel der beskæftiger sig med at afprøve de forskellige teknologier.

<https://github.com/kimo1ucl/Webscraping>: Kode du kan downloade og prøve selv.

Men husk, at hvis der er forudsætninger som frameworks, så skal de også installeres for at download koden fungerer. Enkelte kræver installation af filer på dit operativ system, men mange danner alene filer i det projekt du arbejder med.

Når du installerer et framework, installerer du oftest kun filer i den mappe hvorfra du eksekverer. Der dannes en mappe: 'vendor'. I 'vendor' mappen ser du mapper og kildekode, som det blev hentet/opdateret sidste gang kommandoen 'composer require <framework>' blev kørt fra roden af webprojektet. Kræver disse framework installation af filer på operativsystemet, skal man være meget grundig med at registrere afhængigheden. Oplysninger af denne art er design dokumentation, og skal bruges hver gang en ny server i et givent miljø skal konfigureres.

### Etablering af staging databaser

Vi skal have importeret vores udtræk. I første omgang handler det om at få alle kolonner og alle rækker med. Vi importerer til en tabel vi vil kalde noget med 'ext', så ved vi at dette er vores 'Extraction' database. Alle kolonner er en overdimensioneret string type, fordi vi kompromisløst vil have data fra fil og ind i databasen. Derfra kan vi langt lettere lokalisere fejl og forekomster.

I det næste trin skal vi type konvertere, fordi i første omgang importerede vi bare det hele som strings.

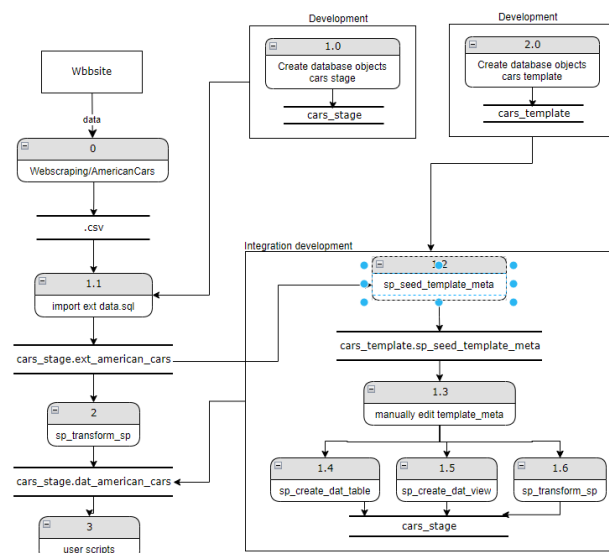
- Kolonne for kolonne skal vi bestemme
  - o passende datatype.
  - o Om der er behov for datatransform

Database objekterne vi arbejder med i det videre forløb er views og stored procedures, i en replika af arbejdsgangen i et BI udviklingsflow hvor vi arbejder med rådata. Fra en 'cars\_template' database genererer vi de nødvendige database objekter, der skal realisere en extract, transform, load proces så data lander i vores dat (for data) tabel.

I dfd diagrammet er det færdige load præsenteret til venstre, til højre er udviklingsindsats.

I repositoriet til denne del finder du komplet source kode, som du blot skal eksekvere mod din mysql server.

Læser du scriptsene ser du masser af SQL som du grundlæggende lærer fra w3school sitet mfl.



# American Cars Database Case

## Screenscraping data, etl og normalisering

### Opgave: Etablering af RDBMS

Vi skal have fundet sammenhænge i vores data. Klart at det handler om amerikanske biler i forskellige mærker, men givet at vi vil have data repræsenteret i en relationel database på 3. normal form, dvs. At vi forventer en tabel som f.eks. 'Make' hvor der i rækkerne står Ford, Chevrolet, Cadillac. En anden tabel der helt sikkert også springer i øjnene er Model. Men hvordan med Make, Model og motortype? Ford har vel en serie motorer, men hvordan så med denne motor og de forskellige modeller? Er farveskemaerne Model eller Make specifikke?

**Du skal lave script(s) der kopierer data fra cars\_stage.dat\_ til tabeller, du også selv designer og danner, i cars\_rdbms databasen. Det kan være en god ide at dele scriptsene op i en ordnet sekvens, så du trinvis opbygger din samlede query.**

Målet er at lande vores screenscraping data i en RDBMS med jeres forståelse af disse sammenhænge.

- Der er ikke en sand løsning til opgaven, det handler også om de muligheder du ser i anvendelsen af data og den måde de repræsenteret på. Hvilken kardinalitet ser du gælder?.
- I opgaven med at etablere rdbms skal du alene tage udgangspunkt i de data der er tilgængelige.
- Selvfølgelig må du ikke opdatere rådata. Men hvis du vælger en løsning med pseudo nøgler (autoid) må de(n) nødvendige kolonner selvfølgelig tilføjes.
- Opgaven sigter ikke mod en tilgang, hvor du er fremsynet, og tænker i overordnet modellering, sådan at vi kunne have mange webscrabes, der med dine SQL scripts mellem stage og rdbms database, kunne gemme data fra alle webscrabes. Dette er en rådata extract-transform-load. Når data skal herfra og til BI brugerne vil man måske lave datamarts, eller noget powerbi til at lave en tilsvarende extract-transform-load abstraktion.

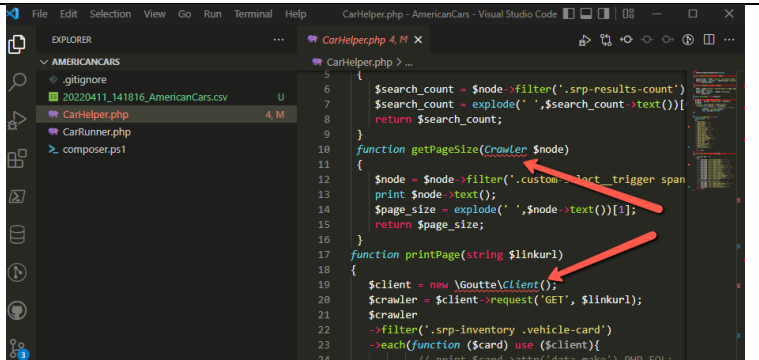
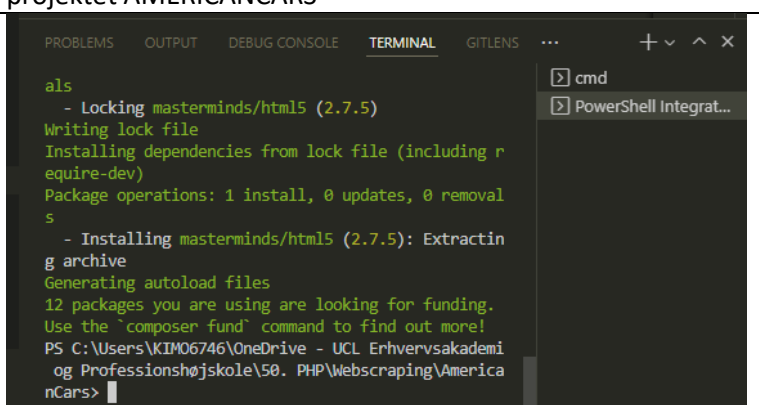
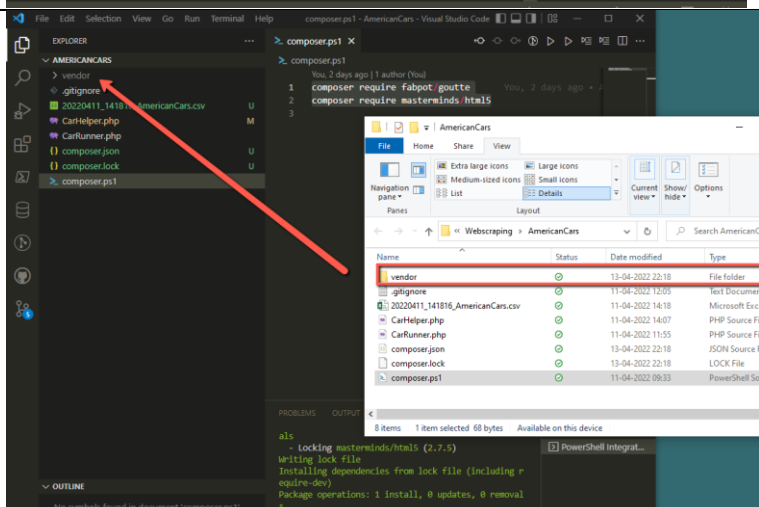
# American Cars Database Case

## Screenscraping data, etl og normalisering

### Instruks: PHP screenscraping

Denne instruks hjælper dig med at ekskvare php screenscraping af et website tilhørende end amerikansk brugtogsforhandler der udstiller brugte biler til salg.

Download kode fra: <https://github.com/kimo1ucl/Webscraping>

Åbn kode filen CarHelper.php og se at alle vores typer og instantieringer fejler. Vi har ikke det fornødne framework	
Hvis du ikke har composer installeret skal dette gøre først	<a href="https://getcomposer.org/download/">https://getcomposer.org/download/</a>
Du kan køre PS scriptet cmposer.ps1 – alt det gør er at ekskvare disse to kommandoer i roden af projektet AMERICANCARS	composer require fabpot/goutte composer require masterminds/html5  Alternativt skyd dem af i en dos command, igen i roden af projektet AMERICANCARS
Output ser du terminal vinduet, læs det igennem, der må ikke være errors, men warnings er som regel ok. Se også efter en konklussion på hver framework overførsel	
'composer require' kommandoen danner vendor mappen, og composer.json og .lock	

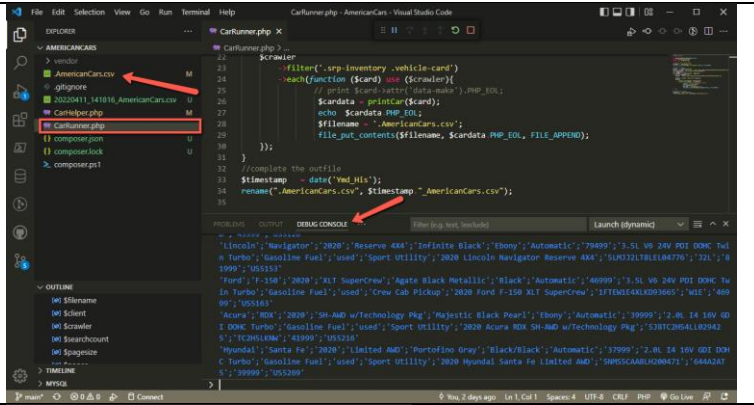
# American Cars Database Case

## Screenscraping data, etl og normalisering

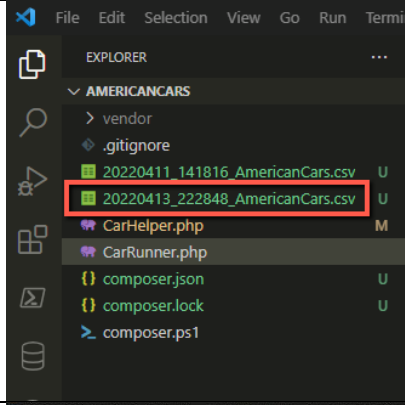
Kør 'CarRunner.php'

En midlertidig fil dannes ('.AmericanCars.csv')

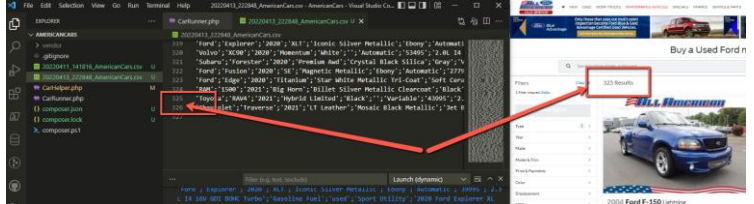
Outputtet fra screenscrabet vises i debug console



Når scriptet slutter omdøbes filen med et timestamp



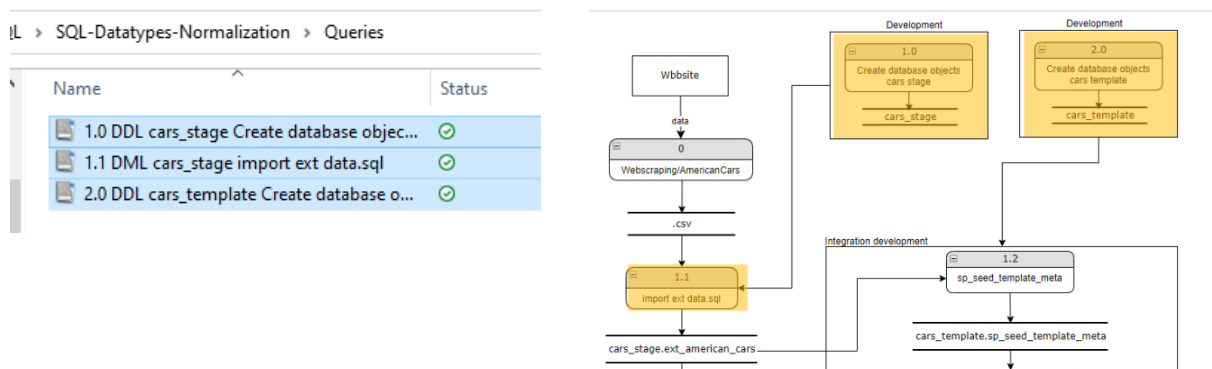
En kontrol kunne være at checke antal linjer i output fil. Der skal være en linje pr. annonceret bil + header.



## Instruks: Etablering af stage og template database

Nu skal vi arbejde med databaser. Alle scripts er lavet klar til eksekvering., alt hvad du skal gøre er at eksekvere dem. Find dem i mappen **Queries**

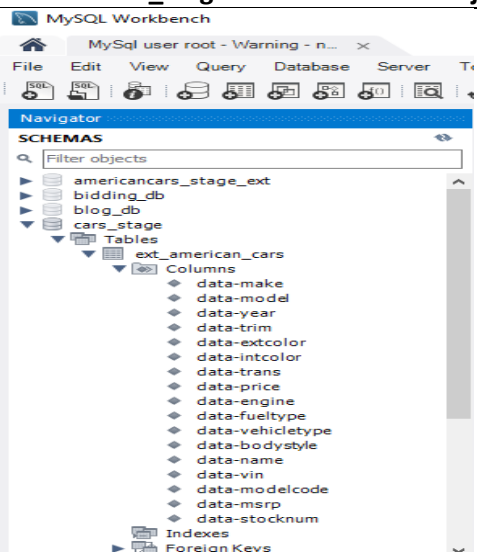
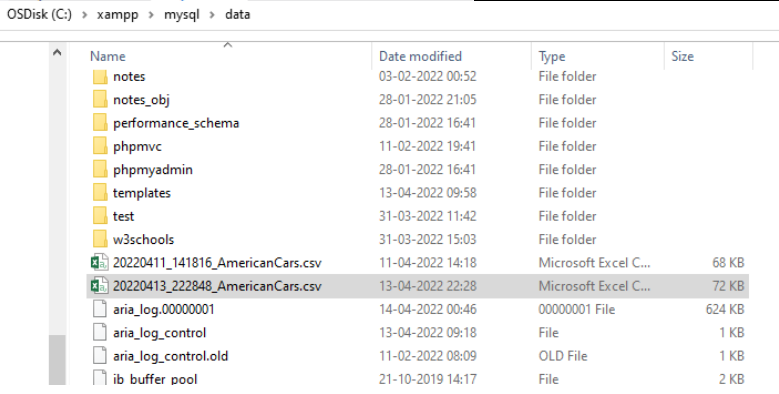
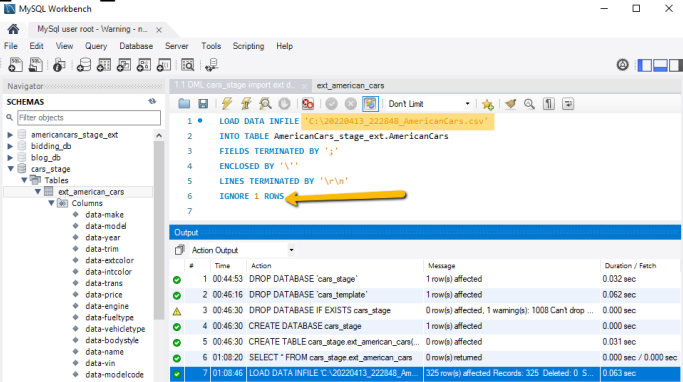
Kildekoden kan du hente her: <https://github.com/kimo1ucl/SQL-Datatypes-Normalization>:



# American Cars Database Case

## Screenscraping data, etl og normalisering

### Stage database

<p>Kør koden i filen</p> <p>Check at der er oprettet en database <b>cars_stage</b> og tabellen <b>ext_american_cars</b></p>	<p><b>1.0 DDL cars_stage Create database objects.sql</b></p> 
<p>Placer filen du vil importere i data mappen til din mysql installation. Bruger du xamp kan dette være et hint: <b>C:\xampp\mysql\data</b></p> <p>Dette er en "beskidt" måde at gøre det på, man kunne have brugt symbolske links feks.</p>	
<p>Ideelt set ville alle dataudtræk ligge i en velordnet filstruktur, organiseret på en eller anden måde så man intuitivt ville finde filer for hver integration. Nogle integrationer indeholder måske mere end en fil feks.</p> <p>Det man i denne øvelse skal vide er, at når der i sql filen står LOAD DATA INFILE 'C:\20220411_141816_AmericanCars.csv' Så indlæses fra: C:\xampp\mysql\data\20220411_141816_AmericanCars.csv'</p>	

# American Cars Database Case

## Screenscraping data, etl og normalisering

Lav en select på tabellen:  
**SELECT \* FROM**  
**cars\_stage.ext\_american\_cars**  
**;**

- Check rækkeantal
- Check 1 datalinje I fil er den samme som første data linje I tabel
- Check alle kolonner er mappet.

data-make	data-model	data-year	data-trim	data-extcolor	data-intcolor
Ford	F-150	2004	Lightning	Sonic Blue Metallic	Dark Graphite
Ford	Super Duty F-550 DRW	2008	XL Open Service Body	Oxford White	Medium stone
Ford	Escape	2013	Titanium 4WD	Ginger Ale Metallic	Charcoal Black
Toyota	Corolla	2014	S	Classic Silver Metallic	Black
Nissan	Rogue	2014	S AWD	Graphite Blue	Almond
Ford	Fusion	2014	SE	Sunset Metallic	Ebony
Jeep	Wrangler Unlimited	2015	Sport 4WD	Billet Silver Metallic Clearcoat	Black
Ford	F-150	2015	Platinum SuperCrew	Guard Metallic	Black
Jeep	Wrangler Unlimited	2015	Sport 4WD	Black Clearcoat	Black
Hyundai	Sonata	2015	2.4 SE	Phantom Black	Gray
Jeep	Wrangler	2016	Rubicon Hard Rock 4...	Black Clearcoat	Black
Honda	Civic Sedan	2016	EX-T	White Orchid Pearl	Gray
Toyota	Corolla	2016	S Plus	Slate Metallic	Black
Ford	Escape	2017	SE 4WD	Oxford White	Medium Light ...
Subaru	Outback	2017	Premium AWD	Crystal Black Silica	Slate Black

## Template database

Kør koden i filen

Check at der er oprettet en database: **cars\_template**  
og tabellen: **template\_meta**  
og stored procedures som:  
**sp\_create\_dat\_table**  
**sp\_seed\_template\_meta**

Optionelt (simpel sp til intro)  
**sp\_sample**

## 2.0 DDL cars\_template Create database objects.sql

**Navigator:**

**SCHEMAS**

Filter objects

- americancars\_stage\_ext
- bidding\_db
- blog\_db
- cars\_stage
- cars\_template**
  - Tables**
    - template\_meta**
      - Columns
        - tablename
        - columnname
        - datatype
        - dimension
        - dimension\_tolerant
      - Indexes
      - Foreign Keys
      - Triggers
    - Views
    - Stored Procedures**
      - sp\_create\_dat\_table
      - sp\_sample
      - sp\_seed\_template\_meta
    - Functions
  - ci4tutorial
  - iidb\_sample

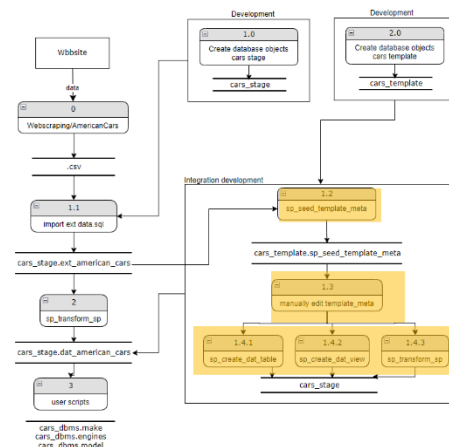


## Screenscraping data, etl og normalisering

## Instruks: Anvendelse af template database

I øvelsen her skal vi bestemme os for datatyper til vores screen scrabing data. Hvor lange skal tekststrengene være? Hvordan er numeriske værdier repræsenteret?

Hvis vi dimensionerer vores datatyper så de eksakt passer den længste streng der måtte være repræsenteret i en given kolonne, risikerer vi at vi i et af de følgende udtræk står med data der ikke passer til vores dimensionering fordi den var for snæver. Det er ofte en skønssag hvor meget ekstra vi vil tillade. I nogle tilfælde vil det fremgå at data kan have intet eller lidt varians, mens andre kolonner måske rummer data med meget stor varians i længde.



Vi vil ”opdatere” vores beslutninger i tabellen `template_meta`, derefter kan vi generere dataobjekter, der skal oprettes i stage databasen, og som skal anvendes hver gang data vi henter en ny datafil og data skal flyttes igennem ETL laget.


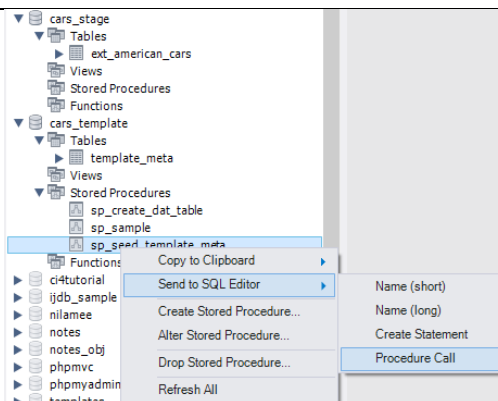
## Development Integration: Seed og Edit

## 1.2 sp\_seed\_template\_meta

I template databasen kør kommandoen **sp\_seed\_template\_meta**:

```
CALL
`cars_template`.`sp_seed_template_meta`('cars_stage'
, 'ext american cars');
```

Til højre er vist hvordan du med MySQL Workbench kan højreklikke den stored procedure kan få genereret kommandoen.



The screenshot shows the SQL Server Enterprise Manager interface. The top window, titled 'SQL File 5', contains the following SQL query:

```
1 CALL `cars_template`.`sp_seed_template_meta`(<{IN databasename nvarchar(64)}>, <{IN tablename nvarchar(64)}>);
2
```

The bottom window, also titled 'SQL File 5', contains the following SQL query:

```
1 CALL `cars_template`.`sp_seed_template_meta`('cars_stage', 'ext_american_cars');
2
```



# American Cars Database Case

## Screenscraping data, etl og normalisering

Nu skal du kunne se rækker i tabellen **template\_meta**

I nogle kolonner er der mere varians end i andre.

Nogle kolonner ses indhold som indikerer at værdierne kommer fra en værdiliste. Lav f.eks. en select distinct på kolonnen data-fueltype:

*SELECT distinct `data-fueltype` from cars\_stage.ext\_american\_cars;*

	data-fueltype
►	Gasoline Fuel
	Diesel Fuel
	Hybrid Fuel
	Flex Fuel

tablename	columnname	ordinal_position	datatype	minlength	maxlength	dimension
ext_american_cars	data-make	1	nvarchar	3	10	10
ext_american_cars	data-model	2	nvarchar	2	21	21
ext_american_cars	data-year	3	nvarchar	4	4	4
ext_american_cars	data-trim	4	nvarchar	1	27	27
ext_american_cars	data-extcolor	5	nvarchar	0	41	41
ext_american_cars	data-intcolor	6	nvarchar	0	26	26
ext_american_cars	data-trans	7	nvarchar	0	9	9
ext_american_cars	data-price	8	nvarchar	1	5	5
ext_american_cars	data-engine	9	nvarchar	4	40	40
ext_american_cars	data-fueltype	10	nvarchar	9	13	13
ext_american_cars	data-vehicletype	11	nvarchar	4	4	4
ext_american_cars	data-bodystyle	12	nvarchar	7	23	23
ext_american_cars	data-name	13	nvarchar	11	51	51
ext_american_cars	data-vin	14	nvarchar	17	17	17

Alternativt kan du bruge csv filen og **Excel** filtre til at identificere disticnte forekomster så længe der er et overskueligt antal rækker i filen.

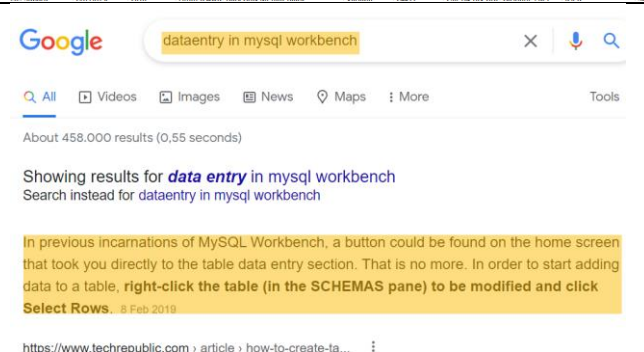
Hvis der er flere rækker end excel kan håndtere, kan **PowerBI** bruges

Men er der mange millioner rækker er **SQL** den eneste option.

### 1.3 manually edit template\_meta

Vi skal til at editere rækkerne i tabellen **template\_meta**.

At åbne en datatabel og editere felterne direkte kaldes data entry. Umiddelbart kunne jeg ikke finde featuren i MySql Workbench



# American Cars Database Case

## Screenscraping data, etl og normalisering

Brug denne select når du skal editere indholdet i tabellen template\_meta

```
SELECT * FROM cars_template.template_meta  
order by ordinal_position;
```

Data i ext tabellen løbes igennem, og vi indhenter oplysninger som min og max length, om der er tomme data i nogle koldonne/rækker.

```
1 • SELECT * FROM cars_template.template_meta  
2   order by ordinal_position;
```

tablename	columnname	ordinal_position	datatype	min_length	max_length	count_blanks	dimension
ext_american_cars	data-make	1	varchar	3	10	0	10
ext_american_cars	data-model	2	varchar	2	21	0	21
ext_american_cars	data-year	3	varchar	4	9	0	9
ext_american_cars	data-trim	4	varchar	1	31	0	31
ext_american_cars	data-extcolor	5	varchar	0	41	11	41
ext_american_cars	data-intcolor	6	varchar	0	30	64	30
ext_american_cars	data-trans	7	varchar	0	10	5	10
ext_american_cars	data-price	8	varchar	1	10	0	10
ext_american_cars	data-engine	9	varchar	2	40	0	40
ext_american_cars	data-fueltype	10	varchar	9	13	0	13
ext_american_cars	data-vehidetype	11	varchar	4	16	0	16
ext_american_cars	data-bodystyle	12	varchar	7	23	0	23
ext_american_cars	data-name	13	varchar	9	51	0	51
ext_american_cars	data-vin	14	varchar	8	17	0	17
ext_american_cars	data-modelcode	15	varchar	0	14	4	14
ext_american_cars	data-mrnp	16	varchar	1	9	0	9
ext_american_cars	data-stocknum	17	varchar	6	13	0	13

Vi skal nu tilrette designet af vores dat tabel, ved at editere data i tabellen, direkte i Workbench Result grid

max_length	count_blanks	dimension
10	0	15
21	0	30
4	0	NULL
27	0	35
41	6	50
26	39	50
9	1	10
5	0	NULL
40	0	50
13	0	13
4	0	4

Kolonnen **data-vin** er temmelig sikkert en nøgle, altid 17 karakterer, datatypen ændres fra varchar til char (fast længde).

**data-year** og **data-price** er helt sikkert numeriske værdier, i dette tilfælde kan vi bruge en int som datatype. En int har ingen dimensionering, så felt i kolonnen dimension skal sættes til NULL.

**dimension** kolonnen ændres til en værdi der kan ligge over max length, som vi ser data repræsenteret i (i denne omgang), for at gøre interfacet mere robust og adresserende den usikkerhed der ligger i at basere design på et smalt udsnit af test data.

Her er mine bud på en forhåbentlig robust dimensionering.  
Tryk på Apply når du er færdig.

tablename	columnname	ordinal_position	datatype	min_length	max_length	count_blanks	dimension
ext_american_cars	data-make	1	varchar	3	10	0	15
ext_american_cars	data-model	2	varchar	2	21	0	30
ext_american_cars	data-year	3	INT	4	4	0	NULL
ext_american_cars	data-trim	4	varchar	1	27	0	35
ext_american_cars	data-extcolor	5	varchar	0	41	6	50
ext_american_cars	data-intcolor	6	varchar	0	26	39	50
ext_american_cars	data-trans	7	varchar	0	9	1	10
ext_american_cars	data-price	8	INT	1	5	0	100
ext_american_cars	data-engine	9	varchar	4	40	0	50
ext_american_cars	data-fueltype	10	varchar	9	13	0	13
ext_american_cars	data-vehidetype	11	varchar	4	4	0	4
ext_american_cars	data-bodystyle	12	varchar	7	23	0	30
ext_american_cars	data-name	13	varchar	11	51	0	100
ext_american_cars	data-vin	14	char	17	17	0	17
ext_american_cars	data-modelcode	15	varchar	0	11	4	11
ext_american_cars	data-mrnp	16	varchar	1	5	0	5
ext_american_cars	data-stocknum	17	varchar	6	6	0	6

I en dialog kan du se dine updates.  
Tryk Apply

Updates i filen  
2.1 DML UPdate template\_meta.sql

```
1. alter table template_meta set dimension = 15 where (tablename = 'ext_american_cars') and (columnname = 'data-make');  
2. alter table template_meta set dimension = 30 where (tablename = 'ext_american_cars') and (columnname = 'data-model');  
3. alter table template_meta set datatype = int, dimension = 4 where (tablename = 'ext_american_cars') and (columnname = 'data-year');  
4. alter table template_meta set dimension = 35 where (tablename = 'ext_american_cars') and (columnname = 'data-trim');  
5. alter table template_meta set dimension = 50 where (tablename = 'ext_american_cars') and (columnname = 'data-extcolor');  
6. alter table template_meta set dimension = 50 where (tablename = 'ext_american_cars') and (columnname = 'data-intcolor');  
7. alter table template_meta set dimension = 10 where (tablename = 'ext_american_cars') and (columnname = 'data-trans');  
8. alter table template_meta set datatype = int, dimension = 10 where (tablename = 'ext_american_cars') and (columnname = 'data-price');  
9. alter table template_meta set dimension = 50 where (tablename = 'ext_american_cars') and (columnname = 'data-engine');  
10. alter table template_meta set dimension = 13 where (tablename = 'ext_american_cars') and (columnname = 'data-fueltype');  
11. alter table template_meta set dimension = 4 where (tablename = 'ext_american_cars') and (columnname = 'data-vehidetype');
```

# American Cars Database Case

## Screenscraping data, etl og normalisering

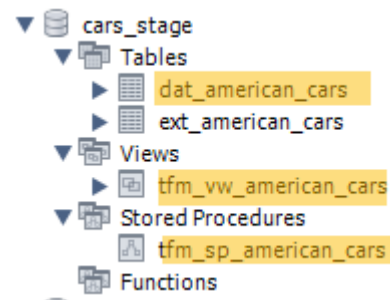
### Development Integration: Generate database objects

Her skal vi generere database objekter i cars\_stage databasen.

Vi ender op med en stored procedure, der kan kaldes med automatik eksekvering. Oracle og MSSQL server har selvstændige schedulere dedikeret denne use case.

Med MySQL server vil man kunne lave en kommandolinje kommando afviklet i MySQL CLI (command line interface) og lade operativ systemets scheduler klare opgaven.


Står MySQL installationen på en Unix/Linux platform, ville crontab være et oplagt implementeringsrum for scheduleringen.



<p><b>1.4.1</b> <b>sp_create_dat_table</b> Kommandoen danner dat tabellen</p>	<pre>USE cars_template; DROP TABLE IF EXISTS `cars_stage`.`dat_american_cars`; CALL `cars_template`.`sp_create_dat_table`('cars_stage', 'american_cars');</pre>
<p>check at tabellen er designet som du forventer</p>	
<p><b>1.4.2</b> <b>sp_create_tfm_view</b> Kommandoen danner et view der typecaster data fra ext tabellen</p>	<pre>USE cars_template; DROP VIEW IF EXISTS `cars_stage`.`tfm_vw_american_cars`; CALL `cars_template`.`sp_create_tfm_view`('cars_stage', 'american_cars');</pre>
<p>check at viewet er designet som du forventer</p> <p>Afprøv viewet med SELECT * FROM `cars_stage`.`tfm_vw_american_cars`</p>	<p>I dette tilfælde vises create staementen i en lang tekststren på 1. linje</p> <p>Der findes en beutifier feature blandt kommandoerne i SQL editor vinduet</p>

# American Cars Database Case

## Screenscraping data, etl og normalisering

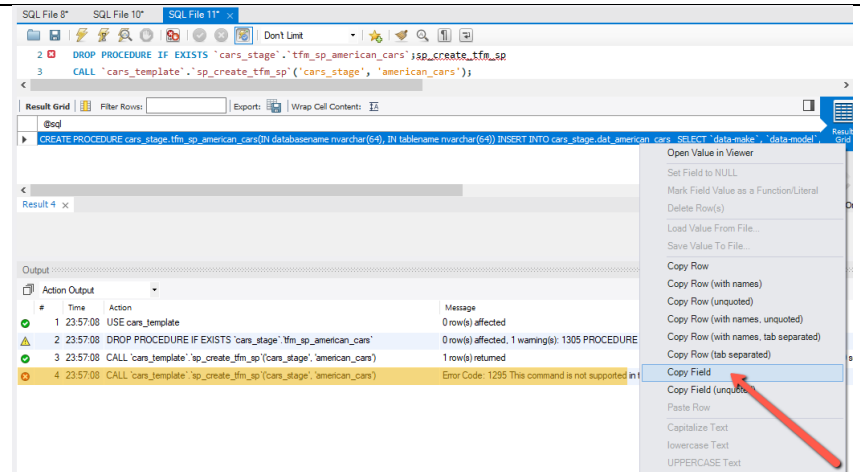
	 <pre>1 CREATE 2     ALGORITHM = UNDEFINED 3     DEFINER = `root`@`localhost` 4     SQL SECURITY DEFINER 5     VIEW `cars_stage`.`tfm_vw_american_cars` AS 6     SELECT 7         CAST(`cars_stage`.`ext_american_cars`.`data-make` 8             AS CHAR (15) CHARSET UTF8) AS `data-make`, 9         CAST(`cars_stage`.`ext_american_cars`.`data-model` 10            AS CHAR (30) CHARSET UTF8) AS `data-model`, 11         CAST(`cars_stage`.`ext_american_cars`.`data-year` 12            AS SIGNED) AS `data-year`, 13         CAST(`cars_stage`.`ext_american_cars`.`data-trim` 14            AS CHAR (35) CHARSET UTF8) AS `data-trim`, 15         CAST(`cars_stage`.`ext_american_cars`.`data-extcolor` 16            AS CHAR (50) CHARSET UTF8) AS `data-extcolor`, 17         CAST(`cars_stage`.`ext_american_cars`.`data-intcolor` 18            AS CHAR (50) CHARSET UTF8) AS `data-intcolor`, 19         CAST(`cars_stage`.`ext_american_cars`.`data-trans` 20            AS CHAR (10) CHARSET UTF8) AS `data-trans`, 21         CAST(`cars_stage`.`ext_american_cars`.`data-price` 22            AS SIGNED) AS `data-price`, 23         CAST(`cars_stage`.`ext_american_cars`.`data-engine` 24            AS CHAR (50) CHARSET UTF8) AS `data-engine`, 25         CAST(`cars_stage`.`ext_american_cars`.`data-fueltype` 26            AS CHAR (13) CHARSET UTF8) AS `data-fueltype`, 27         CAST(`cars_stage`.`ext_american_cars`.`data-vehicletype` 28            AS CHAR (4) CHARSET UTF8) AS `data-vehicletype`, 29         CAST(`cars_stage`.`ext_american_cars`.`data-bodystyle` 30            AS CHAR (30) CHARSET UTF8) AS `data-bodystyle`,</pre>
<b>1.4.3 sp_create_tfm_sp</b> Kommandoen danner en sp, der trækker data fra ext tabellen gennem tfm viewet og lander dem i dat tabellen	USE cars_template; DROP PROCEDURE IF EXISTS `cars_stage`.`tfm_sp_american_cars`; CALL `cars_template`.`sp_create_tfm_sp`('cars_stage','american_cars');
<p>I dette tilfælde får vi denne fejlkode: Error Code: 1295 This command is not supported in the prepared statement protocol yet</p> <p>Jeg lærte at man ikke kan danne sp'ere gennem sp'ere i MySQL.</p>	 <pre>31 END LOOP testLoop; 32 33 #cutoff surplus comma and carriage return 34 SET @sql = LEFT(@sql,length(@sql)-2); 35 36 SET @sql = CONCAT(@sql,'\n',' FROM ', databaseName, '.', 'tfm_vw_', tableName); 37 38 SELECT @sql; 39 40 # This cant be done 41 #Error Code: 1295 This command is not supported in the prepared statement protocol yet 42 #https://www.mysqlfaq.com/mysql-faq/Errors/ERROR-1295-(HY000)-This-command-is-not-supported-in-the-s 43 PREPARE cmd FROM @sql ; 44 EXECUTE cmd; 45 DEALLOCATE PREPARE cmd ; 46 47 IF cur;</pre>
<p>Dette er en major showstopper ift. at bruge sp'ere som halv automatiserings grundlag. Vi kunne måske komme omkring det på mange måder, vi taler trods alt om et operatør drevent "API". Vi fortsætter og anvender en workaroud, der også viser dig hvordan du debugger stored procedures.</p> <p>Bemærk den gule pil i skærbilledet ovenfor. I dette tilfælde selectere vi hvad der står i vores dynamiske sql tekststreng.</p>	

# American Cars Database Case

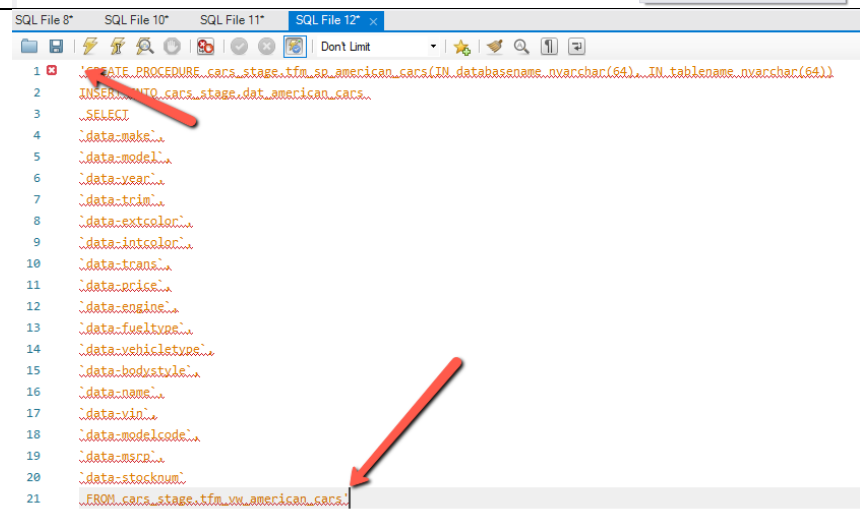
## Screenscraping data, etl og normalisering

Peg på resultatet i 'Result Grid' højreklik og i kontekst menuen vælg 'Copy Field'

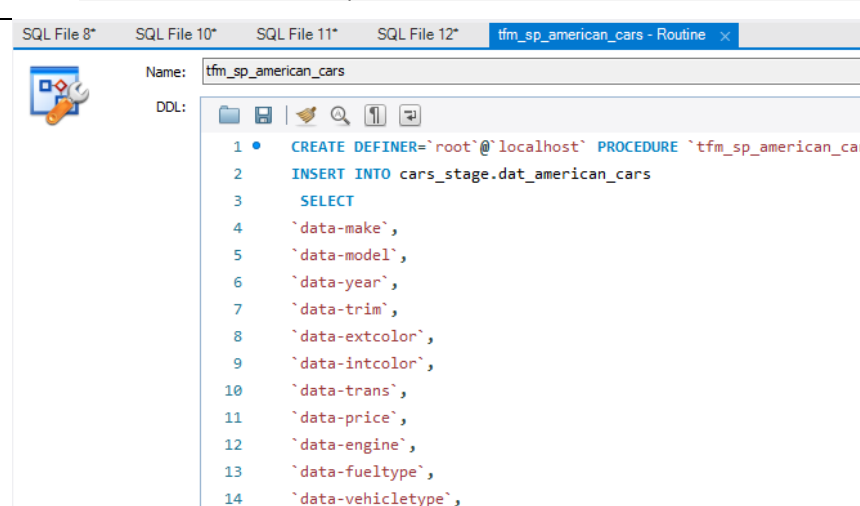
Paste indholdet til et nyt SQL Command vindue



Fjern ping fra start og slut, og du har kommandoen klar, du nu selv må eksekvere.



Check cars\_stage.tfm\_sp\_american\_cars definition



## Konklusion om replika et BI udviklingsflow

Der mangler mange ting endnu før vi kan sige at vi har det fulde featuresæt som er nødvendigt i et produktionsklar BI rådata lag.

- I alle sp'er der afvikles automatiseret, skal der laves try- catch og logges i en fejldatabase, måske er dette et domæne helt for sig selv, og kalder på oprettelse af en egentlig system database

# American Cars Database Case

## Screenscraping data, etl og normalisering

- Vi har slet ikke favnet batch begrebet, som er nødvendigt når vi regelmæssigt importere data. Formålet er at danne historiske data, og det når vi ikke i denne replika, der kun håndtere en enkelt import.
- Historiske data dannes typisk i en merge mellem dat tabellen og vores eksisterende billede af hvad vi fandt sidste gang vi aflæste on-line.

### Afprøvning: Rådata ETL

Vi nåede endeligt til at skulle afprøve vores transform load.

**2. Transform**

Kommandoen kører en extract, transform, load som den ville blive eksekveret automatisk

The screenshot shows a SQL editor with the command `CALL `cars_stage`.`tfm_sp_american_cars`();` entered. Below the editor, the 'Output' pane displays 'Action Output' with a table showing the execution details: 1 row, 00:35:38, and the action 'CALL `cars\_stage`.`tfm\_sp\_american\_cars`()'. A message at the bottom right states '325 row(s) affected'.

Check data i tabellen

Cars\_stage.dat\_american\_cars

The screenshot shows a SQL editor with the command `SELECT * FROM cars_stage.dat_american_cars;` entered. Below the editor, the 'Result Grid' displays a table with 9 columns: data-make, data-model, data-year, data-trim, data-extcolor, data-intcolor, data-trans, data-price, and data-engine. The table contains 15 rows of data, including models like Ford F-150, Ford Super Duty, Ford Escape, Toyota Corolla, Nissan Rogue, Ford Fusion, Jeep Wrangler, and others.

Og dermed blev vi færdige med denne del af vores mainflow

```
graph TD
    Wbbsite --> 1[extract]
    1 --> cars_stage_ext[cars_stage.ext]
    cars_stage_ext --> 2[transform]
    2 --> cars_stage_dat[cars_stage.dat]
    cars_stage_dat --> 3[load your scripts]
    3 --> cars_dbms[cars_dbms.make  
cars_dbms.engines  
cars_dbms.model]
```

The diagram illustrates a data pipeline. It starts with a box labeled 'Wbbsite'. An arrow points down to a box labeled '1' with 'extract' below it. Another arrow points down to a box labeled 'cars\_stage.ext'. A third arrow points down to a box labeled '2' with 'transform' below it. A fourth arrow points down to a box labeled 'cars\_stage.dat'. A fifth arrow points down to a box labeled '3' with 'load (your scripts)' below it. Finally, an arrow points down to a box labeled 'cars\_dbms.make', 'cars\_dbms.engines', and 'cars\_dbms.model'.