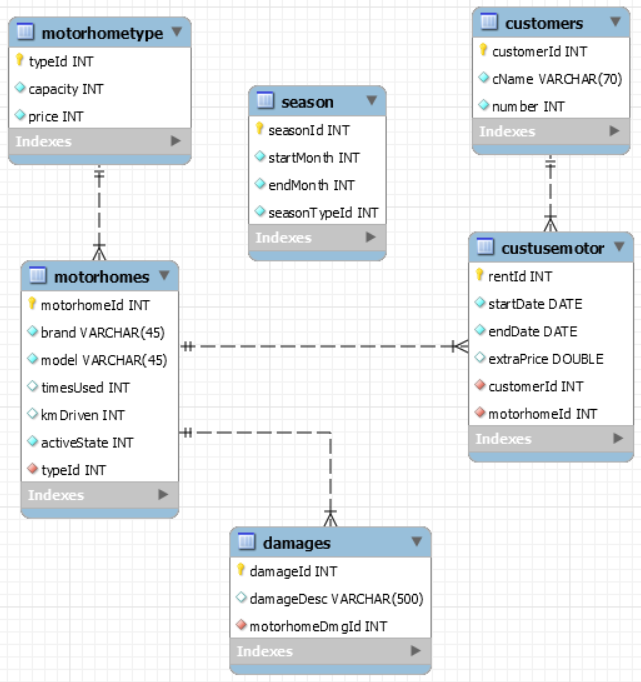
MySQL:

Beskrivelse af ER diagram og tabeller:



Motorhomes:

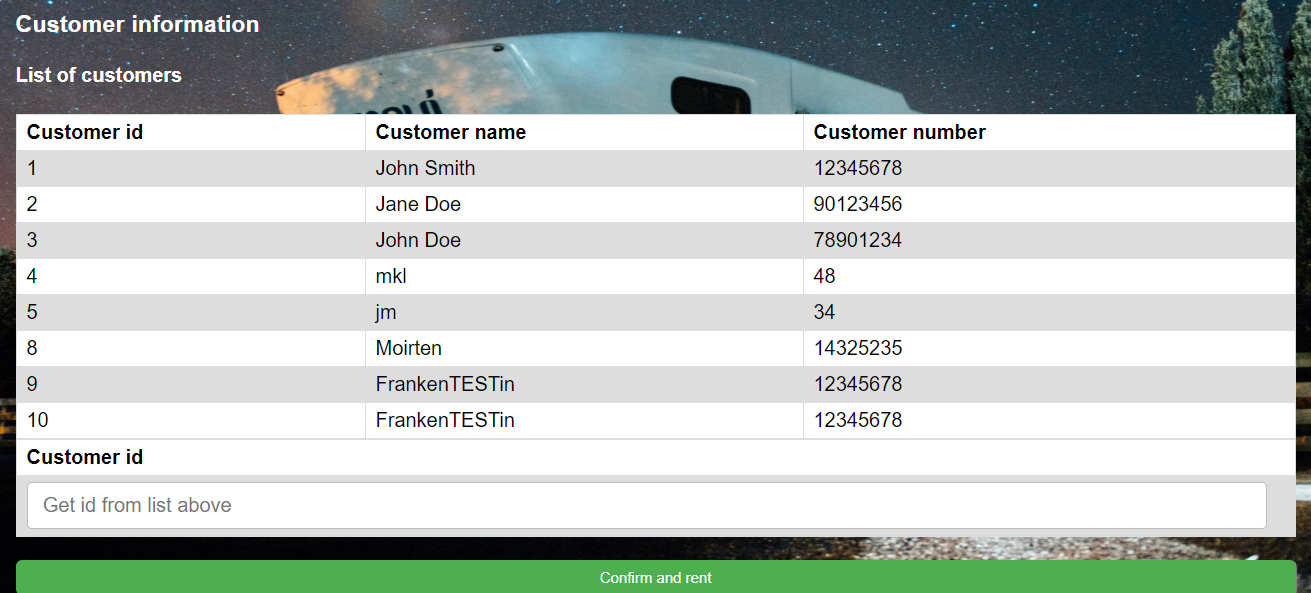
Vores tabel “motorhomes”, er en “liste” over alle de 32 motorhomes, som vores fiktive firma ejer. Der er mulighed for at oprette flere i vores applikation, i tilfælde af, at flere motorhomes skulle købes i fremtiden. Tabellen består af et motorhomeId, som er den primære key og bliver brugt igennem hele applikationen til at identificere det enkelte motorhome. Brand, model, timesUsed og km Driven beskriver alle egenskaber omkring det enkelte motorkøretøj, navnlig hvilket mærke og model køretøjet er, hvor mange gange det er blevet brugt, og hvor langt det har kørt. ActiveState bliver brugt til at se om det enkelte køretøj er leget ud, eller ej. TypeId viser hvilken type motorhomet er, sengepladser, dagspris, og typeId er en foreign key til “motorhometype” tabellen som bliver beskrevet senere.   
Der findes en model klasse i applikationen som hedder Motorhome, og data fra tabellen kan direkte oversættes til tabellen.

Custusemotor:

Custusemotor er vores tabel, som beskriver lejeaftaler, navnet er en sammentrækning af customers use of motorhomes. Den har dens primary key rentId som også bliver brugt af java, til at identificere de forskellige lejeaftaler. Tabellen indeholder start datoen og slut datoen for lejeperioden, startDate og endDate, ekstraPrice som er den pris som kunden vil betale, hvis han ikke får flere gebyrer, fx kørt mere end de tilladte 400 km gennemsnit om dagen. EkstraPrice har allerede fået indregnet en eventuel stigning af pris, hvis kunden lejer i en højsæson. CustomerId og motorhomeId, referer til det motorhome som er blevet udlejet, og den kunde som lejer motorhomet.

Customers:

Customers er vores database over kunder, som er registreret i vores system. Man skal være registreret før man får lov at leje, da man vælger fra en liste over allerede registrerede kunder når man lejer, som ses på billedet under:



Customers primary key er customerId, som også bliver brugt til at identificere kunder i java koden.  
Hver kunde har meget lidt information stående om dem, nemlig et navn og et nummer. Det er gjort, da det ikke har nogen betydning for de usecases der er udviklet. Det er set som spild af energi at gøre klassen større og inkludere information, som ville være en del af et virkeligt projekt, fx cpr-nummer, e-mail, mm.   
Hvis man skulle fuldt udvikle dette projekt, ligger det frit for, at man tilføjer flere søjler i Customer tabellen, og tilføjer felterne i de relevante html filer. Det vil ikke ændre eller forstyrre nogen logik i applikationen (low coupling).

Motorhometype:

Motorhometype tabellen beskriver de forskellige typer af motorhomes. Det er en tabel for sig selv, så man kun skal ændre informationen om typen 1 sted. Tabellen beskriver hvad prisen er, for hver enkelt motorhome type, og hvor mange sengepladser der findes. På samme måde som customer tabellen, er type tabellen også ret tom. I den virkelige verden er det meget muligt, at der vil være mere relevant information, fx om der er toilet, køkken, etc. men der er igen ingen grund til at bruge tid på at finde på en masse information.

Damages:

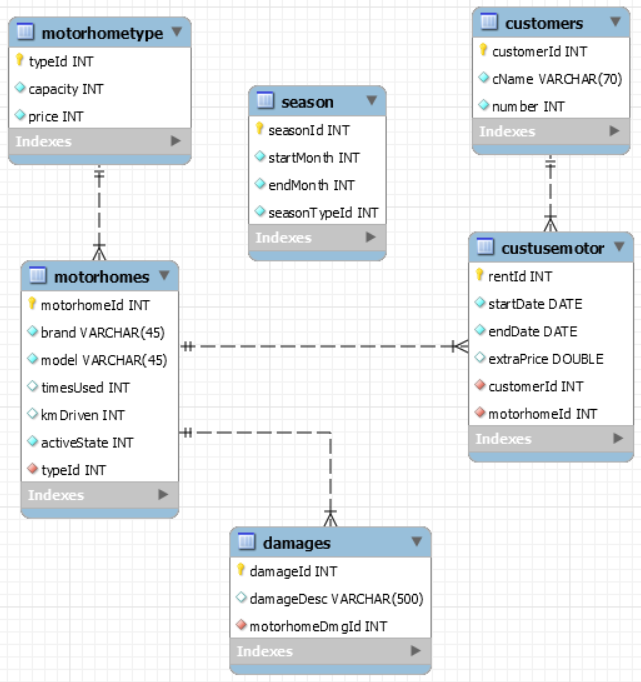
Damages tabellen indeholder beskrivelser om hvilke skader der er på et køretøj (damageDesc), hvilket køretøj der har skaden (motorhomeId), og tabellens primary key som bliver brugt til identifikation af hver enkelt skade. Grunden til at informationen ikke ligger i motorhomes tabellen, er at normalform 1 siger: Man ikke må have lister i en database. Hvis man har 2 skader på samme motorhome, vil det være en liste over skader.

Season:

Season indeholder informatiom om høj, midel og lav sæson, som har betydning for prisudregningen. Hver primære id har tilknyttet en startdato, slut dato og hvilken type sæson der snakkes om (høj, midel og normal). Java tjekker ved prisudregning den her tabel, og season tabellen har derfor ingen relation til de andre tabeller i databasen. Det er gjort, da det er java, der står for prisudregningen og ikke MySQL.

ER diagram:

Vi har i vores arbejde valgt at oprette ER diagrammet som noget af det første i projektets forløb. Det er gjort, fordi det giver mulighed for at se, hvordan vores data kommer til at se ud med det samme. Det gør det lettere i resten af udviklingsprocessen, når man kan se, hvordan de forskellige dele af projektet skal udarbejdes, som fx domænemodellen, html sider osv.   
Den endelige version af diagrammet kan ses nedenfor. Diagrammet har ændret sig minimalt siden første version, men der har været småting. Blandt andet ændrede vi prisen fra at ligge i motorhomes, til at ligge i motorhometype. Det betyder at prisen er baseret på typen, og ikke på det enkelte motorhome. Det gør det lettere at ændre prisen for flere hjem, men giver den ulempe at det er svære at differentiere mellem de forskellige motorhomes. Diagrammet er udviklet med normalformer i tankerne.



Normalformer:

Normalform 1:

Vores database overholder alle normalformerne. Første normalform siger kort sagt, at ingen kolonner gentager en andens værdier, og at databasen skal være atomisk (når man googler atomisk, kommer der kun fysik frem, hvilket jeg fandt lidt morsomt).  
Det kan bedst ses, at vi ikke har nogen gentagne kolonner ved tabellen damages. Damages indeholder information om motorhomes, men fordi at motorhome kan have flere skader vil det være nødvendigt med flere kolonner der indeholder skader, eller lave en liste i en celle. Begge ting er naturligvis imod normalfom 1, og derfor har vi lavet en tabel som indeholder skadebeskrivelsen, og et foreign key som linker til det motorhome der har skaden.

Normalform 2:

Normalform 2 gælder for tabeller med sammensatte primary keys. Alt information i en tabel med sammensatte primary keys skal direkte beskrive og være afhængige af primary key.

Normalform 2 bliver ikke brugt i databasen, ikke fordi den er uvæsentlig, men fordi vi ikke har nogen tabeller med sammensatte primary keys. Det har været meget let at undgå, fordi alle tabellerne har en simpel integer id key. Det eneste sted, hvor det kunne have været relevant er motorhomes. Det kunne have været muligt at lave model og brand til unikke keys, men det vil kun være muligt, hvis man har et af hvert køretøj. Det er meget tænkeligt at et firma køber 2 eller flere af same mærke og model, derfor har motorhomes deres egen individuelle key.

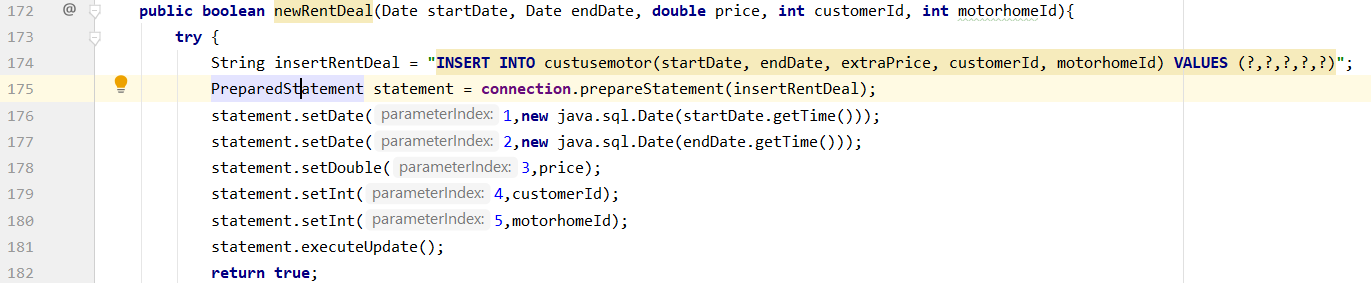
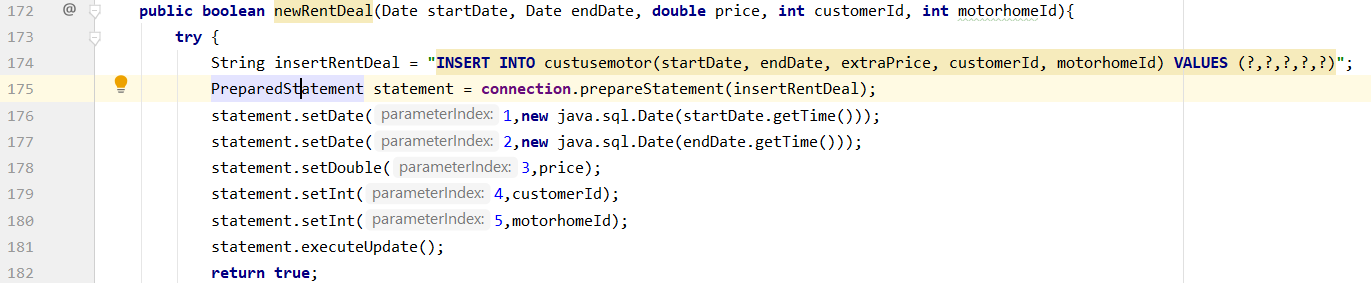
Normalform 3:

Normalform 3 siger, at alle kolonner kan bestemmes ud fra primary key, og at der ikke må være transitive funktionelle afhængigheder.

Det kan bedst ses i vores database ved motorhometype tabellen. Den beskriver forskellen på de forskellige typer motorhomes. Den kunne i teorien ligge under motorhomes tabellen, men det ville ikke overholde 3 normalform, da motorhomes tabellen ville indeholde data der ikke afhænger af motorhomeId, men derimoed af typeId. Derfor er den blevet lagt ud i sin egen tabel, og motorhomes tabellen har så fået et foreign key, der referer til type tabellen.

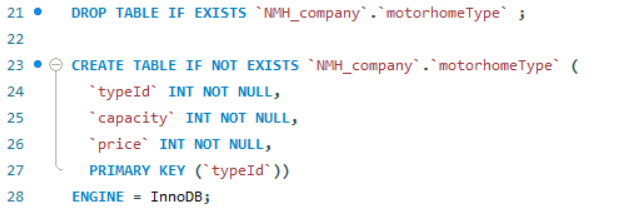
SQL dokumentation:

Prepared statement:

Gennem hele vores projekt gør vi brug af PreparedStatement, som er en del af java.sql. Det gøres for at undgå SQL injektion angreb. SQL injektion fungerer ved, at man som bruger indsætter en SQL statement, ofte med et ødelæggende bagtanke, i et felt som fx brugernavn, og håber at logikken der ligger inde bagved kører ens statement. Fx kunne en hacker skrive som brugernavn, “SELECT\* FROM user WHERE 1=1” i håb om at modtage en liste, der indeholder passwords eller e-mails.   
Java koden til at lave en PreparedStatement, ser ud som billedet ovenfor. Man skriver sin SQL statement, og alt data der skal indsættes i statement bliver lavet annoteret med ”?“. Den data man indsætter i sin string, bliver derefter tjekket og ens preparedStatement kører ikke hvis den fx har ”select \* from user” i sig.

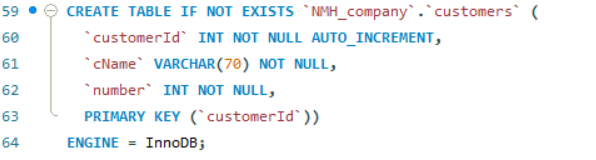
Create og drop (DDL):

I vores projekt har vi gjort brug af DDL og DML. DDL står for Data definition Language, og beskæftiger sig med at oprette databaser og ændre på allerede oprettede databasers struktur. DDL bliver derfor brugt i vores create script, til at oprette selve databasen og de tabeller der skal være i databasen. De DDL funktioner vi har brugt er create og drop. Begge funktioner bliver kun brugt i forbindelse med vores create script. Create bliver brugt til at oprette schema (databasen)[[1]](#footnote-26543) og tabellerne. Eksempelvis ved oprettelse af tabellen motorhometype:



Her oprettes en ny tabel i databasen “NMH\_company”, som er navnet på vores database.   
Drop table if exists bliver brugt, i tilfælde af, at man har kørt en ældre version af vores script eller scriptet fejler halvvejs igennem og man er nødt til at køre scriptet igen. Det har naturligvis den ulempe, at man risikere at droppe en masse vigtig data.

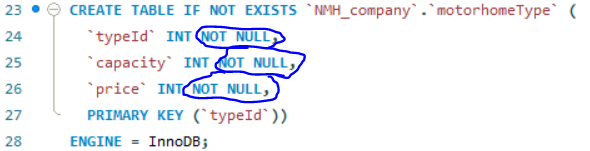
Motorhometype tabellen er en af de få tabeller, hvor id ikke har brugt AUTO\_INCREMENT som en del af oprettelsen af primary id. Det er gjort fordi typeId bliver brugt til at beskrive en bestemt type, og det er ikke ligegyldigt hvad nummer de har. Da det helst skal give nogenlunde mening, og fx ikke springe tal over i tilfældigheder. Det er mest så mennesker lettere får et overskud, det er lettere at forstå type 1,2,3..8 end 1,9,19,2...8. Et eksempel hvor AUTO\_INCREMENT bliver brugt:

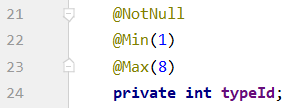


Her har customerId ikke større betydning, om man har fat i kunde 1 eller 199 har ingen betydning. Det er fuldstændig ligegyldig information, både for mennesker og for maskinen. Forhåbentlig for vores firma bliver customerId på et tidspunkt rigtig højt, og det giver endnu mindre mening med manuel styring af id.

Databasens integritet:

Den engine der bliver brugt er InnoDB, det er standard i MySQL i dag. Vi har i løbet af undervisningen ikke haft noget information omkring MySQL engines. Tidligere var det ikke innoDB, som blev brugt som standard, men derimod myisam. Den gamle engine havde den ulempe at den ikke tjekkede, om det var “valid data” man indsatte i tabellerne. Derfor er det min forståelse, at man kunne indsætte “invalid data”, fx en datO som fx 30 februar ville blive accepteret af myisam. Vi bruger innoDB derfor bliver invalid data ikke godtaget i databasen, fx ville d. 30 februar blive afvist og MySQL ville give beskeden Error Code: 1292. InnoDB har også andre fordele, som vi ikke gør brug af. Med innoDB har man fx mulighed for rollback i tilfælde af fejlagtige opdateringer.   
Vi gør brug af NOT NULL funktionen når vi opretter vores tabeller, det betyder at vi ikke kan indsætte tom information. Det bliver også testet hvorvidt det er muligt i vores customerTests klasse:



Derudover bruger vi AUTO\_INCREMENT ved oprettelse af nye primary keys, hvilket giver sikkerhed for, at vi kun har 1 primary key.   
Der er ikke nogen default værdier for nogen kolonner i vores database. Det kommer af, at der som udgangspunkt aldrig bør være tom information uploadet til vores database (bortset fra primary key, som bliver håndteret af AUTO\_INCREMENT). Da vi forventer, at data der bliver sendt til vores MySQL database er udfyldt, har vi heller ikke gjort mere for at tjekke hvorvidt det er valid data udover NOT NULL og type (int, varchar, dateo osv.).   
Der vil også være nogle tjeks i selve java delen af koden. Blandt andet kan man ikke lave et validt motorhome hvis ens type ligger over 8, eller under 1, som er henholdsvis max og min type nummer. Det er opnået ved koden:  


I motorhome modellen.   
I det store hele skulle der ikke være mulighed for at indtaste data, som ikke giver mening I deres respektive celle.

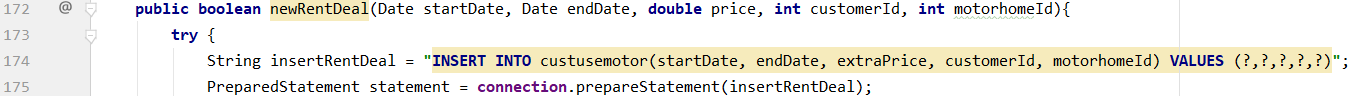
On delete/update no action:

On delete/update no action betyder, at hvis man sletter noget data, der har en foreign key, kan man påvirke de tabeller den respektive foreign key hænger sammen med. Det er ikke gjort i vores database, da vi ikke forventer at slette noget, der har en påvirkning andre steder, vi ønsker fx ikke at slette vores kundekontakt information, bare fordi de ikke længere lejer et motorhome (i tilfælde af, at man finder skader fx).   
Et sted i vores database, hvor det ville være ønskværdigt, ville være ved damages. Det giver ikke stor mening at gemme skadesrapporter om et motorhome, man ikke længere ejer. Det er ikke gjort, da motorhomes tabellen ikke gemmer noget information om damages, og man derfor ikke kan slette data i damages, ved hjælp af “action on delete”. Hvis man sletter en skade fra damages, ønsker man ikke at slette ejerskabet om det motorhome man lige har fixet.

DML:

DML står for data manipulation language. Det beskæftiger sig med manipulation af den data der ligger i de forskellige tabeller, de MySQL statements vi bruger i forbindelse med DML er insert, update og Delete.

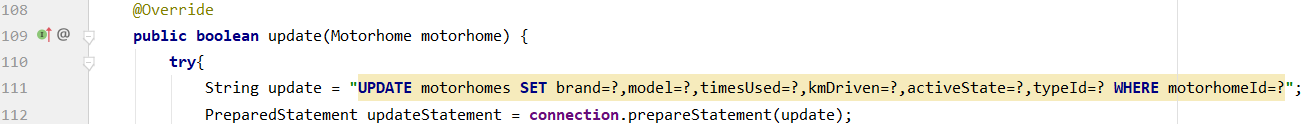
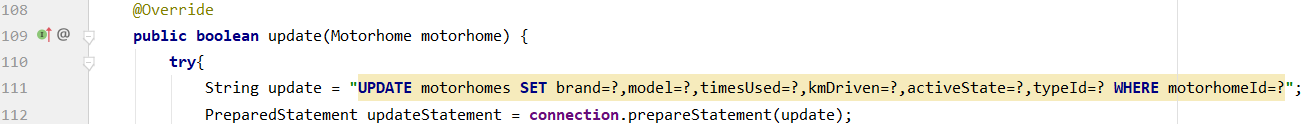
Insert:

Insert bliver brugt til at indsætte nye rækker i en tabel. Alt efter hvordan ens tabel er blevet oprettet, kan man vælge at udfylde dele af rækken, og dermed udelade felter, eller fylde alle cellerne i rækken ud. De fleste kolonner i vores database er af typen NOT NULL, som betyder at de skal have information før MySQL godtager en statement. Derfor vil man være tvunget til at udfylde alle kolonners data når bruger insert i vores database, den største undtagelse for dette er de primary key id, som bliver udfyldt automatisk af MySQL. Det sker da de er blevet oprettet med primary keys der har auto\_increment, som betyder at MySQL automatisk giver primary key 1 højere end tidligere.   
Et eksempel på data der bliver indsat i vores tabel er i metoden newRentDeal som ligger i MotorhomeRepository filen. Her bliver der oprettet en ny lejeaftale (som navnet indikerer), og derfor skal den indsættes i vores MySQL database. Der bliver brugt følgende insert statement: 

Man skriver hvilken tabel man indsætter data og hefter values som er den data der skal indsættes.

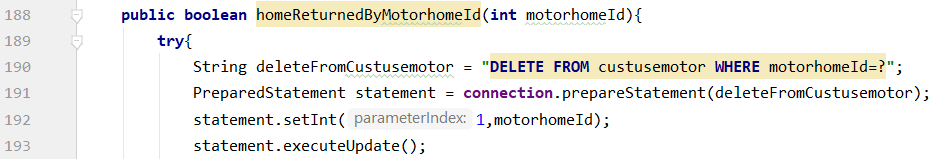
Update:

Update funktionen bliver brugt til at opdatere en allerede eksisterende række, hvis man fx skal ændre en kundes efternavn fordi de blevet gift. Det gøres meget lig insert statement, med den meget vigtige del er, at man skal huske at bruge WHERE, så man ikke opdaterer alle rækkerne i en tabel, det ses fx i update metoden fra MotorhomeRepository:

som ses i eksemplet ovenfor, skriver man update, den tabel man vil opdatere, også setter man de værdier man gerne vil ændre. Her er WHERE motorhomeId=? Den vigtigste del af linjen, fordi uden den ville den her metode opdatere ALLE rækker i hele motorhomes tabellen med de nye værdier. Hvis man kommer til det, er det godt innoDB har mulighed for rollbacks, hvis altså man slår det til... Hvilket vi ikke har gjort...

Delete:

Delete er den SQL statement der fjerne en række fra en tabel i databasen, det skal helst ikke forveksles med drop statement, da der er væsentlig forskel. Når man bruger en delete statement skal man, på samme måde som i update, være meget opmærksom på, at man kun deleter lige præcis den række man gerne vil, og altså ikke alle de andre. Derfor er det også vigtigt at huske den rigtige WHERE statement. Et eksempel på en delete metode findes ved homeReturnedByMotorhomeId metoden, der findes i klassen ActiveMotorhomeRepository:



DQL:

DQL står for Data Query Language, og er select statementen. Det er brugt til at trække data ud fra tabellerne i databasen. Select bliver brugt meget ofte i vores projekt, også i forbindelse med funktioner, som giver os behandlet data i stedet for rå data som man ville få med select \* from xx. Et eksempel hvor vi henter behandlet data, og ikke information direkte fra cellerne vil være metoden getCancelPrice i activeMotorhomeRepository, hvor vi trækker forskellen på 2 datoer:   


Her får vi prisen, som bliver læst fra cellen, og forskellen mellem start dato på lejeperioden. Det skal bruges til at udregne hvor meget af prisen man skal betale, baseret på hvornår man aflyser lejeperioden.

1. I MySQL 5.0.2 og nyere er database og schema det samme. Det er ikke det samme i andre SQL produkter. [↑](#footnote-ref-26543)