Hovedprosjekt i Programutvikling

Gruppe 29

Carl Petter Reinsnes s198756

Sebastian Ramsland s198762

Thomas Newman s198753

Forutsetninger

Vi valgte alternativ 2: Boligformidling som vår oppgave til prosjektet. Vi fant tidlig ut at programmets gjennomløping og søking passer godt med databaser som lagringsløsning men dette sto det lite om i den opprinnelige prosjektkrav-lista. Derfor forhørte vi oss med Eva Hadler Vihovde, hovedforeleser i programutvikling, om vi hadde lov til å bruke databaser. Det hadde vi, så lenge vi husket intern programstruktur.

Derfor har vi laget et program med følgende forutsetninger:

* Bruk av databaser som lagringsløsning
* IntelliJ som hovedutviklingsverktøy: vi fikk anbefaling fra sudentassistent Christian om sette oss inn IntelliJ og benytte dette som utviklingsverktøy.

Forord

I vårt prosjektarbeid har vi lagt stor hovedvekt på å utvikle et boligformidlingsprogram som enkelt kan utvikles videre. Vi ønsket å lage et program som er mest likt et reelt scenario. Vi kom derfor fort til en konklusjon om at det er definitivt mest hensiktsmessig å ha felles informasjon lagret i èn database som flere klienter (meglere) kan behandle. Vi har derfor benyttet oss av en MySQL-database som lagringsløsning. Dette betyr at programmet vil kreve tilgang til internett for å virke, da denne databasen ligger på Sebastian sitt webhotell.

Roller

Bruken av programmet innebærer to hovedroller: Potensiell leietaker og megler. Vi lagrer også data om eieren av boligen, men det er megleren som tar seg av arbeidet med opprettelse av leieavtalen mellom de to partene. Det er derfor leietaker og megler som er tilknyttet leiekontraktene. Boligeieren er ikke direkte oppført i en leiekontrakt, men er indirekte tilknyttet, ettersom han står oppført som boligeier. Boligen er naturligvis også oppført i kontrakten.

Milepæler

En gjennomgang av hvordan ukene har sett ut for oss.

Uke 13

Selv om vi fikk mye god planlegging i gang, så var det et faktum at vi måtte lese oss selv opp på det vi drev med. Brukte derfor slutten av denne uken på å lese oss opp på databasebruk og eventuelle løsninger.

Uke 14

Begynte å kode. Mye prøving og feiling. Sebastian begynte med databasen.

Uke 15

Like før påske, ble vi nogen lunde enige om utseende på programstrukturen, og hvem som skulle gjøre hva. Planen var at vi skulle kode så mye som mulig i løpet av påsken, og at Sebastian skulle bli ferdig med databasen.

Uke 16

I påskeuken ble jeg ferdig med Boligtabell-klassen. Sebastian ble ferdig med databasen.

Uke 17

På grunn av prioritering, ble det ikke så mye arbeid denne uken. Vi hadde dessuten noen spørsmål om arbeidet til stud-ass, som vi ikke fikk kontakt med denne uken. Her var vi usikre på hvilken retning prosjektet tok.

Uke 18

Ble denne uken klart at vi fikk to klasser for panelene: En for tabeller og en for vinduer. Dette var først og fremst fordi vi programmerte litt vel individuelt, men separat fant gode løsninger på forskjellige problemer, en også for å kunne lese gjennom koden litt enklere.

På grunn av dårlig tid denne uken, ble vi enige om virkelig å sette inn støtet fra og med neste uke.

Uke 19

Jobbet svært mye og lenge denne uken. Fant ut at det var lettest å vise tabellene med DefaultTableModel. I tabellklassene fant Carl ut at det var litt for vanskelig å få cachedrowset over i JTable. Dermed ble det DefaultTableModel. Han opprettet også et brukervindu og utviklet dette ferdig. Når det ble prøvd med boligdelen av prosjektet, funket det veldig bra, og vi ble dermed enige om at vi skulle bruke Boligvindu-klassen som modell for resten av panelene.

Sebastian ble denne uken helt ferdig med arbeidet rundt klassene, som omfattet spesiell søking, gjennomløpninger, get-metoder, oppdateringsmetoder for kolonner med mer.

Uke 20

Programstruktur

Grunnklasser

Grunnklassene Person, DwellingUnit og Contract håndterer all essensiell informasjon tilknyttet til selve bruken av programmet. Disse klassene er veldig like, og det er stort sett hvilke verdier som mottas/returneres og oppdateringsmetoder som varierer. Grunnklasene behandler et RowSet (av typen Cached eller Filtered), hvor man kan hente ut nødvendige verdier fra en bolig, person eller kontrakt. Det er også mulig å endre på verdier, lage nye rader eller slette rader. Man kan endre på flere rader og verdier lokalt, før man skulle ønske å sende disse endringene samlet til databasen. På denne måten belaster man database-serveren minst mulig, ved å forhindre enkelt-tilkoblinger for hver eneste lille endring man skulle ønske å foreta seg.

Alle tre grunnklassene har mulighet til å ta imot et allerede opprettet SQLInterface, radnummer og RowSet (CachedRowSet eller FilteredRowSet). Dersom klassene opprettes uten parametre, kobler de seg til databasen ved bruk av klassen SQLInterface. SQLInterface oppretter da et nytt CachedRowSet (og FilteredRowSet for fremtidig implementering) som lagres lokalt i cache - og kobler seg så fra databasen igjen. Grunnklassene henter da dette RowSetet fra SQLInterface. Konstruktørene (uten parameterverdier) til grunnklassene sender så med en SQL-query til klassen SQLInterface, som henter ut alle verdier forbundet med denne spørringen. Det er disse verdiene som lagres i et RowSet.

I tillegg til grunnklassene, er det også en “underklasse” med navn Location som benytter seg av tilsvarende metode. Denne klassen brukes utelukkende til å hente ut informasjon tilknyttet et spesifisert postnummer. Location-klassen har en metode som tar imot et postnummer. Metoden finner da informasjon tilhørende dette postnummeret. Så endrer den klassens initialiserte variabler stedsnavn, kommune, fylke og adressekategori. Deretter kan man hente ut disse verdiene ved hjelp av get-metoder. Ettersom det lagres adresseinformasjon om personer og boliger, så benyttes denne klassen av disse to grunnklassene. Allikevel er klasse helt uavhengig av grunnklassene, ettersom det kan være behov for å sjekke opp postnummere ved andre tilfeller.

SQLInterface

SQLInterface er en klasse som benytter seg av JDBC-driveren til å opprette en kobling til vår MySQL-database. Klassen tar imot en SQL-query, sender queryen til databasen, mottar et ResultSet som klassen omgjør til et CachedRowSet og/eller FilteredRowSet. Det er verdt å merke seg at denne klassen nå oppretter både et CachedRowSet og FilteredRowSet. Ved implementering av FilteredRowSet, vil det ikke lengre være behov for å benytte seg av CachedRowSet, ettersom FilteredRowSet er en underklasse av førstnevnte. Vi har valgt å opprette begge av disse RowSetene for å vise at vi har tenkt utviklingspotensiale for programmet i fremtiden.

Grensesnitt

Programmet er delt opp i tre moduler: Boligpanel, Personpanel og Kontraktpanel. Hver av disse delene skal utgjøre en egen fane i et vindu med JTabbedPane implementert.

Hovedvindu

Hovedvinduet har 6 underliggende vinduer. Disse seks er beskrevet nedenfor. Boligdelen av programmet har to klasser Boligvindu og Boligpanel. Boligvindu skriver ut tabellrader, Boligpanelet har alt ansvar for endring, oppretting og sletting av tabellverdier, i tillegg til søk etter enkelte boliger. Det samme oppsettet gjelder for kontrakt- og persondelen.

Boligdelen

Dwellingtable

Struktur

Klassen Dwellingtable er bygd opp kun for å håndtere tabellene som skal skrives ut i vinduet. Opprinnelig skulle programmet også håndtere oppdateringer og innlegging av ny data, men dette ansvaret ble overført til klassen DwellingUnit, som kan opprette en rekke type objekter som i teorien kan hente ut diverse filtrerte lister og gjennomføre masseoppdateringer. Denne klassen er beskrevet grundigere nedenfor.

Dwellingtable-klassens metoder returnerer primært DefaultTableModel. Etter mye undersøking, fant jeg frem til at dette var det letteste for å få skrevet ut resultater basert på enkle databasesøk for å få skrevet ut alt eller skrevet ut spesifikke adresser.

Use-cases

Metoden showEverythingInDwellingtable() returnerer hele tabellen basert på et enkelt søk etter de viktigste verdiene i tabellen. Denne metoden mater data fra ResultSet inn i to vektorer, som blir sendt med DefaultTableModel-en som returneres. Legg merke til at metoden både kan utvides til å gjelde alle kolonneverdier, eller forminskes til å gjelde færre verdier. For full oversikt, se database-skissene, eller erstatt nåværende verdier i String query med «SELECT \* FROM dwelling\_unit».

Det samme gjelder metoden findDwellingUnitByAddress(String adresse), men denne returnerer i stedet et DefaultTableModel med spesifikke data basert på en enkel søking etter adresse. Søkingen er satt opp slik at når man søker etter adresse, kan man søke med etter adresser med en hvilken som helst bokstav eller en hvilken som helst rekkefølge av bokstaver. Man trenger ikke skrive hele adressenavnet.

I tillegg finnes metoden showOwnerDwellingUnits (String pNo). Metoden sender ut en liten tabell med verdier fra alle boligene hvor den medsendte personnummeret er registrert som utleier. Metoden brukes i personpanelet.

DwellingtableWindow

Struktur

Denne klassen er, tross navnet, et JPanel med indre JPanels med faste plasser. Hver av panelene har faste plasser på vinduet. Øverste panel er et buttonpanel hvor man kan legge til en rekke JButtons for andre funksjoner, type masseoppdatering eller lignende. Vinduet blir opprettet med et Dwellingtable()-objekt.

Til høyre er et panel hvor man kan få en lang tekststreng med info utskrevet ved søk etter id eller ved å trykke hvor som helst på en utskrevet tabellrad (Se metodene showEverythingInDwellingtable() og findDwellingunitByAddress() og tilhørende kommentarer).

Til venstre er panelet som skriver ut tabellrader ved søk etter adresse eller ved å trykke på «Vis alle boliger» - knappen. Disse knappene er i buttonpanelet.

Konstruktøren tar imot og deklarer et objekt av Dwellingtable-klassen og et objekt av DwellingUnit-klassen. Dwellingtable-objektet brukes til å opprette en JTable. Ved trykk på en av knappene får man enten opp alt innhold, eller alle som sendes inn i tabellpanelet ved kall på findDwellingUnitByAddress(String adresse).

Legg merke til at brukergrensesnittet på de andre vinduene er fullstendig likt dette: en like nøye beskrivelse kommer derfor ikke til å bli gitt for persontableWindow eller ContracttableWindow.

Use-cases

showTable(DefaultTableModel def) er den eneste metoden. Ved trykk på «Vis alle boliger»-knappen, opprettes et DefaultTableModel med verdi fra showEverythingInDwellingtable() fra Dwellingtable-klassen. Deretter sendes DefaultTableModel-en inn i showTable(), som oppretter en JTable med de tilsendte verdiene.

Ved trykk på søkeknappen, blir man spurt om hva man vil søke etter. Foreløpig er dette kun adresse, men det er store rom for utvidelser. Når man trykker adresse, kommer det opp et vindu der man kan skrive inn adressen. Her kan man skrive inn full adresse, eller søke etter alle adresser med tilfeller av en bokstav eller en bestemt rekke bokstaver. DefaultTableModel blir opprettet, og sendes til showTable().

Denne tabellen blir opprettet med objekt-klasser, slik at boolean-verdier fremstår som check-boxer, dato-verdier blir skrevet ut med satt dato og måned, og lignende. Tabellen får dessuten en muselytter, som ved klikk på en hvilken som helst kolonne sender en tekststreng med info fra hele den tilhørende raden til utskriftsområdet. Tabellen blir opprettet med en JScrollPane.

DwellingUI

Struktur

Dette vinduet er den mest sofistikerte delen av programmet. Her utføres alle operasjoner som endrer databaseverdiene. Panelet inneholder et MainPanel extends JPanel(), der alle underliggende paneler blir lagt inn som egne, private klasser som extender JPanel. De underliggende panelene er HeaderPanel(), DataFieldPanel() og SidePanel().

Vinduet oppretter et objekt av DwellingUnit()-klassen. Det finnes tre forskjellige konstruktører som oppretter forskjellige varianter av rowset til bruk når man skal gjøre operasjoner på databasen. For nærmere beskrivelse, se DwellingUnit(). Foreløpig bruker dette programmet kun den konstruktøren som ikke har parametere. Ved opprettelse fylles et cachedrowset med databaseverdiene i bolig-databasen. Objektet av dette cachedrowset-et gjør operasjoner i DwellingUnit som er fanget i diverse metoder som kan brukes av DwellingUnit()-objektet. Blant metodene som blir brukt er moveToCurrentRow(), insertRow(), updateRow(), acceptChanges(), refreshValues(), moveToInsertRow(), oppdateringsmetoder for integer-verdier, boolske verdier og string-verdier, samt get-verdier for alle datafeltene. Igjen, se nærmere på DwellingUnit() for full beskrivelse.

MainPanel setter rett og slett størrelsen på siden, og en rekke verdier til brukergrensesnittet. HeaderPanel inneholder en JLabel som ved visning av en enkelt bolig sender ut ID og adresse. Dette blir vist i toppen av panelet.

DataFieldScrollPanel() inneholder igjen en rekke private klasser som extender JPanel, men i motsetning til de andre, underliggende panelene extender denne JScrollPane, og oppretter DataFieldPanel() extends JPanel inni denne klassen. Inni her legges tre paneler: MainInfoPanel(), IncludedPanel() og AdditionalPanel(). Alle disse extender JPanel.

I MainInfoPanel finnes en rekke JTextField som all informasjon som er vital for opprettelse av nye boliger legges. IncludedPanel() har en rekke boolske checkboxer. Her lagres nesten all informasjon om nøyaktig hva som er inkludert i boligen. Dessverre, på grunn av problemer med Layout-en som vi ikke fikk fikset, viser ikke dette panelet de boolske verdiene for inkludert heis og inkludert handikap-tilpasning.

AdditionalPanel() inneholder all tilleggsinformasjon som i utgangspunktet får automatiske verdier. Her finnes kun JTextFields. Alt dette inngår i DataFieldPanel().

I SidePanel() extends JPanel ligger ControlPanel()extends JPanel implements ActionListener. ControlPanel() inneholder alle knappene som brukes I programmet. Her ligger også actionPerformed()-metoden. Dette panelet inneholder seks knapper, for søking etter enkelte boliger, opprettelse av nye boliger, sletting av boliger, lagring av nye verdier i allerede eksisterende boliger, avbryting av oppdateringer, samt en lukkeknapp for hele vinduet.

Legg merke til at brukergrensesnittet på de andre panelene er ganske likt dette.

Use-cases

Metoden updateFields() oppdaterer feltene som blir vist i panelet. Ettersom rowset-et som blir vist har en peker som hopper frem og tilbake mellom boligene den viser, oppdaterer denne metoden feltene, slik at man får verdiene som hører til boligen man har pekeren på. Metoden inneholder kun de forskjellige feltenes implementerte set-metoder, som får verdier gjennom DwellingUnit()-klassens get-metoder. Denne metoden brukes ved søking etter enkelte boliger, sletting av enkelte boliger, og kansellering av oppdateringer.

clearFields() brukes ved opprettelse av ny bruker. Når man trykker på «Ny»-knappen, tømmes alle feltene for verdier.

Ved trykking på «Lagre»-knappen, utføres en sjekk. Når man trykker på «Ny-knappen», får en boolsk variabel som sjekker om det skal være en ny bolig eller ikke, verdien true. Ved verdien true, flyttes pekeren til innsettingsraden, og metoden saveNewDwellingUnit() kaller en rekke oppdateringsmetoder fra DwellingUnit-objektet for å sette inn utfylte verdier i den nye raden. På slutten settes raden inn i databasen med insertRow() og acceptChanges().

Dersom den boolske variabelen nevnt i forrige paragraf har verdien false, kalles istedenfor updateDwelling(), som oppdaterer alle verdier som blir satt inn i feltene. Deretter kalles updateRow() og acceptChanges().

Ved trykk på «Avbryt»-knappen, kalles metoden cancelUpdates() fra DwellingUnit-klassen.

Kontraktdelen

Contracttable

Denne klassen er veldig lik Dwellingtablel-klassen. Metodene showEverythingInContracttable() og findContractByDwellingunit(int id) viser henholdsvis alle kontrakter i databasen eller alle kontrakter tilknyttet en spesiell bolig. Som metodene i Dwellingtable-klassen, fyller metodene inn vektorer med ResultSet-data fra databasesøk, og disse blir brukt som parametere i DefaultTableModel-en som sendes ut.

I tillegg har klassen en metode kalt showPersonContracts(String pNo). Denne metoden returner alle kontraktene i tilknytning til en bestemt persons personnummer, men bare hvis personen er leietaker eller megler. Metoden brukes i personpanelet.

Kontraktvindu

Struktur og Use-cases

I likhet med DwellingtableWindow()-klassen inneholder denne klassen et buttonpanel med knappene, et tabell-panel hvor data blir skrevet ut, samt et utskriftsområde som blir brukt av muselytteren. Vinduet blir opprettet med et Kontrakttabell-objekt, som sender DefaultTableModel-er fra metodene showEverythingInContracttable() og findContractByDwellingunit(). Disse blir sendt til showTable()-metoden, som også er fullstendig lik showTable()-metoden i DwellingtableWindow().

Persondelen

Persontabell

Ligner på Contracttable og Dwellingtable-klassene. Har metodene showEverythingInPersontable() og findPersonBySurname(String etternavn), som returnerer henholdsvis alt i persontable, eller alle data med etternavn lik den tilsendte parameteren. I motsetning til den liknende metoden findDwellingUnitbyAddress(), må man skrive inn fullt etternavn her. Begge metodene sender ut et DefaultTableModel, som blir opprettet slik det er forklart i over om Dwellingtable-klassen og Contracttable-klassen.

Personvindu

Struktur og Use-cases

Igjen: brukergrensesnittet er fullstendig likt DwellingtableWindow og ContracttableWindow. Et objekt av Persontable blir opprettet, og showTable(), helt likt som de i ContracttableWindow og DwellingtableWindow, tar imot DefaultTableModel-er fra showEverythingInPersontable() og findPersonBySurname(), og oppretter tabeller med disse, når man trykker henholdsvis «Vis alle personer» og «Søk».

**Tabs**

Struktur og Use-Cases

Denne klassen er for å implementere JTabbedPane og samler JPanels oppi en klasse hvor hvert JPanel legges til en JFrame og blir en tab i vinduet. Det opprettes objekter for JFrame kalt inni programmet frame, Person, Contract og DwellingUnit klassene og PersonUI, ContractUI og DwellingUnitUI klassene. Alle er satt til private.

Den implementerer Person, Contract og DwellingUnit klasser med forkortelsene person, contract og dwellingUnit før konstruktøren for JTabbedPane sånn at info som er lagt inn i databasen kan hentes ut fra JFrame til selve JPanelene ContractUI, PersonUI og DwellingUnitUI.

Etter det så har vi konstruktøren for JTabbedPane, forkorta til tabbedPane, og bygger opp tabs for hovedvinduets JFrame og legger inn JPanelene for brukergrensesnittene.

Så bygges opp JFrame. Konstruktøren deklarerer en ny JFrame kalt frame. Den neste metoden(frame.getContentPane().add(tabbedPane, BorderLayout.CENTER);) legger inn innholdet som er lagt inn i tabbedPane i forrige konstruktør.

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); metoden brukes til å lukke programmet helt og avslutter alle bakgrunns metoder sånn at ekstra minne ikke blir brukt opp av programmet etter at vinduet er lukket.

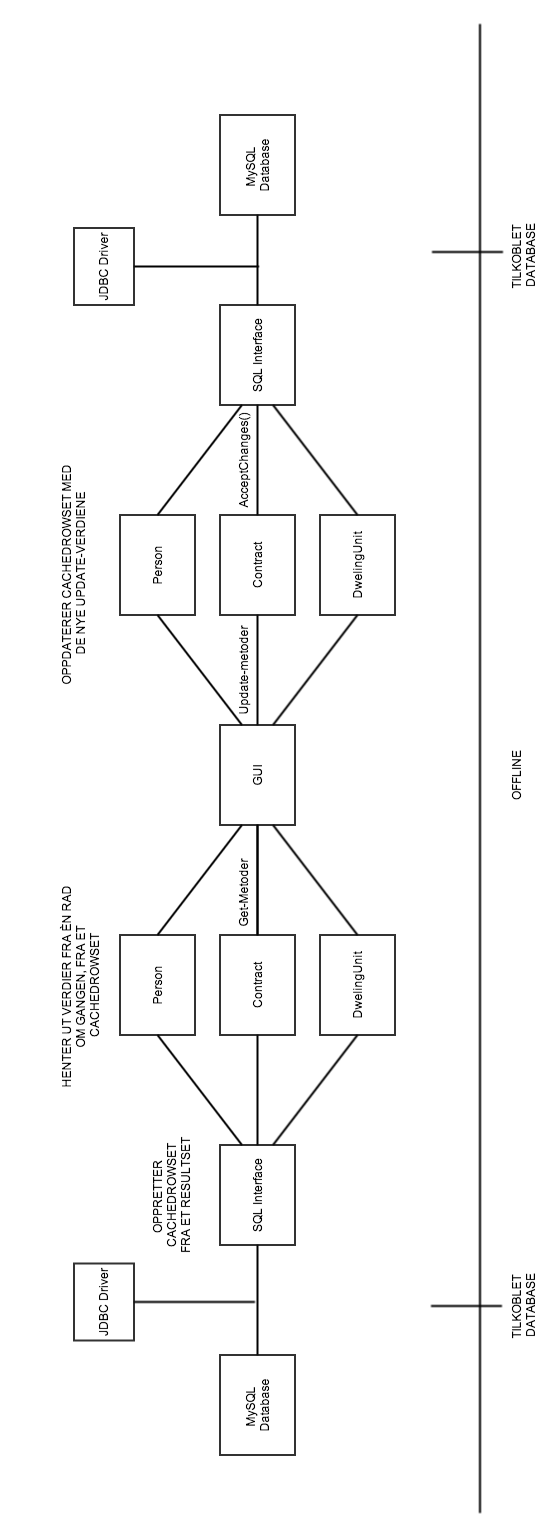
frame.setSize(1280, 720); metoden setter opp JFramens sin størrelse.

frame.setVisible(true); metoden brukes til å sette JFrame til å bli synlig.

frame.validate(); metoden brukes til å validerer JFramen og alle dens sub-komponenter.

Den neste konstruktøren er for å bytte java symbolet med vårt eget symbol for programmet. Bildet er vedlagt sammen med dokumentasjonen.

Sist, men ikke minst, så har vi Main metoden som aktiverer programmet.



Database - Vår lagringsløsning

Struktur

Databasen er inndelt i flere tabeller, men det er spesielt tre tabeller som er helt essensielle: Person, Bolig og Kontrakt. Det er disse som vil inneholde firmaets kundeinformasjon og historikk.

Person

Tabellen Person benyttes ved all lagring av personalia. Det vil si at både leietaker, huseier og megler benytter denne tabellen. Det boolske feltet “is\_broker” angir om en person er megler. Vi har valgt å ikke sette opp et eget felt for om en person er huseier eller leietaker. I realiteten kan en huseier også leie, og vi ser derfor på et slikt felt som overflødig. Dersom en person er å finne i feltet “property\_owner” i tabellen Dwelling Unit, så er det en konkret indikator på at denne personen er en huseier.

Bolig

Tabellen Dwelling\_Unit benyttes ved all informasjonslagring om et leieobjekt. Et leieobjekt er direkte tilknyttet en huseier ved bruk av personnummer. Huseieren må derfor eksistere i vår kundebase før leieobjektetet kan registreres i databasen.

Kontrakt

Tabellen Contract benyttes ved lagring av et leieforhold mellom de ulike partene. En kontrakt inneholder tre essensielle parter: leietaker, megler og bolig. Alle disse tre partene må eksistere i databasen for å kunne opprette en kontrakt. Kontrakten inneholder også start og sluttdato for når leieperioden gjelder. I tillegg lagres det informasjon for om kontrakten er signert av begge parter; det vil si leietakeren og megleren - og om depositum er innbetalt.

Følgende kontroller utføres før en kontrakt settes som status “gyldig” (valid):

Leietaker må ha passert en kredittsjekk (passed\_credit\_check)

Personen som er satt som megler (broker) i kontrakten skal sjekkes ved hjelp av feltet “is\_broker”.

Leietaker kan ikke være samme person som står oppført som eier av boligen. (property\_owner).

Megler kan ikke være samme person som står oppført som eier av boligen. (property\_owner).

Leietaker kan ikke være samme person som megler.

Depositum må være innbetalt. (paid\_depositum).

Kontrakten må være signert av begge parter. (signed\_by\_renter og signed\_by\_broker)

Nåværende dato må være innenfor den angitte kontraktperioden. (in\_effect\_date og expiration\_date).

Postnummer, stedsnavn, kommuner og fylker

Databasen er innlastet med tabeller som inneholder samtlige postnumre, poststed, fylker og kommuner i Norge. Det er derfor kun mulig å sette inn reelle postnumre (zip\_code) på person og dwelling\_unit.

Øvrige kommentarer

Tabellene er satt opp med strenge relasjoner til hverandre, ved bruk av primærnøkler, fremmednøkler og “constraints”. Det er for eksempel ikke mulig å slette en person, før personens kontrakter og boligene han eier også er slettet. Dersom vedkommende skulle av en eller annen grunn endre personnummeret sitt, så vil også dette oppdateres i kundens kontrakter og boenheter.

Tabellene Person, Dwelling Unit og Contract har alle automatiske datostemplingsfelt som oppdateres ved endring/opprettelse av en rad.

Kolonne “created” får automatisk en dato/tidsverdi ved opprettelse.

Kolonne “last\_modified” oppdateres automatisk ved hjelp av Triggers når et (eller flere) felt i raden endres.

Views:

Det er laget forhåndsdefinerte spørringer for enkel opplisting av:

kunder

meglere

kontrakter

postnummer og tilhørende stedsnavn/kommune/fylke

ledige leieobjekter

opptatte leieobjekter

leieobjekter og hva som er inkl. i leien.

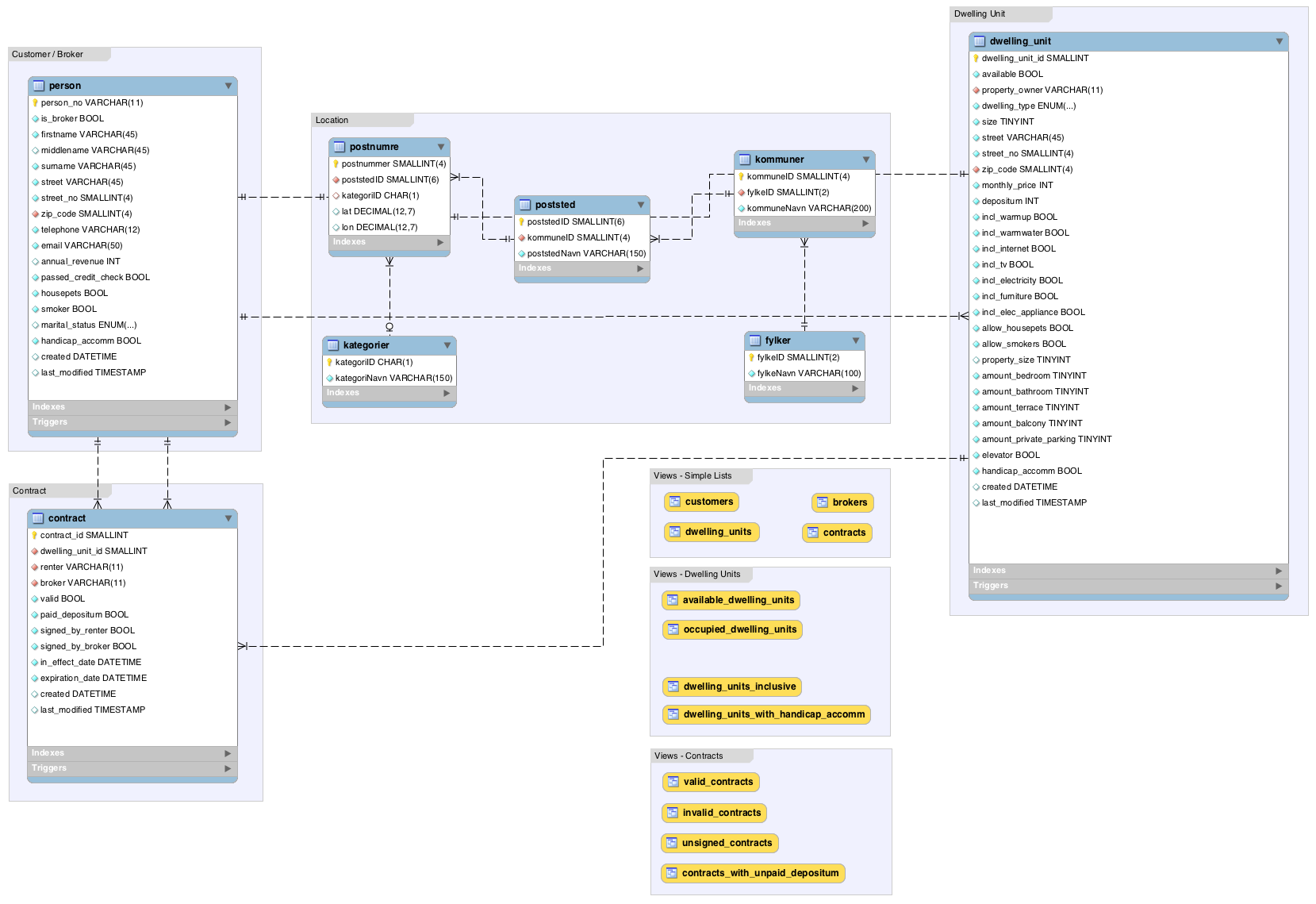
gyldige kontrakter

ugyldige kontrakter

usignerte kontrakter

kontrakter med ubetalt depositum

EER Diagram



Fremtidige forbedringer

**Innloggingssystem**

Innloggingssystem for firmaets ansatte. For øyeblikket har man mulighet til å angi hvem som er megleren i en kontrakt. Men det ville vært mer tidsbesparende for brukeren av programmet, dersom dette feltet ble fylt inn automatisk etter hvem som er logget inn i programmet. Dessuten kunne man også lagt til en ekstra kolonne i tabellene som automatisk fyller ut hvilken megler som sist gjorde en endring i en tabellrad.

**FilteredRowSet**

Implementering av FilteredRowSet til tabellvisning for å kunne (lokalt) filtrere et RowSet hentet fra databasen, og deretter sende dette rowsetet videre til tilhørende grunnklasse (person, bolig, kontrakt) for videre behandling. Grunnklassene støtter allerede bruken av FilteredRowSet, men implementasjonen mangler altså til tabellklassene.

Når FilteredRowSet er implementert til tabellklassene, kan man enkelt sette av/på søkefiltere for å snevre ned et søk i tabellen. For eksempel kan man sette opp et filter for å skjule alle boliger som ikke tillatter røykere eller husdyr.

**Paginering**

Implementering av paginasjon i et CachedRowSet/FilteredRowSet for effektivisering av programmets minnebruk. Ved bruk av paginering kan klassen håndtere mindre deler av et Rowset om gangen. Det finnes muligheter til å definere hvor mange rader man ønsker å hente fra hvert ResultSet. På denne måten forhindrer man å hente ned tusenvis av rader fra en tabell hver gang man ønsker å bruke programmet til å kanskje foreta seg en liten endring.

**Programflyt**

Sømløs navigering mellom tabellpanelene og enkeltvisningspanelene. Dersom en bruker for eksempel dobbeltklikker på et felt i tabellvisningen, åpnes denne boligen/kontrakten/personen i tilhørende enkeltvisningspanel.

**Backupløsning**

Mulighet til å kunne lagre endringer lokalt på en maskin, dersom tilkobling til databasen ikke lar seg gjennomføre på nåværende tidspunkt. For eksempel ved at brukeren får et valg om å lagre endringer som en fil lokalt på datamaskinen. Deretter kan denne filen åpnes på nytt ved en senere anledning i programmet, slik at endringene automatisk lastes opp til databasen.

**Grafisk fremstilling**

Databasen har lengde- og breddegrad-verdier lagret for hvert eneste norske postnummer. Dette er verdier man kunne fremstilt grafisk på et kart, for å vise hvor en bolig eller person befinner seg.