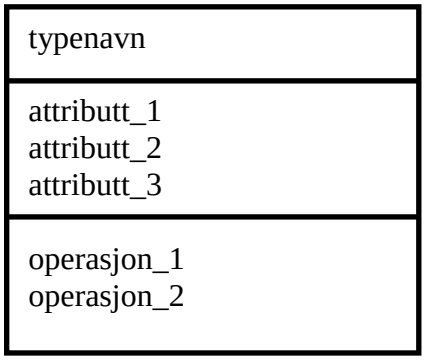
Databaser & objektorientering.
(Connolly & Begg, ch. 27)

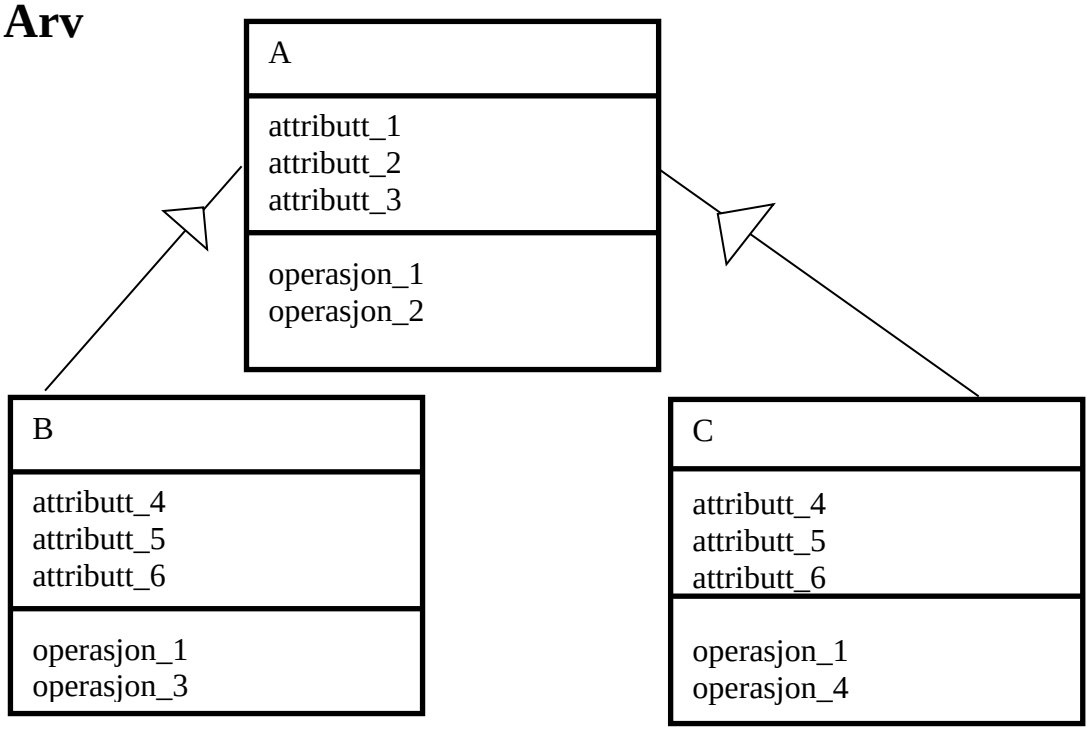
Noen grunnbegreper innen objektorientering.

Klasser og forekomster

- klasser beskriver strukturen for ”noe”. Beskrivelsen inneholder:
  - et navn
  - attributter /egenskaper / tilstander
  - operasjoner / metoder / handlinger
- forekomster av en bestemt klasse. Hver forekomst har sin egen ”identitet”.
- mange språk inneholder eksplisitte funksjoner for skaping og sletting av objekter (konstruktører og destruktører).



Arv



- Egenskaper fra en overordnet vil også gjelde for de underordnede.
- Arv kan skje i et vilkårlig antall nivåer.
- Kan dermed bygge opp komplekse strukturer.
- Noen OO-implementasjoner har ”multippel arv” – dvs. at en klasse kan ha flere overordnede. Vanskeligheten er at man arver egenskaper fra flere overordnede klasser.

Tilgjengelighet av data og handlinger i for et objekt

- Data og handlinger kan være
  - prohibited - ingen får adgang
  - protected - kun underordnede får adgang
  - public - alle får adgang

Informasjonsskjuling.

- Data skjules innenfor objektet. Det lages i stedet operasjoner for å hente, endre data etc. i et gitt objekt. ==> Data er uavhengig av lagringsstruktur.
- Ulike objektorienterte språk har ulik grad av informasjonsskjuling.

Abstrakte datatyper

- Det kan defineres ”datatyper” som ikke inneholder noen data fra applikasjonen som sådan. Andre objektklasser kan arve fra disse, og dermed få en rekke egenskaper ”gratis”.
- Typiske eksempler er lister, køer, ulike former for trær etc. som har operasjoner som finn (….), finn_neste, settinn (…), slett (…) etc.

Meldinger

- Et objekt kan ”be et annet objekt” om å gjøre en av de operasjonene det andre objektet ”kan”.
- Man får dermed kommunikasjon mellom objektene, ”samarbeide” og et ”dynamisk system”.

Polymorfisme

- Det utføres ulike operasjoner avhengig av hvilken type objektet er. Eks:
  For alle A: Utfør operasjon_1. Da blir ulike operasjoner utført avhengig av om det konkrete objektet er av typen A, B eller C.
- Systemet vet ikke hvilken kode som skal utføres før programmet kjøres. Kalles sen binding (koden ”bindes” ved kjøretid, ikke ved kompilering).
- Kraftfull egenskap, og av mange betraktet som det eneste ”helt nye” i OO.

Objektorienterte og objekt-relasjonelle databaser

Mange mener at tradisjonelle relasjonsdatabaser har svakheter, bl.a. fordi strukturen er for enkel for å avspeile virkeligheten på en god måte. Dette gjelder bl.a.

- hvorledes behandle ting som er nesten like, men ikke helt like? Stikkord: arv.
- hvorledes strukturere og behandle komplekse data (bilder, grafer m.m.)? Stikkord: komplekse datatyper. Behandling av BLOB-er.
- hvorledes legge metoder/operasjoner nær knyttet til data inn på en naturlig måte?
  Stikkord: objektorientert tenkning.
- mulighet for navigering/rekkefølge i postene.
  Stikkord: Bør kunne gå til første, forrige, neste, siste etc., relativt til hvor en er i datamengden. Dessuten: mulighet for rekursjon.

Tre retninger:

Behold relasjonsmodellen	Integrer de to modellene:	Forkast relasjonsmodellen
med noen få utvidelser (bl.a. komplekse datatyper)	Objekt-relasjonell (ORDB)	Ny modell: Objektorienterte databaser (OODB)

OODB:

- Hvert objekt har
  - identitet (typisk via en intern peker, OID = object identity), ikke via primærnøkler - systemgenerert.
  - tilstand (verdier på variable) - dvs. som verdier en relasjonsdatabase
  - oppførsel ("hva kan objektet gjøre") - triggere er noe av det samme, men ikke gjennomført.
- I stedet for domener defineres det objektklasser.
- Arv av objektklasser (sub- og supertyper)
- Sen bindig, polymorfisme etc.

Fordeler:
- hastighet
- passer bedre med OO programmeringsspråk (slipper å konvertere mellom relasjonell organisering på disk og objektorientert organisering i programmet/minne). Kort sagt: data lagres også som objekter.
- bedre til å håndtere komplekse strukturer
- etc.

Vesentlig utfordringer:
- er det logisk at en database skal innkapsle sine data ?????
- hvorledes lagre objekter på disk, og hva skal være persistent og hva behøver ikke å være persistent.
- hvordan beholde entydige pekere mellom minne og (muligens flere store) disker. Ulike metoder finnes, bl.a. konverteringstabeller av ulikt slag.
- hvorledes håndtere transaksjoner
- hvorledes håndtere skjemaevolusjon (dvs. at metadata endres over tid), versjoner av de samme data og metadata.
- et godt spørrespråk, som mest mulig er det samme som SQL. OQL (Object Query Language) er foreslått og ser temmelig likt ut, men noe av dette er bare overflatisk likhet. Arbeides med samording SQL - OQL.
- hvorledes få en standard (OMG - Object Management Group har forslagene, men ikke implementert en allmenn standard).
- hvorledes få til en "god teori"

Alternativet er bl.a. et objektorientert lag rundt RDBMS-et, f.eks. mellomvare.

OODB lagringsmodell:

Data kan lagres som

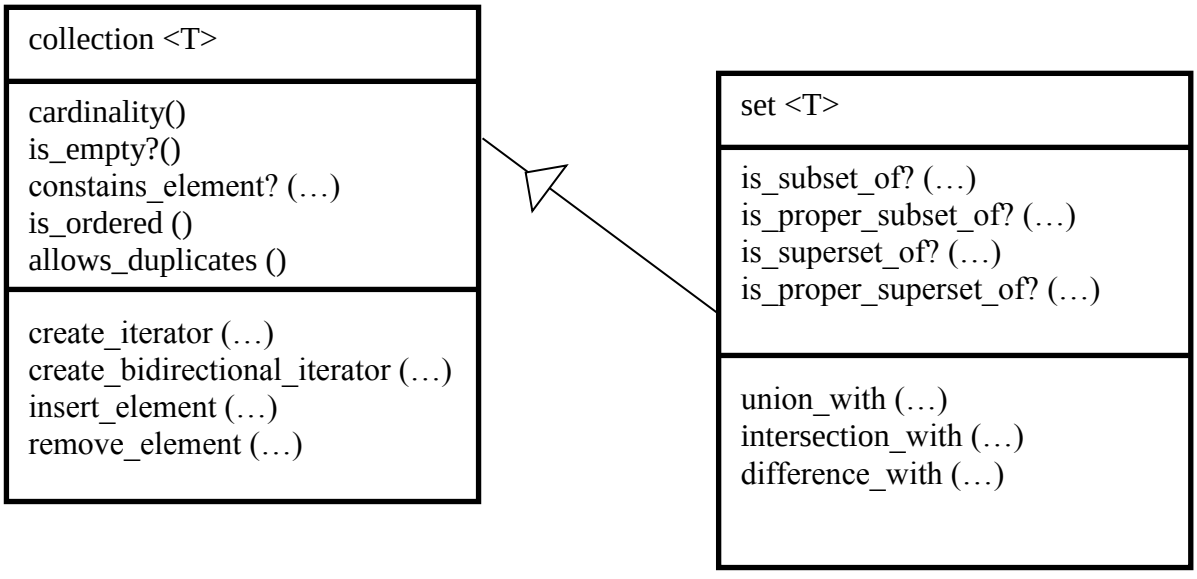
- sett (usortert, tillater ikke dublikater, a la relasjonsdatabaser)
- bags (usortert, tillater dublikater)
- lists (sortert, tillater dublikater)
- arrays (endimensjonal array med variabel lengde)
- dictionary (usortert sekvens av (nøkkel,verdi)-par).
- struktur (består av ulike elementer)

Disse kan igjen brukes som byggestener for mer komplisert elementer, f.eks. et sett, hvor hvert element er en struktur.

Egenskaper:

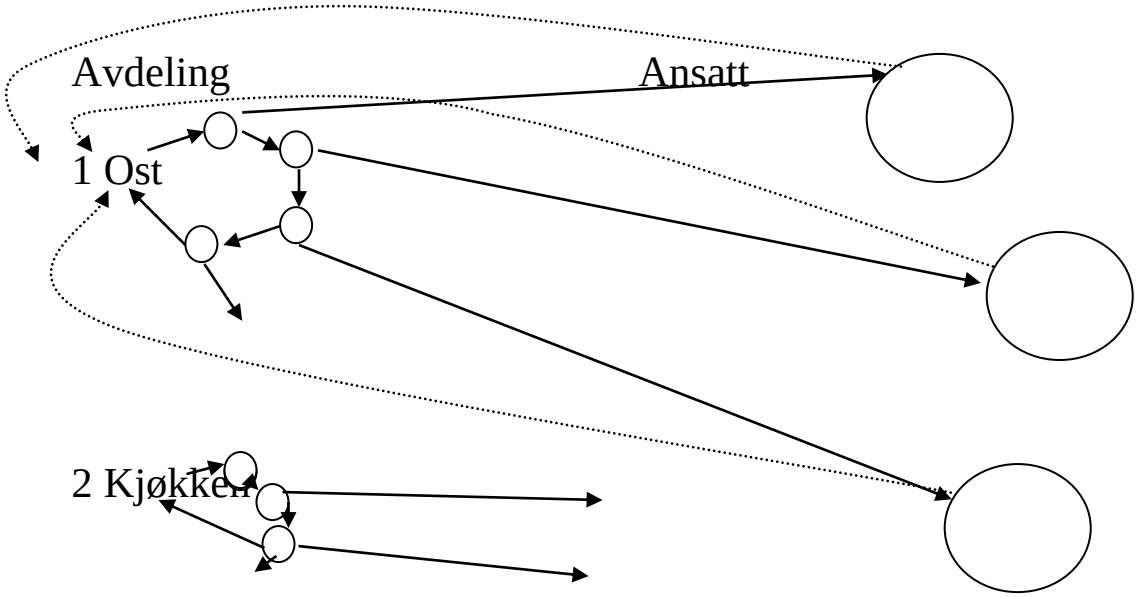
	Samme type	Sortert	Unik	Indeksert / Nummerert	Navngitt
Set	✔		✔		
Bag	✔				
List	✔	✔			
Array	✔	✔		✔	
Dictionary	✔		✔		
Structure					✔

De 5 første er spesialiteter av collections / kolleksjoner, og egenskaper som arves til de mer spesialiserte objektklassene:



1:mange, 1:1, mange:mange.

En 1:mange kan f.eks. lagres som:



Altså:
Fra 1 til mange: kolleksjon av pekere til objektene (ved å lagre det andre objektets OID).
Evt. også fra mange til 1: peker til objektet på 1-siden (ved å lagre det andre objektets OID), stiplet over.

1:1

Toveis pekere direkte til hverandre.

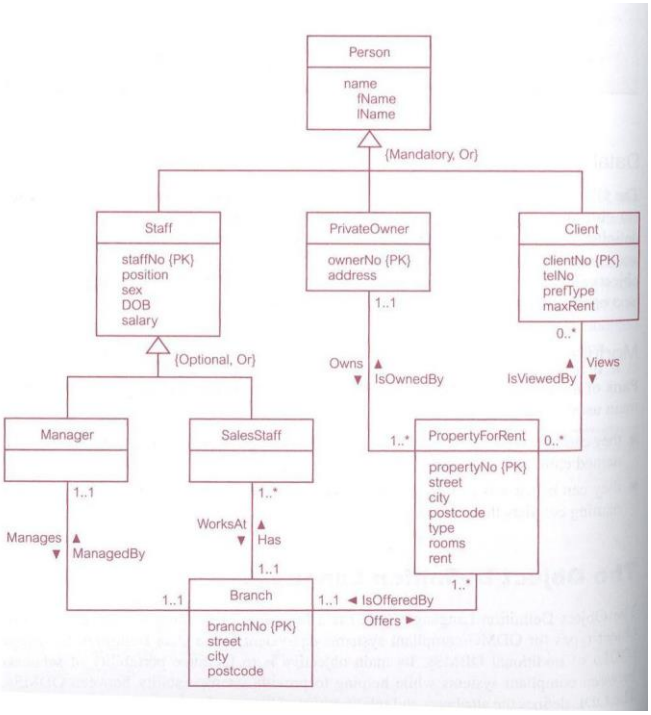
M:m

Enten: liste på begge sider
eller: "entitetisering" og behandling som to 1:mange.

Viktig observasjon:

Begrepene primærnøkkel og fremmednøkkel trengs ikke, derimot er entydighet og evt. nødvendighet/not null vesentlig.

Datamodell m/ arv transformert til definisjon av OODB (Fra Connolly & Begg)




```
module DreamHome {
    class Branch                                    // Define class for Branch
     (extent branchOffices   key branchNo)
    {
    /* Define attributes */
     attribute string branchNo;
     attribute struct BranchAddress {string street, string city, string postcode} address;
    /* Define relationships */
     relationship Manager ManagedBy inverse Manager::Manages;
     relationship set<SalesStaff> Has inverse SalesStaff::WorksAt;
     relationship set<PropertyForRent> Offers inverse PropertyForRent::IsOfferedBy;
    /* Define operations */
     void takeOnPropertyForRent(in string propertyNo) raises(propertyAlreadyForRent);
    };

    class Person {                                  // Define class for Person
    /* Define attributes */
     attribute struct PName {string fName, string lName} name;
    };

    class Staff extends Person                      // Define class for Staff that inherits from Person
     (extent staff   key staffNo)
    {
    /* Define attributes */
     attribute string staffNo;
     attribute enum SexType {M, F} sex;
     attribute enum PositionType {Manager, Supervisor, Assistant} position;
     attribute date DOB;
     attribute float salary;
    /* Define operations */
     short getAge();
     void increaseSalary(in float increment);
    };

    class Manager extends Staff                     // Define class for Manager that inherits from Staff
     (extent managers)
    {
    /* Define relationships */
     relationship Branch Manages inverse Branch::ManagedBy;
    };

    class SalesStaff extends Staff                  // Define class for SalesStaff that inherits from Staff
     (extent salesStaff)
    {
    /* Define relationships */
     relationship Branch WorksAt inverse Branch::Has;
    /* Define operations */
     void transferStaff(in string fromBranchNo, in string toBranchNo) raises(doesNotWorkInBranch);
    };
};
```

ORDB:

Utvidelse av relasjonsmodellen i objektorientert retning.

== > kan beholde nåværende applikasjoner etc.

Vesentlige momenter:

- komplekse datatyper (≈ objektklasser) med arv
- det betyr at en rad kan inneholde et komplekst, strukturert element.
- metoder
- utvidet SQL til å kunne takle dette (vesentlige ingredienser i SQL3), bl.a. for å behandle komplekse data, for å håndtere arv (skal man plukke data bare fra dette nivået i arven, eller skal alle underliggende også plukkes?)
- mulig å referere til et objekt direkte via en ref, ikke bare via primærnøkler. Tilsvarer OID-er i en OODB.
- kolleksjonstyper
- avanserte typer indekser (f.eks. som romlige data)

Det lages automatisk

for hvert attributt:

en observator-funksjon/get-funksjon	(**les** verdien)
en mutator-funksjon/set-funksjon	(**endre** verdien).

for hver tabell:

en kontruktør	(lage nye verdier)

Disse kan redefineres, og dermed skreddersy kontroller osv. for dens lesning hhv. endring hhv. for laging av nye instanser.

ORDB - kort eksempel i Oracle, I:

Generell syntaks - typer:

[…] er frivillige elementer.

- rottyper:
  create [or replace] type <navn> **as object (**
  <attributt- og metodedefinisjon>**) ;**

- typer som arver:
  create type [or replace] <navn> **under** <tidligere definert objekttype> **(**
  <attributt- og metodedefinisjon, evt. basert på tidligere definerte typer>**) ;**

Mellom) og ; kan man definere
[[NOT] INSTANTIABLE] - kan/kan ikke lage instanser av denne. INSTANTIABLE er default.

[[NOT] FINAL]] - kan/kan ikke la denne være grunnlag for videre arv. FINAL er default.

Tabeller basert på typer:

create table <navn> **of** <type> (evt. med flere attributter eller deklarasjoner, evt. basert på de definerte typene som datatyper).

Array-typer:

Enten som fast antall, array(<antall elementer>)) of <datatype>
Eller som et variabelt antall varray(<max. antall elementer>) of <datatype>

Vi ser at disse typene gjør at vi ikke lenger har atomiske verdier, vi har m.a.o. $(NF)^2$.

ORDB - kort eksempel i Oracle, II:

(bare små, utvalgte deler av mulighetene som finnes)

Eksempel – arv og sammensatte typer:

_t blir ofte brukt som suffiks på objektklasser

```
Create or replace type adresse_t as object (gateadresse varchar(30), postnr varchar(10))
not final instantiable;

Create or replace type kunde_t  as object
(kundenr integer, kundenavn varchar (40), adresse adresse_t)
not final instantiable;
```

Ny type som bruker en allerede definert type som type for en kolonne

Ny type som arver alt fra en underliggende type (+ kan ha egne kolonner).

```
Create or replace type utenlandskunde_t under kunde_t (landsnavn varchar(40)) final instantiable;

Create or replace type venner_t under adresse_t (vennid integer) final instantiable;
```

Eksempel på at adresse_t kan brukes i flere sammenhenger.

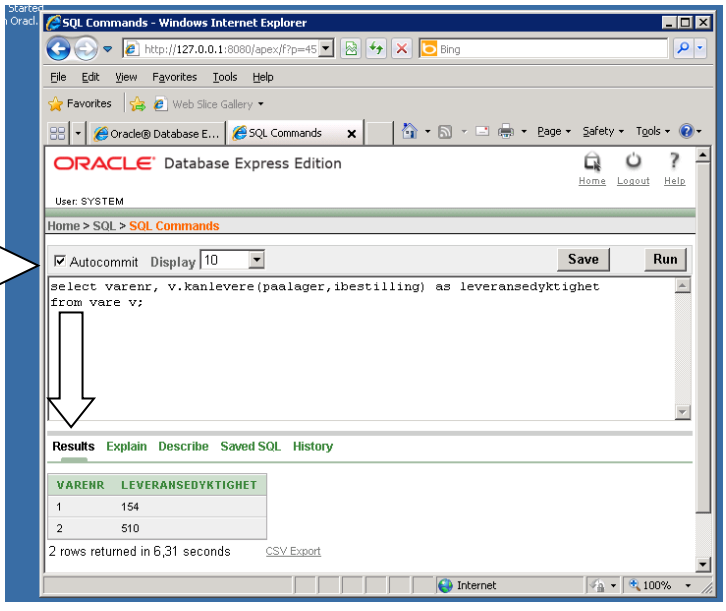
Eksempel - metoder:

```
Create or replace type vare_t as object
(varenr integer,
paalager integer,
ibestilling integer,
Member function kanlevere (paalager integer,ibestilling integer) return integer)
instantiable not final;

Create or replace type body vare_t as
Member function kanlevere return integer is
Begin
   Return paalager + ibestilling;
End;
End;
```

Deklarasjon og innhold kan gjøres i samme, men ofte lurt å splitte, spesielt hvis flere funk/pros.

Test:

```
create table vare of vare_t;

insert into vare values (1,54,100);
insert into vare values (2,10,500);
```



ORDB - kort eksempel i Oracle, III:

Eksempel – array:

Create type mndomsetning_t is array(12) of integer; - evt. varray

Denne kan dermed brukes i

create table omsforkunde (kundenr integer, mndomsetning mndomsetning_t);

Eksempel - referanser:

Create type varelevr_t under vare_t (levr_ref ref kunde_t);

Vi refererer altså til en av våre kunder som leverandør av denne varen (ofte ligger kunder og leverandører sammen i en tabell). Merk forskjellen:

- create type varelevr_t under vare_t (levr_ref kunde_t);
  - da må vi sette inn data (alle verdier i en kunde/leverandør) for hver gang det er samme leverandør for en vare ==> redundans & inskonsistens.

- create type varelevr_t under vare_t (levr_ref ref kunde_t);
  - refererer til et kunde-objekt, dvs. vi setter inn en peker til en kunde, ingen redundans.

NB! Skriver vi ut noe som er en instans av typen varelevr_t vil vi som levr bare få et stort tall, en systemgenerert «objektpeker»/OID.
Skal vi ha fram verdiene som denne peker på, må vi skrive deref(levr_ref).

For andre Oracle-eksempler, se f.eks.

http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14260/adobjint.htm &
http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96594/adobjbas.htm