UNIVERSITETET
I OSLO

Objektdatabaser og objektrelasjonelle databaser. SQL:1999

Databaser vs objektorientering

- Det er en inherent (iboende) konflikt mellom objektorientering og objektdatabasesystemer
- I objektorientering er innkapsling et sentralt begrep: Den eneste mekanismen som finnes for å la et objekt få vite noe om hvordan et objekt fra en annen klasse fungerer, er arv
- Objektdatabaser er databaser, og de skal ha et begrepsmessig skjema
- Dermed trenger objektdatabaser noe i tillegg til arv
- Det viktigste er en mekanisme som erstatter relasjons-databasenes fremmednøkler, men en slik mekanisme passer ikke inn i «ren» objektorientering

OMG og ODMG

- Derfor var det opprinnelig to industri- og universitetsoppnevnte standardiseringskomiteer for objektorientering:
  - OMG (Object Management Group) forvalter en de facto standard for hva objektorientering er.

    OMG har klassisk objektorientering som sitt mantra,

    og de vil ikke gå på akkord for å tilfredsstille databaseindustrien.
  - ODMG (Object Data Management Group) forvaltet en standard for objektdatabaser som tilfredstiller databaseindustriens krav, men som ellers ligger så nær opp til OMG-standarden som mulig.

    ODMG ble nedlagt i 2001.
    Fra 2004 forvaltes ODMG-standarden av OMG.
    Intet har skjedd med standarden siden 2001.

Viktige OMG-standarder

- **CORBA** (Common Object Request Broker Architecture)
  - Arkitektur for mellomvare mellom (objektorienterte) klienter og (objektorienterte) tjenere
  - Omfatter språket IDL (Interface Definition Language) som brukes til å spesifisere objektklasser
- **UML** (Unified Modelling Language)
  - Er på godt og vondt et nærmest enerådende sett av modelleringsspråk for objektorienterte systemer
  - Er i utstrakt bruk i industri og undervisning
- Hjemmesiden for OMG er http://www.omg.org/

Elementer i ODMG-standardens begrepsverden

- Objektidentitet og objektidentifikator (OID)
- Objekter og verdier (verdier er ikke objekter)
- Ekstensjon – instansene i en klasse
- Komplekse objekter og typekonstruktører
- Operatorer
- Programmeringsspråkkompatibilitet
  (sømløs overgang mellom database og applikasjons-programmer)
- Innkapsling (skjulte data)
- Type- eller klassehierarkier med arv og polymorfi

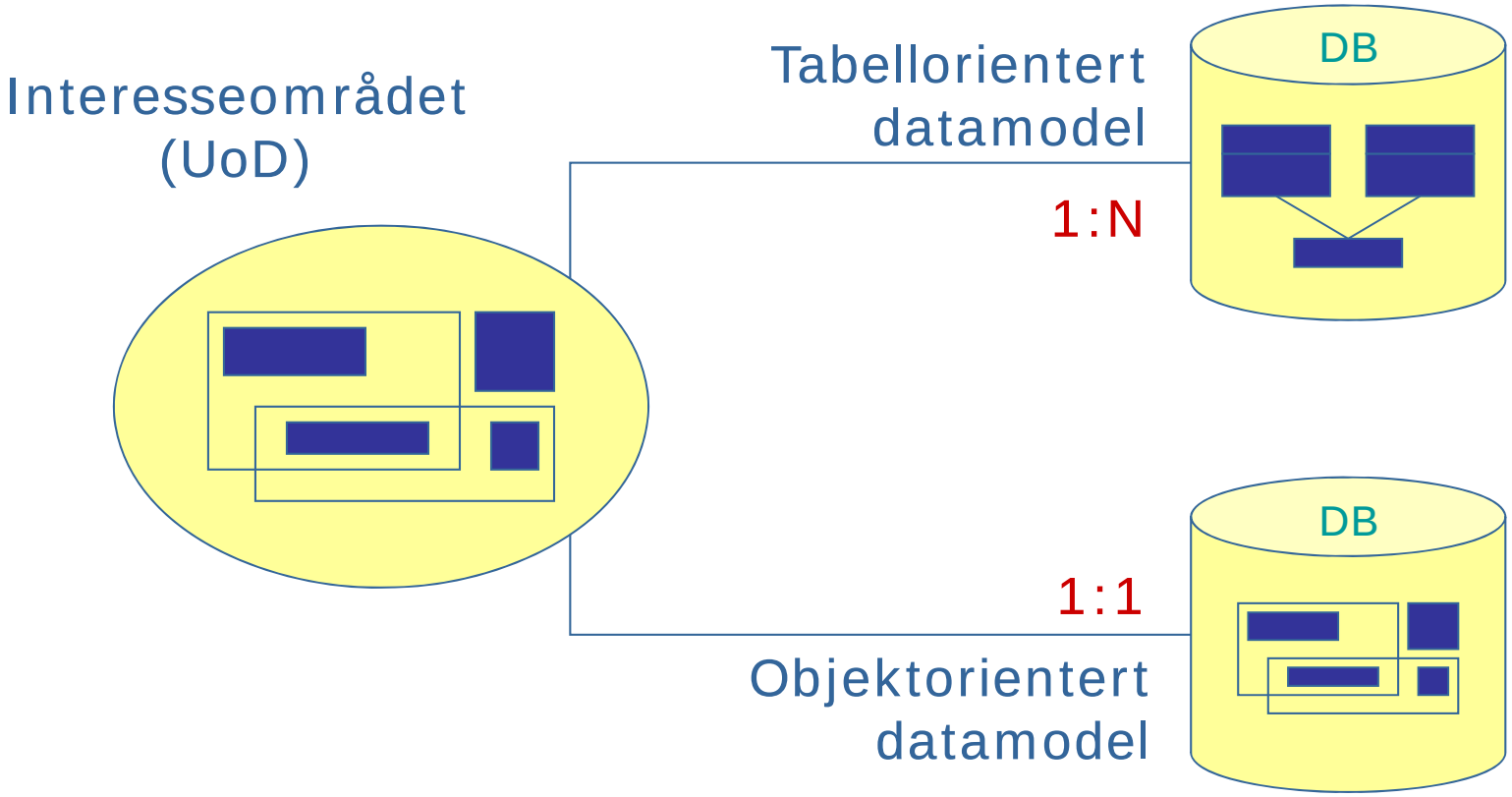
Det objektorienterte paradigme Klassifikasjon og taksonomi

- Paradigme: OO er laget for å kunne modellere et interesseområde (ofte kalt miniverden eller UoD) som en samling av kommuniserende og samarbeidende entiteter (enheter) som vi kaller ***objekter***
- Klassifikasjon: Objektene grupperes i klasser som er en
  - felles beskrivelse (mal) for alle objekter som tilhører samme klasse, kalt klassens ***intensjon*** (intent)
  - samling av objekter med felles egenskaper, kalt klassens ***ekstensjon*** (extent)
- Taksonomi: Klassene ordnes i super- og subklasser Dette gir opphav til arv av egenskaper og polymorfi

Abstraksjon og autonomi

- Objekt: <verdi, {operatorer}>
  - Objektene er distinkte og universelt identifiserbare
  - Operatorene er realisert (implementert) som metoder
- Verdi: Datastruktur
  - Verdier kan være forskjellig fra andre verdier, men ikke distinkte og universelt identifiserbare
- Innkapsling:
  Et objekt kan inneholde og skjule intern informasjon (Forutsetter at objektene oppfører seg «pent» og ikke «snoker i andres interne data»)
- Kontrakt:
  Et krav om at objekter oppfører seg (kommuniserer og samarbeider) i henhold til avtalte regler

Målet for OO-datamodellering: 1:1-modell av interesseområdet



OO-modellering og objektdatabaser

- Sammenlignet med tradisjonelle datamodeller skal en OO-modell gi
  - Høy modularitet gjennom meningsfylte abstraksjoner
  - Bedre kontroll med kompleksitet
  - Klar avgrensning mellom klassen som abstrakt datatype (ADT) og dens implementasjon
- OO-modellering skal fremme evolusjonær systemdesign med inkrementell programmering og gjenbruk
- En objektdatabase (ODB) er en persistent samling av objekter
- Et ODB-objekt har formatet <OID, verdi, {operasjoner}>
- Et OODBMS er et DBMS som understøtter ODBer

ODB-eksempel

- Eksempelklasse:

```
class  Konto  {
       integer        ktonr;
       real           saldo;
       REF Kunde      eier;
}
```

- Eksempelverdier:
  - **V_1 = tuppel (ktonr: 65536, saldo = 34567.50, eier = ^K_1)**
  - **V_2 = tuppel (ktonr: 17289, saldo = 24506.52, eier = ^K_2)**
  - **V_3 = tuppel (ktonr: 87654, saldo = 21451.07, eier = ^K_1)**

- Eksempelobjekter:
  - **O_1 = <•, V_1, •>**
  - **O_2 = <•, V_2, •>**
  - **O_3 = <•, V_3, •>**

Krav til OODBMSer i "The third manifesto" (1990)

Må ha

- OID (objektidentifikator)
- Komplekse og sammensatte objekter
- Brukerdefinerte typer
- Beregningskomplett språk
- Innkapsling
- Arv med type/klasse-hierarkier
- Polymorfi:
  overlasting, redefinisjoner, sen binding

... Alle er ortogonale egenskaper

Bør ha

- Objektversjonering
- Støtte for distribusjon
- Nye transaksjonsmekanismer
- Støtte for regelbaserte systemer (aktive og deduktive DBer)

... og mye mer

OID – I

- Objekter eksisterer uavhengig av sine (nåværende) verdier
  - Uansett hvordan verdiene i et objekt endres, er objektet det samme
  - Objekter identifiseres entydig av sin OID
  - Du får aldri feil objekt hvis objektet refereres via sin OID
- Begrepene ***identitet*** og ***likhet*** eksisterer begge.
  De betyr ikke det samme:
  - At to objekter er identiske, betyr at de er samme objekt, dvs. at de har samme OID
  - At to objekter er like, betyr at de tilhører samme klasse og har de samme verdiene
- En OID baseres aldri på foranderlige (mutable) data

OID – II

- En OID er (i praksis) alltid systemgenerert og -håndtert
- OIDer er unike (systemvidt, globalt og universelt)
  - GUID: Globally Unique ID (egenskap i MS-verden)
  - UUID: Universally Unique ID (egenskap i Unix-verden)
- En OID er uforanderlig (immutable) gjennom hele objektets liv, og den gjenbrukes ikke etter objektets død

- Noen eksempler på objektoperasjoner basert på OIDer:
  - Å sammenligne objekter for identitet, likhet o.l.
  - Å referere objekter
  - Å finne og hente objekter

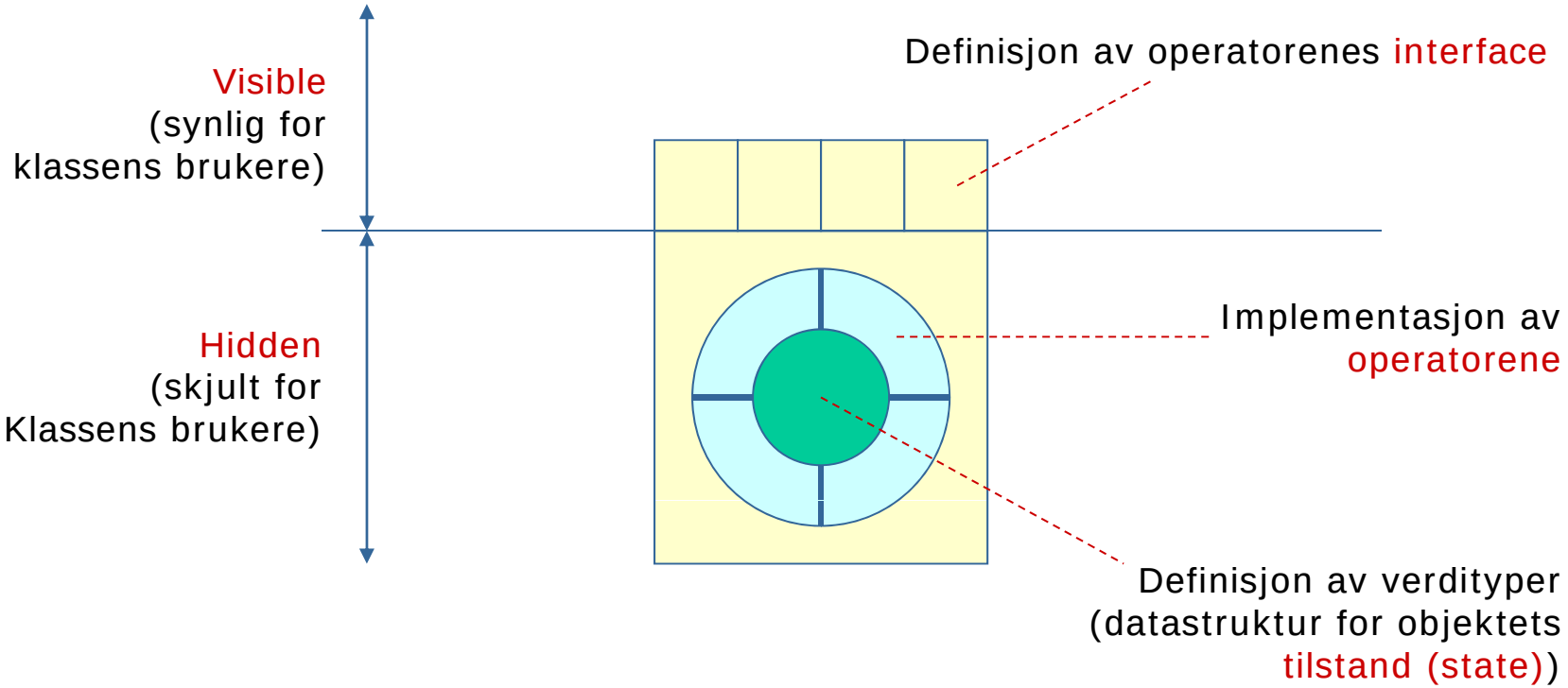
Komplekse og sammensatte objekter

- OO-paradigmet krever støtte for komplekse objekter
- Det er to typer kompleksitet som støttes
  - Ustrukturerte komplekse objekter
    Eksempler er tidsserieobjekter, fritekstobjekter og medieobjekter (lyd, bilde, video)
    Vi deler dem inn i to grupper:
    - BLOBs (Binary Large Objects)
    - CLOBs (Character Large Objects)
  - Strukturerte komplekse objekter
    Det samme som sammensatte objekter, objekter som inneholder andre objekter eller deler av andre objekter

Klasser og typer

- Det er ikke full internasjonal enighet om terminologien
- I Skandinavia (og i det meste av verden) bruker vi ordene slik:
  - En type, eller mer presist, en abstrakt datatype (ADT) er intensjonen til en klasse
  - En klasse er en implementasjon av en type (ADT)
  - Et objekt er en instansiering av en klasse (og ikke av en ADT)
- Brukerne kan lage sine egne klasser og typer
- Oppsummering:
  - Et objekt har en type og er instans av en klasse

Innkapsling



Arv og type- og klassehierarkier

- Vi har to former for arv vi må skille mellom:
  - Typearv hvor en ADT (subtypen) arver egenskaper fra en annen ADT (supertypen)
  - Klassearv hvor en klasse (subklassen) arver implementasjon fra en annen klasse (superklassen)
    - objekter i subklassen er objekter også i superklassen
- Vi skiller mellom
  - Enkel arv som leder til et type- eller klassehierarki
  - Multippel arv som leder til en typelattice (multippel arv for klasser fører til store problemer og er vanligvis ikke tillatt)

Polymorfi

Polymorfi har tre fasetter:

- Overlasting
  Bruk av samme navn på forskjellige operatorer
  (i forskjellige ADTer)
- Redefinisjon
  Reimplementasjon av operatorer lengre ned i klassehierarkiet
- Sen binding
  Binde et operatornavn til en spesifikk implementasjon dynamisk under “runtime”
  (gjøres individuelt for hvert objekt)

ODMG-standarden

- ODMG-standarden for objektdatabaser består av
  - En objektmodell som ligger så nær opp til OMG-standarden som mulig
  - ODL (Object Definition Language), et språk for å definere objektklasser
  - OIF (Object Interchange Format), et format for å skrive og lese objekter til og fra fil
  - OQL (Object Query Language), et SQL-inspirert spørrespråk
  - Språkbindinger for Smalltalk, C++ og Java
- Merk at standarden ikke har noe datamanipuleringsspråk
  - all oppdatering skal foregå via et vertsspråk
- Hovedhensikten med utvekslingsformatet OIF er å kunne utveksle objekter mellom ulike OODBMSer
- Gjeldende ODMG standard er ODMG 3.0 (2001)
  - der er ingen av språkbindingene fullstendig definert, dette var forventet som del av en ny versjon av standarden

- **Gitt standardens usikre fremtid, må den for øyeblikket betraktes som uaktuell/død**

Objektrelasjonelle DBMSer

- ORDBMS =
  Object-Relational Database Management System
- Motivasjon: Utvide relasjonsmodellen med objekt-orienterte ideer
  - utvidet typesystem
    - spesialiserte innebygde typer (fotos, lyd, signaler, ...)
    - brukerdefinerte typer (structer, mengder, bager, lister, ...)
  - med tilhørende spesialiserte metoder
- Relasjonen beholdes som fundamental abstraksjon
  - bakoverkompatibilitet med eksisterende relasjonelle DBMSer

Den objekt-relasjonelle modellen

- ***Tupler spiller rollen som objekter***
- Utvider den relasjonelle modellen med
  - strukturerte typer
    - bygget fra atomære typer ved typekonstruktører
    - konsekvens: fraviker første normalform
  - metoder
    - knyttet til tupler
  - identifikatorer
    - hvert tuppel har entydig identitet
  - referanser (pekere)
    - et attributt i et tuppel kan inneholde pekere til andre tupler

Typesystemet i den objekt-relasjonelle modellen

- **Atomære typer**: integer, real, string, ...,
- **Referansetyper**
- **Strukturerte typer**: bygget fra atomære typer og referansetyper ved typekonstruktører
  - bag, struct, set, ...
- **Skjema**: Navngitt samling av attributtnavn med tilhørende typer

- Et **attributt** kan ha som type en atomær type, en referansetype, en strukturert type eller et skjema
- En **relasjon** kan ha som type et skjema

Eksempel: Nestede relasjoner

Star(name, address(street, city), birthdate, movie(title, year, length))

Star

name	**address**		**birthdate**	**movie**		
Carrie Fisher	**street**	**city**	21.10.1956	**title**	**year**	**length**
	Maple	H'wood		Star Wars	1977	121
	Locust	Malibu		Empire	1980	124
Mark Hamill	**street**	**city**	25.09.1951	**title**	**year**	**length**
	Oak	B'wood		Star Wars	1977	121
				Return	1983	134

NB! Ikke-normalisert! (Bryter 1NF)

Eksempel: Referanser

Star(name, address(street, city), birthdate, movie({*Movie})

Movie(title, year, length)

Star

name	address		birthdate	movie
Carrie Fisher	**street**	**city**	21.10.1956	●
	Maple	H'wood		●
	Locust	Malibu		
Mark Hamill	**street**	**city**	25.09.1951	●
	Oak	B'wood		●

Movie

title	year	length
Star Wars	1977	121
Empire	1980	124
Return	1983	134

SQL:1999

- Objekt-orienterte ideer i SQL:1999
  - Brukerdefinerte typer
  - Metoder knyttet til brukerdefinerte typer
  - Referansetyper
  - Objektidentifikatorer
  - Nye operasjoner for objektrelasjonelle data
- SQL:2003 har ytterligere objekt-orienterte ideer

Brukerdefinerte typer

- **Klasser** realiseres gjennom **brukerdefinerte typer**
- En **UDT** (**User Defined Type**) kan brukes som
  - typen til et attributt
  - typen til en relasjon
- To former for UDTer:
  - brukerdefinerte **distinkte typer**
  - brukerdefinerte **strukturerte typer**

Eksempel

```
-- distinkt UDT
create type persnrtype as char(5);


-- strukturert UDT
create type førertype as
  (fdato date,
   persnr persnrtype,
   navn varchar(50));

-- relasjon hvor tuplene er definert ved en UDT
create table fører of førertype
 (primary key (fdato, persnr));
```

Merk:

- I relasjoner hvor tuplene er definert ved en UDT, spiller tuplene rollen som objekter – i de videre lysarkene bruker vi her betegnelsen **typede relasjoner** om slike relasjoner og **tuppelobjekter** om tuplene i dem
  - attributtene må aksesseres via systemgenererte observatormetoder
  - tuppelobjektene kan tilordnes entydige objektidentifikatorer og kan da aksesseres via referanser
- Metoder er knyttet til UDTer, ikke til typede relasjoner
- Kandidatnøkler og skranker på tuppelobjekter deklareres som del av de typede relasjonene, *ikke* som del av UDTen

Metoder

- Beskrivelsen av en metode er delt i to:
  - **Metodesignaturen** (metodedeklarasjonen)
    - plasseres sammen med UDTen som metoden tilhører
  - **Metodedefinisjonen** (kode) angis separat
    - skrives i SQL:1999 eller i et vertsspråk (PSM/Java/...)
    - likner stored procedures

Eksempel

```
create type førertype as
 (fdato date,
  persnr persnrtype,
  navn varchar(50))
method lovligfnr() returns boolean;

create method lovligfnr() for førertype returns boolean as
begin
  -- sjekker at (fdato, persnr) er et lovlig fødselsnummer
  ....  -- (kode som gjør dette)
end;
```

Objektidentifikatorer

- Typede relasjoner har i utgangspunktet én kolonne: Hver rad består av ett tuppelobjekt.
- I tillegg kan vi tilordne unike OIDer til tuppelobjektene
  - Slike typede relasjoner kalles **refererbare** (**referenceable**).
  - Generering av OIDer kan være brukerdefinert, systemdefinert eller ved avledning fra eksisterende attributter.

Eksempler

- Brukerdefinert OID

```
create type førertype as
  (...fdato...persnr...navn...)
   ref using integer
   method ...;
```

```
create table fører of førertype
   (ref is persid user generated);
```

- Avledet OID

```
create type førertype as
  (...fdato...persnr...navn...)
   ref from (fdato, persnr)
   method ...;
```

```
create table fører of førertype
   (ref is persid derived,
    primary key (fdato, persnr));
```

- Systemgenerert OID

```
create type førertype as
  (...fdato...persnr...navn...)
   ref is system generated
   method ...;
```

```
create table fører of førertype
   (ref is persid system generated);
```

Referansetyper

Referansetyper: `ref(T)` hvor `T` er en UDT

- Referansetyper tjener til å identifisere tuppelobjekter
- Referanser (attributter med referansetype) kan ha et **skop**, navnet på en relasjon definert ved `T`
  - Hvis referansen inngår i en UDT, kan skopet defineres i UDTen
  - Alternativt kan skopet defineres i en relasjon der referansen inngår
  - Relasjonen som angis i skopet, må være refererbar

Eksempel 1

```
create type kjøretøy as
  (regnr char(7), -- registreringsnummer
   vin char(17), -- vehicle identification number (chassisnummer)
   eier ref(førertype) scope fører) -- skop angitt i UDT
   ref is system generated
   method årsavgift() returns integer;
create method årsavgift() for kjøretøy returns integer as
begin
  -- Default årsavgift er 0 – ikke alle kjøretøy betaler årsavgift
  -- Vi kommer tilbake til spesialiseringer av kjøretøy og årsavgift
  return 0;
end;
create table register of kjøretøy
 (ref is kid system generated);
```

Eksempel 2

```
create type kjøretøy as
  (regnr char(7), -- registreringsnummer
   vin char(17), -- vehicle identification number (chassisnummer)
   eier ref(førertype))
   ref is system generated
   method årsavgift() returns integer;
create method årsavgift() for kjøretøy returns integer as
begin
  -- Default årsavgift er 0 – ikke alle kjøretøy betaler årsavgift
  -- Vi kommer tilbake til spesialiseringer av kjøretøy og åsavgift
  return 0;
end;
create table register of kjøretøy
 (ref is kid system generated,
  eier with options scope fører); -- skop angitt i en typet relasjon
```

Typehierarkier

Subtyper arver attributter og metoder fra supertypen:

```
create type a_kjøretøy under kjøretøy as
  (-- eventuelle tilleggsattributter kommer her)
overriding method årsavgift() returns integer; -- redefinisjon av metoden

create type f_kjøretøy under kjøretøy as
  (...)
overriding method årsavgift() returns integer;

create method årsavgift() for a_kjøretøy returns integer as
begin -- personbiler, lastebiler, busser, ...
   return 2790;
end;

create method årsavgift() for f_kjøretøy returns integer as
begin -- drosjer, ambulanser, ...
   return 395;
end;
```

Abstrakte typer

For noen strukturerte UDTer er det ikke aktuelt å lage instanser.
Disse kan deklareres som abstrakte ved **not instantiable**:

```
create type kjøretøy as
  (regnr char(7),
   vin char(17),
   eier ref(førertype) scope fører)
   not instantiable -- alle kjøretøyer må klassifiseres
   method årsavgift() returns integer;
create type a_kjøretøy under kjøretøy as
  (-- personbiler, lastebiler, busser, ...)
   instantiable -- kjøretøy som tilhører avgiftskategori a
   overriding method årsavgift() returns integer;
```

Nye operasjoner

- Generator- og mutatormetoder
- Aksessere komponentene i et tuppelobjekt
- Forfølge referanser
- Ordningsrelasjoner

Generator- og mutatormetoder

- Når en strukturert UDT ved navn T defineres, opprettes automatisk
  - en **generatormetode** T() som kan brukes til å opprette tuppelobjekter av type T
    T() returnerer et ”tomt” tuppelobjekt av type T
  - for hvert attributt x i T, en **mutatormetode** x(v) som kan brukes til å endre verdien i x til v
- Unntaket er hvis T er **not instantiable**, da opprettes ingen systemgenererte metoder
- Distinkte UDTer får en generatormetode på formen T(v) der v er en verdi fra det underliggende atomære domenet

Eksempel

```
create type kjøretøy as
  (regnr char(7),
   vin char(17),
   eier ref(førertype))
   method årsavgift() returns integer,
   method initkj(r char(7), v char(17)) returns null;

create method initkj(r char(7), v char(17))  for kjøretøy returns null as
begin
   self.regnr(r);
   self.vin(v);
end;

create table register  of kjøretøy (…);

insert into register values
(a_kjøretøy().initkj('DK23456', 'WVWZZZ1GZVW012561'));
```

Aksessere komponentene i tuppelobjekter

- Vi har ikke direkte aksess til attributtene i tuppelobjekter
- Attributtene kan aksesseres slik:
  - For hvert attributt `a` opprettes automatisk en **observatormetode** `a()`
  - Hvis `x` er en tuppelvariabel av type `T` og `a` er et attributt i `T`, er `x.a()` dets verdi i tuppelet `x`

Eksempel

```
select r.regnr()
from register r
where r.vin() like 'WVW%';
```

I praksis tillater DBMSer at parentesene sløyfes når tilhørende observatormetode ikke tar argumenter:

```
select r.regnr
from register r
where r.vin like 'WVW%';
```

```
create type kjøretøy as
  (regnr regnrtype,
   vin vintype
   eier ref(førertype)...) ...;

create table register of kjøretøy
  (...);
```

Forfølge referanser

- To måter:
  - x->a hvor x er en referanse:
    verdien til attributtet a i tuppelobjektet som x refererer
  - **select** deref(x) **from** ... **where** ...;
    hvor x er en referanse:
    deref(x) returnerer (alle attributtverdiene i)
    tuppelobjektet som x refererer
- Eksempel:

```
select r.eier->navn
from register r
where r.regnr = 'DK23456';
```

```
create type kjøretøy as
 (...
  eier ref(førertype)...) ...;
create table register of kjøretøy (...);
```

```
select deref(r.eier) -- dvs. fdato, persnr, navn
from register r
where r.regnr = 'DK23456';
```

Subtyper og sen binding

- Når vi har et typehierarki med redefinering av metoder, velges den metoden som stemmer best overens med tuppelobjektets subtype
- Dette krever at subtypetilhørighet og metodeinstanser bestemmes ved runtime (**dynamic dispatch**/**late binding**)
- I spørringer kan man restriktere til tuppelobjekter av en gitt subtype

Eksempel

```
create type kjøretøy as
 (...)
 method årsavgift() returns integer;
create type f_kjøretøy under kjøretøy as
 (...)
 overriding method årsavgift() returns integer;
create table register of kjøretøy
 (ref is kid system derived);
```

```
select sum(a) as totavgift
from (select r.årsavgift() as a
      from register r) as s;

select sum(a) as f_totavgift
from (select r.årsavgift() as a
      from register r
      where deref(r.kid) is of (f_kjøretøy)) as s;
```

Ordningsrelasjoner

- Tuppelobjekter kan ikke sammenliknes – med mindre vi forteller systemet hvordan dette skal gjøres:

```
create ordering for <UDT-navn>
 <ordering form>;
```

Hva slags sammenlikninger som kan defineres, er tildels DBMS-avhengig.

Eksempel 1: Likhet

To tuppelobjekter av type kjøretøy skal anses for å være like hvis attributtenes verdier er parvis like:

```
create type kjøretøy as
  (regnr char(7),
   vin char(17),
   eier ref(førertype) ...)
   ...;
```

```
create ordering for kjøretøy equals only by state;
```

(Dette definerer = og <>, men ikke <, >, <=, ...)

Eksempel 2: Total ordning

Leksikografisk ordning av fødselsnumre:

```
create ordering for førertype
order full by relative with fnrordning;
```

```
create type førertype as
  (fdato date,
   persnr persnrtype,
   navn varchar(50));
```

```
create function fnrordning
                    (x1 førertype, x2 førertype)  returns integer
if x1.fdato() < x2.dato() then return -1;
elseif x1.fdato() > x2.fdato() then return 1;
elseif x1.persnr() < x2.persnr() then return -1;
elseif x1.persnr() > x2.persnr() then return 1;
else return 0;
end if;
```

Eksempel på bruk

```
create type førertype as
  (fdato date,
   persnr persnrtype,
   navn varchar(50));
create type kjøretøy as
  (regnr char(7),
   vin char(17),
   eier ref(førertype)) ...;
create type f_kjøretøy under kjøretøy as (...) ...;
create table register of kjøretøy
  (ref is kid system derived);
```

```
select r.regnr()
from register r
where deref(r.kid) is of (f_kjøretøy) and
        r.eier() < førertype().fdato('1920-01-01').persnr('00000');
```

PostgreSQL 8.4 og SQL:1999

- Brukerdefinerte typer:
  - distinkte: via brukerdefinerte domener (**create domain** ...)
  - strukturerte: ja ('composite types', **create type** .. **as** ..)
  - metoder: nei, bare vanlige brukerdefinerte funksjoner (ála stored procedures)
  - typehierarkier: nei – men se relasjonsarv lenger ned
- Typede relasjoner:
  - implisitt; hver relasjon får automatisk en tilhørende strukturert type
  - oid: kan deklareres, men anbefales ikke brukt
  - relasjonsarv: multippel arv, arver attributter fra foreldrerelasjonene
- Aksessere komponentene i tuppelobjekter:
  - referansetyper: nei
  - komponenter i tuppelobjekter aksesseres ved dotnotasjon og attributtnavn
- Polymorfi:
  - overlasting av brukerdefinerte funksjoner; datatyper og funksjonsinstans bestemmes ved sen binding. Gir ikke samme fleksibilitet som typehierarkier og overlastede metoder
- **Syntaksen avviker fra SQL:1999-standarden**

Oracle9i og SQL:1999

- Støtter strukturerte UDTer, men ikke distinkte
- Man kan definere typehierarkier og redefinere metoder
- Man kan definere typede relasjoner
- Metodeinstans velges ved sen binding

- **Syntaksen avviker fra SQL:1999-standarden**