Innehållsförteckning

1. Inledning

Denna rapport ska ses som en fortsättning på rapporten ”Objektorienterade databas-hanterare – en introduktion”, SISU Publikation 96:05. Dess avsnitt 1 – 3.2 fungerar som en gemensam bakgrundstext till båda rapporterna och förutsätts av den anledningen vara kända av läsaren till föreliggande rapport.

Genomgående för hybrid-produkter är att de hämtar sina impulser från konventionell datamodellering samtidigt som såväl leverantörer som användare (sent omsider) har noterat att relationsmodellens begränsade semantiska uttryckskraft är hämmande för allt fler moderna tillämpningsområden. Moderna databashanterare (dbms) måste kunna stödja en semantiskt rikare modell. Vi är modellmässigt tillbaka i de egenskaper som kännetecknar de olika varianter av så kallade data- eller Entity/Relationship-modeller (ER-modeller) som togs fram under 70- och 80-talen. Hade marknaden vid den tiden givit då befintliga ER-modellbaserade dbms chansen att existera och vidareutvecklas, skulle sannolikt aldrig behovet av så kallade hybrider uppstått. ER-dbms kritiserades framför-
allt för att inte ha relationsmodellens formella stringens och med ”lösligt påhittade” gränssnittsspråk till skillnad från den formellt definierbara relationsalgebran. Att de allra flesta under systemutvecklingens analys- och designfaser föredrog att uttrycka sig i ER-modellens form och språk har alltså först nyligen givit några resultat på databasmarknaden. Att det dessutom mycket väl går att definiera stringenta, lättanvända språk (exv som påbyggnader på SQL) vågar först nu marknaden lita på. Anledningen till det sena intresset har även sin grund i databasforskningens förkärlek för teori framför reell användbarhet.

Den markanta skillnaden mellan tidigare ER-dbms och senare tiders hybrider är att de senare valt att basera sitt gränssnittsspråk på SQL-syntaxen, medan de tidigare ansatserna innehöll olika språkvarianter formulerade mer med tanke på modellens egenskaper än på kompatibilitet med det marknadsledande SQL-gränssnittet.

Nåväl, med dessa inledande, något syrliga reflexioner på pränt, är det dags att ta itu med databastrenden för dagen, nämligen de så kallade hybriderna.

Avsnitt 2 placerar in hybriderna på ”databashanterar-kartan” samt tar upp något om aktuella strävanden inom området. Avsnitt 3 diskuterar utförligt de typiska egenskaperna hos en hybrid-modell i anslutning till ett antal exempel. På sätt och vis är detta rapportens centrala avsnitt. Ett par olika standardiseringsaktiviteter beskrivs kortfattat i avsnitt 4.

Hybrider är en relativt ny företeelse. Antalet produkter på marknaden är följdaktligen begränsat. Även om syftet med dem överensstämmer, uppvisar de egna profiler beträf-fande egenskaper och uppbyggnad beroende på bakgrund, målgrupp, m m. Typiska generella egenskaper och realiseringsprinciper diskuteras i avsnitt 5, följt av en kortare produktgenomgång i avsnitt 6. Funderingar och spekulationer kring hybridernas fortsatta ”liv” ges i avsnitt 7. Avsnitt 8, till sist, ger en avrundande sammanfattning.

2. Hybrid-databashanterare, en inledande överblick

2.1 Olika typer av databashanterare

Rapporten SISU Publikation 96:05 beskriver översiktligt databasområdets utveckling genom åren. Hierarkiska och nätverksdatabaser tycks permanent placerade i historie-kolumnen. I dagsläget fokuseras istället intresset främst mot relationsdatabashanterare (rdbms), objektdatabashanterare (odbms) och det mellanting som brukar gå under beteckningen hybriddatabashanterare eller objektrelationsdatabashanterare (ordbms). Olika skisser används för att förklara deras olika karakteristiska egenskaper, deras förtjänster och brister. Givetvis ser dessa skisser olika ut beroende på vilken kategori framställaren själv representerar och därmed vad denne önskar framhäva. Givetvis måste dessa grova skisser också tas ”med en nypa salt” eftersom de ofrånkomligen blir onyanserade. Dock ger de ett tankemässigt stöd.

En av förgrundsfigurerna inom rdbms, Dr. Michael Stonebraker, brukar använda sig av en kvadrant enligt figur 1 för att förklara varför den produkt han numer representerar (en ordbms) hamnar i den ”bästa” rutan.

	Simple Data	Complex Data
Query	Relational DBMS (rdbms)	Object-Relational DBMS (ordbms)
No Query	File System	Object-Oriented DBMS (odbms)

Figur 1

Enkla tillämpningar, typ ordbehandling, kalkylprogram, m m arbetar med enkla filer och
i enlighet med egen, fördefinierad funktionalitet. Inte sällan är det fråga om enanvändarsystem. De hamnar i den nedre vänstra kvadranten.

Tillkommer sedan mer eller mindre avancerad fleranvändarmiljö och behov av ett enhetligt gränssnitt i form av ett frågespråk mot datamassan hamnar vi i den övre vänstra kvadranten, under förutsättning att datastruktur och datatyper inte är alltför komplexa. Typiska tillämpningar är s k administrativa databassystem, affärsdatabassystem. Modelleringsbegrepp och frågespråk är underförstått baserade på SQL i version SQL-89 eller senare. Förutom frågegränssnitt förutsätts olika typer av tillämpningsstöd, i form av formulärhanterare, 4GL-verktyg, m m finnas tillgängligt som komplement. Tillämpningens och databashanterarens adressområden förutsätts åtskilda, både av ”ideologiska” och av säkerhetsskäl. Realisering sker ofta i form av

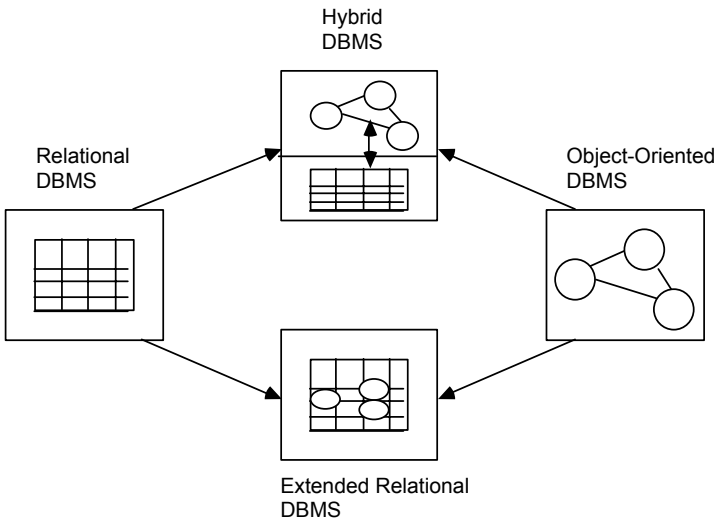
klient/server-arkitekturer. Alla konventionella rdbms hamnar i denna ruta. Marknaden är för närvarande stor och i växande.

Den nedre högra kvadranten representerar de egenskaper som normalt förknippas med odbms. De har en stark integrering med ett objektorienterat språk inom vilket all funktionalitet uttrycks. I denna version av objektmodellen utgör operationer en integrerad del av objekttypsbeskrivningen. Behovet av ett separat frågespråk har först på senare tid uppmärksammats. Vissa anser att separata frågespråk rimmar illa med objektmodellens filosofi medan andra anser att de utgör en naturlig komplettering. Kännetecknande för rutan är också en relativt komplex objektstruktur och/eller komplexa datatyper med tillhörande faciliteter i stödsystemet för operationer på dem.

Den sista kvadranten har den komplexa objektstrukturen gemensam med odbms-rutan. Skillnaden ligger i det klara gränssnittet mellan objekt och operationer på objekt. I gränslinjen finns frågespråket. Grundidén är konventionell och i stort överensstämmande med relationsmodellens syfte. Skillnaden ligger i modellens uttryckskraft.

I dagsläget är marknadsförhållande mellan rdbms och odbms grovt 100:1. Ordbms har lägre volym än odbms. Stonebraker menar att ordbms-rutan både svarar mot de behov dagens rdbms-tillämpningar har och mot det stöd framtida dbms-tillämpningar kommer att kräva. Med detta ”två flugor i en smäll”-resonemang spår Stonebraker, inte förvånande, en våldsam expansion av den övre, högra rutan under nästa årtionde. Mer om detta i avsnitt 6.

En annan förekommande dbms-indelning visas i figur 2. Här är ordbms uppdelad i Hybrid-dbms och Extended-Relational-dbms. Den förra betecknar de varianter som helt bygger på en rdbms ”i botten” men kompletterad med ett objektorienterat gränssnitt mot tillämpningen. Avbildning mellan den externa objektmodellen och den interna relationsmodellen sker i allmänhet ”osynligt” men kan i vissa fall påverkas. Extended-Relational står istället för de ansatser där man helt enkelt utökat dbms-kapaciteten till att kunna hantera en relationsmodell som byggts på med vissa objektmodell-egenskaper. Distinktionen är något svävande, med ett implementeringsorienterat perspektiv.



Figur 2

Nya trender löper sällan i entydiga spår, men går trots detta ofta under något generellt begrepp eller arbetsnamn. Oklarheter och snabba kast gör att begreppet blir ett ”buzzword” att slänga sig med. Hybrid-begreppet är inget undantag därvidlag.

2.2 Strävanden

Ett sätt att få grepp om företeelsen *hybrid*, kan vara att betrakta dess upprinnelse och den korta historia som genomlupits. Figur 3 är ett försök till en kompakt historiebeskrivning.

Figur 3

Tre olika databas-trender kan skönjas inom ”spelplanen”, var och en med sin bakgrund. Odbms hämtar sin idébas från de objektorienterade programmeringsspråken (OO-språk) och dess objektmodell. Rdbms har sin givna bas i relationsmodellen och i behovet av ett databasgränssnitt. Datamodellbaserade dbms bygger på modeller utvecklade för datamodellering.

Observera, att det fortfarande råder en total språkförbistring i samband med användning av ordet ”modell”. Man menar allt från en reell avbildning av någon företeelse i någon skala (verklighetsmodell), över verklighetsabstraktioner i form av en beskrivningsmodell eller schema (verksamhetsmodell) till de begrepp och regler som i någon ansats gäller för att uttrycka verksamhetsmodeller (begreppsmodeller). För den fortsatta diskussionen i denna rapport gäller, om inte annat sägs, den senare betydelsen. När exv relationsmodellen kommer på tal är det begrepp som tabell, kolumn, domän, nyckel, m m som avses. När semantisk datamodell diskuteras är det begrepp som entitetstyp eller objekt-
typ, attributtyp, datatyp, sambandstyp, m m som avses. Som ”lök på laxen” finns ett otal synonymer till begreppet ”semantisk datamodell”. Dit hör ”konceptuell modell”, ”semantisk modell”, ”Entity-Relationship-modell”, ”objektmodell”, ”datamodell”, ”hybridmodell”, m fl. I fortsättningen kommer S-modell att användas för denna typ

av modell. Relationsmodell kommer följdriktigt att kallas R-modell och OO-områdets objektmodell för O-modell.

Åter till figuren. Låt oss börja underst. Relationsdatabasområdet har hunnit genomlöpa ett antal utvecklingsfaser. Som bas har hela tiden legat R-modellens begrepp och villkor, om än i olika nyanser. 1992 antogs den senaste versionen av gränssnittsstandard, i folkmun kallad SQL92. De flesta rdbms-leverantörer strävar efter kompatibilitet med denna standard.

Behoven inom datamodellering har drivit fram olika varianter av S-modeller. Ur vissa har dessutom några dbms-produkter spirat. I brist på standarder blev här gränssnittsspråken olika från produkt till produkt (Egna).

Odbms har sitt ursprung inom OO-programmering och den där förhärskande O-modellen. I tidiga releaser fanns sällan något gränssnittsspråk. Där det fanns var det produktspecifikt.

Relationsfalangen konstaterade redan tidigt bristerna i R-modellen. 1990 påbörjades en standardiseringsaktivitet inom det organ som drev övrig SQL-standardisering (ISO/IEC JTC1/SC21/WG3) med syftet att utvidga modellen med mer semantisk kraft, d v s med mer av det som fanns i vanliga S-modeller. Arbetet kom att gå under arbetsnamnet SQL3. Intensivt arbete genererade snabbt de första förslagen till en utökad syntax baserad på en utvidgad R-modell (i fortsättningen kallad OR-modell). Denna syntax har därefter befun-nit sig under kontinuerlig utveckling och revidering. Arbetet fortgår med oförminskad kraft.

Även om mycket arbete fortfarande återstår till en accepterad standard är SQL3-syntaxen
i dess olika varianter en inspirationskälla för etablerade rdbms-leverantörer i samband med egna SQL92-påbyggnader. Framförallt har den varit ett rättesnöre för de nyutvecklade produkter, som etablerat sig på marknaden fr o m 1992, med det ursprungliga syftet att vara renodlade hybrider. Därav rutan ”SQL3-inspir” från ”SQL92”. Samtidigt fortgår ansträngningar på olika typer av standardisering kring SQL92-nära aspekter (den vågräta pilen).

Odbms har unika egenskaper som gör dem synnerligen lämpliga för vissa typer av tillämpningar. Samtidigt kan konstateras att denna marknad för närvarande varken är speciellt stor eller expansiv. Dessutom visar odbms-produkterna en splittrad bild. Mycket riskkapital har plöjts ner i produkterna. Investerare önskar säkert snart valuta på investeringen. Följden blir en ofrånkomlig press på anpassning till den enorma rdbms-marknaden. Dit hör att realisera ett programmeringsspråkoberoende frågespråk och därmed ett avsteg från den förhärskande integreringen av språk – databas. Ett närmande av O-modell till S-modell-hållet har vuxit fram. 1993 presenterade Object Database Management Group (ODMG), en grupp bestående av alla de större odbms-leverantörerna, en gränssnittsspecifikation under beteckningen ODMG-93. Den innehåller bl a speci-fikation av ett rent frågespråk (d v s utan uppdateringar) med beteckningen Object Query Language (OQL) baserad på en O-modell med inslag av S-modell-anpassningar. OQL finns redan i några odbms-produkter (bl a O2 varifrån specifikationen ursprungligen kommer). Övriga leverantörer har givit utfästelse om snar realisering i sina produkter. Parallellt förfinas och kompletteras den renodlade odbms-ansatsen (den vågräta pilen) med bäring på ”sitt” marknadssegment.

Trenden är helt klar. Såväl OQL, som SQL3, och datamodellfalangerna strävar mot någon form av S-modell med därtill hörande gränssnittsspråk (de gråa pilarna). Modell-olikheterna är numer små. Fortfarande ser OQL och SQL3 ganska olika ut. Datamodell-falangen har inget starkt eget alternativ. Förhoppningarna knyts till det nyligen etablerade samarbetet mellan ODMG och de ISO- och ANSI-organ som företräder SQL3-ansträngningarna i syfte att nå fram till ett enda kompromissförslag (OQL/SQL3). Pressen finns, även viljan. En gemensam specifikation före år 2000 är en inte alltför orealistisk gissning. Dock kommer sannolikt ambitionen att vara ett gränssnittsspråk, snarare än ett fullödigt programmeringsspråk, vilket SQL3 just nu representerar. Mer om standarder i avsnitt 4.

I ett subjektivt försök att finna nyanser i betydelsen av begreppen Extended Relational och Hybrid placeras det senare som någon form av ”modellblandning” mellan O-modell och R-modell, d v s en datamodell av S-modellsnitt med tillhörande gränssnittsspråk. Extended Relational tolkas bokstavligt som trenden från R-modellen, d v s den nedre halvan av hybridernas täckningsområde (figur 3). Förmodligen är det inte meningsfullt att ägna någon större möda på en exakt distinktion i dagsläget. I fortsättningen kallar vi helt kort Extended Relational modellen såväl som Hybrid-modellen för OR-modell.

Till sist; figur 3 redovisar en idéutvecklingssträvan som inte ska sammankopplas med marknadsvolym och marknadsutveckling. Rdms spelar och kommer under överskådlig tid att fortsätta spela en dominerande marknadsroll.

Efter denna svepande inledning är det dags att mer konkret diskutera OR-modellens typiska egenskaper.

3. Hybrid-modellen

3.1 Inledning

Som förhoppningsvis tydligt framgått av föregående avsnitt finns inte en standardiserad OR-modell. Där finns flera ansatser som dessutom modifieras kontinuerligt. Att redovisa alla varianter är inte meningsfullt. Eftersom SQL3-arbetet just nu har ett dominerande inflytande redovisas i detta avsnitt OR-modellen utifrån SQL3-perspektivet. De viktigaste utvidgningarna av R-modellen redovisas stegvis inklusive exempel. Varje steg åskådliggörs genom en motsvarande förutsättning uttryckt i en grafiskt presenterad S-modell. Om vi bortser från att namngivningen är olika och att den ena ritas grafiskt, är den bakomliggande innebörden av modelleringsbegreppen hos OR- och S-modeller uppenbart överensstämmande.

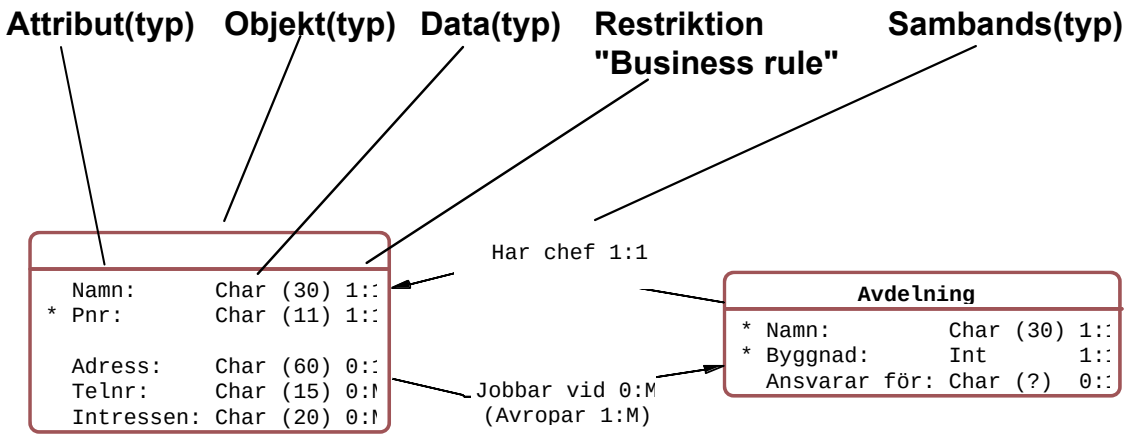
För OR-modell-exemplen används produkten Illustras gränssnittssyntax med vissa mindre modifikationer och förenklingar för att hålla resonemanget rent från onödiga detaljer. Illustras syntax är mycket SQL3-nära och förhoppningsvis något mer stabil än SQL3, genom sin produktkoppling. Dessutom är den behagligt begränsad till att vara ett gränssnittsspråk i samma tradition som SQL92 och inte ett komplett programmerings-språk, som SQL3.

Observera, att den exakta syntaxen i sig är av underordnat intresse för denna rapport. Syftet är endast att spegla hur man idémässigt valt att realisera modellutvidgningar i gränssnittsspråket. För att undvika alltför påtaglig koppling till standard eller produkt används fortsättningsvis det mer neutrala begreppet ”OR-språk”, ”OR-modell”, o s v.

Stabilitet är knappast det mest påtagliga kännemärket för hybrid-området. Följaktligen förekommer även bland OR-modeller olika begrepp för mycket snarlika företeelser. Exempelvis är distinktionen mellan Table, Type, Class, Objectclass, Objecttype inte kristallklar. SQL3 gör dock en distinktion mellan table och type men på ett sätt som föranleder en hel del debatt för närvarande. Vi bortser för enkelhets skull från denna problemställning just nu genom att konsekvent använda begreppet Table även där tanken är att Type ska användas, för att senare återkomma till distinktionen i avsnitt 3.6.

3.2 Flervärda kolumner

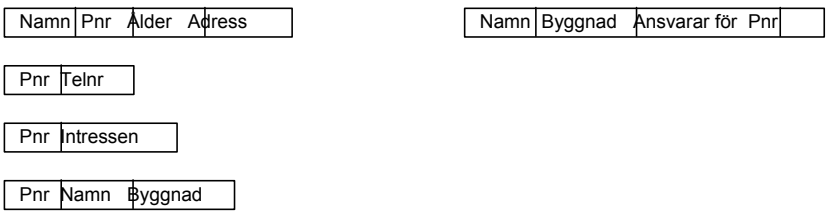
En grafiskt ritad S-modell visas i figur 4. Modellen är framtagen med SISU-produkten Business Modeler. Vanligen använda begrepp i S-modeller anges i överkanten. Där finns två objekttyper, vardera beskriven med ett antal attributtyper. Reglerna för hur attribut av en viss typ får uttryckas anges genom referens till viss datatyp. Olika typer av relateringar mellan objekttyper kan vara av intresse. Dessa kallas sambandstyper. I denna S-modell är sambandstyperna dubbelriktade och binära. Andra S-modeller tillåter andra varianter. Pri-märnyckel anges med asterisker för de attributtyper och/eller samband som tillsammans utgör en unik referens.



Figur 4

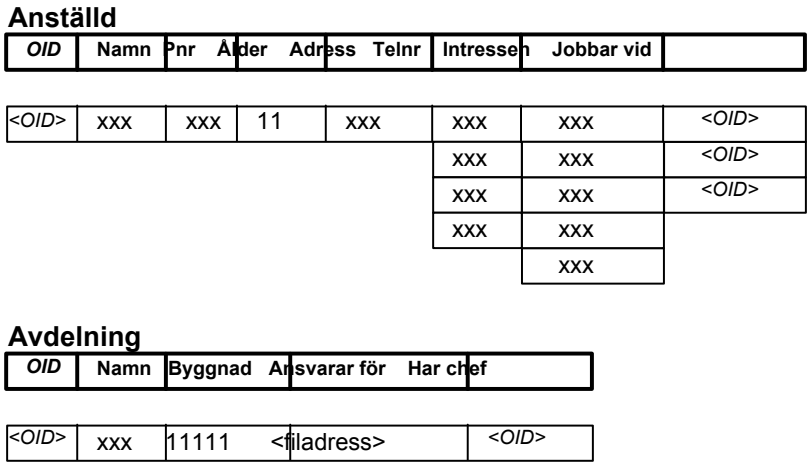
Modellen är enkel. Anställda beskrivs med några uppgifter av intresse. En avdelning är en enhet inom företaget med viss given roll. Denna roll beskrivs under *Ansvarar_för*. Framförallt definieras här respektive avdelnings arbetsuppgifter, ansvar och befogenheter. Beskrivningen är ett rätt omfattande textdokument. Anställda jobbar vid en eller flera avdelningar. Det kan gälla insatser under kortare eller längre perioder. Vid varje tidpunkt kan en avdelning ha en eller flera anställda avropade för något ändamål. Varje avdelning har dessutom en fast chef.

Ska denna modell avbildas till en R-modell enligt 3:e normalformen måste bl a hänsyn tas till eventuella flervärdiga attributtyper och sambandstyper. Dessa måste brytas ut och etableras i separata tabeller tillsammans med sin primärnyckel. I exemplet gäller detta för attributen *Telnr* och *Intressen* samt sambandstypen *Jobbar_vid (Avropar)*. Tabellerna blir enligt figur 5.



Figur 5

Denna tabellstruktur upplever säkert alla som artificiell. De fyra tabellerna till vänster beskriver ju samma sak, nämligen anställda. Samma logiska gruppering som i S-modellen borde kunna accepteras utan att de konsistensrisker som lett fram till 3:e normalformen ska behöva inträffa. OR-modellen hanterar flervärdiga attribut samt flervärdiga enkelriktade samband på sätt som åskådliggörs i figur 6.



Figur 6

En nyhet, förutom flervärdiga kolumner, är att varje rad i en tabell identifieras med en unik identifierare, en Object Identifier (OID). En lustighet i sammanhanget är att samtidigt som man fortfarande talar om tabeller, kolumner och rader i modellen, smyger sig objekt-begreppet in när identifieraren ska ges namn. Förmodligen anses begreppet rad ha blivit något diffusare i och med flervärdigheten. I figur 6 ligger det ju närmare till hands att tolka den anställde med sin gruppering av attribut för ett objekt än som en rad. I alla händelser har OID samma funktion här som den standardmässigt har i ODBMS, nämligen att utgöra objektets unika representation under objektets hela livstid. OID har ingen inneboende semantik. I normalfallet kommer den heller inte att ses av användare. (Vissa undantag finns dock.) I Illustra är en OID uppbyggd av 64 bitar, varav de första 24 bitarna identifierar en viss tabell och de resterande 40 bitarna viss rad inom tabellen.

Som synes under *Anställd*s kolumn *Jobbar_vid* sker nu referenser till andra objekt (rader) med hjälp av OIDs, inte som i tidigare SQL-generationer med hjälp av en främmande nyckel bestående av en eller flera kolumnvärden (i exemplet genom *Namn* och *Byggnad*
i kombination).

Ansvarar_för är ett digert textdokument som lämpligen lagras separerat från tabellstrukturen av utrymmes- och prestandaskäl. Datatypen *external_file* är till för detta ändamål. Istället för hela texten lagras endast det interna namnet på filen i kolumnpositionen för *Ansvarar_för*.

Deklaration av modellen i OR-syntax blir som följer:

```
create table Anställd (
    Namn            char (30),
    Pnr             char (11),
    Ålder           int,
    Adress          char (60),
    Telnr           setof (char (15)),
    Intressen       setof (char (20)),
    Jobbar_vid      setof (ref(Avdelning)) );
```

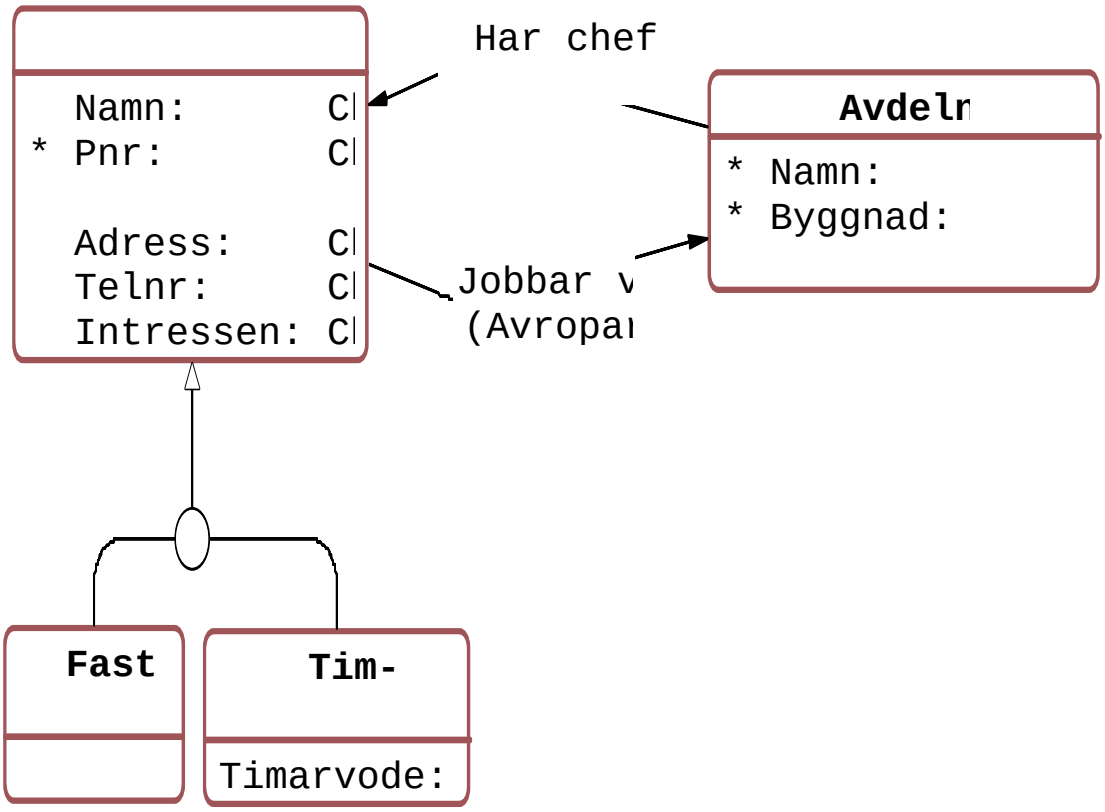
```
create table Avdelning (
    Namn            char (30),
    Byggnad         int,
    Ansvarar_för    external_file,
    Har_chef        ref(Anställd) );
```

(En helt korrekt deklaration skulle först ha etablerat Anställd och Avdelning som **type** och därefter etablerat motsvarande **table** med referens till aktuell **type**. Vi bortser, för enkelhets skull, tills vidare från distinktion mellan **table** och **type**. Se vidare avsnitt 3.6.)

Den enda egentliga kvarvarande skillnaden mot S-modellen så här långt är olika val av namn på modell-begreppen samt att sambandstyper i S-modellen är dubbelriktade, medan OR-modellen tillämpar enkelriktade sambandstyper i R-modellens anda. S-modellen ger (tills vidare) även något utförligare restriktionsangivelser eller ”business rules”.

3.3 Specialiseringsstrukturer och arvsmekanismer

Det visar sig att anställda är av två kategorier, *Fast Anställd* och *Timanställd*. För den första kategorin är *Lön* ett intressant attribut, för den senare kategorin attributet *Timarvode*. De båda kategorierna förs in i modellen som specialiseringar av *Anställd*. Se figur 7.



Figur 7

I OR-språket deklareras utvidgningen enligt nedan som två vanliga tabeller kompletterade med vilken tabell de är specialisering under, med hjälp av nyckelordet ”under”.

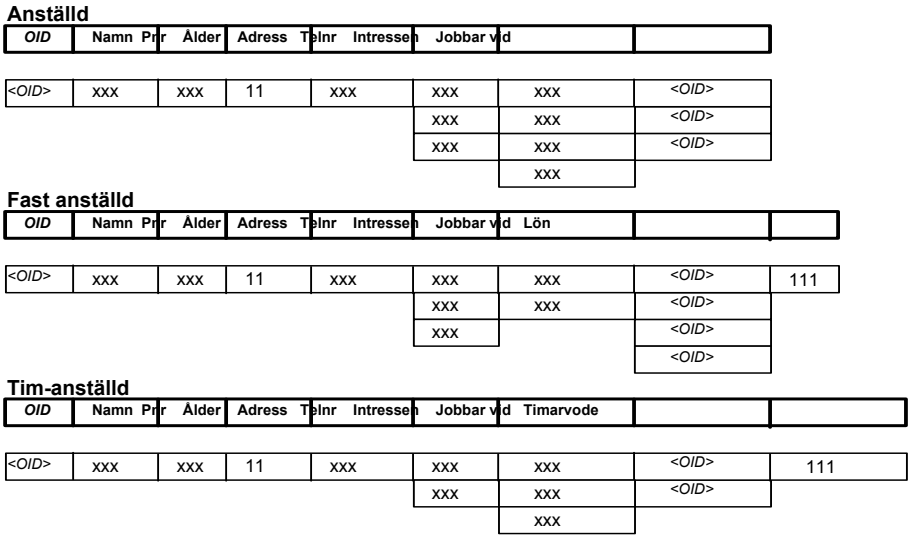
```
create table Fast_Anställd (
     Lön            int)
under Anställd;

create table Tim_Anställd (
     Timarvode      real)
under Anställd;
```

Anställd sägs vara supertabell för subtabellerna *Fast_Anställd* och *Tim_Anställd*.

Specialiseringsstrukturer inbjuder till etablering av automatiska arvsmekanismer. Eftersom vi genom deklarationen vet att en viss *Tim_Anställd* också är en *Anställd* har en viss *Tim_Anställd* både de egna attributen och indirekt de som finns under vederbörande som *Anställd*. Samma sak gäller för samband. En *Tim_Anställd* ärver samtliga attribut och samband från Anställd på så vis att vi exv kan referera till en viss *Tim_Anställd*s *Adress*. Samma resonemang kan appliceras på *Fast_Anställd*.

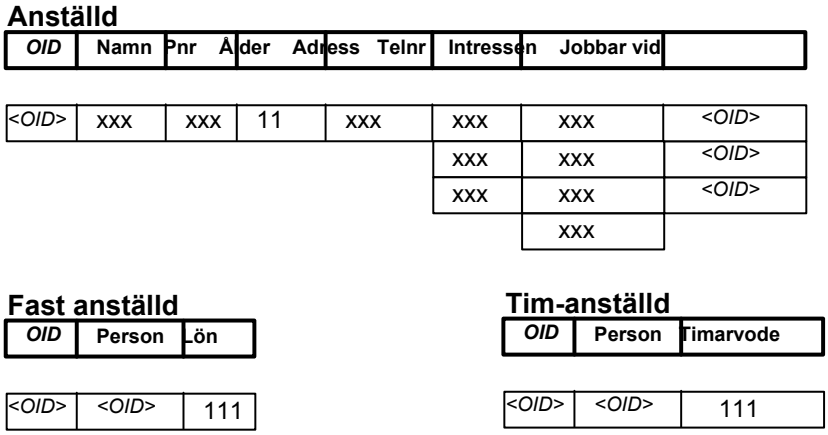
I det generella fallet kan specialiseringsstrukturen ha valfritt djup, d v s Table X kan vara subtabell till Table Y som kan vara subtabell till Table Z som kan vara ..... Arv sker från alla supernivåer, d v s X ärver både från Y och Z och ..... Dessutom kan en tabell i OR-modellen vara subtabell under flera supertabeller på närmast överordnade nivå. Detta leder till vad som vanligtvis kallas multipla arv, d v s att ärvandet löper efter flera grenar.

Sett genom gränssnittet **upplevs** en timanställds samtliga uppgifter finnas i tabellen *Tim_Anställd*. Samma sak gäller *Fast_Anställd*. Se figur 8. De som eventuellt inte tillhör någon av dessa två sub-kategorier, återfinns i tabellen *Anställd*.



Figur 8

För den fysiska lagringen kan ett ordbms välja att realisera detta förhållande på ett flertal olika sätt. Figur 8 ovan är ett sätt. En annan variant skulle kunna vara att lagra *Anställd*-attributen för en timanställd både i *Anställd* och i *Tim_Anställd* för att optimera för vissa typer av utsökningar. Priset får betalas genom tunga uppdateringar och stort sekundärminnesbehov. En annan variant är att internt helt enkelt se specialiseringssambandet som ett vanligt samband på så vis att en rad i en subtabell innehåller referenser till ”sin” rad i supertabellen (figur 9).



Figur 9

För utsökningar används så långt möjligt den ”gamla vanliga” Select-instruktionen. Exempel på utsökningar visas nedan.

a.
```
select Namn, Intressen, Telnr
from Anställd
        where Ålder > 50;
```

b.
```
select Namn, Jobbar_vid.Namn
from Anställd
        where .....;
```

c.
```
select Namn, Jobbar_vid.Har_chef, Lön
from Fast_Anställd
        where .....;
```

d.
```
select Namn, Jobbar_vid.Har_chef.Namn
from Tim_Anställd
        where .....;
```

e.
```
select Namn, Jobbar_vid.Har_chef.Lön
from Anställd
        where .....;
```

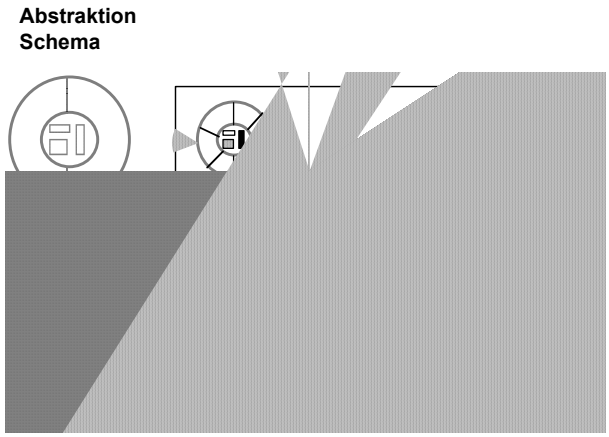
Utökad användning av punkt-notation för att referera till attribut kan noteras. Punktnota-tion kan användas för att ”navigera” sig fram i datastrukturen. Denna navigation kan i princip bestå av valfritt antal steg. Se exv tredje exemplets *Jobbar_vid.Har_chef.Namn* som med startpositionen vid någon anställd söker fram namnet på de anställda som är chefer för de avdelningar den anställde jobbar vid. På

sätt och vis är vi härmed tillbaka till den tidigare, hos relationsmodellföreträdare så förhatliga, hur-orienteringen från nätverksdatabaser. I själva verket är punktnotation knappast en hur-operation, snarare ett sätt att uttrycka en sökning baserad på semantiskt etablerad kunskap om verkligheten dokumenterad i schemat. Vilken skulle våga, eller ens kunna, ställa en SQL-fråga utan att känna till dess schema?

Den vaksamme har förmodligen upptäckt riskerna med e-instruktionen. Exemplet kan ge odefinierat resultat (felavbrott?) om det visar sig att någon av de anställda som svarar mot villkoret inte är en *Fast_Anställd* (där ju Lön finns definierat).

3.4 Beteenden – funktioner i modellen

I objektorienterade modeller (O-modeller) sägs objekt normalt ha beteenden, inte bara tillståndsuppgifter. Möjligheten att knyta operationer/beteenden till modellens komponenter är det som framförallt skiljer O-modeller från S-modeller. Det är lätt att få intrycket att varje objekt (förekomst) har ”sitt” beteende enligt figur 10.



Figur 10

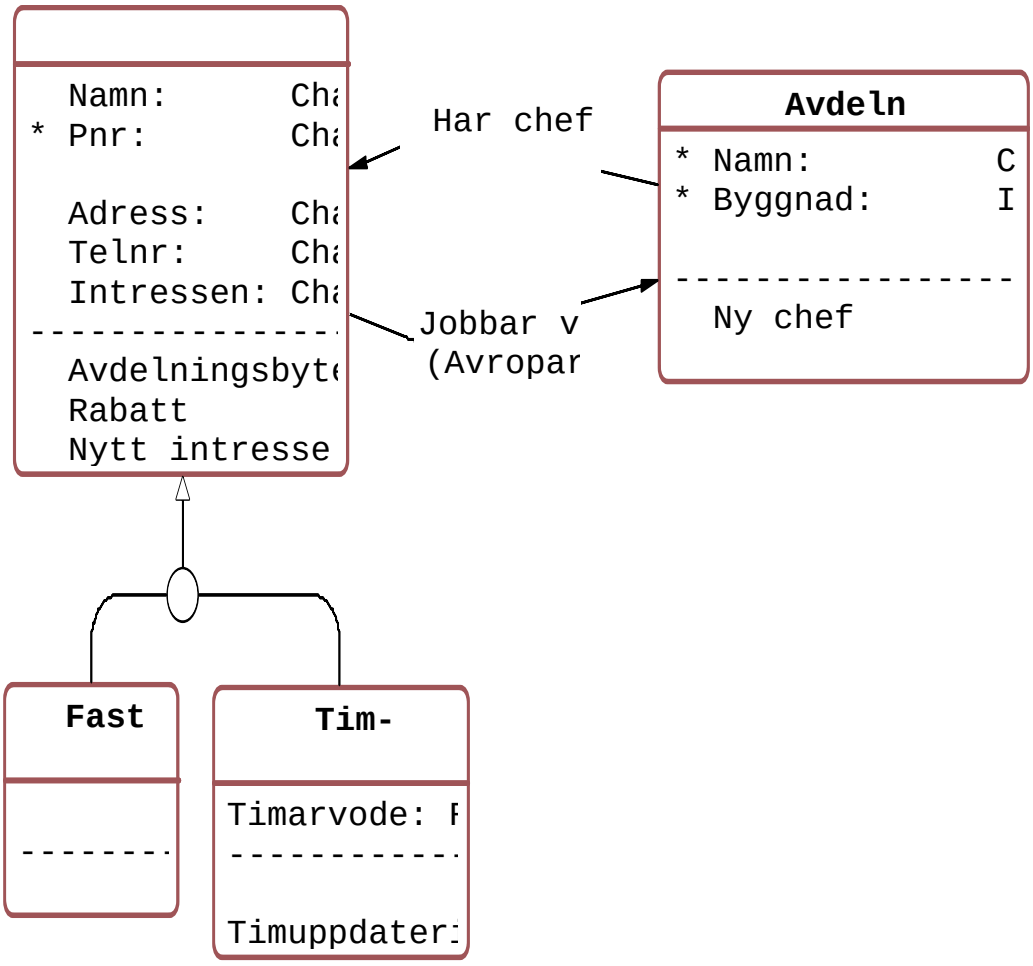
I själva verket definieras dessa beteenden per objekttyp men med syftet att appliceras på förekomster (objekt) tillhöriga objekttypen. I en odbms-databas ligger följdaktligen normalt endast tillståndsuppgifterna (den statiska delen) medan operationerna finns realiserade som en del av tillämpningen. Se figur 11.

Figur 11

I enlighet med O-modellens inkapslingsprincip kan en definierad objekttypsoperation endast direkt operera på de attribut som finns definierade inom objekttypen, dessutom endast på en förekomst (objekt) åt gången. Alla övriga uppgifter hämtas via meddelandeutväxling med andra objekt. Inom typiska OO-tillämpningar är dessa egenskaper alldeles tillräckliga och naturliga. I en mer konventionell databastillämpningsvärld gäller fortfarande behovet av att skilja på data och funktioner på data. Många olika funktioner kan behöva arbeta mot ett objekts attribut av många olika skäl. Till yttermera visso finns funktioner som har behov av att samtidigt operera på attribut från flera olika objekt tillhöriga samma eller olika objekttyper.

OR-modellen har tagit fasta på att beteenden – funktioner kan ha ett vidare existensberättigande än endast ”inom skalet” på en objekttyp. Samtidigt konstaterar man att funktioner många gånger har sin naturliga plats i modellen snarare än i tillämpningen. Dit hör olika typer av härledningar, villkorskontroller, m m. Renodlade funktioner, d v s sådana som resulterar i något värde, kan många gånger vara naturligt att anropa inom någon SQL-sats, inte minst inom Select-satser. Funktioner behöver normalt ingångsvärden (in-parametrar) som i så fall kan hämtas från övriga tillgängliga uppgifter i satsen.

OR-språk tillåter definition och exekvering av funktioner inom ramen för språkets instruktionsrepertoar. Den som så önskar kan anse att en funktion som kräver ett objekt tillhörig viss typ som in-parameter definitionsmässigt är ett beteende tillhörigt den objekttypen. Dessa funktioner redovisas i grafiska modeller standardmässigt längst ner i en objekttypsruta, avskilt från attributtyperna med en linje. Se figur 12. Dock ska inte OR-funktioner generellt sett förväxlas med O-modellers motsvarighet, dels eftersom de internt kan laborera med vilka uppgifter som helst i databasen, dels kan ha en uppsättning in-parametrar där varje parameter tillåts svara mot ett objekt, en objektmängd eller ett värde.



Figur 12

Som senare kommer att framgå erbjuder OR-modellen både definition av funktioner och procedurer. Dessa funktioner/procedurer kan ha noll eller flera in-parametrar. En funktion representerar alltid ett resulterande värde av viss typ medan en procedur är något som utförs (enkelt eller komplext) och som inte nödvändigtvis resulterar i ett värde.

Funktioner definieras i en egen Create-sats. Som exempel visas definition av funktionen *Rabatt* under *Anställd* och *Hur_många_avropade?* under *Avdelning*. Rabatten gäller vid köp av företagets egna produkter. Den är åldersbaserad och ger ett värde som ska användas för att multiplicera ursprungspriset med. Antalet avropade är en antalsuppgift på den mängd anställda som för närvarande jobbar vid avdelningen ifråga.

```
create function Rabatt (Anställd)
returns int
as select (Ålder • 0.5) / 100
     from Anställd
          where oid = $1.oid;

create function Hur_många_avropade? (Avdelning)
returns int
as select count (•)
     from Anställd
```

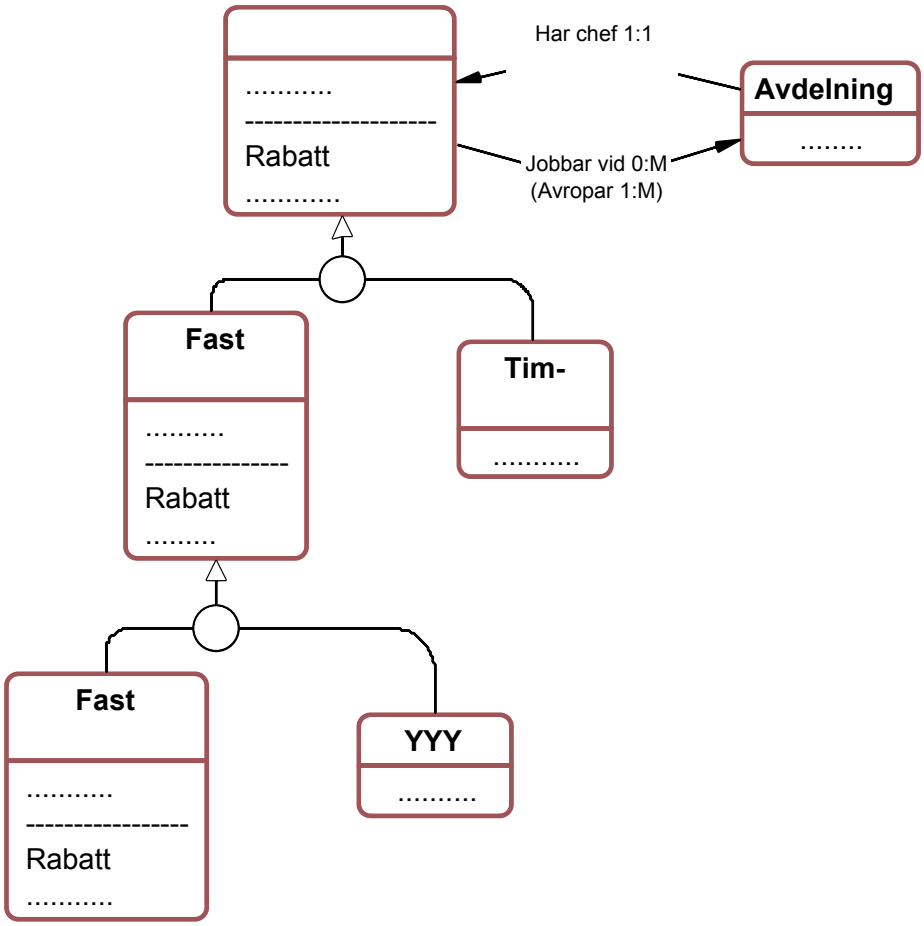
```
        where Jobbar_vid.oid = $1.oid;
```

$1 refererar till den första in-parametern. Om det funnes tre in-parametrar skulle *$3* referera till den tredje. Som synes kan, förutom de användardeklarerade attributtyperna, även tabellens oid refereras.

När väl funktionen är definierad kan den användas i någon sats. Select-satsen nedan tar fram namn och aktuell rabatt för alla de anställda som är chef över en avdelning som har fler än 20 jobbande (avropade).

```
select Namn, Rabatt (Anställd)
from Anställd
    where oid in
        (select oid.Har_chef
         from Avdelning
            where Hur_många_avropade? (oid) >20);
```

För varje funktion definieras typ på in-parametrar inom parentes efter funktionsnamnet och ut-parametrar efter ”returns”. Om en förekomst av *Anställd* förväntas som in-parameter, som är fallet vid *Rabatt*, är det ju rimligt att tillåta att förekomsten även kan vara en subtyp till *Anställd* (eftersom den som hör till en subtyp även har supertypens egenskaper). Att exv begära *Rabatt* för en *Tim_Anställd* är riskfritt. Om man så önskar kan man uppleva det som att inte bara attribut och samband, utan också funktioner ärvs i enlighet med specialiseringsstrukturen. Samtidigt kan förhållandet användas för att åstadkomma polymorfism över både attribut och funktioner (för definition, se SISU-rapport 13). Antag en struktur enligt figur 13.



Figur 13

Funktionen *Rabatt* finns deklarerad i tre olika varianter för *Anställd*, *Fast_Anställd* och *Fast_LO_anställd*, förmodligen för att olika rabattregler gäller beroende på kategoritillhörighet. De nya *Rabatt*-funktionerna deklareras:

```
create function Rabatt (Fast_LO_anställd)
returns int
as .....

create function Rabatt (Fast_anställd)
returns int
as .....
```

Begärs sedan

```
select Namn, Rabatt (oid)
from Fast_LO_anställd
      where .....
```

är det ingen tvekan om att *Rabatt*-funktionen för *Fast_LO_anställd* exekveras. Begärs istället

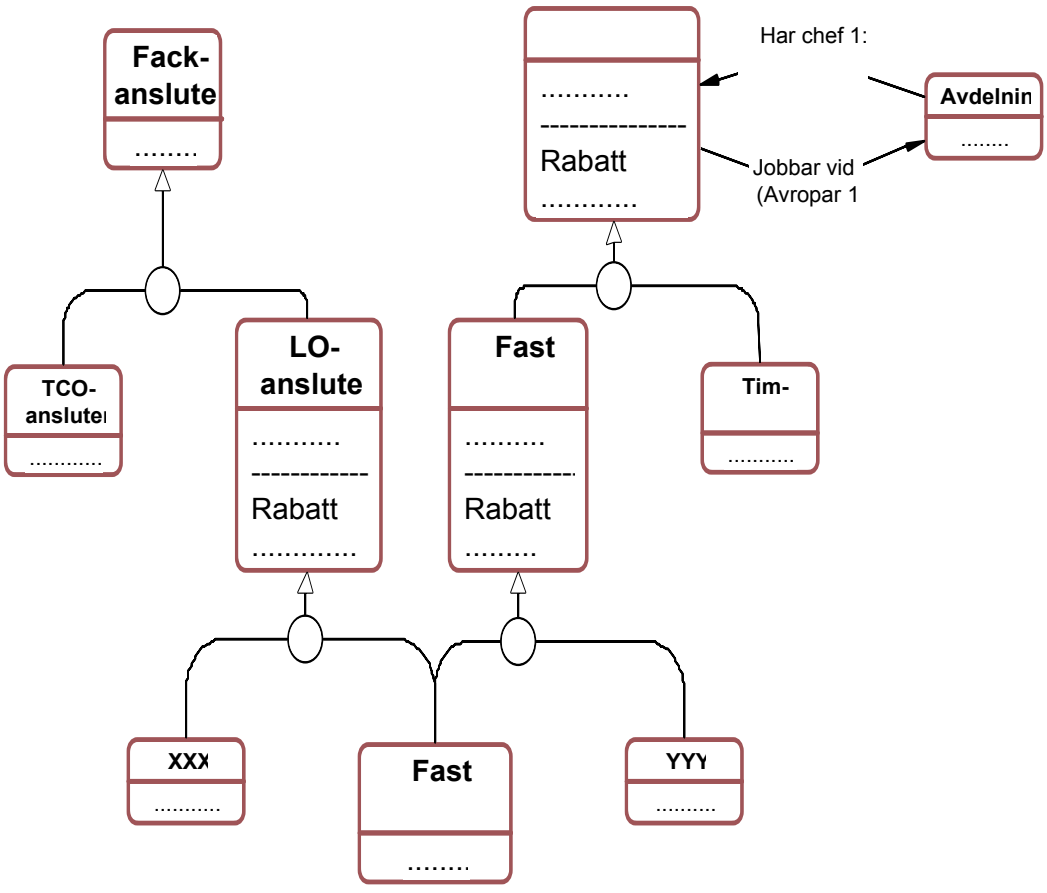
```
select Namn, Rabatt (oid)
from Anställd
```

where .....

och det visar sig att någon av de refererade mer exakt är *Fast_LO_anställd*, uppstår en oklarhet. Denne är både *Anställd*, *Fast_anställd* och *Fast_LO_anställd*. Samtliga tre *Rabatt*-funktioner är applicerbara. Enligt den polymorfistiska principen tas den version som mest exakt svarar mot den aktuella typen, d v s i detta fall *Rabatt* under *Fast_LO_anställd*. Är någon av de utsökta enbart *Fast_anställd* tas *Rabatt* under *Fast_anställd*. Är den anställde *Timanställd*, som saknar egen rabatt-regel, återstår endast *Rabatt* under *Anställd*.

(För att åstadkomma dessa effekter i Illustra krävs i realiteten att deklarationen av *Rabatt* för *Anställd* efter **returns int** kompletteras med **with (late)** för att ”late binding”, d v s polymorfistiskt beteende, ska effektueras.)

Med flexibilitet följer i allmänhet risker, bl a när en multipel arvssituation är för handen. Antag en förändring enligt figur 14. *Fast_LO_anställd* har inte längre en egen rabattregel. Efterfrågas *Rabatt* för en *Fast_LO_anställd* uppstår tvetydighet. *Rabatt* finns både i supertabellen *LO_ansluten* och *Fast_anställd*. Från vilken gren ska uppgiften hämtas? Här finns ingen given, vedertagen lösning. Olika principer används i olika produkter. En ordbms-användare och även OO-språksanvändare bör vara medveten om vilken princip som tillämpas i använd produkt, förutom givetvis att fundera över vad som för den givna tillämpningen är det önskvärda. Här kan inte grundprincipen om att hämta ”längst ner i hierarkin” tillämpas.



Figur 14

Inom parentes kan nämnas att de flesta modelleringsprinciper tillåter multipla arv. Det gör även C++. Däremot accepterar exv både Smalltalk och Java endast enkla arv. Båda ansatserna har alltså starka förespråkare.

Polymorfism är en central och mycket kraftfull mekanism i OO-språk. Samma mekanism används numer i anslutning till OR-modellen.

Andra aspekter att beakta i anslutning till funktioner (och virtuella attribut, se avsnitt 3.5.4):

a. Konsistenskontroller försvåras eftersom ordbms inte har insyn över funktioners implementering och vilka data de kräver. Om visst värde tas bort kanske det exv innebär att funktionen inte kan utföras korrekt. Om funktionens egenskaper ändras så att det representerar en ny typ av värde kommer alla instruktioner som refererar till funktionen att behöva ses över.

b. Prestandaoptimeringar försvåras. Vissa funktioner kanske omfattar några få instruktioner, andra ett stort antal. Andra återigen genomlöper ett antal instruktioner som är beroende av in-parametrars värden, o s v. För att om möjligt undvika tunga funktionsanrop måste systemet känna till funktionens specifika egenskaper, något som i praktiken är svårt att realisera.

c. Användardefinierade funktioner representerar affärsregler, lagar, mer eller mindre tillfälliga förutsättningar m m. De lever därigenom ett dynamiskt liv och bör kunna tillföras eller ändras vid behov utan att en omlänkning ska behöva utföras. Dessutom, antalet användardefinierade funktioner kan förväntas bli ansenligt vid en större ordbms-installation. Att ha dem alla statiskt inlänkade i runtimesystemet ger en onödig börda eftersom en given exekvering kanske endast kommer att utnyttja ett fåtal av dem.

3.5 Datatyper

3.5.1 Inledning

I SQL-92 finns ett antal fördefinierade datatyper, exv

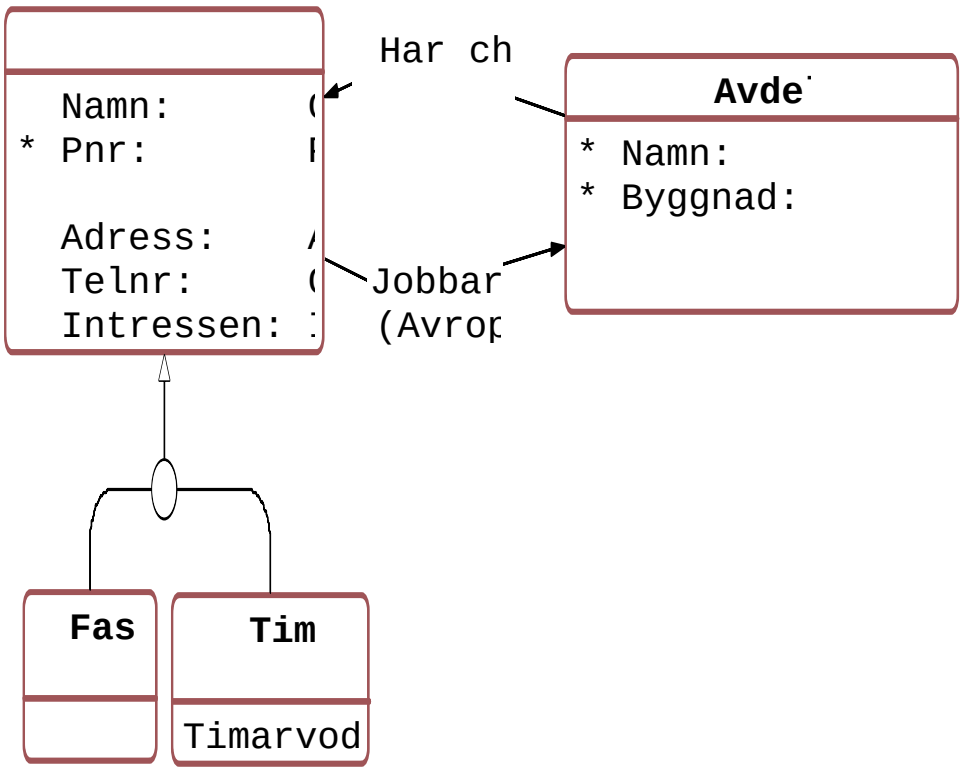
- Text (character, char, varchar, nchar, ...)
- Numerisk (numeric, decimal, integer, smallint, ...)
- Tid (year, month, day, hour, minute, date, time, ...)
- Intervaller (interval xxxxx)

Därutöver brukar vissa rdbms och framförallt ordbms erbjuda, exv

- Penningtyper
- OID
- Utrymmeskrävande typer (BLOB, large text, ...)
- Multimedia-typer
- Specialiseringar av andra datatyper
- Egendefinierade datatyper

Nya tillämpningsområden kommer att ställa fler och delvis helt nya krav på datatyper och hantering av dessa. Varje produkt har idag sin egen repertoar av möjligheter och restriktioner. Standard och samsyn saknas ännu. De exempel som tas upp i detta avsnitt syftar till att påvisa aktuella trender inom området. Vi kan förvänta oss en intensiv aktivitet kring datatyper framöver. Diskussionen baseras på den kapacitet som Illustra tillhandahåller eftersom produkten erbjuder en avancerad datatypshantering.

Exemplen svarar mot en något modifierad S-modell i figur 15.



Figur 15

3.5.2 Egendefinierade, enkla datatyper

I modellen har bl a tillkommit tre nya, enkla datatyper. Attributtypen *Pnr* är nu inte längre uttryckt som en sträng om 11 tecken, genom datatypen *char (11)*, utan mer precist definierad som *Personnummer*. *Personnummer* finns inte standardmässigt i produkten utan är en datatyp som har bedömts vara användbar i anslutning till en viss eller ett antal tillämpningar. Villkoren för datatypens uppbyggnad och innehåll måste definieras. Attributtypen *Byggnad* anges nu i form och innehåll enligt reglerna för en etablerad *Byggnadskod*. Attributtypen *Intressen* anges inte längre som en textsträng, exv ”Ridning”, utan åskådliggörs i sann, modern, multimedial anda som en bild (*image*), exv föreställande den anställde ridande på en häst. De nya datatyperna deklareras enligt följande:

```
create type personnummer (
      internallength = 11,
      input = personnummer_in,
```

```
        output = personnummer_out );

    create type image (
        internallength = variable,
        input = image_in,
        output = image_out );

    create type byggnadskod (
        internallength = 16,
        input = byggnadskod_in,
        output = byggnadskod_out );
```

Bland annat måste anges den nya datatypens interna längd i antal bytes som en uppgift till databashanteraren. För *image* kan det variera kraftigt beroende på vilken typ av bild som avses. Därför anges längden som *variable*. Dessutom anges vilka funktioner som ska aktiveras i samband med lagring i databasen (*input*) av ett värde och utsökning från databasen (*output*) av ett värde. Bland annat kan dessa funktioner användas för att definiera principer för omvandling mellan externt och internt format. Grundprincipen måste vara att det interna formatet alltid måste vara entydigt, oavsett var och hur värdet genererats. Det externa kan däremot vara avhängigt teckenrepertoar, operativsystem, programvara, presentationsgränssnitt, etc.

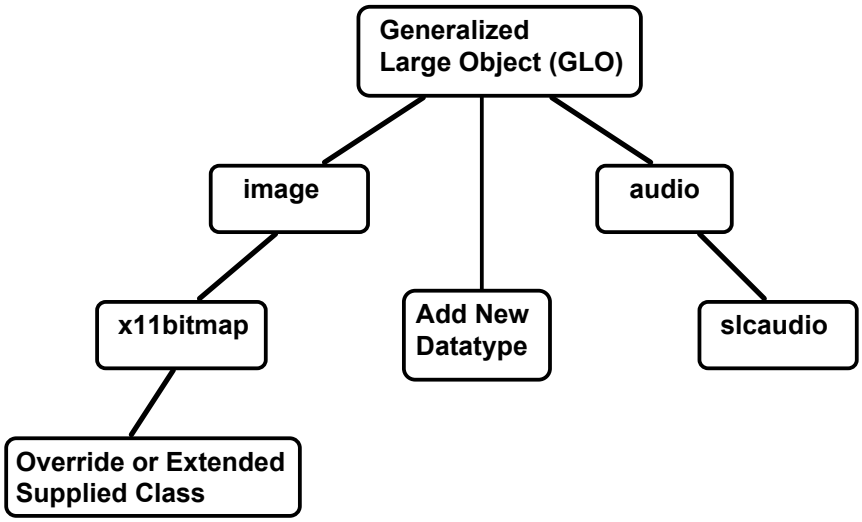
Funktionerna kan även med fördel användas för att utföra diverse kontroller eller för att initiera åtgärder av annat slag. Framförallt input-funktionen kan förväntas kontrollera värdets korrekta uppbyggnad och dess rimliga valör. Funktionen kan uttryckas som en instruktion i OR-språket när så är möjligt, eller – och kanske vanligare, i lämpligt programmeringsspråk.

I realiteten måste de ovan refererade funktionerna finnas definierade innan de tre datatyperna kan definieras. Till varje input-funktion kommer den externa representationen av värdet som en in-parameter av typen text varefter den definitionsmässigt lämnar ifrån sig den interna representationen i enlighet med den egendefinierade datatypen. Motsatsen gäller för output-funktionen.

Deklarationerna nedan exemplifierar endast funktionerna för *Personnummer*. Koden ligger i det här fallet som C-rutiner i filen ’/local/lib/personnummer.so’.

```
    create function personnummer_in (text) returns personnummer
        as external name '/local/lib/personnummer.so'
        language C;

    create function personnummer_out (personnummer) returns text
        as external name '/local/lib/personnummer.so'
        language C;
```

Ordbms UniSQL har istället valt att som standard inkludera ett antal multimediaorienterade datatyper i en specialiseringshierarki. Användaren kan specialisera dessa ytterligare om/när behov uppstår. Se figur 16.



Figur 16

3.5.3 Egendefinierade, sammansatta datatyper

Adress angavs tidigare som en enda lång textsträng. Nu önskar man adressen uppdelad i distinkta delar i form av gata, postnummer och ort. För ändamålet deklareras en sammansatt datatyp *Adress_struktur* enligt följande:

```
create type Adress_struktur (
    Gata        char (30),
    Postnr char (6),
    Ort         char (30) );
```

Utseendet är förvillande likt en tabell-deklaration, mer om detta i avsnitt 3.6.

Även verksamhetsbeskrivningen för en avdelning under attributtypen *Ansvarar_för* får nu en mer strukturerad uppbyggnad under den sammansatta datatypen *Dokument*. *Dokument* har i sin tur en sammansatt datatyp under sin attributtyp *Innehåll*, nämligen datatypen *Kapitel*.

```
create type Dokument (
   Titel        char (120),
   Innehåll     arrayof (Kapitel),
   Litteratur   setof (varchar) );

create type Kapitel (
   Rubrik       char (120),
   Text         large_text );
```

Eftersom vi enkelt vill kunna referera till visst kapitel genom indexering har vi valt att lägga dem i en array istället för som en mängd (*setof*). Dock vet vi inte exakt hur många kapitel det kan vara i de enskilda fallen varför deklarationen blir enligt *arrayof*, d v s med variabelt antal positioner. Visste vi att det alltid var max 10 kapitel i ett dokument kunde deklarationen alternativt ha formulerats som

Innehåll Kapitel [10];

Varje kapitel har en rubrik och en text, där textens längd varierar, men aldrig blir mer än ca 8 kbyte. Den lagras därför som en datatyp *large_text.* Det innebär att texten lagras på
ett separat ställe i databasen under full kontroll av ordbms avseende låsning, backup, recovery, m m (till skillnad från datatypen *external_file*, som ordbms kan referera till men för övrigt saknar möjlighet att kontrollera). I tabellpositionen lagras endast en ”handle”, d v s en referens till texten.

Önskar vi för alla avdelningar, som har chefer som bor i Säffle, ta fram dess namn, byggnadskod och rubriken för det sjunde kapitlet i verksamhetsbeskrivningen, blir select-satsen som följer:

```
select Namn, Byggnad, Ansvarar_för.Innehåll [7].Rubrik
from Avdelning
    where Har_chef.Adress.Ort = ”Säffle”;
```

3.5.4 Virtuella attribut

I de fall man har en funktion som med hjälp av en viss tabell som in-parameter genererar ett värde som kan upplevas som ett attribut till tabellen, kan det också deklareras som en virtuell attributtyp. Både *Ålder* för *Anställd* och *Fast_högavlönade* för *Avdelning* är nu deklarerade som sådana. I realiteten svarar de mot de egendeklarerade funktionerna *Ålder* respektive *Fast_högavlönade*.

```
create function Ålder (Anställd) returns Int
as external name ’/local/lib/kalkyl.so’
language C;

create function Fast_högavlönade (Avdelning)
returns setof (Fast_Anställd)
as select •
    from Fast_Anställd
        where Lön > 200000 and $1.oid in ref (Jobbar_vid);
```

Som synes har respektive funktion som in-parameter ett OID för en förekomst i den tabell, inom vilken det virtuella attributet är deklarerat. Härigenom står det funktionen fritt att operera på vilka uppgifter som helst i denna förekomst eller andra förekomster som den refererar till. I fallet *Ålder* gäller referens till en C-rutin som i sin tur innehåller utsökningar av relevanta uppgifter för aktuellt OID i och för utförande av algoritmen. Ut kommer i alla händelser attributvärdet. Kanske skulle det i det här fallet ha varit mer praktiskt att deklarera en vanlig funktion *Ålder* till vilken in-parametern *Personnummer* lämnas för algoritmen att ta fram åldern ur, utgående från dagens datum.

Fast_högavlönade är exempel på en funktion uttryckt som en OR-sats och som lämnar en mängd (*setof*) till svar, inte ett enskilt värde.

Finessen med virtuella attribut är att de syntaktiskt kan refereras som ”vanliga” attribut, istället för som en funktion med in-parameter. I och med att *Fast_högavlönade*

resulterar i en mängd kan den användas överallt där flervärdigt fält kan förekomma, exempelvis i

a. **select Namn, Byggnad,**
from Avdelning
where Har_chef in Fast_högavlönade;

b. **select Namn, Adress.Gata, Intressen**
from Anställd
where Ålder < 27 and oid in Jobbar_vid.Fast_högavlönade;

c. **select Namn, Byggnad**
from Avdelning
where count (select •
from Fast_högavlönade
where Ålder < 40;) > 10;

3.6 Table eller Type?

SQL3 skiljer mellan Type och Table på så vis att Type är den abstrakta deklarationen och Table en realisering i och för hantering av rader/förekomster. På sätt och vis kan Type sägas vara en intension och Table en extension. För att kunna operera med förekomster av en Type behöver den en extension, d v s en Table skapad för sig enligt

Create table <tablename> of type <typename>;

När tabell deklareras direkt, så som varit fallet hittills ovan, genereras internt en Type med samma namn i konsekvensens namn. SQL3 tillåter specialiseringsdeklaration över såväl Type som Table medan Illustra endast accepterar detta för Type.

En Type deklarerad inom en Table tillhör denna tabell på så vis att om en tabellförekomst, som innehåller en förekomst av en Type, raderas, raderas även Type-förekomsten. Om exv en viss avdelning raderas, raderas även det dokument som är kopplat till avdelningen. Detta är kanske gott och väl i vissa lägen, speciellt om den sammansatta datatypen endast består av lexikala element. I andra fall önskar man att dokumentet fortsätter att existera, och endast referensen från den raderade avdelningen upphör.

SQL3 tycks tillåta att en Type som komponent i en Table får referera både till annan Type och till annan Table. Dessutom tillåter SQL3 referenser från en Table A till en Type deklarerad som sammansatt attribut inom en annan Table, Table B. Här krävs minsann
en extra eftertanke för att undvika oönskade effekter av en radering.

Mer renodlat vore att ha kvar Type som en lexikal konstruktion utan upphöjelse till objekttyp. Som sådan skulle den kunna användas för Adress-strukturen, om syftet är att varje sammansatt värde unikt hör till och hanteras via den objekt/tabell det tillhör. Raderas adressen för förekomst X betyder det bara att adressen upphör att existera för just X. Om syftet däremot är att kunna radera adressen med betydelsen att adressen

upphör att existera och därmed adressuppgiften för alla som bor på denna adress, måste Adress upphöjas till Objekttyp/Table. Försvinner en sådan förekomst försvinner ju även alla referenser till den.

Av det sagda framgår att distinktionen mellan Type och Table är oklar. Detta har även uppmärksammats inom SQL3-arbetet. Flera röster har höjts för att slå ihop begreppen till ett enda och istället på annat sätt deklarera konsistensvillkor. En hel del nyheter kan förväntas inom denna del av standarden framöver.

3.7 Händelsestyrda operationer

Bland faciliteter, som egentligen inte är entydigt specifika för ordbms, men som ändå gärna förs fram i dessa sammanhang, är Rules och Triggers. De bygger grovt sett på mekanismen

On <event>
Do <action>

Illustra erbjuder definition av Rules medan UniSQL erbjuder en motsvarighet i Triggers.

Rule-deklarationer kan referera till både utsöknings- och uppdateringshändelser för viss tabell, attributtyp, rad, m m.

Action kan bestå av vilken OR-sats som helst, inklusive en alerter-operation för att informera användare om något inträffat. Man kan också begära att en uppsättning satser inom Begin ... End ska utföras istället för den ursprungliga operation som föranlett aktuellt ”event”. Några exempel:

a. **create rule <rule1> as**
on update to Anställd.Adress where current.Namn=”Andersson”
do update Anställd
set Adress = new.Adress
where Namn = ”Pettersson”;

b. **create rule <rule1> as**
on select to .....
do insert into .....

c. **create rule <rule1> as**
on update to .....
do alert <alerter>;

d. **create rule <rule1> as**
on select to .....
do instead
begin

....
end;

UniSQLs triggersyntax har överensstämmande syfte. Syntaxen är som följer:

Create trigger <trigger_name> <event_name> <event_type> [<event_target>]
[if <condition>]
execute <action>;

event_name ::=	**{before/after/deferred}**
event_type ::=	**{insert/delete/update}**
event_target ::=	**on <class_name> [(<attribute_name>)]**
action ::=	**{reject/print <message>/invalidate transaction/** **call <method>/insert <stmt>/update<stmt>/** **delete <stmt>/evaluate <method>}**

Exempel på definition:

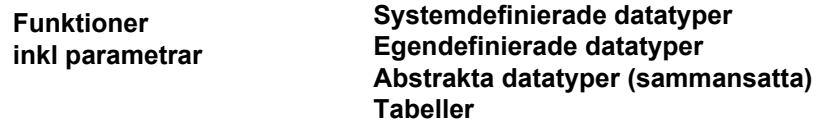
Create trigger trigg1
before update on Avdelning (Byggnad)
execute evaluate konfirmering;

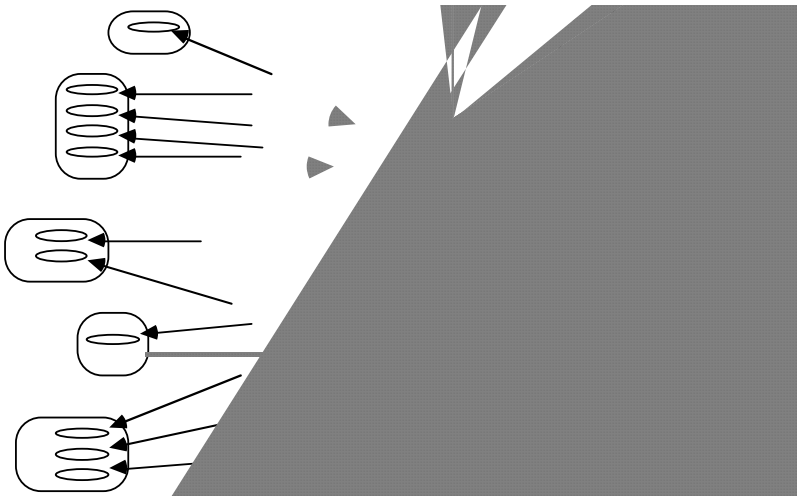
Innan uppdateringen sker begärs konfirmering från ansvarig lokaladministratör via operationen *konfirmering*.

Rules och triggers är, rätt använda, eleganta konstruktioner, användbara för diverse ändamål. Men där finns risker. Om exv ett antal actions svarar mot en och samma event, står det utanför användarens kontroll att bestämma i vilken ordning de ska utföras. Utfallen kan bli olika från gång till gång. Än mindre har man kontroll över om de motverkar varandra eller t o m tillsammans kan generera oändliga loopar. Med andra ord rekommenderas endast en noga övervägd användning, baserad på kunskap om övriga existerande deklarationer.

3.8 Egendefinierade datatyper och funktioner ”revisited”

En funktion kan behöva noll eller flera in-parametrar för att kunna utföra sitt arbete. Funktionen resulterar i ett värde. Både in-parametrar och resultat kan vara av olika typer, se figur 17. De svarta pilarna symboliserar in-parametrar medan de gråa är funktionsvärdet.





Figur 17

Både existerande och nya tillämpningsområden kan exv behöva tillgång till egendefinierade, generella funktioner mot systemdefinierade datatyper. Dessutom tillkommer i allmänhet ett antal tillämpningsberoende funktioner mot tillämpningens deklarerade abstrakta datatyper (schemat). Egna datatyper deklareras för behov som de systemdefinierade inte klarar av. De egendefinierade datatyperna måste normalt kompletteras med en uppsättning funktioner som opererar på dem. Behov av sådana funktioner kan vara tillämpningsspecifika. I ökande utsträckning är de dock ett resultat av nya trender inom informationshantering, exv multimediabaserad information, som existerande dbms inte hunnit svara upp mot eller inte funnit tillräckligt generella för att infoga i en produkt.

Inte sällan finns naturliga grupperingar av datatyper, exv genom att de har logiska beröringspunkter och/eller att de figurerar gemensamt som parametrar för funktioner som bedömts intressanta för visst tillämpningsområde, vissa beräkningsbehov eller ny mediahantering.

Ta exempelvis datatyper för att beskriva tvådimensionella företeelser och samband, tillsammans med en lämplig uppsättning operationer/funktioner på dessa, baserade på etablerade, geometriska funktioner, generella principer, etc. Bland datatyperna återfinns punkt, linje, cirkel, polygon, ... Bland funktioner kan nämnas ”Finns viss punkt inom viss cirkel?”, ”Avståndet mellan punkt X och Y”, ”Överlappar cirkel X polygon Y?”, ”Punkten där linje X och linje Y korsar varandra”, ... Dessa datatyper och funktioner har en bred tillämpbarhet, t ex inom Geografiska Informationssystem (GIS), Datorstödd konstruktion (CAD), Stridsledning, etc.

Motsvarande gruppering av nya datatyper med tillhörande funktioner kan appliceras på tredimensionell hantering, bildhantering, tidsseriehantering, ...

Eftersom Illustra för närvarande är unik när det gäller deklaration av nya datatyper och nya funktioner har leverantören finurligt nog passat på att utveckla en egen affärsidé kring detta koncept under benämningen ”Datablade”. Ett Datablade består av ett antal datatyper och funktioner svarande mot något visst generellt behov, exv ”2D Spatial Datablade”. ”2D Spatial Datablade” består av tio egendefinierade datatyper samt ett sextiotal egendefinierade funktioner opererande på en eller flera förekomster av de egendefinierade data-typerna.

Illustra tillhandahåller grundmekanismerna för att deklarera datatyper och funktioner samt för att infoga dem i produkten. Med fördel skapar sedan tredjepartsleverantörer Data-blades inom sina respektive specialistområden. Datablades ingår inte standardmässigt i produkten. Kunder köper de Datablades de behöver, slipper utveckla själv samt får (förhoppningsvis) garanterad kvalitet. Redan idag erbjuds ett antal Datablades, bl a

- 2D Spatial (analys av tvådimensionella data)
- 3D Spatial (analys av tredimensionella data)
- Web (skapande av WWW-sidor från Illustra-databas)
- Image (lagring och bearbetning av bilder)
- Image query (innehållsbaserade frågor mot bilder)
- Text (textbearbetning och analys)
- Text conversion (text-transformationer)
- Visual Intelligence (innehållsbaserad sökning mot bild, ljud)
- Time series (tidsserieanalyser)

En ström av ytterligare Datablades kan förväntas framöver.

Begreppet Datablade är välfunnet. Tanken är att det ska fogas in i produkten och kunna användas lika enkelt och elegant som ett rakblad i en rakhyvel. Att Datablades samtidigt förts fram och uppmärksammats som ett helt nytt idékoncept i den snabbt framväxande multimedia-trenden, har knappast varit till marknadsmässig nackdel. Andra ordbms-leverantörer kommer säkerligen att ”haka på” denna trend. Exv stödjer IBMs DB2, Version 2 användardefinierade datatyper och funktioner. I realiteten är inte heller skill-naden mot ett konventionellt objektbibliotek uppseendeväckande stor. Det unika med Datablades är snarare tillgången på ett integrerat verktyg för att bygga och infoga dem.

Sannolikt kommer det tyvärr att dröja innan standarder avseende principer och gränssnitt vuxit fram – ett besvärande kundperspektiv.

3.9 Sammanfattning

Avsnitt 3 har introducerat modelleringsbegreppen

- Table – Class – Type – ObjectType (nästan synonymer)
- Flervärdiga attribut/kolumner
- OID som unik representation av förekomster
- Explicita enkelriktade samband med hjälp av OID-referenser
- Enkla datatyper
  - Fler fördefinierade
  - Egendefinierade
- Funktioner
  - Fördefinierade
  - Egendefinierade
  - Virtuella attribut
- Komplexa datatyper (alt sammansatta attribut)

  - Egendefinierade
- Specialisering som medger multipla arv av
  - Enkla datatyper
  - Komplexa datatyper
  - Funktioner
- Business Rules, Integrity Constraints, ...
  - Rules, Triggers, Events, Actions

I mångt och mycket är OR-modellen mycket lik en ”vanlig” S-modell men baserad på R-modellens begrepp. De fortfarande enkelriktade sambandstyperna tycks vara en eftergift
åt R-modellens principer.

Vissa hävdar att OR-modellen inte är annat än ett återupplivande av onormaliserade tabeller, d v s där en kolumn kan representera en sammansatt datatyp. I det perspektivet ligger OR-modellens ”abstrakta datatyp” mycket nära R-modellens ”domän”.

Händelse- och regelstyrda operationer är ett tillägg, så även möjligheten att etablera egna datatyper. En flexiblare datatypshantering bör dock snarare hänföras till en komplettering av relationsmodellen än ett resultat av någon OO-anpassning.

Sättet att se på och hantera funktioner är ett mellanting mellan konventionell funktion-data-ansats och O-modellers inkapsling av funktioner (beteenden) i objektklasser. Den vedertagna SQL-syntaxen används så långt möjligt. Nödvändiga tillägg görs i dess ”anda”.

Observera att det som redovisats i detta avsnitt är en aktuell bild av den pågående mycket dynamiska trenden inom ”extended-relational” – den trend som för närvarande tycks ha den starkaste drivkraften. Vi får inte glömma de krafter som kommer från OO-hållet, i första hand representerade genom ODMG-standarden. Se vidare avsnitt 4.

Dessutom pågår en kritisk debatt kring ”extended-relational”-ansatsen. Vissa hävdar att relationsmodellen dåligt passar för OO-påbyggnad. Resultatet blir med nödvändighet en halvmesyr som inget ”läger” kommer att ta till sig. R-modeller och O-modeller har sina egna kvaliteter och ska fortsätta att användas inom sina naturliga tillämpningsområden. OR-modellen är ingenting annat än en S-modell som bör stödjas genom dess egna förtjänster och genom ett egenavpassat språk.

Vad alla dock tycks vara ense om är att OR-modellen representerar en mycket intressant och viktig trend som i sinom tid och i avslipad form kommer att erbjuda helt nya för-utsättningar för avancerad datahantering.

4. Standardiseringsansträngningar

4.1 SQL3

SQL är ett gränssnittsspråk baserat på R-modellen. SQL-arbetet bedrivs inom den del av ISO, Sub Committee 21, Working Group 3 som arbetar med databasspråk (DBL). Tidigare har denna arbetsgrupp bl a tagit fram olika versioner av SQL-standarder under den gemensamma beteckningen ISO 9075 (”Information Technology – Database Languages – SQL”). Ett antal länder medverkar aktivt i arbetet genom lokala SQL standardkommittéer. Den första standarden kom 1987 under beteckningen ISO 9075:1987. Den innehåller i stort sett det som normalt återfinns i vanliga introducerande SQL-läroböcker. Därefter kom ISO 9075:1989, som inkluderade en mindre komplettering kring referential integrity.

En större utökning resulterade i ISO/IEC 9075:1992, ofta refererad som SQL-92. De två första standarderna finns i allmänhet realiserade i existerande produkter. SQL-92 innehåller en provkarta på önskade tilläggsfaciliteter baserade på ca 15 års forskning, produkter, erfarenheter. Resultatet är en betydligt utökad syntax och därmed funktionalitet. Den bedöms vara mycket välgenomtänkt och stabil. Bland kompletteringar kan nämnas:

– Tabellhantering (DROP TABLE, ALTER TABLE)
– Domänspecifikation (CREATE DOMAIN)
– Nya datatyper (DATE, TIME, BIT string, ...)
– Nya funktioner (CURRENT DATE, SUBSTRING, TRIM, ...)
– Metaschemahantering (DEFINITION SCHEMA, INFORMATION SCHEMA)
– Behörighetskompletteringar
– Temporära tabeller inom transaktion.

Tyvärr har den ökade ambitionsnivån skapat ett dilemma för kunden genom att inte alla rdbms-leverantörer orkat eller önskat inkludera standarden fullt ut i sina produkter. Vissa planerar göra det medan andra inte har för avsikt att erbjuda 100% överensstämmelse. Samma dilemma, men från motsatta perspektivet, har uppstått i och med att SQL-92 trots allt saknar vissa, av kunder önskade, egenskaper. Leverantörer har i olika grad, efter eget huvud, tillfogat sådana faciliteter. Med andra ord, standard och produkter överensstämmer inte, något som givetvis genererar problem, framförallt i större verksamheter med kanske flera olika rdbms, som på olika sätt har att samverka.

Sedan slutet av 80-talet har objektorienterade modeller vunnit ett allt starkare intresse. Dessa har egenskaper som i många tillämpningsområden anses vara effektiva och användbara och som saknas i R-modellen. Denna brist avser man råda bot på i nästa generation av SQL, d v s den som brukar gå under arbetsnamnet SQL3.

SQL3 inkluderar SQL-92 fullt ut samt ett antal komplement som inte ”kom med” i SQL-92, bl a datatyperna BOOLEAN och värdelista samt TRIGGER-funktion. Dessutom, det helt nya, nämligen stöd för en uppsättning objektorienterade begrepp. Till råga på allt har man kompletterat syntaxen med uttrycksmöjligheter som återfinns i

vanliga program-meringsspråk. Språket är därmed ett fullödigt programmeringsspråk inklusive databasfaciliteter, d v s långt ifrån sitt relativt enkla ursprung som frågespråk.

De specifikt objektorienterade delarna brukar gå under beteckningen MOOSE (Major Object-Oriented SQL Extensions) eller Object SQL.

ISO-arbetet har pågått sedan 1990, under hela perioden med stor intensitet. Svårigheterna har visat sig betydligt större än man från början räknade med. Dels innebär utvidgningen till ett generellt programmeringsspråk en avsevärd komplexitetsökning. Dels är inord-ningen av O-modellbegrepp inom en R-modell en komplexitet i sig. Vissa anser idén befängd. Resultatet blir en mastodont utan profil. I alla händelser räknar man för närvarande inte att vara framme vid målet, en ISO-standard, förrän tidigast 1998. Av praktiska skäl och för att skynda på standardiseringsprocessen har man numer delat in
det tänkta resultatet i ett antal fristående delar.

Man kan fråga sig om SQL3 någonsin får praktisk bärkraft. Två saker motverkar. Det ena är att arbetet drar ut på tiden. Behov och andra kommersiella drivkrafter gör att rdbms-leverantörer väljer att gå sina egna vägar, d v s implementera en del OO-kapacitet i sina produkter efter ”eget huvud”. Visserligen kan man förmoda att dessa tillägg får en form och syntax som någotsånär svarar mot dagsläge i SQL3-arbetet eftersom de flesta leverantörer också medverkar i standardiseringsaktiviteterna. Likafullt kan man utgå ifrån att skillnader i den egna databashanterarens uppbyggnad, krav hos den primära kundbasen, m m kommer att resultera i ett antal inkompatibla varianter.

Det andra har att göra med standardens omfattning. Ju mer omfattande och komplex standard (i dagsläget ca 1000 sidor) desto större resurser kommer att krävas för att förverkliga kompatibla produkter. Leverantörerna har redan problem med SQL-92. Dessutom ökar sannolikheten att standarden inte är stabil. Nya, modifierade versioner kan förväntas p g a syntaktiska fel eller ofullständigheter, icke förutsedda eller förutsebara konsekvenser, m m. Med en marknad som numer är splittrad i relations-, hybrid-, objekt-modeller samt en mångfald ickekompatibla produkter, är det inte heller sannolikt att kunna uppnå samma uppslutning kring SQL3 som tidigare kunde åstadkommas kring SQL.

En tangerande, intressant standardiseringsaktivitet under benämningen SQL/MM pågår sedan 1993. Syftet är att i SQL3 definiera ett antal paket (klassbibliotek) av abstrakta datatyper (ADTs) där varje paket svarar mot modell och operationer på någon typ av multimedial datatyp. Bland paket under framtagande av specialister inom respektive datatyp kan nämnas paket för bild, ljud, musik, video, animation, dokumenthantering. Jmf Illustras Datablades.

4.2 ODMG-93

Vi har redan konstaterat att odbms-branschen är splittrad. Varje produkt har utvecklats efter egna riktlinjer. Resultatet har blivit en rikedom av mångskiftande funktionaliteter.

1991 kom de större odbms-leverantörerna underfund med att branschen levde farligt, att något måste göras. Följdriktigt etablerade de en gemensam organisation med syfte att ensa produktegenskaper och återställa förtroende. Gruppen kom att kallas Object Database Management Group, i vardagligt tal kortfattat ODMG. Röstande medlemmar är Objectivity, Object Design, Ontos, O2, Poet, Servio och Versant. Därutöver finns ett antal reviewers.

ODMGs primära mål var inledningsvis att skapa förutsättningar för att kunna flytta odbms-tillämpningar mellan olika odbms-leverantörers produkter. För detta krävs ett enhetligt odbms-gränssnitt. Formuleras all samverkan mellan tillämpning och odbms uttryckt i gränssnittets språk, kan i princip valfri produkt, som svarar upp mot gränssnittet, användas. Kunderna binder sig inte till viss leverantör, bara till den modell och det språk som gäller för gränssnittet.

Med beundransvärd snabbhet tog man fram en gränssnittsspecifikation som publicerades i september 1993 under den något pretentiösa titeln ”The Object Database Standard: ODMG-93, Release 1.0”. Sedermera har uppdateringar i form av Release 1.1 (1994) och Release 1.2 (slutet av 1995) publicerats.

ODMG-93 omfattar specifikation av

- Object Model
- Object Definition Language (ODL)
- Object query language (OQL)
- C++-bindning
- Smalltalk bindning

Genom att en mycket klar majoritet av odbms-leverantörerna ställer sig bakom dokumentet, anser man sig etablera en ”de facto”-standard. Man hoppas samtidigt lägga grunden för en stabilitet och mognad, som marknaden ska uppleva vara både trovärdig och tilltalande. För att ytterligare höja trovärdigheten har ODMGs medlemmar förbundit sig att inkludera gränssnittet i sina produkter.

4.3 Samverkan

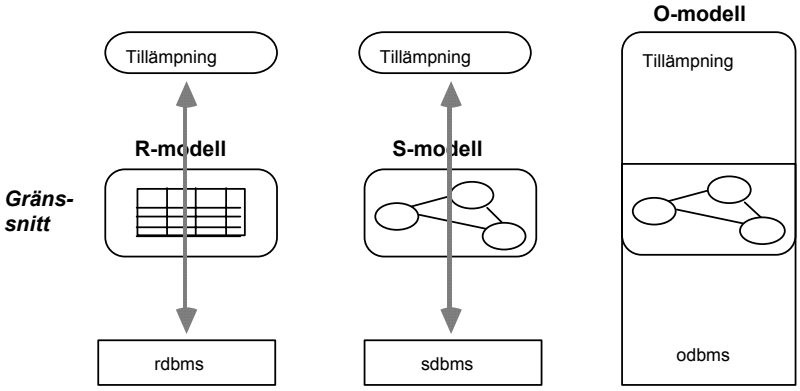
O-modellen och OR-modellen har det mesta gemensamt. Kvarstående skillnader är att hänföra till respektive modells ursprung. SQL3 och OQL har till vissa delar snarlik syntax, i andra ganska stora olikheter. Med god överensstämmelse i modellerna borde inte skillnaderna i språksyntaxen behöva vara så stor. Kunderna är knappast betjänta av flera språk mot samma modell. Av den anledningen har under det senaste året etablerats en samverkan mellan ODMG, ANSI/X3H7 (amerikanska kommittén för Object Information Management) och ANSI/X3H2 (amerikanska kommittén för SQL-standardisering) för att överbrygga skillnaderna. Påtryckningsfaktor har bl a varit en allt starkare användarmarkering för en enhetlig syntax. Även ett antal inlägg i media har haft samma syfte.

5. Typiska produktegenskaper

5.1 Gränssnittsprincip

Antalet ordbms-produkter är få. Att därifrån dra generella slutsatser om vad som är dominerande lösningsstrategier är knappast meningsfullt. Vi kommer endast att beröra några olika lösningsalternativ.

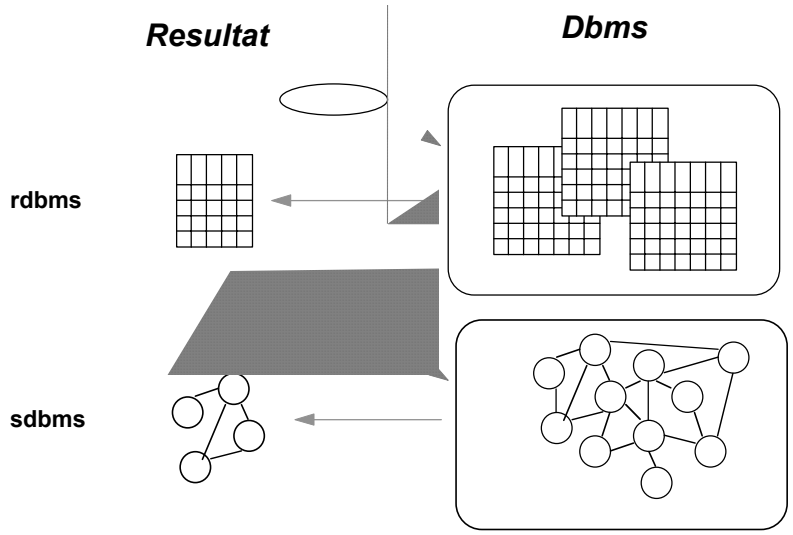
Gemensamt för ordbms är en funktion-data-ansats. Databas hanteras av en databas-hanterare som opererar på direktiv som kommer över ett standardiserat gränssnitt. Direktiven formuleras enligt gränssnittets syntax och med referens till komponenter (abstraktioner) i databasens schema och eventuellt även komponenter i databasen. Denna grundprincip har av tradition gällt i huvudsak all databashantering oavsett vilken datamodell (nätverksmodell, relationsmodell, semantisk datamodell, ...) de baserats på. Odbms utgör härvidlag ett undantag med sin starka integrering med OO-språk.
Se figur 18.



Figur 18

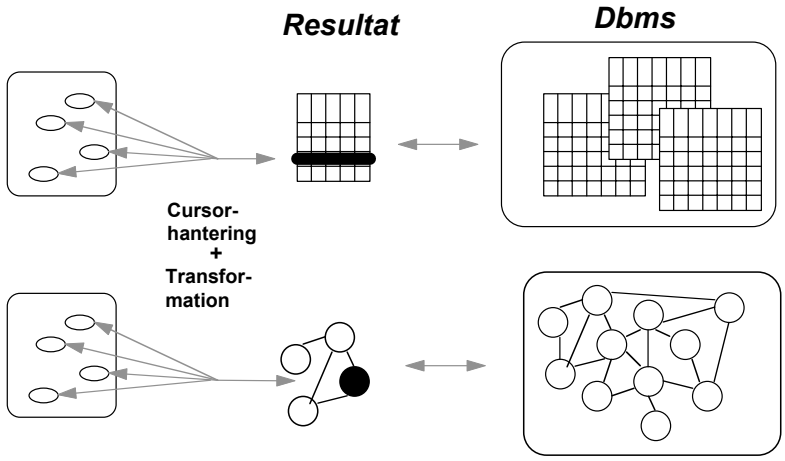
Med ett standardiserat databasgränssnitt, i form av modell och språk, kan i princip god-tycklig dbms-produkt som svarar mot gränssnittet utnyttjas. Nya kan ersätta gamla, o s v utan att de tillämpningar som opererar på databasen behöver ändras.

I det enklaste fallet levereras en fråga till dbms som tar fram ett svar och sänder det enligt uppgjort format tillbaka till frågeställaren. I rdbms-miljö kommer svaret som en tabell, i en sdbms-miljö kanske som en mängd, objektstruktur, hierarkiskt uppbyggd svars-sekvens, eller dylikt (standard saknas). Se figur 19.



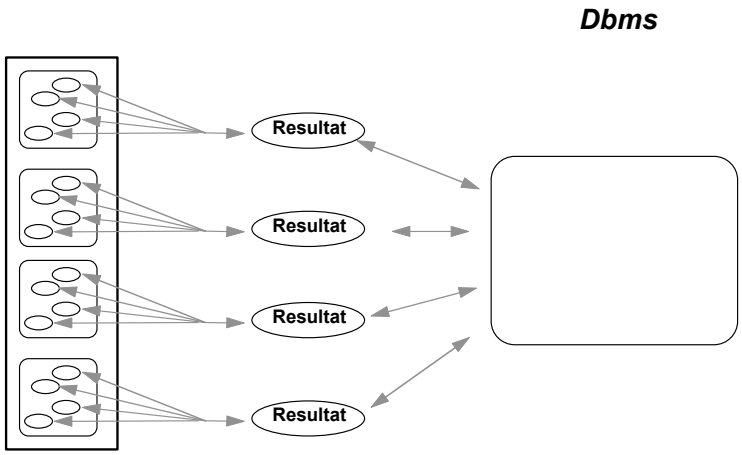
Figur 19

Ofta önskar tillämpningen inte bara ta emot och vidarebefordra svaret utan också arbeta vidare med det. För detta ändamål krävs en översättning från svarsformatet till data-strukturen för det programmeringsspråk som tillämpningen implementerats i. I grunden är problemet detsamma oavsett vilket gränssnitt som tillämpas. Målet är att skapa möjligheter för tillämpningen att nå svarets olika komponenter. Används rdbms refererar tillämp-ningen till en rad i svarstabellen åt gången genom användning av en cursor (pekare). Respektive komponent i raden förs till matchande variabel. Motsvarande hantering måste utföras även vid andra gränssnitt, t ex per objekt vid sdbms (figur 20).



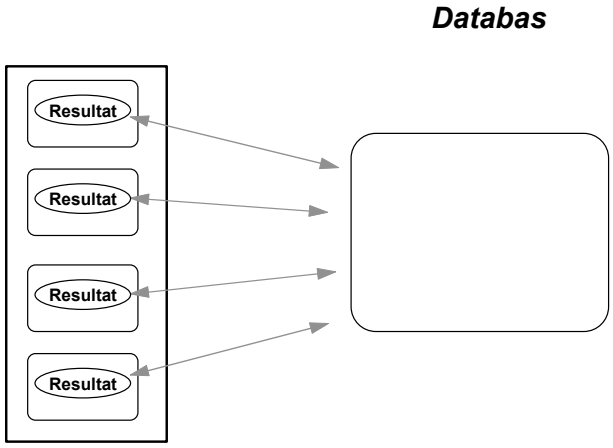
Figur 20

Funktion-data-ansatsen bygger på filosofin att data i databasen representerar en kunskaps-massa, en uppsättning utsagor om någon verklighet i och för något visst syfte. Data-modellen eller schemat redovisar den abstraktion av verkligheten som gjorts för syftet ifråga. Denna kunskapsmassa kan många behöva tillgång till för många olika behov och bidraga till från många olika situationer (figur 21).



Figur 21

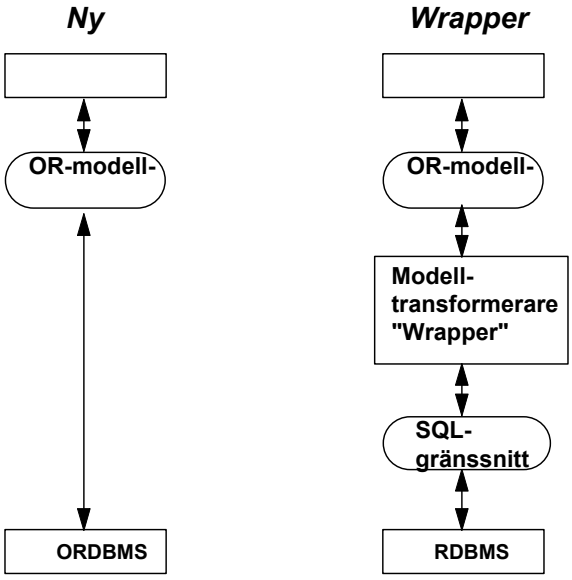
SQL3-visionen går längre. Där ser man framför sig tillämpningar helt utvecklade i SQL3. Detta är möjligt eftersom SQL3 i sitt nuvarande skick är ett generellt programmerings-språk inklusive databasoperationer. Någon transformation till och från språkvariabler behöver då inte deklareras. Transformation och cursorhantering sköts inom den egna kapaciteten. Se figur 22. Frågan är dock om alla i framtiden önskar använda SQL3 som tillämpningsspråk?



Figur 22

5.2 Realiseringsprincip

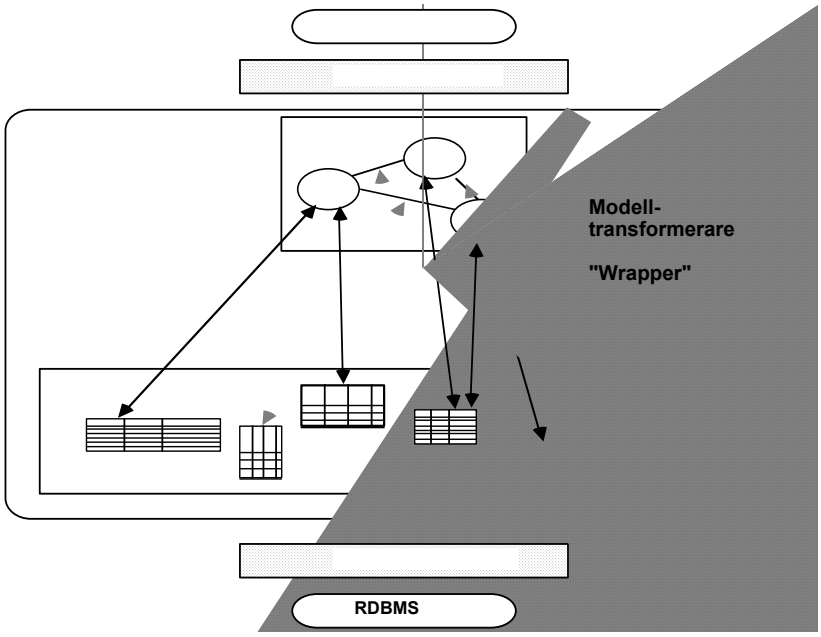
Åter till gränssnittsförutsättningen. Att bygga en helt ny dbms som svarar mot en OR-modell (S-modell) är ett mycket resurskrävande jobb som kräver höggradig kompetens inom ett antal av databasområdets och tangerande discipliner. Dessutom tar det lång tid – ännu längre att nå stabilitet och full prestandatrimning. Fördelen ligger i att kunna gripa sig an problemet med friska ögon och utan barlast i form av olika typer av hänsyn till existerande teknologi. UniSQL har tillämpat denna strategi. Se figur 23 a.



Figur 23a *Figur 23b*

Ett betydligt enklare förfaringssätt är att bygga på en existerande databasteknologi, t ex ett befintligt rdbms, enligt wrapper-teknologi (figur 23 b). Eftersom tillämpningen ska kunna uppleva samma gränssnitt som i figur 23 a blir problemet att transformera en given OR-modell till en motsvarande R-modell och att översätta en fråga i OR-gränssnittet till en motsvarande SQL-sats(er). Allt detta ska ske under ytan, dolt för tillämpningen – anpassningen till rdbms är insvept, höljt (wrapped).

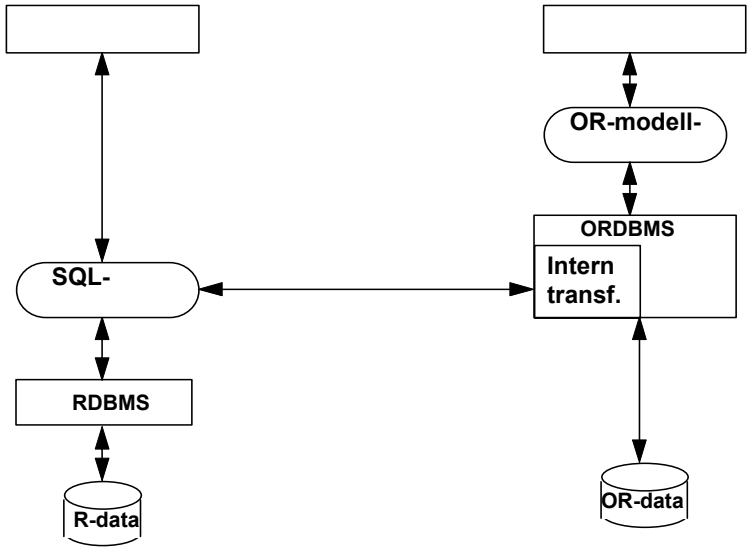
Den stora fördelen ligger i användningen av beprövad teknologi och med full uppsättning välfungerande databasfunktionalitet. Bekymret är främst prestandarelaterat. Översätt-ningen sker först till SQL-sats – en komplex operation. Dels ska hänsyn tas till den definierade avbildningen mellan modellerna. Dels ska korrekt syntaktisk översättning åstadkommas. Ju större skillnader mellan modellegenskaper desto komplexare operation. Se figur 24.



Figur 24

Därefter ska SQL-satsen i sin tur översättas och omformas till exekveringsstrategi mot databasen på vanligt sätt enligt respektive rdbms-produkts principer. Optimeringar av databasoperationer baserade direkt på OR-modellens egenskaper kan knappast påräknas, endast på dess R-modellmotsvarighet. Efter utförd exekvering ska till sist resultatet transformeras från SQL-tabell till OR-gränssnittets format. Produkten Odapter tillämpar denna strategi.

En tilltalande indirekt fördel med wrapper-alternativet är möjligheten till smidig övergång från rdbms- till ordbms-miljö. I och med att ett rdbms utnyttjas kan vissa tillämpningar arbeta direkt mot rdbms medan andra arbetar via ett OR-gränssnitt. En förutsättning är givetvis att användaren kan påverka transformeringsprincipen så att OR-modellen resulterar i önskad R-modell. Under ett övergångsskede kan viss del av databasen komma att hanteras explicit under existerande rdbms medan, exv ny typ av information läggs i en OR-databas under intern kontroll av ett ordbms. Det ankommer på ordbms att fördela trafiken korrekt. När väl R-tillämpningen skrivits om till en OR-tillämpning överförs
den ”gamla” informationen till OR-databasen och den gamla R-databasen avvecklas. Se figur 25.



Figur 25

En kompromiss mellan ovanstående två ansatser är att delvis utnyttja existerande teknologi enligt följande. Många rdbms är uppdelade i en gränssnittsmodul (Interface Manager) och en intern lagringsmodul (Relational Storage Manager). Den första har att

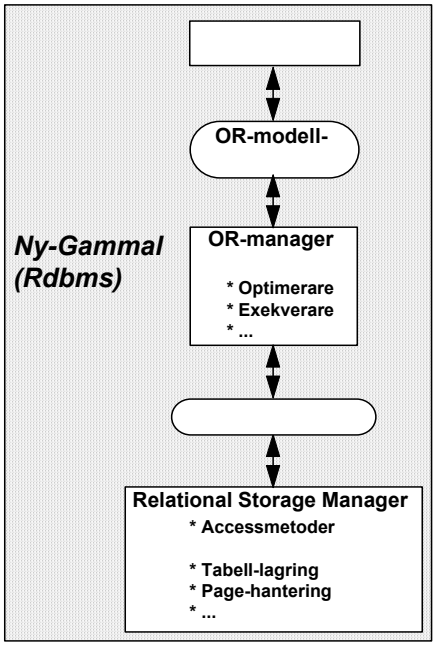
- översätta SQL-satser till internt format
- optimera exekveringsstrategin med hänsyn till den aktuella satsens uppbyggnad och diverse vägledande information i schemat
- utföra exekveringen, bl a genom anrop till lagringsmodulen.

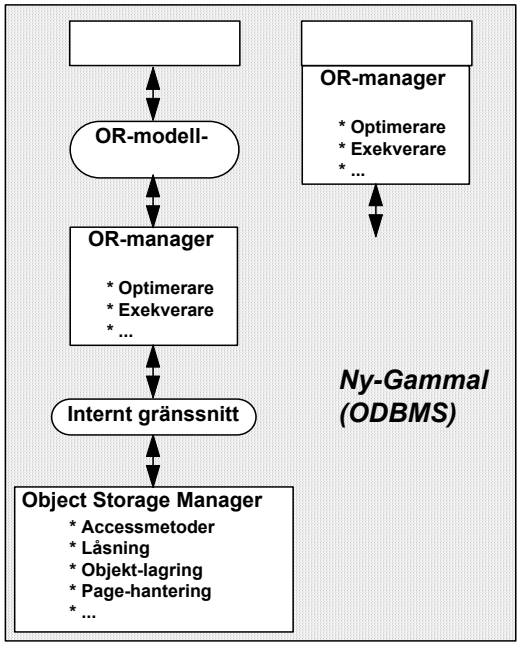
Lagringsmodulen opererar genom ett internt gränssnitt och är helt koncentrerad på att effektivt hantera lagring och utsökning i databasen. Till sitt förfogande har modulen

diverse accessmetoder, låsningsmekanismer, tabell-lagringsprinciper, fysisk sidhantering, m m.

Genom att byta ut gränssnittsmodulen men utnyttja lagringsmodulen kan en rimlig kompromiss åstadkommas. Den nya gränssnittsmodulen opererar direkt på OR-gränssnittet och kan därigenom optimera med hänsyn till såväl OR-modell som OR-syntax. Gränssnittsmodulen blir komplexare eftersom den direkt måste klara av modell-transformationen mellan OR-modell och intern lagringsmodell. Finessen är att en språköversättning undviks samtidigt som systemet har mer kontroll över händelse-förloppet. Exekveringen kan utföras direkt via lagringsmodulens interna gränssnitt. Betydligt högre prestanda kan påräknas än i alternativet i figur 23 b. Däremot sker den interna bearbetningen fortfarande enligt relations-teknologi, vilket rimligtvis borde resultera i en del prestandaförlust jämfört med en specialkonstruerad figur 23 a-lösning. Se vidare figur 26 a. Produkten Illustra tillämpar denna princip.

Samma tankegång är tillämpbar gentemot ett existerande odbms i den mån det har samma principiella uppbyggnad. Se figur 26 b. Bl a produkten Versant tillämpar denna princip.





Figur 26a *Figur 26b*

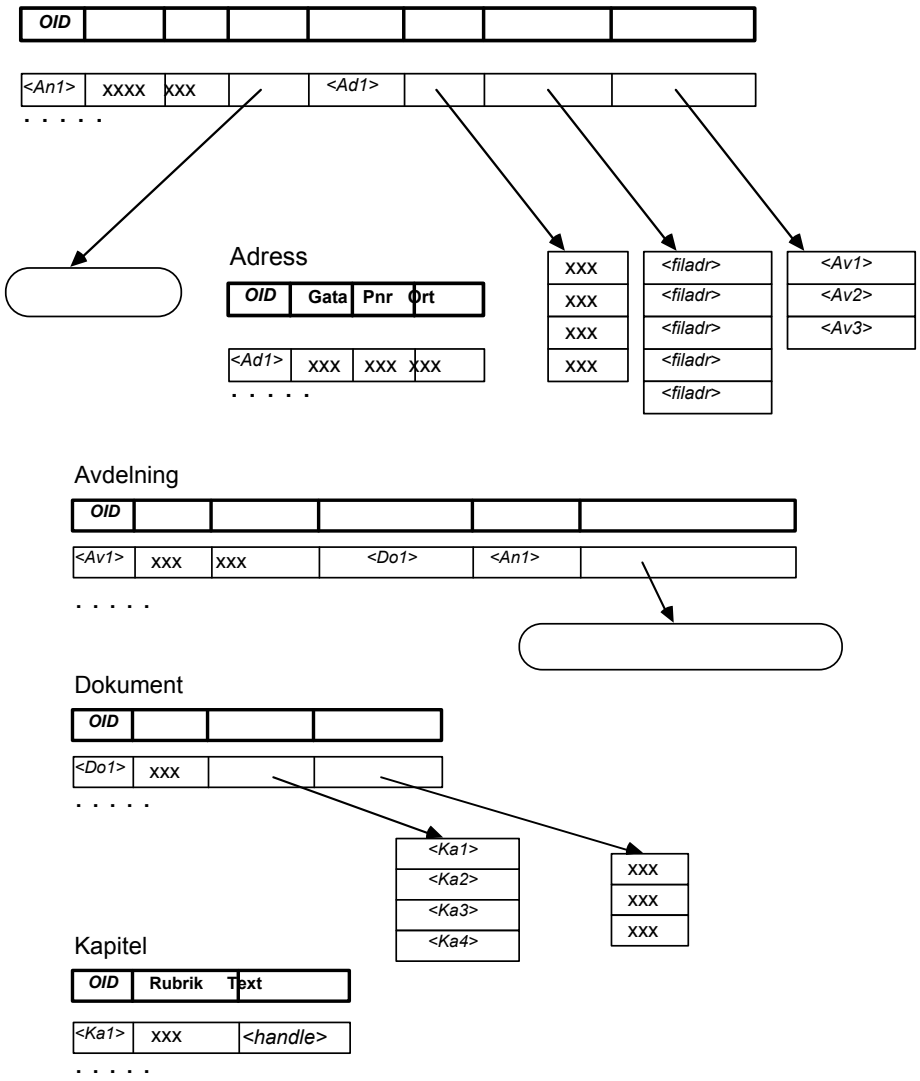
5.3 Modelltransformation; Exempel

För att något mer åskådliggöra transformationsproblematiken mellan OR-modell och R-modell utgår vi från tabellerna Anställd och Avdelning i OR-modellen i figur 15, ovan och visar hur de i princip realiseras i Illustras interna relationsbaserade lagringsmodul. Andra produkter tillämpar andra strategier.

Värden av datatypen *Image* lagras i separata filer, p g a sin storlek. I det flervärdiga fältet *Intressen* läggs endast filadressen. Som tidigare nämnts ligger dessa filer inte under full kontroll av dbms. De kan åtkommas från andra processer. Kapiteltexten är en

mellan-storlek (*large_text*) som helt administreras av dbms men som av olika skäl anses olämplig att placeras direkt i tabellen. Där läggs istället en intern referens (handle).

Innehållet i varje flervärdigt fält i en rad bryts **logiskt** ut och ersätts med en intern pekare till den egentliga lagringsplatsen. Ifråga om virtuella attribut sker referens till motsvarande funktion. Se figur 27.



Figur 27

Varje utbrutet flervärdigt fält (värdemängd) måste kunna hanteras som en tabell. Det vore orealistiskt att skapa en tabell per flervärdigt fält i varenda förekomst (rad) i varenda tabell. En kontinuerlig uppdatering av det interna schemat skulle bli följden. Istället etableras en tabell per flervärdig attributtyp. Tabellen innehåller attributvärdena i första kolumnen samt OID för den eller de rader i tabellen som har detta värde. Denna typ av tabell använder sig inte av egna OIDs. Namngivning sker med ett understrykningstecken (underscore), därefter namnet på den tabell attributtypen hör till, underscore igen samt attributtypens namn. På platsen för det flervärdiga fältet placeras identifikationen för en cursor, som sedermera kan användas för referens till specifika

värden i värdemängden.
Se vidare exempel för tabellen Anställd i figur 28.

Anställd

OID	Namn	Pnr	Ålder	Adress	Telnr	Intressen	Jobbar_vid	
<An1>						<cursor>		<cursor>

. . . . .

_Anställd_Telnr

08-111	*<An1>*
08-222	*<An2>*
08-222	*<An3>*
08-333	*<An1>*
08-444	*<An4>*
08-555	*<An2>*
08-555	*<An3>*
08-666	*<An1>*
08-777	*<An5>*
08-888	*<An1>*

. . . . .

_Anställd_Jobbar_vid

<Av1>	*<An1>*
<Av1>	*<An2>*
<Av1>	*<An3>*
<Av2>	*<An1>*
<Av2>	*<An2>*
<Av2>	*<An4>*
<Av3>	*<An1>*
<Av3>	*<An5>*
<Av4>	*<An6>*
<Av4>	*<An7>*

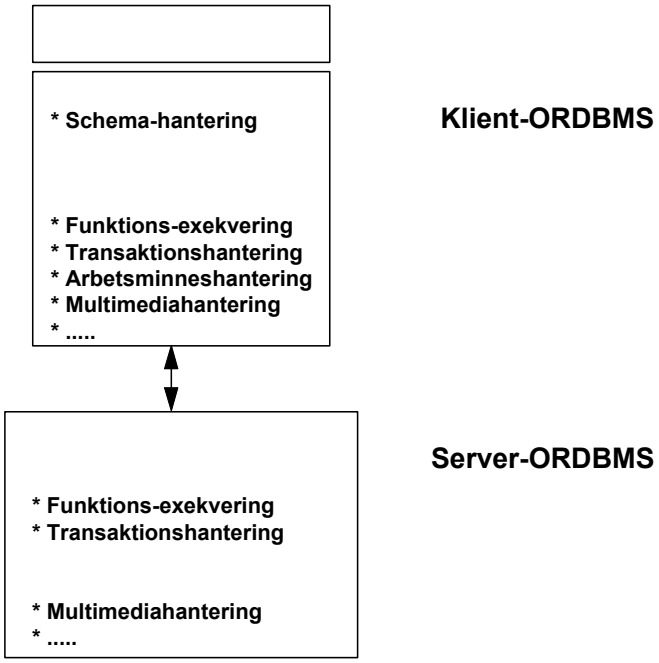
. . . . .

Figur 28

Motiveringen till att placera värdet i den första kolumnen kan förmodligen sökas i att värden många gånger återkommer i where-delen av en SQL-sats, snarare än att värden refereras via ett OID.

5.4 Klient/server-miljö

Den funktionella balansen mellan klient och server varierar från ordbms till ordbms. I vissa fall kan användaren påverka balansen, exv var exekvering av funktioner ska ligga, i andra fall är man hänvisad till produktens inbyggda egenskaper. För att ge en uppfattning om en möjlig klient/server-uppdelning och för att belysa ordbms-produkters komplexitet presenteras översiktligt i figur 29 den lösning som valts för produkten UniSQL.



Figur 29

Några kommentarer kring UniSQL-ansatsen:

Schemat kopieras till varje klient, dels för att det behövs vid frågeöversättning och optimering, dels för att avlasta servern diverse frågeverksamhet riktat mot schemat. Även behörighetsinformation kopieras till klienterna för att möjliggöra kontroll redan ”vid källan” och som en avlastning av servern. Att frågor översätts och optimeras vid klienten är lika givet som att frågeexekveringen sker nära databasen, d v s hos servern.

Transaktionshantering med låsning, samtidighetskontroll, recovery, m m sker vid servern eftersom den har det direkta ansvaret för databasens rätta konsistens. På klientsidan genereras låsningsbehoven för en transaktion i samband med frågeöversättningen. Dessa behov lämnas sedan över till servern innan frågan lämnas över för exekvering. Klienten kan ”spara” erhållna lås mellan transaktioner så länge som ingen annan klient efterfrågar uppgifterna.

Funktioner, definierade i ett UniSQL-schema, kan vara skrivna i C eller C++. De exekveras alltid hos servern. Därutöver har UniSQL faciliteter för integrering med C++ och Smalltalk i enlighet med hur odbms fungerar. På så vis kan tillämpningsfunktioner enligt normala OO-principer definieras i OO-språkets klassbeskrivning. Dessa funktioner (methods) exekveras normalt på klientsidan. De lever ett från databasen och schemat fristående liv.

En finurlig arbetsminneshantering hos klienten kan vara höggradigt prestandabefrämjande både för klienten, nätbelastning och indirekt för servern (som då behöver jobba mindre). I arbetet ingår att så långt möjligt hålla kvar kopior av objekt (rader), även mellan transaktioner samt att rensa bort inaktuella objekt (garbage collection). Multimediahantering erbjuder sina speciella problem hos både klient och server på grund av sin storlek. Olika strategier för nättransport kan tillämpas beroende på förväntad användning.

Observera att till detta kommer i flertalet produkter diverse ytterligare dbms-funktioner såsom

- loggning
- backup
- distribution av data och/eller funktion
- replikering av data

- versionshantering
- hantering av långa transaktioner
- aktiv konsistenskontroll
- .....

6. Produkter

6.1 Översikt

Antalet ordbms-produkter är i dagsläget begränsat. Gränsen mot rdbms blir dessutom otydligare allteftersom rdbms-leverantörer inkluderar olika varianter av stöd för multimediala datatyper i sina produkter. Exv inkluderar Oracle7, Release 7.3 flera nya datatyper. IBMs DB2, Version 2 inkluderar stöd för användardefinierade datatyper och funktioner samt för bl a ljud, bild, video.

Vad som saknas är stöd för en mer objektorienterad modell (S-modell). Dock förutskickar rdbms-företagen att deras kommande releaser kommer att vara objektorienterade. Vad detta i realiteten kommer att innebära är dock ännu inte känt. Tydligt är ändå att man av marknadshänsyn känner behov av att ”markera revir”.

Över till produkter som stödjer någon typ av S-modell.

En tidig ordbms-representant är Hewlett-Packards **Odapter**, tidigare OpenODB. Den utvecklades först som en forskningsprototyp (IRIS) vid laboratoriet i Palo Alto, Kalifornien under sent åttiotal. Innan kategorin ordbms ”uppfunnits” klassades produkten som en odbms. Den tillhör wrapper-kategorin i och med att den baserar sig på en rdbms i botten. Numer används Oracle, tidigare var det Allbase. Eftersom man var tidigt ute blev av naturliga skäl gränssnittet egenutvecklat och präglat av visionerna hos IRIS-utvecklarna. När så SQL3-arbetet tog fart och valde en delvis annan modell och syntax kom Odapter att hamna något vid sidan om ”mainstream”. Sannolikt överväger HP om man ska anpassa produkten mot SQL3 eller helt enkelt låta den ebba ut.

De två produkter som främst kommit att förknippas med ordbms är **UniSQL** och **Illustra**.

UniSQL Inc, Austin, Texas startades 1990 av Dr. Won Kim, en av kändisarna inom databasområdet under 1980-talet, med bl a förflutet inom IBM Research och MCC (ORION-projektet). Själv utnämner han sig anspråkslöst till ”the father of Object-Relational DBMS technology”. Den första releasen av UniSQL kom redan 1992. UniSQL Inc är privatägt men med ett minoritetsägande av Nippon Telephone and Telegraph. Med denna jätte som stöd och genom en stabil kundbas, bl a mycket tidigt inom olje- och gasindustrin, har man kunnat undvika beroende av riskkapital.

Produkten tillhör Ny-kategorin eftersom den nyutvecklats. Egenskaperna har därigenom kunnat formas utan negativ barlast av R-modellberoenden. Det interna lagringsformatet tycks också mer O-modell-anpassat. Genom detta har man utan större möda kunnat skapa en OO-språksintegrering mot C++ och Smalltalk. Det återstår att se i vad mån NTTs inflytande över företaget motverkar risken (möjligheten) att bli uppköpt av någon stor rdbms-leverantör.

Illustra från Informix, Menlo Park, Kalifornien är en produktifiering av forskningsprototypen Postgres, utvecklad vid Berkeley-universitetet under ledning av Dr. Michael Stonebraker. Från samma plats kom en gång i tiden rdbms-produkten Ingres, också med

Stonebraker som huvudansvarig. Stonebraker var en av de mest framstående profilerna under rdbms-eran på 70- och 80-talen. När han så ”växlade spår” in mot ordbms under 80-talet kunde det ske utifrån en sällsynt gedigen kompetensplattform. Den första releasen baserad på Postgres benämndes Montage. Sedermera kom både produkt och lanserande företag att kallas Illustra (av namnrättsliga skäl?).

Illustra har tagit sig in på marknaden genom en kombination av skicklig orientering mot spännande tillämpningsområden (inte minst multimedia-baserade), kompetent ledning och en väletablerad ”guru” som Stonebraker. Företaget Illustra Information Technologies blev i slutet av 1995 uppköpt av Informix.

Produkten tillhör i dagsläget kategorin Ny-Gammal (rdbms) eftersom den baserar sig på en Relational Storage Manager i botten. Mycket möda har lagts på att skapa en funktionalitet med sikte på nya typer av tillämpningar, inte minst multimediabaserade sådana. En följd av detta är möjligheten att definiera nya datatyper samt etableringen av Datablade-konceptet (se vidare avsnitt 3.8). Intensivt arbete bedrivs för närvarande i och för integ-rering av Informix och Illustra. Slutresultatet, med planerad release i slutet av 1996, kommer att kallas Informix-Universal Server och kommer förhoppningsvis att innehålla en mix av det bästa från respektive produkt.

En relativt ny aktör på marknaden är **Omniscience**, Santa Clara, Kalifornien. Företaget som startades 1992 är privatägt. Delar av det ansvariga teamet kommer från odbms-leverantörenVersant. En av huvudägarna arbetade dessutom tillsammans med Won Kim (UniSQL) vid MCC. Funktionsmässigt ligger Omniscience också närmare UniSQL än Illustra, d v s med en hel del avancerade odbms-egenskaper.

OSMOS från Unisys, Mission Viejo, Kalifornien är en sammansmältning av ett antal existerande produkter för att möta en förväntad expansion inom ordbms-sektorn. Bl a bygger OSMOS på SIM, en ER-baserad dbms. Lansering pågår. Produkten har testats i tillämpningsmiljö sedan 1994.

Som tidigare diskuterats inkluderas numer fler och fler ordbms-egenskaper i flertalet av tidigare renodlade odbms. Bland dessa kan exv nämnas **Versant** (Menlo Park, Kalifornien). Genom att integrera sin Storage Manager med UniSQLs multidatabas-gränssnitt UniSQL/M har man åstadkommit en lösning enligt kategorin Ny-Gammal (odbms).

Jasmine är en produkt från Computer Associates som annonseras till hösten 1996. Den representerar en trend snarlik Informix-Illustra såtillvida att Jasmine står för en integrering av CAs OpenIngres och Fujitsus odbms ODBMS2.

Produkten **Persistence** (Persistence Software, San Mateo, Kalifornien) kan på sätt och vis hänföras till kategorin ordbms. Den är funktionellt en odbms gentemot tillämpningen, definierad i C++. Som databas utnyttjas däremot någon existerande rdbms, inte en odbms. Avancerad mappning mellan O-modell och R-modell är en av de mer fram-trädande dragen.

Som synes har samtliga företag utom UniSQL sin hemvist i Kalifornien, huvudsakligen i Silicon Valley. Det är knappast en slump. Där finns en våldsam innovationskraft och kritisk massa av lätt tillgänglig bredd- och djupkompetens. Det var här R-modellen, de första prototyperna och sedermera produkterna såg dagens ljus (IBM Research

Laboratory och University of Berkeley). De stora rdbms-leverantörerna (bl a Oracle, Informix, Sybase, CA-Ingres) har sina huvudkontor och utvecklingsavdelningar i området.

Uppköpet av Illustra visar på de stora rdbms-leverantörernas ökande intresse åt ordbms-hållet. Förmodligen är det snarare kompetens än produkt som köpes. Att inom det egna företaget utveckla en motsvarande kunskap tar tid – och tid är tillsammans med ordbms-kompetens en bristvara. Ett köp av ett innovationsföretag ger flygande start och genererar, om skött på rätt sätt, en pånyttfödelse och vitamin-injektion i det egna företaget, kan man förmoda.

Hur länge kommer UniSQL och Omniscience med flera att stå utanför detta sug?

6.2 Grov karakteristik

Produkterna utvecklas kontinuerligt. En exakt egenskapslista är därför ointressant. Det som inte finns ena månaden kan myckt väl dyka upp nästa. En presumtiv produktköpare bör mycket noga matcha sina krav direkt med ett antal tänkbara leverantörer. Förmodligen är det dessutom minst lika viktigt att titta på företagets allmänna egenskaper (exempelvis omsättning, ledning, rykte, success stories, ...) som på produktens. Med reservation för eventuella felaktigheter eller inaktuella uppgifter redovisas i figur 30 produktkarakteristik under några rubriker.

	attribut	OID	Explicita samband	samband	Egendef. enkla datatyper	Egendef. komplexa datatyper
Illustra	Ja	Ja	Ja	Nej	Ja	Ja
UniSQL	Ja	Ja	Ja	Nej	Nej	?
Odapter	Ja	Ja	Ja	?	Nej	Ja
Omniscience	Ja	Ja	Ja	Nej	Nej	?
OSMOS	Ja	Ja	Ja	Ja	Nej	Nej?
Versant	Ja	Ja	Ja	Ja	Nej	Nej?

	Egendef. funktioner	Data-arv	Funktions-arv	Dynamisk av funktioner	'Callback'-mekanism	Rules, Triggers
Illustra	Ja	Ja	Ja	Ja	Ja	Ja
UniSQL	Ja	Ja	Ja	Ja	Ja	Ja
Odapter	Ja	Ja	Ja	Nej	Nej?	?
Omniscience	Ja	Ja	Ja	?	?	?
OSMOS	Nej?	Ja	Nej?	Nej	Nej?	Nej?
Versant	Ja	Ja	Ja	Nej	Ja?	Nej?

Figur 30

7. Trender

7.1 Nya tillämpningar och tillämpningsområden

Det råder ingen tvekan om att trenden vid nyutvecklade tillämpningar går mot

- Mer komplex funktionalitet
- Komplexa, föränderliga datastrukturer
- Generaliseringsstrukturer
- Distribution och replikering av data och funktion
- Versionshantering av data och funktion
- Långa transaktioner
- Nya typer av data; grafik, röst, video, bild, blobs, m m
- Samverkan mellan fristående system.

Existerande tillämpningsområden förnyas eller omdefinieras. Bland nyare tillämpningsområden där avancerade krav är en självklarhet kan nämnas

- Kommunikation
  - Network Management
  - Kundstöd
  - Multimediahantering
- Processkontroll
  - Produktionsövervakning
  - Samordning av arbetsflöden
- Finansservice
  - Komplexa data, algoritmer
  - ”Prestanda är pengar”
  - Valutahandel
  - Analyser, bedömningar
- Energi, Distribution
  - Styrning, övervakning
  - Kundstöd, fakturering
- Avancerat kontorsstöd
  - Dokumenthantering
  - Workflow-stöd
  - Grupparbete
- Avancerad informationsbearbetning
  - Data Warehouse
  - Beslutsstöd
- Multimedia
  - Avancerad dokumenthantering
  - Virtual reality-tillämpningar
- Datorstödd utveckling
  - Case
  - CAD
- Geografiska Informationssystem (GIS)
- Medicin

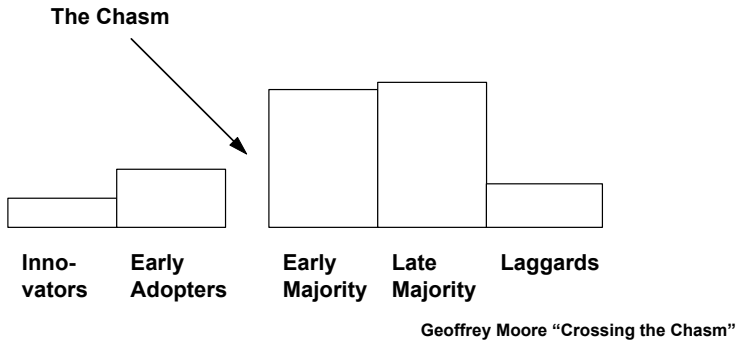
- Internet-orienterade
  - Informationspresentation
  - Informationsåtkomst
  - Databasbasering
  - Dynamiska exekveringsmiljöer
  - Affärsverksamhet

Inte minst den senare miljön är under mycket stark expansion.

Databashantering måste kunna svara upp mot dessa behov både vad gäller modell, gränssnitt och funktionalitet. SQL räcker inte till. Odbms svarar i sin grundideologi upp mot vissa specifika krav. Ordbms har vuxit fram med målsättningen att täppa till SQLs brister för att effektivt kunna svara upp mot ovan nämnda tillämpningsområdens krav. Vad säger då marknaden?

7.2 Marknad; idag

I dagsläget är marknadsförhållandet mellan rdbms och odbms grovt 100 till 1, d v s odbms har drygt 1% av rdbms-omsättningen. Ordbms-marknaden är blygsam men under expansion. Antalet ordbms-produkter är mycket begränsat. De har funnits på marknaden endast en kortare tid. Intresset är mycket stort men avvaktande. Kunderna har fullt upp med att införa rdbms-tillämpningar och underhålla s k ”legacy systems”. Man orkar endast i begränsad utsträckning ta sig an en ny, oprövad teknologi. Teknologin baseras visserligen på en väletablerad bas – SQL – men skillnaderna i modell och gränssnitt är tillräckligt stora för att skapa tveksamhet. Dessutom är avsaknaden av en standard en given hämsko. Följden blir en låsning till valt ordbms. (Å andra sidan är det heller inte alldeles lätt att flytta en tillämpning som utnyttjar en leverantörs utvidgningar av SQL.) Med referens till Geoffrey Moore's välkända figur har ordbms ännu inte passerat ”The chasm”, d v s klyftan över till en väletablerad marknad.



Figur 31

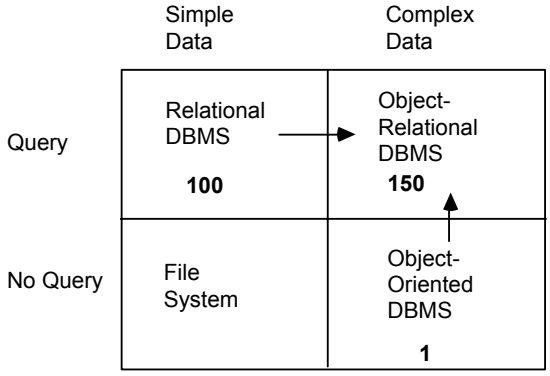
En annan aspekt att väga in är att OR-modellen visserligen ger mer semantisk kraft men till priset av den ursprungliga relationsmodellens enkelhet och elegans – det som mycket aktivt bidrog till dess ursprungliga popularitet. Där finns dessutom en stor och gedigen kompetens som en trygghetsfaktor. Att gå från denna stabila värld till en visserligen mer avancerad men betydligt mer osäker mark kräver mod eller ett starkt behov. Ur ett industriellt perspektiv är det en omöjlighet att hela tiden byta teknologier

och så kallade paradigmer och därmed kasta bort redan gjorda, tunga investeringar i funktionalitet och data. En smidig övergång, där gjorda investeringar kan fortsätta och ha relevans är det enda praktiskt genomförbara alternativet.

7.3 Marknad; trend

I sinom tid, vilken kan ligga närmare än konventionella prognoser förutskickar, kommer en intensiv utveckling av nya tillämpningar (se avsnitt 7.1) och därmed vidhängande krav att bana väg för dbms med en OR-profil. Inte förvånande anser Stonebraker det mesta tala för en mycket klar trend från de övriga dbms-rutorna mot ordbms framöver. Sålunda gör han bedömningen att förhållandet 2005 kommer att vara enligt figur 32. Förhållandet mellan rdbms och odbms kommer att vara ungefär detsamma. Däremot kommer ordbms att vara klart marknadsledande beroende på att de tar över rdbms-tillämpningar och kommer att helt dominera inom nya tillämpningsområden.

Observera, att marknadsviktningen inte nödvändigtvis uttrycker något förändrat styrkeförhållande mellan dagens aktörer på marknaden. Rdbms-leverantörerna strävar aktivt ”åt höger”, odbms-leverantörerna i lika hög grad ”uppåt”. De nyetablerade ordbms-leverantörerna kommer att möta intensiv konkurrens.



Figur 32

Med SQL som en trygg, marknadsetablerad och stringent bas anses inom ordbms den enda realistiska utvecklingsvägen vara att utgå från SQL och komplettera med objekt-orienterade faciliteter. Marknaden önskar evolution, inte revolution. Att bygga på en renodlad O-modell anses av samma skäl ha små chanser att lyckas på marknaden, i alla händelser på kort sikt. Odbms-företrädarna har i princip börjat acceptera detta synsätt. Eftersom man inte har för avsikt att kompromissa in relationmodellens begrepp i sin O-modell återstår för dem att så långt möjligt inkludera SQLs syntaktiska konstruktioner i objekt-frågespråket – allt för att användarna ska ”känna igen sig”. Andra anser att denna enträgna strävan efter SQL-kompatibilitet inom alla läger är ett hinder för områdets utveckling. SQL är avpassat för R-modellen. O-modellen har helt andra egenskaper som möjliggör helt andra uttryckssätt. Dessa kan formuleras i betydligt elegantare språkkonstruktioner än SQL.

I dagsläget vilar dock SQL som en ande, om än i olika varianter, över alla rutor utom den nedre vänstra.

Det förtjänar påpekas att alla varianter av dbms, må det vara rdbms, odbms eller ordbms, har sina starka och svaga sidor. De vänder sig till delvis olika tillämpningsområden. Där finns tillämpningar ”som gjorda för” odbms och antagligen mindre lämpliga att realisera i ett rdbms eller ordbms. Där finns tillämpningar som fungerar alldeles utmärkt i rdbms-miljö, inte minst p g a dess enkla gränssnitt och stabila, omgivande funktionalitetsstöd,
t ex för transaktionshantering och behörighet. Eftersom ordbms i allmänhet har ett gränssnitt som är ett superset av SQL kan man förmoda att även typiska rdbms-tillämpningar i framtiden i större utsträckning kommer att utvecklas med stöd av ordbms. Dock först när dessa fått en motsvarande produktstabilitet.

Observera, att en framtidsbetonad tillämpning mycket väl kan behöva olika databasstöd – kanske från både odbms, ordbms och rdbms. Ta exv följande tänkta tillämpningar, skisserade i Stonebrakers bok ”Object-Relational DBMSs – the next great wawe”, här återgivna i sammandrag:

> Den första tillämpningen är tänkt för en nästa generations ”videobutik”. Tillämpningen har att leverera filmer till kunder samt hantera diverse annan information av intresse om filmer, uthyrningsvillkor, kunduppgifter, m m. Själva leveransen av en film tillhör naturligt nedre vänstra rutan eftersom det sannolikt är fråga om att leverera en film som ligger i en fil snarare än i en databas. Att söka uppgifter om en film kan innebära sökning i ett antal tabeller med lämpliga joins, d v s en funktion som hör till den övre vänstra rutan. Vill man söka ut de filmer som i någon av dess lagrade filmrecensioner (i form av dokument) har ordet ”fantastisk” innebär det en funktion mot ett dokumentattribut, något som kan utföras i den högra, övre rutan eftersom nya datatyper (dokument och funktioner på dessa) kan hanteras. Datatypen dokument kan ha en intern struktur avpassad efter de funktioner som har att operera på den. Operationer är uttryckbara i det utökade SQL. I klient/servermiljö undviks onödig datatransport eftersom operationen kan utföras på servern i nära anslutning till databasen. (Å andra sidan har allt detta väldigt lite med objektorientering att göra.)
>
> Ett annat exempel är en försäkringstillämpning med försäkringstagare, vad som försäkrats, div ekonomiska uppgifter, ersättningsanspråk över tiden, m m. Mycket av datahanteringen kan här ske i en vanlig rdbms. Men önskar man tillföra och operera på bilder över händelser (exv jämförelse av bilder på bildeformation per hastighet, biltyp, olyckstyp), platsangivelser i form av koordinater, matchning av försvunna föremål, m m kommer dels nya datatyper in i bilden, dels ett antal funktioner på dessa. Kan dessa deklareras in i tillämpningens schema kan de sedan användas i SQL-uttryck av olika slag. Dessa datatypers interna uppbyggnad och accessprinciper kan specialanpassas efter de tillhörande funktionerna för optimal prestanda. En finess är att nya funktioner i OR-miljö kan deklareras och initieras utan att först behöva ”nudda” tillämpningen med tillhörande omkompileringar, länkningar m m.

I den snabba Web-expansionens spår bedöms mycket snart följa en intensiv expansion av multimedial, objektbaserad databassupport. Dels för användarrelaterade behov på servern, dels för avancerad, global informationshantering. Databashantering verkar här stå inför helt nya utmaningar och nya marknader. Långsiktiga tidplaner kan plötsligt bli omkullkastade och bedömda orealistiskt tröga. Företagen positionerar sig just nu för att inte ”missa tåget”. Ett lika plötsligt som angeläget behov av avancerad ordbms-kompetens tycks förestående. Kanske var det därför ryktena talar om en mycket hög köpesumma för Illustra, ca 300 miljoner dollar, för ett företag med en omsättning kring 10 miljoner dollar (Software Magazine, May 1996).

När väl de större rdbms-leverantörerna kliver in på marknaden kommer läget snabbt att förändras. Dels blir OR-ansatsen ”rumsren”, dels skapas ett antal ”ringar på vattnet” i form av stödprodukter av olika slag från många olika leverantörer. De facto-standarder kommer att etableras, osäkerheten kommer att minska. Många leverantörer planerar kraftigt uppdaterade releaser under 1997. Blir det året starten för en intensiv expansion?

En hämmande faktor är givetvis bristen på tillräckligt djup och spridd kompetens kring modell, produkter och anpassade utvecklingsmetoder. Denna brist åtgärdas inte i brådrasket. Å andra sidan kan exv en ”killer application” i anslutning till Internet skapa underverk när det gäller snabb kunskapsuppbyggnad och intensiv kreativitet. Gamla sanningar om tioårsspann mellan forskning och marknadsetablering håller på att vittra sönder, i alla händelser i anslutning till Internet. Kanske kommer även databasområdet snart in i denna turbulenta dans. Eller är allt bara en upptrissad suggestion?

8. Sammanfattning

De flesta ordbms-leverantörer arbetar efter sin begreppsmodell. Alltså finns ingen mall för att avgöra om ett system är att anse som ordbms eller ej. SQL3 är dock basen, katalysatorn, inspirationskällan för en kursändring mot objektorienterat inflytande inom rdbms-området. De flesta ordbms-leverantörer deltar också aktivt i SQL3-arbetet. Tyvärr är SQL3 långt ifrån en standard. Dess långsamma utvecklings-takt kommer att fortsätta generera en vildvuxen flora av olika egenskaper i olika produkter, eftersom marknaden tycks ha valt att inte vänta på standarden. Kunderna är heller inte ännu några kraftfulla kravställare.

Antagligen är det synnerligen instabila läget en orsak till att de stora dbms-leverantörerna avvaktar insteget i ordbms-världen. Att ”bränna sig” där inför den existerande kund-kretsen kan vara förödande. Alldeles givet är däremot att de stora rdbms-leverantörerna kommer att infoga fler OR-faciliteter i sina produkter när marknaden så kräver. Balansen mellan konventinell SQL-bas och nya finesser kräver fingertoppskänsla. För närvarande nöjer man sig med att markera revir genom mindre tillägg och genom att förutskicka OR-egenskaper i kommande releaser. Detta läge kan dock snabbt förändras när kundtrycket ökar, i och med en stor leverantörs intåg på marknaden eller genom att nya Internet-baserade behov kullkastar alla prognoser och planer. Den långsiktiga trenden mot hybrider är därmed given.
Bara tidtabellen är fortfarande något oklar.

Följdfrågan blir givetvis om existerande mindre leverantörer på sikt ska kunna hävda sig. En förutsättning är att man kan rida på sitt momentum genom att vara tidigt ute och antagligen representera en hel del unik kompetens alternativt ha en stark position inom viss bransch. Med denna kapacitet kan företag bryta ny mark alternativt ta andras om man agerar snabbt. Historien visar att det knappast är långsiktigt möjligt, såvida man inte snabbt blir stor, gärna hela tiden är innovativt överlägsen, samtidigt som man har ”näsa” för marknadens behov. Att skapa upprepade succéer är få förunnat. Med detta konstaterande ligger det nära till hands att vidare konstatera att där finns ett kortsiktigt vaccum att positionera sig inom för att utöka överlevnadspotentialen. Det sannolika är annars att kompetensen kommer att införlivas i de stora bolagen. Så har för övrigt redan skett med Illustra.

Nyanserade uppfattningar hävdar att det kommer att finnas utrymme för både rdbms, odbms och ordbms. Den långsiktiga trenden ligger inte i *antingen eller* utan i *både och*. Respektive produktprofil uppvisar olika egenskaper syftande mot delvis olika tillämpningsområden. Samverkan mellan tillämpningar baserade på olika modeller blir visserligen svårare. Å andra sidan kommer sådana diskrepanser alltid att vara en realitet i de flesta verksamheter. Tillämpningar utvecklas vid olika tidpunkter, under olika förutsättningar, för olika ändamål. Så kallade legacy systems kommer alltid att finnas. Att ställa förhoppningen till en slutlig, ensad modell är att hoppas på magiska krafter.

Mycket av ”röran” kring den aktuella ordbms-utvecklingen skulle aldrig behövt uppstå om dbms baserade på någon S-modell på ett tidigt stadium (1970-talet) blivit accepterade på marknaden. Man kan för övrigt undra om det långsiktigt är

vettigt att hänga kvar vid SQL eller om ett mer anpassat språk vore bättre att satsa på. Det är inte osannolikt att vi i ett längre perspektiv får se framväxten av ett standardiserat språk specifikt anpassat till den underliggande modellen. Kanske borde OQL vara en större inspirationskälla?

Med etableringen av multimedia i databaser kommer vanliga databasers volym att mångdubblas, förmodligen till volymer som i dagsläget ter sig osannolika. Förmodligen kommer vissa utrymmeskrävande multimediaelement (bild, animering, video, ...) att behöva hanteras inom ett något billigare medium än ett direktaccessminne, i alla händelser under en övergångstid. Dagens enkla frågor kommer att ersättas med betydligt mer strukturellt komplexa. Dessutom kommer en mycket stor portion av exekveringskraften att gå åt till innehållsmässig behandling av multi-mediaelement (exv ”finn alla bilder med röda, fyradörrars bilar med spoiler i avvikande färg, tagna snett framifrån”), snarare än vanliga data. Avancerade beslutsstödstillämpningar i Data Warehouse-miljö kommer att borga för detta.

Multimedia, Internet, komplexa informationsstrukturer, avancerad informationsbearbetning, m m skapar tillsammans utsökta framtidsutsikter för ordbms!