Modern databasteknik för underrättelsehantering i det nätverksbaserade försvaret

En introduktion

PER SVENSSON

FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1000 anställda varav ungefär 800 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömning av olika typer av hot, system för ledning och hantering av kriser, skydd mot och hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Informationssystem
164 90 Stockholm

Tel: 08-55 50 30 00
Fax: 08-55 50 31 00

www.foi.se

FOI-R--2568--SE Användarrapport
ISSN 1650-1942 September 2008

Informationssystem

Per Svensson

Modern databasteknik för underrättelsehantering i det nätverksbaserade försvaret

En introduktion

Titel	**Modern databasteknik för underrättelsehantering i det nätverksbaserade försvaret – en introduktion**
Title	Modern database technology for intelligence management in a Network-Enabled Defence – an introduction
Rapportnr/Report no	FOI-R—2568--SE
Rapporttyp Report Type	Användarrapport User report
Sidor/Pages	52 p
Månad/Month	September
Utgivningsår/Year	2008
ISSN	ISSN 1650-1942
Kund/Customer	FM
Forskningsområde Programme area	7. Ledning med MSI 7. C4I
Delområde Subcategory	
Projektnr/Project no	E 7113
Godkänd av/Approved by	Martin Rantzer

FOI, Totalförsvarets Forskningsinstitut
Avdelningen för Informationssystem

164 90 Stockholm

FOI, Swedish Defence Research Agency
Information Systems

SE-164 90 Stockholm

Sammanfattning

Denna rapport har tillkommit på uppdrag av FOI-projektet "*Situations- och hotanalys för Nordic Battle Group (NBG) 2011*". Syftet har främst varit att bidra till den kunskapsuppbyggnad som krävs för att FM ska kunna skapa en effektiv informationshanterings- och analysprocess för framtida stridsgruppers underrättelsefunktion.

Allt fler tillämpningsområden har successivt kommit inom räckhåll för databastekniken, samtidigt som användarnas krav på lagrings- och åtkomstkapacitet, driftssäkerhet och driftsekonomi har blivit lättare att tillgodose. Det sistnämnda beror främst på den exponentiella ökningen av datorers prestanda och den samtidiga pris- och storleksreduktionen.

Valet av lagringsteknik, antingen den utnyttjar vanliga filer eller det långt mer avancerade databaskonceptet, är idag ofta mindre kritiskt för en tillämpning än det var när databashanterarna började införas omkring 1970. Det är dock inte databasteknikens förmåga att snabbt och effektivt hantera mycket stora datamängder som är dess avgörande fördel, utan den mycket högre abstraktionsnivån och funktionaliteten, jämfört med vanlig programstyrd bearbetning och filhantering.

Sedan 80-talet har Codd's *relationsdatamodell* dominerat inom den administrativa databehandlingen, men *objektorienterade* och *objektrelationella* modeller har också lanserats, den sistnämnda kategorin som en syntes av de två förstnämnda. Objektorienterade databashanterare skaffade sig en nisch under 90-talet, främst inom tekniska konstruktionstillämpningar för vilka relationsdatabaserna inte hade den funktionalitet och kapacitet som krävdes.

En databashanterare ska effektivt och säkert kunna hantera mycket stora datamängder. I många tillämpningar är förmåga att hantera ett stort antal samtidiga förfrågningar ett huvudkrav, s k *On-Line Transaction Processing* (OLTP). Men i analys- och beslutsstödstillämpningar kan konventionella tekniker för konfigurering av databassystem inte användas, eftersom de inte kan hantera alla de frågeställningar som är typiska för sådana tillämpningar, som t ex dataurval, hantering av temporala och aggregerade data samt kontrollerat utnyttjande av redundant lagring. Användning av s k *datalager* har därför blivit en viktig strategi för att integrera heterogena datakällor och för att möjliggöra s k *On-Line Analytic Processing* (OLAP).

Nyckelord:Underrättelsehantering, Analysstöd, Beslutsstöd,Vertikala databaser

Summary

This report was commissioned by the FOI research project ”*Situation and threat analysis for the Nordic Battle Group 2011*”. The objective of this work has mainly been to contribute to the knowledge build-up required to allow the Swedish Armed Forces to create an effective information management and analysis process for the intelligence function of the Battle Group.

Ever more areas of application have successively come within reach of database technology, while users’ requirements for storage and access capacity, as well as reliability and economy of processing, have become easier to satisfy. The latter is mainly due to the exponential increase of the performance of computers and the concurrent reduction in their size and cost of acquisition.

The choice of storage technology, whether it is based on ordinary files or on the far more advanced database concept, is often much less critical today than when database management systems were introduced around 1970. It is however not its capability of fast and efficient management of very large data sets that is the key advantage of database technology, but rather its much higher level of abstraction and far greater functionality compared to conventional application-controlled data processing and file management.

Since the 1980’s the *relational model of data* introduced by E. F. Codd has dominated business data management, but *object-oriented* and *object-relational* models have also been introduced, the latter category as a synthesis of the two first-mentioned.

Object-oriented database management systems created a market niche for themselves during the 90’s, predominantly in technological design applications for which relational databases lacked the functionality and structuring capacity that was required.

A database management system must be able to handle very large sets of data. In many applications, capability to manage a large number of concurrent requests involving updates is a key requirement, so-called *On-Line Transaction Processing* (OLTP). In analysis and decision support applications, however, conventional techniques for database configuration can not be used, since they are unable to resolve all the issues that are characteristic of such applications, such as data selection, management of temporal and aggregated data and the controlled use of redundant storage. The use of so-called *data warehouses* has therefore become an important strategy to integrate heterogeneous data sources and to enable so-called *On-Line Analytic Processing* (OLAP).

Keywords: Intelligence management, Analysis support, Decision support, Vertical databases

Innehållsförteckning

Förord

Denna rapport har tillkommit på uppdrag av FOI-projektet "*Situations- och hotanalys för Nordic Battle Group (NBG) 2011*". Syftet har främst varit att bidra till den kunskapsuppbyggnad som krävs för att FM ska kunna skapa en effektiv informationshanterings- och analysprocess för stridsgruppens underrättelsefunktion. En välkommen bieffekt skulle vara att höja FOIs och FMVs allmänna kompetensnivå inom databasteknik, ett område av informationstekniken som kanske inte längre uppfattas som särskilt ”hett” eller trendigt, men som i verkligheten spelar fundamentala roller i väldigt många komplexa tillämpningar, inte minst inom underrättelseområdet.

I början av den här studien ägnade jag mest intresse åt objektorienterade databaser, främst därför att de till skillnad från klassiska relationsdatabaser erbjuder stor flexibilitet när det gäller att utforma databasens datastrukturer, ett krav som vi bedömt vara viktigt i många underrättelsetillämpningar. Eftersom jag ägnat en stor del av mitt forskarliv åt att bygga, testa och använda relationsdatabassystem [Sve79, KS83, KS86, BSS97, FMS+08], men också att då och då presentera dem i mer översiktliga eller ”populära” former [Sve78, Sve88, SH08, SBI+09], utgör väl dessa inte heller samma frestelse för min nyfikenhet.

Men våra erfarenheter från EU-projektet HiTS-ISAC, som avslutades våren 2008, pekade på att det finns andra möjligheter än att bygga ett objektorienterat databassystem från grunden för att passa tillämpningsområdet och bl a tillhandahålla effektiv representation av komplexa nätverksstrukturer. Efter en tid drog jag slutsatsen att det bästa sättet att använda den tillgängliga tiden nog skulle vara att beskriva nuläget inom databasområdet ganska opartiskt, men med tonvikt på analytiska tillämpningar.

Jag har använt flera olika texter som underlag, i första hand [SBI+09, FMS+08, SH08, FNP+00], och kompletterat dem med erfarenheter och reflexioner från olika databasintensiva forsknings- och utvecklingsprojekt. Det är min förhoppning att denna FOI-rapport blir till hjälp och nytta i utvecklingsarbetet för NBG 2011.

Till sist vill jag tacka mina kolleger Christian Mårtenson och Pontus Svenson för deras granskning av manuskriptet, som gjort att texten kunnat förbättras och förenklas på flera sätt.

Per Svensson

1 Inledning

Tillsammans med förmåga att utföra logisk och aritmetisk databearbetning, är förmåga att lagra och återfinna data en fundamental egenskap hos alla processer för informationsbehandling. I engelsmannen Charles Babbage's idémässigt nyskapande men snabbt bortglömda mekaniska datorkonstruktion *Analytical engine* från mitten av 1800-talet lagrades data genom komplexa och sinnrika mekaniska länksystem som kunde inta ett antal bestämda (diskreta) lägen, motsvarande olika siffror eller logiska tillstånd.

Under och efter andra världskriget skedde så en serie tekniska genombrott när informationsbehandlingsprocesser började kunna göras helautomatiska och baserade på elektronik, med processbeskrivningar, *program*, lagrade som *mjukvara* på samma sätt som processdata. Man hade uppfunnit nya sätt att *representera* eller *koda* bearbetningsprocesser som teckenföljder. Därigenom öppnades möjligheter att realisera informationsbehandlingsprocesser med först tiotusentals, ett par decennier senare miljontals och idag miljardtals gånger både större *och* snabbare datalager och bearbetningsprocesser än de mänskliga rutiner, stödda på relativt enkla mekaniska räknehjälpmedel, som varit förhärskande under första hälften av 1900-talet.

För att kunna spara och överföra information mellan olika bearbetningsprocesser, och för att kunna ta emot och leverera indata och resultat till processerna, utvecklades efterhand en lång rad tekniska lösningar, först realiserade som hål i remsor eller kort gjorda av papper, senare som en växande familj apparater baserade på magnetomekaniska, optomekaniska eller halvledarelektroniska principer. Ur användarsynpunkt är detaljutformningen av dessa numera extremt komplexa men ändå billiga tekniska lösningar av sekundärt intresse. Det som i första hand betyder något för användaren är i stället hur snabbt och i vilken ordning som data kan läsas och skrivas på lagringsmediet. Naturligtvis är det också av stor praktisk betydelse hur stor lagringsenheten är, om den är flyttbar eller stationär, hur mycket energi den kräver, och hur mycket den kostar.

I dessa avseenden har utvecklingen sedan 40-talet varit utan teknikhistoriskt motstycke, och den som då vågat påstå att vanliga privatpersoner i början av 2000-talet skulle kunna lagra 64 miljarder tecken (som är dagens rekordnotering, till ett pris av ca 2000:-), tillräckligt för att lagra text och bild för en hel encyklopedi[1], i en löstagbar, batterilös "pinne" mindre än ett lillfinger, och att mer än tio miljoner tecken skulle kunna läsas från och skrivas till pinnen på en sekund, skulle blivit utskrattad och hans idéer betraktade som ren science fiction.

Yttre minne organiseras för användarprogrammen av operativsystemet i *filstrukturer*, varifrån data läses in i och skrivs ut från användarprogrammet med hjälp av läs- och skrivprocesser, i vanliga programspråk oftast åtkomliga via speciella *läs- och skrivsatser*. Sådana läs- och skrivsatser läser/skriver oftast en av användaren definierad "post" eller "rad" i filen varje gång den anropas.

[1] Den svenska Nationalencyklopedin innehåller uppskattningsvis 2 bytes/tecken X 35 tecken/rad X 80 rader/sida X 3 kolumner/sida X 600 sidor/band X 30 band = 300 MB text och, med en grov gissning, ca 20000 bilder på i genomsnitt 3 MB = 60GB bilddata; bildernas utrymmesbehov är alltså helt dominerande.

Därefter flyttar operativsystemet en dold pekare till nästa post, som därför står i tur nästa gång läs/skrivsatsen utförs av programmet. En filstruktur som arbetar på detta sätt kallas *sekvensiell*. Sekvensiell läsning/skrivning i filstrukturer är en för användaren okomplicerad process som passar perfekt om man enbart vill läsa in en post i taget från en fil i den ordning de ligger på det yttre minnet.

Den teknik för yttre minne som hängt med i teknikutvecklingen längst av alla och fortfarande är vanligast är det magnetiska *skivminnet*. Idag finns sådana med total kapacitet 1 TB (1 *terabyte*, dvs. 1 biljon bytes om vardera 8 bitar). Som jämförelse kan nämnas att en av IBMs mest framgångsrika produkter för ca 40 år sedan var skivminnet IBM 2314, stor som en dubbelsidig golvbokhylla med en lagringskapacitet på ca 230 MB (1 megabyte motsvarar 1 miljon bytes). Ett skivminne som kan lagra mer än 4000 sådana enheter får alltså några decennier senare lätt plats i handflatan och kostar några tusenlappar...

Datorns *primärminne* adresseras däremot direkt av processorn ända ner på teckennivå och organiseras tillfälligt och momentant, dvs. enbart under den tid programmet är aktivt, av olika tillämpningsprogram som statiska eller dynamiska *datastrukturer.*

Mot denna historiska bakgrund är det kanske lättare att förstå att valet av lagringsteknik, antingen den utnyttjar vanliga filer eller det långt mer avancerade databaskonceptet, idag i regel är mindre kritiskt för en tillämpning än det var när databashanterarna började införas omkring 1970. Man kan idag i sin egen bärbara dator på mycket kort tid läsa in ett stort datamaterial till ett standardprogram (t.ex. ett *kalkylprogram* som MS Excel) som på någon eller några sekunder utför de sökningar och beräkningar man vill ha gjorda. Frågan om databasteknik överhuvud taget behövs i dagens IT-system ställs därför inte så sällan numera.

2 Databasteknik

Som vi sett möter man idag ganska ofta uppfattningen att datorers lagrings- och bearbetningskapacitet numera är så stor att databasteknik inte behövs, och att det går utmärkt att lagra även stora mängder data i filer som hanteras av datorns operativsystem. Men det är inte databasteknikens förmåga att snabbt och effektivt hantera mycket stora datamängder som är dess avgörande fördel, utan den mycket högre abstraktionsnivån och funktionaliteten, jämfört med vanlig programstyrd bearbetning och filhantering. En databashanterare bör vara baserad på en välgrundad generell och abstrakt teori (ofta, med ett ganska olyckligt ordval, helt kort kallad "modell" eller "datamodell") för hur data och information ska beskrivas och struktureras. Ett viktigt problemområde var från början hur man skulle utforma datamängders struktur och egenskaper på ett sätt som möjliggör en effektiv kompromiss mellan de logiskt/funktionella och prestandamässiga användarkraven, och samtidigt ta hänsyn till förmågor och begränsningar hos de maskinvarutekniska metoder som efterhand skapades för att möta dessa prestandakrav.

Allt fler tillämpningsområden har successivt kommit inom räckhåll för databastekniken, samtidigt som användarnas krav på lagrings- och åtkomstkapacitet, driftssäkerhet och driftsekonomi har blivit lättare att tillgodose. Detta beror främst på den i inledningen illustrerade exponentiella ökningen av datorers prestanda och den samtidiga pris- och storleksreduktionen.

Den förhållandevis långsamma, gradvisa men systematiska tekniska utvecklingen av databassystemens konstruktion och realisering får allt större betydelse när man vill täcka in nya tillämpningsområden vars kravbild inte nödvändigtvis liknar den som gäller för konventionella transaktionshanteringstillämpningar, som platsbokning på flyg eller uttag från bankkonton.

Sedan 80-talet har Codd's *relationsdatamodell* [Cod70, EN06] dominerat inom den administrativa databehandlingen, men *objektorienterade* [Cat94, CZ01] och *objektrelationella* [SMB99] modeller har också lanserats, den sistnämnda kategorin som en syntes av de två förstnämnda. Objektorienterade databashanterare skaffade sig en nisch under 90-talet, främst inom områdena CAD/CAM (*Computer-Aided Design* / *Computer-Aided Manufacturing*) och CASE (*Computer-Aided Software Engineering*), tekniska konstruktionstillämpningar för vilka relationsdatabaserna inte hade den funktionalitet och kapacitet som krävdes.

Inte minst för geografiska och andra rumsliga (*spatiella*) tillämpningar, där algoritmer och lagringsstrukturer kan vara mycket komplexa, visar det sig att en långt driven standardisering och abstrahering av databasernas uppbyggnad i många fall kan ge stora fördelar.

En databashanterare ska effektivt och säkert kunna hantera mycket stora datamängder, endast begränsat av tillgången på fysisk lagringskapacitet och intensiteten hos åtkomsttrafiken. I många tillämpningar är förmåga att hantera ett stort antal samtidiga förfrågningar ett huvudkrav. Men storlek och driftsekonomi är förstås relativa begrepp, och även om många verksamhetskritiska databastillämpningar har vuxit snabbt sedan början av 80-talet har pris/prestandautvecklingen för datorer och minnen i de allra flesta fall skett

ännu mycket snabbare och därmed dämpat det tekniska innovationstrycket på databassystemen.

2.1 Vad skiljer lagring i databaser från lagring i filer?

Vid lagring av data i ett filsystem har programmen direkt åtkomst av data i filerna. Filsystemet är vanligen egenutvecklat och dess detaljutformning inte sällan känd av en enda programmerare. Detta är en enkel lösning som kan vara användbar i mindre tillämpningar när datamängderna är små och det finns få användare av data. Men att förlita sig helt på filhantering i mer omfattande och komplexa tillämpningar visar sig snart vara ekonomiskt och tekniskt ohållbart. Därför har lagring av data i filsystem i komplexa tillämpningar sedan länge övergivits av de flesta stora organisationer.

I de fall data lagras i en databas har användarprogrammet inte direkt tillgång till filerna utan kan bara kommunicera med en *databashanterare* (*database management system*, DBMS). En databashanterare är en programvara som är konstruerad och uppbyggd för att kunna hantera de tekniska aspekterna av förvaltning av databaser och erbjuda deras användare en rad olika generellt användbara tjänster. Den är en generell programprodukt, byggd för att i samverkan med datorns operativsystem definiera och realisera hur komplexa, ofta verksamhetskritiska datamängder ska struktureras, beskrivas, utfrågas och uppdateras av tillämpningsprogram eller tillfälliga användare. Det ska också skydda dessa datamängder mot accesskonflikter och driftsstörningar, allt med krav på god driftsekonomi och goda prestanda, inte minst i form av korta svarstider. Hög driftssäkerhet och god driftsekonomi är särskilt viktigt för datamängder som är mycket stora, har många samtidiga användare och är ständigt utsatta för förändringar och förfrågningar. Tidigt aktuella exempel var centrala register över kundtransaktioner inom banker och försäkringsbolag, eller personal- och kundregister inom stora företag.

Centraliserad lagring av data i form av en integrerad databaslösning möjliggör att flera personer arbetar med data samtidigt. Ett problem blir då att säkerställa att inte flera personer uppdaterar samma data samtidigt, vilket skulle leda till konsistensproblem för databasen. Kravet på korrekt hantering av frågor och uppdateringar från flera samtidiga användare har haft stor betydelse för hur databastekniken utformats - i en situation där hundratals eller tusentals samtidiga användare av t ex ett bankkontosystem eller ett platsbokningssystem för flygresor oavbrutet modifierar datamängdens innehåll måste man ständigt kunna garantera databasens korrekthet. Sådana krav på *korrekt transaktionshantering* måste ställas på alla datalagringssystem som hanterar uppdateringar och tillåter flera samtidiga användare (transaktion är en operation som görs mot databasen, t ex en uppdatering av innehållet).

Ett större företag i dagligvarubranschen behöver normalt för sin bokföring hålla reda på, för varje kund i varje försäljningskassa, vilka varor och förpackningar kunden köpt, i vilken kvantitet och till vilket pris. Denna information skickas normalt i realtid från kassaterminalen till den operativa databas som försörjer ett eller en hel grupp av varuhus, och det är inte ovanligt att ett större företags operativa databas får behandla flera hundra transaktioner varje sekund.

De operativa databaserna är strukturerade och optimerade för att registrera alla slags affärsaktiviteter på mikronivå på ett ur redovisningssynpunkt korrekt sätt. Men företagsledningen som behöver veta hur försäljningen går i olika delar av verksamheten för att kunna planera och styra hela företagets verksamhet, inklusive inköp, reklam, personalförsörjning, kapitalförsörjning, kapitalmarknadskommunikation etc, kan i regel inte utnyttja dem direkt för sin planering och styrning. Därför är det vanligt att man speciellt för dessa användare skapar särskilda s k *datalager* vars information inte är minutaktuell, utan uppdateras från de operativa databaserna t ex en gång i veckan.

Databassystem för s k *On-Line Transaction Processing* (OLTP) är alltså olämpliga för beslutsstödstillämpningar och bredbandiga nätverk kan inte ensamma lösa problemet att göra rätt information tillgänglig för beslut. Konventionella tekniker för konfigurering av databassystem kan inte heller användas eftersom de inte kan hantera alla de frågeställningar som är typiska för datalager, som dataurval, hantering av temporala och aggregerade data samt kontrollerat utnyttjande av redundant lagring. Användning av datalager har därför blivit en viktig strategi för att integrera heterogena datakällor och för att möjliggöra s k *On-Line Analytic Processing* (OLAP).

2.2 Allmänna krav på databassystem

Här följer en lista över allmänna krav på konventionella databassystem (CDBMS). De flesta av dessa krav har slagits fast som normer för kommersiella databassystem redan i början av 80-talet och är därför inte med säkerhet vare sig nödvändiga eller tillräckliga för varje tänkbar tillämpning. En mer ingående diskussion av motiv för och krav på databassystem finns i första kapitlet av [EN06], en bok på vars mer än 1100 sidor man kan hitta information om det mesta som rör databasteknik:

- Stort, persistent (beständigt) lagringsutrymme för dataposter.
- Ett *schema* som beskriver viktiga egenskaper hos alla lagrade data. Schemats beskrivning kallas ofta *metadata* (data om data). Användare måste kunna ta del av schemat som persistent (meta)data, och kunna lägga till, ändra och ta bort bl a tabeller, attribut och nycklar i schemat.
- Frågespråk – inte alltid nödvändigt men alltid bra att ha. Kompatibilitet med SQL behövs inte alltid, andra lösningar existerar numera (XQuery, RDQL, SPARQL,…).
- Exekvering av "*triggers*" – viktigt i realtidstillämpningar, kontorsautomation och vissa konstruktionstillämpningar. En trigger är en triplett (Händelse, Villkor, Åtgärd) sådan att den angivna åtgärden utlöses om det givna villkoret är uppfyllt i databasen när händelsen (t ex en viss uppdateringsoperation) inträffar. Eftersom villkorets form enbart begränsas av databasens struktur och innehåll och åtgärden enbart av den omgivande programmeringsmiljön är detta en generellt användbar mekanism.
- Finkornig samtidighet – ofta ett krav i konventionella affärstillämpningar av OLTP-karaktär. I konstruktionstillämpningar hämtar användare däremot ofta ut en stor delmängd av data under en lång tid och man kan inte tillåta den prestandaförlust som det innebär att låsa alla berörda individuella datakomponenter för varje läs- och skrivoperation. Men ofta begärda data ("heta fläckar"), t ex bibliotek av delade komponenter i en konstruktionstillämpning, kan fortfarande vara i behov av finkornig låsning vid uppdatering.

- Automatisk återställning efter en krasch är en önskvärd egenskap hos alla tillämpningar. I vissa konstruktionstillämpningar är det möjligt att återgå till en tidigare version av det aktuella objektet eller den aktuella databasen i stället.
- Dataoberoende – i nästan alla tillämpningar är det önskvärt att kunna byta ut de data som lagrats i databasen utan att behöva förändra existerande program. I relationsdatabaser åstadkoms sådant dataoberoende delvis med hjälp av vyer. I objektorienterade databassystem (ODBMS) uppnås denna egenskap vanligen med hjälp av procedurer (”metoder”) som är de enda utifrån synliga attributen hos objekt.
- Blanketthanterare – de flesta CDBMS tillhandahåller systemspecifika verktyg för skärmdesign och dataformatkontroll. Flexibla formathanterare bygger idag i regel på XML, JAVA etc. och ibland på HTML.
- Distribuerade databaser – distribuerad förmåga, där multipla databaser på flera noder ter sig för användarna som en enda databas, är värdefull i många slags tillämpningar.

2.3 Frågespråk

En viktig del av datamodellen utgörs av dess *frågespråk*, som definierar en uppsättning datatyper och operationer, tillsammans med grammatiska sammansättningsregler för dessa. Frågespråket, som i regel kan användas både direkt av en användare och som ett språkgränssnitt mellan tillämpningsprogram och databas, avgränsar vilka slags frågor som är möjliga att ställa mot systemet, och uppsättningen av datatyper avgränsar vilka slags objekt som systemet kan hantera. De frågespråksprinciper som är en integrerad del av relationsdatamodellen har utan tvekan bidragit starkt till modellens popularitet bland databasforskare och andra specialister, och därmed åtminstone indirekt till dess tekniska och kommersiella dominans. Objektorienterade och objektrelationella system erbjuder *öppna* eller *utbyggbara* typdefinitioner, som gör det möjligt för användaren, eller snarare för särskilt tekniskt bevandrade användarkategorier, att utöka systemets repertoar av objekttyper och operationer på dessa. En svaghet hos många tidiga objektdatabaser var att de saknade frågespråk.

Särskilt i analytiska tillämpningar, och inte minst för spatiella analyser, är ett kraftfullt och flexibelt frågespråk ett värdefullt verktyg. Utan tillgång till ett sådant blir formulering och tolkning av komplexa dataanalyser en resurskrävande verksamhet, dessutom särskilt svår att genomföra i form av projekt där flera analytiker samarbetar.

Frågespråket måste förstås kunna tolkas av systemet. En stor mängd ofta sofistikerade metoder för dels *frågeoptimering*, dels *sökprocesser* av olika slag har efterhand utvecklats. I vissa avseenden är dock människans förmåga att förstå och avväga olika tillämpningars behov fortfarande överlägsen automatiska optimeringsmetoder, och i större organisationer behövs särskilda *databasadministratörer* som har till uppgift att se till att databasdriften sker så effektivt och störningsfritt som möjligt.

2.4 Index

En viktig egenskap hos en databashanterare är att den ska stödja snabb sökning i data. För att tillgodose detta krav används *index*. Ett index är en extra datastruktur som möjliggör mer effektiva sökprocesser men som enbart innehåller ur tillämpningssynpunkt redundant information, liksom ett index i en bok inte

innehåller någon ny information utan enbart gör det lättare att hitta informationen i boken. Och på samma sätt som index i en bok, så är indexen i en databas strukturerade på speciella sätt (t ex ordnade i bokstavsordning eller efter ökande numeriska värden) för att möjliggöra snabb sökning.

Användning av index har också nackdelar, framför allt därför att indexstrukturer kräver extra utrymme i databasen och att attribut som är indexerade kräver längre tid för uppdateringar. Därför bör endast attribut som kommer att användas som argument i ett sökuttryck indexeras.

3 Relationsdatabaser

En relationsdatabas är i regel strukturerad i tre nivåer: *intern*, *konceptuell* och *extern*. Den *interna* nivån beskriver hur data fysiskt lagras hos databasen, hur data är länkade till varandra, och hur olika åtkomstvägar och index som systemet använder för att snabbt hitta eftersökta fakta är utformade. Vanligen är det bara databasspecialister som arbetar på den interna nivån. Den *konceptuella* nivån beskriver den logiska strukturen hos databasen. Detta schema döljer de lagringstekniska detaljerna för användarna, och beskriver enbart vilka data som lagrats samt vilka *operationer* som kan utföras på dessa data. Flertalet användare arbetar på den konceptuella nivån, och det är denna nivå hos relationsdatabaser vi beskriver i detta avsnitt. I många fall är dock även den konceptuella nivån för komplicerad för slutanvändaren, vilket gör att man inför en tredje, extern nivå. På denna nivå beskrivs enbart de aspekter av databasen som behövs för en viss tillämpning eller en viss grupp användare.

På den konceptuella nivån består en relationsdatabas av tabeller (på engelska: *relation tables*). Varje tabell måste ges ett för databasen unikt namn, och består av namngivna *attribut* (kolumner i tabellerna) med angivna datatyper. Vilka datatyper en relationsdatabas kan hantera kan variera mellan olika system, men alla kan hantera de grundläggande typerna:

- text (*string*),

- heltal (*integer*),

- decimaltal (*float*).

Varje rad i tabellen motsvaras av ett objekt (benämns ofta post).

Varje tabell måste innehålla en primärnyckel (primary key). En primärnyckel består av ett eller flera attribut som måste ha ett unikt värde för varje objekt. Vanligen är de attributnamn som ingår i primärnycklarna understrukna i beskrivningen av tabellen.

Exempel: En enkel databas över ett lands vägar och trafikolyckor kan tänkas innehålla följande tabeller:

Vägsegment(segment_id: integer, väg_nr: string, vägtyp: string)

Olyckshändelser(segment_id: integer, väg_nr: string, olycka_nr: integer, olyckstyp: string)

Det visar sig att det finns många sätt att definiera tabeller och nycklar i en relationsdatabas, och att bara vissa logiska strukturer garanterar att enkla och naturliga regler gäller för hur databasen ska tolkas i samband med frågor och uppdateringar. För att undvika dessa problem strävar man normalt efter att bygga upp databasen enbart av relationer som följer vissa *semantiska* regler, s.k. *normalformer*. Den *första normalformen* föreskriver att varje cell i tabellen endast får innehålla ett skalärt (icke sammansatt, atomärt) värde. Detta krav har tidigare gällt nästan alla relationsdatabaser men är inte längre obligatoriskt. Detta gör teori och konstruktion av relationsdatabassystem mer komplex men samtidigt möjliggörs en rad praktiska förenklingar och kraftfulla generaliseringar. Ett grundläggande syfte med "högre" normalformer (än den första) är

att se till att redundant information inte förekommer i databasschemat, något som måste avgöras genom att analysera tillämpningens informationsbehov och inte nödvändigtvis framgår av en viss given databas.

3.1 Frågespråket SQL

En av anledningarna till att relationsdatabaser är så flitigt använda är att det finns ett standardiserat språk för att kommunicera med databasen. Detta språk benämns SQL (Structured Query Language) och har kommit ut i en rad versioner. Den senaste versionen är SQL 2008. Den version som lanserades 1999 var den första som inkluderade objektorienterade (objektrelationella) begrepp.

SQL kan användas bland annat för att skapa en databastabell, lägga till nya attribut, fylla tabellen med nya objekt och fråga efter data i en eller flera tabeller. Nedan exemplifierar vi några av dessa kommandon utifrån exemplet med vägdatabasen.

För att skapa tabellen vagsegment används SQL-kommandot CREATE:

CREATE TABLE vagsegment (*segment_id: INTEGER,*

vag_nr: CHARACTER VARYING (15),

vagtyp: CHARACTER VARYING(15),

PRIMARY KEY (*segment_id, vag_nr*))

En sökfråga i SQL kan bestå av upp till 6 "klausuler" eller satsdelar. De två första, SELECT ... och FROM ..., är obligatoriska, de övriga valfria (markerat med [ ]):

SELECT <*attribut- och funktionslista*>

FROM <*tabellista*>

[WHERE <*villkorsuttryck*>]

[GROUP BY <*attributlista*>]

[HAVING <*grupperingsvillkor*>]

[ORDER BY <*attributlista*>]

SELECT-klausulen anger de attribut eller funktioner som ska bearbetas. FROM-klausulen anger vilka tabeller som ingår i frågan. WHERE-klausulen anger kriterierna för urval av objekt från dessa tabeller, GROUP BY anger vilka grupperingsattribut som ska användas, medan HAVING anger ett eventuellt villkor som ska gälla för de resulterande grupperna. De "inbyggda" aggregeringsfunktionerna COUNT, SUM, MIN, MAX och AVG (average) används ofta tillsammans med gruppering, men om GROUP BY - klausulen utelämnas appliceras de i stället på samtliga selekterade objekt (tabellrader). ORDER BY anger sorteringsordning för resultatet, och är främst avsedd för att skapa läsbara tabellutskrifter.

För att få veta exempelvis vilka vägsegment av E18 som drabbats av singelolyckor kan man ställa följande databasfråga:

SELECT DISTINCT *segment_id*

FROM olyckshandelser

WHERE *vag_nr* = ’E18’ AND *olyckstyp* = 'singel';

Notera nyckelordet DISTINCT här, som medför att varje vägsegment tas med bara en gång. SQL levererar annars i detta fall ett värde på *segment_id* för varje olycka som inträffat där.

Är man intresserad av vilka vägtyper de vägsegment av E18 har, som har drabbats av singelolyckor, kan man ställa följande SQL-fråga:

SELECT DISTINCT *vagtyp*

FROM vagsegment

WHERE olyckshandelser.*olyckstyp* = 'singel' AND olyckshandelser.*vag_nr* = ’E18’ AND olyckshandelser.*segment_id* = vagsegment.*segment_id* AND olyckshandelser.*vag_nr* = vagsegment.*vag_nr*;

Detta är en koppling mellan tabeller (s k *join*) där de objekt som bidrar till resultatet är de som uppfyller villkoret i WHERE-satsen. I detta fall är villkoret att attributvärdena för *segment_id* samt *vag_nr* måste vara identiska för objekten i de bägge tabellerna.

I SQL kan vad som logiskt sett är en och samma fråga ofta ställas på många olika sätt. Tillsammans med SQLs allmänna komplexitet är detta naturligtvis en källa till svårigheter för nybörjaren, som kan ha svårt att förstå hur en invecklad frågeformulering är tänkt att fungera. Till detta kommer att olika sätt att ställa samma fråga kan ta väsentligt olika lång tid att utföra i ett visst databassystem, medan ett system med en mer avancerad frågeoptimering kanske använder exakt samma snabba sätt att utvärdera de två logiskt ekvivalenta frågorna.

4 Objektorienterade databaser

En trend som började framträda under 80-talet är olika försök att vidga tillämpningsområdet för databastekniken genom att lämna den ortodoxi kring framför allt relationsmodellen som tidigt uppstod. Många databasforskare valde att i stället utveckla olika varianter av *objektorienterade databassystem* men kommersiellt har dessa idéer och lösningar, i alla fall hittills, inte varit särskilt framgångsrika.

Under 90-talet och fortfarande har också olika kompromisser eller hybridlösningar föreslagits, delvis under rubriken *objektrelationella* (eller *utvidgade relationsdatabaser*). Språkstandarden SQL ger numera (sedan versionen SQL: 1999) goda möjligheter att kombinera koncept från relationsdatabasmodellen och objektorienterade databasmodeller.

Objektorienterade databashanterare (ODBMS) ger tillgång till begrepp liknande dem som introducerats ovan för relationsdatabaser när det gäller att beskriva de objekt som lagras i databasen. Beträffande möjligheten att definiera operationer på data finns olika skolor. En del leverantörer av ODBMS uppfattar sin produkt som en utvidgning av ett programspråk, typiskt C, med en säker och abstrakt metodik för persistent lagring av komplexa datastrukturer. Andra ser sin produkt som ett generellt databassystem med en teknologi som ska kunna användas för alla de uppgifter som relationsdatabaserna hanterat tidigare, och som därutöver ska vara utvidgningsbart till en rad nya tillämpningsområden. Inte minst i konstruktionstillämpningar (antingen de är mekaniska, elektroniska, arkitektoniska, forminriktade eller programvaruinriktade) har ODBMS fått stor spridning, och en av deras främsta fördelar är att de ofta erbjuder kraftfull och effektiv versionshantering, vilket är särskilt viktigt när flera konstruktörer arbetar med samma konstruktion.

Den frågespråksstandard som ursprungligen utvecklades för objektorienterade databaser kallades OQL [Cat94]. Denna standard har dock inte alls fått samma spridning som SQL, och därför har många leverantörer av objektorienterade databaser numera gått över till SQL som frågespråk för att kunna konkurrera med de idag dominerande objektrelationella databaserna.

Nedan beskriver vi några av de viktigaste begreppen inom objektorienteringen som *objektklasser*, *metoder*, *inkapsling* och *arv*. Dessa begrepp utvecklades ursprungligen inom programspråksvärlden och därför ger vi även en kort beskrivning av hur de objektorienterade programspråken har utvecklats.

4.1 Objektklasser och metoder

De fördelar som objektorientering för med sig är framför allt mycket bättre metodik för modellering av komplexa system och processer, antingen de är realistiska och konkreta eller helt abstrakta. Detta görs genom att användaren kan definiera sina egna objektklasser samt attribut och metoder kopplade till dessa. Attribut är en egenskap hos ett objekt (t.ex. vägnummer för en väg), medan metoder är regler för hur objektklasser ska kommunicera med varandra.

4.2 Inkapsling

Ett centralt begrepp inom objektorienteringen är *inkapsling*. Inkapslingen innebär att varken den logiska strukturen hos en objektklass eller dess implementering är synlig eller åtkomlig utifrån. Endast genom metoder (objektklassens *gränssnitt*) kan man växelverka med den, påverka den eller få information från den. T ex kan man endast ändra eller fråga efter vägnumret i klassen väg genom metoderna *andraVagnummer* eller *hamtaVagnummer*. En viktig konsekvens av systematiskt genomförd objektorientering är att det blir möjligt att skapa välstrukturerade, generella programbibliotek (klassbibliotek) för olika typer av generellt förekommande uppgifter och att få sådana bibliotek, även från olika leverantörer, att fungera tillsammans i nya tillämpningar.

4.3 Arv

En klass kan skapas utgående från en existerande klass genom arv. Utgående från objektklassen *Vag* skulle vi t.ex. kunna skapa objektklasserna *Motorvag* och *Traktorvag*. De senare benämns *subklasser* medan *Vag* är en *superklass*. Subklasserna ärver attribut och metoder från superklassen men kan ha sina egna implementationer av metoderna.

4.4 Objektorienterade programspråk

Redan i slutet på 60-talet introducerades det norska objektorienterade språket Simula. Simulas begreppsapparat för strukturering av data och bearbetningsprocesser representerade ett revolutionerande framsteg i jämförelse med den tidens konkurrerande språk, men då liksom nu var det i första hand amerikanska företag och universitet som avgjorde vilka trender inom informationstekniken som skulle bli bestående, och trots betydande satsningar, inte minst av FOA, fick Simula inte särskilt många anhängare utanför Norden och Europa. I början av 80-talet lanserades emellertid ett nytt interaktivt språk och system av Xerox i USA, Smalltalk, som hade anammat och vidareutvecklat idéerna från Simula, och också populariserade det nya begreppet *objektorientering* som en sammanfattande benämning på dessa idéer. På 70- och 80-talen vidareutvecklades också "AI-språket" Lisp till objektorienterade versioner, senare standardiserade under namnet CommonLisp. Smalltalk och CommonLisp fick stor spridning, särskilt vid amerikanska universitet, och objektorienteringskonceptet hade kommit för att stanna. Det blev dock en objektorienterad vidareutveckling av det generella men ”maskinnära” programspråket C++ som innebar det definitiva genombrottet för objektorienterad programmering i stor skala. Senare har också Java, även det inspirerat av det ursprungliga C, fått en stark ställning som objektorienterat programspråk, och det är i flera avseenden modernare än C++. Numera har objektorienteringsprincipen slagit igenom helt inom programspråkstekniken.

4.5 Objektorienterade spatiella databaser

Mycket forskning och utveckling har lagts ner på att utveckla objektorienterade databaser som är specialkonstruerade för rumslig information. Dessa brukar man inom IT-området benämna *spatiella databaser* eller *geografiska informationssystem* (GIS). Sådana databaser blir allt vanligare inom geografisk informationsbehandling.

Det finns idag endast ett fåtal kommersiella GIS som är byggda på objektorienterade databaser. En av de mer välkända är Smallworld. Karakteristiskt för detta program är att det har effektiv versionshantering, förmåga att bearbeta stora mängder spatiella data i såväl raster- som vektorform, stöd för "sömlös" hantering av mycket stora polygoner och ett spatiellt utvecklings- och frågespråk. Smallworld har sin största kundbas inom tillämpningar för utbyggnad, administration och underhåll av storskaliga kraftnät, telenät, vägnät och andra typer av infrastrukturella nätverk.

4.6 Objektorienterade databassystem historiskt och på marknaden

ODBMS var av sina upphovsmän och tidiga förespråkare [A+89] ursprungligen avsedda att ersätta RDBMS. Deras främsta fördel är att de är mycket bättre anpassade till de objektorienterade programmeringsspråk som idag är dominerande. Men höga uppgraderingskostnader, framgångsrik inkorporering av viktiga objektorienteringsbegrepp i existerande RDBMS samt lanseringen av O/R-M-system (Object-Relational Mappers, se bilaga 2) för att underlätta dataöverföring mellan databas och applikationer har gjort att RDBMS framgångsrikt kunnat försvara sin dominans inom området persistent lagring på serversidan i datacentraler. ODBMS är idag ett komplement, inte en ersättare, till relationsdatabaser. De har hittat en nisch som inbyggnadsbara persistenslösningar i apparater, i klientdatorer, i paketerad programvara, i realtidsstyrsystem och på webbplatser.

Enligt den introduktion till ODBMS av Rick Grehan och Doug Barry som man kan hitta på Nätet [Bar05] växte marknaden för ODBMS-produkter under 90-talet till ca 100 MUSD, stagnerade under år 2000 och minskade sedan. Ett initiativ av industrigruppen Object Data Management Group (ODMG) för att skapa ett standardiserat objektfrågespråk (OQL) övergavs 2001. Ett annat försök att standardisera dataåtkomsten i ODBMS var Java Data Objects (JDO), som dock inte heller har rönt någon större acceptans. År 2004 släpptes Open Source-produkten *db4o* som en fri programprodukt med öppen källkod, som har skapat ett nyvaknat intresse för ODBMS [ODB08]. Sett i stort måste man säga att ODBMS är ett kommersiellt misslyckande, i alla fall hittills, men det finns ett antal framgångsrika produkter. *Objectivity* anses vara en av dessa.

En ganska livaktig debatt om ODBMS pågår fortfarande, men den verkar inte vilja ta upp frågan om databaser ur ett organisationsperspektiv utan handlar huvudsakligen om programmerarens eller det enskilda projektets perspektiv på DBMS. Men om man ser på utveckling av applikationer och tjänster som isolerade företeelser kommer man aldrig åt huvudargumentet i databasfilosofin, som

handlar om utveckling, utnyttjande och förvaltning av organisationens verksamhetskritiska och ofta strategiska informationsresurser, ”*data ownership*”.

Det mest framgångsrika tillämpningsområdet för ODBMS är av allt att döma konstruktionstillämpningar. Arkitektur, produktdesign, mekanisk konstruktion, elektronikkonstruktion, LSI-konstruktion m fl är områden som idag nästan uteslutande använder datorgrafik i sitt designarbete. I organisationer där sådan verksamhet bedrivs rutinmässigt i stor skala, t ex flyg- och bilindustrier, programvaruindustri, elektronikindustri, speciellt vid konstruktion av komplexa integrerade kretsar, stora byggföretag m fl, kommer behovet av systematisk och säker lagring, uppdatering och återläsning av konstruktioner snabbt att motivera att databasteknik används. Ett specialfall är utarbetande och hantering av anläggningsbeskrivningar, t ex för kraftnät, där geografiska data och tekniska konstruktionsdata används tillsammans.

Bara ett fåtal egenskaper, som dataskydd och transaktionskapacitet för stora mängder användare, hos konventionella DBMS är av väsentligt mindre betydelse i konstruktionstillämpningar. De tillämpningar som har särskilt goda utsikter att dra fördel av ODBMS-teknik karakteriseras av tre egenskaper:

1. Deras behov uppfylls inte väl av de egenskaper och prestanda som erbjuds av RDBMS, och dessa egenskaper kan inte heller enkelt anpassas till behoven med hjälp av nya funktioner i ett extra lager ovanpå databassystemet.
2. Deras databas är tillräckligt stor eller komplex för att dra nytta av ett databassystems funktioner, medan ett behovsanpassat filhanteringssystem har svårt att räcka till.
3. Deras prestandakrav är inte så stränga att tillämpningarna bara kan fungera väl i ett specialkonstruerat system. Att skapa och underhålla ett stort och komplext sådant system skulle dock sannolikt bli mycket dyrt. Det är möjligt att också en DBMS-baserad lösning kan komma att kräva en alltför stor utvecklingskostnad för de flesta organisationer om nödvändiga verktyg inte redan finns tillgängliga. Ofta är det svårt att avgöra var linjen ska dras: moderna organisationer kräver ofta stor flexibilitet, och hemmabyggda system, eller över huvud taget enstaka systembyggen, kan inte erbjuda detta. De kräver istället att betydande fasta resurser avsätts för underhåll och kontinuerlig vidareutveckling. För stora affärskritiska system kan en arkitektur som bygger på en verktygslådeprincip ibland vara att föredra, där en stor del av systemet består av kommersiellt tillgängliga komponenter, eventuellt på en ”lägre nivå” än kompletta databassystem.

Bland de situationer då ODBMS är att föredra finns databaser med mycket komplexa länkade datasamband (NATOs datautbytesstandard JC3IEDM kan tjäna som exempel, om den skulle användas som databasschema). Ett ODBMS kan i regel hantera hela kluster av sådana länkade poster som en accessenhet och alltså ge tillgång till hela klustret med ett enda anrop. Detta kan underlätta programmeringsarbetet avsevärt om databasen behöver hantera komplexa logiska datastrukturer, t ex komplexa sociala nätverk.

4.7 Sammanfattande krav på objektdatahantering (se också bilaga 1)

- **Unika objektidentifierare** – viktigt i alla teknik- och konstruktionstillämpningar; objekt har inga identifierare med namn som är meningsfulla för människor, eller deras identifierare kan förändras (detaljnr, modulnamn).

- **Sammansatta objekt** – i de flesta konstruktionstillämpningar behövs förmåga att definiera objekt som innehåller andra objekt.
- **Referenser och integritet** – viktigt att kunna referera ett objekt från ett annat (relationer) och låta databassystemet bevaka integriteten hos dessa referenser.
- **Objekttypshierarki** – arvshierarkier behövs när subtyper ska definieras.
- **Associerade procedurer** – förmåga att lagra procedurer (och inte enbart data) i en databas är viktig i många tillämpningar.
- **Objektinkapsling** – förmåga att använda procedurer associerade med objekttyper för att kapsla in datastrukturer är mycket viktig i ett ODBMS.
- **Ordnade mängder och referenser** – behövs i många tillämpningar men kan inte hanteras i alla RDBMS.
- **Stora datablock** – "binära" stora datablock (BLOBs) som bilder eller text lagrade som attribut till ett objekt är användbara i nästan alla tillämpningar men kan inte hanteras i alla RDBMS.
- **Effektiv åtkomst på distans** – enkel åtkomst till databasen via ett nätverk är idag ett självklart krav.
- **Lätthet att göra schemaförändringar** – viktigt att kunna modifiera ett schema med minsta möjliga sidoeffekter på existerande program. Konventionella DBMSs saknar bra metoder för att migrera data till nya scheman. Men det är inte uppenbart att det är mycket enklare i ODBMS.
- **Gränssnitt mot programmeringsspråk** – att kunna integrera ett datahanteringsspråk i ett DBMS med användarnas programmeringsspråk är ett behov i många tillämpningar. RDBMS kan inte lösa detta, vilket leder till s k "impedansmissanpassning" [EN06, kap. 9.1]. Å andra sidan, om ett RDBMS eller ett relationsfrågespråk är givet, uppstår problemet när man behöver skriva program som flyttar data mellan ett RDBMS och ett ODBMS. Se också avsnittet om O-R/M-missanpassning i bilaga 2.
- **Multipla databasversioner** – förmåga att underhålla mer än en version av individuella objekt eller en hel databas är viktig i konstruktionstillämpningar. Tidigare utgåvor måste underhållas, parallella delprojekt måste koordineras och deras resultat kombineras.
- **Låsning och check-out över långa tidsperioder** – förmåga att låsa en stor delmängd av data, som ett sammansatt objekt eller en hel konstruktion, är ofta viktigt. Existerande data kan t ex göras enbart läsbara, vilket möjliggör att nya versioner skapas så snart uppdateringar görs.
- **Enanvändarprestanda** – acceptabla prestanda också när en användare har låst ut alla andra skrivare från en stor del av databasen är mycket viktig i konstruktions- och tekniktillämpningar, där finkornig samtidig åtkomst förekommer mycket sällan. I motsats till detta kräver OLTP-tillämpningar höga fleranvändarprestanda och finkorniga lås.

5 Objektrelationella databaser

En tendens inom databastekniken är att man successivt och evolutionärt övergår från den ca trettio år gamla ursprungliga relationsdatamodellen till modernare s k objektrelationella databassystem [SMB99], som kan lagra, söka och presentera data med i princip godtycklig struktur, alltså även t ex multimediadata och digitala kartdata. Till skillnad mot tidiga objektdatabaser tillhandahåller, som tidigare nämnts, objektrelationella databaser i regel en vidareutvecklad version av frågespråket SQL. Vidare kan man i objekt-relationella databaser söka i och lagra inte bara relationstabeller (som i de klassiska relationsdatabaserna) utan också mer komplexa användardefinierade datastrukturer.

Objektrelationella databasservrar, t ex *Oracle 11g*, *IBM DB2 9.5* och *Microsoft SQL Server*, är vanligen också utbyggbara (*extensible*) med nya datatyper och bearbetningsfunktioner. Sådana utbyggnader kräver specialistkunnande, men behöver inte alltid vara förbehållna leverantören, utan kan också, i alla fall för vissa produkter, vara tillgängliga för större användarorganisationer och tillverkare av påbyggnadsprogram. För *Microsoft SQL Server* har Microsoft utvecklat ett gränssnitt, *OLE DB*, som tillåter att man ansluter indexerade externa datakällor som virtuella relationsdatabastabeller. OLE DB-gränssnitt finns också för andra ledande databashanterare.

För att olika programsystem och databassystem ska kunna integreras med rimliga ansträngningar krävs att de följer vedertagna standarder för gränssnitten mellan systemen. Detta avsnitt avslutas med en genomgång av arbete som genomförts av *Open Geospatial Consortium* (OGC, se nedan) för att standardisera användningen av databaser inom geografisk informationsbehandling.

5.1 Objektrelationella spatiella databaser

Mycket forskning och utveckling har lagts ner på att utveckla objektrelationella databassystem som har förmåga att hantera även rumslig information. Dessa brukar man inom IT-området benämna *spatiella databaser*. Sådana databassystem har förmåga att representera, söka i, bearbeta och presentera information som har grundläggande rumsliga aspekter. Geografiska databaser är förstås en viktig undergrupp till de spatiella databaserna, men det finns åtskilliga andra exempel på spatiella databaser, t ex CAD-databaser eller medicinsk/anatomiska databaser.

Av en spatiell databashanterare förväntar man sig att förmåga att hantera rumslig information ska finnas samtidigt med konventionella databasförmågor, som snabb sökning, möjlighet att lägga till eller förändra objekt i databasen, förmåga att hantera flera samtidiga användare, etc. Det vore alltså en styrka hos en spatiell databashanterare om den kunde uppfattas som en utvidgning av en vanlig databashanterare. I så fall kunde man använda samma kärna av tjänster och förmågor för alla tillämpningar, spatiella eller ej, eventuellt kompletterade med av användaren beställda paket av tjänster för speciella behov, t ex kartering eller spatiella analyser. Just så är de objektrelationella databassystemen tänkta att fungera. De ska samtidigt erbjuda en kärna av samma tjänster som relationsdatabashanterare i allmänhet, kompletterade med olika paket av tjänster och

begrepp från olika generella tillämpningsområden, som bildhantering, geografiska operationer, datorstödd konstruktion (CAD), etc. Naturliga generaliseringar av operationer på de nya datatyperna erkänns av systemet och kan lätt kännas igen av användaren.

En spatiell databas innehåller förutom de vanliga datatyperna (text, heltal, etc.) speciella datatyper för att hantera geometri. Dessa datatyper kan vara t ex punkt, linje och polygon; i vissa system finns även tillgång till tredimensionella objekttyper. Vidare går det att göra operationer på dessa datatyper. Det är t ex möjligt att söka de punkter som ligger inom en given polygon eller de punkter som ligger inom ett visst avstånd från en given linje. Flera av de spatiella databaser som är inriktade på GIS-tillämpningar använder standarder från *Open Geospatial Consortium* (OGC) som beskrivs nedan. Dessa standarder beskriver såväl vilka geometriska datatyper som skall användas som vilka operationer som ska gå att göra på dessa datatyper.

5.2 Interoperabilitet för geoinformation och Open Geospatial Consortium (OGC)

Open Geospatial Consortium (OGC) (http://www.opengis.org) är en fristående, icke vinstdrivande organisation som skapats med syfte att specificera geografisk informationsteknik som kan möjliggöra interoperabilitet över Internet mellan olika organisationers system och databaser för geodata. OGC har också etablerat nära samarbete med ISO TC 211, den internationella standardiseringsorganisationens utskott för geoinformation.

Att åstadkomma interoperabilitet har varit ett svårt problem för användare av geografisk informationsteknik. Under de år som GIS-tekniken vunnit spridning har detta problem alltmer kommit i förgrunden. Olika ansträngningar har gjorts för att minska svårigheterna, inte minst genom standardisering av geodataformat. OGC avser att möjliggöra interoperabilitet mellan databaser för geodata av alla slag: geografiska kartdata, rasterdata av olika slag inklusive t ex satellitbilder och fjärranalysdata, punkt- och vektordata, tredimensionell information som 3D-terrängdata, utbredningsfält för t ex väderdata eller föroreningsspridning, och spatio-temporala ("fyrdimensionella") data av olika karaktär och ursprung.

Man har angripit detta problem genom att utforma en serie leverantörsoberoende specifikationer för interoperabel programvara. De består av en överordnad, "abstrakt" specifikation samt ett antal implementerings-specifikationer för olika, konkurrerande distribuerade datorplattformar (*Distributed Computing Platforms*, DCP), bland dem CORBA, OLE/COM, DCE och Java.

Denna standard beskrivs bl a i dokumenten *Simple Feature Access Part 1: Common Architecture, version 1.2.0* [Her06a], respektive *Simple Feature Access Part 2: SQL Option, version 1.2.0* [Her06b]. Det första dokumentet specificerar objekttyper och operationer, medan det andra visar hur dessa definitioner skall tolkas i spatiella databassystem med frågespråk baserade på objektrelationell SQL.

5.3 OGC:s definition av spatiellt utvidgad SQL

OGC har definierat en *spatiell utvidgning av SQL* benämnd SFSQL. SFSQL tillför både spatiella objekttyper och operationer till SQL. Med hjälp av SFSQL kan man formulera en mängd olika slags frågor med geografisk anknytning. Vi ger här ett par exempel på frågor i SFSQL-syntax. Vi utgår ifrån två tabeller *Städer* och *Länder* med följande attribut:

Stader (id: integer, stadsnamn: string, geometri: Point)

Lander(id: integer, landsnamn: string, geometri: Polygon)

Observera att de bägge attributen *geometri* har geometriska datatyper.

Om man vill se vilka svenska städer som finns lagrade i databasen kan man ställa följande (topologiska) fråga:

```
SELECT Stader.stadsnamn
FROM Stader, Lander
WHERE contains(Lander.geometri, Stader.geometri) AND Lander.landsnamn
= 'Sverige'
```

I WHERE-satsen undersöks villkoret att geometrin för länder innehåller (*contains*) geometrin för städerna. Vill man veta avståndet mellan Stockholm och Oslo ställer man den (geometriska) frågan:

```
SELECT a. stadsnamn, b.stadsnamn, distance(a.geometri, b.geometri)
FROM Stader AS a, Stader AS b
WHERE a.stadsnamn = 'Stockholm' AND b.stadsnamn = 'Oslo'
```

I detta fall har den geometriska operationen *distance* använts för att beräkna avståndet. Observera användningen av AS i FROM-satsen. Genom att använda denna sats här får vi två instanser av tabellen *Städer* (döpta till *a* respektive *b*) vilket underlättar formuleringen av SFSQL frågan. Svaret på frågan blir, som väntat, en tabell med tre kolumner och en rad. På raden står helt enkelt *Stockholm, Oslo*, samt avståndet mellan städerna. Hur man definierar och beräknar sådana avstånd är dock en långtifrån trivial fråga som faller utanför ramen för denna rapport.

5.4 Spatio-temporala databaser

Spatio-temporala databaser [GBE+00] hanterar geometrier som förändras över tiden. För att hantera sådana tidsberoende geometrier behövs en datamodell och ett frågespråk som kan representera och operera på abstraktioner av rörliga objekt, speciellt rörliga punkter och rörliga regioner. Förutom dessa grundläggande objekttyper visar det sig att ett ganska stort antal ytterligare objekttyper behövs. Man behöver t ex datatypen linje för att representera projektionen av en rörlig punkt på ett plan.

Spatio-temporala databaser är fortfarande på forskningsstadiet, men det är fullt möjligt att databasprodukter med spatio-temporal förmåga kommer att lanseras inom några år.

5.5 Bild- och multimediadatabaser

Ett teknikområde som blir allt viktigare när datorer nu fått sådan kapacitet att de effektivt kan hantera bild- och videoinformation är databasteknik för bilder och multimedia. Sådan teknik kommer att få stor betydelse inte minst för framtida underrättelsesystem, eftersom allt fler sensortyper får förmåga till bild- och videoregistrering. Kravet att kunna lagra och söka bland foton tagna med vanliga digitalkameror eller mobiltelefonkameror medför också behov av sådan registrering. I svenska utlandsmissioner lagras idag sådana bilder i vanliga filsystem, vilket gör det svårt att hitta rätt bild när det behövs. Tekniken på detta område är än så länge ganska lite utvecklad, särskilt om man vill kunna utföra effektiva sökningar på stora mängder lagrade data.

6 Viktiga databasarkitekturer

6.1 Klient-servermodellen

I många fall är det praktiskt att data är lagrade på en annan dator än den man arbetar på. Det är också möjligt att datorerna delar på funktionalitet, dvs att en dator gör beräkningar åt en annan dator (eller *nod*, som man ofta säger när man beskriver distribuerade system och nätverk av datorer). Inom datalogin benämns den idag vanligaste modellen för samverkande datorer *klient-server*-modell. *Klienten* är i detta fall en nod som skickar förfrågningar om data eller beräkningar till en annan nod, som benämns *servern*.

Inom en organisation arbetar ofta flera personer med att skapa, uppdatera och analysera data. För att arbetet skall vara effektivt måste alla medarbetare inom organisationen arbeta med aktuella data. Detta innebär att så fort en medarbetare gjort en uppdatering måste dessa uppdateringar spridas till övriga inom organisationen. Vi kan lätt inse att detta kommer att leda till en orimlig situation om uppdateringar måste skickas mellan alla anställda. En naturlig lösning på prob-lemet är att ha en centraliserad lagringsstruktur där alla arbetar mot samma datamängd.

6.2 Distribuerade databassystem och replikering

I *distribuerade databassystem* [ÖV91] är databasen fördelad över flera noder, och databasservern tillhandahåller data för klienterna utan att dessa behöver ha kunskap om detaljerna i datadistributionen. Behov av sådan *distribution* uppstår både av prestanda- och tillförlitlighetsskäl.

Redundans och konsistens i distribuerade databaser är viktiga krav som kommer att kunna uppfyllas i ett ledningssystem förutsatt att frågan ägnas tillräcklig uppmärksamhet vid systemkonstruktion och systemutveckling. Två grundläggande metoder som måste beaktas vid konstruktion av distribuerade databaser är *fragmentering* och *replikering*. Fragmentering innebär att en logisk databasstruktur (t ex en relationstabell) är fysiskt distribuerad över flera datanoder, främst i syfte att fördela frågebelastningen på flera noder. Replikering innebär att samma data återfinns i flera datanoder, alltså en form av redundans. Ett konstruktionsproblem för databashanteraren blir då att bevara konsistensen hos fragmenterade och replikerade data då data uppdateras utan att klienterna behöver vara medvetna om distributionen. Andra konstruktionsproblem innefattar att minimera frågebelastningen genom att hämta data parallellt från flera datanoder, och att utnyttja redundanta noder för att maskera kommunikationsavbrott och nodsammanbrott.

Ett problem vid replikering är att kostnaden för att upprätthålla konsistens mellan replikerade data kan vara hög när uppdateringsfrekvensen är hög. Ett antal replikeringsalgoritmer har utvecklats för detta. En enkel algoritm är att alltid utföra databasuppdateringarna i en primär datakopia (eng. *primary copy*) och sedan propagera uppdateringarna till den övriga datanoderna (eng. *backup copies*) [AG76]. För full datakonsistens måste man använda det s k *two-phase commit*-protokollet [SKS96] och vänta tills alla kopior blivit konsistenta, vilket

kraftigt försämrar skalbarheten när data är replikerade på många noder. Replikering fungerar dock tillfredställande om replikerade data inte uppdateras ofta eller om antalet replikat är litet.

I många moderna databashanteringssystem har man möjlighet att uppdatera en huvudkopia av replikerade data, varefter systemet asynkront vidarebefordrar uppdateringarna till ”slav”-replikaten utan att den uppdaterande databasoperationen väntar på att replikaten skall bli konsistenta [PL91].

6.3 Federerade databaser och medlarteknik

Federerade databaser eller *multidatabaser* är en relativt ny teknologi. Med detta menas system som kan fungera som automatiska översättare mellan en mängd geografiskt distribuerade databassystem av olika typ och med delvis olika data. På detta sätt är tanken att man skall kunna uppnå full distribuerad funktionalitet och homogenitet ur användarsynpunkt hos ett system som i själva verket består en rad olika, sinsemellan egentligen tekniskt inkompatibla databaser.

Skillnaden mellan distribuerade och federerade databaser kan beskrivas på följande sätt. En distribuerad databas är logiskt sett en central databas, som är implementerad över flera distribuerade datanoder. En federerad databas är å andra sidan en samverkande kollektion av oberoende databaser.

Ledningssystem behöver kunna kombinera data från olika källor, både olika typer av databaser, simuleringssystem, och andra system. *Medlartekniken* är en viktig ansats för att skapa sådana federerade system som kombinerar data från främmande databaser och andra system. I den öppna Internetvärlden behövs federerade databaser och medlare för att kombinera sådana *heterogena* data medan distribuerade databaser behövs för att tillhandahålla snabba och tillförlitliga databasservrar. Ett medlarsystem tillhandahåller *dataintegreringsprimitiver* som används i medlarmodellen för att *kombinera* data från externa källor och presenterar de integrerade data för andra medlare och applikationer som vyer på ett homogent sätt.

För varje typ av datakälla måste en *inkapslare* (eng. *wrapper*) utvecklas, vilket är programvara som upprättar ett gränssnitt mot data från en viss typ av datakälla. Inkapslaren överför dessa data till ett standardiserat format som används av medlare och tillämpningsprogram. Inkapslarna gömmer detaljer och fysisk datarepresentation hos datakällorna och höjer abstraktionsnivån, samt tillåter förändringar i datakällorna utan att gränssnitten mot medlare och tillämpningsprogram behöver ändras.

6.4 Personliga databaser

En utvecklingstendens är att databastekniken snabbt vinner spridning till områden där den från början inte haft någon plats, exempelvis för personligt bruk och för att byggas in som komponenter i stora komplexa system. Denna utveckling förutsätter enklare och mer lättunderhållna system än som är typiskt för dagens stora databasprodukter. Exempelvis måste systemet automatiskt anpassa sig efter varierande behov och inte kräva den upprepade manuella in-

ställning (eng. *tuning*) som stora databashanterare kräver. Spridningen av persondatorer har medfört att sådana system har etablerats på marknaden, och de bästa av dem har idag i många avseenden en funktionalitet som är jämförbar med de stora fleranvändarsystemens. Exempel på databassystem avsedda för inbyggnad är *PostgreSQL* och svenska *MySQL*. Ett mycket spritt databassystem för personligt bruk är *Microsoft Access*, vilket dock inte anses kunna mäta sig med fullskaliga databashanterare beträffande prestanda och skalbarhet.

7 Enterprise Information Integration

En grupp teknologier under framväxt är EII, Enterprise Information Integration [BH08], på svenska *informationsintegration*.Till detta samlingsnamn brukar man hänföra en rad olika tekniska lösningar för att kombinera information från olika källor till en gemensam form (tyvärr används ibland termen informationsfusion synonymt med EEI, men vi vill understryka att termerna informationsintegration och informationsfusion officiellt betecknar två till sin karaktär mycket olika grupper av teknologier, som dock kan behöva användas parallellt exempelvis i en del underrättelsetillämpningar).

Philip Bernstein och Laura Haas, två framstående seniora databasforskare från Microsoft respektive IBM, beskriver i sin översiktsartikel [BH08] vilka problem- och metodtyper som brukar räknas till EII. Deras lista över olika typer av EII-verktyg ser ut som följer:

- Data Warehouse Loading (Extract-Transform-Load, ETL)
- Virtual Data Integration (medlarteknik)
- Message Mapping
- Object-to-Relational Mappers
- Document Management Tools and Techniques
- Portal Management Tools and Techniques

De viktigaste grundläggande teknikerna som används för att lösa dessa uppgifter är, fortfarande enligt [BH08]:

- Extensible Markup Language (XML)
- Schema Standards
- Data Cleansing
- Schema Mapping
- Schema Matching
- Keyword Search
- Information Extraction
- Dynamic Web Technologies

Artikeln är översiktlig och ganska kortfattad men har en referenslista med 36 artiklar. I stället för att sammanfatta den, vilket skulle göra en redan översiktlig text ännu tunnare, rekommenderar vi den till läsning. För att nå verklig förståelse av ämnet bör man gå till originallitteraturen, t ex de artiklar som refereras av [BH08]. Bernstein och Haas understryker att EII är ett område som fortfarande är tekniskt omoget och vars metoder är ”sköra”, även om det också innehåller delar som är mycket väl utforskade. Problemet ”schemaevolution” har exempelvis fått mycket uppmärksamhet från forskare, men trots detta är få kommersiella verktyg idag tillgängliga för att hantera det. En annan orsak till svårigheter är de komplexa regler som behöver tolkas och följas då man vill kombinera data från flera oberoende källor. Dessa regler måste beskrivas korrekt och följas i detalj för att man ska kunna undvika de inkonsistenser och ofullständigheter som kan uppstå.

8 Datalager och datautvinning

En annan utvecklingstendens är att stora ansträngningar görs för att kommersiellt introducera nya typer av databastillämpningar, som *datalager* (eng. *data warehousing*) och *datautvinning* (eng. *data mining*).

Ett datalager är en samling teknologier som syftar till att erbjuda en kunskapsarbetare (chef, administratör, analytiker) möjligheter att fatta bättre och snabbare beslut. Datalager är alltså avsedda att försörja analytiker och beslutsfattare med information om en organisations operativa verksamhet. Ett datalager förväntas innehålla rätt information på rätt plats vid rätt tidpunkt till acceptabel kostnad så att det kan ge underlag för rätt beslut.

Datalager innebär att verksamhetskritiska data samlas i särskilda databaser, där de kan utsättas för mycket mer ingående analyser än i vanliga, operativa databaser. I ett datalager är maskin- och programvara anpassad för att stödja komplexa, analytiska bearbetningar, exempelvis för effektiv och flexibel produktion av försäljningsstatistik uppdelad på en mängd olika kategorier. Ett datalager är en stor central databas i vilken data från ett antal datakällor är materialiserade. Syftet med datalager är att understödja beslut genom att göra det möjligt för användarna att ställa avancerade frågor som analyserar data om försäljning, lagerinnehåll, trender, prognoser etc. Datakällorna är ofta andra databaser i kontinuerlig drift.

Det har visat sig att databassystem som ska användas för analys- och beslutsstödsändamål bör ha andra tekniska egenskaper än de som är avsedda för att hantera operativa databaser [MF04]. I båda fallen handlar det om hantering av information av vital betydelse för hela organisationen. Om analysfrågorna ställs direkt till dessa databaser försämras deras transaktionsprestanda. I stället materialiseras (dvs kopieras eller, om så krävs, utvärderas; se även avsnitt 8.1) de data från datakällorna som behövs för beslutsstödet i datalagret med jämna mellanrum. Man bör således vara medveten om att data i datalager inte alltid är helt aktuella, vilket kan ge felaktiga beslut, speciellt i en krissituation.

Man kan hävda att om analysdatabasen inte innehåller någon annan information än den som finns i den operativa databasen så är den redundant och inte lika viktig att skydda mot t ex maskinfel. Man bör då komma ihåg att analysdatabasen antagligen också innehåller information från andra källor, liksom tidigare skapade, historiska data, ännu ej färdigbehandlade analysresultat, och underrättelsedata som hämtats från operativa databaser vid vissa bestämda tidpunkter och kanske inte längre finns kvar i dessa.

Två böcker som behandlar detta område är [JLV+02] och [MZ08]. Den förra är huvudsakligen en översikt över området och dess olika produkter. Den senare är en lärobok som bl a visar hur man kan organisera datalager för, och ställa frågor mot, två konkreta kommersiella produkter (från Microsoft respektive Oracle). Den innehåller också konkreta förslag till hur man kan organisera datalager för olika ändamål, inklusive sådana som även innehåller spatiella eller temporala data.

Data mining [FV07, TSK05, HK06], eller *datautvinning* som ett försök till svensk översättning lyder, betecknar en klass analysmetoder med vars hjälp

man kan upptäcka samband i och dra slutsatser ur t ex driftdata för att få insikter som kan leda till bättre planering. En detaljhandelskedja kan t ex tillämpa datautvinningsmetoder på data som samlats in från dess kassadatorer för att finna mönster i kunders köpbeteende, mönster som kan utnyttjas vid planering av varuexponering, annonsering, butikslayout etc. Med sådana metoder förväntar man sig naturligtvis att kunna öka försäljning och lönsamhet. Datautvinning utförs ofta mot datalager.

8.1 Materialiserade vyer

En speciell form av replikering är s k *materialiserade vyer* där databashanteraren av prestandaskäl upprätthåller en fysisk version av (materialiserar) ofta efterfrågade databasvyer. En icke materialiserad vy kan ses som en virtuell relationstabell (definierad genom en fråga) som är sökbar som andra tabeller, men som inte har något eget fysiskt innehåll lagrat i databasen. Materialiserade vyer innebär att databashanteraren kontinuerligt upprätthåller det fysiska innehållet i tabeller uttryckta som vyer. Liksom replikerade data kan vyers innehåll uppdateras synkront eller asynkront. I det senare fallet kan den materialiserade vyns innehåll vara mer eller mindre aktuellt.

En viktig skillnad mellan datalager och medlarsystem är att datalager alltid materialiserar data medan medlarsystem tillhandahåller virtuella högnivådatabaser i form av ett antal mer eller mindre komplexa vyer. Medlarsystem kan således oftast tillhandahålla mer aktuella data än datalager, men ställer högre krav på effektiv frågehantering. Internt kan medlarsystem materialisera data för effektiv frågehantering och för att spåra förändringar i data, men till skillnad mot datalager är sådan materialisering vanligen osynlig för användaren och står helt under kontroll av medlarsystemet.

9 Relationsdatabaser i analytiska tillämpningar

9.1 Argument för relationsdatabaser

Enligt vår mening är det i första hand fyra argument som fortfarande talar för RDBMS (eller hellre ORDBMS) i analystillämpningar, inklusive underrättelseanalys av data från ”alla källor”:

1. Kravet i många organisationer på central överblick och kontroll av organisationens hela dataförråd, inklusive alla dess begrepp och dess schema. Detta innebär att man har en DBA (databasadministratör) som ansvarar för datas kvalitet och integritet, och som är den instans som ger slutligt tillstånd att utnyttja dataresursen i en ny tillämpning och då måste kontrollera att den nya användningen följer alla gällande bestämmelser. Att införa en sådan roll torde vara ett stort steg att ta i en organisations tillvaro, och vi känner inte till hur vanligt det är att stora företag håller sig med en sådan. Risken för att en begränsande byråkrati uppstår är uppenbar, men det är också tydligt att verksamheten i många moderna organisationer är helt beroende av tillgång till en korrekt och konsistent databas. En del organisationer måste dessutom ständigt uppfylla legala krav på korrekthet och sekretess hos data. I och för sig är det möjligt att välja ett ODBMS med möjlighet att tillhandahålla ett centralt databasschema, en nödvändig resurs för en DBA. Tillgång till ett kraftfullt objekt-relationellt frågespråk är nästan lika viktigt som schemat för att göra det möjligt att kontrollera databasens innehåll när som helst och var som helst. Om det ODBMS man väljer erbjuder båda dessa resurser och dessutom allmänt hög driftssäkerhet, så faller de viktigaste argumenten mot ett sådant system.
2. Även i samband med applikationsutveckling är tillgång till ett kraftfullt frågespråk ofta en stor fördel. ODBMS som helt saknar sådant frågespråk bör man undvika att använda i ”öppna” tillämpningar där man inte redan från början kan se en skarp avgränsning av systemets tillämpningsområde. I den mån man har i uppgift att hantera ”all source intelligence” måste det vara av stor vikt att kunna lägga in nya typer av data i databasen efter behov, men också att lätt kunna göra utdrag, sammanställningar och andra analyser och beräkningar på alla slags data. Utan tillgång till ett kraftfullt frågespråk måste man antingen beskriva lösningen till varje nytt analytiskt problem i form av ett program eller ha tillgång till en mycket stor meny av olika standardanalyser. Den förstnämnda metoden att skriva ett program för varje ny uppgift som ska lösas kommer att kräva tillgång till exklusiva personalresurser under hela den tid systemet är i drift. Det kommer också att innebära förhållandevis långa väntetider och ett svårt underhållsproblem för att skapa återanvändbara bearbetningskedjor. Den senare modellen är svår att specificera och dyr att realisera men kommer ändå aldrig att kunna erbjuda samma uttryckskraft för datahantering som ett modernt och utbyggbart frågespråk.
3. Det faktum att relationsdatabastekniken fortfarande dominerar på marknaden för databassystem (”*the winner takes all*”) gör att olika slags kompetenser och andra resurser som behövs för utveckling, drift och underhåll av databaser är lättare att få tag på om man väljer ett sådant system.
4. Det fjärde argumentet är att system för OLAP blir allt kraftfullare och allt vanligare, och dessa system bygger idag så gott som uteslutande på relationsmodellen eller dess olika generaliseringar. Att man dessutom numera har möjlighet att välja s k vertikala databaser för dessa tillämpningar innebär att de klassiska relationsdatabasernas

kapacitet och prestanda kan överträffas med en eller två storleksordningar i sådana tillämpningar.

9.2 Vertikala databaser för analytiska tillämpningar

I ett experiment inom högenergifysik accelereras elementarpartiklar till hastigheter nära ljusets och bringas sedan att kollidera [SBI+09]. Dessa kollisioner genererar ett stort antal ytterligare partiklar. För varje kollision, kallad händelse (”*event*”), samlar man in ca 1-10 MB rådata. Det sker ungefär 10 kollisioner varje sekund, vilket motsvarar hundratals miljoner upp till några miljarder händelser per år. Händelsedata genereras också i storskaliga simuleringar. Sedan rådata samlats in undergår de en ”rekonstruktionsfas”, där varje händelse analyseras för att avgöra vilka partiklar den producerade och för att extrahera hundratals sammanfattande egenskaper (som händelsens totala energi, impuls och antal partiklar som bildats av varje typ).

För att illustrera begreppet vertikal kontra horisontell organisation av data betraktar vi en datamängd som består av en miljard händelser, var och en med 200 egenskaper, med värden etiketterade V0,1, V0,2, etc.

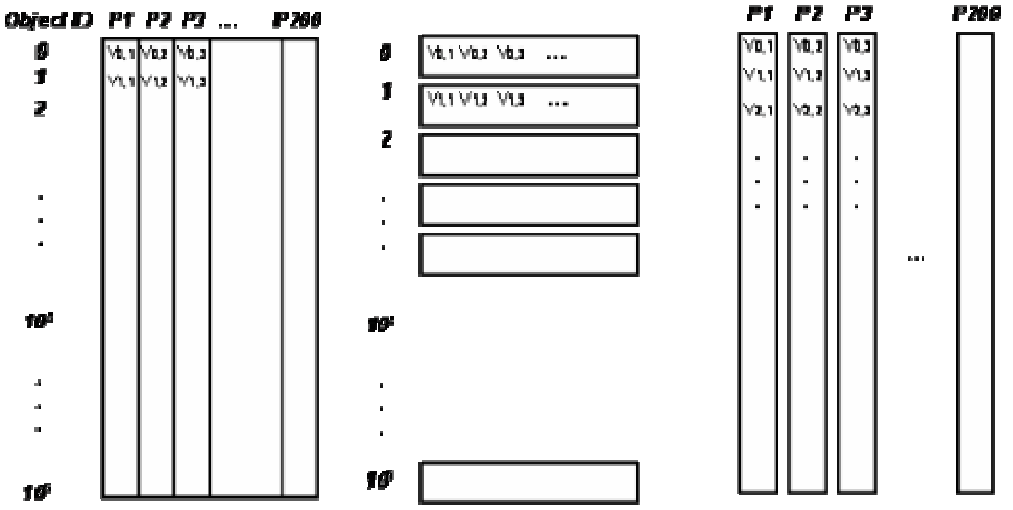
Konceptuellt kan hela samlingen av sammanfattande data representeras som en tabell med en miljard rader och 200 kolumner enligt figur 1(a). Med horisontell organisation av tabellen menas helt enkelt att det fysiska arrangemanget av data är radvis, som i figur 1(b).

Vanligen lagras hela tabellen på skivminnessidor eller filer som var och en innehåller flera rader. Med vertikal organisation menas att data är arrangerade kolumnvis som i figur 1(c).

Notera att hela kolumnen bestående av en miljard värden vanligen lagras på flera skivminnessidor eller i flera filer.

Antag att en användare vill komma åt de händelseidentiteter vars energi E är större än 10 MeV (miljoner elektronvolt) och som har ett pionantal, N_p, mellan 100 och 200, där ”pion” är en viss typ av elementarpartikel. Detta sökvillkor (eller predikat) kan skrivas som ((E > 10) & (100 < Np < 200)). I detta fall är det uppenbart att sökning genom de vertikalt organiserade data sannolikt kommer att gå snabbare, eftersom bara de data som ligger i kolumnerna för E och N_p behöver läsas in i minnet och jämföras med sökvillkoret. Däremot kommer den horisontella organisationen att kräva att hela tabellen läses.

Givet denna enkla iakttagelse kan man fråga sig varför relationsdatabassystem typiskt har byggts med horisontell organisation. Svaret har att göra med att de flesta tidiga databassystem konstruerades för att utföra transaktionsbearbetning där täta uppdateringar av slumpvis valda rader kunde förväntas. Sedan dess har förslag att i stället välja vertikal lagring framlagts vid flera tillfällen, men de har inte mött något nämnvärt gensvar från användare förrän på senare tid.



Figur 1. *Vertikal och horisontell databasarkitektur.*

9.3 Konstruktionsregler och användarbehov

Ett relativt sent tillskott till litteraturen om konstruktion av databassystem avsedda för vad dess författare kallar *complex analytics* är [MF04], som beskriver konstruktionsprinciperna bakom *Sybase IQ Multiplex*, ett parallellexekverande, multinods, vertikalt databassystem med delat minne [Syb08] vars främsta konstruktionsmål är att hantera arbetsflödet i storskaliga datalager på ett effektivt sätt.

Det sägs i [MF04] att det främsta konstruktionsmålet för transaktionshanterande, skrivorienterade databaser är att ”minimera den andel av de lagrade data som måste låsas för exklusiv åtkomst samt den tidrymd som låsen måste hållas stängda”. Enligt artikeln har detta mål generellt lett till följande uppsättning konstruktionsregler för transaktionshanterande, skrivorienterade databaser:

- Eftersom data vanligen läses och uppdateras en post i taget, bör de lagras radvis för att göra det möjligt att uppdatera varje post med en enda skrivoperation. Data bör också lagras på små skivminnessidor för att minimera mängden data som måste överföras mellan primär- och skivminne liksom för att minimera den andel av skivminnesfilen som behöver låsas under en transaktion.
- Index bör begränsas till ett fåtal attribut för att undvika låsning av hela indexträdstrukturer på skivminnet och därigenom tillfälligt omöjliggöra åtkomst till hela kluster av rader, vilket annars vore nödvändigt när index behöver uppdateras.
- Datakomprimering är vanligtvis inte lönsamt eftersom det ofta förekommer en blandning av olika datatyper och okorrelerade datavärden i en och samma tabellrad. Den CPU-tid som krävs för komprimering och dekomprimering kan därför inte återvinnas som minskad tidsåtgång för dataöverföring.
- Tillägg och borttagning av attribut och index kommer vanligen att vara tidskrävande operationer eftersom hela eller en stor del av de datasidor som används av den berörda relationstabellen kan påverkas.
- Uppdatering av ett attribut även styrt av ett enkelt predikat kan förväntas vara kostsamt eftersom en hel rad måste läsas och skrivas när ett enstaka attribut behöver uppdateras.

Men när det främsta kriteriet för konstruktion av ett databassystem i stället är att uppnå höga prestanda i komplexa analystillämpningar, bör denna uppsättning konstruktionsregler ändras på följande sätt:

- Genom att lagra data kolumnvis i stället för radvis är det möjligt att undvika att befatta sig med de skivminnessidor i en tabell vars data inte berörs av en viss fråga. Detta leder ofta till betydande prestandaförbättringar. Effektiviteten hos cacheminnet ökar eftersom ofta efterfrågade kolumner tenderar att ligga kvar där.
- Data kommer vanligen att läsas mycket oftare än de skrivs eller uppdateras, vilket innebär att ”investeringar” i CPU-tid för att skapa effektiva lagringsstrukturer har bättre utsikter att bli lönsamma. Data bör också lagras i stora skivminnessidor så att ett stort antal relevanta dataelement kan läsas in med en enda operation, vilket ger en genomsnittligt hög ”träffprocent”. Radvis lagring tenderar å andra sidan att missgynna stora sidor eftersom varje läsoperation också drar med sig attribut som inte är relevanta för den fråga som ska besvaras, vilket leder till låg träffprocent.
- Genom att använda versionshantering i stället för postvis låsning (vilket blir möjligt främst på grund av det vanligtvis mycket mindre antalet samtidiga uppdateringar) kommer varje fråga att se ett konsistent databastillstånd där inga, eller mycket få, lås någonsin behöver aktiveras. Versionshantering är också vad användare som arbetar med komplexa analyser behöver för att hålla reda på sina många olika analysspår genom databasen.
- Det är möjligt (fast troligen sällan nödvändigt) att indexera varje attribut eftersom sökningar dominerar starkt över uppdateringar och eftersom tillägg av index till ett attribut enbart kräver att detta attribut läses, inte hela raden.
- Datakomprimering kommer sannolikt att vara lönsamt eftersom data som hör till ett attribut med stor sannolikhet kommer att vara homogent och även autokorrelerat.
- Tillägg och borttagning av attribut i en tabell kan förväntas vara billigt eftersom bara relevanta data skulle behöva beröras. Uppdatering av ett attribut kommer sannolikt att vara relativt billigt eftersom inga irrelevanta attributvärden behöver läsas eller skrivas.

Liknande observationer gjordes långt tidigare [Sve79], men ignorerades länge av databasforskningens huvudfåra. Faktum är att dessa alternativa konstruktionsprinciper först på senare tid (2000-talet) mött seriöst intresse. Detta nyvaknade intresse har åtminstone delvis att göra med att många mycket stora databaser idag verkligen används som datalager för analytiska ändamål i beslutsstöds- och underrättelsetillämpningar inom affärslivet eller säkerhets- och försvarssektorerna, tillämpningar som ställer just sådana krav på databasprestanda som framhållits ovan.

Ett av de första system som utvecklats enligt de principer som formulerats ovan var *Cantor* [Sve79, KS83, KS86]. Cantorprojektet, som drevs vid FOA i olika former under perioden 1976-1994 var en föregångare beträffande studier av och koordinerad tillämpning i relationsdatasystem av många av de principer som nämnts ovan. I [Sve79] framfördes bl a följande användarkarakterisering:

- I vetenskaplig dataanalys är antalet samtidiga användare typiskt mycket mindre än i storskaliga kommersiella tillämpningar men å andra sidan tenderar användarna att ställa mer komplexa, oftast oförutsedda, frågor till systemet.
- En mer automatisk respons på komplexa användarbeställningar krävs av systemets komponenter, eftersom inga systemspecialister eller databasspecialister normalt är tillgängliga.
- Ett system bör vara transportabelt till många olika typer av datorsystem, inklusive medelstora datorer, eftersom många användare med vetenskapliga dataanalystillämpning-

ar föredrar att arbeta med en egen dator som är reserverad för insamling och analys av mätdata.

9.4 Arkitektoniska möjligheter för läsoptimerade, vertikala databaser

Intresset för vad som har kallats *läsoptimerade* databassystem växer för närvarande snabbt [SAB+05, HLA06]. Detta är system avsedda för spontan frågeverksamhet mot stora mängder data som innefattar måttliga mängder uppdateringar eller kanske i vissa tillämpningar inga alls.

Datalager representerar en klass läsoptimerade system där satsvis laddning av nya data utförs periodiskt, följt av en relativt lång period av ren frågeverksamhet utan uppdateringar.

Tidigt intresse för denna typ av system visades från olika statistiska och naturvetenskapliga tillämpningar såsom epidemiologiska, farmakologiska och andra dataanalytiska studier inom medicinen [WFW75], men också tillämpningar inom underrättelseanalys [BSS97].

Ett viktigt begreppsmässigt steg förknippat med användningen av vertikala läsoptimerade databassystem är att en hel rad nya arkitektoniska möjligheter öppnar sig. Nedan följer en uppräkning av ett antal sådana möjligheter. Notera att bara en del av dessa tekniker har funnit tillämpning i de system som f n finns på marknaden.

- *kolumnvis lagring* av data i stället för den konventionella radvisa lagringsstruktur som används i de allra flesta relationsdatabassystem (RDBMS) kan eliminera onödiga läs- och skrivoperationer om bara en del av kolumnerna i en tabell berörs av en fråga,
- *klustring*, speciellt *sortering*, av attributvärden kan leda till snabbare sökning över kolumndata,
- olika slag av "lättviktskomprimering" av data: *minsta bytestorlek, kodning med ordlista*, *differenskodning av sekvenser* m fl kan reducera mängden data som behöver överföras mellan sekundär- och primärminne,
- *svitlängdskompression* (*run-length encoding*, *RLE*) för fullständigt eller partiellt sorterade kolumner kan också minska mängden data som behöver överföras,
- *dynamiskt optimerade kombinationer* av olika sekvensiella komprimeringstekniker kan minska mängden överförda data och reducera tiden som åtgår för optimering,
- *B-trädsvarianter eller andra indexeringstekniker* konstruerade för att effektivt lagra och återvinna datablock av varierande fysisk längd tabellkolumner kan vara nödvändiga för att effektivt kunna utnyttja datakomprimeringsmetoder,
- *algoritmer för konjunktiv sökning, joinoperationer och mängdalgebraoperationer* som kan utnyttja den kolumnvisa lagringsformen och arbeta direkt på komprimerade data, ett attribut i taget,
- *lat dekomprimering av data*, dvs. data dekomprimeras bara när de behövs av en konsument i stället för så snart de överförts till primärminnet, är ett krav om sådana algoritmer som arbetar direkt på komprimerade data ska kunna utnyttjas,
- *komprimerade listor av tupelidentifierare* behövs för att representera intermediära och slutliga resultat i sådana algoritmer,
- *vektoriserade operationer* på dataströmmar tillsammans med en arkitekturparadigm som är baserad på *vektoriserade dataflödesgrafer* gör det möjligt att minska olika overheadkostnader i programkoden, som kostnader för proceduranrop, och åstadkomma ef-

fektiv avkodning av algebraiska uttryck genom interpretering istället för genom kompilering till lågnivåkod,

- *specialkonstruerade buffringsmetoder* för metadata och resultat från enkla transaktioner i ett separat cacheminne; frågor mot metadata har liksom enkla transaktioner ett åtkomstmönster som i allmänhet inte är väl anpassat till en kolumnvis lagringsform.

9.5 Kommersiella system för analytiska tillämpningar

Två ”vertikala” datalagersystem som idag erbjuds kommersiellt är *Vertica* [Ver08] och *Sybase IQ Multiplex* [Syb08]. Intresset för denna typ av system stimulerades på ett avgörande sätt när *MonetDB* presenterades 1999 [BK99]. MonetDB är ett vertikalt dataanalyssystem med öppen källkod från CWI (Centrum Wiskunde & Informatica) i Amsterdam. Se vidare [SBI+08].

10 Organisation av datalager för analysändamål

Försvarsmaktens användningsbehov av underrättelsedatabaser har knappast karaktären av avtappning av operationella data. Ändå påminner underrättelsedatabaser mycket om datalagertillämpningar, låt vara att de skapas och fylls på med hjälp av andra processer än de som är typiska i den kommersiella företagsvärlden. Det är dessa processer man måste förstå och analysera, och eventuellt också påverka, om man ska kunna bygga användbara underrättelsedatabaser för praktiskt bruk (till skillnad från demoprototyper). Utan en kontinuerligt verkande insamlingsprocess, som dessutom äger rum på ett strukturerat, dokumenterat och tillräckligt enhetligt sätt, kan inga användbara underrättelsedatabaser i längden existera.

Därför vore det troligen ett misstag att börja en utvecklingsprocess med att välja teknisk plattform i form av ett databassystem. Först behöver man beskriva och avgränsa tillämpningen och dess samlade behov av tekniskt stöd. För detta väljer vi här underrättelsetjänst i internationella fredsbevarande (och kanske ”fredsframtvingande”) insatser. Den organisatoriska nivån är också viktig; bataljons- eller brigadnivå är väl den mest troliga för NBGs del. NBGs dataförsörjningsbehov är i första hand insamling, värdering och analys av operativt och taktiskt verksamhetskritiska underrättelser. Data består av personuppgifter, sociala nätverk och organisationer av betydelse för insatsmiljön, och särskilt aggressivt oppositionella grupper, ”organiserad” kriminalitet (dvs. systematiskt bedriven, storskalig, grov kriminalitet, oberoende av hur den är strukturerad), indikatorer, observerad militär utrustning och militärt uppträdande, kritiska samhällsresurser och deras fördelning i befolkningen, geografiska, kulturella, religiösa, etniska, demografiska, politiska, historiska förhållanden, grupper och sammanslutningar som kan antas vara av betydelse, med mera. FOI-rapporterna [EHS+04][ESY05][HLM+08] beskriver på ett generellt plan syfte och struktur hos underrättelsedata och underrättelsedatabaser ur flera olika aspekter.

Hur går det till att skapa en beskrivning av insatsområdet enligt detta mönster? Till att börja med kommer man att behöva texter och multimediala beskrivningar av dessa förhållanden ur ett nutidshistoriskt och politiskt perspektiv. I nästa steg måste man skapa eller skaffa strukturerade versioner av dessa beskrivningar, kanske med användning av ett semantiskt sök- och referenssystem eller liknande hjälpmedel.

När insatsstyrkan har kommit på plats, övergår en stor del av ansvaret för den fortsatta databasuppbyggnaden till bataljonens/brigadens underrättelseenhet, även om viktig bakgrundsinformation även fortsättningsvis kommer att tillföras från Stockholm och/eller annan ledningsplats i Europa. Underrättelseinhämtningen kommer i samband med detta att övergå från en strategisk fas till en taktisk/teknisk. Rapportering från spaningspatruller och civila uppgiftslämnare, liksom uppgifter från lokala myndigheter, kan nu förväntas få stor betydelse. Se t ex [Sve08] för en beskrivning av hur samverkansteam i Kosovo arbetar med insamling av information.

FM:s Analytical Concept Paper *Knowledge Support* [Lar08] är en omfattande och ganska akademisk-filosofisk beskrivning av bakgrund, informationsbehov och arbetssätt vid insamling, bearbetning och delgivning av underrättelseinformation i Försvarsmaktens internationella insatsorganisation på operativ nivå. Dokumentet innehåller många värdefulla observationer och principiella påpekanden men ter sig för en tekniker som väl abstrakt och i stort behov av genomarbetade konkreta användningsfall. Men återigen, de viktiga principer och allmänna iakttagelser som görs i dokumentet måste bevaras under den förmodligen långa och komplexa process som kommer att behövas för att praktiskt förverkliga dessa idéer.

En bärande tes i dokumentet är behovet av att använda en systemansats vid beskrivningen (modelleringen) av de situationer och fenomen man har att hantera i underrättelsearbetet. Däremot finns inte mycket av konkreta förslag till hur detta ska gå till.

Den bästa ansatsen för ett lyckat införande är ofta att välja en billig och enkel startlösning ”off-the-shelf” (eller i forskningsvärlden oftare ”open source”) och låta tillämpningens utveckling styra när och hur systemet ska uppgraderas och ny teknik tillföras. För att klara detta behöver problemägaren ha tillgång till rätt kompetens under hela uppbyggnadsfasen.

För en systemtekniker ligger det nära till hands att föreslå att man använder RUP-metodiken för att få tillgång till de verktyg som behövs för att göra system- och processmodeller som kan kommuniceras, utvecklas och diskuteras i arbetsgrupper samt arkiveras och distribueras till en större krets när så behövs. Om industristandarden RUP och ISO-standarden UML (Unified Modeling Language), som RUP bygger på, inte anses lämpliga för tillämpningen kan man i alla fall behöva se till att deras viktigaste begrepp och metoder får motsvarigheter i den nya metodiken.

Ett näraliggande men betydligt mer systematiskt utvecklat sätt att representera s k Mission Space Models beskrivs i [MAK08]. Det bygger på användning av ontologier som kan representeras och visualiseras grafiskt med stöd av konventionerna i UML. Figur 5.14 i [MAK08] visar som ett konkret exempel hur en gisslansituation i Försvarsmaktens scenario ”Bogaland” kan tänkas bli representerad.

Å andra sidan är det inte säkert att all den generalitet som UML erbjuder är behövlig eller önskvärd i alla situationer. Om man ska modellera ett socialt nätverk är det troligen bättre att utgå ifrån etiketterade grafer, som kan redigeras med ett särskilt verktyg, än från en generell UML-modell. Om man exempelvis vill bygga upp influensdiagram eller Bayesianska nätverk (ett enkelt exempel på ett sådant presenteras i [FMS+08]; en modern lärobok i ämnet är [KM08]), kommer man att behöva verktyg som är speciellt anpassade för detta ändamål, som helst dessutom är anpassade till editorn för sociala nätverk, etc. Samtidigt är det troligt att redigering av olika typer av modellstrukturer bäst görs inom ett och samma programmatiska ramverk, så att olika tillämpningar och analysmetoder delar på redigerings- och presentationsfunktioner i största möjliga utsträckning.

Det kommer inte heller att bli lätt att skapa dessa förmågor på nytt i en helt ny programvaruomgivning, såvida man inte kan utgå ifrån något liknande system

som bygger på öppen källkod, t ex OpenUP – men om just detta system har acceptabel grundfunktionalitet vet vi inte. En idé skulle kunna vara att skapa systemmodeller med hjälp av något verktyg liknande OpenUP, och sedan lagra dessa modeller tillsammans med deras ontologier och taxonomier i en tillhörande databas. I detta sammanhang är det värt att notera att kapitel 2 i boken [CZ01] just handlar om avbildning av UML-diagram på objektrelationella scheman i *Oracle 8*.

10.1 Data ownership

En särskild problematik som måste hanteras när man bygger upp databaskapacitet inom stora organisationer som FM och MUST är vem eller vilka som har ansvar för att styra utvecklingen och utnyttjandet av organisationens verksamhetskritiska informationsresurser, ”*data ownership*”. Man behöver vara särskilt uppmärksam på tekniska argument för den ena eller andra databaslösningen som mycket ofta är baserade på ”tillämpningens behov”, ett vanligt sätt att argumentera, i alla fall när det gäller ODBMS. Men i sammanhang där databasen/databaserna som ska administreras sammantagna betjänar en hel organisation och försörjer den med verksamhetskritisk, ofta strategisk, information, kan vanligtvis inte en viss ”tillämpning” tillåtas avgöra hur databassystem eller databasarkitektur ska väljas. I stället är organisationens verksamhetskritiska databaser dess viktigaste informationstillgång som måste planeras, utvecklas, skyddas och kontrolleras som den centrala resurs den är.

10.2 Hur kan FMs underrättelsedatabas vara strukturerad?

Frågan om FMs underrättelsedatabas ska vara strukturerad som en ODBMS, en RDBMS, en ORDBMS (objektrelationell databas, dvs en relationsdatabas utvidgad med OO-begrepp, se ovan och [SMB99]), ett datalager (eventuellt *vertikalt strukturerat*, se ovan), ett medlarsystem, ett socialt-nätverkssystem eller på något annat sätt är inte lätt att avgöra. Några viktiga aspekter på ett sådant beslut är följande:

Vilka resurser och vilket ansvar har den organisation som upprätthåller databasen?

En organisationsform som har kompetensmässiga förutsättningar att ge ett bra resultat är att användarna skaffar sig tillgång till en eller flera analytiker som är väl utbildade i matematik, statistik och datavetenskap och har eller får specialutbildning i metodik för sökning och analys i datalager.

Det är av stor vikt att dessa analytiker får åtminstone samma ansvar och rättigheter som övriga användare då det gäller att välja metoder och frågeställningar för analysen, annars kommer man inte att kunna tillgodogöra sig deras kreativa förmåga i full utsträckning.

Hur stor funktionell bredd ska systemet täcka? Innehåller databasen exempelvis både geografisk information, information om egna förband och underrättelseinformation? För att få ner de funktionsvisa kostnaderna, särskilt för kvalificerad utvecklingspersonal, bör databasen täcka ett så brett användningsområde som möjligt.

Systemets flexibilitet är en annan viktig aspekt. Om man kräver att det ska gå att hantera nya problemtyper allt eftersom insatsernas karaktär förändras, så måste systemet kunna anpassas efter olika användarkrav.

Graden av flexibilitet i användningen av ett sådant system har i första hand att göra med hur starkt anpassat systemet är till en given klass av tillämpningar. Om vi tar ett av de *Objectivity*-baserade system som US DoD utvecklat som exempel, *Analyst Support Architecture* (ASA), så är det troligt att det är ett system som saknar frågespråk av SQL-typ och istället erbjuder en omfattande uppsättning menyer eller liknande valmöjligheter för olika frågeställningar och beräkningar.

Om man vill lägga till en ny funktion till ett sådant system så måste det göras av programmerare, på beställning av en person som ansvarar för systemets vidareutveckling och disponerar resurser för detta.

I ett helt eller delvis frågespråksbaserat system läggs i stället ansvaret direkt på användarna, vilket inte behöver betyda att alla användare har samma ansvar för frågeformuleringen. En nackdel med ett sådant förfarande kan förstås vara att användargruppen inte alltid har rätt kompetens för sådan utveckling, även om den nästan definitionsmässigt är betydligt enklare än konventionell programmering.

Hur man än väljer att organisera sin metodutveckling, kommer man att behöva ha förmåga att hantera problemtyper som inte förutsågs då systemet beställdes. Det är högst troligt att man upplever stark tidspress när en ny, viktig problemtyp dyker upp; till en början vet man förmodligen inte ens hur problemet ska angripas. Att så snart ett nytt problem behöver lösas tvingas anlita en hel utvecklingsgrupp, bestående av minst en systemanalytiker och en eller flera programmerare som utvecklar ny kod i systemet, kan vara starkt frustrerande.

Hur sker datainsamlingen och inmatningen i databasen?

Hur många samtidiga användare behöver systemet dimensioneras för och vilka typer av svar på frågor och andra bearbetningar begär dessa från systemet? Har varje användare tillgång till en egen, modern klientdator?

Om man tvingas uppfylla stränga krav på säkerhet och tillgänglighet alla dygnets timmar kommer systemet troligen att behöva vara både ganska stelt och ganska (eller mycket) dyrt att utveckla och underhålla. Ju högre säkerhetskraven ställs, desto färre system och desto mindre flexibla lösningar kan man troligen räkna med att få.

10.3 Exempel: HiTS-ISAC-projektet

Om vi tar EU PASR-projektet HiTS-ISAC [FMS+08] som exempel, visste vi från början att analys av sociala nätverk skulle ha en central roll, liksom integration av ”*legacy databases*” med hjälp av databasmedlare (i form av *Denodo Virtual DataPort*). Projektets databaser var testexempel med obetydliga datavolymer, vilket dock var tillräckligt för att demonstrera de metodmässiga principerna. Prestandatester kunde inte göras med de i projektet tillgängliga resurserna.

Vårt viktigaste syfte med HiTS-ISAC-projektet var att uppnå och sprida förståelse av hur dataflödet i en avancerad, modern polisunderrättelsetillämpning skulle kunna se ut och vilka krav på funktionalitet som en sådan tillämpning ställer. HiTS-ISACs demosystem är ur arkitektursynpunkt tillräckligt komplett för att det ska vara möjligt att analysera hur ett fullskaligt driftsystem baserat på likartade principer kan byggas. För att en sådan studie ska vara meningsfull måste analytikern förstås vara övertygad om att systemets huvudprinciper är väl valda för tillämpningen, eller annars relativt enkelt kan bytas ut för att tillfredsställa nya behov som har identifierats.

När det som här främst handlar om att skapa och underhålla ett datalager, måste man visa att det kan fyllas med relevanta data i den takt och utsträckning som tillämpningen kräver, samt naturligtvis också möjliggöra uttag, sökningar och presentationer i den omfattning och av den komplexitet som användarna förväntar sig. Vidare måste man se till att systemet har de integritets- och prestandaegenskaper som krävs, och att det är tillräckligt skalbart, dvs kan absorbera förutsebar framtida utbyggnad, både beträffande lagringskapacitet och förmåga att stödja ny funktionalitet, inte minst med effektiva sökmetoder.

Det är stor skillnad mellan att skapa en väl fungerande prototyp och att bygga ett system som ska fungera i drift under en längre tid, vanligtvis minst 10 år. En prototyp kan vara lyckad och ett demoprojekt framgångsrikt om man kan visa att den har vissa önskade egenskaper som är (eller uppfattas som) nya och

värdefulla. Driftssystemet måste framför allt erbjuda en tillräcklig grundfunktionalitet, rimlig användbarhet och god driftsekonomi över tiden. Speciell funktionalitet, exempelvis för avancerad analys, utvecklas i många fall bäst evolutionärt och behöver inte finnas eller i varje fall inte vara komplett i systemets första utgåva.

Litteratur

[AG76] Alsberg, P.A., Gray, J.D., A principle for resilient sharing of distributed resources, *Proc. Int. Conf. on Software Engineering*, 627-644 (1976).
[A+89] Atkinson, M. et al., *The Object-Oriented Database System Manifesto* (1989). http://www-2.cs.cmu.edu/People/clamen/ OODBMS/Manifesto/htManifesto/Manifesto.html Accessed 2008-09-11.
[Bar05] Barry, D., Grehan, R., *Introduction to ODBMS*, http://www.odbms.org/about_contributors_barry.html Accessed 2008-09-11.
[BH08] Bernstein, P.A., Haas, L.M., Information Integration in the Enterprise, *Comm. ACM* 51(9), 72-79 (2008).
[BK99] Boncz, P.-A., Kersten, M.L.: MIL primitives for querying a fragmented world, *The VLDB Journal* 8(2), 101-119 (1999).
[BSS97] Bergsten, U., Schubert, J., Svensson, P.: Applying data mining and machine learning techniques to submarine intelligence analysis, i *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA, USA (1997).
[Cat94] Cattell, R.G.G., *Object Data Management*, Addison-Wesley (1994).
[Cod70] Codd, E. F., A Relational Model for Large, Shared Data Banks, *CACM*, 13(6) (June 1970).
[CZ01] Chaudhri, A.B., Zicari, R., *Succeeding with Object Databases – A Practical Look at Today's Implementations with Java and XML*, Wiley Computer Publishing, New York, NY, USA (2001).
[EHS+04] Eklöf, M., Hörling, P., Svan, P., Suzic, R., Yi, C, *Information & Knowledge Management for NBI (Network-Based Intelligence),* FOI-R—1417—SE, FOI, Stockholm (November 2004).
[EN06] Elmasri, R., Navathe, S. B., *Fundamentals of Database Systems, 5th ed.* Pearson-Addison Wesley, Boston, MA (2006).
[ESY05] Eklöf, M., Suzic, R., Yi, C., *Network-Based Intelligence (NBI) – Knowledge Base Development,* FOI-R—1757—SE, FOI, Stockholm (October 2005).
[FMS+08] Ferrara, L., Mårtenson, C., Svenson, P., Svensson, P., Hidalgo, J., Molano, A., Madsen, A.L:, Integrating Data Sources and Network Analysis Tools to Support the Fight Against Organized Crime, i C. C. Yang *et al*. (red.): *ISI Workshops*, LNCS 5075, 171-182. Springer-Verlag Berlin Heidelberg (2008).
[FNP+00] Fjällström, P.-O., Neider, G., Persson, M., Risch, T., Svensson, P., *Arkitekturprinciper för informationsöverlägsenhet i framtidens ledningsstödsystem*, FOA-R--00-01435-505—SE, Linköping (Februari 2000.)
[FV07] Felici, G., Vercellis, C. (red.), *Mathematical Methods for Knowledge Discovery and Data Mining*, Idea Group Reference (2007).
[GBE+00] Gueting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S:, Lorentzos, N.A., Schneider, M., Vazirgiannis, M., A foundation for representing and querying moving objects, *ACM Transactions on Database Systems (TODS)*, 25(1), 1-42 (2000).
[HK06] Han, J., Kamber, M., *Data Mining – Concepts and Techniques*, 2nd ed., Elsevier Morgan Kaufmann (2006).
[Her06a] Herring, R., *Simple Feature Access Part 1: Common Architecture, version 1.2.0,* OpenGIS Consortium (2006).
[Her06b] Herring, R., *Simple Feature Access Part 2: SQL Option, version 1.2.0,* OpenGIS Consortium (2006).
[HLA+06] Harizopoulos, S., Liang, V., Abadi, D.J., Madden, S.: Performance tradeoffs in read-optimized databases, i *Proc. 32nd Int. Conf. on Very Large Databases*, September 12-15, Seoul, Korea (2006).
[HLM+08] Hörling, P., Lundin, M., Mårtenson, C., Svenson, P., *Informationsmodeller och indikatorer för situations- och hotbeskrivning*, FOI-R—2535—SE, FOI, Stockholm (Juni 2008).
[JLV+02] Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P., *Fundamentals of Data Warehouses*, 2nd ed., Springer-Verlag, Berlin Heidelberg (2002).

[KM08] Kjaerulff, U.B., Madsen, A.L., *Bayesian Networks and Influence Diagrams. A Guide to Construction and Analysis*. Springer-Verlag, New York (2008).
[KS83] Karasalo, I., Svensson, P.: An overview of Cantor – a new system for data analysis, i *Proc. 2nd Int. Workshop on Statistical Database Management (SSDBM)* (1983).
[KS86] Karasalo, I., Svensson, P.: The design of Cantor – a new system for data analysis, i *Proc. 3rd International Workshop on Statistical and Scientific Database Management (SSDBM)* (1986).
[Lar08] Larsson, A., *Knowledge Support*. Analytical Concept Paper, Försvarsmakten, Stockholm (2008).
[MAK08] Mojtahed, V., Andersson, B., Kabilan, V., *BOM and DCMF*. FOI-R—2362--SE, FOI, Stockholm (Januari 2008).
[MF04] MacNicol, R., French, B.: Sybase IQ Multiplex – Designed for Analytics, i *Proc. of the 30th Int. Conf. on Very Large Databases*, Toronto, Canada (2004).
[MZ08] Malinowski, E. & Zimanyi, E., *Advanced Data Warehouse Design*, Springer-Verlag, Berlin Heidelberg (2008).
[New06] Neward, T., *The Vietnam of Computer Science*, http://www.odbms.org/about_contributors_neward.html Accessed 2008-09-11.
[ODB08] http://www.odbms.org Accessed 2008-09-11.
[PL91] Pu, C., Leff, A., Replica control in distributed systems: An asynchronous approach, *Proc. 1991 SIGMOD Conf.* (1991).
[SAB+05] Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E.J., O'Neil, P.E., Rasin, A., Tran, N., Zdonik, S.B.: C-Store: A Column-oriented DBMS, i *Proc. 31st Int. Conf. on Very Large Databases*, Trondheim, Norway, August 30 - September 2, 553-564 (2005).
[SBI+09] Svensson, P., Boncz, P., Ivanova, M., Kersten, M., Nes, N., Emerging database systems in support of scientific data, kap. 11 i Shoshani, A., Rotem, D. (red.), *Scientific Data Management: Challenges, Existing Technology, and Deployment*, Taylor & Francis Publishers, London (2009).
[SH08] Svensson, P., Harrie, L., Lagring av geografiska data, kap. 7 i Harrie, L. (red.), *Geografisk Informationsbehandling – metoder och tillämpningar*, 4e uppl., Lönegårds förlag, Lund (2008).
[SKS96] Silberschatz, A., Korth, H., Sudarshan, S., *Database System Concepts*, McGraw-Hill (1996).
[SMB99] Stonebraker, M., Moore, D., Brown, P., *Object-Relational Databases – Tracking the Next Great Wave*, Morgan Kaufmann Publishers, Menlo Park, CA (1999).
[Sve78] Svensson, P., *Om forskarens datormiljö*. FOA rapport C 20215-D8, Försvarets forskningsanstalt, Stockholm (1978)..
[Sve79] Svensson, P.: *Contributions to the design of efficient relational data base systems. Summary of the author's doctoral thesis*. Report TRITA-NA-7909, Kungliga Tekniska Högskolan, Stockholm (1979).
[Sve88] Svensson, P., Database Management Systems for Statistical and Scientific Applications: Are commercially available DBMS good enough? I Rafanelli, Klensin, Svensson (red.), *Statistical and Scientific Database Management.* Lecture Notes in Computer Science 339. Springer-Verlag Berlin Heidelberg (1988).
[Sve08] Svensson, E., *De svenska samverkansteamen. En utvärdering av de svenska samverkansteamen i Kosovo och Bosnien-Hercegovina*, FOI-R—2554—SE, FOI, Stockholm (Augusti 2008).
[Syb08] http://www.sybase.com/products/datawarehousing/sybaseiq Accessed 2008-05-22.
[TSK05] Tan, P.-N., Steinbach, M., Kumar, V., *Introduction to Data Mining*, Addison Wesley (2005).

[Ver08] http://www.vertica.com/product/relational-database-management-system-overview Accessed 2008-05-22.
[WFW75] Wiederhold, G., Fries, J.F., Weyl, S.: Structured Organization of Clinical Data Bases, i *Proc. of the National Computer Conference*, AFIPS Press (1975).
[ÖV91] Özsu, M.T., Valduriez, P., *Principles of Distributed Database Systems*, Prentice-Hall (1991).

Bilaga 1. Detaljerade krav på objektdatabaser

Nedanstående teser är hämtade från [A+89] och [Cat94], där också mer utförliga kommentarer går att hitta.

- T1 – **Persistens**. Ett ODBMS måste lagra data på sådant sätt att de bevaras när exekveringen av ett applikationsprogram som accessar dessa data har avslutats.
- T2 – **Schema**. Ett ODBMS måste tillhandahålla ett *schema* över sina data. Användare måste kunna ta del av schemat som persistent data, och kunna skapa och ta bort typer i schemat.
- T3 – **Sekundär lagring**. Ett ODBMS måste erbjuda förmåga att effektivt hantera stora datavolymer genom sekundärminneshantering, användning av index, länkar och andra tekniker. Man bör dock inte förbise möjligheten att använda MMDBMS (main memory DBMS) i tillämpningar där de passar, eftersom ett MMDBMS kan utnyttja andra lösningar än sekundärminnesbaserade databassystem, bl a länkade listor även för data som refereras ofta. De flesta MMDBMS tycks dock fortfarande vara forskningsprototyper (MonetDB är dock ett undantag som har fått betydande spridning).
- T4 – **Transaktioner**. Ett ODBMS måste tillhandahålla transaktioner eller en likvärdig mekanism för att åstadkomma korrekt hantering av samtidig begäran av dataåtkomst från flera användare.
- T5 – **Säkerhet**. Ett ODBMS bör tillhandahålla mekanismer för auktorisation av användare och för att kontrollera åtkomst av data.
- T6 – **Objekt**. Ett ODBMS måste tillhandahålla förmåga att lagra objekt. Användaren måste kunna referera till dessa objekt med en primärnyckel och med en systemgenererad unik objektidentifierare (OID). OIDer behöver inte vara persistenta. Strukturerade eller eventuellt hashade surrogat-OID är de enda effektiva representationerna.
- T7 – **Attribut**. Ett ODBMS måste tillåta att objekt har attribut som kan vara enkla literalvärden som heltal och strängar, stora värden som bitvektorer och text, och aggregatvärden som mängder och listor. Ett ODBMS tillhandahåller i regel alla dessa datatyper.
- T8 – **Relationer**. Ett ODBMS måste tillhandahålla en mekanism för att representera relationer mellan objekt. Det måste vara möjligt att definiera relaterade objekt med hjälp av primärnyckel eller OID.
- T9 – **Referentiell integritet**. Mekanismen för att representera relationer måste upprätthålla referentiell integritet och undvika att uppdateringsanomalier uppstår när objekt tas bort eller relationer definieras om.
- T10 – **Sammansatta objekt**. Ett ODBMS bör tillhandahålla förmåga att aggregera objekt för att kunna utföra sådana operationer som borttagning, kopiering och samtidighetskontroll. Om aggregering tillhandahålls så måste den kunna utnyttjas för att åstadkomma fysisk klustring av data. Det måste vara valfritt att utföra operationer på hela aggregat.
- T11 – **Procedurer**. Det bör vara möjligt att associera procedurer med objekttyper.
- T12 – **Arv**. Ett ODBMS måste tillåta att egenskaper ärvs från supertyper. Multipelt arv, multipla typer hos objekt och instansundantag (en-instans-typer) är valfritt att tillhandahålla.
- T13 – **Typer som binds vid körtillfället.** Ett ODBMS bör tillåta dynamisk bindning och enkla typändringar under körning.
- T14 – **Schemautveckling**. Ett ODBMS bör tillhandahålla verktyg som underlättar för användaren att göra förändringar i schemat. Förmåga att vid körtillfället byta namn,

lägga till och ta bort objekttyper och egenskaper hos objekt krävs i T13 för att möjliggöra T14.

- T15 – **Frågespråk**. Ett ODBMS måste tillhandahålla ett deklarativt frågespråk som kan användas för ad hoc-operationer av en slutanvändare.
- T16 – **Fysiskt dataoberoende**. Frågespråksprocessorn måste tillhandahålla en hög grad av fysiskt dataoberoende och automatiskt kunna välja de fysiska åtkomstmetoder som behövs för att besvara en fråga.
- T17 – **Frågor i program**. Ett ODBMS måste stödja ett programmeringsspråk som innehåller frågespråket som en delmängd. Det måste vara möjligt att anropa frågespråket från programmeringsspråket och tvärtom.
- T18 – **Frågor föredras**. Program bör specificera dataåtkomst i frågespråket snarare än direkt i ett programmeringsspråk, när det är praktiskt möjligt att uttrycka den mängd av operationer och objekt som önskas i frågespråket. [Detta var en av de ursprungliga motiveringarna för relationsmodellen, till skillnad från tidigare system som utförde navigation genom data på fysisk nivå.]
- T19 – **Fullständighet hos frågespråk**. Datamanipuleringsspråket (frågespråket) måste tillhandahålla åtminstone samma förmåga som relationskalkyl med mängdresultat, och bör tillåta åtkomst till alla slags datastrukturer, inklusive mängder och mängder av mängder.
- T20 – **Fullständighet hos programmeringsspråk**. Ett språk för programmering av ett ODBMS måste erbjuda både beräkningsmässig fullständighet och resursfullständighet. [Det deklarativa språket (frågespråket) är vanligen inte tillräckligt kraftfullt för att kunna uttrycka alla operationer som behövs i tillämpningar. När så är fallet, är det nödvändigt att övergå till programmeringsspråket.]
- T21 – **Integration av programmeringsspråket**. Ett ODBMS bör vara integrerat med en existerande programspråksomgivning och därmed tillhandahålla en persistent utökning av ett programmeringsspråk som användarna redan känner till.
- T22 – **Döljande av egenskaper**. Ett ODBMS måste möjliggöra att egenskaper (attribut, relationer eller procedurer) som är externt synliga kan åtskiljas från interna egenskaper.
- T23 – **Avbildning av egenskaper**. Ett ODBMS måste tillåta att en användare definierar procedurellt virtuella egenskaper hos ett databasobjekt. Virtuella egenskaper är sådana som kan ”dölja” andra egenskaper. Det bör vara transparent för användaren om en egenskap är virtuell eller direktlagrad.
- T24 – **Portabilitet**. Ett ODBMS bör tillhandahålla förmåga att skriva program som också kan köras på andra ODBMS.
- T25 – **Kompatibilitet med SQL.** Ett ODBMS bör tillhandahålla bakåtkompatibilitet med SQL, eller växelstationer till och från SQL-databassystem.
- T26 – **Minsta accesskostnad**. Ett ODBMS måste minimera kostnaden för enkla operationer på data, som att läsa ett enstaka objekt.
- T27 – **Primärminnesutnyttjande**. Ett ODBMS måste maximera sannolikheten för att data ska befinna sig i primärminnet när de läses till applikationen. Det ska åtminstone tillhandahålla en datacache i applikationens virtuellminne och förmåga att klustra data i sidor eller segment som hämtas från skivminnet.
- T28 – **Fjärroperationer**. Det måste vara möjligt att få tillgång till data som är lokaliserade på en annan dator inom ett ODBMS. Det måste också vara möjligt att utföra operationer på avstånd, om sådan förmåga inte bekvämt erbjuds av operativsystemfunktioner. [Mer avancerad distribuerad databashantering kommer att krävas för tillämpningar som måste kunna utföra distribuerad transaktionshantering]
- T29 – **Databasdistribution**. Ett ODBMS bör tillhandahålla förmåga att exekvera frågor och program som accessar mer än en databas. Objekt i en databas bör kunna referera objekt i en annan databas.

- T30 – **Låsning**. Ett ODBMS måste tillhandahålla en låsmekanism, med granulariteten av ett objekt eller en skivminnessida, som tillåter att data checkas ut under långa perioder.
- T31 – **Versioner**. Ett ODBMS bör tillhandahålla primitiver för att hantera multipla versioner av data. Det kan också tillhandahålla kapacitet för konfigurationshantering.
- T32 – **Regler**. Ett ODBMS bör tillhandahålla en mekanism för exekvering av regler som kan utföra åtgärder när specificerade villkor uppfylls. Åtgärden som är associerad med en regel kan vara en godtycklig procedur (*trigger*) eller ett feltillstånd (*condition*).
- T33 – **Verktyg**. Ett ODBMS bör tillhandahålla verktyg på hög nivå för åtkomst av data och scheman samt för att ge programmeraren stöd vid generering av applikationer.

Bilaga 2. Överföring av data mellan objekt- och relationsdatabaser, O/R-M

Flera lösningar är möjliga på problemet att flytta data mellan objekt- och relationsdatabaser [New06]. Vissa av dem kräver något slags ”global” åtgärd inom databasvärlden, andra är mer åtkomliga för lokala utvecklingsgrupper.

1. **Uppgivande**. Utvecklare ger helt enkelt upp objekthantering helt och hållet, och återgår till en programmeringsmodell som inte orsakar objekt-relationell missanpassning. Även om detta är en oaptitlig lösning, är det ett faktum att objektorienterade ansatser i vissa sammanhang skapar mer prestanda- och designproblem problem än de löser, och det går då helt enkelt inte att tjäna in den investeringskostnad som krävs för att skapa en tillräckligt komplett domänmodell. Denna lösning eliminerar problemet, eftersom det inte uppstår någon missanpassning om det inte finns några objekt.
2. **Helhjärtat accepterande.** Utvecklarna ger upp relationsdatabaslagring helt och hållet, och använder i stället en lagringsmodell som passar till det sätt som deras favoritspråk uppfattar världen. Vissa objektdatabaser, t ex **db4o**-projektet, löser problemet elegant genom att lagra objekten direkt i sekundärminnet, vilket eliminerar många (men inte alla) problem; det finns inget “sekundärt schema”, t ex, eftersom det enda schema som används är definitionen av objekten själva. Även om många databasadministratörer skulle svimma vid tanken, är det fullt möjligt att tänka sig att utvecklare lagrar data i en form som är mycket enklare för dem att använda än den metod som förespråkas av databasadministratören. I en alltmer tjänsteorienterad värld, som skyr tanken på direkt åtkomst till data kan man i stället kräva att all dataåtkomst ska ske via accesstjänsten, så att lagringsmekanismen kapslas in och döljs för spejande blickar.
3. **Manuell avbildning.** Utvecklarna accepterar att problemet inte är så svårt att lösa trots allt, och skriver vanlig kod för relationell dataåtkomst som returnerar relationer till det objektorienterade språket, skannar dess tupler en i taget och flyttar över informationen till objekten efter behov. I många fall kan denna kod till och med genereras automatiskt av ett programverktyg som läser metadata, vilket kan eliminera en del av den principiella kritik som kan riktas mot denna teknik (nämligen att ”det blir alldeles för mycket kod att skriva och underhålla”).
4. **Accepterande av O/R-avbildningens begränsningar.** Utvecklarna accepterar helt enkelt att det inte finns något sätt att enkelt och effektivt eliminera O/R-missanpassningen, och väljer ett O/R-M-program för att lösa 80% (eller 50% eller 95% eller vilken procentsats som förefaller lämpligast) av problemet och utnyttjar SQL och relationsdatabasaccess (som t ex ”rå” JDBC eller ADO NET) för att ta sig igenom de avsnitt där ett mappningsprogram annars måste känna till vilken cachningsteknik som O/R-M-lösningen själv använder.
5. **Integration av relationsbegrepp i OO-språk.** Utvecklare accepterar att detta är ett problem som bör lösas av språket, inte av ett bibliotek eller ramverk. Under det senaste decenniet och längre ändå har lösningar av O/R-M-problemet inriktats på att försöka bringa objekten närmare databasen, så att utvecklarna kan koncentrera sig på att programmera i en enda paradigm (objektorienterat naturligtvis). Under senare år har emellertid ”scriptspråk” av olika slag blivit allt populärare, och vissa sådana språk, som Ruby, med starkt stöd för mängder och listor, har fött idén att en annan lösningsmetod kan vara bättre: flytta in relationsbegrepp (som i grunden är mängdorienterade) i de vanliga programmeringsspråken, så att det blir lättare att överbrygga gapet mellan ”mängder” och ”objekt”. Sådant utvecklingsarbete har hittills varit begränsat och mestadels rört sig om forskningsprojekt och ”udda” språk,

men flera intressanta ansatser har gjorts, t ex med funktionella/objekthybrider som Scala eller F#, men också direkt integration i konventionella OO-språk, som LINQ-projektet för Microsoft C# och Visual Basic. Ett sådant försök, som tyvärr misslyckades, var SQL/J-strategin, men även i det fallet var ansatsen begränsad och försökte inte inbegripa mängdvärda objekt i Java, utan bara göra det möjligt att förbehandla och översätta inbäddade SQL-anrop till JDBC med hjälp av en översättare.

6. **Integration av relationsbegrepp i ramverk.** Utvecklare medger att problemet går att lösa, men bara om man byter perspektiv. Istället för att lita till att språk- eller bibliotekskonstruktörer ska lösa problemet, väljer utvecklarna att se på ”objekt” på ett mer relationsorienterat sätt, och skapar problemorienterade ramverk som bygger direkt på relationella begrepp. Istället för att skapa en klass Person som håller sina instansdata direkt i objektets inre, kan man t ex skapa en klass som lagrar sina instansdata i en RowSet-(Java) eller DataSet-(C#)instans, som kan sättas samman med andra RowSets/DataSets till ett block som är lätt att flytta till en databas för uppdatering, eller som kan packas upp från databasen direkt in i de individuella objekten.