TRIAD Arbetsrapport K 14

Krav på IA
Nästa Generation Modellering
Avancerad utbildning för handledare
Katalogprinciper
Verktyg
Informationspridning

AD/Cycle I Information Model

Relationsdatabasmodellering

Rapport K nr 1: IRDS
Rapport K nr 2: IRDS Modeller och modellnivåer
Rapport K nr 3: Kopping begreppsmodell - relationsmodell
Rapport K nr 4: IBM:s Repository Manager- en Introduktion
Rapport K nr 5: IBM:s Repository Manager: Datamodelleringsbegreppen
Rapport K nr 6: IBM:s Repository Manager: Begreppsmodellering i Information Model
Rapport K nr 7: IBM Repository Manager: Attribut- och värdemodellering i Enterprise Submodel
Rapport K nr 8: Navigering i Repository
Rapport K nr 9: TRIAD Newsletter – IRDS inom ISO. Dagsläget
Rapport K nr 10: TRIAD Newsletter –ISO/IRDS. Händelseutvecklingen 91/92
Rapport K nr 11: Samverkan mellan resurskataloger – visioner eller behov
Rapport K nr 12: AD/Cycle I Information Model – Processer och och informationsflöden mellan processer
Rapport K nr 13: AD/Cycle I Information Model – Info Flows inom Processmodellen
Rapport K nr 14: AD/Cycle I Information Model – Relationsdatabasmodellering

Stig Berild
SISU & Sveriges Tekniska Attachéer

Rapporten är skriven i och för TRIAD delprojekt Katalogprinciper.

TRIAD-projektet drivs gemensamt av Televerket, Posten, Statskontoret, Ericsson Data Services och SISU, Svenska Institutet för Systemutveckling.
För vidare information kontakta SISU Informationscentrum, tel 08-752 16 00, fax 08-752 68 00, Box 1250, 164 28 Kista.

SISU

Läsarrapport för TRIAD rapport K 14: AD/Cycle I Information Model – Relationsdatabasmodellering

Datum:

Namn:

Företag:

returneras till:

SISU
Lars Bergman
Box 1250
164 28 Kista

Lämna gärna kommentar till enkäten på baksidan.
Kryssa om du kommenterar. ☐

SVARET önskas när du läst!

01 Jag har lämnat den vidare till följande person/er:

02 Jag har cirkulerat den till följande personer:

03 Jag har läst rapporten (ange ungefärlig tid om du kan)		
04 Värdet för mig i mitt jobb nu (Stort=3, ganska stort=2, något=1, inget=0)		
05 Värdet för mig i mitt jobb framöver (Stort=3, ganska stort=2, något=1, inget=0)		
06 Värdet för min allmänna kompetensutveckling (Stort=3, ganska stort=2, något=1, inget=0)		
07 Detaljeringsgraden var (För djup=3, lagom=2, för ytlig=1, ingen åsikt=0)		
08 Dispositionen av innehållet var (Bra=2, Inte bra=1, vet ej=0)		
09 Innehållets tillgänglighet (Lättillgängligt=3, krävde viss ansträngning=2, krävde stor ansträngning=1, vet ej=0)		
10 Språket (klart=3, något oklart=2, mycket oklart=1, vet ej =0)		
11 Illustrationerna som lässtöd (gav gott stöd vid läsningen=3, gav visst stöd vid läsningen=2, gav dåligt stöd vid läsningen=1, vet ej=0)		
12 Användningen av illustrationer (För få=3, Lagom=2, För många=1, Vet ej=0).		

Returneras till
SISU, Lars Bergman
Box 1250, 164 28 Kista

Läsarrapport för TRIAD rapport K 14: AD/Cycle I Information Model -Relations-databasmodellering

Jag har lämnat dessa uppgifter tidigare. ☐

Din befattning i korta ord

Din enhet i korta ord

Ditt personliga intresse i sammanhanget

Din erfarenhet i korta drag

Din utbildning

Din kommentar till rapporten i övrigt:

AD/Cycle Information Model:
Relationsdatabasmodellering

Innehåll

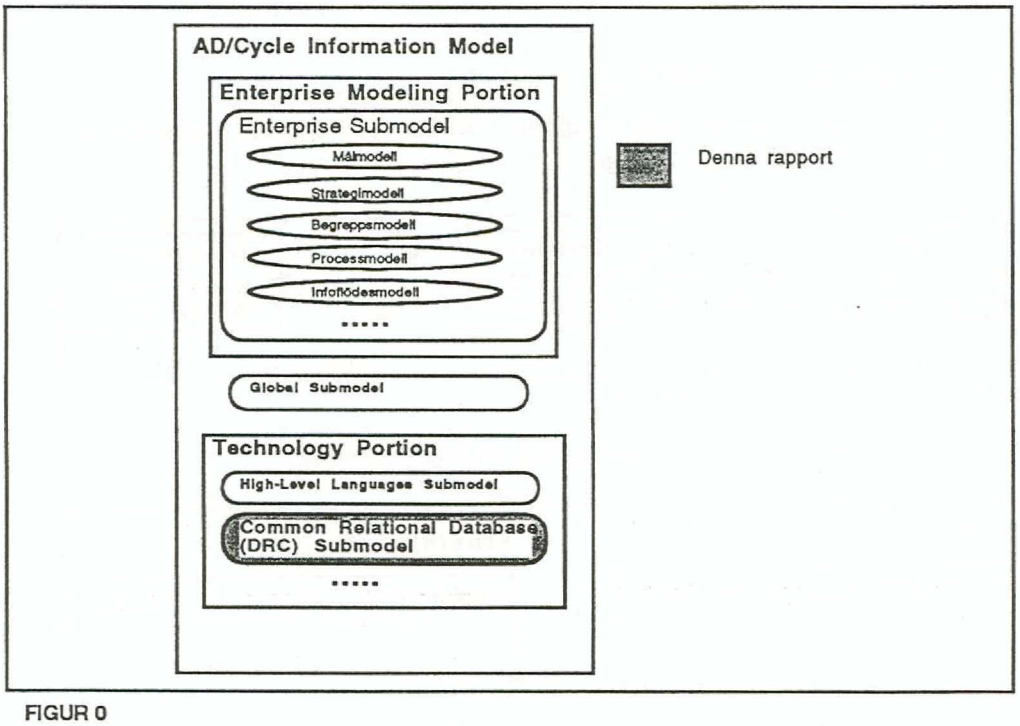
1. Inledning

1.1 Rapportens omfattning

IBMs AD/Cycle Information Model (IM) består av ett antal submodeller, grupperade enligt figur 0. Figuren visar endast några av submodellerna. Det finns fler. Nya tillförs med jämna mellanrum.

Denna rapport beskriver den submodell inom "technology portion", som går under benämningen "Common Relational Database (DRC) Submodel". DRC omfattar konstruktioner för att formulera den del av en relationsdatabasmodell som är produktoberoende. Relationsmodellen är ju i sin grundläggande definition helt produktoberoende. Produktberoende delar avses formuleras genom en, för varje produkt specifik, submodell. DRC avlastar på så vis de produktberoende submodellerna från de uttrycksmöjligheter som ändå är gemensamma för dem alla. "MVS Relational Database (DRM) Submodel" är ett exempel på en produktberoende submodell, som, tillsammans med DRC, innehåller källinformation för generering av en DB2 databasmodell. En grundläggande kännedom om relationsmodellen förutsätts.

Innehållet i rapporten svarar mot IM version 1, release 2, modification 2.



FIGUR 0

Technology Portion omfattar i dagsläget (version 1, release 2) drygt 130 entity types och nästan 550 relationship types.

1.2 Använd notation

Denna rapport använder samma notation som TRIAD K12. Komponenterna i begreppsmodellen skrivs i fet stil när de först introduceras. Entity types börjar med stor bokstav medan övriga bokstäver är gemena. Relationship types och attribute types skrivs alltid med små bokstäver och omgärdas med citationstecken. Notationen skiljer inte längre på typ och förekomst. Vad som gäller, framgår förhoppningsvis av sammanhanget.

Med verksamhetsmodell menar vi generellt förekomstnivå av Enterprise Submodel. En verksamhetsmodell kan alltså bestå av förekomster av processmodellen, infoflödesmodellen, begreppsmodellen eller en kombination av dessa. Exempel på verksamhetsmodellnivå anges kursiverat i texten. Ofta börjar de med stor bokstav.

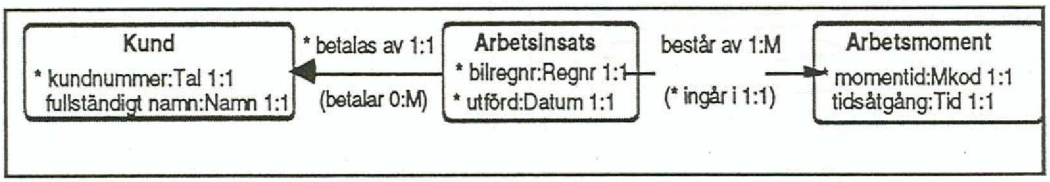
Graferna är som tidigare framställda med Business Modeler (BM).

1.3 Rapportens disposition

Avsnitt 2 beskriver hur relationsmodellens grundläggande begrepp modelleras i DRC. Views är ett kapitel för sig, nämligen avsnitt 3. Av olika skäl har man valt att lägga namn på olika företeelser som separata entity types, åtskilda från de företeelser de är namn på. Hur det ser ut, beskrivs i avsnitt 4. Observera att de modeller som visas i de tre första avsnitten har namnet integrerat som attribute type i respektive entity type, på vanligt sätt. Avsnitt 5 rymmer några avslutande reflexioner.

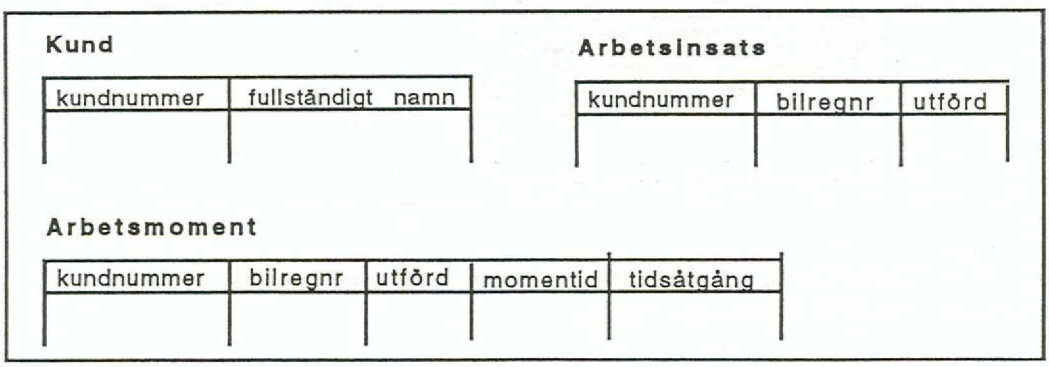
1.4 Exempelunderlag

Vi använder samma modell som i rapport TRIAD K13 för att exemplifiera olika delar av DRC-modellen. Se figur 1.



FIGUR 1

Figur 2 visar dess tänkta motsvarighet enligt en relationsmodell.



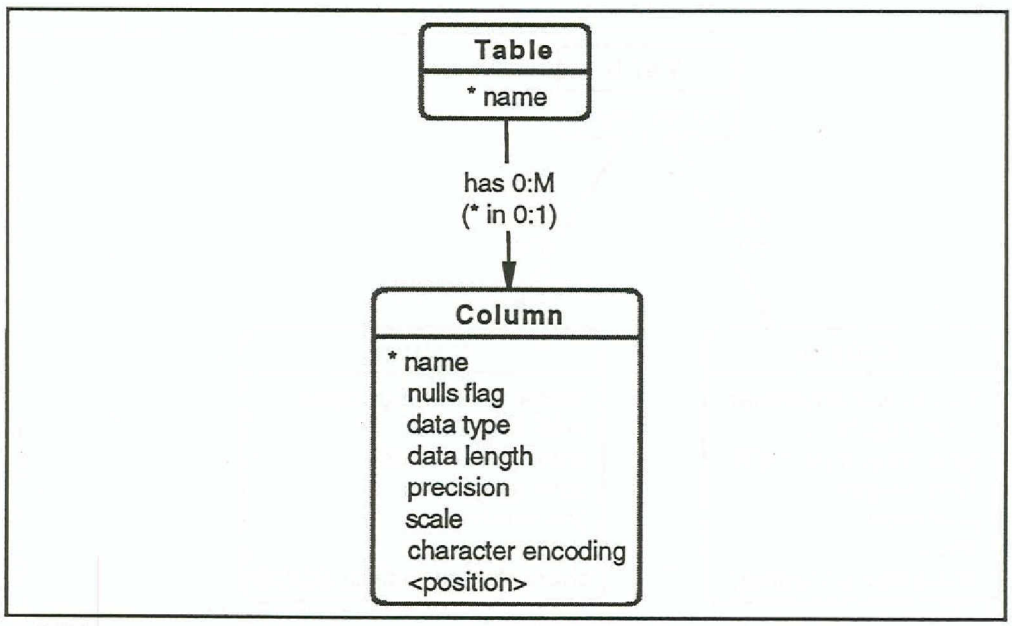
FIGUR 2

2. Grundläggande entity types

2.1 Table, Column

I idealfallet innehåller ett repository alla definitioner som behövs för att generera ett schema med hjälp av DDL-satser. Vissa Case-verktyg kanske också önskar modellera operationer. I så fall måste modellen klara (åtminstone en delmängd av) uttryckskraften i DML. DRC har långt ifrån denna ambitionsnivå, men relationsmodellens viktigaste konstruktioner kan definieras. Dit hör givetvis begreppen **Table** och **Column**.

En Table har ett namn och är uppbyggt av ett antal kolumner. Varje kolumn har också ett namn. Det ska vara unikt inom tabellen. Dessutom beskrivs Column bla med vilka värden som är tillåtna, vilket format värdet ska uttryckas i, osv. Figur 3 visar några attribute types.



FIGUR 3

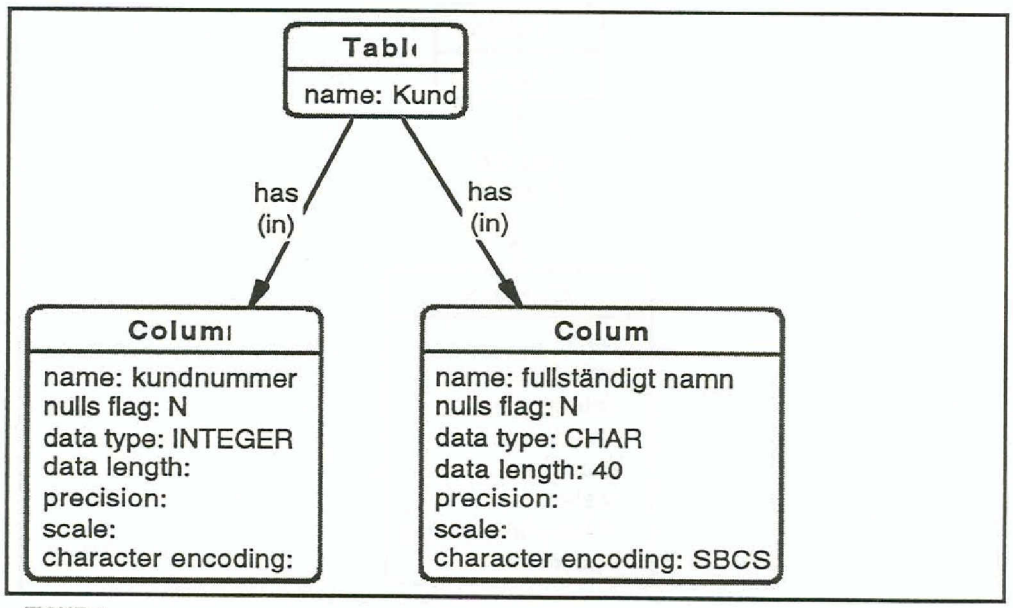
"Nulls flag" har värdet Y(es) om null-värden accepteras, annars N(o). "Data type" kan anta ett av ett tiotal olika värden. Acceptabla värden är bla INTEGER, CHAR, DATE, UNKNOWN. Beroende på aktuell "data type" anger "data length" längden på ett värde, antingen i antal bytes eller i antal dubbel-bytes. "Precision" uttrycker antal decimaler för ett kolumnvärde. Uppgiften är bara aktuell om "data type" har värde DECIMAL eller UNKNOWN. Behöver man vid decimaltal tala om hur många positioner till höger om decimalpunkten värdet hör hemma, anges detta antal under attribute

type "scale". "Character encoding" anger om värdet hanteras som BIT, SBCS (Single Byte Character Set) eller MIXED.

Varje Column tillhör normalt en viss Table. Som synes ställer modellen inte detta krav (0:1). Anledningen är att det i vissa modelleringsansatser är tillåtet att under arbetets gång skapa temporärt fristående Columns. När man är klar över vad kolumnen beskriver relateras den in under "sin" Table. Se vidare avsnitt 4. Detta är ett exempel på den flexibilitet som måste finnas i en repositorymodell för att den ska bli användbar för många syften och ansatser.

I sammanhanget kan nämnas att relationship type "has/in" mellan Table och Column har angivits som Ordered i DRC-modellen. Det betyder att varje Column alltid implicit ges en bestämd plats inom sin Table över relationship type "has/in". Ordningen bestäms i gränssnittet mellan RM och Case-verktyget, dvs genom den ordningsföljd Columns placeras in i aktuell template (se rapport TRIAD K4) för överföring till repositoryt. Lösningen är kanske praktisk och prestandabefrämjande, men knappast modellmässigt snygg. Vår grafiska notation har ingen motsvarighet till detta. Vi tillför istället Column en attribute type "<position>", inom hakar för att notera att den inte ingår i den formella modellen. Codds ursprungliga definition innehöll för övrigt inga krav på viss ordning mellan kolumnerna. (Dock gör SQL det.)

Table *Kund* från figur 2 beskrivs, givet hittillsvarande DRC-modell (figur 3), enligt figur 4.



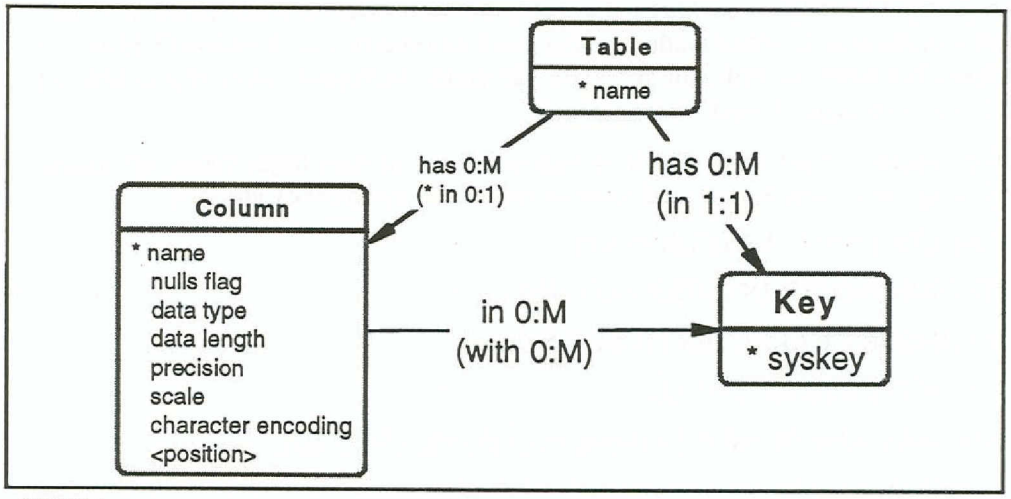
FIGUR 4

2.2 Key

En Table har normalt en primärnyckel (primary key) bestående av en eller flera Columns. En Table kan referera till en annan Table via en främmande nyckel (foreign key). Den främmande nyckelns uppbyggnad svarar mot refererad Tables primärnyckel. Av bla prestanda- och konsistensskäl skapas ibland index

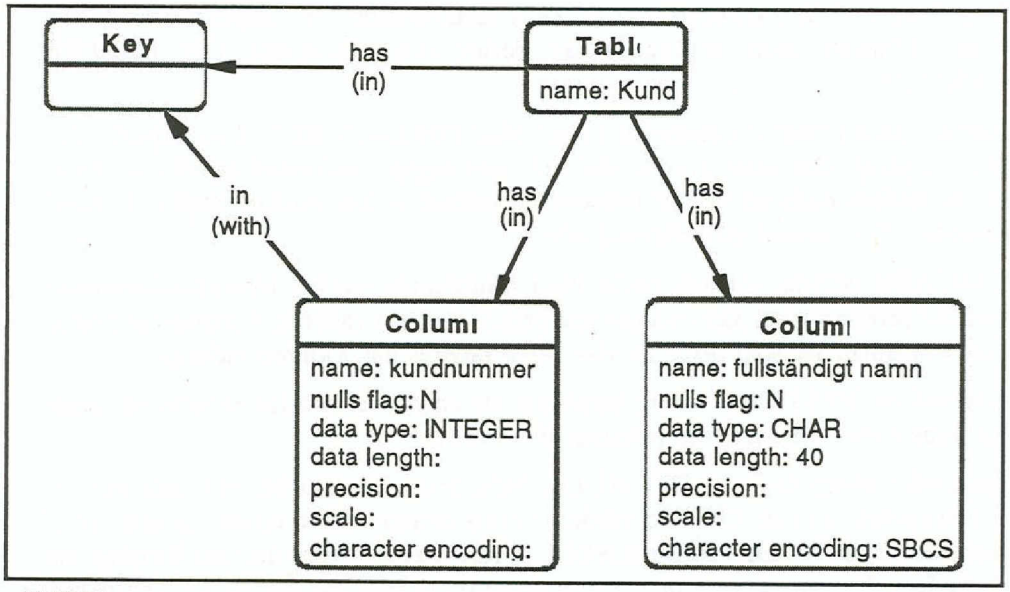
bestående av en eller flera Columns. Gemensamt för primärnyckel, främmande nyckel och index är att de är sammansatta av en eller flera Columns ur en viss Table. Denna gemensamma omständighet modelleras under entity type **Key**. En Key har inget namn utan endast en intern identifierare (syskey). Vad som behövs för att ge en unik referens till viss Key är därför oklart.

I praktiken identifieras de allra flesta entity types i IM med en syskey. Vi noterar ändå en kombination av attribute types och/eller relationship links som identifierare i våra modeller för att klargöra vad som i princip behövs för unik extern referens.



FIGUR 5

Table *Kund* kompletteras med beskrivning av primärnyckeln.



FIGUR 6

Att göra samma sak med Table *Arbetsinsats* blir mer problematiskt eftersom vi där, förutom primärnyckeln, har att beskriva en främmande nyckel. Över till nästa avsnitt.

2.3 Främmande nyckel och referential constraint

En främmande nyckel definieras inom ramen av sin Table som en Key på samma sätt som samma Tables primärnyckel. Därutöver måste man kunna ange vilken Table som refereras, dvs vilken primärnyckel den främmande nyckeln svarar emot. Uppgiften är nödvändig för att modellens semantik ska kunna tolkas. Referensen mellan primär- och främmande nyckel kompletteras normalt med uppgift om villkor för sambandets existens. Vad ska exv hända med främmande nycklar när dess motsvarande primärnyckel önskas raderad? Ska radering inte tillåtas så länge det existerar motsvarande främmande nycklar? Ska motsvarande främmande nycklar med automatik också raderas? Eller ...? Specifikationen av vad som ska gälla går under begreppet referential constraint (finns vedertagen svensk översättning?).

SQL-satsen för att skapa Table *Arbetsinsats* ser ut som följer (utan full namngivning, se avsnitt 4, och med reservation för ev. syntaktiska grodor):

```
CREATE TABLE ARBETSINSATS
(KUNDNUMMER INTEGER NOT NULL,
BILREGNR CHARACTER (7) NOT NULL,
UTFÖRD DATE NOT NULL,
PRIMARY KEY (KUNDNUMMER, BILREGNR, UTFÖRD),
FOREIGN KEY KUNDREF (KUNDNUMMER)
REFERENCES KUND ON DELETE CASCADE)
```

De två sista raderna länkar en främmande nyckel i en Table (Column *KUNDNUMMER* i *Arbetsinsats)* till dess motsvarande primärnyckel i en annan Table (*Kund*) med angivande av referential constraint ON DELETE CASCADE.

Namnet på referential constraint anges efter FOREIGN KEY. I exemplet heter den *Kundref*. Om man, som i vårt fall, inte uppger refererad tabells kolumnnamn, underförstås att de har överensstämmande namn. Annars anges de inom parentes efter namnet på refererad tabell.

CASCADE betyder att alla rader med främmande nyckel överensstämmande med en primärnyckel ska raderas när primärnyckeln raderas. I exemplet skulle alla *Arbetsinsats*er som berör en viss *Kund* raderas när *Kund*en raderas.

Alternativet RESTRICT skulle inneburit att en *Kund* inte får raderas så länge det finns *Arbetsinsats*er som refererar till *Kund*en.

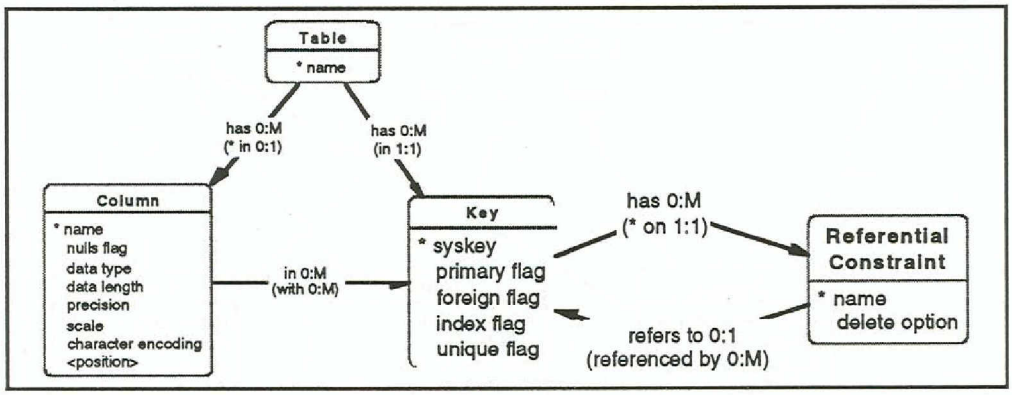
Det tredje alternativet är SET NULL. Med detta satt skulle, vid radering av primärnyckel, alla de Column-värden i motsvarande främmande nycklar, som tillåts ha värdet NULL, sättas till NULL. I vårt exempel vore det inte lämpligt att använda, eftersom *kundnummer* i *Arbetsinsats* inte får bli NULL. Effekten skulle bli att *Kund*en raderas, men inte de *Arbetsinsats*er som betalats.

Om vi jämför med begreppsmodellen (figur 1) tar den inte ställning till vad som bör göras vid en operation. Där konstateras bara vad som ska gälla för att

databasen ska vara konsistent, dvs att om en *Arbetsinsats* finns, måste också dess betalande *Kund* finnas (1:1-förhållandet).

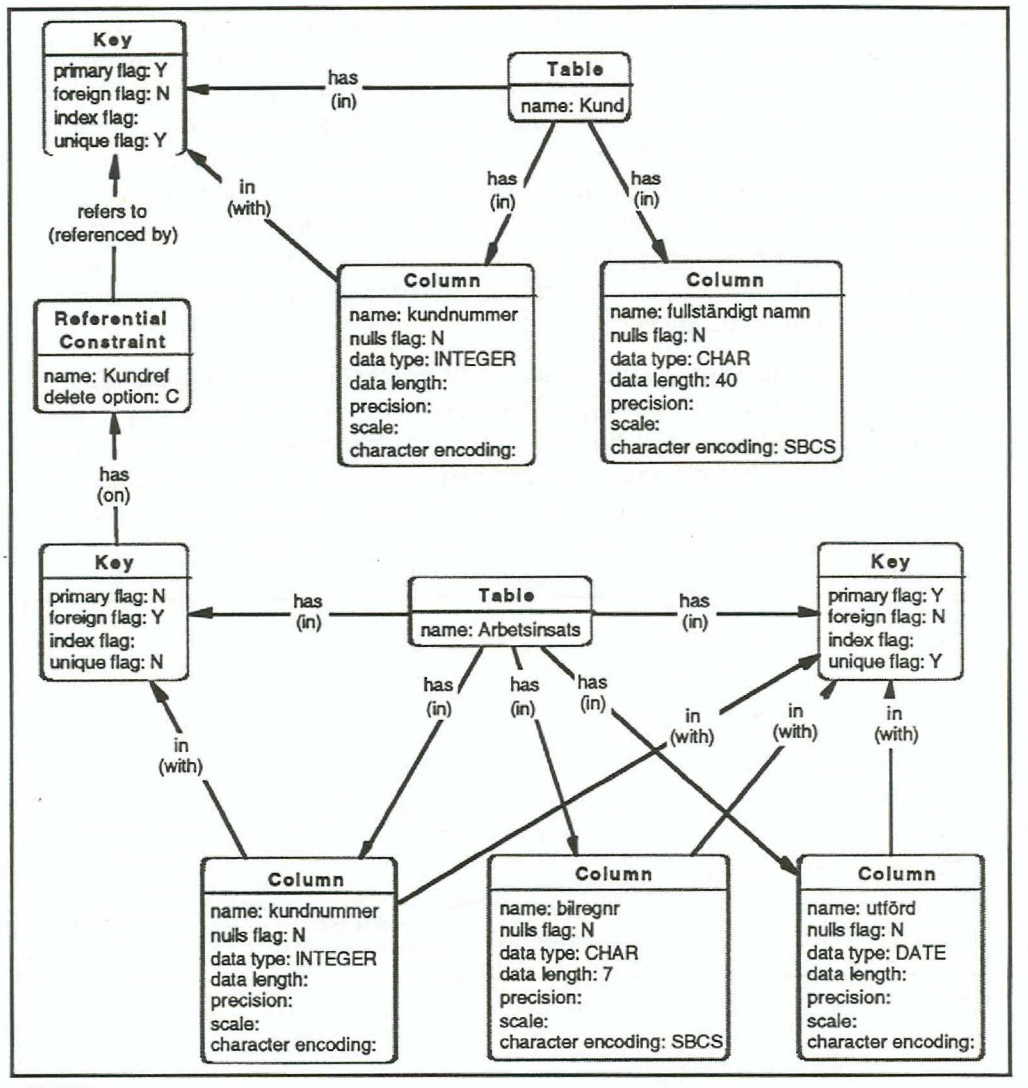
I DRC-modellen uttrycks villkoren genom entity type **Referential Constraint.** Den länkar samman en främmande nyckel ("has/on") och dess motsvarande primärnyckel ("refers to/referenced by"). Attribute type "delete option" anger just vad som ska gälla vid en radering. Dess alternativa värden är (R=RESTRICT, C=CASCADE och N=SET NULL). Identifieringen av Referential Constraint är något missvisande. Egentligen är den "name" tillsammans med "name" på den Table där den främmande nyckeln ingår. Eftersom denna relatering endast finns indirekt över Key har "on" asteriskmärkts istället. Ät det för övrigt rimligt för en Key att ha flera Referential Constraints så som DRC anger?

Vi behöver from nu kunna skilja på Key som primär- och som främmande nyckel. Det sker genom attribut types "primary flag" och "foreign flag". Key har också kompletterats med ett par andra attribute types "index flag" och "unique flag". Den senare anger om ett visst nyckelvärde bara återfinns i en rad eller ej. Om "primary flag" satts till Y måste "unique flag" också ha värdet Y. "Index flag" återkommer vi till i avsnitt 2.5. Se figur 7.



FIGUR 7

Arbetsinsats och *Kund*, modellerade enligt ovan, visas i figur 8.



FIGUR 8

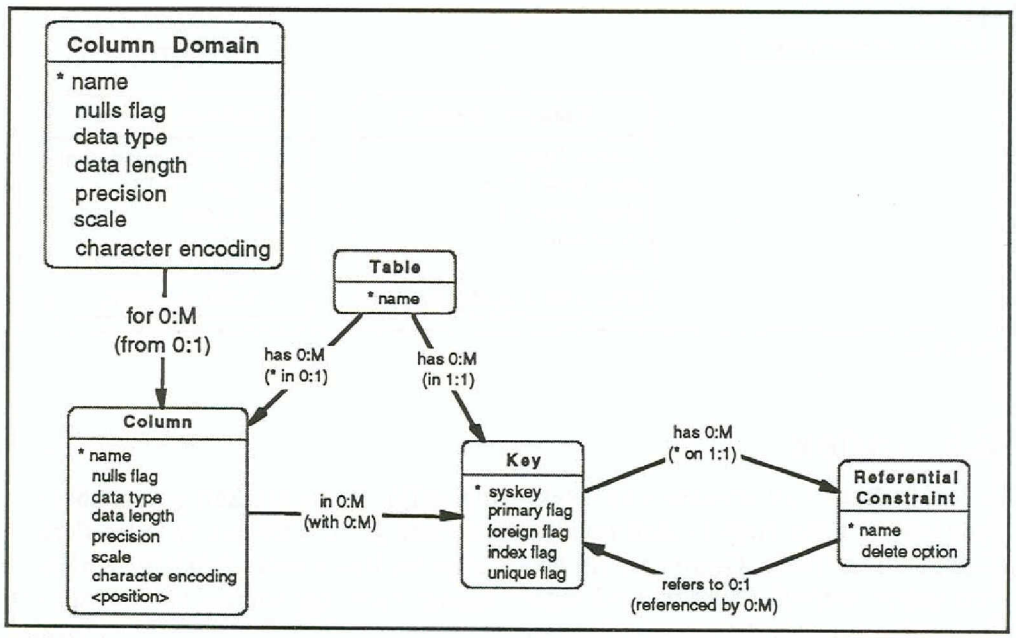
2.4 Column Domain

I begreppsmodellen skiljer man på Attribute Type (vad som formuleras), Info Type (vilka abstrakta värden som får användas för att uttrycka en Attribute Type) och Symbol Set (en exakt syntax för att formulera en Info Type). Codds ursprungliga relationsmodell gör åtskillnad mellan Attribute Type och Info Type (domain). Motsvarande åtskillnad gör inte alltid ett RDBMS.

För att erbjuda de modelleringsansatser, som önskar göra en distinktion, möjlighet att formulera detta, har entity type **Column Domain** inkluderats i modellen. Column Domain beskrivs med de sista fem attribute types som exemplifierats under Column. En Column tillhör högst en Table. En Column Domain, däremot, är en fristående företeelse - en beskrivning som kan gälla för flera

Columns. Genom referens till en Column Domain slipper en Column själv ange motsvarande uppgifter. Huvudprincipen är att om "data type" för en Column har ett null värde och det finns ett samband med en Column Domain, gäller denna Column Domains uppgifter för aktuell Column. I övriga fall hämtas uppgifterna från Columns egen beskrivning.

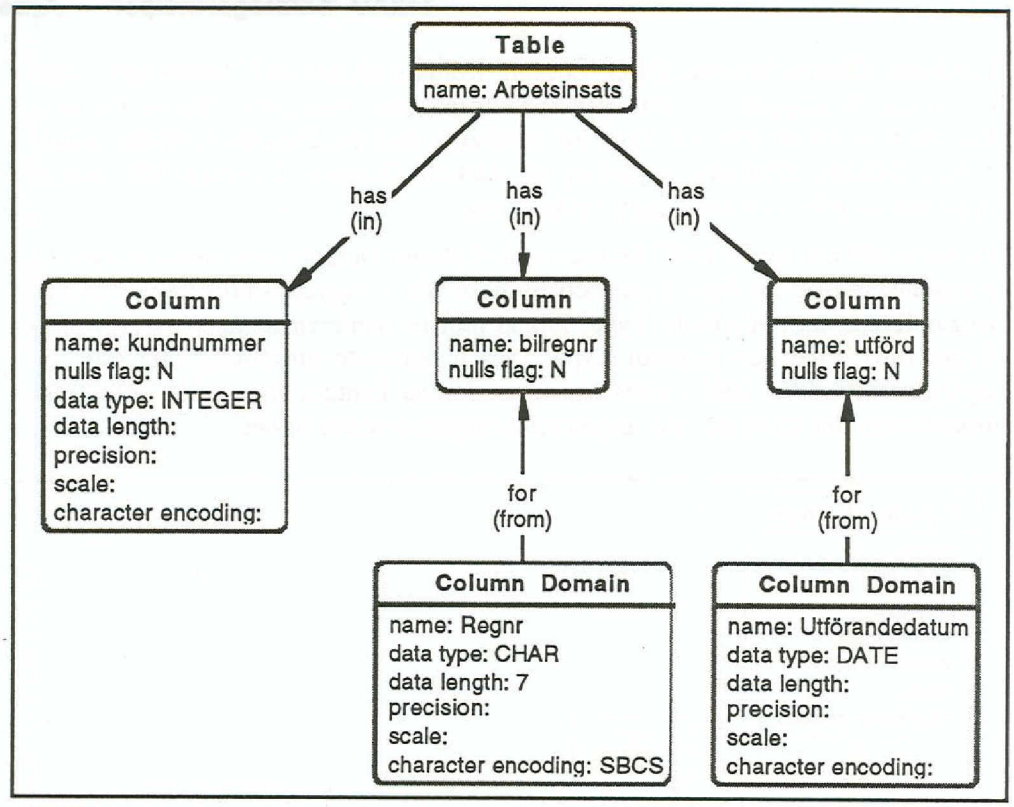
På detta sätt har man i DRC åstadkommit Column Domain som motsvarande en sammanslagning av Info Type och Symbol Set i begreppsmodellen. Column svarar ibland mot Attribute Type, ibland mot en sammanslagning av Attribute Type, Info Type och Symbol Type. Den förvirrande omständigheten får antagligen tillskrivas DRCs målsättning att kunna hantera flera olika relationsmodelldialekter även på den implementeringsoberoende nivån.



FIGUR 9

I sammanhanget kan det vara lämpligt att notera den totala avsaknaden av koppling mellan DRC och begreppsmodellen, en brist som kan förmodas bli åtgärdad i någon kommande release.

Antag att vi gör bedömningen att bilregistreringsnummer och utförandedatum kommer att förekomma med samma definition på ett antal ställen i den fullständiga bilservicemodellen. De "upphöjs" till Column Domains under namnen *Regnr* och *Utförandedatum*. Exemplet visar bara en användning var av dem. I det generella fallet kan man anta att de var och en kommer att peka på ("for/from") ett antal Columns.



FIGUR 10

2.5 Index

Index är främst av intresse i en exekveringsmiljö. Ett index kan ge snabbare referens till rader (tupler) som refereras via vissa givna värden i en eller flera Columns i en Table. Genom att vid behov ange ett Index som UNIQUE, erhålls automatisk kontroll på att inte två rader i aktuell Table har samma värdekombination för indexets definierade Columns.

Index definieras genom följande SQL-syntax (återigen med reservation för syntaxfel):

CREATE [UNIQUE] INDEX <indexnamn>

ON <tabellnamn> (<kolumnnamn> [ASC | DESC], ...)

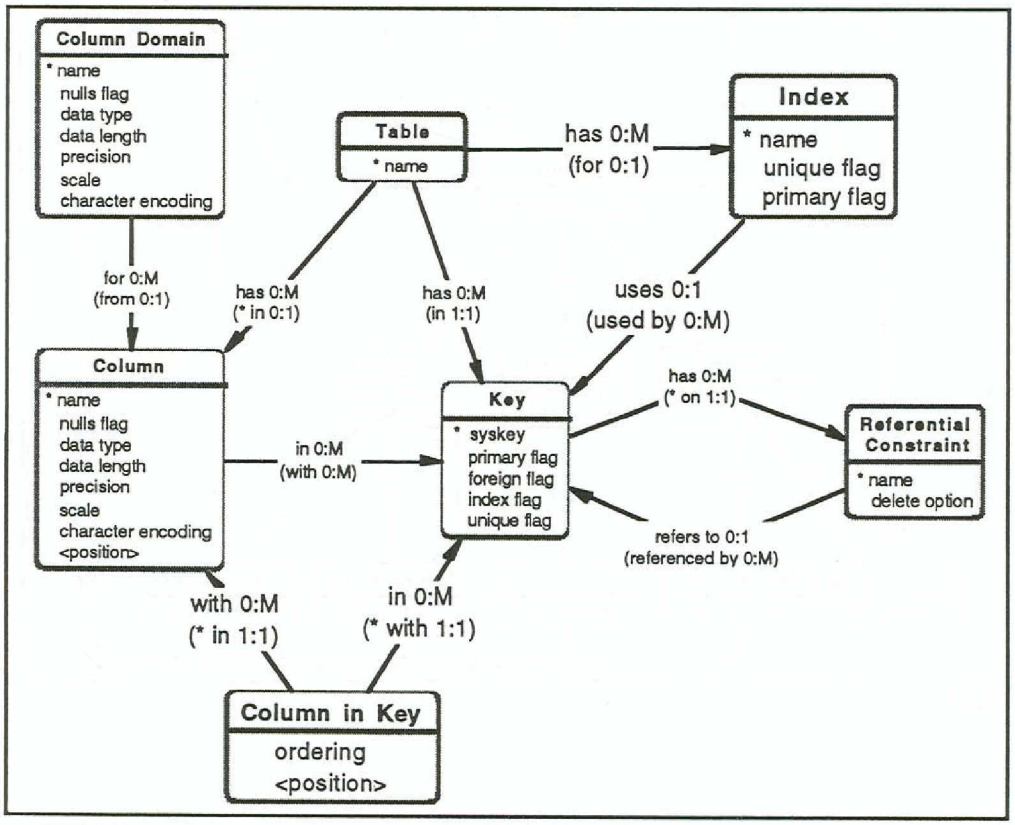
Som synes ingår en eller flera Columns i indexet, något som också gäller för Key. Av den anledningen väljer man i DRC-modellen att etablera en entity type **Index,** men låter detta peka på Key för att slippa upprepa samma sak. Modellmässigt är det praktiskt, men ett Index är inte alltid en Key. Vi kan ju välja att skapa ett Index för exv Column *fullständigt namn*. Dessutom måste man följdaktligen i Key-definitionen tillföra uppgiften om stigande (ASC) eller fallande (DESC) sorteringsordning, något som endast är av intresse för den händelse ett Key fungerar som ett Index.

Sorteringsordning anges för varje Column ingående i ett Key. För detta behövs i en binär modell en ny entity type **Column in Key**. IBMs metamodell (se rapport TRIAD K5), dvs den modell som används för att bla uttrycka DRC-mod-

ellen, tillåter inte attribute types för relationship types. Däremot är flerställiga relationship types tillåtna. DRC-modellen uttrycker sorteringsordning i en treställig relationship type mellan Key, Column och något man kallar Col in Key A (dit attribute type "ordering" placeras).

I BM modellerar vi samma sak enligt figur 11. Som synes pekar Table ut vilka Index som genererats för den genom relationship type "has/for". Index å sin sida pekar ut vilken Key som beskriver dess uppbyggnad. Column in Key har smugit sig in mellan Column och Key för att beskriva sorteringsordning för en Column i en Key. Obs, att Column C1 kan förekomma i Key K1 som ASC, i K2 som DESC, i K3 som .....

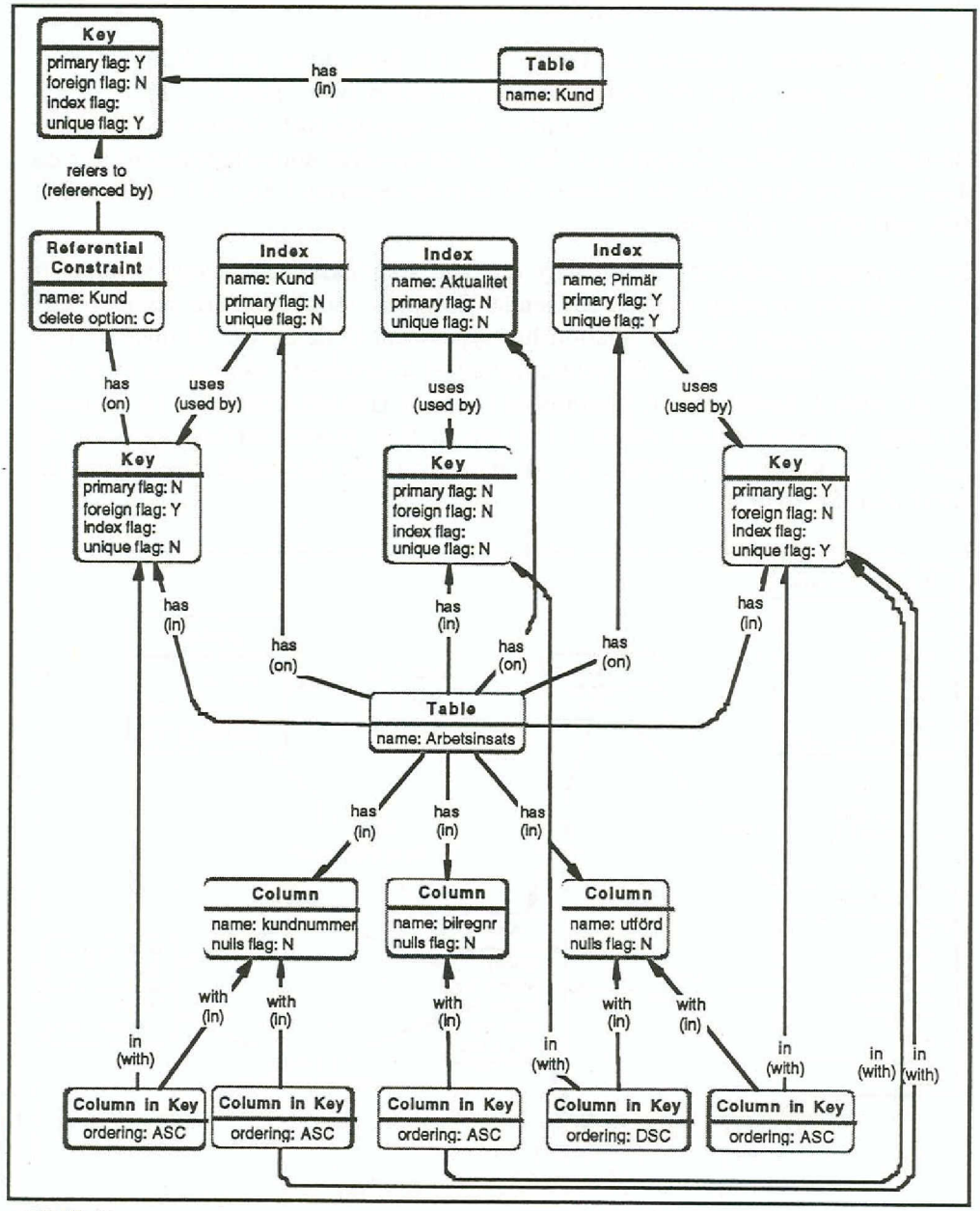
En fullständig definition av en Key (använd som Index) ska även ha uppgift om inbördes ordning mellan ingående Columns. Som tidigare nämnts kan IBMs metamodell uttrycka relationship types som Ordered när så önskas. Vill vi exv att Columns i index för primärnyckeln i *Arbetsinsats* ska ordnas efter *bilregnr, utförd, kundnummer,* placeras de i avsedd gränssnittstemplate in i samma ordning. Vi noterar detta behov i Column in Key genom attribute type "position". Den är satt inom hakar, för att notera att den inte ingår i den formella modellen. Se figur 11.



FIGUR 11

Behovet av "unique flag" och "primary flag" under Index är oklart. Uppgifterna finns ju under refererad Key. Dock står det i manualen att det kan finnas anledning att definiera en Key som unikt och refererande Index som ickeunikt. !? I alla händelser sätta en Keys attribute type "index flag" till Y(es) när det är refererat från ett Index.

Antag att vi för *Arbetsinsats* vill skapa ett index för primärnyckeln (underförstådd ordning mellan Columns enligt ovan), ett annat för *kundnummer* och ett tredje för *utförd*.



FIGUR 12

3. Hantering av views

En view är en sammanställning av data definierade i form av en SELECT-sats. Dess uppbyggnad baseras därmed på referenser till en eller flera andra relationer och/eller views. En formell modell av den fulla syntaxen för en SELECT-sats skulle bli mycket komplex. En minsta ambitionsnivå vore istället att, förutom ingående Columns, bara lagra SELECT-satsen som en textsträng. Ett något mer ambitiöst alternativ vore att komplettera med referens till de relationer och/eller views, som en given view baseras på. En ytterligare mer utvecklad ansats skulle också innehålla referens till refererade kolumner.

Föreliggande release av DRC speglar mellanalternativet. I långa loppet måste ett repository dock kunna hantera en full definition, om dess innehåll ska kunna vara bas för en automatisk generering av ett schema, speciellt om genereringen ska kunna genomföras mot olika dialekter av SQL.

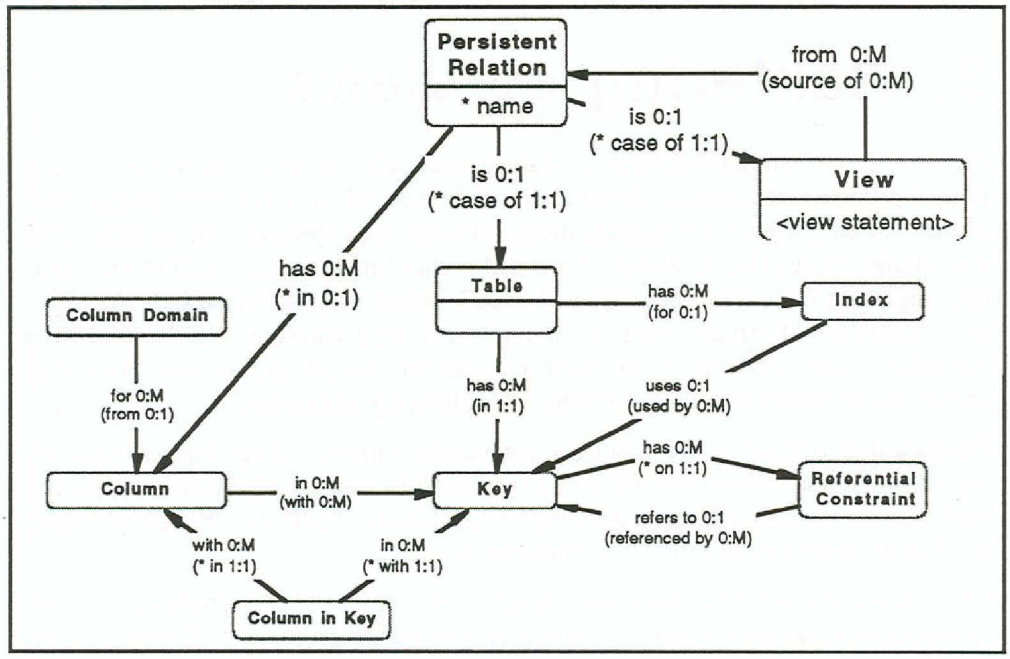
DRC-modellen kompletteras med entity type **View.** Dess uppbyggnad överensstämmer väl med Table i så motto att den består av ett antal Columns. Den ska därutöver, enligt förutsättningen ovan, referera till sitt "ursprung" i form av Tables och eller Views. "Och/eller"-situationen löses genom att skapa en generalisering av de gemensamma egenskaperna från Table och View. Table och View behåller sina specifika egenskaper som subtyper. Eftersom beskrivningsspråket för AD Cycle Information Model (IM) inte kan formulera generaliseringar/specialiseringar upprättas motsvarande kopplingar i form av vanliga relationship types. Generaliseringen har fått namnet **Persistent Relation**. Den tar över rollen från Table att referera till Columns och blir samtidigt den gemensamma punkten en View kan peka på för att redovisa sitt ursprung.

Som specifika egenskaper har View SELECT-satsen. Däremot är inte Index aktuellt för en View eftersom en View materialiseras först när den refereras. Inte heller har en View (alltid) en väldefinierbar Key, än mindre Referential Constraint på dessa.

Text, hanteras generellt i IM genom referens till Entity Types under Global Submodel. För att inte onödigtvis tynga modellen med dessa, lägger vi in SELECT-satsens text i form av en attribute type "view statement" inom hakar för att indikera ett avsteg från DRC-definitionen.

Till sist, den som tittar i manualen upptäcker att Persistent Relation är relaterat till en entity type kallad **Pers Rel Pred Stat** i ett 1:1-förhållande. Samma sak gäller också Column med **Col Pred Stat** och Key med **Key Pred Stat**. Alla tre innehåller ett antal frivilliga attribute types för statistikändamål. Statistiken kan exv vara till nytta vid generering av ett operationellt schema. Antagligen har man valt att lägga dessa uppgifter separerat från övriga attribute types i respektive "värd"-entity type pga deras speciella karaktär. Vi har valt att inte inkludera dem i "vår" DRC-modell, bara notera deras existens.

Den ändrade och kompletterade DRC-modellen visas i figur 13. "Gamla" attribute types har vi valt att undertrycka för att lätta upp modellen något.



FIGUR 13

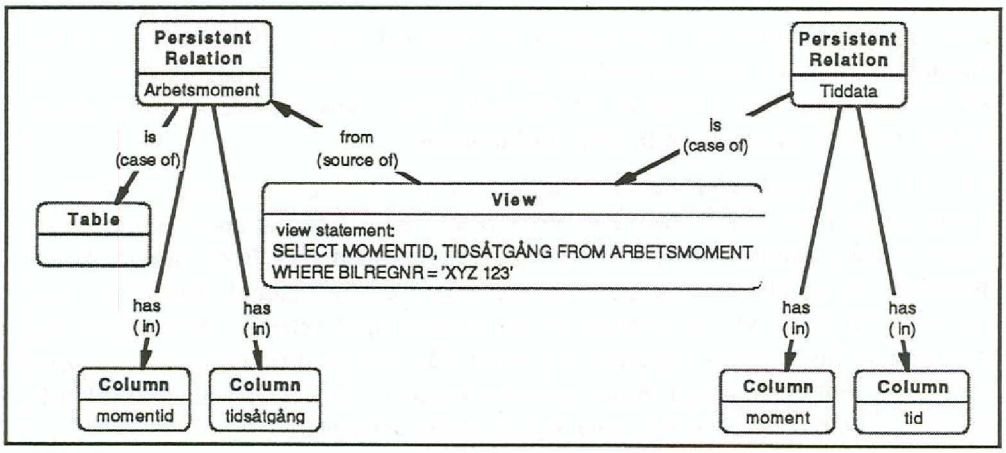
Över till ett exempel. Antag att vi vill ta fram statistik över hur lång tid olika moment har tagit att utföra genom åren för bilen 'XYZ 123'. Vi skapar en View av Table *Arbetsmoments* Columns *momentid* och *tidsåtgång* med följande SQL-sats (reservation för syntaxfel):

CREATE VIEW TIDDATA (MOMENT, TID) AS

SELECT MOMENTID, TIDSÅTGÅNG FROM ARBETSMOMENT

WHERE BILREGNR = 'XYZ 123'

Resulterande förekomstmodell framgår av figur 14. Endast de entity types som är involverade i View-definitionen visas och för dem endast deras namn.



FIGUR 14

4. Separat namnmodell

Namn på en Table, View och Index kan vara kvalificerat. Syntaktiskt uttrycks en kvalificering i SQL med hjälp av punktnotation. En kvalificering svarar mot en gruppering av ingående element för något behov. Exv kan ägaren till en uppsättning Tables önska kvalificera var och en med sitt namn (TRIAD.Table1, TRIAD.Table2, ...). En annan kvalifikation kan gälla ett behörighetsnamn. I båda fallen underförstås element ur samma databassystem. Eftersom man, speciellt i distribuerade sammanhang, behöver kunna referera till element i andra databassystem kan en ytterligare kvalificering vara nödvändig för entydig referens. Denna kvalificering står för beteckningen på databassystemet (TVT.TRIAD.Table1, Posten.TRIAD.Table2, ...).

I DRC-modellen har man valt att placera samtliga dessa namn som separata entity types. Databassystemet modelleras under entity type **Rdbnam** (var). Nästa lägre kvalificering hanteras som en godtycklig andranivå-kvalificering under namnet **Collection** (vem). Det direkta, lokala namnet på Persistent Relation och Index har också brutits ut och placerats under entity type **Relation Name** (vad) respektive **Index Name** (vad). I denna rapport tidigare visade versioner av DRC-modellen har haft dessa namn som attribute types under respektive entity type av praktiska skäl, för att inte introducera alltför mycket detaljer på en gång. Figur 15 visar den formellt korrekta hanteringen av namn. Där framgår också att temporärt fria Columns kan föras in under en Collection.

Notera att en Table enligt modellerade namnkonventioner inte tillåts vara uppdelad under flera databassystem med samma namn (0:1-avbildningar).

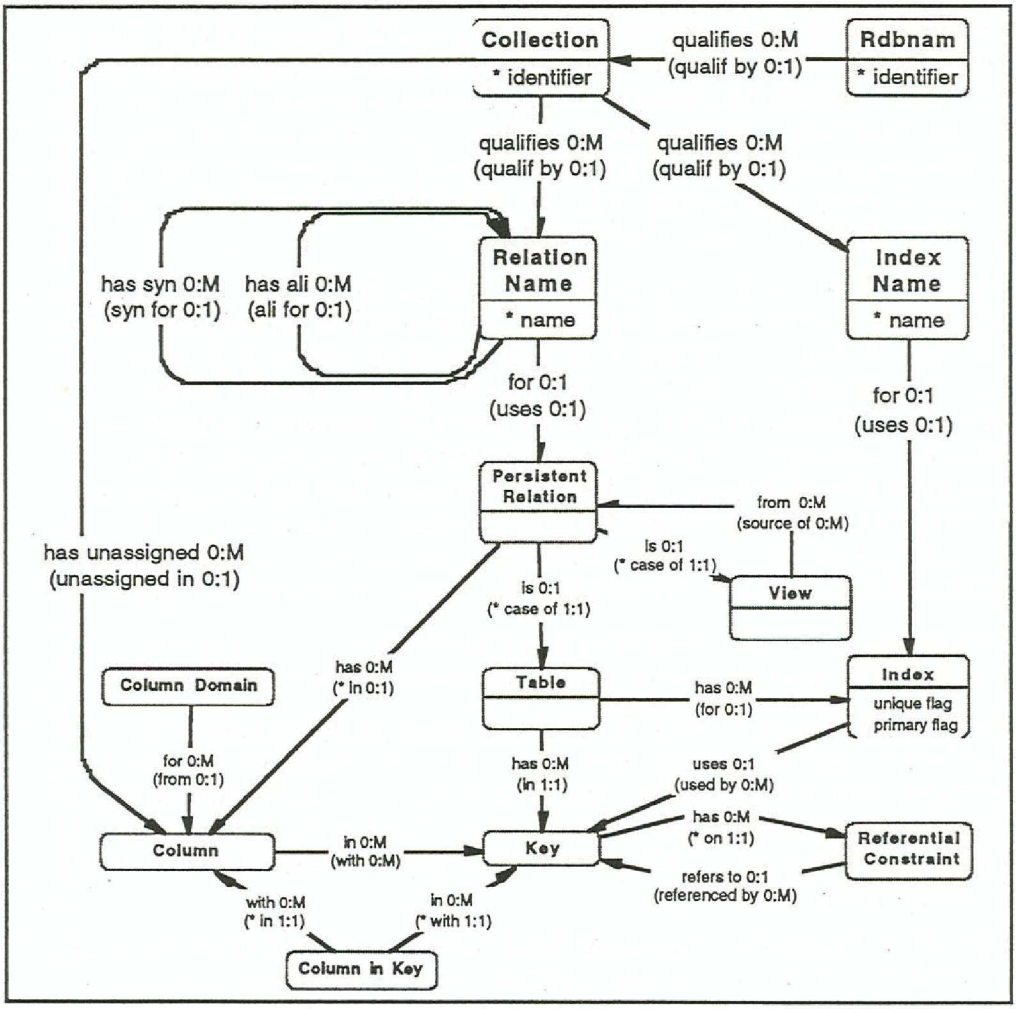
Eftersom både Persistent Relation och Index under en modellering temporärt kan definieras utan att ha namn, undviker vi att asteriskmärka deras respektive relationship link "uses". Samma sak gäller relationship links "qualif by" i namn-hierarkin. Återigen ett exempel på den flexibilitet en repositorymodell måste ha mot många behov och metodansatser. Priset är lägre stringens, kontroll och precision.

Varför det lokala namnet brutits ut till Relation Name respektive Index Name är oklart. Till viss del kan det kanske förklaras i behovet av att kunna ange synonym- och aliasnamn, se nedan. Varför databassystemet går under beteckningen Rdbnam istället för en mer namnfrikopplad beteckning, exv Dbsystem, är också oklart. Till sist är distinktionen mellan "name" och "identifier" oklar.

Vissa SQL-dialekter tillåter användning av synonymer eller alias vid referens till en Persistent Relation. Synonymer och aliases definieras på namnnivån. Ett Relation Name kan ha en synonym. Synonymen kan också referera till en Persistent Relation inom annan Collection.

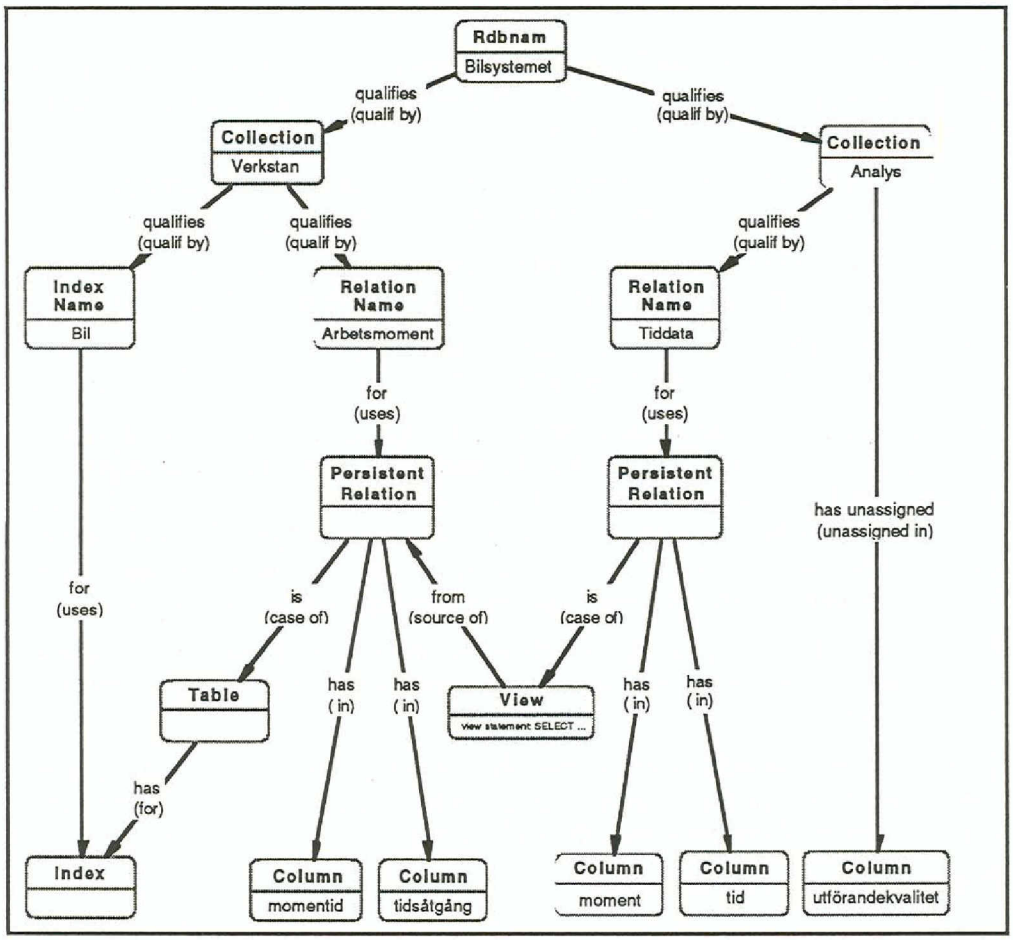
Skillnaden mellan synonym och alias tycks vara den att ett alias-namn tillåts ersätta båda kvalificeringsnivåerna. Dvs, ett alias kan referera till en Persistent Relation i ett annat databassystem.

Synonymer och alias modelleras som självreferenser till Relation Name, en lösning som känns otydlig. Vissa Relation Names är genom detta synonymer, vissa andra aliaser medan andra fungerar som grundnamn. Endast de senare tillåts relateras till en Persistent Relation (1:1-förhållande). En synonym-Relation Name får inte vara synonym till ett synonym- eller alias-Relation Name, ett alias-Relation Name får inte vara alias till ett annat alias-Relation Name men tydligen till ett synonym-Relation Name. Osv. Utan exempel eller vidare försök till förklaring har dessa relationship types inkluderats i modellen. Se figur 15.



FIGUR 15

Figuren 14 omritad efter namnsepareringen och med ett tillskott på ett tänkt Index *Bil* över *bilregnr* inom Table *Arbetsmoment* visas i figur 16. Vi har, för fullständighets skull, också tagit med Column *utförandekvalitet* , som en än så länge fristående Column.



FIGUR 16

5. Sammanfattning

DRC-modellen kommer säkerligen att kompletteras för att göra det möjligt att mer fullständigt generera både DDL- och DML-satser. Hur långt man i praktiken behöver/orkar gå är svårt att bedöma.

Bör det vara en målsättning att med hjälp av DRC och kompletterande produktspecifik modell kunna erbjuda ett 100% underlag för SQL-generering?

Om schemat i framtiden ska fylla rollen som ett aktivt repository, gäller automatiskt 100%-principen. Om inte, kommer behov och efterfrågan att styra.

Hur komplicerade modeller orkar modellörer hantera (även om de arbetar via ett Case-gränssnitt), och göra det på ett konsistent sätt?

SQL vidareutvecklas hela tiden. Detta ställer successivt nya krav på DRC-modellens uttryckskraft. SQL3 blir förmodligen en standard om några år. Förutom en komplettering med objektorienterade uttrycksmöjligheter är den samtidigt i praktiken ett komplett, generellt programmeringsspråk. Vem klarar av en sådan anpassning av DRC-modellen - för att inte tala om Case-leverantörer och -användare? Var ligger den lämpliga kompromissnivån?

Ett mer närliggande och starkt efterfrågat önskemål är möjligheten att relatera komponenter i DRC-modellen med komponenter i begreppsmodellen. Det lär vara på gång.

Kanske kommer modelleringen i framtiden endast att göras i begreppsmodellens termer och med direkt generering därifrån av DDL och DML? Vem behöver då den produktoberoende DRC-modellen?

Framtiden är oviss men spännande!

TRIAD utvecklar IA

Televerket har just tagit första steget in i sin nya IA-organisation och Posten håller på att bygga upp sin nya DA-organisation. Båda organisationerna har sett nyttan att inför 90-talet gå vidare tillsammans i TRIAD-projektet som drivs tillsammans med SISU. Statskontoret deltar också i projektet för att på sikt kunna föra ut nya synsätt och hjälpmedel inom den civila statliga sektorn.

Ericsson Data Services deltar med tyngdpunkten i den del som handlar om att utveckla kompetenta modelleringsledare, delprojektet "Avancerad utbildning för modelleringsledare".

Modelleringsmetoder är centrala i bedrivandet av verksamheten inom informationsadministrationen. Därför arbetar ett delprojekt med utvecklandet av "nästa generation modelleringsmetod" som skall sättas i händerna på informationsadministratören. Siktet är att fördjupa och bredda dagens modelleringsmetoder och där hämta in kunskap från pågående forskning och utveckling internationellt. (faktaruta om IAS91).

Som stöd för informationsadministrationen behövs verktyg. Inom TRIAD arbetar man där inom två områden, kataloger och verktyg.

Delprojektet kataloger arbetar dels med att utforma den informationsmodell som måste kunna täckas av en katalog, dels med att granska och följa utvecklingen av produkter inom området t ex IBM:s "Repository" och Digitals "CDD". Dessutom följer man standardiseringen internationellt kring IRDS. För parterna i projektet liksom för andra organisationer är detta ett tungt område både vad gäller kommande investeringar ekonomiskt och vad gäller kompetenta resurser för en kommande övergång till "repository-världen". - Det inledande skedet syftar till att bygga upp en kunskapsplattform, som sedan kommer att kunna utnyttjas för kravställande och planering och genomförande av övergång från dagens kataloghantering till morgondagens.

Den andra verktygshanterande delen inom TRIAD-projektet, delprojektet "verktyg för informationsadministration", syftar till att ta fram verktyg för uttag och dokumentering av modeller. Betoningen ligger på människa datorgränssnitt och i första skedet görs utveckling av HYBRIS-gränssnittet med prototyper för Posten och för Televerket.

För att hålla ett helhetsperspektiv på projektets delar och för att ha inpassningen av funktionen Informationsadministration i organisationens övriga verksamhet arbetar delprojektet "Krav på IA". I delprojektet arbetar man dels med att kartlägga dagens krav på dataadministration och projicera till morgondagens krav på IA. Dessutom skall man skapa en bild av IA-verksamhetens innehåll och organisation. Från detta i sin tur ställer man krav på övriga delprojekt. Vilka krav skall ställas på kompetens, metoder, hjälpmedel typ kataloger och gränssnitt?

TRIAD projektet är stort

Budgeten för TRIAD-projektet löper på 10 MSEK per år under en treårsperiod som startar vid kalenderåret 1991 års början och som alltså beräknas avslutad vid utgången av 1993.

TRIAD-projektet är ett tillämpningsprojekt

Det innebär att parterna, Televerket, Posten, Statskontoret, EDS och SISU går in med såväl persontidssatsningar som ekonomiska och att STU, Styrelsen för Teknisk Utveckling, bidrar med ett ekonomiskt tillskott som svarar mot ungefär 40 % av den insatta persontiden.

Öppet för fler deltagare

Parterna i TRIAD-projektet vill gärna öka tempot och bredda perspektivet och vill därför gärna ha fler parter in i projektet. Dessa parter får då enligt SISU:s tårtprincip "betala för en tårtbit, men ät hela tårtan", tillgång till projektets resultat med en insats som ger stor "price performance".

Nya deltagare kan gå in i hela projektet eller i det eller de delprojekt som verkar intressantast. En förutsättning är att man framförallt är beredd att satsa kompetent personal. För de flesta intressenter bord detta vara ett utmärkt sätt att driva personalutveckling för personer t ex inom DA-området, samtidigt som man bygger upp beredskapen inför 90-talets IA-verksamhet.

Kompetensutveckling viktigt resultat

En viktig effekt för parterna av deras medverkan i TRIAD är kompetensutveckling. Man satsar på att ta in personer som så småningom eller redan idag arbetar med DA och IA för att ge dem en djup och "frontlinje"-mässig kompetens. Detta skall utnyttjas när man successivt för in resultaten i den egna organisationen. Projektdeltagarna har alltså en viktig roll som kunskapsförmedlare i den egna organisationen. Dessutom ger projektarbetet deltagarna tillfälle till en egen utveckling inom det professionella området som är unik.

Informationsspridning

Det sjätte delprojektet "Informationsspridning" har till uppgift att sörja för att i första hand parterna men också SISU:s övriga intressenter successivt kan följa och tillgodogöra sig resultat från TRIADprojektet. Seminarier, rapporter och referensgruppsverksamhet är led i den verksamheten.