Modul DB1-2
Datamodellering

Antal föreläsningar: **2**
Antal laborationer: **1**
Förkunskapskrav: **Grundläggande kännedom om databaser (Modul DB1-1)**

Kurslitteratur: **”Praktisk datamodellering”**
ISBN: 91-44-38001-1

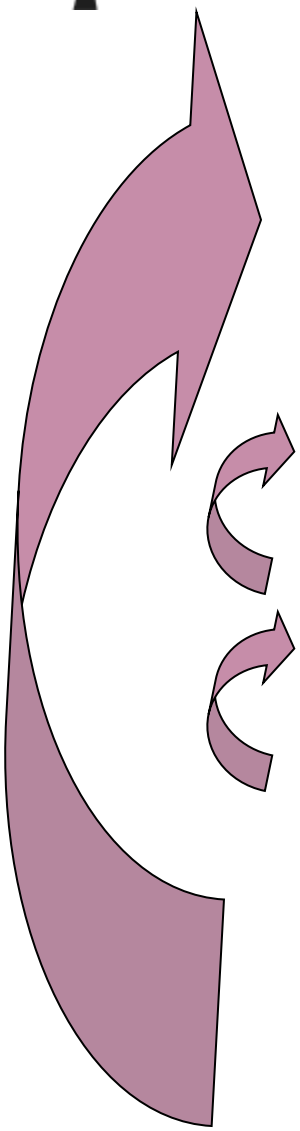
Referenslitteratur:

Innehållsförteckning

Innehållsförteckning:

- Utveckling av Logiska datamodellen
- Objektifiering
- Egenrelationer
- Funktionellt beroende
- Normalisering
- Ej önskvärda bieffekter

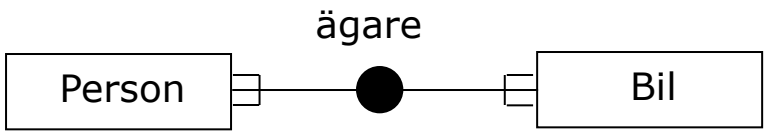
Utvecklingsprocessen



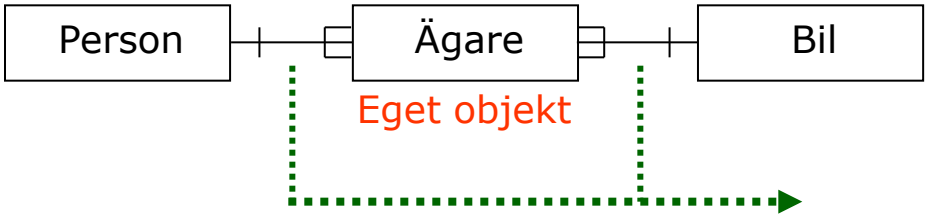
Steg 1	Verksamheten, nulägesanalys, livscykelmodellen, datamodellering, kravspecifikation, projektbeskrivning
Steg 2	Konceptuell modell, objekt, relationer, tabeller. Olika objektstyper och relationstyper. Attribut egenskaper, identifierare, nycklar
Steg 3	Logisk modell, utför normalisering (fyra normalformerna). Objektifiera dina relationsobjekt. Generalisera.
Steg 4	Fysisk modell, generalisering, denormalisering, optimering radvis/kolumnvis delning/sammanslagning, Index, tabellprecisering, dokumentera avvikelser från logisk modell. Anpassa för den valda databasen
Steg 5	Volymberäkningar, belastningsanalys, borttagsanalys, skärmbilder, svarstider
Steg 6	Databaskonstruktion, SQL-anpassningar, skärmbilder, rapporter, tester
Steg 7	Dokumentation och implementation (installation)

Objektifiering av n:m relation

Objektifiering av relationsobjektet



Observa att relationer vänds vid objektifiering



Ägare

PID	BilID
100	10
200	20
300	30

Före objektifiering

Efter objektifiering

Ägare

EgarID	PID (Fk)	BilID (Fk)
1	100	10
2	200	20
3	300	30

Objektifiering innebär att man ritar ett objekt (=en rektangel).

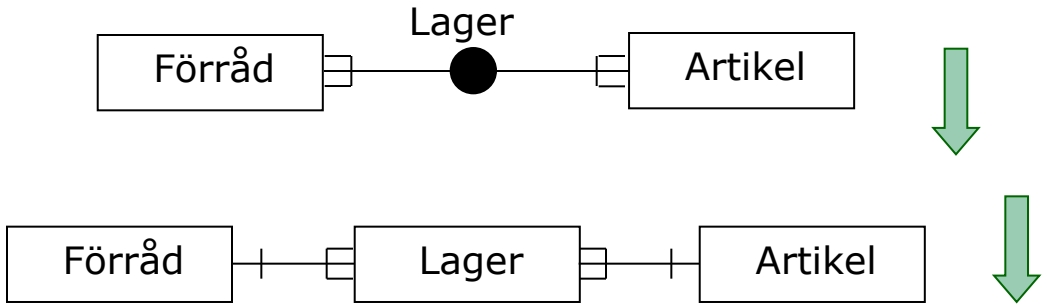
Dessutom tilldelar man det nya objektet en egen Pk.

Ägare

EgarID	PID	BilID	...
1	100	10	
2	200	20	
3	300	30	

Objektifiering av n:m relation

Objektifiering av relationsobjektet – flera varianter



Förråd

FörrådID	Namn	Adress	...
1	Stora	Sgatan	
2	Lilla	Lgatan	
3	Centrum	Cgatan	

Lager

FörrådID	ArtID	Antal
1	001	14
2	003	2
3	001	8

Artikel

ArtID	Namn	Färg	...
001	Bord	Furu	
002	Stol	Blå	
003	Fåtölj	Grön	

Det finns flera olika varianter av utförande av lagertabellen dvs relationsobjektet.

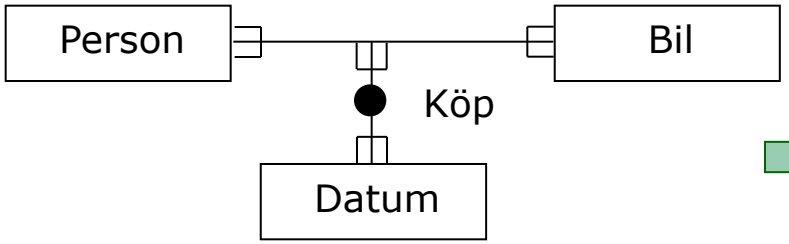
LagerID	FörrådID	ArtID	Antal
1	1	001	14
2	2	003	2
3	3	001	8

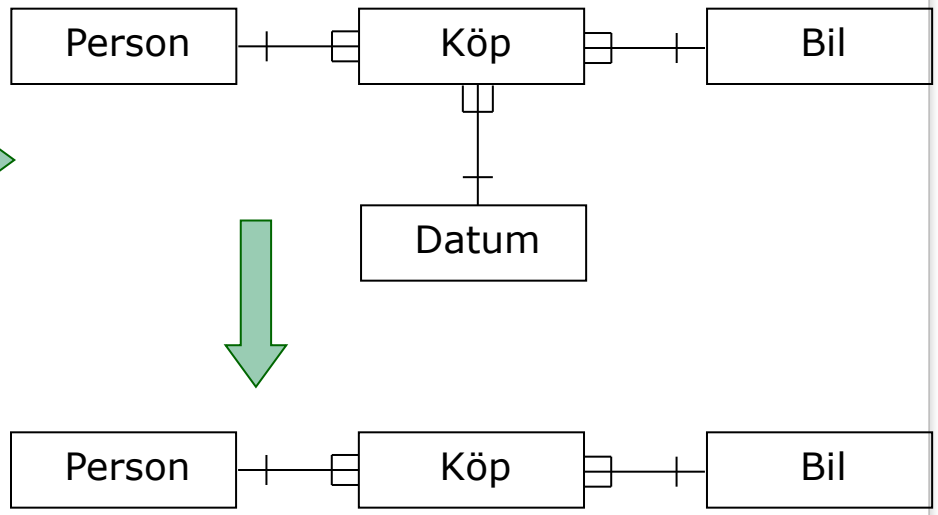
LagerID	FörrådID	ArtID	Antal
1	1	001	14
2	2	003	2
3	3	001	8

Bilköp forts.

Objektifiering (Konceptuell ➔ Logisk modell)

- I den logiska datamodellen gör du relationsobjektet till ett eget objekt
- Alla ”***gafflar***” på relationerna vänds emot det nya objektet





- Eftersom det är onödigt att bara ha en tabell för datum kan vi nu göra en optimering genom att lägga datum i ”***köp***”-tabellen
- Observera att vi inte gör optimeringen förrän på den fysiska datamodellen av den goda anledningen att optimera bort en databastabell och slippa ”***join***”-funktioner

Köp

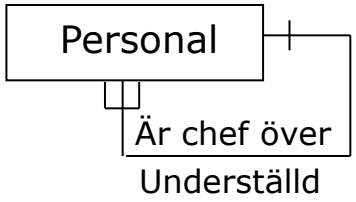
PersID	RegNr	RegDatum	Pris
1	ABC001	030101	550000
1	DFG234	030610	475000
3	KLM755	030825	345000

Egenrelationer

Relationer mellan olika förekomster (rader) av objekt

Exempelvis:

- Här registreras personalen i en hierarki indelad i chef och underställda

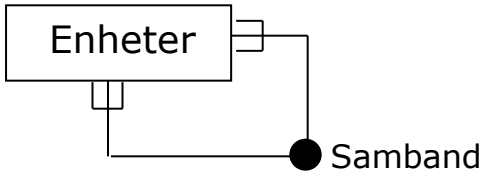


Personal

PersID	Namn	Chef	...
10	Kalle	0	
20	Linda	10	
30	Olle	20	

Exempelvis:

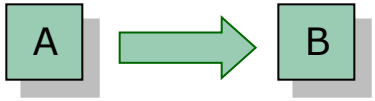
- En egenrelation kan även vara av typen många till många



Samband

Enhet1	Enhet2	Faktor
N/m^2	Pa	1
N/m2	bar	$1*10^{-5}$
bar	N/m2	$1*10^{5}$

Funktionellt beroende



B är funktionellt beroende av A om varje instans av A unikt bestämmer B. Betecknas: A →B (Om vi vet A så vet vi också B)

Exempel I:

Om du vet ett ***produktID*** så kan du hitta produktnamnet D.v.s. Produktnamnet är funktionellt beroende av ProduktID. ***ProduktID → ProduktNamn***

Produkt

ProduktID	ProduktNamn	...
1	T56	
2	X610	
3	S70	

Exempel II:

Nettoinkomsten = Bruttoinkomst - inkomstskatt
D.v.s. Nettoinkomsten är funktionellt beroende av bruttoinkomsten och inkomstskatten
(Bruttoinkomst, skatt) → Nettoinkomst

Inkomst

Brutto	Skatt	Netto
18000	31%	12420
23000	29%	16330
34000	32%	23120

Exempel III:

(PersID, Regnr, Regdatum) → Pris
Men för (PersID, Regnr) ↛ Pris eller PersID ↛ Pris finns inget funktionellt beroende

Köp

PersID	Regnr	Regdatum	Pris
1	ABC 120	030101	750000
2	DFG 456	990103	130000
3	YQW 321	000213	250000

Normalisering

- Normalisering innebär att befintliga tabeller testas m.a.p. olika beroende för att bl.a. avlägsna redundans och olika oönskade bieffekter (*anomalies*) vid radering, insättning och uppdatering av poster i databasen.

- Leder vanligen till att tabeller delas upp och struktureras om.

- Tabeller kan uppfylla olika normalformer beroende på hur långt normaliseringen drivs.

Första normalformen

Första normalformen (1 NF)

- Unik nyckel – En tabell måste ha en unik nyckel för varje post.
- Atomära fält – Ett fält får inte växa på bredden.

- Ett namn är ingen bra unik primärnyckel då det är talande och det kan finnas dubbletter
- Telefon får inte innehålla fler än en datapost

Kund

Namn	Postadress	Telefon	...
Guld AB	Gatan 1	123 45, 678 91, 100 00	
Silver AB	Vägen 2	101 00, 200 02, 555 55	
Platina AB	Stranden 3	555 02, 666 01, 444 77	

Dela upp Kundtabellen

- Uppdelningen av kundtabellen i två separata tabeller ger tabeller som uppfyller 1 NF

Kund

KundID	Namn	Postadress	...
1	Guld AB	Gatan 1	
2	Silver AB	Vägen 2	
3	Platina AB	Stranden 3	

Telefon

KundID	TelID	Telefon
1	1	123 45
1	2	678 91
2	3	101 00

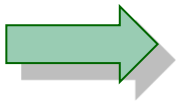
Första normalformen forts.

Postadress räknas som en adress och räknas inte som ett "***multivalued field***" och behöver INTE delas upp enligt 1 NF

Det är dock för det mesta en fördel att dela upp postadressen i flera fält

- Det underlättar sökning på de olika delarna av adressen.
- Det gör det enklare att presentera informationen på olika sätt.

Adress
Nobelvägen 1 130 00 Lund
Gastvägen 2 120 00 Kalmar
Amiralsvägen 3 122 00 Malmö



Postnr	**Gatuadress**	**Postadress**
130 00	Nobelvägen 1	Lund
120 00	Gastvägen 2	Kalmar
122 00	Amiralsvägen 3	Malmö

Namn
Kalle Petterson
Pelle Johansson
Lotta Persson

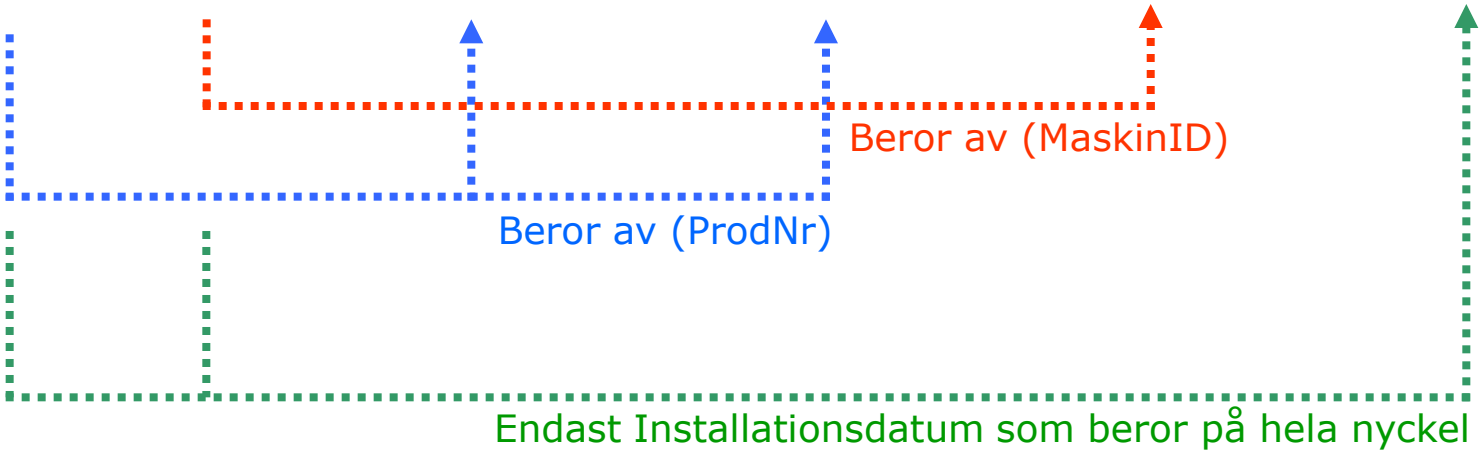
Förnamn	**Efternamn**
Kalle	Petterson
Pelle	Johansson
Lotta	Persson

Andra normalformen (2 NF)

- 1 NF + Alla icke nyckelfält ska vara funktionellt beroende av hela nyckeln

Tabellen nedan visar en databastabell som var tänkt att hålla reda på olika programinstallationer på olika datorer. Tabeller likt nedan skapas ofta då du utvecklar i fel ordning. D.v.s. du tittar bara på vad som ska presenteras och gör en tabell efter det

ProdNr	MaskinID	Produktnamn	Produkttyp	Maskinplacering	Installationsdatum
1	1	Win 2000	OS	BOM340	2002-01-01
2	2	Notepad	Texteditor	BBS210	1999-05-09
3	3	Gel	Programmeringseditor	Te215	2003-05-14



Andra normalformen forts.

Vi delar upp tabellen i sina logiska delar så att varje kolumn beror på hela nyckeln. Nu är alla tabeller i andra normalformen

Produkt

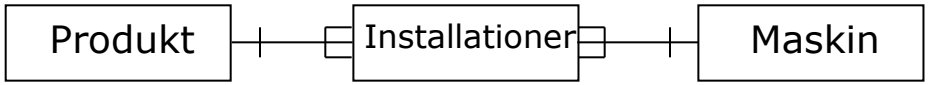
ProdNr	Produktnamn	Produkttyp
1	Win 2000	OS
2	Notepad	Texteditor
3	Gel	Programmeringseditor

Maskin

MaskinID	Maskinplacering
1	BOM340
2	BBS210
3	Te215

Installationer

ProdNr	MaskinID	Installationsdatum
1	1	2002-01-01
2	2	1999-05-09
3	3	2003-05-14



Beror på både (ProdNr och MaskinID)

2 NF gäller endast tabeller med komposit pk (sammansatt primärnyckel)

Tredje normalformen (3 NF)

- 2NF + Det får inte finnas några funktionella beroende mellan icke nyckelfält

På en jsp-sida ska information om kunder och vilket distrikt de tillhör presenteras, det är då lätt att skapa en databastabell som innehåller just den information vilket är FEL. TÄNK PÅ att inte skapa databastabellerna efter vad som ska presenteras

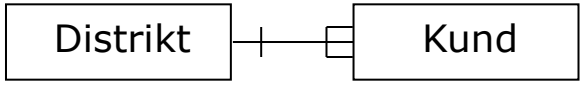
Kund

Kund	Namn	Postadress	Telefon	Distrikt	Diskriktnamn
1	Guld AB	Gatan 1	0480-556611	100	Kalmar
2	Silver AB	Vägen 6	08-56457821	200	Stockholm
3	Platina AB	Gränden 9	08-4589781	300	Stockholm

- Distriktnamn är funktionellt beroende av distrikt
- Du får onödig redundans genom att du måste dubbellagra ett distriktnamn för varje kund

Tredje normalformen forts.

För att uppnå 3 NF tas inbördes beroende bort genom att göra en separat Distriktstabell och endast behålla Distrikt som främmande nyckel i Kundtabellen



Kund

Kund	Namn	Postadress	Telefon	Distrikt (fk)
1	Guld AB	Gatan 1	0480-556611	100
2	Silver AB	Vägen 6	08-56457821	200
3	Platina AB	Gränden 9	08-4589781	300

Distrikt

Distrikt	Diskriktnamn
100	Kalmar
200	Stockholm
300	Stockholm

Mer exempel:
Spara inte fält som kan beräknas i databasen

(Antal * Pris) → Totalpris

Orderrad

OrderID	Radnr	Benämning	Antal	Pris	Totalpris
1001	1	Hårddisk	10	1700	17000
1001	2	RAM-minne	20	2500	50000
1001	3	Moderkort	1	1300	1300

Ska beräknas vid Visning istället

Fjärde normalformen

Fjärde normalformen i sin ursprungliga definition är krånglig att förstå och efterleva och brukar inte återfinnas i många databasböcker, därför presenterar jag en enklare och mycket användbar version av 4 NF

Fjärde normalformen (4 NF) i en förenklad variant

- 3 NF + Ett attribut får endast finns en gång per tabell

Kund

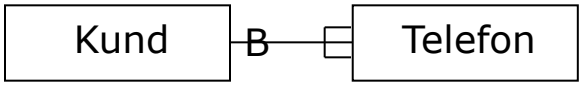
Kund	Namn	Postadress	Telefon1	Telefon2	Telefon3
1	Guld AB	Gatan 1	111 22	070 333 12	111 44
2	Silver AB	Vägen 6	222 22	222 33	
3	Platina AB	Gränden 9	333 22		

- Du slösar med minne när du antar att alla personer har tre telefoner och får tomma attribut
- Vad händer om en person har fler än tre telefoner?

Fjärde normalformen forts.

Lösning:

Gör en separat telefontabell. Denna kan växa obegränsat vi slipper på detta sätt tomma poster i databasen samt att en kund kan ha ett obegränsat antal telefoner



Kund

Kundnr	Namn	Postadress
1	Guld AB	Gatan 1
2	Silver AB	Vägen 6
3	Platina AB	Gränden 9

Telefon

Kundnr	Nr	Telefon	Typ
1	1	111 22	Hem
1	2	070 333 12	Mobil
1	3	111 44	Fax
2	1	222 22	Hem
2	2	222 33	Fax
3	1	333 22	Hem

Liknade situationer uppkommer vid lagring av:
- Kontaktpersoner
- Adresser
- m.m.

Det finns fler normalformer
Exempelvis:
- Boyce-Codd's och 5 NF

Dessa kan du för det mesta bortse från

Fördelar och nackdelar

Normalisering innebär att det skapas fler tabeller

- Fördelar:
  - icke redundanta tabeller är enklare att uppdatera och skala i storlek
    - Exempelvis:

      Om en felstavning upptäcks behöver du bara ändra på ett ställe istället för att gå igenom hela databasen.
  - det tar ofta mindre plats
    - Exempelvis:

      Du slipper lagra samma information på flera ställen i databasen.
- Nackdelar:
  - kan ta längre tid att söka i databasen
    - Exempelvis:

      Du måste göra ”***join***” på flera tabeller vilket gör att det går långsammare.
  - besvärligare programmering
    - Exempelvis:

      Det blir fler tabeller att hålla reda på (Utmaningar är till för att övervinnas).

Ej önskvärda bieffekter

- Deletion anomaly
  - Antag att vi raderar ”Uppland” då förlorar vi inte bara Uppland utan även Stockholmskommun och dess invånare.

Landskap	Kommun	Distrikt
Uppland	Stockholm	Norra
Småland	Kalmar	Centrum
Öland	Borgholm	Södra

- Insertion anomaly
  - Antag att vi vill registrera att EU-Masters kostar 80 000 Euro, så kan vi inte göra detta förrän en golfspelare har anmält sig till tävlingen.

PersID	Tävling	Avgift
770113	US Open	50000
801012	US Masters	65000
750206	EU Masters	70000

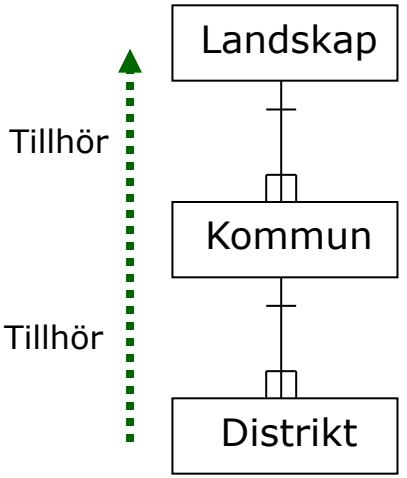
- Update anomaly
  - Antag att avgiften för US-Open ska ändras. Då måste alla poster där US-Open förekommer uppdateras.

PersID	Tävling	Avgift
770113	US Open	50000
801012	US Open	50000
750206	US Open	50000

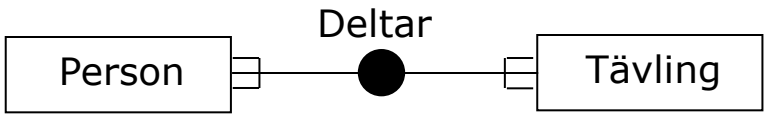
Bättre lösning

En bättre lösningar för att undvika *anomalies*

- En bättre lösning på det första problemet är en så kallad stabil hierarkistruktur



- En bättre lösning på det andra problemet



Person

PersID	Namn
1	Kalle
2	Linda
3	Olle

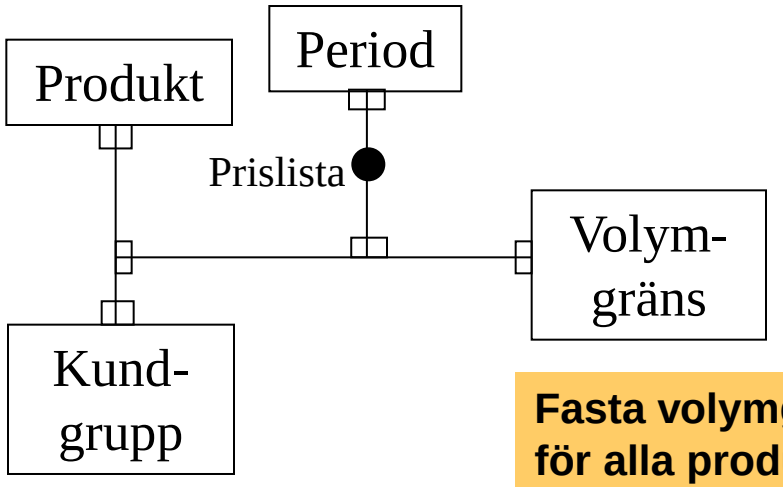
Deltar

PersID	TävlingID
1	1001
2	1002
3	1003

Tävling

TävlingID	Namn	Avgift
1001	US Open	50000
1002	US Masters	65000
1003	EU Masters	75000

Regelverk



Regelverk beskriver ett antal kriterier som ska vara uppfyllda för att erhålla ett visst resultat.

Ex. Stafflade priser, bonusskalor, provisionstabeller mm.

Fasta volymgränser för alla produkter

Prislista

ProdID	KundgrpID	Från	VolymGränsID	Pris
100	10	020825	1	100
100	10	020825	2	90
100	10	020825	3	80
100	10	021231	1	105
100	10	021231	2	95
100	10	021231	3	85

För att erhålla rätt pris måste fyra värden i relationen anges: Produkt, Kundgrupp, Datum och Volym

Sammanfattning

Sammanfattning:

- Hur skapar du en databasmodell
  - Skapa en konceptuell databasmodell, generalisera, identifiera objekten ur verksamheten.
  - Normalisera och objektifiera din modell så får du en logisk modell.
  - Skapa den fysiska modellen genom denormalisering, tabellprecisering, optimera utifrån logiska modellen. Använd sunt förnuft men dokumenterna noga alla ändringar.
- Primärnyckeln identifierar varje post i en tabell unikt
- Främmande nyckel identifierar vilken post som är relaterad till vem
- De olika relationerna
  - 1:1 en till en
  - 1:n en till många
  - n:1 många till en
  - n:m många till många (skapar alltid ett relationsobjekt)
- Ej önskvärda bieffekter
  - Deletion anomaly
  - Insertion anomaly
  - Update anomaly
- De fyra första normalformerna
  - 1NF → Unik nyckel och Atomära fält
  - 2NF → 1NF + Alla icke nyckelfält ska vara funktionellt beroende av hela nyckeln
  - 3NF → 2NF + Det får inte finnas några funktionella beroende mellan icke nyckelfält
  - 4NF → 3NF + Ett attribut får endast finns en gång per tabell