Kapitel 1

Databaser och databashanterare

I det här kapitlet går vi igenom databasteknikens grunder, och förklarar vad som menas med termer som databaser, databashanterare, datamodeller och scheman.

1.1 Vad är en databas?

Data är uppgifter av olika slag. Ibland skiljer man data från *information*, som är data som man gett en tolkning. Alltså är **23** ett exempel på data, medan det är information om vi vet att **det är 23 grader varmt ute**. Ibland talar man också om *kunskap*, som i "kunskapsbaserade system". Kunskap är anvisningar om beteende, exempelvis regler för hur datorn ska dra slutsatser.

Med ordet *databas* brukar man mena

- en *samling data* som
- *hör ihop*,
- som *modellerar en del av världen*, till exempel ett företag och dess verksamhet,
- och som är *persistent*, dvs inte försvinner när man avslutar programmet eller stänger av datorn. [1]

Folk som håller på med databasteknik brukar också mena att en databas ska

- ha ett *schema*, dvs det ska finnas en explicit beskrivning av vad det är för data,
- vara *konsistent* eller *logiskt koherent*, dvs den får inte innehålla motsägelser.
- Åtminstone i det följande antar vi också att databasen lagras och hanteras av en *dator*.

1.2 Vad är en databashanterare?

Det finns program som har till uppgift att lagra och hantera databaser. Ett sådant program kallas för *databashanterare*, *databashanteringssystem* eller *DBHS*. På engelska kallas det *database management system* och förkortas *DBMS*.

[1] Ordet *persistent* kommer från ett ord på engelska som betyder envis.

Många databashanterare är mycket stora och komplicerade, och är egentligen hela system av olika program.

Några kända exempel på databashanterare är **DB2** från IBM, **Informix**, **INGRES**, **InterBase** från Borland, **Microsoft Access**, **Microsoft SQL Server**, **Mimer**, **MySQL**, **ObjectStore**, **Oracle** och **Postgres**.

1.3 Språkförbistring

Termerna används ibland slarvigt eller med lite olika betydelser. Exempelvis används ordet "databas" ofta för att beteckna det som vi här kallar för "databashanterare". Ordet "databassystem" används ibland för att beteckna det som vi kallar för "databashanterare", och ibland för kombinationen av det vi kallar "databas" och "databashanterare". Hur orden "data" och "information" används ska vi bara inte tala om.

1.4 Databashanterare - varför?

Om man ska förstå fördelarna med att använda databasteknik, dvs att låta en databashanterare hantera ens data, måste man jämföra med alternativet. Alternativet är för det mesta att ha en eller flera *vanliga filer* med data. Sen skriver man ett program, i något vanligt programmeringsspråk som C eller Java, och låter det programmet hantera datafilerna.

Om man exempelvis vill ha ett kundregister, med nummer, namn och adress, börjar man kanske med att definiera en post:

```
struct kund {
    int nummer;
    char namn[50 + 1];
    char adress[50 + 1];
    struct kund* nextp;
};
```

Sen fortsätter man med ungefär 2000 rader programkod, som sköter dialogen med användaren och som läser och skriver datafilen med kunder.

Det finns ganska många fördelar med att i stället använda en databashanterare. De viktigaste fördelarna är att det är

- *enkelt*,
- *kraftfullt*, och
- *flexibelt*.

Fördel 1: Enkelt!

Många databashanterare erbjuder ett textbaserat gränssnitt. Starta databashanteraren och skriv:

```
create table kund
(nummer integer,
namn char(50),
adress char(50));
```

Klart! Nu finns det en tabell som ser ut så här (när man stoppat in lite data):

Nummer	Namn	Adress
7	Hjalmar	Vägen 7
113	Hulda	Stora torget 18
4	Lotta	Vägen 7

Faktaruta: Ett sånt språk som vi använde för att skapa tabellen brukar kallas för *datadefinitionsspråk* eller *DDL*. Just det här språket heter *SQL*, och är mycket vanligt i databassammanhang.

Uppgifterna i tabellen kallar vi (förstås) för *data*. Tabellens utseende, dvs vilka kolumner som finns, vad de heter, och vad de får innehålla, kallas för *schema*. Schemat bestämmer vilka data som kan lagras i databasen. I schemat ovan använder vi oss av tabeller och kolumner. Vi kunde i stället ha använt något annat, t ex objekt och klasser. Hur schemat kan se ut bestäms av databasens *datamodell*. Om man vill kan man säga att datamodellen är ett "schema för schemat". Datamodellen bestämmer alltså hur schemat får se ut, och schemat bestämmer vilka data som får lagras i databasen. (Datamodellen med tabeller kallas *relationsmodellen*.)

En varning för språkförbistring: Ordet "datamodell" används ofta som namn på det vi har kallat "schema", alltså en beskrivning av hur innehållet i en viss databas ser ut.

Nu ska vi använda vår databas också. Antag att vi vill ha reda på namnet och adressen för kunden med kundnummer 17. Då skriver vi bara:

```
select namn, adress
from kund
where nummer = 17;
```

Klart!

Faktaruta: En sökning i databasen brukar kallas en *fråga*. Ett språk som man använder för att ställa frågor till databashanteraren brukar kallas för *frågespråk*, *datamanipuleringsspråk* eller *DML*. Vi använde SQL här också. SQL innehåller alltså både ett datadefinitionsspråk och ett datamanipuleringsspråk.
Ibland kallar man alla operationer, alltså även t. ex. att lägga in data i databasen, för frågor.

Fördel 2: Kraftfullt!

Att ett system är *kraftfullt* betyder att komplicerade saker går att göra på ett enkelt sätt.

Antag att vi vill ha reda på alla kunder som har namn som börjar med **S**, och få dem utskrivna i bokstavsordning efter adressen. Då skriver vi bara så här:

```
select *
from kund
where namn like 'S%'
order by adress;
```

Vill du veta hur många personer som heter **Anders** som bor på varje adress? Enkelt:

```
select adress, count(*)
from kund
where namn = 'Anders'
group by adress;
```

Fördel 3: Flexibelt!

Att ett system är *flexibelt* betyder att det är lätt att ändra.

Kommer du plötsligt på att du vill göra en helt ny sorts sökning i databasen? Enkelt! Skriv bara in sökningen som en fråga:

```
select namn
from kund
where adress = 'Vägen 8'
and namn like 'S%';
```

Kommer du plötsligt på att du vill ha med telefonnummer också i kundregistret? Enkelt! Skriv:

```
alter table kund add telefon char(10);
```

Nu har du utökat tabellen **kund** med en ny kolumn, som heter **telefon**!

Faktaruta: Att det går att ändra den logiska strukturen på datat så här, utan att man måste skriva om en massa program, kallas *logiskt dataoberoende.*

Har kundregistret använts ett tag, och nu innehåller det så många kunder att det tar lång tid att till exempel söka efter en kund med visst namn? Enkelt! Skriv:

```
create index foo on kund(namn);
```

Nu har du ändrat den fysiska lagringsstrukturen så att det går fortare att söka efter ett visst namn. Tabellen ser fortfarande likadan ut, men nästa gång du söker på ett namn kommer frågan att gå mycket fortare att köra. Databashanteraren utnyttjar automatiskt den nya lagringsstrukturen.

Faktaruta: Att det går att ändra den fysiska lagringsstrukturen på datat, utan att man måste skriva om en massa program, kallas *fysiskt dataoberoende.*

Andra fördelar

Det finns flera saker som är mycket besvärliga att få att fungera om man skriver ett program själv,men som finns inbyggda från början i de flesta databashanterare.

Databasteknik möjliggör samtidig åtkomst av data.

Vad händer om flera personer samtidigt håller på och ändrar i kundregistret? Då är det lätt hänt att en person skriver över ändringar som en annan person gjort. Databashanteraren ser till att det inte blir några skadliga krockar.

Databasteknik möjliggör återställande efter krascher.

Vad händer om strömmen plötsligt går, så att datorn "tappar minnet"? Om ens program läser in datafilen i primärminnet, och skriver tillbaka filen när man jobbat klart, blir man kanske av med alla de ändringar man gjort. Och ännu värre: om man precis höll på att spara när strömmen gick, kanske en del av de data som finns på disken är de nya, och en del är de gamla, och man vet inte vilka som är vilka. I värsta fall får man kasta bort hela datafilen!

Med en databashanterare slipper man sådana problem. Den ser till att inga data någonsin försvinner, hur olyckligt ett strömavbrott än skulle komma.

Databasteknik gör det lättare att möta olika användares behov.

Olika användare kan ha helt olika behov, även om de arbetar med samma data. En databashanterare erbjuder ofta flera olika *gränssnitt*, dvs sätt för användaren att kommunicera med databasen. Man brukar också kunna definiera olika *vyer*. En vy är ett sätt att se databasen. Till exempel kan man skriva så här:

```
create view kund2
as select nummer, namn
from kund;
```

Den användare som går genom vyn **kund2** för att titta på databasen, ser bara kolumnerna **nummer** och **namn** i tabellen med kunder.

Det går att definiera mer komplicerade vyer. Till exempel kanske en annan användare bara är intresserad av antalet kunder, och bryr sig inte om vilka de är? En vy löser det problemet:

```
create view kund3
as count(*)
from kund;
```

Databasteknik möjliggör bättre säkerhet.

De flesta databashanterare har mekanismer för att ge olika användare olika rättigheter i databasen, och för att skydda data mot obehörig åtkomst. Till exempel kan man ge en användare rätt att ändra i vissa delar av databasen och att söka i andra delar, medan hon inte alls får se andra delar av databasen.

1.5 Nackdelar med databaser

Databasteknik passar inte för alla tillämpningar. Exempelvis brukar all den där enkelheten och flexibiliteten som vi pratade om ovan, göra att en databashanterare kräver mer resurser än ett specialskrivet program. Det går alltså åt mer minne och diskutrymme. Kanske går det också långsammare att köra.

Å andra sidan kan en lösning med databashanterare i stället gå mycket fortare än ett specialskrivet program! Orsaken är att databashanteraren innehåller avancerade datastrukturer och algoritmer, som man sällan orkar ta med i det specialskrivna programmet.

1.6 Användningsområden för databasteknik

Allt. Låt inte lura er av att databasböcker bara brukar ha kundregister och studentdatabaser som exempel. Databaser används också i CAD-system ("Computer-Aided Design"), CASE-system ("Computer-Aided Software Engineering"), telefonväxlar, styr- och reglertillämpningar, och mycket annat.

1.7 Datamodeller

Vilken datamodell man använder bestämmer hur schemat får se ut. Schemat bestämmer sen i sin tur vilka data som kan lagras i databasen.

Man brukar dela in datamodellerna i tre klasser:

- *Konceptuella* eller *begreppsmässiga* datamodeller. Om man ska skapa en databas som beskriver en del av verkligheten, t ex ett företag, brukar man börja med att göra en beskrivning av hur den delen av verkligheten ser ut och fungerar. Till det kan man använda en konceptuell datamodell. Den konceptuella modellen behöver egentligen inte ha någonting med datorer att göra, utan är bara en beskrivning av verkligheten som lika gärna skulle kunna användas av någon som bara vill analysera hur företaget fungerar.
  Ett exempel på en konceptuell datamodell är den så kallade *Entity-Relationship-modellen*, ofta förkortat till *ER-modellen*. När man använder ER-modellen ritar man upp de saker ("entities") som man vill lagra i databasen, och de förhållanden eller samband ("relationships") som finns mellan dem.
- *Implementationsmodeller*. Det här är de datamodeller som används i databashanterare. Om man vill skapa en databas måste den beskrivning man gjort med hjälp av en konceptuell datamodell först översättas till något som går att stoppa in i en dator, nämligen en beskrivning enligt en implementationsmodell.
  Den vanligaste implementationsmodellerna numera är *relationsmodellen*, som går ut på att man lagrar data i tabeller. Dessutom har det dykt upp olika typer av *objektorienterade modeller*. Tidigare var den *hierarkiska* datamodellen och *nätverksmodellen* vanliga.
- *Fysiska datamodeller*, som kan användas för att beskriva hur data lagras fysiskt.

1.8 Tre-schema-arkitekturen

Det kan vara praktiskt att betrakta sin databas på tre olika nivåer. Det är hela tiden samma data, men man använder *tre olika scheman* för att beskriva dem:

- Den *externa* nivån, eller *vy-nivån*, beskriver hur de olika användaren ser databasen. Detta beskrivs av det *externa schemat*.
- Den *logiska* nivån beskriver hela databasen, uttryckt i den implementationsmodell som databashanteraren använder. I en relationsdatabas består den logiska nivån av alla tabellerna i databasen. Detta beskrivs av det *logiska schemat*.
- Den *interna* eller *fysiska* nivån, som beskriver hur data är lagrat. Som användare märker man oftast inte alls av den här nivån, utom som hur fort det går att göra olika operationer. Den fysiska nivån beskrivs av det *fysiska schemat*.

Nu kan vi passa på att definiera *logiskt* och *fysiskt dataoberoende* på ett bättre sätt än vi kunde göra tidigare:

- *Logskt dataoberoende* innebär att man kan ändra det logiska schemat, utan att det externa schemat påverkas.
- *Fysiskt dataoberoende* innebär att man kan ändra det fysiska schemat, utan att det logiska schemat påverkas.

Tre-schema-arkitekturen kallas ibland "tre-nivå-arkitekturen".

1.9 Olika typer av användare

Ibland skiljer man på olika typer av användare, dvs personer som arbetar med databasen:

- **Expertanvändare** använder databasen ofta, kanske dagligen. De är väl insatta i databasens uppbyggnad och databashanterarens funktion, och sitter kanske och skriver SQL-frågor.
- **Tillfälliga** eller **naiva** användare kan inte lika mycket om databasen, kanske för att de använder den mer sällan. De behöver enklare och mer nybörjarvänliga verktyg än experterna. Man kan inte kräva att alla biljettförsäljare på SJ ska lära sig SQL. (Termen "naiv användare" betyder förstås inte att personerna i fråga är naiva i största allmänhet, ifall någon nu trodde det, utan bara att de inte har en djup förståelse just när det gäller databassystemet.)
- **Databasadministratören**, förkortat **DBA**, ansvarar för databasens drift, ungefär som en systemansvarig för vilket datorsystem som helst. För stora databassystem kan det vara en hel grupp av **DBA-personal**. DBA-personalen utformar och förändrar databasschemat, registrerar nya användare, och ser med olika justeringar till att databassystemet fungerar bra och effektivt.
- **Andra datorprogram.** Det är inte ovanligt att en del av databasens "användare" är andra datorprogram, som hämtar data från databasen, och skickar data till den. En del databaser har inga mänskliga användare alls, utan det är bara ett eller flera datorprogram som använder sig av dem. Kanske kommunicerar de där andra programmen sen i sin tur med människor, men inte ens det är säkert.

Om jag samlar mina kakrecept i en databas i min hemdator, är det antagligen jag själv som spelar alla dessa roller. I stora system, som en stor biljettbokningsdatabas, kan det finnas hundratals eller tusentals personer som arbetar med databasen.

1.10 Databashanterare - hur?

En databashanterare är oftast ett stort och komplicerat program, eller ett helt system av program. Förutom själva "kärnan" i databashanteraren, som hanterar den lagrade databasen, finns det ofta flera olika **användargränssnitt**, dvs program (eller delprogram eller system av program) som användaren kan använda för att söka eller ändra i databasen. Det brukar finnas ett eller flera frågespråk som SQL, men också olika grafiska verktyg. Olika gränssnitt passar för olika användare och för olika användningsområden.

Det är också vanligt att en databashanterare innehåller verktyg för att bygga **applikationsprogram**. Ett applikationsprogram är ett program som är avsett för ett specifikt ändamål, till exempel för att låta biljettförsäljarna på SJ boka tågresor. I det här sammanhanget tänker vi oss att alla data lagras i en databas som hanteras av databashanteraren, och applikationsprogrammet kommunicerar med databashanteraren, men applikationsprogrammet är enklare att använda för biljettförsäljaren. I det programmet kan man kanske klicka på knappar för att välja resmål. Det är lättare än att kommunicera med databashanteraren direkt, i värsta fall genom att skriva SQL-kommandon. Med verktygen för att bygga applikationsprogram kan man (mer eller mindre enkelt) sätta ihop ett sådant program, kanske utan att man behöver skriva någon programkod.

Applikationsprogrammerare är ännu en typ av databasanvändare. Applikationsprogrammeraren tillverkar applikationsprogram, antingen med hjälp av särskilda verktyg eller genom att skriva program i ett vanligt programmeringsspråk med inlagda anrop till databashanteraren.

En normal databashanterare lagrar alla data på disk, och hämtar bara in de data som den för tillfället behöver till primärminnet. Men det finns också databashanterare som har alla data i primärminnet hela tiden.

När man installerat en databashanterare på sin dator, kan den ofta hantera flera olika databaser samtidigt. Varje databas består egentligen av två samlingar data: dels databasens innehåll, dels schemat. Schemat kallas också **datakatalog** eller **meta-data**, vilket betyder "data om data".

1.11 De viktigaste begreppen

De viktigaste begreppen från det här avsnittet finns också med i ordlistan:

data information kunskap databas databashanterare (DBMS) schema datamodell SQL fråga frågespråk logiskt dataoberoende fysiskt dataoberoende tre-schema-arkitekturen gränssnitt vy konceptuell datamodell implementationsmodell relationsmodellen DBA applikationsprogram applikationsprogrammerare datakatalog meta-data

1.12 Litteratur

Det finns många databasböcker på grundnivå, och de brukar alltid ha ett eller två kapitel i början som ger en inledning till området. Följande fyra böcker är de som brukar användas mest i grundläggande databaskurser på högsokolor och universitet runt om i världen. (Se kapitlet *Litteratur om databaser* sist i den här boken för fullständiga uppgiter om böckerna.)

- Connolly et al: *Database Systems - A Practical Approach to Design, Implementation and Management*, tredje upplagan: kapitel 1-2
- Date: *An Introduction to Database Systems*, sjunde upplagan: kapitel 1-2
- Elmasri/Navathe: *Fundamentals of Database Systems*, tredje upplagan: kapitel 1-2
- Silberschatz et al: *Database System Concepts*, fjärde upplagan: kapitel 1