Prova på-laboration i SQL

Peter Dalenius
petda@ida.liu.se
Institutionen för datavetenskap, Linköpings universitet
2006-09-19

1. Introduktion till databaser

Databaser finns i så gott som alla sammanhang där datorer används. Företag har register över sina kunder och lagrar information om sina produkter i en databas. Myndigheter lagrar information om medborgarna i olika register. Alla svenskar finns t.ex. med i *Statens person- och adressregister* (SPAR). Här på universitetet finns den välkända databasen LADOK i vilken lagras information om alla studenter och deras studieresultat.

Vad är en databas?

En *databas* är en samling data som innehåller relationer mellan olika delar. Som exempel kan vi ta en lista med namn och adresser. Den är i sin enklaste form ingen databas, men om man kopplar samman den med en lista över varje persons senast deklarerade årsinkomst har man skapat en relation. Varje person är relaterad till en årsinkomst.

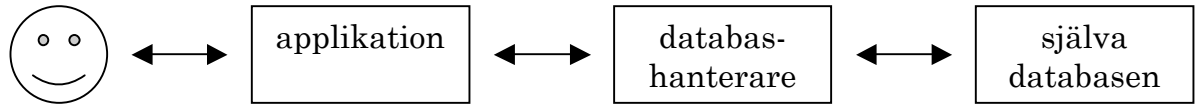
Något som kännetecknar databaser är att man kan välja att betrakta informationen från olika håll. Man kan å ena sidan hämta adresserna till alla personer med en årsinkomst under 300 000 kr. Å andra sidan kan man beräkna medelinkomsten för alla personer inom ett visst postnummerområde.

Man brukar säga att följande kriterier gäller för databaser:

- En databas representerar en liten del av verkligheten, dvs informationen som finns i databasen är ”på riktigt”.
- En databas är en logiskt koherent samling data som har en inbyggd mening. Vilken slumpartad samling data som helst är inte en databas.
- En databas är designad och fylld med data för en specifik uppgift. Den har en tänkt grupp av användare och några på förhand uttänkta tillämpnings-områden.

Användning av databaser

En databas är en behändig abstraktion av data. Som programmerare slipper vi fundera över exakt hur data ska lagras. Detta hanteras av ett särskilt program som vi kallar databashanterare eller *database mangagement system* (*DBMS*). De typiska lagren i ett färdigt system illustreras av figuren nedan. Användaren kommunicerar med applikationen som i sin tur kommunicerar med databas-hanteraren. Databashanteraren tar hand om själva databasen som t.ex. kan representeras av en fil på hårddisken.



Även om den exakta representationen av data är gömd behöver vi veta hur vi ska hantera informationen i databasen. För detta ändamål finns olika *databas-modeller*. En databasmodell ger oss en konceptuell bild av hur data är organiserat. Den vanligaste modellen, och den enda som vi kommer att ta upp i denna laboration, är *relationsmodellen*.

Relationsdatabaser

I en *relationsdatabas* kan vi tänka oss att data är organiserat i *relationer*, i vardagslag kallade *tabeller*. Varje rad i en sådan tabell representerar en relation mellan ett antal olika uppgifter. I tabellen *Adressregister* nedan är varje rad en relation mellan ett personnummer, ett namn och en adress. Raderna kallas ibland för *tupler* och kolumnerna för *attribut*.

Adressregister

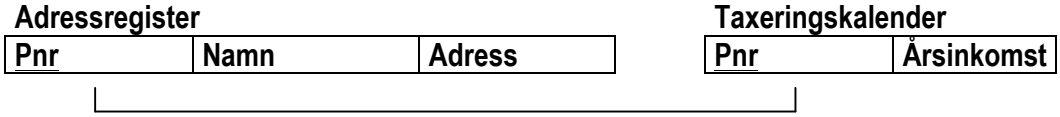
Pnr	Namn	Adress
430917-1234	Johan Karlsson	Ringv. 3
561020-5678	Karl Persson	Drottningg. 7
510205-9012	Per Svensson	Tulpanv. 82 A
490115-3456	Sven Johansson	Spång. 14

Taxeringskalender

Pnr	Årsinkomst
430917-1234	410 000
490115-3456	367 000
510205-9012	375 000
561020-5678	350 000

För att man lätt ska kunna hitta information i tabellerna brukar man vid designtillfället bestämma att en eller flera kolumner får utgöra en *nyckel*. En nyckel är ett unikt värde som bara får förekomma en enda gång och som används för att skapa *index*. Indexet är en intern tabell som gör att man snabbt kan hitta en rad i en stor databas om man känner till radens nyckel. När det gäller databaser med personregister är det naturligt att använda personnumret som nyckel, eftersom det är unikt för varje person.

Man skiljer på själva databasen och databasens *schema*. Det senare är en beskrivning av hur databasen ser ut, vilken typ av information som finns i den och vilka beroenden som finns mellan olika delar. I en relationsdatabas utgörs schemat i princip av tabellernas rubrikrader. Nedanstående figur är ett schema över vår exemepldatabas.



I schemat ovan har vi markerat nycklarna genom att stryka under kolumn-rubrikerna. Vi har också dragit en linje mellan de båda kolumnerna som heter *Pnr* för att markera att det finns en relation mellan tabellerna. Det person-nummer som förekommer i tabellen *Taxeringskalender* syftar ju på samma person som personnumret i tabellen *Adressregister*.

Varför är det två olika tabeller i exemplet ovan? Vi hade ju lika gärna kunnat lägga till en extra kolumn i adressregistret för att lagra årsinkomsten. Det kan finnas många anledningar till att dela upp informationen i separata tabeller. En anledning kan vara att man enbart vill tillåta vissa personer att titta på och ändra information om årsinkomst, medan en större grupp personer ska har rättigheter att bearbeta adressregistret.

Sammanfattning

Följande begrepp har introducerats i detta avsnitt:

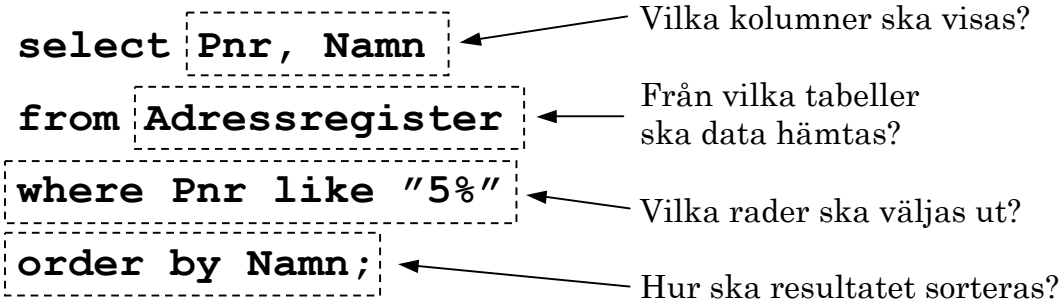
- databas
- databashanterare (eng. *database management system*)
- databasmodell
- relationsdatabas
- tabell (eller relation)
- rad (eller tupel)
- kolumn (eller attribut)
- nyckel
- index
- schema

2. Introduktion till SQL

För att underlätta för programmerare som vill använda databashanterare i sina program har man tagit fram *frågespråk*, formella språk som liknar vanliga programmeringsspråk men vars enda användningsområde är att hantera databaser. Ett av de mest kända frågespråken är SQL (eng. *structured query language*) vars ursprung kan spåras tillbaka till IBM i mitten av 1970-talet. SQL är dock idag ett fritt språk som inte är kopplat till något företag. Det har standardiserats i tre olika omgångar av både ANSI[1] och ISO[2] under 1989, 1992 och 1999.

Enkla frågor

SQL är ett frågespråk och dess vanligaste uppgift är att formulera frågor till en databas. Frågorna tar ofta formen av en `select`-sats som väljer ut rader och/eller kolumner från en eller flera tabeller. Nedanstående exempel väljer ut alla personer födda på 1950-talet och sorterar dessa i bokstavsordning efter namnet. Enbart kolumnerna *Pnr* och *Namn* visas.



Frågan består av flera delar, men alla behöver inte vara med varje gång. De två viktigaste delarna talar om vilka kolumner som ska visas och från vilken eller vilka tabeller som data ska hämtas. Exempel:

`select * from Adressregister;`	Visar alla kolumner från tabellen *Adressregister*.
`select Pnr, Namn` `from Adressregister;`	Visar kolumnerna *Pnr* och *Namn* från tabellen *Adressregister*.

Med hjälp av `where`-delen kan vi välja ut bara vissa rader. Exempel:

`select * from Adressregister` `where namn="Per Svensson";`	Visar enbart rader där namnet är Per Svensson.

[1] *American National Standars Institute*, http://www.ansi.org/
[2] *International Organisation for Standardization*, http://www.iso.org/

`select * from Adressregister where Pnr like ”5%”;`	Visar enbart rader där personnumret börjar med siffran 5. Procentttecknet matchar godtyckligt antal tecken.
`select * from Adressregister where Pnr like ”__10%”;`	Visar enbart personer födda i oktober. Varje understreck matchar ett tecken.

Man kan lägga till flera villkor efter `where` genom att placera `and` eller `or` mellan. Om vi vill kan vi också lägga till en del som talar om hur resultatet ska sorteras. Exempel:

`select * from Adressregister order by Adress;`	Sorterar resultatet i bokstavsordning efter adressen.

Det finns en lång rad inbyggda funktioner i SQL som man kan använda i olika delar av `select`-frågan. Här följer några exempel:

`select * from Adressregister where length(Namn)=10;`	Visar enbart rader där namnet är 10 tecken långt.
`select * from Adressregister where locate(”an”,Namn)>0;`	Visar enbart personer som har teckensekvensen *an* någonstans i namnet. Funktionen `locate` returnerar ett tal som motsvarar posititionen för delsträngen som söks.
`select count(*) from Adressregister where Pnr like ”5%”.`	Visar inte raderna, utan enbart hur många rader som finns i resultatet.
`select avg(Årsinkomst) from Taxeringskalender;`	Visar inte raderna, utan enbart medelvärdet av alla tal i kolumnen *Årsinkomst*.
`select sum(Årsinkomst) from Taxeringskalender;`	Visar inte raderna, utan enbart summan av alla tal i kolumnen *Årsinkomst*.

Frågor som använder flera tabeller

Frågorna i föregående avsnitt arbetar enbart på en tabell, men poängen med databaser är ju att man kan ha flera tabeller som på olika sätt är relaterade till varandra. SQL är ett kraftfullt språk som kan användas för att uttrycka ganska komplicerade operationer där flera tabeller kopplas samman.

Som exempel kan vi tänka oss en fruktaffär. I affärens databas finns två tabeller. *Produktinformation* innehåller allmänna data om olika typer av frukter som man säljer. Denna tabell har endast chefen rätt att modifiera. Tabellen *Lager* innehåller information om hur många frukter av varje sort som man just nu har till försäljning. Denna tabell kan modiferas av alla anställda.

Produktinformation

Frukt	Färg
Banan	Gul
Kiwi	Grön
Citron	Gul

Lager

Frukt	Antal
Banan	120
Kiwi	40
Citron	35

Om chefen är intresserad av att veta hur många gula frukter som för närvarande finns i lager måste informationen från båda tabellerna kombineras på något sätt. Vi kan slå ihop tabellerna med frågan

```
select * from Produktinformation, Lager;
```

men detta kommer att ge resultatet

Frukt	Färg	Frukt	Antal
Banan	Gul	Banan	120
Banan	Gul	Kiwi	40
Banan	Gul	Citron	35
Kiwi	Grön	Banan	120
Kiwi	Grön	Kiwi	40
Kiwi	Grön	Citron	35
Citron	Gul	Banan	120
Citron	Gul	Kiwi	40
Citron	Gul	Citron	35

När SQL slår ihop två tabeller bildas den *kartesiska produkten* av alla rader, dvs varje rad i den ena tabellen kombineras med varje rad i den andra tabellen. Om tabellerna har n respektive m rader blir resultatet $n \cdot m$ rader. Vi är dock bara intresserade av de rader där frukten i den ena tabellen är densamma som frukten i den andra tabellen. Detta måste vi tala om för SQL genom att skriva

```
select * from Produktinformation, Lager
where Produktinformation.Frukt=Lager.Frukt;
```

Eftersom vi har en kolumn Frukt i vardera tabellen måste vi tala om vilken tabell vi menar genom att skriva tabellnamnet följt av en punkt och därefter kolumnen. Resultatet av ovanstående fråga blir

Frukt	Färg	Frukt	Antal
Banan	Gul	Banan	120
Kiwi	Grön	Kiwi	40
Citron	Gul	Citron	35

För att välja ut enbart de frukter som är gula, samt för att summera antalet skriver vi följande fråga

```
select sum(Antal) from Produktinformation,Lager
where Produktinformation.Frukt=Lager.Frukt and Färg="Gul";
```

Detta kommer ge resultatet

sum(Antal)
155

3. Förberedelser för övningar

I denna laboration ska vi använda oss av en databashanterare som heter MySQL.[3] Det är en av världens mest använda databashanterare som bygger på öppen källkod och den passar utmärkt för små och medelstora applikationer. *Observera att MySQL-servernkräver ett separat lösenord som du har fått på mailen. Om du inte fått ett lösenord genererat så ska du maila helpdesk@ida.liu.se från din studentmail och be dem göra detta.*

För att starta en MySQL-klient och ansluta till databashanteraren från IDA:s Sun-stationer öppnar du ett skalfönster och skriver nedanstående.

```
zaza12 <1> module add office/mysql
zaza12 <2> module initadd office/mysql
zaza12 <3> mysql -h db-und.ida.liu.se -p
Enter password:

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.23.53-max-debug

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

När du har startat MySQL-klienten får du upp en prompt där du kan skriva in kommandon, ungefär som i ett vanligt skalfönster. Börja med att välja din användares databas: detta görs med kommandot ”use”, om din användare heter XXXXX123 blir kommandot alltså:

```
mysql> use XXXXX123;
```

Importera sedan den färdiga databasen du ska använda:

```
mysql> source /home/TDP001/www-pub/material/prova-pa/pp-tables.sql
mysql> show tables;
+-----------------------+
| Tables_in_pp_turte123 |
+-----------------------+
| course                |
| registry              |
| user                  |
+-----------------------+
3 rows in set (0.00 sec)

mysql>
```

Alla förfrågningar till databasen avslutas med semikolon (;). Om du glömmer skriva semikolon i slutet visas en ny rad där din förfrågan fortsätter. Du kan då fylla på med semikolon. Det är alltså möjligt, och ibland önskvärt, att dela upp en förfrågan på flera rader.

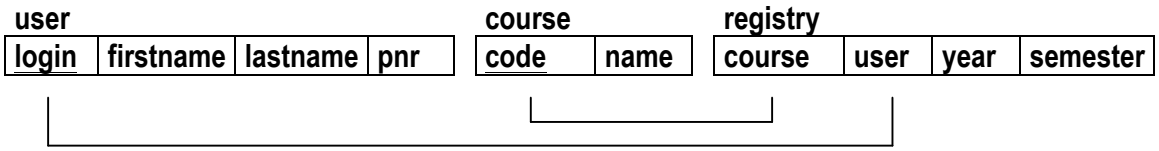
[3] Se http://www.mysql.com/

Om du lyckats logga in och fått fram en lista över tabellerna i din databas som stämmer överens med exemplet på föregående sida är du redo att ta itu med själva övningarna.

4. Övningar på SQL

Övning 1

I exempeldatabasen finns tre tabeller som innehåller information om några studenter och vilka kurser de läser. Tabellen *user* innehåller studenterna, *course* innehåller information om kurserna och *registry* kopplar samman de båda andra tabellerna genom att hålla reda på vem som är registrerad på vilken kurs. Databasschemat ser ut så här:



Formulera och testa SQL-frågor som löser följande uppgifter. Anteckna gärna frågan på respektive tomrad nedan.

1. Visa alla kurser!

_____

2. Visa alla studenter som har ett förnamn som börjar på *T*!

_____

3. Räkna antalet studenter som har ett efternamn som slutar på *-son*!

_____

4. Visa användarnamn, förnamn och efternamn för alla studenter som är registrerade på kursen *TGTU50*!

_____

_____

Övning 2

Vi kan lägga till information i en tabell med kommandot `insert`. Vi måste då komma ihåg i vilken ordning kolumnerna är lagrade. För att lägga till en ny student kan vi t.ex. skriva:

```
mysql> insert user values
    -> ("zaran700", "Zara", "Andersson", "850102-2225");
Query OK, 1 row affected (0.00 sec)

mysql> select * from user where firstname="Zara";
+----------+-----------+-----------+-------------+
| login    | firstname | lastname  | pnr         |
+----------+-----------+-----------+-------------+
| zaran700 | Zara      | Andersson | 850102-2225 |
+----------+-----------+-----------+-------------+
1 row in set (0.00 sec)

mysql>
```

I exemplet ovan är kommandot uppdelat på två rader. Vi skriver alltså `insert` följt av tabellens namn, därefter `values` och en lista med värden som kommer i samma ordning som kolumnerna. Formulera och testkör kommandon som utför följande uppgifter:

5. Lägg till dig själv i tabellen över studenter!

---

6. Lägg till en ny kurs *Föreläsningssömn* med kurskod *TZZZ01*!

---

7. Registrera dig själv på den kurs du nyss lade till genom att lägga till lämplig information i tabellen *registry*!

Kontrollera med lämpliga `select`-frågor att tabellerna har uppdaterats korrekt. Om du lagt till rader i en tabell med sedan ångrar dig kan du ta bort dem med kommandot `delete from` *`tabellnamn`*. Kommandot `delete` använder sig av en `where`-del precis som `select`. Du kan alltså skriva t.ex.:

```
mysql> delete from user where firstname="Zara";
Query OK, 1 row affected (0.00 sec)

mysql>
```

Observera att om du inte anger någon `where`-del kommer alla rader att försvinna ur tabellen och du har ingen möjlighet att ångra!

Övning 3

Vi kan skapa nya tabeller med kommandot `create table`. När vi skapar en tabell måste vi ange namn och datatyp för alla kolumner. Vi kan också ange vilken eller vilka av kolumnerna som ska fungera som nyckel. Tabellen *user* är t.ex. skapad med nedanstående kommando:

```
mysql> create table user (
    ->   login varchar(8) not null,
    ->   firstname varchar(20),
    ->   lastname varchar(30),
    ->   pnr varchar(11),
    ->   primary key (login)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Vi skriver alltså `create table` följt av namnet på vår ny tabell. Därefter följer inom parenteser en lista med kolumner separerade med komma. Varje kolumn-specifikation består av namnet på kolumnen följt av datatypen. Vi kan också ange specialönskemål. För kolumnen *login* i tabellen *user* har vi angivit `not null` vilket innebär att den inte får vara tom. Detta är ett krav för att kolumnen ska kunna utgöra nyckel, vilket specificeras på näst sista raden i kommandot.

Exempel på datatyper är `varchar(`*n*`)` som innebär att kolumnen är en textsträng på maximalt *n* tecken, `integer` som innebär heltal och `float` som innebär flyttal. Formulera och testkör kommando som löser följande uppgift:

8. Skapa en ny tabell som kan användas för att lagra information om vilka studenter som blivit godkända på vilka kurser! Börja med att tänka över vilken typ av information som behöver finnas och översätt detta sedan till lämpliga kolumner.

9. Lägg till en rad i din nya tabell som indikerar att du själv är godkänd på kursen *Föreläsningssömn*!

Om du råkat skapa en tabell som du inte vill ha kan du ta bort den med kommandot `drop table` *`tabellnamn`*.

5. Referenser

Framtida kurser

Både på C- och D-programmet finns kurser i databasteknik som går igenom både teori och praktik kring databaser. Här får man lära sig hur man designar databaser, olika sätt att modellera dem samt inte minst öva mer på SQL. Kursen tar också upp hur man kan säkerställa att informationen i databaser som används av många personer inte blir felaktig.

Litteratur

I den rekommenderade referensboken *Computer Science: An Overview* av J. Glenn Brookshear är särskilt avsnitt 9.1, 9.2, 9.6 och 9.7 intressanta. Avsnittet om operationerna SELECT, PROJECT och JOIN som börjar på s. 369 kan läsas kursivt. Det viktiga är avsnittet om SQL som börjar på s. 373.

I databaskursen används sedan flera år tillbaka Elmasri, R. & Navathe, S.B. (2004) *Fundamentals of Database Systems*. Den finns att låna på biblioteket.

På adressen http://www.databasteknik.se/webbkursen/ finns en webbkurs om databaser. Det är främst det sista avsnittet i del 1 som är intressant.

6. Frågor att fundera över

Med de erfarenheter ni har av programmering så här långt, hur svårt känns det att formulera frågor i SQL? Vad tycker ni om den lilla del av SQL som ni har fått stifta bekantskap med?

Med *data mining* avses att hitta mönster i stora mängder data. Såväl nätbutiker som vanliga affärer samlar idag in information om vem som köper vilka produkter. På detta sätt kan affären planera sina inköp bättre och vara förberedd på hög efterfrågan. En annan anledning kan vara att man vill kunna göra riktade erbjudanden. Kan ni se några tekniska problem med den här typen av tips om produkter och tjänster? Under vilka förhållanden kan tipsen bli missledande? Vad kan man göra för att motverka detta?

Flera dagligvaruhandelskedjor erbjuder kundkort till vilket rabatter och andra erbjudanden är knutna. Samtidigt gör det att affären kan kontrollera exakt vilka varor man köper. Hur skulle det kännas att veta att alla dina inköp kontrolleras? Finns det sammanhang där det är godtagbart? Var går gränsen?