Databaser, design och programmering

Databasspråket SQL - onLine.

Innehåll: Viktiga kommandon och konstruktioner i SQL, både DDL och DML.

SQL är baserat på Relationsalgebra, den matematik-gren som arbetar med mängder och relationer. För varje SQL-kommando som används i denna text visas också hur det skrivs i relationsalgebra.

Utgångspunkt: en databas om ett varuhus (The Jonson Brothers Company).

Förbered installation av databasen:

1. Öppna en webläsare och gå till kurshemsidan och hitta (under "LABBKURS"), menyalternativet "Komma igång med databasklienten" (länken är www.ida.liu.se/~732G16/tasks/lab1.sv.shtml
2. Följ instruktionerna under rubriken "Starta mysql-klienten". Missa inte texten efter tabellen.
3. Följ instruktionerna under rubriken "Skapa databasen inför föreläsning och labb".

Klart! Nu har du installerat JohnssonBrothers' databas.

Hur undersöka en okänd databas?

- **`show tables;`**
- **`describe`** `<tabellnamn>;`
- **`show create table`** `<tabellnamn>;`

Först datamanipulering med SQL (SQL som DML)

Sammanfattning relationsalgebra:

Urval (σ) - plocka ut tupler (rader) som uppfyller ett visst villkor.

$$\sigma_{\text{villkor}}(\text{relation})$$

Projektion, (π) - plocka ut attributvärden (kolumner).

$$\pi_{\text{kolumner}}(\text{relation})$$

Union, (U) - lägg ihop unionskompatibla relationer.

$$\text{relation } \cup \text{ relation}$$

Kartesisk produkt (x) - para ihop alla tupler ur den ena med alla tupler ur den andra relationen.

$$\text{relation } X \text{ relation}$$

Sammansättning (join, ⋈) - para ihop alla tupler ur den ena relationen med alla tupler ur den andra och välj ut de som uppfyller villkoret.

$$\text{relation} \bowtie_{\text{villkor}} \text{relation}$$

Sammansättning är samma sak som:

$$\sigma_{\text{villkor}}(\text{relation} X \text{ relation})$$

Notera att resultatet av relationsalgebra-operationer alltid är en ny relation.

1. Relationsalgebrans urval (σ) och projektion (π)

I SQL finns ett enda kommando som kombinerar urval och projektion:

```
SELECT kolumner
FROM relation (eller kombination av relationer)
WHERE villkor;
```

motsvarar:

$$\pi_{\text{kolumner}} (\sigma_{\text{villkor}} (\text{relation}))$$

Mål: Lista namnet på alla anställda som tjänar över 9000 och anställdes efter 1968?

som relationsalgebra:

$$\pi_{\text{NAME}} (\sigma_{\text{SALARY}>9000 \wedge \text{Startyear} > 1968} (\text{Employee}))$$

```
select name from employee where salary>9000 and startyear>1968;
```

Resultatet är en tabell. **Kan innehålla dubletter!**

Välja flera kolumner?
Mål: Vad heter de och vilket id-nummer har de?

```
select pk_employee, name from employee where salary>9000 and startyear>1968;
```

Ordningen odefinierad. Råkar vara ordnad på nyckeln. Om vi vill ha en definierad ordning, då? Sortera på lönen, till exempel?

```
select pk_employee, name, salary
from employee where salary>9000 and
startyear>1968 order by salary;
```

Man kan också visa en hel tabell:

```
Select * from employee;
```

Om man vet att tabellen har många rader men vill se några;

```
Select * from employee limit 5;
```

Uppgift:

Utforska tabellerna Employee, Sale, Debit, och Item med hjälp av select-kommandot. Plocka ut enstaka rader, enstaka kolumner och enstaka celler.

2. Olika sätt att kombinera information från flera relationer: Relationsalgebrans union, kartesisk produkt och sammansättning.

Det är så sällan det finns unionskompatibla relationer i en databas att unions-kommandot i sql är konstruerat för att ta resultatet från två select-satser och göra union på dem, dvs kan ej ta två tabeller direkt.

SQL-kommandot UNION:

```
select ... from ... where ... UNION
select ... from ... where
```

Mål: Lista namnen på alla saker som bolaget hanterar (items och parts)

```
select name from item union select name from parts;
```

Not: Gör sann union. Behöver inga parenteser.

Operationen Kartesisk produkt:

Görs i SQL som en del av SELECT-kommandot:

```
SELECT * FROM tabellA, tabellB;
```

Motsvarar:
tabellA × tabellB

Mål: Vilka produkter ingick i de olika inköpsposterna som de anställda hade handlat?

Skrivs i relationsalgebra som:

$$\text{Sale} \times \text{Debit}$$

och i SQL som:

```
select * from debit, sale;
```

ger alla rader i Sale kombinerade med alla rader i Debit.

För att sedan hitta de rader som är "äkta" kan vi göra urval av de rader där sale.debit är lika med debit.pk_debit.

I relationsalgebra skrivs det:

$$\sigma_{\text{Sale.Debit=Debit.pk\_d}}(\text{Sale} \times \text{Debit})$$

och i SQL:

```
select debit.pk_debit, debit.employee,
sale.item, sale.quantity
from debit, sale
where debit.pk_debit=sale.debit;
```

Uppgift:

1. Prova detta kommando.
2. Gör en liknande fråga där du söker ut vilka saker (i Items) som säljs på avdelningen som heter "Toys" (i Dept).

Relationsalgebraoperationen Join (sammansättning)?

Kartesisk produkt följd av urval är just JOIN:

$$\text{tabellA} \bowtie_{\text{villkor}} \text{tabellB}$$

Görs som en del av SELECT-kommandot:

```
select ... from tabellA join tabellB on
villkor
```

Mål: Vilka produkter ingick i de olika inköpsposterna som de anställda hade handlat?

I relationsalgebra:

$$\text{Sale} \bowtie_{\text{sale.Debit=Debit.pk\_d}} \text{Debit}$$

Skrivs i SQLs select-sats, istället för kartesisk produkt:

```
select *
from debit join sale
on debit.pk_debit=sale.debit;
```

Uppgift:
Gör en liknande fråga där du söker ut vilka saker (i Items) som säljs på avdelningen som heter "Toys" (i Dept).

Men: om jag vill söka ut vilka varor en viss anställd köpt, behöver jag använda join eller kartesisik produkt då? Kan jag inte bara göra flera frågor?

Mål: Hitta de Items som köpts av anställd nr 35.
Hitta dennes debitpost:

```
select pk_debit
from debit
where employee=35;
```

```
(svaret blir nr 100586 och 100593)
```

Utifrån dessa nummer, hitta artikelnr i Sale

```
select item
from sale
where debit in (100586, 100593);
```

Men värdena vi använder är ju resultatet från en select-sats, kan man slå ihop dem? Jodå:

```
select item from sale
where debit in (select pk_debit from debit
where employee=35);
```

In= finns i mängden (tabellen)

Detta kallas att **Nästla** (eller Kapsla) satser. Den engelska benämningen är Subquery.

Uppgift:
Prova att söka ut vilka items som säljs på dept med namnet "Jewelry" först steg för steg och sedan nästlat.

3. Kombinera flera tabeller, exempel:

Vi kan alltså sätta ihop tabeller antingen genom kartesisk produkt, join eller nästling. Alla formerna kan kombinera fler än två tabeller.

Mål: Vilka saker (item) säljs på de avdelningar (dept) som styrs av Raveen Lemont? I tabellen item finns referensattributet "dept", som är nyckel i dept-tabellen, som i sin tur har ett referensattribut, manager, som är nyckel i employee-tabellen, där vi hittar namnet.

Manuellt, steg för steg:

```
select pk_employee from employee where name
= 'Raveen, Lemont';
select pk_dept from dept where manager=37;
select item.name from item where dept=1 or
dept=65;
```

Nästlat:

```
select item.name from item
where item.dept in
(select pk_dept from dept
  where dept.manager in
    (select pk_employee from employee
     where employee.name ='Raveen, Lemont'));
```

Med kartesisk produkt:

```
select item.name from item, dept, employee
where item.dept=dept.pk_dept and
dept.manager = employee.pk_employee and
employee.name = 'Raveen, Lemont';
```

Med Join:

```
select item.name from (item join dept on
item.dept=dept.pk_dept) join employee on
dept.manager = employee.pk_employee
where employee.name = 'Raveen, Lemont';
```

OBS: En sats som sätter ihop tabeller med join (sammansättning) går **alltid** att skriva som en kartesisk produkt med join-villkoren i where-delen (jfr definitionen att en join är ekvivalent med ett urval ur en kartesisk produkt).

En nästlad sats går **ofta** att skriva om till kartesisk produkt eller join, men inte alltid. En kartesisk produkt eller join går **ofta** att skriva om på nästlad form, men inte alltid.

Not: Nu har vi sett "and" och "or" i where. Andra operationer i Where-villkoret är:
Jämförelser med siffror, strängar, =, !=, <, >, >=, <=, and, or, och i nästlade satser med in, not in, exists mm.

Dessutom jämförelser med delar av strängar:

```
select pk_employee from employee
where name like '%Lemont%';
```

_ matchar ett tecken
% matchar noll, ett eller flera tecken

4. Mer Join:

Join sätter ihop rader ur två tabeller sådana att ett villkor är uppfyllt. Det är då inte säkert att alla rader ur tabellerna kommer med, om det inte finns någon rad som "matchar".

Det finns varianter på Join sådana att man tar med rader från tabellerna även om villkoret inte är uppfyllt för raden. Det kallas yttre join och finns i tre varianter:

`tabellA left outer join tabellB on villkor`
tar med alla rader ur tabellA även om de inte uppfyller villkoret.

`tabellA right outer join tabellB on villkor`
tar med alla rader ur tabellB även om de inte uppfyller villkoret.

`tabellA full outer join tabellB on villkor`
tar med alla rader ur båda tabellerna även om de inte uppfyller villkoret.

Notera att MariaDB just nu (2023) inte implementerar full outer join.

Exempel: Lista alla anställda (employees), och OM de har en post i debit-tabellen, lista även IDnumret på den debit-posten (pk_debit):

```
select employee.name, debit.pk_debit
from employee left outer join debit
on employee.pk_employee=debit.employee;
```

Uppgift:
Lista namnet på alla avdelningar (dept), och om de säljer någon vara (item), namnet på den varan.

5. Ytterligare användbara konstruktioner och kommandon i SQL (finns ej i relationsalgebra) :

Jämförelser med sig själv.

Mål: Carol Smythe vill veta vilka som tjänar mer än hon gör. Lista alla anställda med lön över hennes. Göra det i två steg går bra:

```
select salary from employee where
name='Smythe, Carol';
select name, salary from employee where
salary >9050;
```

Går att göra den nästlad:

```
select name, salary from employee
where salary > (select salary from employee
where name='Smythe, Carol');
```

Men alla nästlade satser går ju platta ut (nästan)?

```
select other.name, other.salary
from employee as carol, employee as other
where other.salary > carol.salary and
carol.name='Smythe, Carol';
```

Dessa nya namn kallas **tabellalias** eller **tupelvariabler**.

(Uppgift?)

6. Enkla beräkningar

görs i select-satsen. Dels enkla matematiska beräkningar på kolumnvärden:

Mål: Räkna ut kostnaderna för sakerna i sale (behöver item också):

```
select sale.debit, sale.item, sale.quantity,
item.price, sale.quantity*item.price
from sale join item on
sale.item=item.pk_item;
```

Lite otydlig resultat-tabell, skulle ju gärna vilja ha en priskolumn istället för uträkningen. Byta namn på kolumnerna:

```
select sale.debit as Ordernr, sale.item as
Artikel, sale.quantity as Antal, item.price
as Styckpris, sale.quantity*item.price
as Pris from sale join item on
sale.item=item.pk_item;
```

Uppgift:
Det är rea i butiken, 20% på allt! Skriv ut alla varor i Item i en ny prislista (namn, ordinarie pris och rabatterat pris).

Aggregeringsfunktioner: funktioner som gör beräkningar över ett antal rader i tabellen.

Mål: Vad kostar den dyraste respektive billigaste prylen (i item), medel?

```
select max(price), min(price), avg(price)
from item;
```

Finns också bl.a. sum, count, stdev, variance mm

Uppgift:
Plocka ut den högsta lönen ur employee.

Utvidgning: Hur hittar vi namn och id-nummer för den som har högst lön? (detta är ett exempel på när man måste nästla, detta går inte skriva med join)

Beräkningar över del av tabellen:
Aggregering med group by

Mål: Om man vill veta vad varje debitpost (en order?) i debit summerar till? Vi hade ju summan för varje post...

```
select sale.debit as Ordernr,
sum(sale.quantity*item.price)
as Pris from sale join item on
sale.item=item.pk_item
group by sale.debit;
```

(**OBS** att **group by** bara kan användas tillsammans med aggregeringsfunktioner!)

Man kan också göra urval av vilka rader som tas med:

Mål: vi vill beräkna varje chefs lönekostnad för sina underlydande...

```
select manager, sum(salary) from employee
group by manager;
```

Mål: Vi vill beräkna löneskostnaden för de chefer som har mer än 2 underlydande:

```
select manager, sum(salary) from employee
group by manager
having count(pk_employee)>2;
```

(**OBS** att **having** bara kan användas tillsammans med group by.)

Jämförelser med aggregeringar:

Mål: Carol Smythe vill veta vilka som har högre lön än medel:
Går att göra steg för steg:

```
select avg(salary) from employee;
select name, salary from employee where
salary>11868;
```

Men om vi vill sätta ihop den till ett uttryck? Nästla:

```
select salary, name from employee where
salary> (select avg(salary) from employee);
```

Alla nästlade satser går väl platta ut? Eller?

```
select salary, name from employee where
salary>avg(salary);  ???
```

Inte denna! Aggregeringsfunktionerna går bara använda i attributlistan!

Definition av SELECT (se nedan för förklaring)

```
SELECT attribut-lista
FROM tabell-lista
[WHERE villkor]
[GROUP BY attribut-lista
    [HAVING villkor]]
[ORDER BY attribut-lista]
```

`Attribut-lista` behöver unika attributnamn (ange vid behov
tabellnamnet) och kan innehålla beräkningar och aggregeringsoperationer och kan ges nya kolumnnamn (as).
`Tabell-lista` består av en lista av tabeller, både vanliga och temporära tabeller genererade av nästlade operationer (`select`) (då med `as nyttnamn`), eller sammansättningar (join) av sådana tabeller och kan också innehålla mängdoperationer på sådana tabeller.
`villkor` i WHERE byggs upp av attributnamn och operatorer och kan också innehålla temporära tabeller (eller enkla värden) genererade av nästlade operationer (`select`).

7. Manipulering av data

Man kan också lägga in och ta bort värden i tabellerna.
Sätta in helt nya rader:

```
Insert into tabell values (tupel)
```

Mål: lägg in en ny vara (i Item):

```
insert into item
values (305, 'Jacket', 60, 750, 35, 33);
```

(måste anges i rätt ordning!)

Om man inte orkar ange alla, och har defaultvärden (dept och supplier har inte default):

```
insert into item (dept, pk_item, supplier)
values (10, 306,33);
```

Ta bort data (hela rader):

```
Delete from tabell where villkor
```

Mål: Ta bort den där oifyllda:

```
delete from item where pk_item=306;
```

OBS: where fungerar som where i select.

Referensintegritet kommer att förhindra att vissa rader tas bort. Exempel: Om en sale-post visar att ett visst item har köpts, så kommer inte det item-et att gå att ta bort.

Ändra enskilda värden (celler):

```
Update tabell set attr=värde where villkor
```

Mål: nytt pris på den där jackan, och inte så många i lager:

```
update item
set price=500, qoh=20
where pk_item=305;
```

Det är rea på hela varuhuset: 10% på allt, lagra det nya priset i databasen!

```
update item set price=price*0.9;
```

(OBSERVERA! Om ni testar dessa kommandon, se till att återställa innehållet innan ni börjar Laboration 2).

SQL som DDL

Skapa tabeller:
```
create table tabell (attribut typ
egenskaper)
```

Mål: en tabell för att lagra vilken avdelning anställda jobbar på:

```
create table works_in
(employee int not null,
dept int,
primary key (employee)) Engine=innoDB;
```

Typen kan vara t.ex. char(antal tecken), varchar(maxantal tecken), int mm. Man kan också indikera vad som är primärnyckel eller om något värde måste vara unikt, inte får var null osv.

Man kan också skapa en tabell som matchar (och innehåller) resultatet av en select-sats:
```
create table managers as
select distinct mgr.pk_employee, mgr.name
from employee as emp, employee as mgr
where emp.manager=mgr.pk_employee;
```

Ändra befintlig tabell
```
Alter table tabell add attribut/egenskap ...
```

Mål: vi vill lagra avdelningen i employee istället för i separat tabell:
```
alter table employee
add dept int;
```

Kan också göra t.ex drop, modify samt add constraints (för referensattributen) i Alter table:

Works_in.employee ska ju vara en anställd (constraint-et kan också läggas in create table-satsen). Constraints bör ges namn för att kunna manipulera dem utan att ta bort tabellen):

```
alter table works_in add constraint
fk_works_in_employee
    foreign key (employee) references
employee (pk_employee);
```

Ta bort tabeller:

```
drop table works_in;
```

Uppgift

Vyer

En Vy är en abstrakt tabell. En relation som inte finns egentligen. Vyer används för att enkelt ge olika användargrupper olika urval av databasen. Man kan säga att de får ett eget databasschema för sina specifika uppgifter.

```
Create view vynamn as select-kommando
```

För att skapa en vy som visar vad ordrarna i debit innehåller och kostar (som tidigare) skriver man

```
create view debcont (OrderNo, ArtNo, qty,
price, tot_price) as select sale.debit,
sale.item, sale.quantity, item.price,
sale.quantity*item.price from sale join item
on sale.item=item.pk_item;
```

Vyn debcont skapas inte fysiskt utan den konstrueras vid användning.

Vyer kan tas bort med

```
Drop view debcont
```

Observera att kommandot show tables listar även vyer, men vyer kan inte tas bort med drop table.