Webprogrammering och databaser Fö 5

Databasspråket SQL - onLine.

Innehåll: Viktiga kommandon och konstruktioner i SQL, både DDL och DML. Utgångspunkt: en databas om ett varuhus (The Jonson Brothers Company.

Installation

Förbered installation av databasen:

1. Öppna en webläsare (t.ex. Firefox) och gå till kurshemsidan, Labbar (översikten) **www.ida.liu.se/~729G28/labbar/**
2. Hitta länken till Johnason Brothers database. Ladda ner den länkade zip-filen och packa upp filerna (CompanySchema.sql och CompanyData.sql) till en lämplig mapp.
3. Sök upp det ebrev som du fått som har rubriken "Database created". Det skickades i augusti.
4. I det brevet, hitta lösenordet till ditt databaskonto (=din databas).

Starta en databasklient och installera genom att:

1. Starta en Terminal, t.ex. MATE terminal eller liknande.
2. Gå till mappen där du lade filerna med hjälp av kommandot **cd** följt av lämplig sökväg.
3. Starta mysql's kommandoradstolk och koppla upp den mot universitetets databas genom att ge nedanstående kommando (byt ut liuid mot ditt eget liuid):
   mysql -h mariadb.edu.liu.se -u liuid -p

4. mysql svarar med att fråga efter lösenordet, mata in det som stod i brevet.
5. mysql svarar med en lång harang. Du kan nu mata in textkommandon.
6. Ange vilken databas du ska arbeta med (du har bara en, som heter som ditt liuid) genom att ge följande kommando till mysql:
   use liuid
7. Installera nu databasens schema genom att ge följande kommando:
   source CompanySchema.sql
8. mysql svarar med en lång rad "Query OK" (skrolla upp och kolla så att inga fel inträffat).
9. Installera databasens innehåll genom följande kommando:
   source CompanyData.sql
10. mysql svarar med ytterligare en lång rad "Query OK" (kolla igen)

Klart! Nu har du installerat JohnssonBrothers' databas.

Hur undersöka en okänd databas?

- `show tables;`
- `describe employee;`
- `show create table employee;`

Uppgift:
Använd `show tables` för att se vilka tabeller som finns. Använd sedan `show create table` för att utforska några tabeller.

(Notera primary key, constraint, foreign key, on delete samt engine).

Först datamanipulering med SQL (SQL som DML)

Söka ut information

Vilka anställda har vi egentligen?
Kommandot SELECT:
Enklaste formen (* för alla attribut/kolumner).

```
select * from employee;
```

Alla kolumner. Bara vissa: rada upp kolumnnamn:

```
select name, salary from employee;
```

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 order by salary;
```

Om vi inte vill ha med alla rader i en tabell?

Ex: alla anställda som har lön över 10000:

```
select * from employee where salary>10000;
```

Går också kombinera villkor i "where"-delen:

```
select * from employee
where salary>10000 and startyear >1960;
```

Uppgift:

Utforska innehållet i tabellerna Employee, Sale, Debit, Item och Dept med hjälp av Select. Plocka ut enstaka rader, enstaka kolumner och enstaka celler.
Operationer i Where kan man göra jämförelser med siffror, strängar, =, !=, <, >, >=, <=, and, or, is null och is not null.

Dessutom kan jämförelser göras med delar av strängar:

```
select pk_employee, name from employee
where name like '%Ross%';
```

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

Uppgift:
Prova att söka ut alla anställda som heter Paul.

Om man vill ha information som finns i flera tabeller?
Ta reda på vilka items som säljs på Leksaksavdelningen (Toys) (hitta id-numren för Toys, lista items som säljs på den):

```
select pk_dept from dept where name="Toys";
```

(svar: 49)

```
select * from item where dept =49;
```

Kan göras som en sammansatt (nästlad) fråga, dvs använd resultatet från en fråga som komponent i en annan fråga:

```
select * from item
where dept=(select pk_dept
    from dept
    where name="Toys");
```

Uppgift:
Anställda kan handla på kredit på företaget, när de gör det skapas en rad i tabellen debit för varje köp och rader i tabellen Sale som visar vilka varor (items) som inköpts.
Prova att söka ut ID-numret för de varor som inköpts av den anställde som har ID-nummer 157, först genom att söka ut numret på debit-posten och sedan söka ut ur tabellen Sale där detta debit-

nummer finns, och sedan genom att kombinera bägge kommandona till ett som ovan.

Uppgift:
Vad händer om det inre (nästlade) kommandot returnerar flera värden? Anställd nr 35 har handlat flera gånger. Prova samma kommando för denne.

Jämförelser i where kan göras med mängdjämförelse: IN. Andra mängdoperatorer att använda i where: not in, exists, not exists.

Kombinera tabeller i From (Kartesisk produkt):
Varje rad ur en tabell kombineras med varje rad ur den andra: Notera: Attributet name finns i båda tabellerna. För att skilja dem åt används s.k. punktnotation: tabellnamn.attributnamn:

```
select item.pk_item, item.name, item.dept,
    dept.pk_dept, dept.name
from item, dept;
```

Men detta är ju bara rappakalja. Använd den främmande nyckeln! Vilka prylar säljs på vilken avdelning? Om item.dept=dept.pk_dept?

```
select item.pk_item, item.name, item.dept,
    dept.pk_dept, dept.name
from item, dept
where item.dept=dept.pk_dept;
```

Och så begränsa urvalet till avdelningen "Toys":

```
select item.pk_item, item.name, item.dept,
    dept.pk_dept, dept.name
from item, dept
```

```
where item.dept=dept.pk_dept and
    dept.name="Toys";
```

Uppgift:

1. Prova ovanstående kommando.
2. Gör en liknande fråga där du söker ut vilka Sale-poster som hör till Debit-posten för den anställde med ID 157.

Detta sätt blir lätt komplext då många tabeller kombineras och många andra villkor används.

Kombinera tabeller med Join, Sammansättning

Sammansättningen av två tabeller (på samma sätt som kartesisik produkt) sådan att varje resulterande rad uppfyller ett villkor (vanligen det villkor som beskriver den främmande nyckeln mellan dem):

```
item join dept on item.dept=dept.pk_dept
```

Resultatet är en tabell och används i FROM:

```
select item.pk_item, item.name, item.dept,
    dept.pk_dept, dept.name
from item join dept on
    item.dept=dept.pk_dept;
```

Eller med ytterligare villkor (bara avdelningen Toys)

```
select item.pk_item, item.name, item.dept,
    dept.pk_dept, dept.name
from item join dept on
    item.dept=dept.pk_dept
```

```
where dept.name="Toys";
```

Uppgift:
Prova detta sätt att skriva genom att söka ut alla sale-poster som ingår i debit-poster som hör till en anställd med ID=157

OBS: Alla uttryck med kartesisk produkt går att skriva med join, och omvänt.

Varianter på sammansättning (outer join):

Lista employees och deras debit-poster, dvs om de har handlat:

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

Om vi vill lista alla employees och, ifall de har en debit-post, numret på den... Employees utan debit-post uppfyller ju inte join-villkoret.

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

Yttre join: Left (outer) join, right (outer) join samt full (outer join).

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

Alltså; kombinera tabeller med varandra

Kombinera information ur flera tabeller kan alltså göras på tre olika sätt: nästlad selectsats, med join och med flera tabeller i FROM (kartesisk produkt). De är i många fall utbytbara, men inte i alla.

Ex: Lista alla items som säljs på den avdelning som styrs av Lemont Raveen? (i tabellen employee, hitta nyckeln pk_employee, vilken eller vilka avdelningar har denne som manager -dept - vilka saker säljs på dessa avdelningar - item-)
Kräver tre tabeller!
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 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';
```

Med kartesisk produkt i FROM:

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

Notera: Join och kartesisk produkt är ekvivalenta, dvs alla frågor som går att skriva på den ena formen går alltid att skriva som den andra.
Vissa frågor som skrivs på nästlad form kan också skrivas på join-form och som kartesisk produkt, och omvänt. MEN inte alla!

Repetition från föregående föreläsning:

Att söka ut information:

```
select [attrbutliststa]
from [tabeller - antingen join eller
    kartesisk produkt]
where [villkor, som kan innehålla ny select-
    sats] ;
```

Exempel: Lista varor: välj alla kolumner för de rader där priset är över 8000:

```
Select * from item where price > 1000;
```

Exempel: Lista namn, pris och avdelning för de varor som säljs på avdelningen "Bargain"
Först steg för steg:

```
Select pk_dept from dept where
    name="Bargain";
Select name, price, dept from item where
    dept=1;
```

Sedan nästlat:

```
Select name, price, dept from item where
    dept=(select pk_dept from dept where
    name="Bargain");
```

Och med join:

```
Select item.name, item.price, item.dept
from item join dept on
    item.dept=dept.pk_dept
where dept.name="Bargain";
```

Men: Jämförelser med sig själv.

Carol Smythe vill veta vilka som tjänar mer än hon gör. Lista alla anställda med lön över hennes. Två steg går bra:

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

Sedan nästlad:

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

Men alla (nästan) nästlade satser går ju platta ut, dvs skriva med kartesisk produkt eller join?

Kan vi skriva:

```
select name, salary from employee where
    salary < salary (carols?)?
```

```
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:

Plocka ut namnet/n på den/de varor som har färre antal i lager än det finns jackor (Jacket) i lager.

Beräkningar i SQL

Enkla beräkningar och alias.

Beräkningar kan göras i select-satsen. Dels enkla matematiska beräkningar på kolumnvärden:

Ex: Räkna ut kostnaderna för posterna 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 (med rubrikerna namn, ordinarie_pris och rabatterat_pris).

Aggregeringsfunktioner:

funktioner som gör beräkningar över ett antal rader i tabellen.

Ex: 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. Beräknar resultatet på alla rader i tabellen (Notera här betydelsen av nullvärden).

OBS: att dessa funktioner bara kan användas i attributlistan (dvs efter ordet SELECT och före FROM).

Uppgift:
Plocka ut den högsta lönen ur employee.

Aggregering över en del av tabellen,

med kommandot group by:

Om man vill veta vad varje order i debit summerar till? Vi hade ju summan för varje artikel...

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

Group by grupperar alltså raderna i tabellen (den i FROM) och gör så att den aggregeringsfunktion som finns i SELECT-delen utförs på varje delgrupp.
(OBS att group by bara kan användas tillsammans med aggregeringsfunktioner!)

Man kan också göra urval av vilka rader som tas med:
Om vi vill beräkna varje chefs lönekostnad för sina underlydande...

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

Om vi bara vill titta på dem 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.)

Andra användningar av aggregering:

Vi vill veta vilka anställda som har högre lön än medel:

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

Men om vi vill sätta ihop den till ett uttryck?

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

Hur blir det om man försöker "platta ut" denna nästlade sats med join eller kartesisk produkt?

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

Denna typ av nästlad sats (då aggregeringsfunktion ska användas för ett urval) kan inte "plattas ut", dvs skrivas med hjälp av join eller kartesisk produkt. Detta kan ställa till problem då vi vill använda kombinerad information:

Lista avdelning och item (namn och nummer på båda) som säljer den dyraste varan.
Steg för steg:

```
Select max(price) from item;
```

```
select dept.pk_dept, dept.name,
    item.pk_item, item.name
from dept join item on
    item.dept=dept.pk_dept
where item.price=4995;
```

I ett enda kommando:

```
select dept.pk_dept, dept.name,
    item.pk_item, item.name
from dept join item on
    item.dept=dept.pk_dept
where item.price=(select max(price) from
    item);
```

Vi måste alltså göra join med item fastän vi också har item i den nästlade satsen. Att göra detta med kartesisk produkt blir mycket lätt krångligt:

```
select dept.pk_dept, dept.name,
    item.pk_item, item.name
```

```
from dept, item
where item.dept=dept.pk_dept
and item.price=(select max(price) from
    item);
```

Definition av SELECT

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

Attr-lista behöver unika attributnamn (använd vid behov punktnotation) och kan innehålla beräkningar och aggregeringsfunktioner 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`).

Datamanipulering: ändring, insättning, borttagning

Man kan också lägga in och ta bort värden i tabellerna.
En ny produkt:

```
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, medan item har null för namn, pris och antal):

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

Uppgift:
Lägg in dig själv i tabellen employee

Ta bort data (hela rader):

Kommandot Delete from
Bort med den där oifyllda varan:

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

where fungerar som where i select.

Referensintegritet:

Notera att om vi försöker ta bort något som pekas ut av en främmande nyckel i en rad i en annan tabell kan det gå det fel.

Uppgift:
prova att ta bort Michael Evans (ID 35). Vad händer?

Om vi t.ex. försöker ta bort en anställd som har en debit-post (främmande nyckel från debit.employee till

employee.pk_employee). Michael Evans har till och med två poster. Om han försvinner finns ingen som ska betala...

Främmande nycklar kan definieras med information om vad som ska göras vid borttagning:

Show create table employee;

T.ex: den främmande nyckeln från manager i employee till sig själv (employee.manger till employee.pk_employee) har on delete set null, ta bort en employee som är chef, t.ex. nr 10, Stanley Ross, som är chef för 215 Joanne Colins).

Uppgift:

1. Kolla vem som är chef för Joanne Collins (IDnr 215)
2. Ta bort den personen.
3. Kolla igen på Joanne, vem är chef nu?

```
(delete from employee where pk_employee=10;
select * from employee;)
```

Andra uppdateringar

Ändra enskilda värden (celler): 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 alla items med nyckelvärden från 101 till 120

```
update item set price=price*0.9 where
    pk_item between 101 and 120;
```

Uppgift:
Alla får löneförhöjning; dubbla lönen för alla anställda!

SQL som Databeskrivnings-språk; DDL

Skapa tabeller,
Create table [tabellnamn] (lista av attribut-datatyp-par):Engine=innoDB;

t.ex. en för att lagra vilken avdelning anställda jobbar på:

```
create table works_in
(employee int not null Primary key,
dept int);
```

Typen kan vara t.ex. char(antal tecken), varchar(maxantal tecken), int mm. Man kan också direkt 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 (ger bara kolumner och datatyper, inga nycklar el. dyl):

```
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

Ändra befintlig tabell, t.ex. utnämna attributet pk_employee i managers till nyckel:

```
alter table managers add primary key
    (pk_employee);
```

Eller t.ex. vi vill lagra avdelningen i employee istället för i separat tabell:

```
alter table employee add dept int;
```

Det nya attributet får null för alla gamla rader.

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

Lägg t.ex. till främmande nyckel (referenskontroll) på att dept i works-in refererar till tabellen dept (pk_dept):

```
alter table works_in add constraint
    fk_works_dept
    foreign key(dept)
    references dept (pk_dept);
```

Notera att det verkar finnas en bugg som gör att om man ändrar en nyckel som det finns foreign key constraint till (t.ex. genom att byta datatyp på nyckeln) kan man få problem med sina constraints.

Ta bort tabeller:

```
drop table works_in;
```

Vyer

En Vy är en abstrakt tabell. En relation som inte finns egentligen. Vyer kan användas för att enkelt ge olika användargrupper olika urval av databasen. Man kan säga att de får ett eget databasschema för sina specifika uppgifter. Vyer kan också användas för att förenkla (dela upp) komplexa urval och beräkningar.
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.

Uppgift:

Skapa en vy vara som består av pk_item, varans namn och pris.