EDA216-Sammanfattning

Måns Ansgariusson, C14

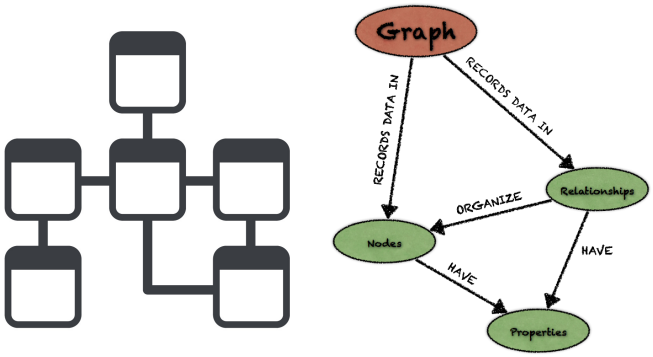
2016-05-07

1 Vad är en databas?

En databas är mängd data som är konstant och strukturerad. En databas drivs av ett system som kallas för DBMS, Database managment system. DBMS's uppgift är att ta hand om de, i det flesta fall, enorma kvantiteterna av data som är lagrad i databasen. DBMS tar hand om, tar bort och utför instruktioner från en användare. Instruktionerna som dbms ska utföra skrivs i ett "query-language" ex. MySQL. [1]

Vad finns det för olika databaser?

Det finns,för tillfället, två stora sorters databaser: relationsdatabaser och graf-databaser.



(a) Relationsdatabas

(b) Grafdatabas

Figure 1: Två sorters databaser

Relationsdatabas är uppbyggd av tabeller, otroligt många tabeller. Dessa tabeller, om de skapats av någon som vet vad de sysslar med, har en relation mellan sig vilket låter en användare att hämta information från den.
En Grafdatabas består av Noder istället som har information kopplad till sig.

[1] Query - förfrågan om information

2 Query's

Vad är en Query?

En query är en begäran av information till en databas. Dessa är skrivna på sådant sätt att DBMS kan ta hand om dessa och skicka tillbaka den begärda informationen. Ett exempel:

Select * from Students;

Denna simpla queryn kommer att be databasen ta fram all information i tabellen Students. Vi bryter ner queryn:
Select är början på en fråga där du begär information.
* betyder att jag vill ha all information eller alla tuples.
from definjerar vart informationen finns.
Students är vårat fiktiva tabellen.
Om vi vill ha specifik information från ett tabellen så kan vi skriva så här:

Select LastName,PersNbr from Students where FirstName = "John";

Här frågar vi databasen endast om kolumnerna LastName och PersNbr i tabellen Students där FirstName är John.
Where lägger till begränsningar för en query.
Generellt fall:

Select <Vad du vill veta>from <Var informationen finns >where <Under följande omständigheter>;

Joins

En join är en operation som returnerar en tabell som skapats av två "permanenta" tabeller.

Inner Join

Ex:
Tänk dig att vi har två permanenta tabeller:Students som innehåller namnet och personnumret och StudentContactINfo som innehåller e-mail adress och telefonnummer. Följande query kommer då returnera en tabell som innehåller all information som är gemensamt i de ursprungliga tabellerna.

Select * from students join StudentContactINfo;

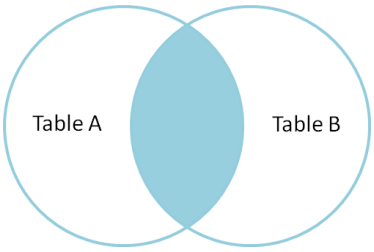


Figure 2: InnerJoin mellan tabellerna A(Student) och B(StudentContactINfo)

Outer Join

En outer join är en operation som används för att returnera alla värden i tabellen som ansluter till den andra tabellen med de värden där tabellerna går ihop, se figur 3. Ex.

En left outer join tabellen på vänster sida och returnerar en tabell med alla

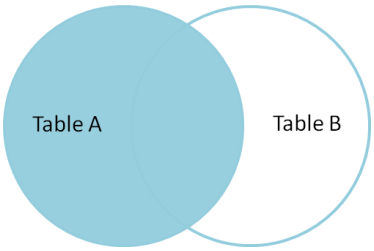


Figure 3: en left outer join

dess värden och de tupels i den högra tabellen som har en gemensam nämnare som specificeras av queryn.
Generell left outer join:

```
Select * from <Tabell A> left outer join <Tabell B>
where A.<Variabelnamn>=B.<Variabelnamn>;
```

Skillnaden mellan en inner join och en outer join är att en outer join kan returnerar alla värden i en av de två tabellerna (om inget annat skrivs i where satsen) och de värden där tabellerna går ihop.

För att försöka förklara bättre:

```
Tabellen person:
 +------------+--------------+------+
    | name       | phone        | pid  |
    +------------+--------------+------+
    | Mr Brown   | 01225 708225 |    1 |
    | Miss Smith | 01225 899360 |    2 |
    | Mr Pullen  | 01380 724040 |    3 |
    +------------+--------------+------+
    Tabellen property:
     +------+------+----------------------+
    | pid  | spid | selling              |
    +------+------+----------------------+
    |    1 |    1 | Old House Farm       |
    |    3 |    2 | The Willows          |
    |    3 |    3 | Tall Trees           |
    |    3 |    4 | The Melksham Florist |
    |    4 |    5 | Dun Roamin           |
    +------+------+----------------------+

    queryn: select name, phone, selling
    from people left join property
    on people.pid = property.pid;
    kommer returnera:
    +------------+--------------+----------------------+
    | name       | phone        | selling              |
    +------------+--------------+----------------------+
    | Mr Brown   | 01225 708225 | Old House Farm       |
    | Miss Smith | 01225 899360 | NULL    |
    | Mr Pullen  | 01380 724040 | The Willows          |
    | Mr Pullen  | 01380 724040 | Tall Trees           |
    | Mr Pullen  | 01380 724040 | The Melksham Florist |
    +------------+--------------+----------------------+
```

Om vi kollar på den returnerade tabellen så ser vi att Miss Smith inte säljer något hus. Denna information hade vi gått miste om ifall vi gjort en inner join då vi endast hade fått de personer som säljer ett hus.
En **right outer join** är exakt samma sak som en left outer join bara att en right outer join behandlar den högra tabellen som den ”primära” tabellen i queryn.

En **full outer join** är en outer join som slår ihop de båda tabellerna och sorterar informationen efter det specificerade värdet i where satsen. All information behålls i båda tabellerna men detta betyder att värdena i tabellerna som inte går ihop resulterar i null.

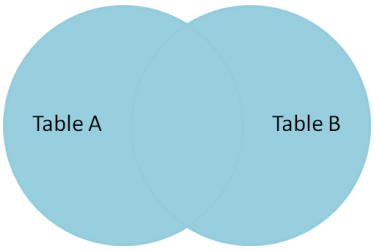


Figure 4: Full outer join

Natural Join

Natural join är ett special fall av en inner join. Den hämtar två tabeller och slår ihop dom till en tabell. där alla kolumner som är ”samma” blir till en så den returnerade tabellen inte har duplicerade kolumner. Extremt användbart i querys som har en relation med en forigienkey!

Order By

Order By är en feature som låter dig att sortera den returnerade tabellen efter bokstavsordning eller storlek i fallande eller stigande ordning. Ex.
Du ska sortera en lista på eleverna på en skola. Listan ska börjar med de äldsta eleverna och sluta med de yngsta.

```
Select * from Students order by birthDate ASC
```

Detta kommer returnera en tabell med alla studenter och deras information. Tabellen kommer vara sorterad efter elevernas personnummer och ha de med lägst personnummer högst upp och de med störst längst ner.

Select Distinct

Select Distinct är ett verktyg som returnerar endast returnerar unika värden av de värden som queryn frågar efter.
Ex.
Du har en webb shop som har en databas över sina kunder. Du vill nu veta till vilka städer du har dina kunder i för att bättre kunna annonsera.

```
Select Distinct CityName from Customers;
```

Denna kod rad kommer att returnera en lista på alla städer där du har en eller flera kunder.

View

View är en temporär tabell som du skapar för att ”gömma” komplexitet eller som en säkerhetsmekanism.
Med glömma komplexitet så menar jag att du kan skriva en avancerad query för att skapa viewn men sen simplare för att hämta information i från just den viewn.
En view kan också funka som en säkerhets mekanism för känslig information genom att dölja information från de ursprungliga tabellerna och där igenom endast låta en användare se den information den behöver och inget mer.

```
create view PlaceHolder as PersNbr,FirstName,BankID from Person, Bankrecords
                                    where bankrecords.persNbr=Person.persNbr;
```

Nu skapar vi en view som vi döper till placeHolder och innehåller persNbr,FirstName och bankID från tabellerna Person och Bankrecords där bankid tillhör personen.
Ingen avancerad query och innehåller ingen speciell logik men den skyddar de underliggande tabellerna från insyn.

Tips ’N Tricks

Det finns några svåra och inte speciellt intuitiva querys som kan komma att begäras av dig i EDA216. En sådan är när du ska hämta information som är beroende av sitt egna tabell men i en annan tuple. Ett sätt som du kan lösa en sån här query är att deklarera två tabellfall(?) för att sedan använda dessa i where satsen av queryn.
Ex.

```
Select persNbr from persons, father person
        where father.persNbr=p.fatherPersNbr
and father.level=level+1 and name=”john”;
```

Ovan så begär vi information endast ifrån tabellen person. En person är definierad av sitt personnummer och vilken nivå i släktträdet denna är på samt persNbr på föräldrarna.

Ett annat exempel av query som är extremt användbar och inte intuitiv är ifall du ska söka igenom en databas efter en delvis matchande uttryck.
Ex.

Du ska få en lista på alla starwarsfilmerna i en filmdatabas.
Hur göra detta?
Det finns ett väldigt snyggt sätt att göra detta på:

Select filmName from movies where filmName="Star wars%";

Det denna queryn gör är att sålla ut alla filmer med ett film-namn som börjar på "Star wars".
% -kommandot säger att det innan alt före den definierade strängen kan vara vad som helst och vilken längd. Det är därför filmName="Star wars%" villkoret ser till att alla filmnamn som börjar på "Star Wars" kommer att returneras.

3 Kommandon

Det finns fem bas kommandon i mySQL: Select, Create,Update, insert into och Deleate. Dessa använder du tillsammans med villkor för att uppdatera en databas med den information du vill ska finnas i den.

Create

Create är det kommando som skapar en tabell eller view.
Ex.

```
Create table person (
age int,
name varChar(25),
persNbr char(10),
Primarykey (persNbr)
) ;
```

Här skapar vi en tabell som vi döper till person med värdena: age,namn, persNbr.
Generellt fall:

```
Create table < Namn >
( < variabelnamn >  < typ >,
  < variabelnamn2 >  < typ > ,
primary key(< variabelnamn >)
);
```

```
Create view <Namn> as
Select <Variabelnamn>,<VariabelNamn2> from <tabellnamn>,<Tabellnamn2>
Where <Vilkor>;
```

Man kan även göra en koll på en variabel, dvs se till så att den hålla sig inom ett bestämt värde. Detta görs via metoden "check" som i exemplet:

```
Create table booties
bootyName varchar(20),
bootySize int check(bootySize > 0 and bootySize < 100)
primary key (bootyName)
);
```

I detta fall beskriver vi en booty, och den kan ju inte vara negativ där av ej mindre än 0. Och sen kan den inte vara orimligt stor enligt mig, så vi sätter en gräns till via "and" större än 100.

Ok, jag har skapat tabeller men hur fyller jag dom med information?
Genom att använda insert into!

```
INSERT INTO <tabellNamn> (<attribut1>,<attribut2>,<attribut3>,...)
VALUES (värde1,värde2,värde3,...);
```

Du använder Insert endast när du har skapat en ny tabell och ska fylla den med information eller om du lägger till ytterligare tuples annars använder du update!

Update

Används för att updatera en eller flera tuples med information.

```
UPDATE <tabellnamn>
SET <VariabelNamn>=value, <VariabelNamn2>=value2,...
WHERE <Vilkor>;
```

On update cascade är ett grymt bra verktyg ifall du ska hålla din databas med bra och användbar information, rekommenderas. ifall vi skapar en tabell:

```
Create table < Namn >
( < variabelnamn >  < typ >,
  < variabelnamn2 >  < typ > ,
primarykey(< variabelnamn >),
forigienkey(< variabelnamn> ) references <tabellnamn>(<variabelnamn>)
on update cascade;
);
```

En forigienkey deklareras och det betyder att denna tabellen är i ett beroende av en annan tabell eller snarare att det värdet som anges som forigienkey syftar på samma sak som i en annan tabell.
Ifall vi nu gör en update på tabellen som har variabeln som vår nya tabell refererar till så kommer även denna tabellen att förändra sig efter de instruktioner du angivit. Grymt användbart!

Deleate

```
DELETE from <tabellnamn>
WHERE <Vilkor>;
```

On deleate cascade är ett extremt bra verktyg för att hålla din databas så ren som möjligt från icke användbar information.
hur funkar det?
Ifall vi skapar ett tabel:

```
Create table < Namn >
( < variabelnamn >  < typ >,
  ........ ,
primarykey(< variabelnamn >)
Foriginkey(<VariabelNamn>) references <tabel2>(<variabelNamn>)
on delete cascade
);
```

Som ni ser så deklarerar vi en forigien key och sen skriver on delete cascade. Detta gör att ifall vi skulle ta bort den refererade tuplen i <table2 >så skulle DBMS också ta bort denna tuple som är direkt kopplad till den andra. på så sätt så slipper man att ha onödig information i en databas och framförallt skriva mindre delete statments!

4 Uppbyggnad

E/R

Vad är en E/R modell?
E/R står för Entity-Relation. Det är alltså ett diagram över entiteter[2] och dess relationer.
Ett Exempel på hur en E/R-modell kan se ut:

Här ser vi en E/R - modell över en rekord databas.

Alla rektanglarna är en entitet. Entiteterna innehåller attribut. Attributen är det som en entitet (framtida tabell innehåller). Mellan entiteter finns det relationer. Alla Entiteter måste ha en relation mellan sig själv och en annan entitet annars är poängen med en databas borta. En entitet utan relationer till andra är endast en tabell med dess egna värden i sig. En relation beskriver hur en entitet förhållersig till en annan.
Ex.

Om vi kollar på E/R - modellen över så ser vi att entiteterna person och club har en relation mellan sig. Den beskriver: 1 club kan ha 0 till flera personer assosierad till sig men en person kan bara tillhöra en club.

[2]Entitet - är en i filosofin använd term för någonting över huvud taget; det måste alltså inte vara ett fysiskt ting, utan en entitet kan lika gärna utgöras av exempelvis en idé eller en teori

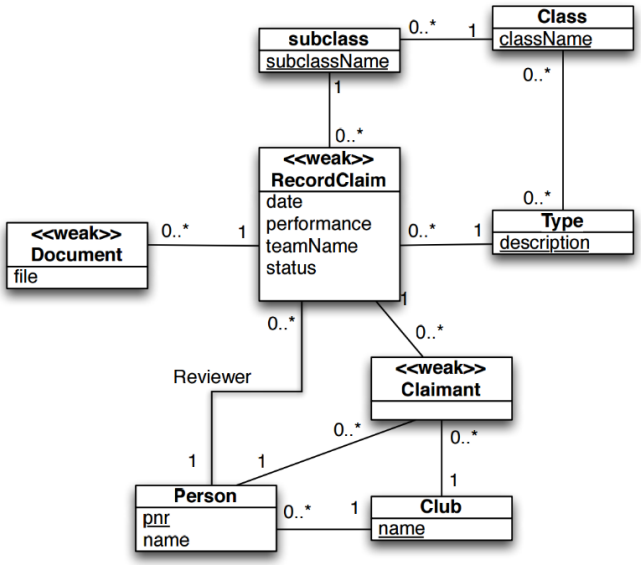


Figure 5: ER modell

E/R till Relation

En relation och hur det fungerar kommer beskrivas mer i detalj nästkommande sidor men nu ska jag försöka förklara hur du tar fram relationerna mellan tabellerna och genom att göra detta få fram hur tabellerna ser ut till fullo. Relation är att en av tabellerna har en forigien key som refererar till den andra tabellen.

Special fall:

Many to many relationer:

Ifall du ser en many-to-many så måste en extra tabell skapas för att hålla reda

Figure 6: Many-To-Many relation

på alla fall då en tuples A har en relation till tuples B och tupels C. Samtidigt som tupels B refererar till en tupels D och X medan tupels C refererar till Y och Z.

Ex.

Ni ser en relation mellan actor och film i Figure 5. En actor kan vara med i flera filmer och en film kan ha flera actors i sig. Därför måste du skapa en ”tear drop”, som jag kallar det, i E/R-modellen. Det är en extra tabell som ofta kallas för ett ”bridge-tabel” men jag tycker tear drop är roligare. En tear drop kan innehålla fler attribut än det som är ”tvingat” av many-to-many relationen.

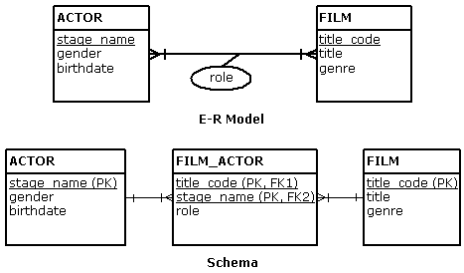


Figure 7: Exempel på Many-to-Many relation

Det som är ”tvingat” att vara med i en tear drop är primarykeys från båda tabellerna som har many-to-many relationen mellan varandra.

5 Relationer

Normalform

–Lär dig detta utantill –

systematiskt sätt att se till att databasstrukturen är lämplig för normala frågedatabaser så att inga oönskade anomalier vid uppdatering eller borttagning kan ske och redundans i databasen, och därmed att skydda databasens integritet.

Finns i första(1NF), andra (2NF) , tredje(3NF) , Boyce-Codds(BCNF) ... 6NF.

1NF:

Första normalformen innebär att varje cell endast innehåller ett värde.Det förutsätts dessutom att raderna är unika, annars är det ingen tabell.

2NF:

Att 1NF uppfylls Dessutom får det inte finnas några fullständiga funktionella beroenden [3] mellan delar av primärnyckeln och attribut i tabellen.

3NF:

Att 2NF uppfylls Att alla attribut på högra sidan om dependencies är del av en nyckel.

BCNF:

Att 3NF uppfylls Att alla attribut på vänstersidan ska vara supernyckel. varje del nyckel kan bara vara beroende på en superkey.

[3] Ett fullständigt funktionellt beroende innebär dels att ett attribut är beroende av ett eller flera andra attribut, dels att de attribut som styr beroendet är så få som de kan vara utan att beroendet upphör.

Keys

Supernyckel - En kombination av attribut som ger alla värden i relationen.
Canidatekey - Den kortaste supernyckeln.
pimarykey - oftast canidatekey. Attribut som används för att identifiera en tuple.
ForigienKey - En nyckel som refererar till en annan nyckel i en annan tabell.

Paired Attribute Algorithm

Paired Attrubute Algorithm är algoritmen du använder för att reducera(?) en relation till mindre relationer som uppfyller BCNF.

Den ser ut så här:

$$\mathsf{R} = (\mathsf{a}, ..., \mathsf{n}) \tag{1}$$

$$\mathsf{X}^{+} = \mathsf{FD}(\mathsf{x} \mapsto \mathsf{y}) \tag{2}$$

$$\mathsf{R}_1 = \mathsf{X}^{+} \tag{3}$$

$$\mathsf{R}_2 = \mathsf{R} - (\mathsf{X}^{+} - \mathsf{x}) \tag{4}$$

$$!((\mathsf{R2\&\&R1}) \in \mathsf{BCNF})? :\mapsto (1) : \mathsf{Done!} \tag{5}$$

Förklaring:
(1): Definiera Relationen
(2): Välj en FD som går emot BCNF
(3): R1 är en av de nya relationerna från relationen R
(4): R2 är den andra delen av relationen R
(5):Se ifall R2 och R1 är i BCNF. Om nej börja om på steg 1 med R=R2.

Exempel:

$$
\begin{aligned}
R &= (a, b, c, d, e) \\
FD1 &: a \mapsto cde \\
FD2 &: d \mapsto e \\
FD3 &: cd \mapsto a
\end{aligned}
$$

Nycklar: {ab} & {CDB}
Vi ser nu att vår relation R inte är i BCNF då alla argument till vänster i FD's inte är supernycklar.(Vill understryka att ingen av FD's är en supernyckel och alla argument är inte del av en supernyckel så den är inte i 3NF heller).
Vi måste därför reducera(?) vår relation R med hjälp av Paired Attribute Algorithm.

$$
\begin{aligned}
R &= (a, b, c, d, e) \\
X^{+} &= FD(a \mapsto cde) \\
R_1 &= X^{+} = \{a, c, d, e\} \\
R_2 &= R - (X^{+} - a) = \{a, b\} \\
R_1 &= (a, c, d, e) \\
X^{+} &= FD(d \mapsto e) \\
R_3 &= (d, e) \\
R_4 &= (a, c, d) \\
reduceratR &\mapsto R_2, R_3, R_4
\end{aligned}
$$

Det vi åstadkommit i exemplet är att reducera relationen R till 3 relationer som tillsammans är i BCNF enligt FD's som var angivna. Varför stannade vi efter att ha reducerat relationen för andra gången? Jo, för nu är a ,cd och d supernycklar!
OBS!!
Svaret till uppgiften är beroende på vilken FD som du utgick från! (a,c,d), (d,e), (b,c,d) och (a,c,d),(a,b),(d,e) är också lösningar!

6 JDBC - Basics

Connection

Nedan ser ni en typisk metod för att öppna en förbindelse från ett java program till en mysql databas.

```
public boolean openConnection(String userName, String password) {
 Connection conn = null;
       try {
           Class.forName("com.mysql.jdbc.Driver");
           conn = DriverManager.getConnection
               ("jdbc:mysql://puccini.cs.lth.se/" + userName,
                userName, password);
       }
       catch (....){
       return false;
       }
       return true;
   }
```

PreparedStatement

Ett preparedStatment är förmodligen den bästa redskapet ni kommer lära er i den här kursen som berör java kod. Det som gör PreparedStatment till ett sån bra verktyg behandlas mycket mer i Datasäkerhets kursen EIT060 men den korta versionen är att du skyddar ditt program från SQL-Injection[4] genom att inte tillåta en användare att manipulera sql querysna som finns i programmet.

```
String sql = "update BankAccounts "
+ "set balance = balance + ? "
+ "where accntNbr = ?";
PreparedStatement ps = conn.prepareStatement(sql);
```

[4]SQL-Injection är ett sätt att utnyttja säkerhetsproblem i hanteringen av indata i vissa datorprogram som arbetar mot en databas

ResultSet

Ett ResultSet är ett prepairdStatment som läser datan från en databas query, den returnerar datan i ett ResultSet.
Nedan ser ni java kod som behandlar ett ResultSet.

```
ResultSet rs = ps.executeQuery(sql);

while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         int age = rs.getInt("age");
         String first = rs.getString("first");
         String last = rs.getString("last");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Age: " + age);
         System.out.print(", First: " + first);
         System.out.println(", Last: " + last);
      }
```

Close();

GLÖM INTE ATT STÄNGA ALLT! i så fall så kommer ditt program vara värdelöst då det kommer dra mycket mer resurser än det behöver.