Joins

Relationer

Hittills har vi bara skrivit queries som hämtar data från en tabell åt gången. Det vanliga i relationsdatabaser är dock att man har flera tabeller där data i tabell A har en relation till data i tabell B.

Att man delar upp data i flera tabeller beror på att man inte vill ha flera kopior av samma data. Dels eftersom det tar onödig plats, men även för att man inte ska behöva uppdatera samma data på flera ställen (och därmed löpa en risk att det finns motsägelser i datat).

Böcker och författare i samma tabell

En databas över böcker och dess författare skulle kunna se ut så här:

Id	Titel	Utgivningsår	Författare	Födelsedatum
1	Ondskan	1981	Jan Guillou	1944-01-17
2	Villospår	1995	Henning Mankell	1948-02-03
3	Brandvägg	1998	Henning Mankell	1948-02-03
4	Innan frosten	2002	Henning Mankell	1947-04-13

Vi behöver då lagra Henning Mankells födelsdatum i flera fält, och i detta fallet har det dessutom smugit sig in ett fel. Dvs, vi har motsägelsefulla uppgifter i våran databas.

Böcker och författare i olika tabeller

Vi skulle istället kunna ha olika tabeller för böcker och författare:

Id	Titel	Utgivningsår	FörfattareID
1	Ondskan	1981	1
2	Villospår	1995	2
3	Brandvägg	1998	2
4	Innan frosten	2002	2

Id	Författare	Födelsedatum
1	Jan Guillou	1944-01-17
2	Henning Mankell	1948-02-03

Nu behövs inte upprepade uppgifter om Mankells födelsedatum, och eftersom varje bok har ett ”FörfattareID” som pekar ut vilken författare som skrivit boken så kan man ändå få ut den informationen.

Primary and foreign keys

Med SQL-kommandot ”join” kan vi länka ihop data från två eller fler tabeller så länge dessa har något gemensamt fält som talar om hur data från de olika tabellerna är relaterade till varandra.

Eftersom alla författare i tabellen ovan har ett unikt Id, en så kallad primärnyckel, så kan vi spara det värdet i en kolumn i boktabellen för att koppla varje bok till en specifik författare. Kolumnen (FörfattareID) som pekar ut en rad i en annan tabell brukar kallas foreign key.

Länka ihop data via query

Antag att vi har två tabeller enligt:

LandsID	Land
1	Sverige
2	Norge
3	Danmark

LandsID	Stad
2	Oslo
3	Köpenhamn
4	Helsingfors

Relationen mellan de två tabellerna kan ses genom LandsID. På så vis är Oslo kopplat till Norge och Köpenhamn till Danmark, medan Sverige inte har någon koppling till stad, samt att Helsingfors inte har någon koppling till land. Låt oss se hur vi kan länka ihop informationen i SQL genom att använda olika typer av ”joins”.

Cross join

Sverige
Norge
Danmark

CROSS JOIN

Oslo
Köpenhamn
Helsingfors

Sverige	Oslo
Sverige	Köpenhamn
Sverige	Helsingfors
Norge	Oslo
Norge	Köpenhamn
Norge	Helsingfors
Danmark	Oslo
Danmark	Köpenhamn
Danmark	Helsingfors

Inner join vs Full join

Sverige
Norge
Danmark

INNER JOIN

Oslo
Köpenhamn
Helsingfors

Norge	Oslo
Danmark	Köpenhamn

Sverige
Norge
Danmark

FULL JOIN

Oslo
Köpenhamn
Helsingfors

Sverige	NULL
Norge	Oslo
Danmark	Köpenhamn
NULL	Helsingfors

Left join vs Right join

Sverige
Norge
Danmark

LEFT JOIN

Oslo
Köpenhamn
Helsingfors

Sverige	NULL
Norge	Oslo
Danmark	Köpenhamn

Sverige
Norge
Danmark

RIGHT JOIN

Oslo
Köpenhamn
Helsingfors

Norge	Oslo
Danmark	Köpenhamn
NULL	Helsingfors

Exempel query

```
Select * from [länder] l
join [städer] s on s.landsID = l.landsID;
```

- Använd ”on” för att ange vilka kolumner man ska ”joina” på.
- När vi bara skriver ”join” så är det samma som ”inner join”.
- Vi har uteslutit ”as” när vi skapar alias för tabellerna.

Junction table

Exemplet ovan är en så kallad ”One to many”-relation. Varje land kan ha flera städer, men varje stad kan bara finnas i ett land.

Om man däremot har en ”Many to many”-relation (t.ex. varje student kan läsa flera kurser, och på varje kurs finns det flera studenter) så behöver man använda en ”Junction table”.

Den tabellen innehåller då (endast) två kolumner. Den ena pekar på primarykey i student-tabellen, medan den andra pekar på PK i tabellen med kurser. Via denna kan man då hitta relationer åt båda håll.