Dataintegritet

Integritetsvilkor och normalisering

EVERYLOOP

www.everyloop.com

Dataintegritet

Dataintegritet handlar om att data i databasen hänger ihop på ett korrekt sätt utan massa felaktigheter, luckor eller motsägelser.

När man skapar en databas kan man sätt upp integritetsvilkor för att tala om för databasen hur, och vilken typ av, data som får lagras. Om man därefter försöker att sätta in data som bryter mot vilkoren så kommer databashanteraren säga ifrån, dvs ej godkänna dessa data.

Det finns ett flertal olika typer av integritetsvilkor (constraints) som man kan sätta på tabeller. Låt oss kolla på några av dessa:

Not NULL

Ett vanligt integritetsvilkor man kan välja att sätta på kolumner är att tala om att en viss kolumn inte får innehålla några NULL-värden.

Det anger man när man skapar tabellen genom att skriva ”not null” vid varje kolumn där man vill att detta ska gälla.

Därefter kommer databashanteraren inte tillåta att det saknas värden (NULL) när man sätter in nya rader i tabellen.

Unique

Man kan även ange att alla värden i en kolumn måste vara unika. Dvs, att kolumnen inte får innehålla några dubbletter. Ett exempel är att varje person man lagrar i en tabell måste ha ett unikt personnummer.

Primary key

En primärnyckel talar om vilken kolumn (kan även vara en komposit av flera kolumner) i databasen som används för att identifiera en unik rad.

En tabell måste inte ha en primärnyckel, men *om* den har det så kan det bara finnas en enda. Den kan dock vara en komposit av flera kolumner.

Primärnyckeln är även *unique* och *not null*. Detta behöver inte anges.

Foreign key

En tabell kan innehålla en eller flera relationsnycklar. Dessa talar om att en viss kolumn används som referens mot en annan tabells primärnyckel för att påvisa en relation mellan tabellerna.

Om man sätter upp ett referensvilkor så anger man för databas-hanteraren att det måste existera en primärnyckel i referenstabellen som motsvarar den relationsnyckel som finns angiven.

Referensvilkor

Antag att vi har två tabeller: En över anställda, och en över avdelningar.

AnställningsID	Namn	AvdelningsID
100	Anders Andersson	1
101	Bengt Bengtsson	1
102	Fredrik Johansson	4

AvdelningsID	Avdelning
1	Data
2	Ekonomi
3	Lager

Om vi hade satt ett referensvilkor i tabellen över anställda, där vi angett att AvdelningsID är en referens till tabellen över avdelningar, så hade vi inte kunna lägga till den sista raden ’Fredrik Johansson’ eftersom det inte finns någon avdelning 4. Vi säkerställer på så vis att referensnycklar inte refererar till data som inte existerar.

Check condition

Med ”Check” så kan vi ange ett vilkor som måste gälla på en kolumn för att data ska anses som godkänt. Vi kan alltså skriva ett vilkor på samma sätt som vi gör i en ”where”-sats för att säkerställa att all data som lagras i kolumnen kommer uppfylla det vilkoret.

Exempel: check(len(personnummer) = 10);

check(totalsumma >= 0);

Default

Man kan även ange default-värde på en kolumn. Alltså det värde som ska antas om man inte anger något värde alls.

Triggers

Ett ytterligare knep man kan använda för att säkerställa dataintegritet är att använda triggers. En trigger är en speciell typ av procedur som körs automatisk efter specifika händelser.

Det finns flera typer av triggers. Bland annat sådana som körs efter insert-, update- eller delete-kommandon.

Låt oss se ett exempel på hur vi automatisk kan uppdatera kolumnen ”kön” så att värdet stämmer överens med det personnummer man anger när man sätter in en ny rad i tabellen ”Personer”:

Trigger after insert

```
create trigger Personer_AfterInsert_TRG
        on personer
after insert
as
begin
        update personer
        set personer.kön = case
                when convert(int, substring(i.personnummer, 9, 1)) % 2 = 0
                then 'Kvinna' else 'Man' end
        from inserted i
        where personer.personnummer = i.personnummer;
end
```

Databasstruktur

När man designar en ny databas bör man tänka på hur man strukturerar de olika tabellerna och datarelationerna så att man får goda förutsättningar att hålla en god dataintegritet och undviker redundant (upprepad) data.

Grundregeln är **en typ av sak per tabell och en sådan sak per rad**.

Exempel: En tabell innehåller böcker (inte böcker och författare), och varje rad innehåller då information om en (1) bok. Tabellen bör då endast innehålla information som är **direkt** kopplad till böckerna.

Normalisering

För att underlätta processen att ta fram en god struktur på sin databas finns det ett antal regler, så kallade **normalformer**. Processen att anpassa sina tabeller efter normalformerna kallas **normalisering**.

Normalformer är alltså vilkor som en tabell kan uppfylla. Den första och enklaste normalformen kallas **1NF**, och genom att lägga på fler vilkor så defineras den andra normalformen **2NF**, sedan **3NF** och så vidare.

Den främsta anledning att normalisera sina tabeller är för att få bort redundant data som påverkar databasens integritet och prestanda.

Första normalformen (1NF)

Det enda som den första normalformen säger är att tabellen ska innehålla **atomära värden** – ett (1) värde per cell.

I de flesta relationsdatabaser går det oftast inte att stoppa in mer än ett värde per cell ändå, med undantag om man skulle välja att göra t.ex komma-separerade textfält.

Man bör alltså undvika att lagra ”sak1, sak2, sak3” i ett fält, utan isåfall välja att göra 3 olika kolumner, alternativt lagra sakerna radvis i en ytterligare tabell och referera dem tillbaks till ursprungstabellen.

Funktionellt beroende

Om värdet på ett (eller flera) attribut A entydigt bestämmer värdet på ett annat attribut B, så är B **funktionellt beroende** av A. (A → B)

Vi kallar A för **determinant**, eftersom den bestämmer B.

I tabellen med böcker nedan så har vi ett flertal sådana beroenden:

Titel	Format	Författare	Sidor	Pris
Lär dig SQL nu!	Inbunden	Fredrik Johansson	318	499
Lär dig SQL nu!	Pocket	Fredrik Johansson	318	179
C# på 7 dagar	Pocket	Johan Fredriksson	453	199
C# på 7 dagar	Inbunden	Johan Fredriksson	453	579

Titel → Författare
Titel → Sidor
(Titel, Format) → Pris

Andra normalformen (2NF)

För att uppfylla 2NF behöver vi uppfylla 1NF + **inga partiella beroenden** dvs, alla värden (icke-nycklar) beror på helheten av varje kandidatnyckel

Boktabellen ovan har en (1) kandidatnyckel, nämligen kompositnyckeln (Titel, Format) som pekar ut en unik rad och därmed även är PK.

Alla fält som inte är del av nyckeln beror på ”Titel”, men endast ”Pris” beror på nyckeln i sin helhet (Titel, Format). För att normalisera tabellen ovan (2NF) så bryter vi ut Format och Pris till en egen tabell och gör på så vis Titel till ensam nyckel. (Alla fält beror på PK i helhet).

1NF ➡ 2NF

För att anpassa den tidigare boktabellen efter andra normalformen så har vi brutit ner den till två tabeller enligt nedan:

Titel	Författare	Sidor
Lär dig SQL nu!	Fredrik Johansson	318
C# på 7 dagar	Johan Fredriksson	453

Titel	Format	Pris
Lär dig SQL nu!	Inbunden	499
Lär dig SQL nu!	Pocket	179
C# på 7 dagar	Pocket	199
C# på 7 dagar	Inbunden	579

Tredje normalformen (3NF)

3NF får man genom att uppfylla 2NF, plus att **inget ”icke nyckel”-attribut får vara beroende av något annat ”icke nyckel”-attribut**.

Titel	Författare	Sidor	Genre_ID	Genre_Namn	Förlag_ID
Lär dig SQL nu!	Fredrik Johansson	318	1	Utbildning	1
Mer SQL nu!	Fredrik Johansson	215	1	Utbildning	1
Jorden runt i år	Annika Rees	273	2	Resor	2
C# på 7 dagar	Johan Fredriksson	453	1	Utbildning	3

Genre_ID och Genre_Namn är beroende av primärnyckeln (Titel), men de är inte oberoende av varandra. Eftersom det finns fler titlar än genrer så introduceras redundant data som kan elimineras genom att bryta ut Genre_ID och Genre_Namn till en fristående tabell.

2NF ➡ 3NF

För att anpassa den tidigare boktabellen efter tredje normalformen så har vi brutit ner den till två tabeller enligt nedan:

Titel	Författare	Sidor	Genre_ID	Förlag_ID
Lär dig SQL nu!	Fredrik Johansson	318	1	1
Mer SQL nu!	Fredrik Johansson	215	1	1
Jorden runt i år	Annika Rees	273	2	2
C# på 7 dagar	Johan Fredriksson	453	1	3

Genre_ID	Genre_Namn
1	Utbildning
2	Resor

Sammanfattning

Med normalisering menar man oftast 3NF, även om förstås även en tabell i 1NF är normaliserad i viss mån.

Det finns dessutom fler normalformer än 3NF, men dem kommer vi inte gå igenom i kursen. Googla ”database normalization” för att se mer.

Sammanfattningsvis är ett gott råd när man designar databaser att tänka: **varje attribut ska representera en fakta om nyckeln, hela nyckeln, och inget annat än nyckeln.**