

SQL: Integritetsregler, triggere og views

Integritetsregler i SQL

- Kandidat- og primærnøkler
- Referanseintegritet – fremmednøkler
- Domenebegrensende integritetsregler – skranker på attributter og tupler
- Interrelasjonsskranker – assertions
- Triggere

Primærnøkler

- Kan deklarerer i **create table** sammen med primærnøkkelattributtet (bare hvis attributtet utgjør primærnøkkelen alene)

```
create table MovieStar(  
  name char(30) primary key,  
  ... );
```

- Kan deklarerer separat i **create table** etter attributtdeklarasjonene

```
create table MovieStar(  
  name char(30),  
  ...  
  primary key (name)  
);
```

Regler for primærnøkler

- Maks én primærnøkkeldeklarasjon pr. relasjon
- Konsekvenser av deklarasjonen:
 - To tupler i relasjonen får ikke stemme overens i alle attributtene i primærnøkkelen (selv ikke med bagrepresentasjon!)
Forsøk på brudd ved **insert** eller **update** skal avvises av DBMSet
 - Attributtene i primærnøkkelen får ikke inneholde **null**
- Primærnøkkelskranken må sjekkes ved hver **insert** og hver **update**

Kandidatnøkler

- Kan deklarerer i **create table** sammen med nøkkelattributtet (bare hvis attributtet utgjør kandidatnøkkel alene)

```
create table MovieStar(  
  address varchar(255) unique,  
  ... );
```

- Kan deklarerer separat i **create table** etter attributtdeklarasjonene

```
create table MovieStar(  
  name varchar(40),  
  address varchar(255),  
  ...  
  unique (name,address)  
);
```

Kandidatnøkler, regler

- Flere kandidatnøkkeldeklarasjoner er tillatt pr. relasjon
- Konsekvenser av deklarasjonen:
 - To tupler i relasjonen får ikke stemme overens i alle attributtene i kandidatnøkkelen
 - Kan brytes hvis ett eller flere av attributtene i kandidatnøkkelen inneholder **null**
- Kandidatnøkkelsranken må sjekkes ved hver **insert** og hver **update**

Indekser på kandidatnøkler

- DBMSer bygger vanligvis indekser automatisk på primærnøklerne.
- For hver kandidatnøkkel må man vurdere spesielt om det bør deklarerer indeks på nøkkelen
 - Syntaks avhenger av DBMSet
 - I Postgres bygges automatisk indekser på kandidatnøklerne
 - Noen SQL-implementasjoner tillater deklarasjon av kandidatnøkkel + indeks i en og samme setning:
create unique index AddressIndex
on MovieStar(address);
- Hvis det er opprettet indeks på en nøkkel, benyttes denne under sjekk av flerforekomster
Ellers: Må i verste fall søke gjennom hele relasjonen

Referanseintegritet = fremmednøkler

- Deklarasjon av fremmednøkler:

```
create table Studio(  
  ..  
  presC# int  
    references MovieExec(cert#),  
  ..);
```

- Alternativt:

```
create table Studio(  
  ..  
  presC# int,  
  ..  
  foreign key (presC#) references MovieExec(cert#)  
  ..);
```


Fremmednøkler, regler

- Konsekvenser av deklarasjonen:
 - De refererte attributtene må være deklarerert **unique** eller **primary key**
 - Verdier (\neq **null**) som opptrer i fremmednøkkelens refererende attributter *må* opptre i de refererte attributtene
- Referanseintegritet må sjekkes både ved **insert**, **update** og **delete**

Opprettholdelse av referanseintegritet

Tre strategier (aktuelle bare ved **update**, **delete**):

- **no action** (= default strategi):
Avvis modifikasjoner som bryter regelen
- **cascade**: Endringer i referert verdi påtvinges også refererende verdi (Hvis du sletter et tuppel t, sletter du alle tupler som refererer t (rekursivt))
- **set null**: Sletting av referert verdi forårsaker endring av refererende verdi til **null**

Strategiene velges separat for hver **update**, **delete** på den enkelte fremmednøkkel:

foreign key (presC#) **references** MovieExec(cert#)
on delete cascade

Midlertidige brudd på referanseintegriteten

Nødvendig ved sirkulære avhengigheter mellom relasjoner

1. Gruppér flere SQL-setninger i en **transaksjon**:

```
start transaction;  
SQL-setninger;  
commit/rollback;
```

2. Kan instruere SQL-systemet om ikke å sjekke referanseintegritet før transaksjonen er endelig avsluttet (committed)
 - i. **not deferrable** (default): Skranken skal sjekkes umiddelbart når sjekk er påkrevd
 - ii. **deferrable**: Sjekking (kan) utsettes
 - a. **initially deferred**: Sjekking utsettes i utgangspunktet til slutten av inneværende transaksjon
 - b. **initially immediate**: Sjekking gjøres i utgangspunktet før modifikasjon av databasen

Deklarasjon av deferrable

- **I create table:**
 - (i) Deklarer deferrable
 - (ii) Navngi skranken (kan sløyfes):

```
presC# int
  unique
  constraint PresC#ForeignKey
  references MovieExec(cert#)
  deferrable initially deferred
```

- Kan senere endre fra **deferred** til **immediate** og omvendt:

```
set constraint PresC#ForeignKey immediate;
..
set constraint PresC#ForeignKey deferred;
```

Skranke på attributter og tupler

- Bidrar til å begrense lovlige (kombinasjoner av) verdier i attributtene innen en relasjon
- Tommelfingerregel:
Bare *svært enkle* skranke bør formuleres på denne måten!

Skranke på ett attributt: **not null**

- I create table:

```
presC# int  
  references MovieExec(cert#) not null
```

- Konsekvenser:
 - Kan ikke sette inn tuppel med verdien **null** i attributtet
 - Kan ikke endre verdien til **null** senere
 - **set null**-policyen kan ikke benyttes (dette innvirker på hvilke views som blir oppdaterbare)

Skranke på ett attributt: **check**

- I create table:

```
presC# int
references MovieExec(cert#)
check (presC# >= 100000 and
presC# < 1000000);
```

- Angir en betingelse på attributtet
- Betingelsen kan henvise til attributtet selv, og til andre attributter – via subquery
- Sjekkes ved hver endring av attributtets verdi
- **NB** Sjekkes *ikke* hvis **check** refererer til andre attributter og disse endres, mens attributtet med **check**-skranke ikke endres!

Skranke på et tuppel: **check**

- I **create table**:

...

```
check ((gender = 'F' or name not like 'Ms.%')  
        and (gender = 'M' or name not like 'Mr.%'))
```

...

- Angir en betingelse på et tuppel i relasjonen
- Betingelsen kan henvise til attributter i relasjonen, og til attributter i andre relasjoner – via subquery
- Sjekkes hver gang et tuppel settes inn eller oppdateres
- **NB** Sjekkes *ikke* hvis **check** refererer til attributter i andre relasjoner og disse endres, mens det ikke er endringer i inneværende relasjon!

Endring av skranker

- Skranker kan legges til, endres, slettes, når som helst
Man må i så fall navngi skrankene sammen med deklarasjonen av dem
- Endring av tabellskranker:
 - **immediate** ↔ **deferred** ved **set constraint** ...
 - **alter table**: Legg til eller slett skranker
 - **alter table** MovieStar **drop constraint** NamelsKey;
 - **alter table** MovieStar **add constraint** MyKey **primary key** (name);
 - **alter table** MovieStar **add constraint** NoAndro **check** (gender in ('F', 'M'));

Assertions–I

- Assertions er en del av databaseskjemaet, på lik linje med relasjonsskjemaer og views:

```
create assertion SumLength  
check (10000 >= all  
  (select sum(length)  
  from Movie  
  group by studioName));
```

Assertions–II

- Assertions formuleres som et boolsk uttrykk som alltid må evalueres til **true**
- Kan involvere flere relasjoner
- Sjekkes når en av de involverte relasjonene endres
- Vanskelig å implementere effektivt
- Kan fjernes igjen:
drop assertion SumLength
- *Assertions støttes ikke av Postgres*

Triggere

- Serie aksjoner assosiert med angitte hendelser
Utføres når hendelsen inntreffer
- Vekkes når en *hendelse* skal til å inntreffe
(typisk: **insert**, **update**, **delete** mhp. gitt relasjon)
- Deretter testes en *betingelse*
Hvis betingelsen ikke holder, avbrytes triggeringen
- Hvis betingelsen er oppfylt, utføres en *aksjon*
Aksjonen kan hindre hendelsen i å inntreffe eller omgjøre den mm.
Generelt er aksjonen en sekvens av database-operasjoner

Valgmuligheter for triggere i SQL

1. Aksjonen kan eksekveres før eller etter den triggende hendelsen
2. Aksjonen kan referere til gamle og/eller nye tuppelverdier som blir satt inn, slettet eller oppdatert i den triggende hendelsen
3. Oppdateringshendelser kan begrenses til et gitt attributt/ en gruppe attributter
4. En betingelse kan angis; da trigges det bare hvis hendelsen inntreffer og betingelsen evaluerer til **true**
Dette gjøres i en **when**-klausul
5. Programmereren kan spesifisere at aksjonen skal utføres
 - enten én gang for hvert modifisert tuppel
 - eller én gang for samtlige modifiserte tupler

Postgres støtter bare et subset av triggerfunksjonaliteten i SQL-standard!

Eksemplene på de neste lysarkene har konstruksjoner som *ikke* støttes i Postgres.

Trigger, eksempel 1

Skjema: MovieExec(name, address, cert#, netWorth)

```
create trigger NetWorthTrigger
after update on MovieExec
referencing
  old row as OldRow,
  new row as NewRow
for each row
when (OldRow.netWorth > NewRow.netWorth)
update MovieExec
set netWorth = OldRow.netWorth
where cert# = NewRow.cert#;
```

Trigger, eksempel 2

```
create trigger AvgNetWorthTrigger
after update of netWorth on MovieExec
referencing
  old table as OldStuff,
  new table as NewStuff
for each statement
when (500000 > (select avg(netWorth) from MovieExec))
begin
  delete from MovieExec
  where (name, address, cert#, netWorth) in NewStuff;
  insert into MovieExec (select * from OldStuff);
end;
```

Virtuelle og materialiserte views

- Tuplene i et vanlig view beregnes først når viewet navngis i en query. Slike views kalles derfor også for **virtuelle views**
- Hvis et view brukes mye, kan det **materialiseres**:
create materialized view viewnavn **as**
select .. from ..;
Tuplene i viewet blir da lagret og må oppdateres
 - enten når de underliggende relasjonene endres (kan være kostbart)
 - eller periodisk (f.eks. hver natt) (dataene i viewet kan bli foreldet i løpet av perioden)
- **Postgres støtter ikke materialiserte views**

Modifisering av viewdata

- **Oppdaterbare views**

Virtuelle views hvor det er mulig å oversette modifikasjon på tupler i viewet til modifikasjoner på tupler i basisrelasjonene

- Forenklet: Et view er oppdaterbart hvis det er definert ved en **select** (*ikke distinct*) på attributter fra én basisrelasjon eller ett annet oppdaterbart view R hvor
 - **where** ikke involverer R i et subquery
 - listen i **select** er fyldig nok til at vi kan etterfylle ikke angitte attributter med defaultverdier eller **null** og slik konstruere et basistuppel som produserer det angitte viewtuppelet

Instead-of triggerere

- En **instead-of trigger** er en trigger som avskjærer hendelser rettet mot et virtuelt view og erstatter hendelsene med alternative aksjoner
- F.eks. kan triggeren ”oversette” en **insert**, **update** eller **delete** mhp. viewet til **insert/ update/ delete** mot de underliggende relasjonene

Eksempel på instead-of trigger

```
create view ParamountMovies as  
select title, year  
from Movies  
where studioName = 'Paramount';
```

```
create trigger ParamountInsert  
instead of insert on ParamountMovies  
referencing new row as NewRow  
for each row  
insert into Movies(title, year, studioName)  
values(NewRow.title, NewRow.year, 'Paramount');
```

Øvingsoppgave

- Skjema:

Prosjekt(P#, Pnavn, Kunde, Pleder, StartDato)

Ansatt(A#, Navn, Tittel, Fdato, Pnr, AnsDato)

Timeliste(A#, Dato, P#, Timer)

Kunde(K#, Knavn, Adresse)

- Oppgave:

Finn navn og tittel på de ansatte som har arbeidet på alle prosjekter som er påbegynt i dette århundret og er bestilt av kunden «Pust og pes AS».

Øvingsoppgave

- Prosjekt(P#, Pnavn, Kunde, Pleder, StartDato)
Ansatt(A#, Navn, Tittel, Fdato, Pnr, AnsDato)
Timeliste(A#, Dato, P#, Timer)
Kunde(K#, Knavn, Adresse)

- Løsning:

```
select Navn, Tittel from Ansatt A where not exists (  
  select * from Kunde K, Prosjekt P  
  where Knavn = 'Pust og pes AS' and K# = Kunde and  
    StartDato > '2000-12-31' and not exists (  
    select * from Timeliste T  
    where T.A# = A.A# and T.P# = P.P# ) )
```