Triggere og lagrede prosedyrer.

(Connolly & Begg kap. 8,1-3, men Oracle-basert)

Hva?

- En **trigger** er en programsnutt som blir "avfyrt" automatisk dersom en bestemt tilstand finnes i databasen, gjerne før eller etter en innsetting, endring eller sletting. NB! Vi har ingen kontroll på når en trigger kjøres.

```
CREATE TRIGGER <triggernavn>
    BEFORE DELETE ON <tabell>          eller  INSERT /  UPDATE OF <kolonner>
    BEGIN
    …..
    END;
```

- En lagret prosedyre er en programsnutt som kan kalles av oss.
  Engelsk: stored procedure, sp.

```
CREATE PROCEDURE <prosedyrenavn> ([parameter : type, parameter : type,…])
BEGIN
    ….
END;
```

- En **lagret funksjon** er en programsnutt kalles av oss, og som returnerer en verdi

```
CREATE FUNCTION <funksjonsnavn> ([parameter : type, parameter : type,…])
RETURNS  text
BEGIN
    ….
END;
```

  Engelsk: stored function.

- Kan også kjøre kode direkte f.eks. i en editor, uten å lagre den i en funksjon (ad-hoc-kode).

NB! Lagres som en del av databasen.

Enkelt språk, med det nødvendigste

Betegnes ofte som databasesystemets programmeringsspråk (f.eks.: i Oracle kalles det PL/SQL = Programming Language / SQL).

- parametre
- mulighet for å lage lokale variable
- kontrollstruktur (if / case, while/repeat/loop etc.)
- tilordning, som regel :=, set ny_verdi = <beregning>
- mulighet for å kjøre SQL-setninger, inkl. legge inn data enkeltdata fra disse i lokale variable.
- kan lese evt. endre verdier i tabeller, tilstander, eksterne variable (f.eks. hvem som er logget på) etc.
- kan skrive meldinger til skjerm el.l.
- kan også referere til eksterne biblioteker, f.eks. et API.

Eksempler på system og tilhørende språk:

Database system	**Implementation language**
DB2	SQL PL (close to the SQL/PSM standard) or Java
Informix	SPL or Java
Microsoft SQL Server	Transact-SQL and various .NET Framework languages
MySQL	own stored procedures, closely adhering to SQL/PSM standard.
Oracle	PL/SQL or Java
PostgreSQL	PL/pgSQL, can also use own function languages such as pl/perl or pl/php
Sybase ASE	Transact-SQL

(https://en.wikipedia.org/wiki/Stored_procedure)

Lagrede funksjoner.

Funksjoner returnerer en verdi av en gitt datatype. Kan ha en eller flere parametre, som alle må være IN-parametre (verdiparametre).

Eksempel 1 – svært enkelt:

Gitt en tabell Ansatt med bl.a. ansattnr og lønn pr. måned. Vi vil skrive ut årslønna i tillegg.

Oracle:

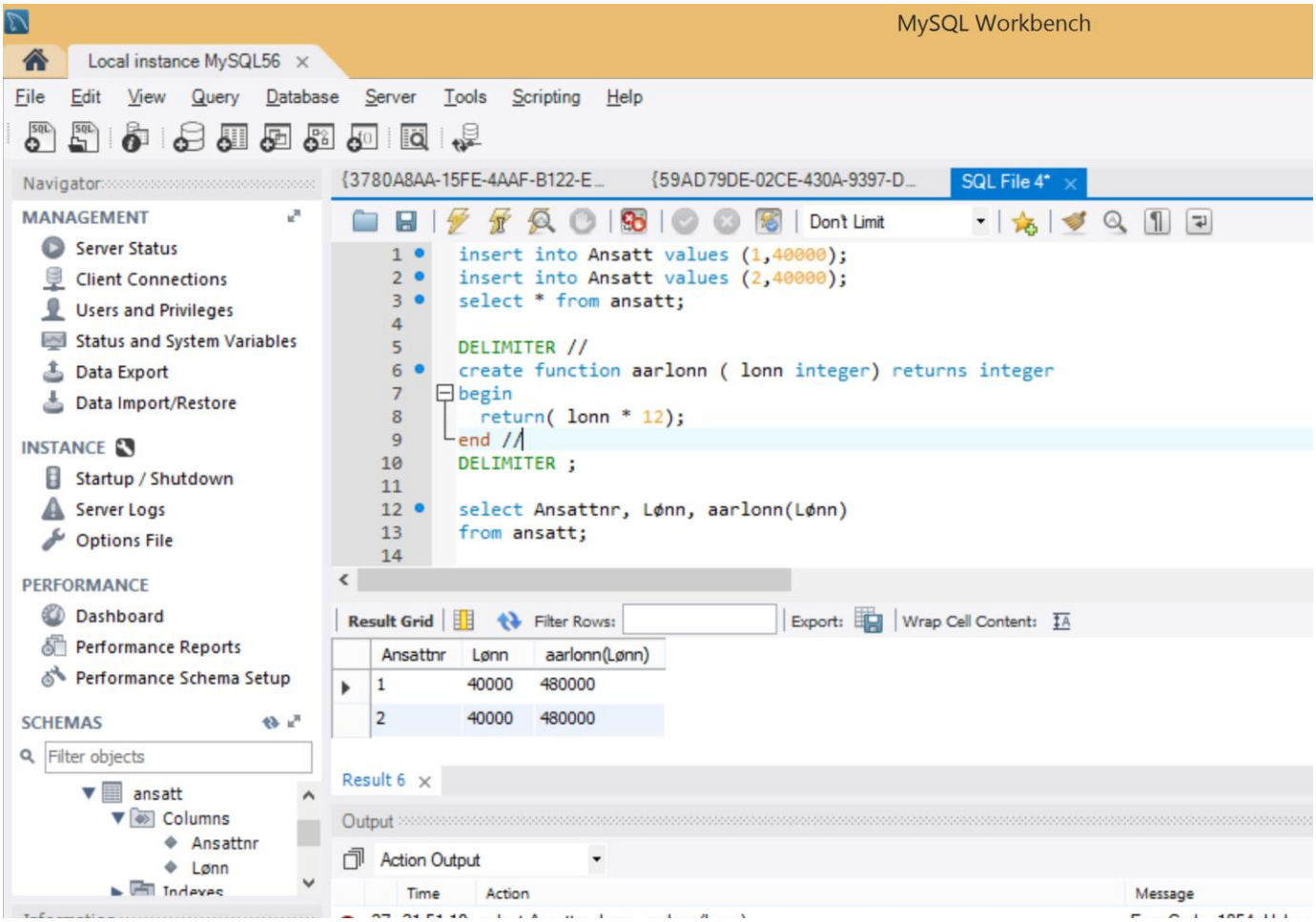
```
create or replace function aarlonn ( lonn in integer) return integer
is
begin
  return (lonn * 12);
end;
```

MySQL:

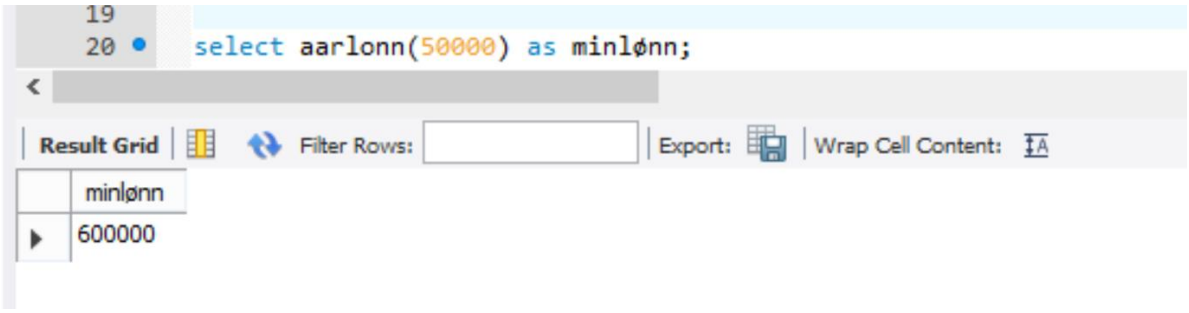
; som skilletegn brukes bade i SQL og i spørrespråket. Derfor må du endre global skilletegn. Dermed oppfattes hele funksjonen som en setning.

```
DELIMITER //
create function aarlonn ( lonn integer)  -- har ikke create or replace, men har
                                            drop function if exists <tabellnavn>.
returns integer
begin
  return( lonn * 12);
end //
DELIMITER ;                  -- tilbakestilling av skilletegn.
```

Legg merke til småforskjeller som return vs. returns m.fl.
Kjøring i MySQL:



Du kan også kjøre funksjonen direkte, som f.eks.



Dette viser at funksjoner (og prosedyrer) ikke er knyttet til en bestemt tabell.

NB! Denne beregningen kunne vært gjort direkte i SQL:
Select ansattnr, lønn, lønn*12 as årslønn

Eksempel 2. Vi kan godt hente data fra tabeller.

```
drop function if exists tellarter;   -- smart å ta med.
delimiter //

create function tellarter()
 returns integer
begin
   return (select count(*) from mod_veritas.art);
end //

delimiter ;
```

Test på at det funker: Kjør select tellarter();

Vi kan også tilordne resultatet til en variabel.

```
create function tell2()
 returns integer
begin
   declare antall integer;
   select count(*) into antall from mod_veritas.art;
 return (antall);
end //
```

Dessverre har mySQL en begrensning at tabellnavn ikke kan være parameter, slik at man ikke kan bruke en SQL-streng som er bygd opp dynamisk for å utføre en mer generalisert spørring.

Dette kan imidlertid gjøres via andre programmeringsspråk eller – miljøer.

Eksempel 3 – litt større eksempel

INN: en vilkårlig streng,
UT: strengen, men med blank mellom hvert tegn.

```
create function annenhver (s varchar(100))
returns varchar(200)
begin
        declare i integer default 1;
        declare sny varchar(200) default "";
        while i <= length(s) do
                set sny = concat(sny , substring(s,i,1) , " ");
                set i = i + 1;
        end while;
        return (sny);
end;
```

Utvidelse: legg inn antall blanke som parameter.

En god del språkelementer som vi bruker kan betraktes som lagrede funksjoner, f.eks. upper (gir store bokstaver), power (opphøyd i), substring m.m.

Mye kan gjøres direkte i spørrespråket, men har naturlig nok en del begrensninger, derfor er et programmeringsspråk nyttig å ha.

Tips når dere skriver funksjoner / prosedyrerer / triggere:

- Lag et tomt skall, fyll deretter på minst mulig om gangen, kompiler til stadighet, test ut at det fungerer etc.

- Du kan skrive ut verdiene på variable med en select-setning, denne trenger ikke å ha noe from-del e.l.

Lagrede prosedyrer

Prosedyrer kan ha parametre som er IN, OUT eller INOUT.
Default er IN hvis det ikke oppgis noe.

Må kalles eksplisitt, som
CALL <prosedyrenavn>; , evt. med parametre.

Eksempel 1. Automatisk reindeksering.

```
drop procedure reindekser;
delimiter //
create procedure reindekser ()
begin
 drop index art_i on art;
 create unique index art_i on art(artsindeks) ;
end //
delimiter ;

call reindekser;
```

Eksempel 2. Kjør en SQL-setning med et enkelt kall, eller kjør mange SQL-setninger samtidig.

```
create procedure kjørtabeller()
begin
        <innviklet SQL-spørring>
        <eller kanskje flere>
end;
```

Startes ved

```
call kjørtabeller;
```

Eksempel 3. Verdier for betingelser kan være parametre.

```
DELIMITER //
CREATE PROCEDURE country_hos
(IN con CHAR(20))
BEGIN
  SELECT Name, HeadOfState FROM Country
  WHERE Continent = con;
END //
DELIMITER ;
```

(hentet fra MySQLs dokumentasjon).

Kalles på med f.eks. country_hos (‘Europe’).

Triggere - 1.

Fordeler med triggere:

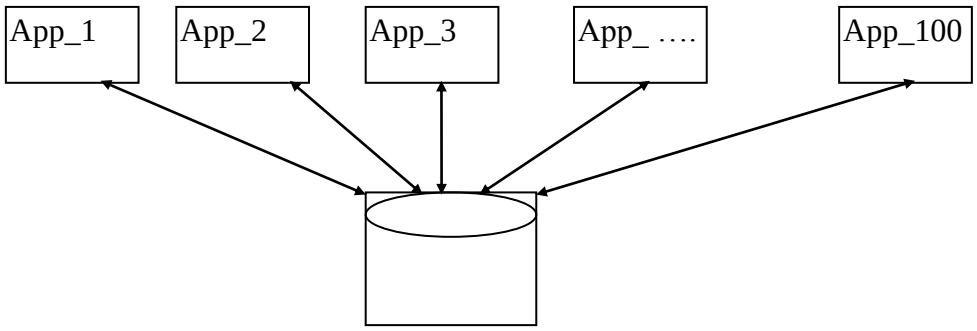
- Kan formulere ting som ikke lar seg formulere i SQL.
- Spesielt: hendelsesstyrte forhold.
- Kan beskrive fundamentale regler for databasen ETT sted ==> behøver ikke å beskrive det i alle applikasjonene som jobber mot databasen.
- Regler kan ikke omgås.

Eksempler på bruksområder:

- Mer komplisert sjekk av inndata før innsetting
- Mer komplisert sjekk av datatilstander før sletting / endring.
- Fullstendig logging av gitte endringer i databasen. Man ønsker at alle endringer av gitte tabeller skal logges, med tidspunkt, dato etc., slik at man siden kan se hvem og hva som er gjort med dataene (f.eks. i et saksbehandlersystem).
- Sørge for konsistensen i avledede data. Vi ønsker f.eks. å lagre en sum fysisk i databasen, f.eks. av ytelseshensyn. Legg en trigger som reagerer på alle endringer i data som innvirker på denne summen.
- Sørge for konsistens ved de-normalisering.
- Mer kompliserte integritetssjekker.
- Data som er ferdigbehandlet skal over til en reservefil, f.eks. hvis status = "Passiv" → skriv til ny fil, slett fra nåværende fil. Evt: Gjøres bare dersom også overskredet en viss dato.
- Passe på overgangs- og forretningsregler av ulik type, f.eks.
    - En person kan ikke gå over fra status gift til status ugift.
    - Varen må ha status "På lager" før den kan sendes ut.
    - Person skal bli sjef for avdelingen ==> må selv jobbe i avdelingen.
    - IF målt_verdi > grenseverdi THEN lag_alarm.

Triggere - 2.

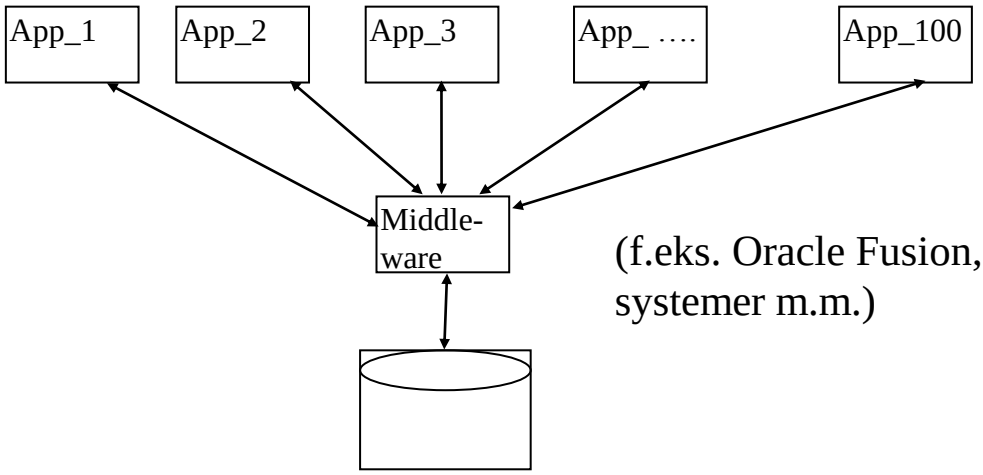
Men: hvor går grensen mellom databasespesifike og applikasjonsspesifike regler?



Hva hvis app_100 ønsker at trigger x ikke skal gjelde?

- Løse opp triggerne? Men hva med de 99 applikasjonene som "stolte på" at triggeren x gjaldt? Omskrive alle?
- Endre triggeren slik at denne sier
  "IF applikasjon100 THEN exit ELSE <gammel trigger>.."? Men: dette betyr uansett at databasen kan få en tilstand som ikke var forventet av de andre applikasjonene.

Ofte brukes «middleware» bl.a. for behandling av forretningslogikk

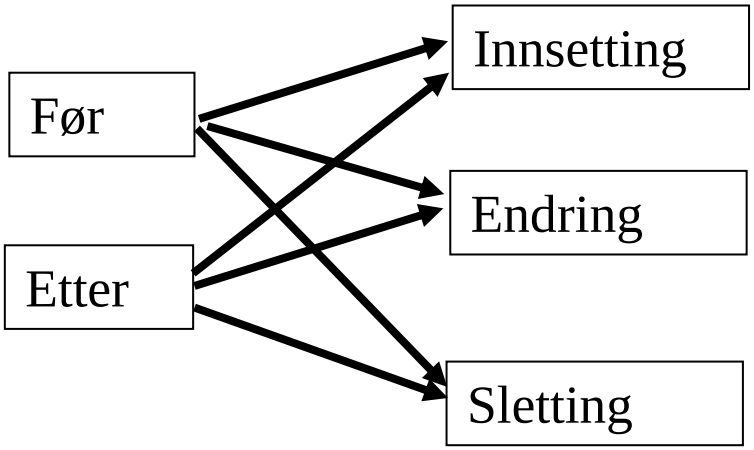


(f.eks. Oracle Fusion, SOA-baserte systemer m.m.)

Gir en ryddig oppdeling, men også her er det ofte være et skjønnsspørsmål hvor koden / logikken bør ligge.

Triggere - 3.

Triggere kan avfyres



BEFORE/AFTER INSERT / UPDATE(<kolonne/liste>)/DELETE

Det kan være lurt å navngi triggerne etter dette, f.eks.

CREATE TRIGGER Før_Kunde_Slettes
BEFORE DELETE ON CUSTOMER
Begin
…….
End;

Det brukes samme språk som for funksjoner og prosedyrer.

Triggere - 4.

Konklusjon:

- Bare regler du vet ikke endres bør legges til triggere.
- Dokumentér hvilke applikasjoner som er avhengig av en gitt trigger.

En advarsel:

En trigger kan trigge andre triggere, fordi du kan endre data i en annen tabell i triggeren. Hvis den andre tabellen har en trigger på seg, vil den også trigges.

Sjekk for trigging "i sirkel".

- direkte, f.eks. hvis man i before insert har en ny insert i samme tabell.
  parallell: «Før jeg er ferdig med å spise, vil jeg ta en skive til»

- Indirekte, ved at man til slutt trigger tilbake til opprinnelig tabell.

```
create trigger før_A_innsettes before insert on A
begin
        insert ….  into B;
end;

create trigger før_B_innsettes before insert on  B
begin
   insert ….. into A;
end;
```

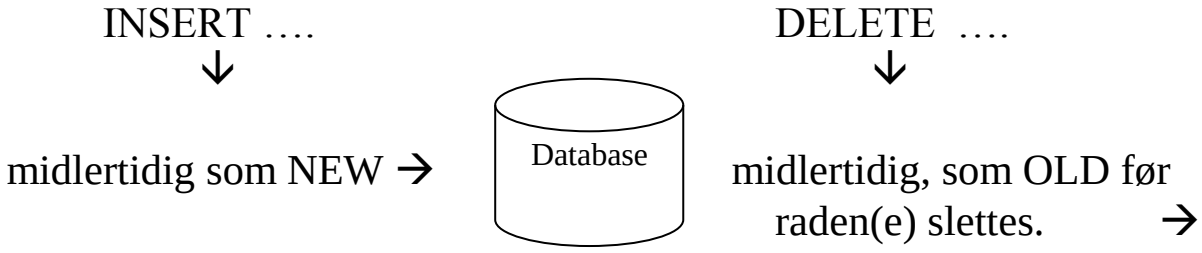
Kan være trigging i mange ledd, derfor vanskelig å oppdage. Tegn graf!

Ulike databasesystemer er laget ulikt med hensyn til oppdagelse av dette.

- Selv-trigging kan i prinsippet oppdages ved kompilering.
- Trigging i sirkel kan oppdages i run-time (Oracle: table .. mutates)
- Hvis ikke dette oppdages, får man en uendelig trigging.

Triggere – 5

Kan referere til verdier som er i ferd med å slettes eller innsettes, gjerne liggende i tenkte tabeller, ofte kalt **OLD** og **NEW**



OLD og NEW har samme struktur som de tabellene man jobber med. Kan f.eks. skrive

if OLD.endringsdato > .. så la være å slette (rollback)

eller NEW.<verdi> := ….. før innsetting.

- Blir som et «venterom» før innsetting / sletting, kan huke tak i dem før de innsettes hhv. slettes.
- Ved update har OLD gammel verdi, mens NEW har ny verdi.

I noen systemer heter det tilsvarende DELETED og INSERTED.

Trigger som hindrer sletting av en post:

(NB! I Oracle)

```
create trigger ikkeslettansatt
before delete on employee
for each row
begin
rollback trigger;
end;
```

Triggere og referanseintegritet.

(ikke viktig i seg selv, men viser eksempel på mulighet for automatisk generering og tilpasning)

Triggere kan også brukes til å definere referanseintegritet, men som regel mer effektivt og enklere å gjøre det deklarativt (… FOREIGN KEY ….. REFERENCES …….). Kan imidlertid være lurt hvis man skal gjøre noe spesielt før sletting el.l.

Eksempel på trigger for referanseintegritet (skrevet for MS SQL server), automatisk generert.

Ansatt er relatert til en avdeling og muligens mange kursdeltagelser.

```
CREATE TRIGGER ansatt_upd
ON ansatt
for update as
    /* At 1-side: Primary key may not be changed if still having members */
    /* At m-side: Foreign key may not be changed if the new value does   */
    /*          not match an owner.                                      */
if update(ansnr )
begin
  if exists (select *
     from kursdeltagelse, deleted
     where kursdeltagelse.ansnr = deleted.ansnr)
  begin
   raiserror 25004 "Det finnes fremdeles relaterte kursdeltagelse(er)"
   rollback transaction
   return
  end
end
if update(avdnr)
begin
  if not exists
     (select *
     from avdeling, inserted
     where avdeling.avdnr = inserted.avdnr)
  begin
   raiserror 25003 "Finnes ingen tilsvarende avdeling."
   rollback transaction
   return
  end
end
end
```

Hvis du oppdaterer ansatt:

Hvis du vil oppdatere ansattnr: Nekt hvis det fremdeles finnes tilhørende kursdeltagelser.

Hvis du oppdaterer avdnr: Ny avdnr må matche med en eksisterende avdnr i avd.

Eksempel.

Vi skal lage en trigger som automatisk holder en sum ved like ("burde" ikke vært lagret fysisk), og som legger slettede poster til ny tabell:

```
drop table ans;
drop table anslager;
drop table sumlonn;
```

Triggere kan operere på

- hele tabellen
- på hver rad som f.eks. skal innsettes eller slettes. Dette markeres med nøkkelordet FOR EACH ROW.

Tabeller og initialisering:

```
create table ans (ansnr integer primary key, lonn integer);
create table anslager (ansnr integer , lonn integer);
```

← burde også hatt dato for sletting

etc.

```
create table sumlonn (id integer primary key, x integer);
insert into sumlonn values (1,0);
```

kunne inneholdt diverse standarddata

ANS

1	450000
2	450000
3	100000

ANSLAGER

ny: legg til lønna her
endre: endre lønna her
slett: trekk fra lønna her , samt flytt raden(e) over til før sletting.

SUMLONN

1	9000000

NB! Tenk gjennom hvilke triggere som trengs, og hva de skal gjøre.

```
create trigger after_ans_insert
after insert on ans
for each row
begin
     update sumlonn set x = x + new.lonn where id = 1;
end;


create trigger before_ans_delete
before delete on ans
for each row
begin
     update sumlonn set x = x - old.lonn where id = 1;
    insert into anslager select * from ans where old.ansnr = ans.ansnr;
end;


create trigger after_ans_update
after update on ans
for each row
begin
     update sumlonn set x = x +new.lonn- old.lonn where id = 1;
end;
```

Bruk av markører/cursors.

Både lagrede prosedyrer, funksjoner og triggere gir anledning til å bruke såkalte markører / cursors, som gjør at vi kan gå gjennom en og en rad i tabellen. Markøren «peker på» en bestemt rad i tabellen / evt. SQL-spørringen, slik at du kan hente data og gjøre en detaljert behandling av disse utover det som er mulig i et spørrespråk.

Mange systemer har begrensninger med hensyn til bruk av cursorer.

Typisk bruk av cursor:

<definer cursoren og bind den til en tabell/spørring>
<åpne den, her finnes det ulike modus i noen språk (for les/skriv)>
<hent (fetch) første rad>
while <flere igjen>
    <behandle denne raden (← kalles ofte current of record) >
    <hent neste>
elihw
<lukk cursoren>

Å se på en og en rad strider jo mot prinsippet i relasjonsdatabaser (såkalt impedans mellom programmeringsspråkene), men er av og til nødvendig.

SQL	Mengde / bag-basert	Behandler en mengde (egentlig bag) om gangen, kraftfulle kommandoer som gjelder alle
De fleste programmeringsspråk	Rad-basert	Behandler en og en rad om gangen i en løkke

Bruk av markører – eksempel.

SQL har generelt problemer med at man ikke har noen «posisjon», og heller ikke forrige, neste osv.

Vi har en tabell resultat med bl.a. kolonnen verdi. Tenk deg f.eks. at det er fra et idrettsstevne (f.eks. høydehopp). Vi skal skrive ut det **x-te beste** resultatet, hvor x er en vilkårlig verdi. Hvis flere har dette resultatet, skal alle disse skrives ut.

RESULTAT

ID	Verdi	Navn
1	205	Alsen
2	203	Bø
3	219	Carlsen
4	205	Dal
5	207	Elvesen

NB! Tabellen er ikke nødvendigvis sortert på verdi. Hvis vi spør om

1ste beste verdi → 219 Carlsen
2ndre « « → 207 Elvesen
3dje « « → 205 Alsen og 205 Dal

Vi lager en prosedyre xte_beste som tar inn en verdi (finn_for_plass) som vi er plasseringen, f.eks. xte_beste (3). Vi trenger bl.a. en cursor (vi kaller den toppbunn) som går gjennom rad for rad, og status som sier om vi er ferdig eller ikke.

```
drop procedure xte_best;
delimiter //
create procedure xte_best (finn_for_plass integer)
begin
      declare ferdig int default false;
      declare denneverdi int;     -- ruller over verdiene som skal sammenlignes
      declare posisjon integer;
      declare toppbunn cursor for SELECT verdi FROM resultat order by verdi;
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET ferdig = TRUE;
      open toppbunn;
     fetch toppbunn into denneverdi;  -- 1. kolonne i denne raden,  ok med kommasep. liste
     set posisjon = 1;
     while not ferdig do
       if posisjon = finn_for_plass then
            select * from resultat where verdi = denneverdi;
       end if;
       fetch toppbunn into denneverdi;
       set posisjon = posisjon + 1;
   end while;
   close toppbunn;
end//      NB! Systemet kan forbedres på mange måter, men dette holder som en grei demo.
```

Bruk av markører - anvendt på sortering.

Hold en sortert liste ved siden av de vanlige tabellene.
Dessverre er markører dårlig implementert i mySQL(bl.a. kan ikke enkeltradene endres), vi tar derfor et eksempel fra Oracle, som ikke er mulig å gjennomføre i mySQL.

Anta at vi har tabellen **testtab(ansnr,ansnavn,lonn,lonnsplass)**

Lønnsplass skal vise nåværende plassering i lønninger blant de ansatte. Vi ønsker å lagre dette, bl.a. slik at det skal være lett å skrive ut hvem som har den 6. beste lønnen. Eksempelet kunne godt vært i forhold til oppnådd bonus, i forhold til plassering i en idrettsøvelse el.l. Poenget er at vi ønsker at plasseringen skal være lagret.

De 3 første kolonnene skrives inn av oss, det siste ordnes automatisk av triggeren.

```
create or replace trigger nyplass after insert or update of lonn or delete on testtab
declare
        cursor c1 is
        select * from testtab order by lonn desc  for update of lonnsplass;
        c1_rec c1%rowtype;
        teller integer;
BEGIN
        open c1;
        teller := 0;
        fetch c1 into c1_rec;
        while c1%found loop
                teller := teller +1;
                update testtab set lonnsplass = teller where current of c1;
                fetch c1 into c1_rec;
        end loop;
END;
```

NB! Triggeren vil ikke markere like verdier med samme tall (“delt 3.plass”) men det er en enkel utvidelse å få til dette.
NB2! Det finnes sikkert mer effektive måter å gjøre dette på – men det spiller mindre rolle for dette eksempelet.