IN2090 – Databaser og datamodellering
06 – Datamanipulering med SQL

Leif Harald Karlsen
leifhka@ifi.uio.no

Universitetet i Oslo

Typer SQL-spørringer

Som sagt tidligere, SQL kan gjøre mye mer enn bare uthenting av data.
Det første ordet i en spørring sier hva spørringen gjør:

SELECT	henter informasjon (svarer på et spørsmål)
CREATE	lager noe (f.eks. en ny tabell)
INSERT	setter inn rader i en tabell
UPDATE	oppdaterer data i en tabell
DELETE	sletter rader fra en tabell
DROP	sletter en hel ting (f.eks. en hel tabell)

SQLs ulike fuksjoner

De ulike spørringene er egentlig deler av ulike under-språk av SQL. Vi har

- SDL (Storage Definition Language): 3-skjemaarkitekturens fysiske lag
- DDL (Data Definition Language): 3-skjemaarkitekturens konseptuelle lag
- VDL (View Definition Language): 3-skjemaarkitekturens presentasjonslag
- DML (Data Manipulation Language): innlegging, endring og sletting av data
- DQL (Data Query Language): spørrespråk
- DCL (Data Control Language): integritet og sikkerhet

Lage ting

- For å lage tabeller, brukere, skjemaer, osv. bruker vi `CREATE`-kommandoer
- For å lage et skjema gjør vi

```
CREATE SCHEMA northwind;
```

- SQL-kommandoen for å lage tabeller har formen:

```
CREATE TABLE <tabellnavn> ( <kolonner> );
```

- hvor `<tabellnavn>` er et tabellnavn (potensielt prefikset med et skjemanavn)
- og `<kolonner>` er kolonne-deklareringer
- En kolonne-deklarering inneholder
  - et kolonnenavn, og
  - en type,
  - og en liste med skranker (constraints)

CREATE-eksempel

- For å lage Students-tabellen kan vi kjøre

```
CREATE TABLE Students (
    SID int,
    StdName text,
    StdBirthdate date
);
```

- Nå vil følgende tomme tabell finnes i databasen:

Students

SID (int)	StdName (text)	StdBirthdate (date)

Skranker: `NOT NULL`

- I mange tilfeller ønsker vi å ikke tillate `NULL`-verdier i en kolonne
- For eksempel dersom verdien er påkrevd for at dataene skal gi mening
  - F.eks. vi vil aldri legge inn en student dersom vi ikke vet navnet på studenten
- eller verdien er nødvendig for at programmene som bruker databasen skal fungere riktig
- Vi kan da legge til en `NOT NULL`-skranke til kolonnen
- For eksempel:

```
CREATE TABLE Students (
    SID int,
    StdName text NOT NULL,
    StdBirthdate date
);
```

Skranker: UNIQUE

- Dersom vi ønsker at en kolonne aldri skal gjenta en verdi (altså inneholde duplikater)
- kan vi bruke UNIQUE-skranken
- For eksempel, student-IDen SID er unik
- Så for at databasen skal håndheve dette kan vi lage tabellen slik:

```
CREATE TABLE Students (
    SID int UNIQUE,
    StdName text NOT NULL,
    StdBirthdate date
);
```

Skranker: PRIMARY KEY

- I tillegg til å være unik, så må SID-verdien aldri være ukjent, ettersom det er primærnøkkelen i tabellen
- Så vi burde derfor ha både UNIQUE og NOT NULL, altså:

```
CREATE TABLE Students (
    SID int UNIQUE NOT NULL,
    StdName text NOT NULL,
    StdBirthdate date
);
```

- Men, det finnes også en egen skranke for dette, nemlig PRIMARY KEY som inneholder UNIQUE NOT NULL. Så,

```
CREATE TABLE Students (
    SID int PRIMARY KEY,
    StdName text NOT NULL,
    StdBirthdate date
);
```

er ekvivalent som over

- Merk, kan kun ha én PRIMARY KEY per tabell, må bruke UNIQUE NOT NULL dersom vi har flere kandidatnøkler

Alternativ syntaks for skranker

- Man kan også skrive skrankene til slutt, slik:

```
CREATE TABLE Students (
    SID int,
    StdName text NOT NULL,
    StdBirthdate date,
    CONSTRAINT sid_pk PRIMARY KEY (SID)
);
```

- Nå har skrankene navn (`sid_pk`, `name_nn`)
- Denne syntaksen er nødvendig om vi ønsker å ha skranker over flere kolonner
- F.eks. om kombinasjonen av `StdName` og `StdBirthdate` alltid er unik:

```
CREATE TABLE Students (
    SID int,
    StdName text NOT NULL,
    StdBirthdate date,
    CONSTRAINT sid_pk PRIMARY KEY (SID),
    CONSTRAINT name_bd_un UNIQUE (StdName, StdBirthdate)
);
```

Skranker: REFERENCES

- Det er vanlig i relasjonelle databaser at en kolonne refererer til en annen
- Fremmednøkler er eksempler på dette
- I slike tilfeller ønsker vi å begrense de tillatte verdiene i kolonnen til kun de som finnes i den den refererer til
- Dette kan gjøres med `REFERENCES`-skranken
- F.eks. for å lage `TakesCourse`-tabellen, kan vi gjøre følgende:

```
CREATE TABLE TakesCourse (
    SID int REFERENCES Students (SID),
    CID int REFERENCES Course (CID),
    Semester text
);
```

- Nå vil man kun kunne legge inn `SID`-verdier som allerede finnes i `Students(SID)` og kun `CID`-verdier som allerede er i `Courses(CID)`

Sette inn data

- For å sette inn data i en tabell bruker vi `INSERT`-kommandoen
- `INSERT` brukes på følgende måte:

```
INSERT INTO <tabell>
VALUES (<rad>),
       (<rad>),
       ...,
       (<rad>);
```

- Så, for å sette inn radene
  - `(0, 'Anna Consuma', '1978-10-09')`, og
  - `(1, 'Peter Young', '2009-03-01')`
- inn i `Students`, kan vi gjøre:

```
INSERT INTO Students
VALUES (0, 'Anna Consuma', '1978-10-09'),
       (1, 'Peter Young', '2009-03-01');
```

Andre måter å sette inn data

- Vi kan bruke resultatet fra en `SELECT`-spørring i stedet for `VALUES`
- For ekesmpel:

```
CREATE TABLE Students2018 (
    SID int PRIMARY KEY,
    StdName text NOT NULL
);

INSERT INTO Students2018
SELECT S.SID, S.StdName
  FROM Students AS S INNER JOIN TakesCourse AS T
       ON (S.SID = T.SID)
 WHERE T.Semester LIKE '%18';
```

Ny tabell basert på `SELECT` direkte

- Vi kan også kombinere de to kommandoene på forige slide slik:

```
CREATE TABLE Students2018 AS
SELECT S.SID, S.StdName
  FROM Students AS S INNER JOIN TakesCourse AS T
       ON (S.SID = T.SID)
 WHERE T.Semester LIKE '%18';
```

- Dette gir samme data, men merk at vi nå ikke har skrankene `PRIMARY KEY` og `NOT NULL`
- Disse må da legges til etterpå

Default-verdier

- Vi kan gi en kolonne en standard/default verdi
- Denne blir brukt dersom vi ikke oppgir en verdi for kolonnen
- For ekesmpel:

```
CREATE TABLE personer (
    pid int PRIMARY KEY,
    navn text NOT NULL,
    nationalitet text DEFAULT 'norge'
);

INSERT INTO personer
VALUES (1, 'carl', 'UK');

INSERT INTO personer(pid, navn) --eksplisitte kolonner
VALUES (2, 'kari');
```

vil gi

personer

pid	navn	nationalitet
1	Carl	UK
2	Kari	norge

SERIAL

- For primærnøkler som bare er heltall, så kan vi bruke `SERIAL`
- Dette gjør at databasen automatisk genererer unike heltall for hver rad
- Så med

```
CREATE TABLE Students (
    SID SERIAL PRIMARY KEY,    -- merk ingen type
    StdName text NOT NULL,
    StdBirthdate date
);

INSERT INTO Students(StdName, StdBirthdate) --eksplisitte kolonner
VALUES ('Anna Consuma', '1978-10-09'),
       ('Peter Young', '2009-03-01'),
       ('Anna Consuma', '1978-10-09');
```

vil vi få

Students

SID	StdName	StdBirthdate
1	Anna Consuma	1978-10-09
2	Peter Young	2009-03-01
3	Anna Consuma	1978-10-09

- Merk at man må være sikker på at radene nå faktisk representerer unike ting!

Hvor kommer data fra? (1)

Man skriver som oftest ikke `INSERT`-spørringer direkte

Den vanligste måten å få data inn i en database på er via programmer som eksekverer `INSERT`-spørringer (Se senere i kurset), f.eks.:

- data generert av simuleringer, analyse, osv.
- data skrevet av brukere via en nettside, brukergrensesnitt, osv.
- data fra sensorer (f.eks. værdata), nettsider (f.eks. aksjedata, klikk), osv.

Hvor kommer data fra? (2)

- Man kan også lese data direkte fra filer (f.eks. regneark eller CSV)
- I PostgreSQL har man `COPY`-kommandoen får å laste inn data fra CSV
- Følgende laster inn innholdet fra CSVen `~/documents/people.csv` (med separator `','` og null-verdi `''`) inn i tabellen `Persons`:

```
COPY persons
FROM '~/documents/people.csv' DELIMITER ',' NULL AS '';
```

- Merk, PostgreSQL krever at man er superuser for å lese filer av sikkerhetsgrunner
- Men man kan alltid lese fra Standard Input (`stdin`), f.eks. ved å eksekvere følgende (i Bash):

```
$ cat persons.csv | psql <flag> -c
    "COPY persons FROM stdin DELIMITER ',' NULL AS ''"
```

(hvor `flag` er de vanlige flaggene man bruker for innlogging til databasen)

- I Postgres finnes det også en egen `\copy`-kommando i `psql`

Eksempler på skrankeovertredelser (violations)

Som sagt tidliere, man har ikke lov til å overtre databaseskjemaet, så hvis vi har

```
CREATE TABLE Students (
    SID int PRIMARY KEY,
    StdName text NOT NULL,
    StdBirthdate date
);
```

så vil

- ```
  INSERT INTO Students
  VALUES (0, 'Anna Consuma', '1978-10-09', 1);
  ```

  gi `ERROR: INSERT has more expressions than target columns`
- ```
  INSERT INTO Students
  VALUES ('zero', 'Anna Consuma', '1978-10-09');
  ```

  gir `ERROR: invalid input syntax for integer: "zero"`
- ```
  INSERT INTO Students
  VALUES (0, NULL, '1978-10-09');
  ```

  gir `ERROR: null value in column "stdname"violates not-null constraint`
  ```

Eksempler på skrankeovertredelser

Og gitt:

Students

SID	StdName	StdBirthdate
0	Anna Consuma	1978-10-09
1	Anna Consuma	1978-10-09
2	Peter Young	2009-03-01
3	Carla Smith	1986-06-14
4	Sam Penny	NULL

Vil

```
INSERT INTO Students
VALUES (0, 'Peter Smith', '1938-11-11');
```

gi `ERROR: duplicate key value violates unique constraint "students_pkey"`

Slette ting

- For å slette ting (tabeller, skjemaer, brukere, osv.) fra databasen bruker vi `DROP`
- For å slette en tabell gjør vi `DROP TABLE <tablename>;`, f.eks.:

```
DROP TABLE Students;
```

- Tilsvarende for skjemaer, f.eks. `DROP SCHEMA northwind;`
- Av og til avhenger ting vi ønsker å slette på andre ting (f.eks. en tabell er avhengig av skjemaet den er i eller tabellene den refererer til)
- Vi kan ikke slette ting som andre ting avhenger av, uten å også slette disse
- For å slette en ting og alt som avhenger av den tingen kan vi bruke `CASCADE`
- Så for å slette `Students`-tabellen og alle tabeller som avhenger av denne (slik som `TakesCourse`):

```
DROP TABLE Students CASCADE;
```

Slette data

- For å slette rader fra en tabell bruker vi `DELETE`:

```
DELETE
  FROM <tabellnavn>
 WHERE <betingelse>
```

- Så sletting av alle studenter født etter `1990-01-01` gjøres slik:

```
DELETE
  FROM Students
 WHERE StdBirthdate > '1990-01-01'
```

Oppdatere ting

- For å oppdatere skjemaelementer bruker vi `ALTER`
- Mens data oppdateres med `UPDATE`
- Vi kan f.eks. gjøre følgende:

```
ALTER TABLE Students
RENAME TO UIOStudents;
```

  for å omdøpe `Students`-tabellen til `UIOStudents`

- Eller

```
ALTER TABLE Courses
ADD COLUMN Teacher text;
```

  for å legge til en kolonne `Teacher` med type `text` til `Courses`-tabellen

- Alt i skjemaet kan endres med `ALTER`, se PostgreSQL-siden[1] for en oversikt

[1] `https://www.postgresql.org/docs/current/sql-altertable.html`

Legge til skranker i ettertid

- Vi kan også legge til skranker etter at en tabell er laget
- Dette gjøres med kombinasjonen av `ALTER TABLE` og `ADD CONSTRAINT`
- For eksempel:

```
ALTER TABLE courses
ADD CONSTRAINT cid_pk PRIMARY KEY (cid);
```

Oppdatere data

- `UPDATE` lar oss oppdatere verdiene i en tabell:

```
UPDATE <tabellnavn>
   SET <oppdateringer>
 WHERE <betingelse>
```

  hvor `<oppdateringer>` er en liste med oppdateringer som blir eksekvert for hver rad som gjør `<betingelse>` sann
- For eksempel:

```
UPDATE Students
   SET StdBirthdate = '1987-10-03'
 WHERE StdName = 'Sam Penny'
```

  oppdaterer fødselsdatoen til studenten Sam Penny til `'1987-10-03'`
- Mens

```
UPDATE products
   SET unit_price = unit_price * 1.1
 WHERE quantity_per_unit LIKE '%bottles%'
```

  øker prisen med 10% på alle produkter som selges i flasker i `products`-tabellen

Takk for nå!

Neste video vil se på typesystemet i SQL og PostgreSQL.