IN2090 – Databaser og datamodellering
06 – SQL script og transaksjoner

Leif Harald Karlsen
leifhka@ifi.uio.no

Universitetet i Oslo

SQL-scripts

- Når man lager en database vil man vanligvis lage et script som inneholder alle SQL-kommandoene som lager skjemaene, tabellene, viewsene, osv.
- Man kan så heller eksekvere dette scriptet, fremfor å kjøre hver spørring manuelt
- Følgende er et eksempel-script som lager `Students`-databasen

```
CREATE SCHEMA uio;
CREATE TABLE uio.students (sid SERIAL PRIMARY KEY, stdname text NOT NULL, stdbrthdate date);
CREATE TABLE uio.courses (cid SERIAL PRIMARY KEY, coursename text NOT NULL, credits int);
CREATE TABLE uio.takescourse (cid int REFERENCES uio.courses(cid),
                              sid int REFERENCES uio.students(sid), semester text);
CREATE VIEW uio.studenttakescourse ( stdname text, coursename text )
AS SELECT s.stdname, s.coursename
   FROM uio.students AS s INNER JOIN uio.takescourse AS t ON (t.sid = t.sid)
        INNER JOIN uio.courses AS c ON (t.cid = c.cid);
INSERT INTO uio.students(stdname, stduiorthdate)
VALUES ('Anna Consuma', '1978-10-09'), ('Anna Consuma', '1978-10-09'),
       ('Peter Young', '2009-03-01'), ('Carla Smith', '1986-06-14');
INSERT INTO uio.courses(coursename, credits)
VALUES ('Data Management', 6), ('Finance', 10);
INSERT INTO uio.takescourse(sid, cid, semester)
VALUES (0,0,'A18'), (1,1,'S17'), (2,1,'S18'),
       (2,0,'S18'), (3,0,'A18');
```

- Et script `script.sql` kjøres ved `psql <flag> -f script.sql` eller fra psql-shellet ved `\i script.sql`

Dump

- Et databasesystem kan også lage et script som gjenskaper dens database(r)
- I PostgreSQL gjøres dette med et eget program `pg_dump` på følgende måte:

```
pg_dump [flag] db > fil
```

  hvor `[flag]` er de vanlige tilkobligsflaggene, `db` er navnet på databasen man vil dumpe, og `fil` er navnet på filen man vil skrive til.
- Andre databasesystemer har tilsvarende programmer
- Dette gjør det enkelt å duplisere eller dele databaser

SQL-scripts: Trygge kommandoer

- Dersom man forsøker å opprette en tabell som allerede finnes eller slette en tabell som ikke finnes så feiler kommandoen
- Dersom denne kommandoen er en del av en transaksjon, så feiler hele transaksjonen
- Dette kan hindres ved å bruke `IF EXISTS` og `IF NOT EXISTS` i kommandoene
- For eksempel:

```
CREATE TABLE IF NOT EXISTS persons(name text, born date); -- Lager ny tabell
CREATE TABLE IF NOT EXISTS persons(name text, born date); -- Gir ingen error/lykkes
CREATE TABLE persons(name text, born date);               -- Gir ERROR og feiler
DROP TABLE IF EXISTS persons;                             -- Sletter tabellen
DROP TABLE IF EXISTS persons;                             -- Gir ingen error/lykkes
DROP TABLE persons;                                       -- Gir error, og feiler
```

- F.eks. nyttig dersom man oppdaterer scriptet som har generert en database
- Kan da kjøre scriptet for å kun få utført oppdateringene

SQL-scripts: Meta-kommandoer

- I et SQL-script har man også en del kommandoer som ikke er en del av SQL-språket
- F.eks. printe en beskjed, lage og gi verdier til variable, be om input fra en bruker, osv.
- Disse kommandoene har forskjellig syntaks fra RDBMS til RDBMS
- I PostgreSQL kan man printe en beskjed ved å bruke `\echo`, f.eks.

```
\echo 'This is a message'
```

  og brukes for å gi informasjon mens scriptet kjører (progresjon ol.)
- Dersom en konstant verdi brukes mye i et script kan man gi den et navn med `\set`, f.eks.

```
\set val 42
INSERT INTO meaning_of_life VALUES (:val);
```

- Merk kolonet foran navnet når verdien brukes
- Disse kan også brukes i `psql` direkte

Transaksjoner

- Når man oppdaterer databasen og noe går galt underveis ønsker man ofte at ingen av oppdateringene skal ha skjedd
- F.eks. kan man få delvis lagde tabeller, delvis insatt data, osv.
- For eksempel, se for dere følgende bank-overføring:

```
UPDATE balances
SET balance = balance - 100
WHERE id = 1;

UPDATE balances
SET balance = balance + 100
WHERE id = 2;
```

- Dersom den første oppdateringen feiler (f.eks. fordi `balance < 100` men vi har en skranke `balances >= 0`) vil vi ikke at den andre skal utføres
- Det samme gjelder dersom vi får en feil midt i et SQL-script
- Vi pakker derfor inn oppdateringer som skal utføres som en ”enhet” i transaksjoner

Transaksjoner – Syntaks

Transaksjoner omsluttes av `BEGIN` og `COMMIT` slik:

```
BEGIN;

UPDATE balances
SET balance = balance - 100
WHERE id = 1;

UPDATE balances
SET balance = balance + 100
WHERE id = 2;

COMMIT;
```

ACID

For at transaksjoner skal fungere som forventet, tilfredstiller de fire kriterier:

- **A**tomicity – Alle kommandoene i en transaksjon ansees som en enhet, og enten skal alle kommandoer lykkes, eller så skal alle kommandoer feile (feiler én så feiler alle)
- **C**oncistency – Dersom en transaksjon lykkes skal databasen ende opp i en konsistent tilstand (altså ingen skranker skal være brutt)
- **I**solation – Transaksjoner skal kunne kjøres i parallell, men resultatet skal da være likt som om transaksjonene ble kjørt sekvensielt
- **D**urability – Etter at en transaksjon lykkes og har utført endringer på databasen, skal disse endringene alltid være utført (f.eks. dersom systemet restartes skal databasen fortsatt ha de samme endringene utført)

Takk for nå!

Neste uke skal vi se på normalformer.