Datamanipulering (Views og transaksjoner ligger i mappen)

CREATE-kommandoen:

Hva gjør kommandoen?

Sørger for at vi kan lage tabeller, brukere, skjemaer.

Hvordan utføres kommandoen?

CREATE TABLE <tabellnavn> (<kolonner>);

- Hvor <tabellnavn> er et tabellnavn altså Person, Film osv og <kolonner> er kolonne-deklarering som Name, FilmId
- Gjennom kolonne-deklarering så har man et kolonnenavn og en type for det kolonnenavnet, akkurat som ved java når vi skal angi en streng så sier vi <String navn = “” >

Eksempel ved CREATE:

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

Skranker: NOT NULL

Noen tilfeller ønsker vi å ikke tillate NULL-verdier i en kolonne, hvor vi ønsker at ved å utføre CREATE, så skal alle kolonner fylles eller at den gitte kolonnen er nødvendig for å lage tabellen. Da kan vi bruke NOT null-skranke til kolonnen som sier at, StdName må fylles ut, vi kan ikke tillate NULL-verdier for denne.

Eksempel ved NOT NULL:

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

Skranker: UNIQUE

UNIQUE sørger for at vi unngår duplikater, fordi SQL oppbevarer multimengder så derfor godtar den duplikater. For å unngå dette så bruker vi UNIQUE-skranken som kan sees som en primærnøkkel som er unik.

Eksempel ved UNIQUE:

```
CREATE TABLE Students (
        SID int UNIQUE(Student-id er unik) ,
        StdName text NOT NULL,
        StdBirthdate date
);
```

Skranker: PRIMARY KEY

For å koble både UNIQUE og NOT NULL-skranken, så kan vi bruke PRIMARY KEY og grunnen skyldes av at PRIMARY KEY sørger for at en kolonne deklarasjon er både unik men også aldri ukjent(altså NULL)

Eksempel ved **PRIMARY KEY:**

```
CREATE TABLE Students (
        SID int PRIMARY KEY(Student id er unik og NOT NULL) ,
        StdName text NOT NULL,
        StdBirthdate date
);
```

Skranker: REFERENCES

I relasjonelle databaser, så er det vanlig at en kolonne referer til en annen hvor frememdnøkler er eksempler på dette(Personer relateres til Hus(obligen)). Dette kan gjøres i SQL gjennom REFERENCES-skranken.

Eksempel ved REFERENCES

```
CREATE TABLE TakesCourse (
        SID int REFERENCES Students (SID) (Referer til Student tabellen)
        CIT int REFERENCES Course (CID) (Referer til Course tabellen)
        Semester text
);
```

INSERT-Kommandoen:

Hva gjør kommandoen?

Gjør at vi kan sette inn data i en tabell

Hvordan utføres kommandoen?

INSERT INTO <tabell>
VALUES (<rad>),
(<rad>), ….

Eksempel ved INSERT:

Ønsker å sette inn følgende rader i Students:
(0, ‘Anna Consuma’ , ‘1978-10-09’) og (1, ‘Peter Young’, ‘2009-03-01’)
INSERT INTO Students
VALUES (0, ‘Anna Consuma’ , ‘1978-10-09’), (1, ‘Peter Young’, ‘2009-03-01’)

Skranke: DEFAULT

- Vi kan gi en kolonne en standard/default verdi, verdien blir da brukt hvis vi ikke angir en verdi for denne kolonnen.

Eksempel ved INSERT:

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

INSERT INTO personer
VALUES  (1, ‘Carl’, ‘UK’) (Dette vil funke siden vi endrer verdien for nationalitet text kolonnen)

INSERT INTO personer(pid, navn)
VALUES  (1, ‘Carl’) (Nå vil “Carl” få “norge” som sin nationalitet siden kolonnen ikke ble fylt inn)
```

SERIAL

Hva er SERIAL?

SERIAL er primærnøkler som er ment kun for heltall. Dette gjør at databasen automatisk genrer unike heltall for hver rad. Starter fra 1, så øker det med 1 for x antall som settes inn i tabellen.

Eksempel ved SERIAL:

CREATE TABLE Students (

SID SERIAL PRIMARY KEY (Slipper å angi verdi for kolonnen da den settes automatisk),

StdName text NOT NULL,

StdBirthdate date

);

Hvordan det sees ut:

```
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

Skrankeovertredelser(violations):

- Det er viktig å overholde skrankene da disse er strenge som i python og java. F.eks, hvis variablen er angitt til å være en streng, så vil det oppstå en kompilatorfeil ved å legge til et tall ved variabelen i java. Slik er det også i SQL, hvor om kolonnen skal oppbevare 'text', så skal den oppbevare en streng, ikke tall. Vi må også overholde NOT NULL skranken også så hvis jeg skriver en kolonne med NULL , når kolonnen er definert med skranke NOT NULL, så vil det oppstå feil.
- Feil kan også oppstå hvis vi skriver flere verdier for hva som er det opprinnelig antall kolonner altså har vi en tabell som har 2 kolonner så vil det oppstå feil hvis vi INSERT, for 3 kolonner. Akkurat som når vi må holde oss til parameterne i java og python

DROP-kommandoen:

Hva gjør kommandoen?

Gjør at vi kan slette tabeller, skjemare, brukere fra en database.

Hvordan utføres kommandoen?

DROP TABLE Students(Da er Student tabellen slettet)

Hva hvis en tabell vi ønsker å slette, er avhengig av en annen tabell?

DROP TABLE Students **CASCADE (**Cascade gjør at vi sletter alle tabeller som er avhengig av Students og selve Students tabellen også)

<u>DELETE-Kommandoen:</u>

Hva gjør kommandoen?

Denne er ganske selvforklarende med kommandoen gjør at vi kan slette rader fra en tabell.

Hvordan utføres kommandoen?

DELETE
FROM <tabellnavn>
WHERE <betingelse>

Eksempel ved DELETE:

DELETE
FROM Students
WHERE StdBirthdate > '1990-01-01'(Sletter alle studenter som er født etter 1990)

<u>Oppdatere ting:</u>

- Gjennom kommandoen ALTER så kan vi oppdatere skjemaelementer
- Mens data i kolonner oppdateres med UPDATE

Eksempler ved ALTER:

ALTER TABLE Students
RENAME TO UIOStudents (Endret tabellen til å hete UIOSTUDENTS)

Eksempler ved UPDATE:

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

Oppdaterer fødselsdatoen til Sam Penny

Typer og Skranking

Casting

Casting som i java, er mulig i SQL.

- Vi kan benytte ‘cast’ til å utføre denne operasjonen så - cast(‘1’ AS int)
- Angi typen til å bestemme hva vi er ute etter så - int ‘1’
- Eller gjennom følgende måte - ‘1’::int

Numeriske typer

Name	Storage Size	Description	Range
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807
decimal	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	variable-precision, inexact	6 decimal digits precision
double precision	8 bytes	variable-precision, inexact	15 decimal digits precision
smallserial	2 bytes	small autoincrementing integer	1 to 32767
serial	4 bytes	autoincrementing integer	1 to 2147483647
bigserial	8 bytes	large autoincrementing integer	1 to 9223372036854775807

Funksjoner og operatorer

- ceil, floor,round - runder opp/ned
- sqrt power - kvadratrot og potens-funksjon
- abs - absoluttverdi (tenkes som FINAL i Java)
- sin, cos, tan - Trigonometriske funksjoner
- random() - tilfeldige tall

Eksempel ved bruk av funksjoner og operatorer:

Ønsker å pakke bestilte varer i kubeformede bokser for hvert produkt, så vil finne 3. roten av antall bestilte varer for de som er bestilt, samt ca. pris avrundet til nærmeste heltall på det som er bestilt.

```
SELECT product_name,
       ceil(power(units_on_order, 1.0/3)) AS box_size,
       round(unit_price * units_on_order) AS ca_price
FROM products
WHERE units_on_order > 0;
```

```
northwind=> SELECT product_name, ceil(power(units_on_order, 1.0/3))
FROM products
WHERE units_on_order > 0;
       product_name        | ceil
---------------------------+------
 Chang                     |    4
 Aniseed Syrup             |    5
 Queso Cabrales            |    4
 Sir Rodney's Scones       |    4
 Gorgonzola Telino         |    5
 Mascarpone Fabioli        |    4
 Gravad lax                |    4
 Ipoh Coffee               |    3
 Rogede sild               |    5
 Chocolade                 |    5
 Maxilaku                  |    4
 Gnocchi di nonna Alice    |    3
 Wimmers gute Semmelknödel |    5
 Louisiana Hot Spiced Okra |    5
 Scottish Longbreads       |    3
 Outback Lager             |    3
 Longlife Tofu             |    3
(17 rows)
```

Streng-typer

Name	Description
character varying(*n*), varchar(*n*)	variable-length with limit
character(*n*), char(*n*)	fixed-length, blank padded
text	variable unlimited length

Funksjoner og operatorer

- position(sub in str) finner posisjonen til ‘sub’ i ‘str’
- substring(str from s for e) finner substrengen i ‘str’ som starter på indeksen ‘s’ og har lengde ‘e’
- strpos(str, sub), uttrykkes i Postgres, samme som position
- substr(str, s , e) uttrykkes i Postgres, samme som substring

Eksempel ved bruk av funksjoner og operatorer:

Lag en pen melding på formen ”Product [name] costs [price] per [quantity]” men hvor ”glasses” er byttet ut med ”jars”.

```
SELECT replace(prod_desc, 'glasses', 'jars') AS prod_desc
FROM (
  SELECT format('Product %s costs %s per %s',
                product_name,
                unit_price,
                quantity_per_unit) AS prod_desc
  FROM products
) AS t;
```

```
                          prod_desc
-------------------------------------------------------------------------
Product Chai costs 18 per 10 boxes x 30 bags.
Product Chang costs 19 per 24 - 12 oz bottles.
Product Aniseed Syrup costs 10 per 12 - 550 ml bottles.
Product Chef Anton's Cajun Seasoning costs 22 per 48 - 6 oz jars.
Product Chef Anton's Gumbo Mix costs 21.35 per 36 boxes.
Product Grandma's Boysenberry Spread costs 25 per 12 - 8 oz jars.
Product Uncle Bob's Organic Dried Pears costs 30 per 12 - 1 lb pkgs..
Product Northwoods Cranberry Sauce costs 40 per 12 - 12 oz jars.
Product Mishi Kobe Niku costs 97 per 18 - 500 g pkgs..
Product Ikura costs 31 per 12 - 200 ml jars.
Product Queso Cabrales costs 21 per 1 kg pkg..
Product Queso Manchego La Pastora costs 38 per 10 - 500 g pkgs..
Product Konbu costs 6 per 2 kg box.
Product Tofu costs 23.25 per 40 - 100 g pkgs..
Product Genen Shouyu costs 13 per 24 - 250 ml bottles.
Product Pavlova costs 17.45 per 32 - 500 g boxes.
Product Alice Mutton costs 39 per 20 - 1 kg tins.
Product Carnarvon Tigers costs 62.5 per 16 kg pkg..
Product Teatime Chocolate Biscuits costs 9.2 per 10 boxes x 12 pieces.
Product Sir Rodney's Marmalade costs 81 per 30 gift boxes.
Product Sir Rodney's Scones costs 10 per 24 pkgs. x 4 pieces.
Product Gustaf's Knäckebröd costs 21 per 24 - 500 g pkgs..
Product Tunnbröd costs 9 per 12 - 250 g pkgs..
Product Guaraná Fantástica costs 4.5 per 12 - 355 ml cans.
Product NuNuCa Nuß-Nougat-Creme costs 14 per 20 - 450 g jars.
```

Tidstyper

Name	Storage Size	Description	Low Value	High Value	Resolution
`timestamp [ (p) ] [ without time zone ]`	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond
`timestamp [ (p) ] with time zone`	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond
`date`	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
`time [ (p) ] [ without time zone ]`	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond
`time [ (p) ] with time zone`	12 bytes	time of day (no date), with time zone	00:00:00+1459	24:00:00-1459	1 microsecond
`interval [ fields ] [ (p) ]`	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond

Datotyper

Funksjoner og operatorer

- Man kan bruke + og – mellom tidstyper, f.eks.:

```
date '2001-09-28' + interval '1 hour' = timestamp '2001-09-28 01:00:00'
date '2001-09-28' - interval '1 hour' = timestamp '2001-09-27 23:00:00'
date '2001-09-28' + 7                 = date '2001-10-05'
```

- Tilsvarende som for strengtyper har SQL noe rar syntaks for å manipulere tidstyper
- Man bruker `extract(part from value)` for å hente ut `part`-delen av `value`
- For eksempel:

```
extract(hour from timestamp '2001-02-16 20:38:40') = 20
extract(year from timestamp '2001-02-16 20:38:40') = 2001
extract(week from timestamp '2001-02-16 20:38:40') = 7
```

- Man kan også bruke `date_part(part, timestamp)`, f.eks.:

```
date_part('hour', timestamp '2001-02-16 20:38:40') = 20
date_part('year', timestamp '2001-02-16 20:38:40') = 2001
date_part('week', timestamp '2001-02-16 20:38:40') = 7
```

- Man har også funksjoner som gir dagens dato, ol.
- For eksempel vil `now()` gi et `timestamp` med nåværende tidspunkt
- Mens `current_date` er en konstant som inneholder dagens dato
- For å lage en dato kan man også bruke
  `make_date(year int, month int, day int)`
- For tidstyper kan man også bruke `OVERLAPS` mellom par, f.eks.:

```
SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Result: true
```

Eksempel ved bruk av funksjoner og operatorer:

Finn gjennomsnittlig tid fra bestilling av en ordre til den må være levert, i perioden fra 1997 frem til nå.

```
SELECT avg(required_date::timestamp - order_date::timestamp) AS ship_to_req
  FROM orders
 WHERE (order_date, required_date) OVERLAPS (make_date(1997, 1, 1), now());
```

```
northwind=> SELECT avg(required_date - order_date) FROM orders;
         avg
---------------------
 27.8819277108433735
(1 row)

northwind=> SELECT avg(required_date::timestamp - order_date::timestamp) FROM orders;
           avg
--------------------------
 27 days 21:09:58.554217
(1 row)

northwind=> SELECT avg(required_date::timestamp - order_date::timestamp) FROM orders
northwind-> WHERE (order_date, required_date) OVERLAPS (make_date(1997, 1, 1), now());
          avg
-----------------------
 27 days 22:05:46.74221
(1 row)
```

Skranker: CHECK

Denne skranken lar oss bruke et generelt uttrykk for å avgjøre om verdier kan settes inn i kolonnen eller ikke. F.eks at en kolonne kun godtar aldre som er under 18 år.

Eksempler ved CHECK:

```
CREATE TABLE students(
        sid int PRIMARY KEY,
        name text NOT NULL,
        birthdate date check (birthdate < date ‘2010-01-01’) (Godtar studenter født før 2010)
)
```