IN2090 – Databaser og datamodellering

04 – Typer og skranker

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Databasers typesystem

- De fleste relasjonelle databaser har et strengt typesystem
- Alle kolonner må ha en tilhørende type, og kun verdier av denne typen kan legges i kolonnen
- I tillegg har man som regel mange funksjoner man kan bruke for å manipulere verdier
- Også disse har en type-signatur som angir lovlig type til argumentene og typen til returnerte verdier

Hvilke datatyper har vi

- PostgreSQL har mange innebygde typer, se:

  `https://www.postgresql.org/docs/current/datatype.html`

- De mest brukte er:
  - Numeriske typer
  - Strengtyper
  - Dato- og tidstyper
  - Boolean
  - Array
- Merk at man også kan lage egne typer (utenfor pensum)
- Man kan også lage egne funksjoner (også utenfor pensum)

SQL-standarden vs. RDBMSer

- Noen typer er en del av SQL-standarden, men mange vanlig brukte typer er ikke
- Også mange funksjoner på SQL-standard-typer som ikke selv er en del av standarden
- Likevel stor likhet mellom ulike RDBMS, ofte kun syntaktiske forskjeller

Casting

- Dersom vi skriver `'1'` er dette en utypet literal for databasen
- Databasen forsøker så å *caste* denne verdien til det den brukes som
- Så dersom vi har tabellene `person(id int)` og `notes(note text)` vil

```
INSERT INTO person VALUES ('1');
INSERT INTO notes VALUES ('1');
```

  vil den første bli en `int` mens den andre `text`
- Vi kan også *caste* ekplisitt, og det er flere måter å gjøre det på
  - Med `::` – `'1'::int`
  - Med `cast`-nøkkelordet – `cast('1' AS int)`
  - Eller å sette typen først – `int '1'`
- Merk: Vi skriver alltid inn utypede litteraler til databasen, enten caster den eller så caster vi

Numeriske typer

Vi har følgende numeriske typer[1]:

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

Større presisjon fører til høyere forbruk av lagringsplass og bergninger/spørringer tar lengre tid.

[1] `https://www.postgresql.org/docs/current/datatype-numeric.html`

Numeriske typer – Funksjoner og operatorer

Har de fleste vanlige matematiske funksjoner og operatorer:

- `ceil`, `floor` – runder opp/ned
- `sqrt`, `power` – kvaderatrot og potens-funksjon
- `abs` – absoluttverdi
- `sin`, `cos`, `tan` – trigonometriske funksjoner
- `random()` – tilfeldig tall
- ...

Se følgende for alle:

`https://www.postgresql.org/docs/11/functions-math.html`

Numeriske typer: eksempel-spørring

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;
```

Streng-typer

Vi har følgende strengtyper[2]:

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

- Insetting av strenger som er lengre enn `n` i kolonner med type `varchar(n)` eller `char(n)` gir error
- `text` er ikke en del av SQL-standarden, men nesten alle RDBMS implementerer en slik type
- I PostgreSQL er det ingen fordel å bruke `varchar(n)` eller `char(n)` med hensyn på effektivitet eller minne
- Bruk `varchar(n)` eller `char(n)` kun som skranker (begrense lovlig lengde)

[2] `https://www.postgresql.org/docs/current/datatype-character.html`

Streng-typer – Funksjoner og operatorer

- SQL-standarden har noe rar syntaks for en del streng-manipulering
- For eksempel:
  - `position(sub in str)` finner posisjonen til `sub` i `str`
  - `substring(str from s for e)` finner substrengen i `str` som starter på indeks `s` og har lengde `e`
- Postgres støtter disse, men har mer naturlige varianter, f.eks.:
  - `strpos(str, sub)`, samme som `position`-uttrykket over
  - `substr(str, s, e)`, samme som `substring`-uttrykket over
- Ellers, se følgende for vanlige strengmanipulerings-funksjoner

  `https://www.postgresql.org/docs/11/functions-string.html`

Strengtyper: eksempel-spørring

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;
```

Tidstyper

Vi har følgende tidstyper[3]:

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

hvor `p` er antall desimaler for sekunder.

[3] `https://www.postgresql.org/docs/current/datatype-datetime.html`

Formater for datoer

Datoer

Example	Description
1999-01-08	ISO 8601; January 8 in any mode (recommended format)
January 8, 1999	unambiguous in any `datestyle` input mode
1/8/1999	January 8 in `MDY` mode; August 1 in `DMY` mode
1/18/1999	January 18 in `MDY` mode; rejected in other modes
01/02/03	January 2, 2003 in `MDY` mode; February 1, 2003 in `DMY` mode; February 3, 2001 in `YMD` mode
1999-Jan-08	January 8 in any mode
Jan-08-1999	January 8 in any mode
08-Jan-1999	January 8 in any mode
99-Jan-08	January 8 in `YMD` mode, else error
08-Jan-99	January 8, except error in `YMD` mode
Jan-08-99	January 8, except error in `YMD` mode
19990108	ISO 8601; January 8, 1999 in any mode
990108	ISO 8601; January 8, 1999 in any mode
1999.008	year and day of year
J2451187	Julian date
January 8, 99 BC	year 99 BC

Formater for tid

Tid

Example	Description
04:05:06.789	ISO 8601
04:05:06	ISO 8601
04:05	ISO 8601
040506	ISO 8601
04:05 AM	same as 04:05; AM does not affect value
04:05 PM	same as 16:05; input hour must be <= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	time zone specified by abbreviation
2003-04-12 04:05:06 America/New_York	time zone specified by full name

Tidstyper – 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
```

Tidstyper – Andre funksjoner og operatorer

- 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
```

- Ellers, se

  `https://www.postgresql.org/docs/11/functions-datetime.html`

Tidstyper: eksempel-spørring

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());
```

Andre nyttige typer

PostgreSQL støtter også

- Array
- XML
- JSON
- Bitstrenger
- Nettverks-adress-typer
- Geometriske typer
- ...

`https://www.postgresql.org/docs/11/datatype.html`

Check-skranker

- Hittil har vi sett på følgende varianter av skranker:
  - `UNIQUE`
  - `NOT NULL`
  - `PRIMARY KEY`
  - `REFERENCES`
  - typer
- `CHECK` er en annen generell og nyttig skranke
- `CHECK` lar oss bruke et generelt uttrykk for å avgjøre om verdier kan settes inn i kolonnen eller ikke
- For eksempel, med

```
CREATE TABLE students (
  sid int PRIMARY KEY,
  name text NOT NULL,
  birthdate date CHECK (birthdate < date '2010-01-01')
);
```

  kan man kun legge inn studenter med fødselsdato før år 2010.
- Merk: Man kan fortsatt sette inn `NULL`

Takk for nå!

Neste video vil se på views.