ORDMS

wim.bertels@ucll.be

ORDBMS : kenmerken

- Uitbreiding van de basis datatypes
    - Complexe objecten
- Overerving
    - Overerving van gegevens
    - Overerving van types
- Regels (bv)
    - Update-update regel
    - Query-update regel
    - Update-query regel
    - Query-query regel
- ..

Inleiding

- Complexere en meer specialistische datatypes

=> zelf gedefinieerde datatypes

Zelf definiëren van datatypes

- CREATE TYPE : creëert datatype
- DROP TYPE : verwijdert datatype
- ALTER TYPE : haalbare aanpassingen
- Bv

```
CREATE TYPE coordinaat
    AS (x int, y int);
-- composiet
```

Toegang tot datatypes

- Eigenaar van datatype : degene die type creëert
- Degene die type wil gebruiken, moet machtiging krijgen :

```
grant       usage
on type     geldbedrag
to          Jim
```

- Verwijderen van machtiging :

```
revoke      usage
on type     geldbedrag
from        Jim
```

- Zie ook notas https://www.postgresql.org/docs/current/sql-createtype.html

Casting van waardes

- Casting : om verschillende datatypes te kunnen vergelijken
- Destructor : transformeert zelfgedefinieerd datatype naar basisdatatype
- Constructor : transformeert basisdatatype naar zelfgedefinieerd datatype

Voorbeelden

- Vb.


```
select  *
from   boetes
where bedrag > geldbedrag(50) ;

select  *
from   boetes
where decimal(bedrag) > 50 ;

insert into plaats (waar,naam) values
  (coordinaat(12,57), ‘niverance’) ;
```

Zelf definiëren van operatoren

- Voorafgaandelijke opmerkingen :
  - Operatoren zijn toegevoegd voor het gemak
  - Bij elk basis-datatype horen operatoren
  - Operatoren hebben een betekenis alnaargelang het datatype
  - Definiëren van operatoren op zelfgedefinieerde datatypes ~ operaties op onderliggend type

Voorbeeld

- Definiëren van operatoren en bijhorende functies :

```
CREATE OR REPLACE FUNCTION plus_vector(in_c1 coordinaat, in_c2 coordinaat)
        RETURNS coordinaat AS
$$
DECLARE
   result coordinaat;
BEGIN
   result.x := in_c1.x + in_c2.x;
   result.y := in_c1.y + in_c2.y;
   RETURN result;
END;
$$
LANGUAGE plpgsql;
```

Voorbeeld

- Bijhorende operator :

```
create operator + (
        leftarg = coordinaat,
        rightarg = coordinaat,
        function = plus_vector,
        commutator = +
);
```

Voorbeeld

- Gebruik :

```
select (1,10)::coordinaat + (20,200)::coordinaat as vector_som;
select cast((1,10) as coordinaat) + cast((20,200) as coordinaat) as vector_som;
```

```
 vector_som
-----------------
 (21,210)
(1 row)
```

Opaque-datatype

- Def. : datatype dat niet afhankelijk is van een basisdatatype
- Definiëren van functies om hiermee te werken is noodzakelijk
- Vb.

```
create type tweedim (internallength = 4) ;
-- base types in postgresql
```

CREATE TYPE

- ISO :
    - Composiet (bv coordinaat)
    - Opaque (pg : base)
- Niet aanwezig in ISO, pg specifiek :
    - Enum (opsomming, M/V/X)
    - Range (bereik)
    - + Array

Named row-datatype

- Def. : groeperen van waarden die logisch bij elkaar horen (idem composiet)
- Vb.

```
create type adres as
        (straat      char(15)  not null,
         huisnr      char(4)           ,
         postcode    char(6)           ,
         plaats      char(10)  not null) ;
-- postgresql : zonder not null constraint
```

Voorbeeld

```
create table spelers
        (spelersnr       integer    primary key,
         …
         woonadres      adres                    ,
         postadres       adres                   ,
         vakantieadres  adres                    ,
         …                                  ) ;


select   spelersnr, woonadres
from     spelers
where    (woonadres).plaats = ‘Leuven’ ;
```

Unnamed row-datatype

Unnamed row-datatype : : groeperen van waarden die logisch bij elkaar horen zonder een naam te geven

•Vb.

```
create table spelers
        (spelersnr      smallint        primary key,
         …
         woonadres      row  (straat    char(15)        not null,
                             huisnr     char(4)                 ,
                             postcode   char(6)                 ,
                             plaats     char(10)        not null),
        telefoon        char(10) ,
        …                           ) ;
```

-- postgresql : unnamed rows worden enkel als input toegelaten, niet in DDL

Getypeerde tabel

- Def. : een datatype toekennen aan een tabel
- Eenvoudig om gelijkende tabellen te creëren
- Vb.

```
create type t_spelers as
        (spelersnr  integer     not null,
        naam        char(15)    not null,
        …
        bondsnr     char(4));

create table spelers of t_spelers
                    (primary key spelersnr) ;
```

Tabel > Tabel

- Basis structuur een bestaande tabel overnemen (fk’s verhuizen niet mee!):
- Vb.

```
create table nieuwe_klanten
        (like ruimtereizen.klanten including indexes);


        -- zie including optie voor verschillende opties
```

Integriteitsregels op datatypes

- Beperkingen op de toegestane waardes
- Vb.

```
create type aantal_sets as smallint
                    check (value in (0, 1, 2, 3));
create table wedstrijden
    (wedstrijdnr   integer          primary key,
    teamnr          integer         not null,
    spelersnr       integer         not null,
    gewonnen        aantal_sets     not null,
    verloren        aantal_sets     not null) ;
-- Wat is een DOMAIN in RDBMS?
-- PG : via enum
```

Sleutels en indexen

- Is volledig analoog bij zelfgedefinieerde datatypes
    - Operator nodig voor index moet kunnen gebruikt worden
- Bij named row-datatypes (composiet):
    - op de volledige waarde
    - op een deel ervan
- CREATE INDEX coordinaat_x_idx ON your_table ((coord_column).x);

Overerving, references, collecties

1. Overerving van datatypes
2. Koppelen van tabellen via rij-identificaties
3. Collecties
4. Overerving van tabellen
5. Regels

Overerving van datatypes

- Def. : alle eigenschappen van één datatype worden overgeërfd door een ander (supertype en subtype)
- Vb.

```
create type adres as
        (straat         char(15)                not null,
                huisnr          char(4),
                poscode         char(6),
                plaats          char(10)        not null);

create type buitenlands_adres as
        (land           char(20)        not null) under adres ;

-- postgresql : ..of rechtstreeks of like in base type..
```

Koppelen van tabellen

- In OO-DB : alle rijen hebben een unieke identificatie (door het systeem)
- REF : om identificatie op te vragen
- Vb.

```
select ref(spelers)
from spelers
where spelersnr = 6 ;
```

-- postgresql : eventueel via oids, weinig gebruikt

Voorbeeld

- REF : om tabellen te koppelen (niet in pg)
- Vb.

```
create table teams
            (teamnr      smallint          primary key,
             speler      ref(spelers)      not null,
             divisie     char(6)           not null) ;


insert into teams (teamnr, speler, divisie)
values (3, (select ref(spelers)
            from   spelers
                   where spelersnr = 112), 'ere') ;


select teamnr, speler.naam
from teams;
```

Pro-Contra

- Voordelen :
  - Altijd het juiste datatype bij de refererende sleutel
  - Indien primary keys breed zijn, bespaart het werken met reference-kolommen opslagruimte
  - Bij wijzigen van primary keys wordt geen tijd verloren door het wijzigen van de refererende sleutel
  - Bepaalde selects worden eenvoudiger
- Nadelen :
  - Bepaalde mutaties zijn moeilijker te definiëren
  - References werken in één richting
  - Bij DB-ontwerp krijgt men meerdere keuzes, dit wordt dus moeilijker
  - References kunnen de integriteit van de gegevens niet bewaken zoals refererende sleutels dat kunnen

Collecties

- Collecties : verzameling waardes in één cel
- Vb.

```
create table spelers
      (spelersnr  smallint      primary key,
       …
       telefoons  setof(char(13)),
       bondsnr    char(4)                          );

insert into spelers (spelersnr, …, telefoons, …) values
    (213, .., {‘016-342654’, ’0475-654387’}, ..);

select spelersnr
      from spelers
      where ‘016-342654’ in (telefoons);

-- pg: meestal via arrays[] ..
```

Overerving van tabellen

- Def. : alle eigenschappen van één tabel worden overgeërfd door een andere tabel (supertabel – subtabel)
- Beperkingen :
  - Geen cyclische structuur

Voorbeeld

```
create table spelers as
                (spelersnr      smallint        not null,
                 naam           char(15)        not null,
                 …
                 bondsnr        char(4));
create table oude_spelers
               (vertrokken       date            not null)
                inherits (spelers, okra);


SELECT    *
FROM      (ONLY) spelers;
-- let op met inserts, constraints ..
-- postgresql
```

RULES

- Gaan verder dan triggers : SELECT, INSERT, UPDATE, DELETE
- FKs >> triggered
- Updateble views
    - rules of triggers
- Maar voorzichtig, systeemlogica

CREATE RULE (bv)

```
CREATE RULE "NeKeerIetsAnders" AS
    ON SELECT TO wedstrijden
        WHERE    spelersnr = 7
    DO INSTEAD
        SELECT 'eerst drie toerkes rond tafel lopen
            en dan nog eens proberen';
```