TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik
Professur Datenverwaltungssysteme

Advanced Management of Data
Exercise 3 Topic 3:
Object-Relational Database Systems

Chemnitz · 22. Januar 2018 · Dipl.-Inf. Daniel Richter

Arrays

- Task: create new types `multiemailtype` and `multitelephonetype`, that can store more than one entry and find a way to enforce the constraints of `emailtype` on the entries of `multiemailtype`

Array Types

```
CREATE FUNCTION enforceOneAtInTheMiddle(emails VARCHAR[]) RETURNS BOOLEAN AS $$
DECLARE
  email VARCHAR;
BEGIN
  FOREACH email IN ARRAY emails LOOP
    IF email !~ '^[^@]+@[^@]+$' THEN
      RETURN FALSE;
    END IF;
  END LOOP;
  RETURN TRUE;
END;
$$ LANGUAGE plpgsql;

CREATE DOMAIN multiemailtype AS VARCHAR[] CHECK(enforceOneAtInTheMiddle(value));

CREATE DOMAIN multitelephonetype AS VARCHAR[];
```

- Task: exchange the old types with the new multi-types in `persons` and add +49 234 56789 and Max@example.com to Max Mustermann

Arrays
Appended

```
ALTER TABLE persons
  ALTER email TYPE multiemailtype
    USING ARRAY[email]::multiemailtype,
  ALTER telephone TYPE multitelephonetype
    USING ARRAY[telephone]::multitelephonetype;
```

- now you could add new email and telephone like

```
UPDATE persons
  SET email = email || 'Max@example.com'::VARCHAR,          -- use simple array concatenation
      telephone = array_append(telephone, '+49 234 56789') -- or a function doing the same
  WHERE name = ('Max', 'Mustermann')::nametype;
```

- as a reminder: as the table `professors` is inherited from `persons`, the columns are changed there, too

Arrays
NF1 ↔ NF^2

- consider the following table in First Normal Form that stores information about books

```
CREATE TABLE book_1nf (title VARCHAR, author VARCHAR, year INT, month VARCHAR, day INT, keyword VARCHAR);

INSERT INTO book_1nf VALUES
  ('Selling', 'Stein', 2009, 'April', 1, 'Profit'), ('Selling', 'Stein', 2009, 'April', 1, 'Strategy'),
  ('Selling', 'Jahn',  2009, 'April', 1, 'Profit'), ('Selling', 'Jahn',  2009, 'April', 1, 'Strategy'),
  ('Report',  'Jahn',  2017, 'June', 14, 'Profit'), ('Report',  'Jahn',  2017, 'June', 14, 'Staff'),
  ('Report',  'Frey',  2017, 'June', 14, 'Profit'), ('Report',  'Frey',  2017, 'June', 14, 'Staff');
```

- as you can see, there is much redundancy as the title and the date information is copied for each combination of author and keyword
- therefore we are going to drop the First Normal Form and storing the information in the Non First Normal Form
- Task: create a new type `datetype`, that aggregates the presented date information into just one field, and also create a new table `book_nf2`, that makes use of this new type `datetype` and also uses lists of authors and keywords to prevent the redundant information in the presented table `book_1nf`
- finally, copy all information from `book_1nf` to `book_nf2` by nesting the information into just two datasets and avoid duplicates

Arrays
Nest from 1NF to NF^2

```
CREATE TYPE datetype AS (year INTEGER, month VARCHAR, day INTEGER);

CREATE TABLE book_nf2
  (title VARCHAR, authors VARCHAR[], dateymd datetype, keywords VARCHAR[]);

INSERT INTO book_nf2
  SELECT title,
         array_agg(DISTINCT author),              -- DISTINCT avoids duplicate entries
         (year, month, day)::datetype as dateymd, -- the DBMS cannot do this cast on its own
         array_agg(DISTINCT keyword)
  FROM book_1nf GROUP BY title, dateymd;
```

- now, you know the way from 1NF to NF^2, but you also should know the other way around
- Task: copy the information back from `book_nf2` to `book_1nf` by unnesting the data

Arrays
Unnest from NF² to 1NF

```
INSERT INTO book_1nf
  SELECT title,
         unnest(authors),              -- unnest authors and dateymd second in main-query
         (dateymd).year,
         (dateymd).month,
         (dateymd).day,
         keyword
  FROM (
    SELECT title,
           authors,
           dateymd,
           unnest(keywords) as keyword -- unnest keywords first in sub-query
    FROM book_nf2
  ) as unnestfirstlevel;               -- the sub-query has to be named but the name doesn’t matter
```

Object Identifiers (OIDs)

- OIDs are used internally by PostgreSQL as primary keys for various system tables
- they are not added to user-created tables by default but this can be explicitly demanded

```
CREATE TABLE tablename (tabledefinition) WITH OIDS;
```

- as OIDs are currently implemented as unsigend 4-byte integer, this will not suffice for database-wide uniqueness in large databases or even large tables and their use as primary keys is discouraged
- they are best used only for references to system tables
- real references/pointers are not supported