Laboratory 7 - Triggers

Author: Piotr Copek, Zuzanna Micorek

Date: 18.06.2025

Task 3.1

```
CREATE TABLE Person (
    name varchar(15),
    surname varchar(15),
    PESEL varchar(11),
    dob timestamp
);
```

Task 3.2

```
CREATE TABLE Emp (
    emp_no integer,
    team_no integer,
    salary real
) INHERITS (Person);
```

Task 3.3

```
INSERT INTO Person VALUES
('Jan', 'Garrick', '11111111111', '1988-01-01'),
('Adam', 'Brown', '22222222222', '1989-10-01'),
('Ann', 'King', '33333333333', '1990-10-15');
```

Task 3.4

```
INSERT INTO Emp VALUES
('Tom', 'White', '44444444444', '1998-12-12', 1, 10, 6500),
('Mary', 'Smith', '55555555555', '2000-12-12', 2, 10, 5000);
```

Task 3.5

```
SELECT pa.attname, pt.typname
FROM pg_class pc, pg_attribute pa, pg_type pt
WHERE pc.relname='person' AND pc.oid=pa.attrelid AND pt.oid=pa.atttypid;
```

Task 3.6

```
SELECT tableoid FROM Emp;
```

Task 3.7

```
SELECT tableoid FROM Person;
```

Observed that the OIDs are different from `Emp's` OIDs.

Task 3.8

```
SELECT tableoid, * FROM Person;
```

Task 3.9

```
SELECT tableoid, * FROM ONLY Person;
```

Task 3.10

```
DELETE FROM Emp WHERE name = 'Mary';
```

Task 3.11

```
SELECT * FROM Emp;
SELECT * FROM Person;
```

Mary Smith was deleted from both `Emp` and `Person` tables due to inheritance.

Task 3.12

```
INSERT INTO Emp VALUES
('Lily', 'Allen', '88888888888', '1997-02-02', 2, 10, 1950),
('John', 'Fruit', '99999999999', '2003-12-12', 3, 20, 2000);
```

Task 3.13

```
SELECT tableoid FROM Person;
```

Observed that the OIDs remain consistent with previous behavior.

Task 3.14

```
CREATE TABLE premium (emp_no integer, quarterly_bonus integer[]);
```

Task 3.15

```
INSERT INTO premium VALUES (1, '{800,950,400,250}');
```

Task 3.16

```
SELECT * FROM premium;
SELECT quarterly_bonus[1] FROM premium;
```

Task 3.17

```
CREATE TABLE borrows (emp_no integer, autor_tytul text[][]);
```

Task 3.18

```
INSERT INTO borrows VALUES
(1, '{{"Tolkien", "Hobbit", "Iskry", "1980"}, {"Dickens", "Pickwick
    Club", "MG", "1989"}, {"Stone", "Deadline", "ZYSK I S-KA", "2009"}}'),
(2, '{{"Pascal", "Guidebook", "Lonely planet", "2010"}, {"Archer",
    "Kane and Abel", "REBIS", "1999"}}');
```

Task 3.19

```
SELECT * FROM borrows;
SELECT emp_no, autor_tytul[1][1] FROM borrows;
SELECT emp_no, autor_tytul[1:3][1] FROM borrows;
SELECT emp_no, autor_tytul[1:3][1:3] FROM borrows;
SELECT emp_no, autor_tytul[1:3][2] FROM borrows;
SELECT emp_no, autor_tytul[2][2] FROM borrows;
SELECT emp_no, autor_tytul[2][1] FROM borrows;
```

Various queries demonstrating array slicing and accessing specific elements in the 2D array.

Task 3.20

```
CREATE FUNCTION pers_data (integer) RETURNS text AS
'SELECT surname FROM Emp WHERE emp_no = $1' LANGUAGE 'sql';
```

Task 3.21

```
SELECT pers_data(1) AS surname;
```

Task 3.22

```
CREATE TYPE complex AS (i text, n text, p text);
CREATE FUNCTION pers_data2 (integer) RETURNS complex AS
'SELECT name, surname, PESEL FROM Emp WHERE emp_no = $1' LANGUAGE 'sql';
```

Task 3.23

```
CREATE FUNCTION pers_data3 () RETURNS setof complex AS
'SELECT name, surname, PESEL FROM Emp' LANGUAGE 'sql';
```

Task 3.24

```
CREATE FUNCTION book_titles(integer) RETURNS SETOF text AS
'SELECT autor_tytul[i][2]
FROM borrows, generate_subscripts(autor_tytul, 1)
AS i WHERE emp_no = $1' LANGUAGE 'sql';
```

Task 3.25

```
CREATE OR REPLACE FUNCTION extra_money (integer) RETURNS real AS
$$
DECLARE var real;
BEGIN
    SELECT 1.25 * salary INTO var FROM Emp WHERE emp_no = $1;
    RETURN var;
END;
$$
LANGUAGE 'plpgsql';
```

Task 3.26

```
CREATE RULE rule1
AS ON UPDATE TO Emp
WHERE NEW.salary <> OLD.salary
DO INSTEAD NOTHING;
```

Task 3.27

```
SELECT * FROM Emp;
UPDATE Emp SET team_no = 30 WHERE team_no = 20;
SELECT * FROM Emp;
UPDATE Emp SET salary = 9000 WHERE name = 'Tom';
SELECT * FROM Emp;
DROP RULE rule1 ON Emp;
```

The team_no updates work but salary updates are blocked, then dropped the rule.

Task 3.28

```
CREATE RULE prevent_invalid_emp_no AS
ON INSERT TO Emp
WHERE NEW.emp_no <= 0
DO INSTEAD NOTHING;
```

Task 3.29

```
CREATE VIEW pers_view AS SELECT name, surname, PESEL FROM Person WHERE name='Witold';
CREATE RULE rule2 AS
ON INSERT TO pers_view
DO INSTEAD INSERT INTO Person (name, surname, PESEL)
VALUES (NEW.name, NEW.surname, NEW.PESEL);
```

Task 3.30

```
INSERT INTO pers_view VALUES ('Witold', 'Smith', '12345678901');
SELECT * FROM Person WHERE name = 'Witold';
```

Task 3.31

```
ALTER TABLE Premium ADD COLUMN last_updated timestamptz;
```

Task 3.32-3.33: Create and test update trigger

```
CREATE OR REPLACE FUNCTION upd() RETURNS trigger AS
$$
BEGIN
    NEW.last_updated = now();
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER last_upd
BEFORE INSERT OR UPDATE ON Premium
FOR EACH ROW
EXECUTE PROCEDURE upd();

-- Test
SELECT * FROM Premium;
INSERT INTO Premium VALUES (2, '{300,150,100,150}');
SELECT * FROM Premium;
```

Task 3.34

```
CREATE TABLE PRODUCTS (id integer, name text, net_price real);
INSERT INTO PRODUCTS VALUES
(1, 'cable', 50),
(2, 'laptop', 1940),
(3, 'monitor', 850);
```

Task 3.35

```
CREATE FUNCTION vat_tax(real) RETURNS real AS
'SELECT $1 * 0.23' LANGUAGE 'sql';

SELECT id, name, net_price, vat_tax(net_price) AS vat,
       net_price + vat_tax(net_price) AS gross_price FROM PRODUCTS;
```

Task 3.36

```
CREATE TABLE PRODUCTS_2 (
    id integer,
    name text,
    price real,
    vat_tax real,
    gross_price real
);

CREATE OR REPLACE FUNCTION calculate_vat() RETURNS trigger AS
$$
BEGIN
    NEW.vat_tax := NEW.price * 0.23;
    NEW.gross_price := NEW.price + NEW.vat_tax;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER update_vat
BEFORE INSERT OR UPDATE ON PRODUCTS_2
FOR EACH ROW
EXECUTE PROCEDURE calculate_vat();
```

Task 3.37

```
INSERT INTO PRODUCTS_2 (id, name, price) VALUES
(1, 'cable', 50),
(2, 'laptop', 1940),
(3, 'monitor', 850);
SELECT * FROM PRODUCTS_2;
```

Trigger correctly calculates VAT and gross price on insert.

Task 3.38

```
DROP DATABASE sandbox_db; --on online compiler it won't work due to safety reasons
```

Conclusions

This lab covered PostgreSQL features including table inheritance, arrays, functions, rules, triggers, and complex data types. The tasks showed how to:

- Create and manage table inheritance
- Work with array data types
- Write pgSQL functions
- Implement business rules using rules and triggers
- Handle complex data structures
- Automate calculations and validations