Databázové systémy

Zadanie 5

Implementácia procesov na dátovej úrovni

Filip Híreš

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

xhires@stuba.sk

21.4.2024

Vytvorenie databázy:

Jednotlivé SQL súbory je treba spustiť v nasledujúcom poradí:

DB.sql – obsahuje ddl pre samotné tabuľky

data.sql – obsahuje príkazy pre naplnenie databázy dátami

overlap.sql – obsahuje funkciu pre kontrolu prekrývania časov

Následne je možné spustiť súbory obsahujúce constrainty v podobe triggerov v ľubovoľnom poradí:

expozicia.sql

kontrola.sql

stav.sql

vlastnik.sql

zony.sql

A nakoniec procedúry a funkcie:

new_ex.sql

show_ex.sql

show_free_zone.sql

new_expo.sql

asign_to_expo.sql

Opis funkcií:

```
CREATE OR REPLACE FUNCTION time_overlap(zaciatok TIMESTAMP WITHOUT TIME ZONE, koniec TIMESTAMP WITHOUT TIME ZONE, new_zaciatok TIMESTAMP WITHOUT TIME ZONE, new_koniec TIMESTAMP WITHOUT TIME ZONE)
RETURNS BOOLEAN AS $$
BEGIN
  IF (zaciatok, koniec) OVERLAPS (new_zaciatok, new_koniec) OR (new_zaciatok, new_koniec) OVERLAPS (zaciatok, koniec) THEN RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

Táto funkcia skontroluje či sa časy na vstupe prekrývajú. Pokiaľ sa prekrývajú vráti TRUE v opačnom prípade FALSE. Funkcia sa využíva na zistenie časových konfliktov.

```
CREATE OR REPLACE FUNCTION check_stav()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1
    FROM public.stav s
    WHERE s.exemplareid = NEW.exemplareid
    AND time_overlap(s.zaciatok, s.koniec, NEW.zaciatok, NEW.koniec)
    AND s.status <> 'zrusene'
  ) THEN
    RAISE EXCEPTION 'Casy sa prekryvaju';
  END IF;
  RETURN NEW;
```

```
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER before_insert_update_stav
BEFORE INSERT OR UPDATE ON public.stav
FOR EACH ROW
EXECUTE FUNCTION check_stav();
```

Tento trigger s a spustí, ak chceme vložiť záznam do tabuľky stav. Skontroluje či exemplár nie je v danom čase na inom mieste. Ak je INSERT neprebehne.

--stav

```
CREATE OR REPLACE FUNCTION check_stav()
RETURNS TRIGGER AS $$
BEGIN
--zistí či exemplár v danom čase nie je niekde inde
  IF EXISTS (
    SELECT 1
    FROM public.stav s
    WHERE s.exemplareid = NEW.exemplareid
    AND time_overlap(s.zaciatok, s.koniec, NEW.zaciatok, NEW.koniec)
    AND s.status <> 'zrusene'
  ) THEN
    RAISE EXCEPTION 'Casy sa prekryvaju';
  END IF;
--ak nie je može insert prebehnúť
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_update_stav

BEFORE INSERT OR UPDATE ON public.stav

FOR EACH ROW

EXECUTE FUNCTION check_stav();
```

Trigger sa spustí ak chceme vytvoriť záznam o kontrole. Z tabuľky stav sa zistí, či exponát nie je v danom čase inde. Ak nie vytvorí sa záznam o kontrole v tabuľke kontroly a taktiež v tabuľke stav.

```
--expozicia


CREATE OR REPLACE FUNCTION check_expozicia()

RETURNS TRIGGER AS $$

DECLARE

  zony_exists BOOLEAN;

BEGIN

--skontroluje sa či v zóne prebieha expozícia

  SELECT EXISTS (

    SELECT expozicie.id FROM zony

        JOIN expozicie_zony ON zony.id = expozicie_zony.zonyid

        JOIN expozicie ON expozicie_zony.expozicieid = expozicie.id

        WHERE time_overlap(expozicie.zaciatok, expozicie.koniec, NEW.zaciatok,
NEW.koniec) AND zony.id = NEW.zonyid AND expozicie.status != 'zrusene'

  ) INTO zony_exists;

  IF zony_exists THEN

--ak áno skontroluje sa ci exemplar nie je v danom čase inde

    INSERT INTO stav (drzitel, zaciatok, koniec, stav, status, exemplareid)

    VALUES ('Moje Muzeum', NEW.zaciatok, NEW.koniec, 'vystavene', NEW.status,
NEW.exemplareid);

--ak nie vloži sa záznam o vystavení v expozícii
```

```
        INSERT INTO exemplare_expozicie(exemplareid, expozicieid) VALUES
        (NEW.exemplareid, (SELECT expozicie.id FROM zony
        JOIN expozicie_zony ON zony.id = expozicie_zony.zonyid
        JOIN expozicie ON expozicie_zony.expozicieid = expozicie.id
        WHERE time_overlap(expozicie.zaciatok, expozicie.koniec, NEW.zaciatok,
NEW.koniec) AND zony.id = NEW.zonyid AND expozicie.status != 'zrusene'));
        RETURN NEW;
  ELSE
    RAISE EXCEPTION 'Zonyid neexistuje vo vyfiltrovanom zozname';
    RETURN NULL;
  END IF;
EXCEPTION
  WHEN others THEN
    RAISE EXCEPTION 'Chyba';
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER before_insert_update_exemplare_zony
BEFORE INSERT OR UPDATE ON public.exemplare_zony
FOR EACH ROW
EXECUTE FUNCTION check_expozicia();
```

Trigger sa spustí ak chceme vytvoriť záznam o vystavení exempláru v určitej zóne. Najprv sa zistí, či v danom čase prebieha v zóne nejaká expozícia. Ak áno, skúsi sa vytvoriť záznam v tabuľke stav o vystavení. Ak prebehne úspešne, vytvorí sa aj záznam o vystavení exempláru v expozícii, a následne o vystavení v zóne.

```
CREATE OR REPLACE FUNCTION check_vlastnik()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1
    FROM public.vlastnik_exemplare ve
    WHERE ve.exemplareid = NEW.exemplareid
    AND time_overlap(ve.zaciatok, ve.koniec, NEW.zaciatok, NEW.koniec)
  ) THEN
    RAISE EXCEPTION 'Casy sa prekryvaju';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_update_vlastnik_exemplare
BEFORE INSERT OR UPDATE ON public.vlastnik_exemplare
FOR EACH ROW
EXECUTE FUNCTION check_vlastnik();
```

Trigger sa spustí ak chceme priradiť exempláru nového vlastníka. Dôjde k overeniu, či v danom momente nevlastní exemplár niekto iný.

```
CREATE OR REPLACE FUNCTION check_zona()
RETURNS TRIGGER AS $$
DECLARE
  zaciatok1 TIMESTAMP WITHOUT TIME ZONE;
  koniec1 TIMESTAMP WITHOUT TIME ZONE;
  row_data RECORD;
```

```
BEGIN
  SELECT zaciatok, koniec INTO zaciatok1, koniec1 FROM expozicie
  WHERE expozicie.id = NEW.expozicieid;
  FOR row_data IN
    SELECT zaciatok, koniec, status
    FROM expozicie_zony
    JOIN expozicie ON expozicieid = expozicie.id
    WHERE zonyid = NEW.zonyid
  LOOP
    IF time_overlap(zaciatok1, koniec1, row_data.zaciatok, row_data.koniec)
        AND row_data.status <> 'zrusene' THEN
    RAISE EXCEPTION 'Casy sa prekrivaju';
        END IF;
  END LOOP;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER before_insert_update_expozicie_zony
BEFORE INSERT OR UPDATE ON public.expozicie_zony
FOR EACH ROW
EXECUTE FUNCTION check_zona();
```

Trigger sa spustí ak chceme zaradiť zónu do expozície. Dôjde k overeniu, či v danom momente zóna nie je súčasťou inej expozície.

```
CREATE OR REPLACE PROCEDURE new_ex(nazov VARCHAR, vl BOOLEAN, po
BOOLEAN, majitel VARCHAR, od TIMESTAMP WITHOUT TIME ZONE, doo TIMESTAMP
WITHOUT TIME ZONE, VARIADIC kateg VARCHAR[])
LANGUAGE plpgsql AS $$
DECLARE kategoria VARCHAR;
BEGIN
	INSERT INTO exemplare (meno, vlastne, pozicane) VALUES
		(nazov, vl, po);
	IF NOT EXISTS (
		SELECT 1 FROM vlastnik
		WHERE meno = majitel
		) THEN
		INSERT INTO vlastnik (meno) VALUES
			(majitel);
	END IF;
	INSERT INTO vlastnik_exemplare (exemplareid, vlastnikid, zaciatok, koniec)
VALUES
		((SELECT id FROM exemplare WHERE meno = nazov), (SELECT id FROM
vlastnik WHERE meno = majitel), od, doo);
	FOREACH kategoria IN ARRAY kateg
	LOOP
		IF NOT EXISTS (
			SELECT 1 FROM kategorie
			WHERE meno = kategoria
			) THEN
				INSERT INTO kategorie (meno) VALUES
					(kategoria);
		END IF;
		INSERT INTO exemplare_kategorie (exemplareid, kategorieid) VALUES
```

```
                ((SELECT id FROM exemplare WHERE meno = nazov), (SELECT id FROM
kategorie WHERE meno = kategoria));
        END LOOP;
END;
$$;
```

Po zavolaní tejto procedúry sa vytvorí nový záznam o exempláre jeho majiteľovi, a kategóriách, na základe vstupných parametrov.

```
CREATE OR REPLACE FUNCTION show_ex(VARIADIC kateg VARCHAR[])
RETURNS TABLE (
  meno VARCHAR,
  kategorie VARCHAR[]
  )
LANGUAGE plpgsql AS $$
BEGIN
  RETURN QUERY
  SELECT exemplare.meno,
      array_agg(kategorie.meno)
  FROM kategorie
  JOIN exemplare_kategorie ON kategorie.id = exemplare_kategorie.kategorieid
  JOIN exemplare ON exemplare_kategorie.exemplareid = exemplare.id
  WHERE exemplare.id IN (SELECT exemplareid FROM exemplare_kategorie WHERE
kategorieid IN (SELECT id FROM kategorie WHERE kategorie.meno = ANY(kateg)))
  GROUP BY exemplare.meno;
END;
$$;
```

Táto funkcia nám vráti zoznam všetkých exemplárov, ktoré spadajú aspoň do jednej kategórie na vstupe, spolu so všetkými ich kategóriami.

```
CREATE OR REPLACE FUNCTION show_free_zone(od TIMESTAMP WITHOUT TIME ZONE,
doo TIMESTAMP WITHOUT TIME ZONE)
RETURNS TABLE (
  meno VARCHAR,
  rozloha INT
  )
LANGUAGE plpgsql AS $$
BEGIN
  RETURN QUERY
  SELECT zony.meno, zony.rozloha FROM zony
LEFT JOIN expozicie_zony ON zony.id = zonyid
LEFT JOIN expozicie ON expozicieid = expozicie.id
WHERE NOT time_overlap('2024-04-01', '2024-07-01', expozicie.zaciatok,
expozicie.koniec);
END;
$$;
```

Táto funkcia vráti zoznam všetkých neobsadených zón v čase, ktorý je zadaný na vstupe.

```
CREATE OR REPLACE PROCEDURE new_expo(nazov VARCHAR, od TIMESTAMP
WITHOUT TIME ZONE, doo TIMESTAMP WITHOUT TIME ZONE, stat enum, VARIADIC
zones VARCHAR[])
LANGUAGE plpgsql AS $$
DECLARE zone VARCHAR;
BEGIN
        INSERT INTO expozicie (meno, zaciatok, koniec, status) VALUES
                (nazov, od, doo, stat);
        FOREACH zone IN ARRAY zones
        LOOP
                INSERT INTO expozicie_zony(expozicieid, zonyid) VALUES
```

```
                ((SELECT id FROM expozicie WHERE meno = nazov), (SELECT id
FROM zony WHERE meno = zone));
        END LOOP;
END;
$$;
```

Procedúra dostane na vstupe názov a čas novej expozície spolu zo všetkými zónami, v ktorých sa bude uskutočňovať. Ak sú všetky zóny v danom čase voľné, vytvorí sa záznam o expozícii.

```
CREATE OR REPLACE PROCEDURE asign_to_expo(nazov VARCHAR, zone VARCHAR,
VARIADIC exemps VARCHAR[])
LANGUAGE plpgsql AS $$
DECLARE ex VARCHAR;
BEGIN
        FOREACH ex IN ARRAY exemps
        LOOP
                INSERT INTO exemplare_zony(exemplareid, zonyid, zaciatok, koniec,
status) VALUES
                ((SELECT id FROM exemplare WHERE meno = ex), (SELECT id FROM
zony WHERE meno = zone),
                (SELECT zaciatok FROM expozicie WHERE meno = nazov), (SELECT
koniec FROM expozicie WHERE meno = nazov),
                (SELECT status FROM expozicie WHERE meno = nazov));
        END LOOP;
END;
$$;
```

Procedúra dostane na vstupe názov expozície a jednu zónu spolu so všetkými exemplármi, ktoré sa v zóne majú vystavovať. Ak sú splnené všetky obmedzenia, vzniknú nové záznamy o vystavení exemplárov v danej zóne.

Príklady volaní:

V tejto časti ukážem konkrétne príklady volaní spolu s výstupmi.

Pridanie nového exempláru:

V databáze sú tieto údaje o exemplároch

	meno character varying (255)	array_agg character varying[]	vlastnik character varying (255)
1	Delo	{zbraň,novovek}	Moje múzeum
2	Meč	{stredovek,zbraň}	Moje múzeum
3	Primitívne nástroje	{pravek,nástroj}	Moje múzeum
4	Rímska prilba	{starovek,zbroj}	Moje múzeum
5	Štít	{zbroj,stredovek}	Moje múzeum

Ak chceme pridať nový exemplár môžeme využiť procedúru new_ex(). Po jej zavolaní

CALL new_ex('Renesančný obraz', TRUE, FALSE, 'Moje múzeum', '2024-04-21', null, 'novovek', 'renesancia', 'obraz');

Nám vzniknú nové záznamy v tabuľkách. Pokiaľ nejaká z kategórií neexistovala, vznikne nový záznam aj o nej.

	meno character varying (255)	array_agg character varying[]	vlastnik character varying (255)
1	Delo	{novovek,zbraň}	Moje múzeum
2	Meč	{zbraň,stredovek}	Moje múzeum
3	Primitívne nástroje	{pravek,nástroj}	Moje múzeum
4	Renesančný obraz	{renesancia,novovek,obraz}	Moje múzeum
5	Rímska prilba	{zbroj,starovek}	Moje múzeum
6	Štít	{stredovek,zbroj}	Moje múzeum

Naplánovanie expozície:

Pri plánovaní expozície vieme využiť dve pomocné funkcie. Jedna z nich nám zobrazí všetky exempláre s určitou kategóriou.

SELECT show_ex('stredovek')

	show_ex record
1	(Meč,"{stredovek,zbraň}")
2	("Štít","{stredovek,zbroj}")

Pokiaľ zadáme viac argumentov zobrazia sa všetky exempláre, ktoré spadajú aspoň do jednej kategórie.

SELECT show_ex('stredovek', 'novovek')

	show_ex record
1	("Renesančný obraz","{novovek,renesancia,obraz}")
2	(Delo,"{novovek,zbraň}")
3	(Meč,"{stredovek,zbraň}")
4	("Štít","{stredovek,zbroj}")

Druhou funkciou si vieme zobraziť všetky zóny aj s ich rozlohou, ktoré sú dostupné v určitom čase. Napríklad máme naplánovanú nasledujúcu expozíciu.

	meno character varying (255)	zaciatok timestamp without time zone	koniec timestamp without time zone	status enum	zony character varying[]
1	Výstava zbraní	2024-05-01 00:00:00	2024-08-01 00:00:00	planovane	{A,C}

Keď teda zavoláme funkciu

SELECT show_free_zone('2024-03-01', '2024-06-30')

Zistime ktoré zóny sú voľné od marca 2024 do júna 2024

	show_free_zone record
1	(B,100)
2	(E,50)
3	(D,80)

Ak chceme vytvoriť novú expozíciu, môžeme to spraviť pomocou procedúry

CALL new_expo('Stredovek', '2024-03-01', '2024-06-01', 'planovane', 'A', 'B', 'E')

V predošlom príklade sme zistili, že zóna A v tomto čase nie je voľná a systém nás upozorní, že sa prekrývajú časy.

```
ERROR:  Casy sa prekrivaju
CONTEXT:  PL/pgSQL function
```

Keď nahradíme zónu A zónou D:

CALL new_expo('Stredovek', '2024-03-01', '2024-06-01', 'planovane', 'D', 'B', 'E')

Tak sa pridanie expozície podarí.

	meno character varying (255)	zaciatok timestamp without time zone	koniec timestamp without time zone	status enum	zony character varying[]
1	Výstava zbraní	2024-05-01 00:00:00	2024-08-01 00:00:00	planovane	{A,C}
2	Stredovek	2024-03-01 00:00:00	2024-06-01 00:00:00	planovane	{D,B,E}

Pre samotné priraďovanie do zón použijeme procedúru

CALL asign_to_expo('Stredovek', 'D', 'Meč', 'Štít')

V ktorej si vyberieme expozíciu, zónu a exempláre, ktoré do nej priradíme.

	meno character varying (255)	zaciatok timestamp without time zone	koniec timestamp without time zone	status enum	zony character varying (255)	exponaty character varying[]
1	Stredovek	2024-03-01 00:00:00	2024-06-01 00:00:00	planovane	B	{NULL}
2	Stredovek	2024-03-01 00:00:00	2024-06-01 00:00:00	planovane	D	{Štít,Meč}
3	Stredovek	2024-03-01 00:00:00	2024-06-01 00:00:00	planovane	E	{NULL}

Pokiaľ by sme sa snažili prideliť rovnaké exempláre do inej ďalšej expozície v rovnakom čase, systém nás upozorní na chybu.

CALL asign_to_expo('Výstava zbraní', 'A', 'Meč', 'Štít')

```
ERROR:  Chyba
```

Prevzatie exemplára z inej inštitúcie:

Pokiaľ chceme prevziať exemplár z inej inštitúcie platí, že žiaden exemplár nemôže patriť dvom majiteľom súčasne. Pre príklad vytvorme exemplár ktorý patrí inej inštitúcii.

CALL new_ex('Egyptská koruna', FALSE, FALSE, 'Múzeum histórie', '2022-02-01', '2024_04-22', 'Egypt', 'Koruna')

	meno character varying (255)	array_agg character varying[]	vlastnik character varying (255)
1	Delo	{novovek,zbraň}	Moje múzeum
2	Egyptská koruna	{Koruna,Egypt}	Múzeum histórie

Tento exemplár patrí múzeu histórie do 22.4.2024. Ak by sme ho chceli prevziať skôr

INSERT INTO vlastnik_exemplare (vlastnikid, exemplareid, zaciatok) VALUES

(1, 7, '2024-03-01');

```
ERROR:  Casy sa prekryvaju
```

Systém nás upozorní na časový konflikt.

INSERT INTO vlastnik_exemplare (vlastnikid, exemplareid, zaciatok) VALUES

(1, 7, '2024-04-23');

7	7	3	2022-02-01 00:00:00	2024-04-22 00:00:00
8	7	1	2024-04-23 00:00:00	[null]

Po zadaní správneho času sa vlastník zmení.

Ak si exemplár len zapožičiavame, vlastník sa nemení.

Presun exempláru do inej zóny:

Pri presune exempláru sa jednoducho updatujú hodnoty v príslušných tabuľkách stav a exempláre zóny. Takisto ako vo všetkých prípadoch nás systém upozorní na prekrývajúce sa časy.

```
ERROR:  Casy sa prekrivaju
CONTEXT:  PL/pgSQL function
```