## Turnaj v počítačové hře – CP4

1. Pro transakci jsem vytvořil funkci aktualizovat\_strategii níže, která v ní je použita. Její činnost je jasná, změní text strategie zvoleného hráče na strategii požadovanou. Ovšem zvolený hráč musí existovat v databázi, jinak funkce vrací false a nic se neprovede. V transakci níže je využit level izolace SERIALIZABLE, protože nesmí dojít k provedení dvou transakcí naráz od více uživatelů. Kdyby se nepoužilo transakce, tak by to mohlo vést k neaktuálnosti strategie konkrétního hráče, kterou použil v zápasu či v celém turnaji. Přidávám také obrázek změněho záznamu v tabulce Strategie.

CREATE FUNCTION aktualizovat\_strategii(chteny\_hrac int4, nova\_strategie text)

```
RETURNS BOOLEAN
```

```
AS $$
```

```
DECLARE
```

h int4;

## **BEGIN**

h := (SELECT hrac FROM strategie WHERE hrac = chteny\_hrac);

IF (h IS NULL) THEN

RETURN false;

END IF;

UPDATE strategie SET strategie = nova\_strategie WHERE hrac = chteny\_hrac;

RETURN true;

END;

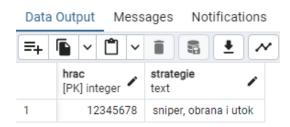
\$\$

LANGUAGE plpgsql;

## BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT aktualizovat strategii(12345678, 'sniper, obrana i utok');

## COMMIT TRANSACTION;



2. Pohled jsem vytvořil a použil pro tabulku Organizator pro vložení dalšího organizátora prvního turnaje. Pohled kontroluje pomocí WITH LOCAL CHECK OPTION, jestli vložený organizátor má atribut roven 'ESL Majors Series One', jak je specifikováno. Přidávám také obrázek, že organizátor byl vložen.

CREATE VIEW organizatori\_prvniho\_turnaje AS

**SELECT** \*

FROM organizator

WHERE nazev\_turnaje = 'ESL Majors Series One'

WITH LOCAL CHECK OPTION;

INSERT INTO organizatori\_prvniho\_turnaje

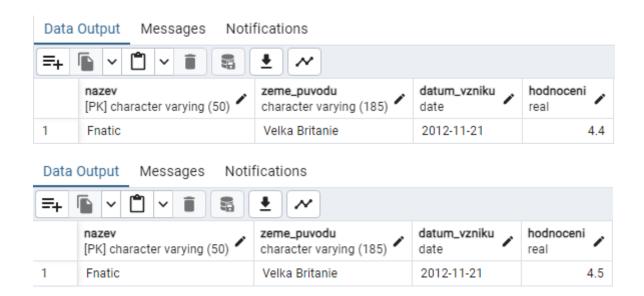
VALUES ('ESL Majors Series One', 'Jozko', 'Ferko');

Data Output Messages Notifications

	nazev_turnaje [PK] character varying (50)	jmeno [PK] character varying (50)	prijmeni [PK] character varying (50)
1	ESL Majors Series One	Jozko	Ferko
2	ESL Majors Series One	Kyle	Broflovski
3	ESL Majors Series One	Michal	Wazowski
4	ESL Majors Series One	Tadeusz	Kozlowski

3. Trigger jsem vytvořil a použil pro tabulku Cena a Tym. Nejdříve jsem vytvořil funkci zvysit\_hodnoceni\_tymu(), a poté Trigger, který funkci využívá. Činnost je jasná, po vložení ceny či cen do tabulky Cena se pro každou vloženou cenu zkontroluje, jaký tým cenu vyhrál a podle toho zvýší jeho hodnocení o 0.1 na maximálně 5.0. Přikládám obrázky hodnocení týmu Fnatic před a po vykonání insertu ceny do tabulky Cena.

```
CREATE FUNCTION zvysit_hodnoceni_tymu()
RETURNS TRIGGER
AS $$
       DECLARE
              vitez varchar(50);
              hodnoceni_viteze float4;
       BEGIN
              vitez := (SELECT nazev_tymu FROM cena WHERE nazev_tymu = NEW.nazev_tymu);
              hodnoceni_viteze := (SELECT hodnoceni FROM Tym WHERE nazev = vitez);
              IF (hodnoceni_viteze + 0.1 > 5.0) THEN
                      UPDATE Tym SET hodnoceni = 5.0 WHERE nazev = vitez;
              ELSE
                      UPDATE Tym SET hodnoceni = hodnoceni_viteze + 0.1 WHERE nazev = vitez;
              END IF;
              RETURN NULL;
       END;
$$
LANGUAGE plpgsql;
CREATE TRIGGER tym_trigger_hodnoceni AFTER INSERT ON cena
FOR EACH ROW EXECUTE FUNCTION zvysit_hodnoceni_tymu();
INSERT INTO cena
VALUES ('Katowice 2014', '2014-8-10', 'zlaty pohar', 'ESL Majors Series One', 'Fnatic');
```



4. Dva indexy jsem vytvořil pro tabulku Kolo, jeden pro vyhledání na základě id kola, druhý na základě id\_zapasu, který kolo má. V tabulce Kolo je přibližně 3600 záznamů. Kvůli tomu jsem tyto indexy vytvořil, aby se při vyhledání určitého kola (podle id) či skupiny kol (podle id\_zapasu) použil pro onen dotaz daný index. Pomocí EXPLAIN jsem si zkontroloval, jestli QUERY PLAN použije indexy pro rychlejší vyhledání záznamu či záznamů místo přímého hledání v tabulce pro dotazy níže. Přikládám také obrázky výsledku EXPLAIN dotazu SQL.

CREATE INDEX kolo\_index\_id ON kolo (id);

EXPLAIN SELECT \* FROM kolo WHERE id = 2400;

CREATE INDEX kolo\_index\_id\_zapasu ON kolo (id\_zapasu);

