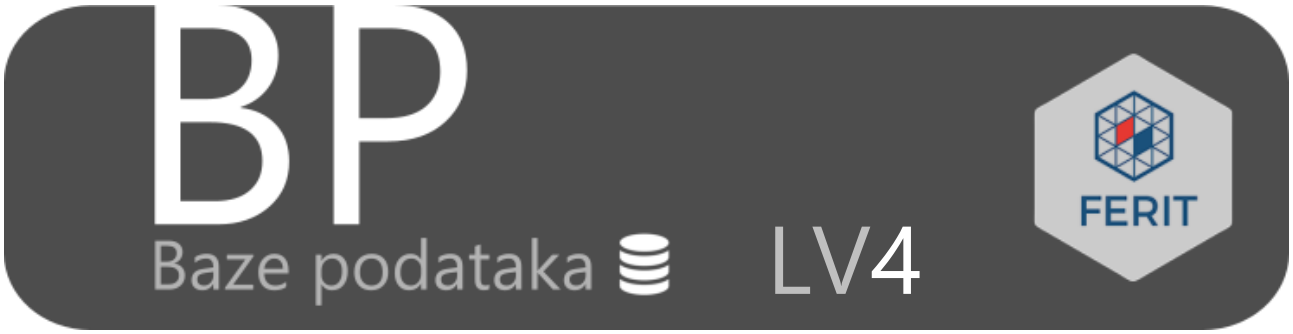

1. GRANT naredba

Pridjeljivanje prava korisnicima da izvršavaju SQL naredbe i/ili kreiraju objekte.
Općenita se naredba koristi kao:

```
GRANT <some permission> ON <some object> TO <some user, login, or group>
```

Skraćena sintaksa:

```
GRANT privilege_name
[ ON object_name ]
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

privilege_name
Lista naredbi/privilegija za koje se pravo dodjeljuje. Neke od mogućih naredbi su sljedeće: ALL, SELECT, INSERT, UPDATE, DELETE, CREATE DATABASE, CREATE TABLE, CREATE PROCEDURE, CREATE FUNCTION, CREATE VIEW, BACKUP DATABASE, ...

object_name
Ime objekta nad kojim se dodjeljuju gore navedena privilegije, kao npr. TABLE, VIEW, STORED PROC i SEQUENCE.

user_name
Naziv korisnika kojemu se pridjeljuju prava.

PUBLIC
Dodjeljivanje prava svim korisnicima.

Primjeri:

```
GRANT CREATE TABLE, CREATE VIEW TO stuslu;

GRANT CREATE PROCEDURE TO stuslu;
```

2. REVOKE naredba

Oduzimanje prethodno pridijeljenih prava korisnicima na izvršavanje SQL naredbi i/ili kreiranje objekata (samo ona prava koja su prethodno dodijeljena GRANT naredbom se mogu i oduzeti).

Skraćena sintaksa:

```
REVOKE privilege_name
[ ON object_name ]
FROM {user_name |PUBLIC |role_name}
```

REVOKE GRANT OPTION FOR user_name se koristi za oduzimanja prava daljnjeg dodjeljivanja prava (ako je to neki korisnik dobio sa WITH GRANT OPTION).

Primjer:

```
REVOKE CREATE TABLE, CREATE VIEW TO stuslu;
```

3. DENY naredba

Zabranjuje određena prava korisniku i sprječava korisnika da naslijedi prava kroz svoju grupu prava/ulogu.

Skraćena sintaksa**:**

```
DENY privilege_name
[ ON object_name ]
TO {user_name |PUBLIC |role_name}
```

Primjer:

```
DENY CREATE TABLE TO stuslu;
```

4. TRANSAKCIJE

4.1. BEGIN TRANSACTION

- Označava trenutak do kojega su svi podaci referencirani aktivnom konekcijom logički i fizički konzistentni.
- Sve izmjene nakon BEGIN TRANSACTION se mogu vratiti u stanje prije BEGIN TRANSACTION.
- Transakcija traje dok se:
    - ne završi uspješno naredbom COMMIT TRANSACTION
    - ili ne završi greškom, odnosno naredbom ROLLBACK TRANSACTION.

4.2. COMMIT TRANSACTION

- Označava uspješan završetak transakcije.
- Sve izmjene tijekom transakcije postaju trajne.
- Svi resursi konekcije zauzeti tijekom transakcije se oslobađaju.

4.3. ROLLBACK TRANSACTION

- Vraća stanje podataka na ono koje je vrijedilo prije početka transakcije.
- Oslobađa resurse konekcije koji su bili zauzeti tijekom transakcije.

Primjeri:

```
BEGIN TRANSACTION;
  UPDATE vozilo SET km = km * 1.25;
  UPDATE vozilo SET dat_proizvodnje = NULL WHERE km > 200000;
COMMIT TRANSACTION;
```

```
BEGIN TRAN;
DELETE vozilo;
ROLLBACK;
SELECT * FROM vozilo;
```

```
BEGIN TRANSACTION;
 UPDATE Account SET amount=amount-200 WHERE account_number=1234;
 UPDATE Account SET amount=amount+200 WHERE account_number=2345;
IF @@ERROR=0 COMMIT;
IF @@ERROR<>0 ROLLBACK;
```

5. SQL kao programski jezik

- Transact-SQL
  - Microsoft SQL Server
- PL/SQL
  - Oracle
- Specifična implementacija standardnog SQL jezika koja je obogaćena proceduralnim mogućnostima, varijablama, kontrolom tijeka programa, petljama, i sl.

Primjeri:

```
PRINT @@version; -- globalne varijable pocinju sa „@@“


DECLARE @moja_varijabla VARCHAR(20);
SET @moja_varijabla = 'bla bla';-- lokalne pocinju sa „@“
PRINT @moja_varijabla;
IF @moja_varijabla = 'bla bla'
      BEGIN
        PRINT 'jednako!';
        PRINT '100%';
      END
ELSE PRINT 'razlicito!'


DECLARE @brojac INT
SET @brojac = 0
WHILE @brojac < 10
BEGIN
  SET @brojac = @brojac + 1
  PRINT 'brojac je ' + CAST(@brojac AS CHAR)
END
```

5.1. CREATE FUNCTION

- Kreiranje korisničke funkcije.
- Spremljeni Transact-SQL kod koji kao rezultat vraća neku vrijednost.
- Da bi korisnik mogao kreirati funkciju mora imati CREATE FUNCTION privilegiju, a pravo izvršavanja (EXECUTE) je korisniku implicitno dodijeljeno za sve funkcije kojima je on vlasnik.
- Funkcije se uklanjaju naredbom DROP FUNCTION. Sintaksa:

```
DROP FUNCTION { [ owner_name . ] function_name } [ ,...n ]
```

- Sintaksa CREATE FUNCTION naredbe:

```
CREATE FUNCTION [owner_name.] function_name
    ( [ { @parameter_name AS] scalar_parameter_data_type [ = default ] }  [
,...n ] ] )
RETURNS scalar_return_data_type
[ WITH < function_option> [ [,] ...n] ]
[ AS ]
BEGIN
    function_body
    RETURN scalar_expression
END
```

Primjer:

```
CREATE FUNCTION Inicijali (@ime VARCHAR(30), @prezime VARCHAR(30))
RETURNS CHAR(4)
BEGIN
  DECLARE @tmp CHAR(4);
  SET @tmp = UPPER(SUBSTRING((@ime), 1, 1)) + '.' +
             UPPER(SUBSTRING(@prezime, 1, 1)) + '.';
  RETURN @tmp;
END  -- funkcija ne zavrsava sa ';' u T-SQLu!
```

► Funkciju možemo izvršiti pozivom iz SELECT naredbe, npr:

```
SELECT Inicijali('Ivan', 'Horvat') AS Inicijali;

SELECT mbr_stud AS 'Matični broj', Inicijali(ime_stud, prez_stud) AS
'Inicijali studenata'
FROM student ORDER BY 1;
```

5.2. CREATE PROCEDURE

- Kreiranje pohranjene procedure.
- Transact-SQL kod koji može primiti i vratiti korisničke parametre.
- Da bi korisnik mogao kreirati procedure mora imati CREATE PROCEDURE privilegiju, a pravo izvršavanja (EXECUTE) je korisniku implicitno dodijeljeno za sve procedure kojima je on vlasnik.
- Procedure se uklanjaju naredbom DROP PROCEDURE. Sintaksa:

```
DROP PROCEDURE { procedure_name } [ ,...n ]
```

- Sintaksa CREATE PROCEDURE:

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
    [ { @parameter data_type }
        [ VARYING ] [ = default ] [ OUTPUT ]
    ] [ ,...n ]

[ WITH
    { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]

[ FOR REPLICATION ]

AS sql_statement [ ...n ]
```

Primjer:

```
CREATE PROCEDURE HelloWorld
AS
    DECLARE @mynvarchar NVARCHAR(50),
            @myfloat FLOAT
    SET @mynvarchar  = @@VERSION
    SET @mynvarchar  = 'Hello, world!'
    SET @myfloat     = 1.6180
    PRINT @mynvarchar
RETURN (0)-- procedura ce biti ispravna i bez ove linije
```

- Proceduru pokrećemo sa *exec*:

```
exec HelloWorld
```

```
CREATE PROCEDURE ispisiStudente
AS
SELECT * FROM student

exec ispisiStudente
```

5.3. CREATE TRIGGER

- Programski kod koji se izvršava automatski prilikom izvođenja specificirane DML operacije nad specificiranom tablicom.
- Važno je znati da se okidači kreiraju nad točno određenom tablicom i za točno određenu namjenu. Naime, moguće je definirati okidač za slučaj INSERT ili UPDATE naredbe i to na način da se okidač izvrši umjesto (INSTEAD OF) promatrane naredbe ili nakon nje (AFTER).

- Sintaksa:

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
    { { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
        [ WITH APPEND ]
        [ NOT FOR REPLICATION ]
        AS
        [ { IF UPDATE ( column )
            [ { AND | OR } UPDATE ( column ) ]
                [ ...n ]
        | IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
                { comparison_operator } column_bitmask [ ...n ]
        } ]
        sql_statement [ ...n ]
    }
}
```

Primjer:

```
CREATE TRIGGER IzracunUkupno
ON stavke_racuna_<mbr>
FOR INSERT, UPDATE
AS
DECLARE @broj_racuna INT;
DECLARE @rb INT;
SELECT @broj_racuna = broj_racuna, @rb = rb
  FROM inserted;
UPDATE stavke_racuna_<mbr> SET ukupno = kolicina * cijena * 1.25;
```

- ► Da bismo pokrenuli ovaj primjer – ponovit ćemo ono što smo radili na vježbama do sada...

- Kreirajte tablicu *racuni* s Vašim brojem indeksa kao dio imena na mjestu <*mbr*> (npr. „racuni_585“)

```
CREATE TABLE racuni_<mbr>(
 broj INT,
 datum DATETIME NOT NULL,
 kupac CHAR(100),
 CONSTRAINT racuni_pk_<mbr> PRIMARY KEY(broj)
);
```

- Isto napravite s tablicom *stavke_racuna*, kreirajte npr. „stavke_racuna_585“ (koristite Vaš broj indeksa)

```
CREATE TABLE stavke_racuna_<mbr>(
 broj_racuna INT,
 rb INT,
 kolicina DECIMAL(5,2) NOT NULL DEFAULT 1,
 jm CHAR(3) NOT NULL DEFAULT 'kom',
 naziv_robe VARCHAR(30) NOT NULL,
 cijena DECIMAL(6,2) NOT NULL,
 CONSTRAINT stavke_pk_<mbr> PRIMARY KEY(broj_racuna, rb),
 CONSTRAINT stavke_fk_racuni_<mbr> FOREIGN KEY(broj_racuna) REFERENCES
racuni_<mbr> (broj) ON DELETE CASCADE,
 CONSTRAINT stavke_chk_jm_<mbr> CHECK(jm IN ('kom', 'lit', 'kg')),
 CONSTRAINT stavke_chk_kolicina_<mbr> CHECK(kolicina > 0)
);
```

- Nedostaje stupac *ukupno* koji je potreban za okidač:

```
ALTER TABLE stavke_racuna_<mbr> ADD ukupno DECIMAL(8,2);
```

- Tablice su prazne:

```
select * from stavke_racuna_<mbr>;
select * from racuni_<mbr>;
```

- Popunjavanje tablice podacima (umjesto <*ime*> koristite ime po izboru):

```
INSERT INTO racuni_<mbr> VALUES(1, '01/01/2002', '<ime>');

INSERT INTO stavke_racuna_<mbr> VALUES(1, 1, 10, 'kom', 'jogurt', 2, NULL);
INSERT INTO stavke_racuna_<mbr> VALUES(1, 2, 12, 'kom', 'mlijeko', 2.5, NULL);
INSERT INTO stavke_racuna_<mbr> VALUES(1, 3, 5, 'kom', 'kruh', 4, NULL);
```

- Sada tablice sadržavaju podatke:

```
SELECT * FROM stavke_racuna_<mbr>;
SELECT * FROM racuni_<mbr>;
```

- Izradite okidač s početka primjera, provjerite sadržaj tablice *stavke_racuna*. Dodajte nove stavke u tablicu ili promijenite postojeće, npr.:

```
UPDATE stavke_racuna_<mbr> SET kolicina = 20 WHERE broj_racuna = 1 AND rb = 1;

SELECT * FROM stavke_racuna_<mbr> WHERE broj_racuna = 1;
```

6. Zadatak:

1. Kreirati korisničku funkciju (eng. *user function*), spremljenu proceduru (eng. *stored procedure*) i okidač (eng. *trigger*) koji će uzimati parametar u obliku niza znakova i okretati ga naopako, npr.:

ulazni parametar: "ivan" -> izlazni parametar: "navi".

Ukoliko je primljeni podatak palindrom (riječ koja se čita isto unatrag kao i od naprijed, npr. „Ana“, „kapak“ itd.) – ispiše se tekst „palindrom“. Ukoliko je unesena riječ duža od 30 znakova – ispiše se tekst „Ulazna riječ je predugačka, maksimalan broj znakova je 30“.

Cilj zadatka je pokazati na istom primjeru razlike/sličnosti između funkcije i procedure, te pokazati upotrebu okidača.

Objekte definirati na slijedeći način:

- funkcija prima znakovni podatak i vraća opisan izlaz
- procedura prima znakovni podatak i ispisuje opisan izlaz (ne vraća vrijednost)
- okidač se definira tako da se izvrši umjesto insert naredbe na tablici računi - naopako upiše ime kupca pri izvršavanju insert naredbe na tablici (ne provjerava se dužina riječi niti da li je riječ palindrom).

Isprobati rad sva tri objekta.

6.1. BONUS ZADATAK:

Kreirati spremljenu proceduru koja će uzimati datumski parametar i ispisivati horoskopski znak, npr. ulazni parametar: '16.1.1985' -> izlazni parametar: "jarac".

7. Assignment

Create a user function, stored procedure and a trigger takes a character array as a parameter and outputs it in reverse, for example:

input parameter: „John“ -> output: „nhoJ“.

If an input parameter is a palindrome (a word which is the same regardless of the direction of reading, e.g. „racecar“, „madam“, etc.) the output should be „An input parameter is a

palindrome“. If the input parameter is a word more than 30 characters long, the output should be „The input word is too long, maximum number of characters is 30.“.

The goal of this assignment is to show the similarities and differences between user functions, stored procedures and to show how to use triggers.

The database objects should be defined in a following way:

- A user function takes an input data as a parameter and returns the required output,
- A stored procedure takes an input data as a parameter and prints the required output (does not return anything),
- A trigger should be created so that it executes instead of the insert command on the table „Računi“ - prints out the buyers name in reverse when running the insert command on the table (does not check the word length nor if the word is a palindrome).

Test the functionality of all the objects.

7.1. BONUS ASSIGNMENT

Create a stored procedure wthich takes date parameter and prints out the matching horoscope signs (for example for '16.01.1985' print "capricorn").