Pogledi

- Pogledi predstavljaju funkcionalnost SQL-a koja omogućava definisanje specijalne reprezentacije jedne ili više tabela.
- Ovo znači da je korišćenjem pogleda moguće kreirati takozvane virtualne tabele, koje mogu biti kombinacija kolona koje pripadaju različitim tabelama. Nad pogledima je zatim moguće vršiti SELECT upite i, u zavisnosti od definicije pogleda i prava korisnika, izmene podataka INSERT, UPDATE i DELETE naredbama.

Zašto koristimo poglede?

- **Dva najbitnija razloga korišćenja pogleda su sledeći:**
- **Sigurnost**
  - Veoma često može postojati situacija u kojoj želimo da određenim korisnicima baze podataka onemogućimo pun pristup određenoj tabeli ili tabelama.
  - Tipičan primer ove situacije su tabele za smeštanje podataka o zaposlenima. Mi ćemo svakako želeti da omogućimo svim zaposlenima pristup njihovim ličnim podacima, kao što su ime, prezime, broj telefona ili adresa. Ipak, zaposlenima nije dobro omogućiti pristup podacima o visini zarada i sličnim osetljivim podacima.
- **Komfor**
  - Već smo videli da je neophodno često vršiti spajanje podataka više tabela. Na primer, u bazi podataka Sakila, ukoliko bismo želeli da dobijemo kompletan set podataka o mušterijama, morali bismo da izvršimo upit kojim bismo spojili podatke nekoliko tabela. Kako bi se olakšao rad korisnicima baze podataka ili programerima koji će pisati aplikacije koje će komunicirati sa bazom, mi možemo olakšati stvari i kreirati pogled.

Kreiranje pogleda

- Pogledi (View) su objekti u bazi podataka. I zbog toga se, kao i ostali objekti, mogu kreirati DDL naredbom

```
CREATE VIEW myView AS query
```

```
create view uzmifilmove
as
select film.*,film_actor.actor_id from film_actor
join film on film_actor.film_id = film.film_id
```

Ažuriranje izvornih podataka

- Pogledi, osim mogućnosti prikazivanja aktuelnog sadržaja po zadatoj formi, imaju i mogućnost ažuriranja izvora na kojima su formirani, ali pri tom moraju biti ispoštovana neka pravila. Ova pravila se odnose na kreiranje SELECT upita prilikom definisanja pogleda. Ovo znači da se prilikom kreiranja pogleda pri pisanju SELECT upita moraju poštovati sledeća pravila:
- SELECT upit ne sme sadržati ključne reči GROUP BY, DISTINCT, LIMIT, UNION ili HAVING
- Pogledi koji grupišu podatke iz više tabela gotovo nikada se ne mogu koristiti za izmenu konkretnih podataka, tj. SELECT upit mora baratati samo jednom tabelom
- pogled mora sadržati sve kolone određene tabele koje su definisane kao primarni ili strani ključevi

Prava

- Pogledi su usko vezani sa bezbednošću i pravima korisnika. Ovi pojmovi će biti detaljno objašnjeni u sledećem modulu, ali ćemo i sada, ovde, napraviti kratku demonstraciju.
- Prilikom kreiranja pogleda, kreator pogleda mora imati prava nad tabelama nad kojima se pogled kreira. Sa druge strane, korisnik ne mora imati SELECT prava nad tabelama, sve dok ima prava nad pogledom.

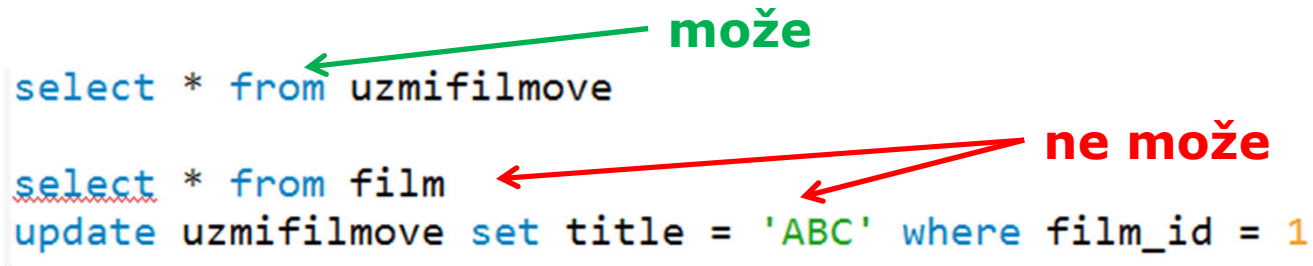
Vežba, kreiranje prava nad pogledom

- Kreirati korisnika John i dodeliti mu prava na pogled uzmifilmove

```
CREATE USER 'John'@'localhost' identified by '123'

grant select on uzmifilmove to John
```

- Ulogovati se kao John, a zatim isprobati sledeće upite:



Uskladištene rutine

- Uskladištene rutine su procedure i funkcije koje predstavljaju setove SQL naredbi koje su smeštene na serveru. Na ovaj način nije potrebno da klijenti ponavljaju određene naredbe više puta, već jednostavno mogu pozivati rutine
- Uskladištene rutine mogu biti posebno korisne u sledećim situacijama:
  - kada postoji više klijentskih aplikacija napisanih u različitim programskim jezicima, koje komuniciraju sa istom bazom podataka
  - kada je sigurnost imperativ; u bankarskim sistemima se na primer koriste uskladištene procedure i funkcije za sve operacije; ovo omogućava konzistentno i sigurno okruženje, a korišćenje rutina omogućava da je svaka operacija adekvatno logovana; u takvim scenarijima aplikacije i korisnici, nemaju direktan pristup tabelama baze podataka, već samo mogu da izvršavaju specifične uskladištene rutine

Stored procedures VS User-defined function

U sledećoj tabeli prikazane su osnovne razlike između procedura i funkcija

Stored procedures	**User-definied functions**
mogu vratiti null ili proizvoljan broj vrednosti	mogu vratiti samo jednu vrednost
mogu imati ulazne/uzlazne parametre	mogu imati samo ulazne parametre
moguće korišćenje SELECT i DML upita	moguće korišćenje samo SELECT upita
iz procedure mogu biti pozvane funkcije	iz funkcije ne može biti pozvana procedura
moguća obrada izuzetaka korišćenjem try-catch bloka	try-catch blokovi ne mogu biti korišćeni
ne mogu biti korišćene unutar SQL naredbi	mogu se koristiti unutar SQL naredbi
podržavaju transakcije	ne podržavaju transakcije

Uskladištene procedure

- Uskladištena (Stored) procedura je objekat na serveru, kao i svaki drugi. Stoga, da bismo je kreirali, koristimo DDL naredbu CREATE koja ima sledeću sintaksu:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body
```

```
DELIMITER //
    CREATE PROCEDURE my_procedure()
        BEGIN
            SELECT 'hello from stored procedure';
        END//
DELIMITER ;
```

Strukturni parametri procedure

- Procedura može biti izgrađena sa nekolicinom strukturnih parametara (**CONTAINS SQL, NO SQL, READS SQL DATA, MODIFIES SQL DATA**) koje opisuju prirodu procedure, ali nemaju sintaksnog uticaja na naše rukovanje njome, već pomažu samom MySQL-u prilikom kreiranja statistika i optimizacije:

```
CREATE PROCEDURE my_procedure ()
MODIFIES SQL DATA
BEGIN
//procedure body
END;
```

Determinističke i nedeterminističke procedure

Sličnu svrhu ima i parametar DETERMINISTIC (i NON DETERMINISTIC). Ovaj parametar, takođe, veoma utiče na rad optimizacionog Enginea, ali nema preteran uticaj na naše sintaksne obaveze prilikom pisanja procedure. Ako je procedura označena kao deterministic, znači da se njen izlaz nikada ne menja, kao u prvoj proceduri koju smo napisali, a koja emituje poruku *hello from stored procedure*. Ako procedura nije deterministik (što je podrazumevani parametar), MySQL ne očekuje da rezultat procedure bude uvek isti (na primer, ako procedura sadrži funkciju now(), koja prikazuje tačno vreme i naravno, nikada ne daje identičan rezultat).

Parametrizacija procedure

- Procedura može prihvatiti parametre, i to:
  - Ulazne parametre **(in)**
  - Ulazno izlazne parametre **(inout)**
  - Izlazne parametre **(out)**
- Podrazumevana vrsta (kretanje) parametara procedure su ulazni parametri **(in)**

```
create procedure myproc(p1 int, p2 int)
select p1+p2
```

Parametrizacija procedure

- Osim in parametara, procedura može imati i **out** i **inout** parametre
- Oba se ponašaju kao reference (na primer objektni parametri u Javi)

```
delimiter //
create procedure proba(out p1 int)
begin
set p1 = 25;
end//
delimiter ;

call proba(@a);
select @a;
```

Parametrizacija procedure

- Razlika između out i inout je u tome što **out** nije u stanju da prihvati ulaznu vrednost parametra, a **inout** jeste

Ne može, vraća null

```
delimiter //
create procedure proba(out p1 int)
begin
set p1 = p1 + 1;
end//
delimiter ;

set @a=1;
call proba(@a);
select @a;
```

Može, vraća 2

```
delimiter //
create procedure proba(inout p1 int)
begin
set p1 = p1 + 1;
end//
delimiter ;

set @a=1;
call proba(@a);
select @a;
```

Vežba 1

- Kreirati proceduru za unos korisnika, sa nazivom usp_insertuser koja kao parametar prihvata ime korisnika. Korisnik će biti unet sa statusom user (broj 1)

Vežba 1 - rešenje

```
delimiter //
create PROCEDURE insertuser(newname varchar(50))
BEGIN
    insert into users (name,status) values (newname,1);
END//
```

Vežba 2

- Potrebno je napraviti uskladištenu proceduru koja kreira korisnika prema parametrizovanom imenu. Ukoliko korisnik sa tim imenom već postoji u tabeli, neće biti kreiran. Korisnik se unosi sa statusom user.

Vežba 2 - rešenje

```
delimiter //
create procedure insertuser(newname varchar(50))
begin
declare usersCount int;
select count(*) from users where username = newname into usersCount;
select usersCount;
if usersCount<1 then
    insert into users values (null,newname);
    select '0';
else
    select '-1';
end if;
end //
delimiter ;
```

Vežba 3

- Potrebno je napraviti proceduru koja će unositi korisnika u bazu. Procedura prihvata kao parametre ime i šifru korisnika. Ukoliko korisnik sa tim imenom ne postoji, uneće novog. Ukoliko korisnik postoji, biće mu zamenjena šifra novom. Korisnik se unosi sa statusom user.

Vežba 3 - rešenje

```
delimiter //
create PROCEDURE insertuser(newname varchar(50),newpass varchar(50))
BEGIN
   declare usersCount int;
   select count(id) from users where name=newname into usersCount;
   if usersCount<1 then
    insert into users (name,password,status) values (newname,password,1);
   else
    update users set password=newpass where name = newname;
   end if;
END//
delimiter ;
```