LINKgroup

Distance Learning System

MySql

Transakcije

Transakcije

- Transakcije omogućavaju da se jedna ili više SQL komandi enkapsulira u jedinstvenu operaciju, odnosno celinu, te na taj način one predstavljaju važnu komponentu kada se govori o integritetu podataka u bazi
- Odgovor na pitanje zašto koristiti transakcije sadrži se u jednoj reči - **ACID**.
  - ***Atomicity*** (atomičnost ili nedeljivost) – ovaj pojam označava da je transakcija nedeljiva, tj. da se može izvršiti ili ne izvršiti; polovična izvršavanja nisu moguća
  - ***Consistency*** (konzistentnost)– ovaj pojam označava da će nakon izvršavanja transakcije baza biti u konzistentnom stanju; ukoliko transakcija izazove neko nedozvoljeno stanje ili neadekvatno ažuriranje podataka, sve će biti vraćeno na početno stanje
  - ***Isolation*** (izolacija) – ovaj pojam označava da se više transakcija može izvršavati simultano, tj. u isto vreme bez ikakvih loših posledica
  - ***Durability*** (izdržljivost) – pojam koji se odnosi na osobinu transakcija da budu sačuvane čak iako se odmah nakon njihovog završetka dogodi otkaz sistema i prekid rada baze

Kontrola transakcija

- Podrazumevano, MySQL server se nalazi u takozvanom autocommit modu. Ovo znači da se svaka SQL komanda izvršava u okviru jedne male transakcije. Zato se za naredbe može reći da se odigravaju transakciono, u smislu toga da će ili biti urađene, ili neće biti urađene, ali svakako, neće biti urađene napola. Ovo znači da ukoliko brišemo neki zapis iz baze podataka komandom DELETE mi u stvari izvršavamo jednu transakciju, (naravno pod uslovom da baratamo tabelom koja je tipa InnoDB).
- Najčešće će nam u realnom radu biti potrebne transakcije koje će se sastojati iz nekoliko SQL naredbi
- Ovako nešto, tj. izvršavanje više SQL naredbi u jednoj transakciji moguće je postići na dva načina. Prvi način podrazumeva isključivanje autocommit moda

Kontrola autocommit moda

- Da bismo isključili autocommit mod, koristimo sistemsku promenljivu **autocommit** (sistemske promenljive u SQL–u označavaju se sa @@). Kada je autocommit podešen na 0 (isključen), čak ni osnovne naredbe neće biti izvršene pre nego što mi to odobrimo. Drugim rečima, kada je autocommit mod isključen sve SQL naredbe se smatraju jednom transakcijom sve dok se transakcija ne potvrdi ključnom rečju COMMIT ili otkaže ključnom rečju ROLLBACK.

```
SET @@autocommit=0;
INSERT INTO country(country) VALUES('Serbia');
INSERT INTO country(country) VALUES('Bosnia');
ROLLBACK
```

Eksplicitna transakcija

- Ako pogledamo transakciju iz prethodnog primera, videćemo da ona nema početak. Njen početak je, zapravo, početak skripte. Sve nakon početka skripte do naredbe rollback ili commit spada pod jednu transakciju. Kada se aktivira neka od ove dve naredbe, transakcija ponovo počinje, sve do sledeće naredbe rollback ili commit. To je zato što su ove transakcije implicitne.
- Da bi jedna transakcija bila eksplicitna, potrebno je da naglasimo njen početak i kraj naredbama **starttransaction i rollback/commit:**

```
START TRANSACTION;
///sql dml
COMMIT/ROLLBACK
```

Kontrolne tačke

- Moguće je postaviti i kontrolne tačke transakcije i tako kontrolisati njen tok. Kontrolne tačke označavaju se ključnom rečju **savepoint**. Primer koji smo koristili u prethodnim situacijama sa definisanom kontrolnom tačkom bi izgledao ovako:

```
START TRANSACTION;
INSERT INTO country(country) VALUES('Serbia');
SAVEPOINT created_one_country;
INSERT INTO country(country) VALUES('Bosnia');
ROLLBACK TO SAVEPOINT created_one_country;
```

Konkurencija

- Problem konkurencije se može javiti prilikom korišćenja baze podataka od strane više klijenata:
  - Recimo da je upravo počela transakcija na bankomatu, novac je skinut sa računa i čeka se da bude isplaćen korisniku bankomata kako bi transakcija bila zatvorena. U tom trenutku, ako bismo pogledali stanje računa, videli bismo da je umanjeno, iako transakcija još uvek nije izvršena. Što znači, da stanje nije realno, jer novac još uvek nije u rukama korisnika. Šta bi video službenik u banci ako bi baš u tom trenutku odlučio da pogleda stanje računa tog korisnika? Na ovo pitanje postoji nekoliko odgovora koji zavise od načina na koji se transakcija izvršava. Postoji četiri moguća scenarija za ovakvu situaciju.

Konkurencija

- ***Prljavo čitanje* (*Dirty Read/Uncommited Read*)**
  - Ovo se događa ako jedna transakcija pristupa podacima koji su pod drugom transakcijom. Postoji realna mogućnost da ovi podaci nisu tačni što može dovesti do komplikacija prilikom rada jedne i druge transakcije.
- ***Non-Repeatable Read (Inconsistent Analysis)***
  - Slučaj je sličan prethodnom. Događa se ukoliko jedna transakcija ponavlja jedno isto čitanje, dok u međuvremenu, druga transakcija izvršava izmene na podacima, zbog čega dolazi do nepravilnih podataka u prvoj transakciji.
- ***Fantomsko čitanje (Phantom Read)***
  - Događa se kada se u toku jedne transakcije dodaju podaci od strane druge transakcije. To dovodi do pojave neidentifikovanog reda, a samim tim i nepravilnosti u radu.
- ***Izgubljeno ažuriranje (Lost Update)***
  - Kada dve transakcije ažuriraju podatke update naredbom, podaci bivaju ažurirani prema poslednjoj izvršenoj transakciji. Recimo da postoji neka vrednost u bazi: 10 i prva transakcija želi da podigne vrednost za 10, a druga transakcija hoće da je podigne za 20. Prva transakcija očekuje krajnju vrednost od 20, dok druga očekuje krajnju vrednost od 30. Na kraju, vrednost će biti 40, jer će se izvršiti prva transakcija (i podići vrednost za 10), a zatim i druga (koja će podići za još 20), što verovatno nije željeni rezultat

Zaključavanje (Locks)

- Nabrojane probleme konkurencije rešićemo na različite načine i to u zavisnosti od toga koji tip tabela koristimo. Svakako najelegantniji način za rešavanje pomenutih problema je upotreba transakcija. Ipak, kao što smo rekli, transakcije su podržane samo na tabelama tipa InnoDB, ne i MyISAM.
- Kod MyISAM tabela, kako bi se rešili pomenuti problemi, potrebno je pribeći tehnici koja se naziva zaključavanje. Kod MyISAM tabela, moguće je jedino primeniti zaključavanje na nivou cele tabele, i ovo prosto rečeno znači da će tabela koja je zaključana, tj. na kojoj je primenjen lock biti dostupna samo korisniku koji lock aktivirao na određeni vremenski period. Za vreme trajanja lock-a, drugi korisnici neće biti ustanju da izvrše bilo kakve promene na podacima zaključane tabele, a u nekim situacijama, u zavisnosti od tipa lock, neće biti u stanju ni da ih čitaju.

Zaključavanje tabela

- Da bi se jedna tabela rezervisala za ekskluzivno korišćenje od strane jednog korisnika, koristi se sintaksa LOCK TABLE[S].
- Prilikom pozivanja ove naredbe, postoji nekoliko tipova zaključavanja koje možemo aktivirati:
  - ***READ*** – svi MySQL korisnici mogu čitati podatke iz zaključane tabele, ali niko ne može da vrši izmene, uključujući i korisnika koji je aktivirao LOCK komandu
  - ***READ LOCAL*** – isto ako i prethodno opisani *READ*, s tim što je moguće izvršavanje ne konfliktnih INSERT upita
  - ***WRITE*** – samo korisnik koji je aktivirao lock može da čita i vrši izmene na tabeli, dok su svi ostali korisnici praktično blokirani i ne mogu da vrše na čitanje, ni pisanje
  - ***LOW PRIORITYWRITE*** – isto što i WRITE, stim što se ovakav lock ne može aktivirati sve dok se ne završe svi trenutno aktivni lock-ovi

- Kako bi zaključavanje tabele okončalo, odnosno kao bi se tabela otključala, koristi se naredba UNLOCK TABLE[S] bez dodatnih parametra. Ova komanda će okončati sve aktivne lock-ove trenutnog korisnika

Transakcije i zaključavanje

- Svakako najbolji način za rešavanje problema konkurencije je upotreba transakcija, naravno ukoliko koristimo InnoDB tip tabela. Transakcije nam omogućavaju mnogo bolju kontrolu zaključavanja, tako da nije potrebno zaključati kompletnu tabelu nad kojom se vrše operacije, već je zaključavanje moguće obaviti na mnogo nižem nivou, na nivou reda.
  InnoDB implementira dva osnovna oblika zaključavanja:
  - **shared** (deljeno)
  - **exclusive** (ekskluzivno)

Konekcija A	Konekcija B	Trenutak
USE innotest INSERT INTO table1 VALUES (1, 10) SELECT * FROM table1 colA = 1 colB = 10	USE innotest	Obe konekcije selektuju istu tabelu za korišćenje; konekcija A unosi vrednosti u table1, i to vrednosti 1 i 10 u kolone colA i colB, respektivno; unutar konekcije A vrši se selektovanje vrednosti koje su onakve kave su i insertovane
START TRANSACTION UPDATE table1 SET colB=11 WHERE colA=1		Konekcija A započinje transakciju; transakcija počinje naredbom za ažuriranje malopre unetih vrednosti, tako da vrednost kolone B umesto 10 bude 11
SELECT * FROM table1 colA = 1 colB = 11	SELECT * FROM table1 colA = 1 colB = 10	Drugi korak transakcije jeste selektovanje vrednosti koje su ažurirane prethodnom linijom; u tom trenutku konekcija B zahteva vrednosti kolona colA i colB; praktično ove SELECT naredbe se odvijaju u istom trenutku; konekcija A dobija vrednosti koje su ažurirane prethodnom linijom u transakciji, dok konekcija B dobija stare vrednosti
	BEGIN UPDATE table1 SET colB=colB+3 WHERE colA=1	Konekcija B započinje transakciju; unutar transakcije se vrši ažuriranje kolone colB tako da se njena vrednost uvećava za tri
COMMIT		Konekcija A aktivira naredbu COMMIT koja označava kraj transakcije
SELECT * FROM table1 colA = 1 colB = 11	`SELECT * FROM table1` `colA = 1 colB = 14`	Obe konekcije izvršavaju SELECT naredbu nad kolonama colA i colB, stim što konekcija B to obavlja unutar transakcije; konekcija A dobija vrednosti koje su ažurirane transakcijom ove konekcije, dok konekcija B dobija vrednosti koje su adekvatne promeni unutar transakcije koja još traje; pri tom je i uzeta u obzir promena koja je izvršena unutar transakcije konekcije B koja je zatvorena naredbom COMMIT, pa je vrednost kolone colB 10+1+3 = 14
	ROLLBACK	Konekcija B aktivira naredbu ROLLBACK i tako poništava efekat transakcije
SELECT * FROM table1 colA = 1 colB = 11	SELECT * FROM table1 colA = 1 colB = 11	Obe transakcije vrše SELECT podataka i dobijaju iste vrednovati

SELECT ... LOCK IN SHARE MODE

- Podrazumevano ponašanje InnoDB tabela je da se SELECT komande izvršavaju praktično istog trenutka, pa čak i nad blokiranim resursom. Jednostavno se ne uzima u obzir mogućnost da već postoji otvorena transakcija i da se na taj način mogu preuzeti neadekvatni podaci. Ovakvo ponašanje vidimo i u gornjoj tabeli, i to u trenutku vremena 2, kada konekcija B selektuje resurs koji izmenjen transakcijom koja postoji u konekciji A. Rezultat je različita vrednost za kolonu colB u odnosu na vrednost koju dobija konekcija A.
- Ukoliko se SELECT naredba izvrši sa dodatnom sintaksom LOCK IN SHARE MODE, tada će SELECT naredbe pre izvršena čekati sve dok se sve transakcije koje su u toku ne završe. Naravno pod uslovom da dolazi do sudaranja resursa, tj. da transakcija barata sa redovima koje SELECT naredbe zahteva

```
select * from country  LOCK IN SHARE MODE
```

SELECT ... FOR UPDATE

- Ovo je još jedna dodatna sintaksa koja se može iskoristiti za formulisanje SELECT upita. Ona se koristiti za postavljanje ekskluzivnog zaključavanja prilikom korišćenja SELECT upita, isto kao što se to podrazumevano događa prilikom UPDATE upita. Na taj način bismo mogli ovo iskoristiti unutar transakcije konekcije A i tada konekcija B ne bi mogla da dobije vrednost kolone colB odmah, i pored korišćenja SELECTG naredbe

Timeout i brzina pristupa

- Ponekad, obično prilikom loše realizovane konkurencije, dolazi do pojave **deadlock**-a. To je slučaj kada dve transakcije blokiraju jedna drugu. Recimo da jedna transakcija drži zaključan resurs, na koji druga transakcija čeka kako bi nastavila sa izvršavanjem. A ta druga transakcija drži zaključan resurs, koji je potreban prvoj transakciji kako bi nastavila sa izvršavanjem. Konačno, obe transakcije beznadežno čekaju jedna drugu da izađe iz zatvorenog kruga.
- InnoDB je u stanju da detektuje pojavu dead lock-a i u tom slučaju izvršiće automatski opoziv transakcija koje se u njemu nalaze. Ali, ponekad (kada u transakciji učestvuju i tabele koje nemaju ovaj mehanizam skladištenja ili kada su tabele zaključane ručno, lock table direktivom) InnoDB, jednostavno, ne detektuje ovakvu situaciju i nije u stanju da prepozna da su transakcije uzajamno zaključane.
- Zato je moguće odrediti vremenski period koji predstavlja to koliko će transakcije čekati da se ciljni resurs oslobodi, pre nego što ga napuste i budu opozvane.
- Atribut koji sadrži ovu vrednost je **innodb_lock_wait_timeout**, ali se do njega ne može doći kroz upite (na primer putem naredbe SET), već mora biti inicijalizovan prilikom startovanja (restarta) servera.

Izolacioni nivoi (Isolated Levels)

- Izolacioni nivoi utiču na način izvršavanja komandi koje se nalaze unutar transakcija. Drugim rečima, izolacioni nivoi su neka vrsta izolacionih šema pod kojima se transakcije izvršavaju

Izolacioni nivoi (Isolated Levels)

Izolacioni nivoi su sledeći:

- **READ UNCOMMITED**
  - Zaključavanje traje samo dok traju i izmene. Ako se dogode izmene u jednoj transakciji, odmah su vidljive i za druge transakcije, iako još uvek nisu prihvaćene. Zbog toga se ovaj izolacioni nivo naziva još i **dirty read**.
- **READ COMMITED**
  - Promene unutar transakcija vidljive su u drugim transakcijama tek nakon naredbe COMMIT, tj. uspešnog završetka transakcije. Ovo znači da će SELECT unutar transakcija može imati različite vrednosti u zavisnosti od toga da su podaci promenjenu u nekoj drugoj transakciji koja je završena. Za razliku od READ UNCOMMITED, izolacija SELECT naredbi je bolja, ali ne perfektna.
- **REPEATABLE READ**
  - Transakcija zaključava ekskluzivno resurse na kojima piše i deljeno (shared) resurse koje čita. Zaključavanje traje sve dok traje i transakcija, pa ne može doći do izmena u podacima u toku transakcije. Ovo je podrazumevani izolacioni nivo MySQL servera.
- **SERIALIZABLE**
  - Podrazumeva najviši stupanj izolacije. Svaka transakcija je izolovana i drži resurse ekskluzivno. Za razliku od izolacionog nivoa REPEATABLE READ, jedina razlika je ta što se SELECT komande automatski izvršavaju kao da su napisane upotrebom sintakse SELECT … LOCK IN SHARE MODE

```
SHOW GLOBAL VARIABLES LIKE 'tx_isolation';
SELECT @@global.tx_isolation;
```

Izolacioni nivoi (Isolated Levels)

Izolacioni nivo se podešava na nivou servera i može se videti naredbama:

```
SHOW GLOBAL VARIABLES LIKE 'tx_isolation';
SELECT @@global.tx_isolation;
```

Za izmenu izolacionog nivoa koristimo naredbu **set transaction isolation level**

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

Ovako napisana naredba važiće samo na nivou trenutne konekcije. I to ima identičan rezultat kao da smo je napisali i u obliku

```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

Ako bismo želeli da izolacioni nivo važi za sve konekcije na nivou servera, morali bismo da ga deklarišemo kao globalni

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

Vežba 1

- Postoje sledeće dve tabele

```
create table orders
(
    id int primary key auto_increment,
    code varchar(5)
);

create table basket
(
    id int primary key auto_increment,
    code varchar(5)
);
```

Potrebno je napraviti proceduru koja će u transakciji preneti određeni red iz tabele basket u tabelu orders

Rešenje

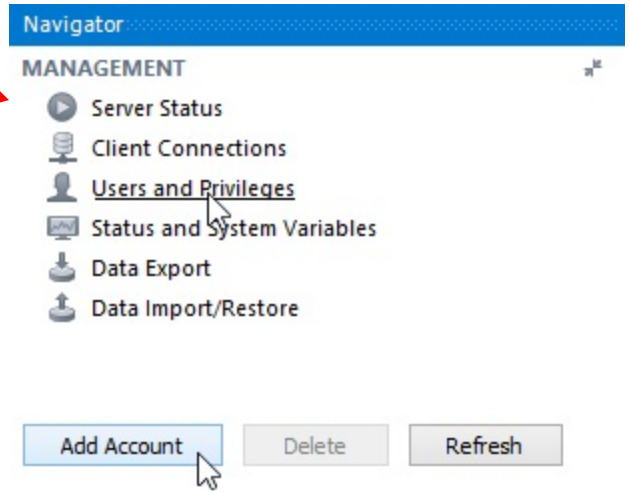
```
delimiter //
create PROCEDURE submitOrder(newcode varchar(5))
BEGIN
start transaction;
      insert into orders (code) select code
        from basket where code = newcode;
     delete from basket where code = newcode;
commit;
END//
```

Korisnici i prava pristupa

- Do sada, gotovo sve manipulacije nad bazama i podacima vršili smo korišćenjem jednog korisničkog naloga. To je korisnik root
- Ovaj korisnik automatski biva kreiran prilikom instalacije MySQL-a i da se on smatra super administratorom.
- U realnim uslovima kreiraćemo konkretne korisnike za različite tipove korisnika ili aplikacija koje će komunicirati sa bazom podataka. Različite vrste korisnika će imati i različita prava pristupa objektima na serveru.

Kreiranje korisnika

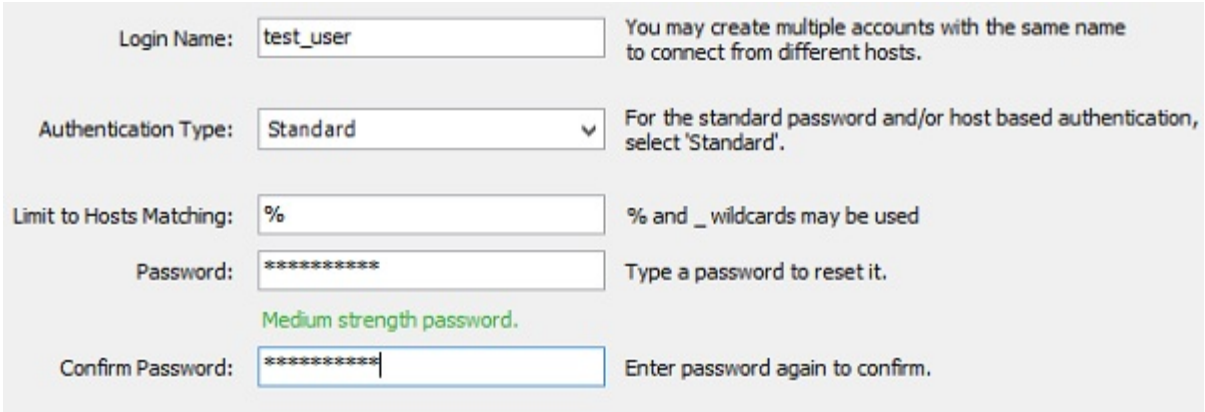
- Da bismo novog korisnika kreirali korišćenjem MySQL Wokbeanch alata, potrebno je prvo izvršiti konektovanje na server korišćenjem root naloga. Nakon ovoga dovoljno je odabrati opciju Users and Privileges iz odeljka Management sa levog dela prozora



- Nakon ovoga potrebno je odabrati opciju Add Account sa dna prozora

Kreiranje korisnika

- U polja unesite odgovarajuće podatke za ime korisnika i njegovu šifru, a zatim pritisnite taster Apply



- Programabilno je moguće kreirati istog korisnika, sledećom naredbom:

```
CREATE USER 'test_user'@'%' IDENTIFIED BY '123'
```

Kreiranje korisnika

- Host, u ovom kontekstu, predstavlja adresu sa koje će korisnik moći da pristupa serveru. Karakter % označava da korisnik može pristupiti serveru sa bilo koje adrese. Sa sigurnosnog aspekta ovo nije dobra praksa, obzirom da je nabolje prihvatati korisnike samo sa poznatih host adresa. Kada bismo hteli da ispoštujemo ovo pravilo, mogli bismo da napišemo

```
CREATE USER 'test_user'@'209.85.135.106' IDENTIFIED BY '123'
```

Kako je serveru rečeno ime korisnika, šifra, host i ništa više, on podrazumeva da taj korisnik nema nikakva prava na samom serveru. Zato, nakon naše naredbe CREATEUSER... server, u stvari, izvršava sledeći upit u bazi MySQL:

```
INSERT INTO user VALUES('%', 'test_user',password('123'),
'N','N','N','N','N','N','N','N','N','N','N','N','N','N','N',
'N','N','N','N','N','N','N','N','N','N','N','',1,1,1,0,0,0,0);
```

Prava pristupa

- Korisnik se podrazumevano kreira bez privilegija
- Ako hoćemo da mu dodelimo privilegije, koristimo DCL naredbu **grant**:

```
GRANT SELECT, UPDATE, CREATE, DELETE ON *.* TO 'test_user'@'%'
```

- Za oduzimanje prava korisniku, koristi se DCL naredba **revoke**:

```
REVOKE SELECT, UPDATE, CREATE, DELETE ON *.* FROM 'test_user'@'%'
```

- Za dodavanje svih privilegija:

```
GRANT ALL PRIVILEGES ON *.* TO 'test_user'@'%'
```

Prava pristupa

- Dodeljivanje privilegija u MySQL počiva na relacionom principu. Određeni objekti sistema imaju definisane setove prava za neke druge objekta u sistemu.
- Ovakav tip menadžmenta pristupa naziva se **Access Control List**.

Vežba 1

- Potrebno je kreirati bazu podataka application_04.
  Ova baza treba da ima tri korisnika:
- *app_04_admin*, sa šifrom *xyz* koji će moći da se uloguje samo sa lokalnog računara i imaće sva prava nad bazom
- *app_04_user*, sa šifrom *123*, koji će moći da se uloguje samo sa određene IP adrese i imaće mogućnost select, insert, update i delete naredbi u bazi
- *app_04_viewer*, sa šifrom *abc* koji će moći da se uloguje sa svih IP adresa i imaće samo mogućnost select naredbe u bazi

Rešenje

```
create database application_04;
create user 'app_04_admin'@'localhost' identified by 'xyz';
grant all privileges on application_04.* to
'app_04_admin'@'localhost';
create user 'app_04_user'@'ip address' identified by '123';
grant select,insert,update,delete on application_04.* to
'app_04_user'@'ip address'
create user 'app_04_vewer'@'%' identified by 'abc';
grant select on application_04.* to 'app_04_viewer'@'%'
```

Vežba 2

- Potrebno je izmeniti prava dodeljena u prethodnoj vežbi, tako da:
- korisnik *user_04_user*, ima prava samo na *select* naredbu
- korisnik *user_04_viewer* više ne postoji
- korisnik *user_04_admin* ima prava samo na *select, update, delete* i *insert*

Rešenje

show grants for 'app_04_user'@'ip address';

revoke update, insert, delete on *.* from 'app_04_user'@'ip address';

drop user 'app_04_viewer'@'ip address';

show grants for 'app_04_admin'@'localhost';

revoke all on application_04.* from 'app_04_admin'@'localhost';

grant select,insert,update,delete on application_04.* to 'app_04_admin'@'localhost'

Bekap mysql baze

- Full backup
  - **mysqldump**
    - ***mysqldump -ubekaper -p123 sakila > sakila_bkp.sql***
- Full restore

- Bekap loga
  - Log će automatski biti restoreovan ukoliko je u konfiguraciji aktivirana opcija: log-bin=mysql-bin

```
# Replication Master Server (default)
# binary logging is required for replication
log-bin=mysql-bin
```

Replikacija

- Replikacija podrazumeva dve identične baze (ili više njih) od kojih je jedna distribuira izmene ostalima

https://www.digitalocean.com/community/tutorials/how-to-set-up-master-slave-replication-in-mysql