2-7 Napredne teme

U ovom dijelu, dotaknut ćemo se nekih naprednih tema u SQLu bez detaljnije analize.

PODUPITI

Vidjeli smo da nam `SELECT` upiti kao rezultat daju rezultni skup. Ovaj izlazni skup je zapravo tabela. Ono što bi bilo zgodno je da se ova tabela može koristiti kao svaka druga tabela. Ovo je zapravo moguće ako umjesto naziva neke druge tabele koristimo novi `SELECT` upit unutar zagrada. Podupiti se mogu koristiti u `SELECT`, `FROM` i `WHERE` dijelovima.

Recimo da želimo pronaći sve studente koji su se upisali na na najkasniji dostupni datum:

```
SELECT *
FROM studenti
WHERE upis = (SELECT MAX(upis) FROM studenti)
```

Ili ako želimo da pronađemo najstariji datum rođenja od svih studenata koji su se zadnji upisali

```
SELECT MIN(datum_rodjenja)
FROM (SELECT *
      FROM studenti
      WHERE upis = (SELECT MAX(upis) FROM studenti))
```

Isto tako, podupiti se mogu koristiti unutar SELECT izraza, npr. broj ocjena svakog studenta:

```
SELECT s.id, s.ime, s.prezime,
      (SELECT COUNT(ocjena)
       FROM ocjene o WHERE o.student_id = s.id) brocjena
FROM studenti s
```

HAVING

`HAVING` je klauzula kojom se filtriraju grupe. Koristi se isključivo kod agregacijskih slučajeva, tj. kada postoji potreba filtrirati `GROUP BY` rezultate. U prevodu, kako `WHERE` klauzula filtrira redove, `HAVING` filtrira rezultate agregacijskih upita. Obje vrste filtriranja se mogu koristiti u istom upitu. Npr. pronaći sve predmete koji za broj ocjena sa komentarom je veći od 1 sortiran po datumu:

```
SELECT *, COUNT(*) as brocjena
FROM ocjene
WHERE komentar IS NOT NULL
GROUP BY predmet_id
HAVING brocjena>1
ORDER BY datum
```

Transakcije

Transakcije su karakteristika baza podataka koje omogućavaju konsistentnost podataka. Sve baze podataka koje implementiraju transakcije, to ostvaruju tako što zadovoljavaju ACID pravila. U suštini, zadovoljavanje ovih pravila, garantuje konsistentnost i trajnost podataka. Npr. ako Alisa želi da pošalje Bakiru 100KM, onda se mogu napisati sljedeće komande:

```
UPDATE saldo SET saldo=saldo-100 WHERE user='Alisa'
UPDATE saldo SET saldo=saldo+100 WHERE user='Bakir'
```

Međutim, šta ako nakon izvršavanja prve komande, a prije izvršavanja druge komande dođe do neke greške, disk crkne, struje nestane, kosmički zraci prouzrokuju nepoznatu grešku ili jednostavno neko nasilno ugasi bazu podataka? Onda će Alisini novci biti izgubljeni, Bakir neće dobiti novac a povjerenje u bazu podataka nikad neće biti povraćeno. Zato, za ove svrhe, koriste se transakcije.

Transakcija je lista komandi za koje baza garantuje da će se ili sve izvršiti ili neće nijedna. Npr. ako transakcija ima četri upita, i tokom izvršavanja trećeg upita, desi se neka greška, baza podataka će povratiti sve izmjene načinjene upitima jedan i dva. Npr.

```
START TRANSACTION;
      UPDATE izvod SET saldo=saldo-100 WHERE korisnik='Alisa';
      UPDATE izvod SET saldo=saldo+100 WHERE korisnik='Bakir';
COMIT;
```

Ako se ponvi prethodni scenarij, da dođe do prekida rada baze između dvije komande, i da se transakcija u potpunosti ne izvrši, onda baza podataka garantuje da će vratiti izmjene učinjene prvom komandom. Drugim riječima, ako se oduzme novac od Alise, a iz nekog nepoznatog razloga ne prebaci Bakiru, i transakcija ne uspije, onda će se oduzeti novac vratiti Alisi.

ROLLBACK

Ukoliko se, tokom izvršenja transakcije, desi greška, ili nisu zadovoljeni određeni uvijeti, moguće je izmjene napravljene tokom transakcijom vratiti. Npr.:

```
START TRANSACTION;
      UPDATE izvod SET saldo=saldo-100 WHERE korisnik='Alisa';
      UPDATE izvod SET saldo=saldo+100 WHERE korisnik='Bakir';
ROLLBACK;
```

Kada se `ROLLBACK` izvrši, sve napravljene izmjene bit će resetovane. To svojevrsna *undo* opcija dostupna za transakcije. Ovo je dostupno u slučaju da korisnik želi, do sada napravljene izmjene unutar transakcije, poništiti.

FOR UPDATE

U prethodnom primjeru prebacivanja novca od Alise ka Bakiru nije bilo značajnijih promjena osim samog iznosa. Tako da kada bi dva klijenta zatražile istu transakciju od baze, onda bi ta interakcija na kraju dala isti rezultat bez obzira kakav redoslijed bio komandi koje se izvršavaju. Zapravo dvije transakcije se mogu izvršavati paralelno, stoga ako recimo imamo dvije transakcije gdje obje imaju dva koraka:

Transakcija A:
1. Smanji Alisin iznos za 100
2. Povećaj Bakirov iznos za 100

Transakcija B:
1. Smanji Alisin iznos za 300
2. Povećaj Bakirov iznos za 300

Pošto je ovo samo primjer, razlog zbog kojeg je nastao ovakav slučaj nije bitan. Možda je Alisa napravila dvije uplate u dvije različite poslovnice banaka i kasnije tog dana, bankovni sistem procesira naloge uplate, te se u tom trenutku nađu ove dvije transakcije ***u isto vrijeme***. Ove dvije transakcije baza može izvršiti paralelno, u isto vrijeme. Tako da su neke od ovih kombinacija redoslijeda izvršavanja, sasvim moguće:

1. A1, B1, A2, B2
2. B1, A1, A2, B2
3. B1, A1, B2, A2, itd…

Naravno, komande iz iste transakcije će se uvijek izvršavati istim redoslijedom. Ali ne postoji garancija da će se prvo izvršiti transakcija A, pa onda transakcija B, ili obratno. Međutim, postoje slučajevi kada je ova garancija neophodna. Npr. ako imamo sljedeći primjer dvije transakcije:

Transakcija A:
1. Pronaći koliko novca ima korisnik 1
2. Ako ima dovoljan iznos, prebaciti određen iznos na račun korisnika 2

Transakcija B:
1. Pronaći koliko novca ima korisnik 1
2. Ako ima dovoljan iznos, prebaciti određen iznos na račun korisnika 3

Npr. potrebno je da korisnik 1 prebaci 100 novaca korisnicima 2 i 3. Međutim, postoji korak provjere da li korisnik 1 ima dovoljno novca. Ukoliko se dopusti scenarij iz prethodnog primjera onda je moguće da se desi A1, B1, A2, B2. Međutim, šta ako korisnik 1 u zbiru ima 100 novaca? U tom slučaju A1 i B1 će kao rezultat pronaći da postoji dovoljno novca, i obje transakcije će se nastaviti izvršavati i tako će isti iznos biti potrošen dva puta. Ovo znači da će naš sistem biti neispravan.

Kako bi se riješio ovaj problem, baze podataka posjeduju sposobnost zaključavanja tabela i redova protiv modifikacije od strane drugih transakcija. Stoga, u ovom primjeru, ukoliko A1 zaključa mogućnost skidanja novca sa računa korisnika 1, onda će transakcija B morati čekati da se otključa taj korisnik, kako bi se mogla nastaviti izvršavati.

Zaključavanje redova se može postići koristeći klauzulu `FOR UPDATE`. Npr. ako u upitu u transakciji stavimo:

```
SELECT iznos FROM korisnici WHERE id=1 FOR UPDATE;
```

Onda transakcija B neće moći koristiti ovaj red u tabeli za izmjene sve dok se transakcija A ne završi. Kada se transakcija B nastavi izvršavati nakon transakcije A, vidjet će da korisnik 1 nema dovoljno sredstava na računu da napravi isplatu. Podaci o tekućim transakcijama se mogu pronaći u bazi `information_schema` u tabelama `INNODB_LOCKS` i `INNODB_LOCK_WAITS`.

Okidači

Okidači su izrazi koji se izvršavaju, ukoliko su definisani, prije ili poslije naredbi `INSERT`, `UPDATE` ili `DELETE` nad redovima u bazi podataka. Npr.:

```
CREATE TRIGGER povecaj_brstudenata
AFTER INSERT ON studenti
FOR EACH ROW
UPDATE statistika SET statistika.brstudenata=statistika.brstudenata+1
```

Ova komanda će kreirati okidač za svaki unos u tabelu studenti, i za svaki red. Priloženi upit će povećati broj studenata u drugoj tabeli *statistika*. Priloženi upit je mogao biti bilo koji validni SQL upit.

Okidači se mogu i obrisati:

```
DROP TRIGGER povecaj_brstudenata;
```