MY SQL

Crash Course

Kurs je napravljen u svrhu brzog prelaska osnovnih operacija i komandi u mySQL-u
Kod u primjerima napisan obojen je u zeleno kao i svi bitni pojmovi
kao izvor je korišteno više literatura te stoga bih vas zamolio da zanemarite gramaticke greske i tome slično.

MY SQL Tipovi podataka

MySQL podržava širok spektar tipova podataka koji omogućavaju skladištenje različitih vrsta informacija. Tipovi podataka u MySQL-u se mogu podeliti u nekoliko kategorija:

1. **Numerički tipovi podataka**
2. **Tipovi podataka za rad sa tekstom (stringovima)**
3. **Datum i vreme**
4. **Tipovi podataka za BLOB (Binary Large Object)**
5. **Logički tipovi podataka**

Sada ćemo detaljnije opisati ove tipove podataka.

---

1. Numerički tipovi podataka

a) TINYINT

- **Opis**: Koristi se za skladištenje malih celobrojnih vrednosti.
- **Opseg**:
  - `TINYINT` (bez potpisivanja): -128 do 127
  - `TINYINT UNSIGNED` (bez potpisivanja): 0 do 255

b) SMALLINT

- **Opis**: Koristi se za skladištenje celobrojnih vrednosti srednjeg opsega.
- **Opseg**:
  - `SMALLINT` (bez potpisivanja): -32,768 do 32,767
  - `SMALLINT UNSIGNED` (bez potpisivanja): 0 do 65,535

c) MEDIUMINT

- **Opis**: Koristi se za skladištenje celobrojnih vrednosti većeg opsega.
- **Opseg**:
  - `MEDIUMINT` (bez potpisivanja): -8,388,608 do 8,388,607
  - `MEDIUMINT UNSIGNED` (bez potpisivanja): 0 do 16,777,215

d) INT (INTEGER)

- **Opis**: Standardni tip za skladištenje celobrojnih vrednosti.
- **Opseg**:
  - `INT` (bez potpisivanja): -2,147,483,648 do 2,147,483,647
  - `INT UNSIGNED` (bez potpisivanja): 0 do 4,294,967,295

e) BIGINT

- **Opis**: Koristi se za skladištenje veoma velikih celobrojnih vrednosti.

- **Opseg**:
  - `BIGINT` (bez potpisivanja): -9,223,372,036,854,775,808 do 9,223,372,036,854,775,807
  - `BIGINT UNSIGNED` (bez potpisivanja): 0 do 18,446,744,073,709,551,615

f) FLOAT

- **Opis**: Koristi se za skladištenje brojeva sa pomičnim zarezom (decimalni brojevi sa manjom preciznošću).
- **Opseg**: Ovaj tip može da skladišti brojeve sa do 24 cifre preciznosti.

g) DOUBLE

- **Opis**: Koristi se za skladištenje brojeva sa pomičnim zarezom (decimalni brojevi sa većom preciznošću od `FLOAT`).
- **Opseg**: Ovaj tip može da skladišti brojeve sa do 53 cifre preciznosti.

h) DECIMAL (ili NUMERIC)

- **Opis**: Koristi se za tačne decimalne vrednosti sa fiksnom tačnošću. Idealno je za finansijske podatke.
- **Sintaksa**: `DECIMAL(M, D)` gde `M` označava ukupni broj cifara, a `D` označava broj cifara nakon decimale. Na primer, `DECIMAL(10,2)` omogućava skladištenje brojeva sa 10 cifara, od kojih 2 mogu biti nakon decimalne tačke.

---

2. Tipovi podataka za rad sa tekstom (stringovima)

a) CHAR

- **Opis**: Skladišti fiksne dužine stringove.
- **Opseg**: Od 0 do 255 karaktera.
- **Napomena**: Ako unesete string kraće od definisane dužine, ostatak prostora se popunjava razmacima.

b) VARCHAR

- **Opis**: Skladišti varijabilnu dužinu stringova.
- **Opseg**: Do 65,535 karaktera (zavisi od veličine drugih kolona u tabeli).
- **Napomena**: Za razliku od `CHAR`, `VARCHAR` skladišti samo toliko prostora koliko je potrebno za unos (bez dodatnog popunjavanja).

c) TEXT

- **Opis**: Skladišti velike količine tekstualnih podataka.
- **Opseg**: Do 65,535 karaktera.
- **Napomena**: Koristi se za skladištenje većih tekstualnih podataka kao što su opisi ili dugi komentari.

d) TINYTEXT

- **Opis**: Skladišti male tekstualne podatke.
- **Opseg**: Do 255 karaktera.

e) MEDIUMTEXT

- **Opis**: Skladišti tekstualne podatke srednje veličine.
- **Opseg**: Do 16,777,215 karaktera.

f) LONGTEXT

- **Opis**: Skladišti vrlo velike količine tekstualnih podataka.
- **Opseg**: Do 4,294,967,295 karaktera.

---

3. Datum i vreme

a) DATE

- **Opis**: Skladišti datum u formatu `YYYY-MM-DD`.
- **Opseg**: Od '1000-01-01' do '9999-12-31'.

b) DATETIME

- **Opis**: Skladišti datum i vreme u formatu `YYYY-MM-DD HH:MM:SS`.
- **Opseg**: Od '1000-01-01 00:00:00' do '9999-12-31 23:59:59'.

c) TIMESTAMP

- **Opis**: Skladišti datum i vreme kao broj sekundi od '1970-01-01 00:00:00' UTC.
- **Opseg**: Od '1970-01-01 00:00:01' do '2038-01-19 03:14:07'.
- **Napomena**: Koristi se za praćenje vremenskih oznaka.

d) TIME

- **Opis**: Skladišti samo vreme u formatu `HH:MM:SS`.
- **Opseg**: Od '-838:59:59' do '838:59:59'.

e) YEAR

- **Opis**: Skladišti godinu u formatu `YYYY`.
- **Opseg**: Od 1901 do 2155.

---

4. Tipovi podataka za BLOB (Binary Large Object)

a) BLOB

- **Opis**: Skladišti binarne podatke (kao što su slike, video, zvučni fajlovi).
- **Opseg**: Do 65,535 bajtova.

b) TINYBLOB

- **Opis**: Skladišti male binarne podatke.
- **Opseg**: Do 255 bajtova.

c) MEDIUMBLOB

- **Opis**: Skladišti srednje velike binarne podatke.
- **Opseg**: Do 16,777,215 bajtova.

d) LONGBLOB

- **Opis**: Skladišti velike binarne podatke.
- **Opseg**: Do 4,294,967,295 bajtova.

---

5. Logički tipovi podataka

a) BOOLEAN

- **Opis**: Predstavlja logičke vrednosti `TRUE` (1) i `FALSE` (0).
- **Napomena**: MySQL zapravo koristi `TINYINT(1)` za skladištenje vrednosti tipa `BOOLEAN`.

b) BIT

- **Opis**: Skladišti binarne vrijednosti u obliku bitova.
- **Opseg**: Do 64 bita.

Zaključak

MySQL nudi bogat set tipova podataka koji omogućavaju efikasno skladištenje različitih vrsta podataka. Ključ je odabrati pravi tip podataka za svaku kolonu u tabeli kako bi se osigurala efikasnost, tačnost i optimalna upotreba resursa. Na primer, za numeričke vrednosti možete koristiti `INT` ili `DECIMAL` za precizne vrednosti, dok za tekstualne podatke koristite `VARCHAR` ili `TEXT` zavisno od dužine podataka.

Osnove my sql i komande

1. Šta je MySQL?

MySQL je **relacijski sistem za upravljanje bazama podataka (RDBMS)** koji koristi SQL (Structured Query Language) za manipulaciju podacima. Koristi se za pohranu podataka u tablicama i omogućava operacije poput umetanja, ažuriranja, brisanja i pretrage podataka.

2. Instalacija MySQL-a

Da biste koristili MySQL, morate ga instalirati na svom računaru ili koristiti MySQL server na mreži. Možete preuzeti instalacijski paket sa zvanične stranice: MySQL download.

Online MySQL editor i kompajler

3. Osnovne komande u MySQL-u

3.1 Prijava u MySQL

Nakon instalacije, za prijavu u MySQL koristite sljedeću komandu u terminalu (ako koristite Linux/macOS) ili CMD/PowerShell (ako koristite Windows):

```
mysql -u root -p
```

Ovdje:

- `-u root` označava korisničko ime (root je podrazumijevano administratorsko ime).
- `-p` znači da ćete biti traženi da unesete lozinku.

3.2 Kreiranje baze podataka

Da biste kreirali novu bazu podataka, koristite komandu `CREATE DATABASE`:

```
CREATE DATABASE ime_baze;
```

3.3 Odabir baze podataka

Da biste odabrali bazu podataka sa kojom ćete raditi, koristite komandu `USE`:

```
USE ime_baze;
```

3.4 Kreiranje tabele

Tablica u MySQL-u je strukturirani skup podataka organizovan u redove i kolone. Komanda za kreiranje tabele izgleda ovako:

```
CREATE TABLE ime_tabele (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ime VARCHAR(100),
    prezime VARCHAR(100),
    godine INT
);
```

Ovdje:

- `id` je primarni ključ, koji je automatski inkrementiran.
- `ime`, `prezime` i `godine` su kolone koje pohranjuju podatke.

3.5 Umetanje podataka u tabelu

Za unos podataka u tabelu koristimo komandu `INSERT INTO`:

```
INSERT INTO ime_tabele (ime, prezime, godine) VALUES ('Marko',
'Marković', 25);
```

3.6 Odabir podataka

Za pretragu podataka u tabeli koristimo komandu `SELECT`:

```
SELECT * FROM ime_tabele;
```

Ovo će prikazati sve podatke u tabeli.

Ako želite da filtrirate podatke, možete koristiti `WHERE`:

```
SELECT * FROM ime_tabele WHERE godine > 20;
```

3.7 Ažuriranje podataka

Za ažuriranje postojećih podataka koristimo `UPDATE`:

```
UPDATE ime_tabele SET godine = 26 WHERE ime = 'Marko';
```

3.8 Brisanje podataka

Za brisanje podataka koristi se komanda `DELETE`:

```
DELETE FROM ime_tabele WHERE id = 1;
```

3.9 Brisanje tabele

Da biste obrisali celu tabelu, koristite komandu `DROP`:

```
DROP TABLE ime_tabele;
```

4. Napredne funkcionalnosti

4.1 Spajanje tabela (JOIN)

Ako imate više tabela i želite da ih spojite prema nekom zajedničkom atributu, koristite `JOIN`. Na primjer:

```
SELECT korisnici.ime, narudzbe.proizvod
FROM korisnici
JOIN narudzbe ON korisnici.id = narudzbe.korisnik_id;
```

4.2 Grupa i agregatne funkcije

Za grupisanje podataka i korištenje funkcija poput `SUM`, `AVG`, `COUNT` itd., koristite `GROUP BY`:

```
SELECT AVG(godine) FROM ime_tabele GROUP BY prezime;
```

4.3 Indeksi

Indeksi omogućavaju bržu pretragu podataka. Da biste kreirali indeks na određenoj koloni, koristite:

```
CREATE INDEX indeks_na_imenu ON ime_tabele(ime);
```

4.4 Relacije između tabela (Foreign Keys)

Da biste uspostavili relaciju između dvije tabele (npr. korisnici i narudžbe), koristite `FOREIGN KEY`:

```
CREATE TABLE narudzbe (
    id INT AUTO_INCREMENT PRIMARY KEY,
    korisnik_id INT,
    proizvod VARCHAR(100),
    FOREIGN KEY (korisnik_id) REFERENCES korisnici(id)
);
```

5. Backup i Restore baze podataka

Za backup baze podataka, koristite komandu `mysqldump`:

```
mysqldump -u root -p ime_baze > backup.sql
```

Za obnovu baze podataka iz backupa:

```
mysql -u root -p ime_baze < backup.sql
```

6. Koristan Savjet

- **Kreirajte sigurnosne kopije:** Uvijek pravite backup baze podataka, posebno prije velikih promjena.
- **Koristite indeksiranje:** Indeksi poboljšavaju performanse, posebno kod velikih tabela.
- **Izbjegavajte SELECT :* Uvijek odaberite samo potrebne kolone umjesto korištenja `*` (što znači sve kolone), kako biste poboljšali performanse.

7. Zaključak

Ovo je osnovni uvod u MySQL. Uz ove komande, možete kreirati, manipulirati i pretraživati podatke u MySQL bazama podataka. Naravno, MySQL nudi mnogo naprednih opcija i funkcionalnosti, ali ovo je dobar početak.

Ako imate konkretna pitanja ili želite da detaljnije istražiti neku temu, slobodno pitajte!

Projekcija i selekcija

Projekcija u kontekstu MySQL-a (i u SQL-u općenito) se odnosi na selektovanje specifičnih kolona iz tabele ili rezultata upita. To znači da projekcija omogućava da odaberete samo one kolone koje su vam potrebne u rezultatu, umesto da vraćate sve kolone iz tabele. Projekcija se postiže korišćenjem **`SELECT`** klauzule u SQL upitu.

1. Osnovna projekcija - Selektovanje specifičnih kolona

Da biste izabrali samo određene kolone iz tabele, koristite **`SELECT`** klauzulu i navedite nazive kolona koje želite da se prikažu.

Primer:

Ako imamo tabelu `korisnici` koja sadrži kolone `id`, `ime`, `prezime`, `email` i `datum_registracije`, a želimo prikazati samo `ime` i `prezime` svih korisnika, koristimo projekciju na sledeći način:

```
SELECT ime, prezime
FROM korisnici;
```

Ovaj upit će vratiti samo kolone `ime` i `prezime` za sve redove u tabeli `korisnici`.

2. Projekcija svih kolona - `*`

Ako želite da prikažete sve kolone iz tabele, možete koristiti znak `*`, koji označava sve kolone.

Primer:

```
SELECT *
FROM korisnici;
```

Ovaj upit će prikazati sve kolone iz tabele `korisnici` za sve redove.

3. Projekcija sa izrazima (Izračunavanje novih vrednosti)

Projekcija može uključivati i izraze koji računaju nove vrednosti na temelju postojećih podataka u tabeli. Na primer, možete koristiti matematičke operacije, string funkcije ili date funkcije kako biste izračunali nove kolone.

Primer:

Ako želite da prikažete ime korisnika, prezime i broj dana od njihove registracije (na osnovu trenutnog datuma), možete koristiti projekciju sa funkcijama:

```
SELECT ime, prezime, DATEDIFF(CURDATE(), datum_registracije) AS
broj_dana_od_registracije
FROM korisnici;
```

Ovaj upit prikazuje ime, prezime i broj dana od registracije svakog korisnika, koristeći SQL funkciju `DATEDIFF` za izračunavanje razlike između trenutnog datuma (`CURDATE()`) i datuma registracije.

4. Projekcija sa agregatnim funkcijama

Kada koristite agregatne funkcije (kao što su `SUM()`, `COUNT()`, `AVG()`, `MIN()`, `MAX()`), često ćete koristiti projekciju kako biste prikazali agregirane rezultate.

Primer:

Ako želite da prikažete broj narudžbi za svakog korisnika:

```
SELECT korisnik_id, COUNT(*) AS broj_narudzbi
FROM narudzbe
GROUP BY korisnik_id;
```

Ovaj upit koristi projekciju da prikazuje korisnički ID i broj narudžbi (agregiran sa `COUNT(*)`) za svakog korisnika.

5. Projekcija sa filtriranjem - `WHERE` klauzula

Projekcija se često koristi zajedno sa **`WHERE`** klauzulom, koja omogućava filtriranje podataka pre nego što se prikažu.

Primer:

Ako želite da prikažete samo korisnike koji su se registrovali pre više od 30 dana:

```
SELECT ime, prezime, datum_registracije
FROM korisnici
WHERE DATEDIFF(CURDATE(), datum_registracije) > 30;
```

Ovaj upit prikazuje ime, prezime i datum registracije samo onih korisnika koji su se registrovali pre više od 30 dana.

6. Projekcija sa `DISTINCT`

Možete koristiti **`DISTINCT`** da biste uklonili duplikate i prikazali samo jedinstvene vrednosti u rezultatima upita.

Primer:

Ako želite da prikažete jedinstvene email adrese korisnika:

```
SELECT DISTINCT email
FROM korisnici;
```

Ovaj upit prikazuje jedinstvene email adrese iz tabele `korisnici`, eliminišući sve duplikate.

7. Projekcija sa `JOIN` operacijama

Projekcija se takođe može koristiti u kombinaciji sa **`JOIN`** operacijama, kako bi se iz više tabela selektovale relevantne kolone. U ovom slučaju, možete selektovati kolone iz više tabela koje su povezane određenim uslovima.

Primer:

Ako želite da prikažete imena korisnika zajedno sa nazivima proizvoda koje su naručili, koristite **`JOIN`**:

```
SELECT k.ime, k.prezime, p.naziv_proizvoda
FROM korisnici k
JOIN narudzbe n ON k.id = n.korisnik_id
JOIN proizvodi p ON n.proizvod_id = p.id;
```

Ovaj upit prikazuje ime i prezime korisnika zajedno sa nazivima proizvoda koje su naručili, koristeći `JOIN` kako bi povezao tabele `korisnici`, `narudzbe` i `proizvodi`.

8. Projekcija sa `ORDER BY`

Projekcija se može koristiti zajedno sa **`ORDER BY`** klauzulom za sortiranje rezultata na temelju određenih kolona.

Primer:

Ako želite da prikažete imena i prezimena korisnika i sortirate ih po datumu registracije:

```
SELECT ime, prezime, datum_registracije
FROM korisnici
ORDER BY datum_registracije DESC;
```

Ovaj upit prikazuje ime, prezime i datum registracije korisnika, sortirano od najnovijih prema najstarijim korisnicima.

Zaključak

Projekcija u MySQL-u omogućava selektovanje specifičnih kolona iz tabele, kao i izračunavanje novih vrednosti na temelju postojećih podataka. Uz pomoć projekcije možete:

- Selektovati samo potrebne kolone,
- Izračunavati nove vrednosti koristeći SQL funkcije,
- Koristiti agregatne funkcije za sumiranje ili računanje proseka,
- Filtrirati podatke pomoću `WHERE` klauzule,
- Sortirati podatke sa `ORDER BY`.

Korišćenje projekcije omogućava efikasnu i optimizovanu manipulaciju podacima, čineći vaše upite jasnim i preciznim.

Agregatne funkcije

Agregatne funkcije u MySQL-u omogućavaju obavljanje različitih matematičkih i statističkih operacija na skupovima podataka u tabelama. Ove funkcije koriste se sa **`GROUP BY`** klauzulom kako bi se podaci grupirali i zatim analizirali prema određenim kriterijima.

Evo pregleda najčešće korištenih agregatnih funkcija u MySQL-u:

1. `COUNT()`

Funkcija `COUNT()` vraća broj redova u tabeli ili broja ne-NULL vrijednosti u određenoj koloni.

Sintaksa:

```
SELECT COUNT(column_name) FROM table_name;
```

Primjer:

Ako želimo prebrojati koliko narudžbi ima korisnik sa ID-em 1:

```
SELECT COUNT(*) FROM narudzbe WHERE korisnik_id = 1;
```

Ova komanda će vratiti ukupan broj narudžbi korisnika sa ID-em 1.

2. `SUM()`

Funkcija `SUM()` računa zbir svih vrijednosti u određenoj koloni (brojčane vrednosti).

Sintaksa:

```
SELECT SUM(column_name) FROM table_name;
```

Primjer:

Ako želimo izračunati ukupnu vrijednost svih narudžbi za određenog korisnika:

```
SELECT SUM(cijena) FROM narudzbe WHERE korisnik_id = 1;
```

Ova komanda će prikazati ukupan iznos svih narudžbi korisnika sa ID-em 1.

3. `AVG()`

Funkcija `AVG()` vraća prosječnu vrijednost svih vrednosti u određenoj koloni (brojčane vrednosti).

Sintaksa:

```
SELECT AVG(column_name) FROM table_name;
```

Primjer:

Ako želimo izračunati prosječnu cijenu svih proizvoda u tabeli `proizvodi`:

```
SELECT AVG(cijena) FROM proizvodi;
```

Ova komanda će prikazati prosječnu cijenu svih proizvoda.

4. `MIN()`

Funkcija `MIN()` vraća najmanju vrijednost u određenoj koloni.

Sintaksa:

```
SELECT MIN(column_name) FROM table_name;
```

Primjer:

Ako želimo pronaći najmanju cijenu proizvoda u tabeli `proizvodi`:

```
SELECT MIN(cijena) FROM proizvodi;
```

Ova komanda će vratiti najmanju cijenu proizvoda.

5. `MAX()`

Funkcija `MAX()` vraća najveću vrijednost u određenoj koloni.

Sintaksa:

```
SELECT MAX(column_name) FROM table_name;
```

Primjer:

Ako želimo pronaći najveću cenu proizvoda u tabeli `proizvodi`:

```
SELECT MAX(cijena) FROM proizvodi;
```

Ova komanda će vratiti najveću cenu proizvoda.

6. `GROUP_CONCAT()`

Funkcija `GROUP_CONCAT()` omogućava spajanje vrijednosti iz više redova u jedan red, koristeći separator.

Sintaksa:

```
SELECT GROUP_CONCAT(column_name SEPARATOR 'separator') FROM
table_name;
```

Primjer:

Ako želimo dobiti sve proizvode koje je naručio korisnik sa ID-em 1, spajajući ih u jedan niz:

```
SELECT GROUP_CONCAT(proizvod SEPARATOR ', ') FROM narudzbe WHERE
korisnik_id = 1;
```

Ova komanda će prikazati sve proizvode koje je korisnik sa ID-em 1 naručio, razdvojene zarezom.

7. Kombinacija agregatnih funkcija sa `GROUP BY`

Kombinovanjem agregatnih funkcija sa `GROUP BY` klauzulom možete grupisati podatke po određenim kolonama i izračunati agregatne vrijednosti za svaku grupu.

Sintaksa:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name) FROM table_name
GROUP BY column_name;
```

Primjer:

Ako želimo izračunati ukupnu vrijednost narudžbi (`SUM`) i broj narudžbi (`COUNT`) za svakog korisnika:

```
SELECT korisnik_id, COUNT(*), SUM(cijena)
FROM narudzbe
GROUP BY korisnik_id;
```

Ova komanda će prikazati broj narudžbi i ukupan iznos narudžbi za svakog korisnika.

8. Filtriranje sa `HAVING`

Ako želite filtrirati rezultate koji su već grupisani (nakon primjene `GROUP BY`), možete koristiti `HAVING` klauzulu. Razlika između `WHERE` i `HAVING` je ta što se `WHERE` koristi za filtriranje redova prije nego što su podaci grupisani, dok se `HAVING` koristi za filtriranje nakon grupisanja.

Sintaksa:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name
HAVING AGGREGATE_FUNCTION(column_name) condition;
```

Primjer:

Ako želimo izračunati broj narudžbi (`COUNT`) i ukupan iznos narudžbi (`SUM`) za svakog korisnika, ali samo za korisnike koji imaju više od 3 narudžbe:

```
SELECT korisnik_id, COUNT(*), SUM(cijena)
FROM narudzbe
GROUP BY korisnik_id
HAVING COUNT(*) > 3;
```

Ova komanda će prikazati podatke samo za korisnike koji imaju više od 3 narudžbe.

9. Primjer korištenja više agregatnih funkcija zajedno

Možete koristiti više agregatnih funkcija u istom upitu:

```
SELECT korisnik_id, COUNT(*), SUM(cijena), AVG(cijena), MIN(cijena),
MAX(cijena)
FROM narudzbe
GROUP BY korisnik_id;
```

Ova komanda će prikazati broj narudžbi, ukupan iznos, prosječnu cenu, najmanju i najveću cijenu za svakog korisnika.

Zaključak

Agregatne funkcije su izuzetno moćan alat u MySQL-u jer omogućavaju analizu i izračunavanje statističkih podataka na temelju sadržaja u tabelama. Kombinovanjem agregatnih funkcija sa `GROUP BY` i `HAVING` klauzulama možete vrlo efikasno grupisati i analizirati podatke prema specifičnim uslovima.

Ako imate bilo kakva pitanja o specifičnim funkcijama ili primjerima, slobodno pitajte!

Spajanje relacija / tabela: join komande

U MySQL-u, **JOIN** operacija omogućava kombinovanje redova iz dve ili više tabela na temelju povezanih kolona. **JOIN** operacije su ključne za rad sa relacijama između tabela, jer omogućavaju pristup podacima koji su raspodijeljeni u različitim tabelama, ali su logički povezani.

Tipovi JOIN operacija u MySQL-u:

1. **INNER JOIN**
2. **LEFT JOIN (ili LEFT OUTER JOIN)**
3. **RIGHT JOIN (ili RIGHT OUTER JOIN)**
4. **FULL JOIN (ili FULL OUTER JOIN)**
5. **CROSS JOIN**
6. **SELF JOIN**

Svaka od ovih operacija ima različitu svrhu i način kako kombinuje podatke iz povezanih tabela.

1. INNER JOIN

`INNER JOIN` vraća samo one redove koji imaju podudaranje u obe povezane tabele. Ako neki red u jednoj tabeli ne odgovara nijednom redu u drugoj tabeli, on neće biti uključen u rezultat.

Sintaksa:

```
SELECT kolona1, kolona2, ...
FROM tabela1
INNER JOIN tabela2 ON tabela1.kolona = tabela2.kolona;
```

Primer:

Ako imamo tabele `korisnici` i `narudzbe`, želimo da dobijemo sve korisnike koji imaju narudžbe:

```
SELECT k.ime, k.prezime, n.narudzba_id
FROM korisnici k
INNER JOIN narudzbe n ON k.id = n.korisnik_id;
```

Ovaj upit vraća samo one korisnike koji imaju barem jednu narudžbu.

2. LEFT JOIN (ili LEFT OUTER JOIN)

`LEFT JOIN` vraća sve redove iz leve tabele (tabele koja je levo od `JOIN` klauzule) i odgovarajuće redove iz desne tabele. Ako nema podudaranja u desnoj tabeli, rezultati sa leve tabele se i dalje vraćaju, ali sa vrijednostima `NULL` za kolone desne tabele.

Sintaksa:

```
SELECT kolona1, kolona2, ...
FROM tabela1
LEFT JOIN tabela2 ON tabela1.kolona = tabela2.kolona;
```

Primer:

Ako želimo da prikažemo sve korisnike, zajedno sa informacijama o njihovim narudžbama, čak i ako neki korisnici nemaju narudžbe:

```
SELECT k.ime, k.prezime, n.narudzba_id
FROM korisnici k
LEFT JOIN narudzbe n ON k.id = n.korisnik_id;
```

Ovaj upit će prikazati sve korisnike, a za korisnike koji nemaju narudžbe, vrednost `narudzba_id` će biti `NULL`.

3. RIGHT JOIN (ili RIGHT OUTER JOIN)

`RIGHT JOIN` je suprotno od `LEFT JOIN`. Vraća sve redove iz desne tabele i odgovarajuće redove iz leve tabele. Ako nema podudaranja u levoj tabeli, rezultati iz desne tabele se i dalje vraćaju, ali sa vrijednostima `NULL` za kolone leve tabele.

Sintaksa:

```
SELECT kolona1, kolona2, ...
FROM tabela1
RIGHT JOIN tabela2 ON tabela1.kolona = tabela2.kolona;
```

Primer:

Ako želimo da prikažemo sve narudžbe, zajedno sa informacijama o korisnicima, čak i ako neki narudžbi nisu dodijeljeni korisnici:

```
SELECT k.ime, k.prezime, n.narudzba_id
FROM korisnici k
```

```
RIGHT JOIN narudzbe n ON k.id = n.korisnik_id;
```

Ovaj upit će prikazati sve narudžbe, a za narudžbe koje nemaju dodijeljenog korisnika, vrednosti `ime` i `prezime` će biti `NULL`.

4. FULL JOIN (ili FULL OUTER JOIN)

`FULL JOIN` vraća sve redove iz obe tabele, bez obzira da li postoji podudaranje. Ako nema podudaranja u jednoj od tabela, vrijednosti će biti `NULL` za te kolone. **MySQL** ne podržava direktno `FULL JOIN`, ali možete postići isti rezultat koristeći kombinaciju `LEFT JOIN` i `RIGHT JOIN` sa `UNION`.

Sintaksa:

MySQL ne podržava direktno `FULL JOIN`, ali možete koristiti `UNION` da postignete sličan efekat:

```
SELECT kolona1, kolona2, ...
FROM tabela1
LEFT JOIN tabela2 ON tabela1.kolona = tabela2.kolona
UNION
SELECT kolona1, kolona2, ...
FROM tabela1
RIGHT JOIN tabela2 ON tabela1.kolona = tabela2.kolona;
```

Primer:

Ako želimo da dobijemo sve korisnike i sve narudžbe, uključujući korisnike koji nemaju narudžbe i narudžbe koje nisu dodijeljene korisnicima:

```
SELECT k.ime, k.prezime, n.narudzba_id
FROM korisnici k
LEFT JOIN narudzbe n ON k.id = n.korisnik_id
UNION
SELECT k.ime, k.prezime, n.narudzba_id
FROM korisnici k
RIGHT JOIN narudzbe n ON k.id = n.korisnik_id;
```

5. CROSS JOIN

`CROSS JOIN` vraća kartični proizvod dviju tabela, tj. kombinaciju svih mogućih parova redova između dve tabele. Ovaj tip JOIN-a ne koristi nikakav uslov za spajanje tabela, već samo kombinovanje svih redova.

Sintaksa:

```
SELECT kolona1, kolona2, ...
FROM tabela1
CROSS JOIN tabela2;
```

Primer:

Ako imamo tabele `proizvodi` i `boje`, i želimo da prikažemo sve moguće kombinacije proizvoda i boja:

```
SELECT p.proizvod, b.boja
FROM proizvodi p
CROSS JOIN boje b;
```

Ovaj upit će prikazati sve moguće kombinacije proizvoda i boja.

6. SELF JOIN

`SELF JOIN` je JOIN koji se koristi kada spajate tabelu sa samom sobom. Obično se koristi kada želite da uporedite podatke u istoj tabeli.

Sintaksa:

```
SELECT t1.kolona1, t2.kolona2
FROM tabela t1
JOIN tabela t2 ON t1.kolona = t2.kolona;
```

Primer:

Ako imamo tabelu `zaposleni`, gde svaki zaposleni ima `menadzer_id` koji se odnosi na ID njegovog menadžera, možemo koristiti `SELF JOIN` da prikažemo imena zaposlenih zajedno sa imenima njihovih menadžera:

```
SELECT e1.ime AS zaposleni_ime, e2.ime AS menadzer_ime
FROM zaposleni e1
JOIN zaposleni e2 ON e1.menadzer_id = e2.id;
```

Ovaj upit prikazuje ime zaposlenog zajedno sa imenom njegovog menadžera.

Zaključak

- **`INNER JOIN`** vraća samo one redove koji imaju podudaranje u obe tabele.
- **`LEFT JOIN`** vraća sve redove iz leve tabele, uključujući one koji nemaju podudaranje u desnoj tabeli.
- **`RIGHT JOIN`** vraća sve redove iz desne tabele, uključujući one koji nemaju podudaranje u levoj tabeli.
- **`FULL JOIN`** (koji nije direktno podržan u MySQL-u) vraća sve redove iz obe tabele, koristeći kombinaciju `LEFT JOIN` i `RIGHT JOIN`.
- **`CROSS JOIN`** vraća kartični proizvod dviju tabela (kombinacija svih mogućih parova redova).
- **`SELF JOIN`** spaja tabelu sa samom sobom, što je korisno za upoređivanje podataka unutar iste tabele.

Razumevanje različitih tipova JOIN operacija vam omogućava da radite sa više tabela i kombinujete podatke na način koji najbolje odgovara vašim potrebama u analizi podataka.

Podupit

Podupit (subquery) u MySQL-u je SQL upit koji se koristi unutar drugog upita. Podupiti omogućavaju da izvršite jedan upit unutar drugog upita, čime možete efikasno filtrirati, ažurirati ili manipulirati podacima. Postoje različite vrste podupita, uključujući **skalarne podupite**, **podupite u `WHERE` klauzuli**, **podupite u `FROM` klauzuli** i **podupite u `SELECT` klauzuli**.

Evo nekoliko ključnih informacija o podupitima i njihovim primjenama:

1. Vrste podupita

1. **Skalarni podupit (Scalar Subquery)**:
   - Vraća samo jednu vrednost (jedan red i jednu kolonu).
   - Može se koristiti u `SELECT`, `WHERE`, ili `HAVING` klauzulama.
2. **Podupit u `WHERE` klauzuli (WHERE Subquery)**:
   - Najčešće korišćeni tip podupita.
   - Može vraćati više vrijednosti, a koristi se za filtriranje podataka u glavnom upitu.
3. **Podupit u `FROM` klauzuli (FROM Subquery)**:
   - Takođe poznat kao **inline view** ili **derived table**.
   - Vraća privremenu tabelu koju možete koristiti kao osnovu za izvršavanje drugih operacija.
4. **Podupit u `SELECT` klauzuli (SELECT Subquery)**:
   - Može se koristiti za selektovanje vrijednosti koje su izračunate na temelju drugih podataka.

2. Podupit u `WHERE` klauzuli

Podupit unutar `WHERE` klauzule se koristi za filtriranje rezultata na temelju vrijednosti koju vraća podupit. Obično se koristi sa operaterima poput `IN`, `EXISTS`, `ANY`, `ALL`, `NOT IN`, itd.

Primer 1: Podupit sa `IN`

Na primer, ako želimo dobiti sve narudžbe koje je napravio korisnik sa specifičnom email adresom:

```
SELECT *
FROM narudzbe
WHERE korisnik_id IN (SELECT id FROM korisnici WHERE email =
'primer@domain.com');
```

Ovdje, podupit `(SELECT id FROM korisnici WHERE email = 'primer@domain.com')` vraća ID korisnika sa tom email adresom, a glavni upit koristi taj ID za selektovanje narudžbi tog korisnika.

Primer 2: Podupit sa `EXISTS`

Podupit sa `EXISTS` provjerava da li postoji barem jedan red koji zadovoljava uslove unutar podupita.

```
SELECT *
FROM narudzbe n
WHERE EXISTS (SELECT 1 FROM korisnici k WHERE k.id = n.korisnik_id
AND k.status = 'aktivno');
```

Ovaj upit vraća sve narudžbe za koje postoji odgovarajući korisnik sa statusom "aktivno".

3. Podupit u `FROM` klauzuli

Podupit u `FROM` klauzuli kreira privremenu tabelu koja se koristi kao osnov za daljnje upite. Ova metoda je korisna kada želite da izvršite kompleksne operacije nad podacima koji se nalaze u različitim tabelama.

Primer:

Ako želite da izračunate ukupan broj narudžbi po korisniku i zatim prikažete sve korisnike sa brojem narudžbi većim od 5:

```
SELECT korisnik_id, ukupno_narudzbi
FROM (
    SELECT korisnik_id, COUNT(*) AS ukupno_narudzbi
    FROM narudzbe
    GROUP BY korisnik_id
) AS podupit
WHERE ukupno_narudzbi > 5;
```

Ovdje podupit u `FROM` klauzuli računa broj narudžbi po korisniku, a glavni upit filtrira korisnike sa više od 5 narudžbi.

4. Podupit u `SELECT` klauzuli

Podupit može biti korišćen i unutar `SELECT` klauzule kako bi se dobile vrijednosti iz drugih tabela.

Primer:

Ako želite dobiti sve proizvode zajedno sa ukupnim brojem narudžbi za svaki proizvod:

```
SELECT proizvod,
       (SELECT COUNT(*) FROM narudzbe WHERE proizvod = p.proizvod)
AS broj_narudzbi
FROM proizvodi p;
```

U ovom primjeru, podupit unutar `SELECT` klauzule računa broj narudžbi za svaki proizvod.

5. Skalarni podupit (Scalar Subquery)

Skalarni podupit je podupit koji vraća samo jednu vrednost. Može se koristiti u `SELECT`, `WHERE`, `HAVING` klauzulama kada se očekuje samo jedna vrednost.

Primer:

Ako želite da prikažete sve korisnike zajedno sa njihovom najnovijom narudžbom:

```
SELECT ime, prezime,
       (SELECT proizvod FROM narudzbe WHERE korisnik_id = k.id ORDER
BY datum_narudzbe DESC LIMIT 1) AS poslednja_narudzba
FROM korisnici k;
```

Ovdje, podupit vraća posljednju narudžbu za svakog korisnika.

6. Koristi sa `ANY` i `ALL`

Podupiti se mogu koristiti sa `ANY` i `ALL` operaterima kako bi upit bio fleksibilniji.

Primer sa `ANY`:

```
SELECT *
FROM narudzbe
WHERE cijena > ANY (SELECT cijena FROM proizvodi WHERE kategorija =
'Tehnička oprema');
```

Ovaj upit vraća sve narudžbe čija je cena veća od bilo koje cene proizvoda u kategoriji "Tehnička oprema".

Primer sa `ALL`:

```
SELECT *
FROM narudzbe
WHERE cijena > ALL (SELECT cijena FROM proizvodi WHERE kategorija =
'Tehnička oprema');
```

Ovaj upit vraća sve narudžbe čija je cena veća od svih cena proizvoda u kategoriji "Tehnička oprema".

7. Podupit sa `NOT IN`

Ako želite isključiti određene vrednosti iz glavnog upita, možete koristiti `NOT IN` sa podupitima.

Primer:

```
SELECT *
FROM narudzbe
WHERE korisnik_id NOT IN (SELECT id FROM korisnici WHERE status =
'neaktivan');
```

Ovaj upit će prikazati sve narudžbe korisnika koji nemaju status "neaktivan".

8. Zašto koristiti podupite?

Podupiti su korisni jer omogućavaju složenije upite u jednom SQL izrazu bez potrebe za korišćenjem više upita ili spajanja tabela (JOIN). Korišćenje podupita čini kod kraćim i lakšim za razumevanje, ali može imati lošije performanse kod velikih baza podataka, pa je potrebno pažljivo planirati kada i kako ih koristiti.

Zaključak

Podupiti (subqueries) su moćan alat u MySQL-u za rješavanje kompleksnih problema filtriranja, selektovanja i manipulisanja podacima u jednoj ili više tabela. Postoji više tipova podupita, kao što su podupiti u `WHERE`, `FROM` i `SELECT` klauzulama, a svaka od njih može biti korišćena za rješavanje specifičnih zadataka. Pravilno korišćenje podupita omogućava da vaši SQL upiti budu fleksibilniji i efikasniji.

Integriteti, strani kljucevi i indexi

Strani ključevi (Foreign Keys) su ključni koncept u relacijskim bazama podataka, jer omogućavaju uspostavljanje veza između tabela i očuvanje referencijalne integriteta podataka. Ovaj mehanizam pomaže u sprečavanju gubitka podataka i omogućava da podaci u različitim tabelama budu konzistentni i povezani.

1. Šta su strani ključevi?

Strani ključ je kolona (ili grupa kolona) u jednoj tabeli koja se povezuje sa primarnim ključem u drugoj tabeli. Ovaj odnos omogućava povezivanje podataka između različitih tabela, čime se stvara referencijalna integriteta.

Na primjer, ako imate tabelu `korisnici` i tabelu `narudžbe`, možda biste želeli da veza između narudžbi i korisnika bude održavana pomoću stranog ključa, koji osigurava da svaka narudžba bude povezana sa postojećim korisnikom.

2. Zašto koristiti strane ključeve?

Strani ključevi pomažu u:

- **Očuvanju referencijalne integriteta:** Osiguravaju da podaci u povezanim tabelama ostanu dosljedni.
- **Sprečavanju gubitka podataka:** Onemogućuju brisanje ili ažuriranje podataka na način koji bi doveo do nevalidnih odnosa između tabela.

3. Kreiranje stranog ključa

Strani ključ se kreira u tabeli koja sadrži referencu na drugi entitet (drugog korisnika, proizvod, itd.). Obično se strani ključ postavlja na kolonu koja je u istoj tabeli kao primarni ključ u drugoj tabeli.

Evo kako možete kreirati strani ključ prilikom kreiranja tabele:

3.1 Kreiranje stranog ključa u SQL-u

Recimo da imate tabelu `korisnici` i tabelu `narudzbe`, gdje svaka narudžba pripada korisniku. Tabela `narudzbe` može imati strani ključ koji se odnosi na `id` iz tabele `korisnici`.

```
CREATE TABLE korisnici (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
    ime VARCHAR(100),
    prezime VARCHAR(100)
);

CREATE TABLE narudzbe (
    id INT AUTO_INCREMENT PRIMARY KEY,
    korisnik_id INT,
    proizvod VARCHAR(100),
    datum_narudzbe DATE,
    FOREIGN KEY (korisnik_id) REFERENCES korisnici(id)
);
```

U ovom primjeru:

- `korisnik_id` je strani ključ u tabeli `narudzbe`, koji se povezuje s `id` iz tabele `korisnici`.
- Ovaj strani ključ osigurava da svaka narudžba bude povezana s postojećim korisnikom.

3.2 Postavljanje stranog ključa na već postojeću tabelu

Ako već imate tabelu i želite dodati strani ključ, koristite `ALTER TABLE`:

sql
Copy code

```
ALTER TABLE narudzbe
ADD CONSTRAINT fk_korisnik_id FOREIGN KEY (korisnik_id) REFERENCES
korisnici(id);
```

4. Referencijalna integriteta (ON DELETE i ON UPDATE)

Kada postavite strani ključ, možete specifično odrediti kako se ponašati prilikom brisanja ili ažuriranja podataka u referenciranoj tabeli (npr. tabela `korisnici`). Postoje četiri osnovna ponašanja koja možete definirati pomoću `ON DELETE` i `ON UPDATE` opcija:

4.1 ON DELETE

- **CASCADE**: Ako se red u roditeljskoj tabeli (npr. `korisnici`) obriše, svi povezani redovi u tabeli sa stranim ključem (npr. `narudzbe`) takođe će biti obrisani.
- **SET NULL**: Ako se red u roditeljskoj tabeli obriše, strani ključ u povezanoj tabeli biće postavljen na `NULL`.

- **RESTRICT**: Ako postoji bilo kakav povezani red u tabeli sa stranim ključem, nećete moći obrisati red u roditeljskoj tabeli. (Podrazumijevano)
- **NO ACTION**: Slično kao `RESTRICT`, ali može biti implementirano na različite načine u zavisnosti od RDBMS-a.

4.2 ON UPDATE

- **CASCADE**: Ako se vrijednost primarnog ključa promijeni, vrednosti stranog ključa u povezanim tabelama će automatski biti ažurirane.
- **SET NULL**: Ako se vrednost primarnog ključa promijeni, strani ključ u povezanim tabelama postat će `NULL`.
- **RESTRICT**: Neće biti moguće promijeniti vrednost primarnog ključa ako postoji bilo kakva veza u drugim tabelama.
- **NO ACTION**: Isto kao `RESTRICT`, ali sa malo drugačijim ponašanjem.

5. Primjeri sa ON DELETE i ON UPDATE

Evo nekoliko primjera kako postaviti opcije za **ON DELETE** i **ON UPDATE**:

5.1 CASCADE na DELETE i UPDATE

Ako želite da se svi povezani podaci obrišu ili ažuriraju kada se promeni roditeljski podatak, koristite `CASCADE`:

```
CREATE TABLE narudzbe (
    id INT AUTO_INCREMENT PRIMARY KEY,
    korisnik_id INT,
    proizvod VARCHAR(100),
    datum_narudzbe DATE,
    FOREIGN KEY (korisnik_id) REFERENCES korisnici(id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

5.2 SET NULL na DELETE

Ako želite da se strani ključ postavi na `NULL` umesto brisanja povezanih podataka:

```
CREATE TABLE narudzbe (
    id INT AUTO_INCREMENT PRIMARY KEY,
    korisnik_id INT,
    proizvod VARCHAR(100),
    datum_narudzbe DATE,
```

```
    FOREIGN KEY (korisnik_id) REFERENCES korisnici(id)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);
```

5.3 RESTRICT na DELETE

Ako želite da spriječite brisanje korisnika koji ima povezane narudžbe, koristite `RESTRICT`:

```
CREATE TABLE narudzbe (
    id INT AUTO_INCREMENT PRIMARY KEY,
    korisnik_id INT,
    proizvod VARCHAR(100),
    datum_narudzbe DATE,
    FOREIGN KEY (korisnik_id) REFERENCES korisnici(id)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
);
```

6. Prednosti korištenja stranih ključeva

- **Sprečavanje gubitka podataka:** Ako neko pokuša da obriše korisnika koji ima povezane narudžbe, MySQL može onemogućiti to brisanje (koristeći `RESTRICT`), čime se sprečava gubitak podataka.
- **Povezivanje tabela:** Omogućavaju povezivanje podataka u različitim tabelama na logičan način, čineći bazu podataka organizovanijom.
- **Osiguranje dosljednosti:** Pomažu u održavanju integriteta podataka u cijeloj bazi podataka, jer sprječavaju stvaranje "orphan" podataka (podaci u jednoj tabeli koji nemaju odgovarajući red u drugoj tabeli).

7. Zaključak

Strani ključevi su ključni za održavanje referencijalne integriteta između povezanih tabela. Koristeći opcije poput `ON DELETE CASCADE` ili `ON UPDATE CASCADE`, možete efikasno upravljati podacima i spriječiti gubitak informacija. Uvijek pažljivo birajte opcije za brisanje i ažuriranje, jer one mogu imati veliki uticaj na podatke u povezanim tabelama.

Indeksi u MySQL-u su strukture podataka koje se koriste za brzo pronalaženje i pristup podacima u tabeli. Oni značajno poboljšavaju performanse upita, naročito onih koji uključuju operacije pretrage (`SELECT`), filtriranja (`WHERE`), spajanja (`JOIN`) ili sortiranja (`ORDER BY`).

Ključne informacije o indeksima:

1. **Vrste indeksa:**
    - **PRIMARY KEY**: Jedinstveni indeks koji identifikuje svaki red u tabeli. Automatski kreira indeks.
    - **UNIQUE**: Sprečava duplikate vrednosti u indeksiranim kolonama.
    - **INDEX (normalni indeks)**: Standardni indeks koji se koristi za ubrzavanje pretraga.
    - **FULLTEXT**: Koristi se za pretraživanje tekstualnih podataka u velikim tekstualnim kolonama (`CHAR`, `VARCHAR`, `TEXT`).
    - **SPATIAL**: Specijalizovan indeks za prostorne (geografske) podatke u `GEOMETRY` kolonama.
2. **Sintaksa za kreiranje indeksa:**

Dodavanje indeksa prilikom kreiranja tabele:

```
CREATE TABLE ime_tabele (
    kolona1 tip_podatka,
    kolona2 tip_podatka,
    kolona3 tip_podatka,
    INDEX (kolona2)
);
```

    -

Dodavanje indeksa na postojećoj tabeli:

```
CREATE INDEX ime_indeksa ON ime_tabele (kolona);
```

    -

Dodavanje jedinstvenog indeksa:

```
CREATE UNIQUE INDEX ime_indeksa ON ime_tabele (kolona);
```

    -

Brisanje indeksa:

```
DROP INDEX ime_indeksa ON ime_tabele;
```

3.

Prikaz postojećih indeksa:

```
SHOW INDEX FROM ime_tabele;
```

4.

5. **Kako indeksi rade:** Indeksi se obično implementiraju kao B-tree strukture. Kada izvodite pretragu, MySQL koristi indeks da brzo pronađe podatke, umesto da prolazi kroz cijelu tabelu red po red.
6. **Kada koristiti indekse:**
    - Na kolonama koje se često koriste u `WHERE` uslovima.
    - Na kolonama koje se često koriste za sortiranje (`ORDER BY`).
    - Na kolonama koje se koriste u spajanjima (`JOIN`).
    - Na kolonama koje imaju puno jedinstvenih vrednosti (npr. `ID`).
7. **Potencijalni problemi sa indeksima:**
    - **Dodatan prostor:** Indeksi zauzimaju prostor na disku.
    - **Sporije upisivanje:** `INSERT`, `UPDATE`, i `DELETE` operacije mogu biti sporije jer je potrebno ažurirati indekse.
    - **Neefikasnost kod malih tabela:** Na malim tabelama, prednosti indeksa mogu biti zanemarive.

Primjeri:

Kreiranje PRIMARY KEY i indeksa:

```
CREATE TABLE studenti (
    id INT NOT NULL,
    ime VARCHAR(100),
    prezime VARCHAR(100),
    PRIMARY KEY (id),
    INDEX (prezime)
);
```

1.

Dodavanje indeksa na postojeću tabelu:
sql
Copy code

```
CREATE INDEX idx_ime ON studenti (ime);
```

2.

Kombinovani (više-kolona) indeks:

```
CREATE INDEX idx_ime_prezime ON studenti (ime, prezime);
```

3.

Indeksi su kritični za performanse, ali pretjerano korišćenje indeksa može biti kontraproduktivno.

Kraj

The End

ovo je kraj dokumenta očito

ali ukoliko si dragi moj citaoce došao do samog kraja, naravno uz predpostavku da si sve pročitao i isprobao komande trebao bi biti osposobljen za samostalan rad u mySQL-u.

I naravno trebao bi imati osnovni set znanja za pisanje koda

i pravljenje dobre baze

slika je tu nešto kao moj potpis i što već znam