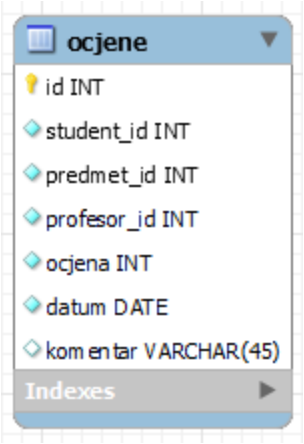
2-3 Agregacijske funkcije



Agregacijske funkcije

Ukoliko postoji potreba primjenjivati dodatne operacije nad skupom podataka, tj. kolonama, to se može učiniti koristeći tzv. agregacijske funkcije. Ove funkcije pružaju mogućnost dodatne manipulacije podacima prije njihovog ispisa i primjenjuju se nad grupisanim skupom izlazne tabele. Već je predstavljena jedna takva funkcija, `COUNT`, koja prebrojava redove. Recimo:

```
SELECT COUNT(*) FROM ocjene;
```

Ovo znači, prebroj sve redove iz tabele *ocjene*. To je kao da smo rekli `SELECT * FROM ocjene;` i ručno prebrojali redove izlazne tabele. Zvijezdicom se simbolizira prebrojavanje po bilo kojoj koloni, ali se može ograničiti na jednu kolonu:

```
SELECT COUNT(komentar) FROM ocjene;
```

U ovom slučaju, sistem će vratiti broj redova gdje je polje komentar popunjeno. Tj. moguće je dobiti drugačiji odgovor od `COUNT(*)` jer je polje *komentar* opcionalno i stoga može biti prazno.

Neke od agregacijskih funkcija su: `SUM, MIN, MAX i AVG`. Npr.:

```
SELECT MIN(ocjena) FROM ocjene;
SELECT MAX(ocjena) FROM ocjene;
```

`MIN` i `MAX` se mogu primjenjivati i na nenumeričke kolone. Tako npr. ako je potrebno vratiti prvu ocjenu u tabeli po datumu, možemo primjeniti `MIN` operaciju nad kolonom *datum*:

```
SELECT MIN(datum) AS datum_najstarije_ocjene FROM ocjene;
```

Ukoliko želimo koristiti više agregacijskih funkcija istovremeno, to i možemo uraditi. Npr. selektovanje najniže i najviše ocjene se može uraditi na sljedeći način:

```
SELECT MAX(ocjena), MIN(ocjena) FROM ocjene;
```

Međutim, problem sa svim ovim funkcijama je što vraćaju jednostavne informacije poput najstarijeg datuma kada je data ocjena ili najniže ili najviše ocjene međutim, ne vraćaju za kojeg je studenta, ili na kojem predmetu ova ocjena data. Stoga je primamljivo napisati dodatnu kolonu, međutim ovakav upit će dati netačne rezultate:

```
SELECT student_id, MAX(ocjena), MIN(ocjena) FROM ocjene;
```

Ovaj upit nema smisla jer najviša ocjena može biti od jednog a najmanja od drugog studenta. Prethodni upit je imao smisla jer se i najviša i najmanja biraju iz cijele tabele. I u ovom slučaju će to biti slučaj, međutim vrijednost *student_id* je nepouzdana i neće biti tačna.

Ono što zapravo želimo da uradimo, je da podijelimo tabelu na podtabele za svakog studenta. Drugim riječima, primjenit ćemo `MIN` i `MAX` na sve redove po studentu. Ovo se realizuje grupisanjem podataka koristeći `GROUP BY` klauzulu. Ova klauzula se uvijek kombinuje sa agregacijskim funkcijama.

```
SELECT student_id, MAX(ocjena), MIN(ocjena)
FROM ocjene
GROUP BY student_id;
```

Ovo u prevodu znači, dobavi sve najmanje i najviše ocjene po studentu. To je isto kao da postoji N tabela, po jedna za svakog studenta, i da se za svaku od njih odjednom pretražuje `MIN` i `MAX`.

Još jedan primjer grupisanja je prebrojavanje ocjena po svakom studentu:

```
SELECT student_id, COUNT(*) brocjena FROM ocjene GROUP BY student_id;
```

Ali šta ako želimo prebrojati broj ocjena po studentu po predmetu. Zapravo, `GROUP BY` omogućava grupisanje po više kolona.

```
SELECT student_id, predmet_id, COUNT(*) brocjena FROM ocjene
GROUP BY student_id, predmet_id;
```

Još jedna agregacijska funkcija je `SUM`. Ova funkcija primjenjuje zbir na kolone numeričkog tipa. `NULL` i nenumeričke vrijednosti su ignorisane.

```
SELECT SUM(ocjena) FROM ocjene;
```

Unutar `SELECT` klauzule, možemo koristiti i aritmetičke operacije, poput dijeljenja. Da se dobije prosječna vrijednost ocjene, to može se uraditi na sljedeći način:

```
SELECT SUM(ocjena)/COUNT(ocjena) AS prosjecna_ocjena FROM ocjene;
```

Međutim da se ovo ne bi pisalo svaki put jer postoji funkcija pod nazivom `AVG`, koja vraća prosječnu vrijednost:

```
SELECT AVG(ocjena) AS prosjecna_ocjena FROM ocjene;
```

Sortiranje izlaza sa ORDER BY

Kada se izlistavaju podaci koristeći upite, korisno je te podatke poredati po rastućem ili opadajućem poretku. Da bi se ovo postiglo, koristi se klauzula `ORDER BY`. Npr.:

```
SELECT * FROM ocjene ORDER BY datum ASC;
```

će izlistati sve redove iz tabele ocjene poredane od najstarijeg do najnovijeg datuma. `ASC` je skraćeno od *ascending*. Ovo je opcionalna ključna riječ i nije je neophodno napisati za uzlazno sortiranje. Slično, ukoliko je potrebno sortirati po datumu od najnovijeg ka najstarijem, to se može uraditi ovako:

```
SELECT * FROM ocjene ORDER BY datum DESC;
```

Ova klauzula se može primjenjivati na sve tipove kolona, uključujući i agregacijske kolone. Na primjeru tabele *`studenti`* recimo da je potrebno vratiti brojeve imena sortirana od najčešćeg imena do najrijeđeg imena:

```
SELECT COUNT(*) AS broj, ime FROM studenti
GROUP BY ime ORDER BY broj DESC;
```

Ovaj upit će prebrojati sva imena iz tabele studenti, tj. pronaći broj redova za svako ime koja su ista, a nakon toga će izlaz sortirati po broju imena. Slično, ukoliko je potrebno sortirati od najrjeđeg do najčešćeg:

```
SELECT COUNT(*) AS broj, ime FROM studenti
GROUP BY ime ORDER BY broj;
```

Potrebno je još napomenuti da je moguće sortirati po više kolona i to u istim ili različitim smjerovima. Ukoliko je potrebno dobaviti sve studente, a sortirati abecedno grad iz kojeg dolaze ali suprotno abecedno po prezimenu, to bi se postiglo na sljedeći način:

```
SELECT * FROM studenti ORDER BY grad ASC, prezime DESC;
```

Na ovaj način je moguće manipulisati sortiranjem izlazne tabele, i sortiranje se odvija kao krajnji korak u nizu SQL operacija navedenih u upitu. `ORDER BY` se može kombinovati i sa `LIMIT`/`OFFSET` klauzulama. U ovom slučaju, MySQL baza će prvo obaviti sortiranje a onda ograničavanje ili paginaciju. Npr.: ukoliko želimo broj studenata po prezimenu, ali želimo 3. stranicu, ukoliko su prezimena sortirana obrnuto abecedno a stranica ima 3 elementa, onda:

```
SELECT COUNT(*), prezime FROM studenti GROUP BY prezime
ORDER BY prezime DESC LIMIT 3 OFFSET 6
```

Može se primjetiti, da prvo ide sortiranje pa onda ograničavanje.