1-1 Uvod i motivacija

Baze podataka su softverski i hardverski sistemi koji vode računa o skladištenju i pretraživanju podataka. Da bi se shvatila potreba za bazama podataka dovoljno je analizirati neke od najvećih softverskih sistema.

Svakodnevna interakcija sa računarom podrazumjeva indirektnu interakciju sa bazama podataka. Svaki softverski sistem, od jednostavnog web pretraživača, personalnog mobitela, pa do velikih bankarskih sistema, kao i svih društvenih mreža, posjeduje niz sistema baza podataka koji se koriste za skladiđtenje i organizovanju podataka.

Prve baze podata su se pojavile 1960-tih i bile su primitivnog tipa. 1970-tih se pojavljuju struktuirane, relacione baze podataka, koje pomažu u izbjegavanju dupliciranja podataka. Ovi koncepti se danas masovno koriste u SQL[1] bazama podataka. Tokom 80-tih i 90-tih su se ovi sistemi usavršavali, tako da danas postoje desetine, a možda i stotine takvih sistema[2].

Tokom 2000-tih, masivnim širenjem upotrebe interneta, potreba za skladištenjem podataka raste značajno, te se za ovu upotrebu pojavljuju poteškoće prilikom korištenja relacionih baza podataka. Za ovu novu svrhu, pojavljuju se novi tipovi baza podataka. Iako ne nov, koncept objektnih baza podataka[3] se širi. Još jedna vrsta baza podataka koja se pojavljuje je NoSQL vrsta baza podataka. Ove baze žrtvuju dupliranje podataka radi jednostavnije strukture.

Danas su sve ove vrste podataka u upotrebi i nije neobično da jedna obična, svakodnevna, interakcija sa mobitelom ili računarom, preko interneta, pronađe put do svih ovih tipova baza podataka.

2010-tih, raste upotreba objektnih i NoSQL[4] baza podataka, ali se pojavljuje i nova vrsta baza podataka. Blockchain tehnologije sa sobom donose jednu vrstu distribuirane baze podataka za specifičnu svrhu, međutim, kako istraživanje napreduje, i ova vrsta baza će pronaći put do svakodnevne upotrebe.

Za potrebu menadžmenta svih ovih sistema, potrebne su osobe koje dobro poznaju način rada ovih sistema na pozicijama administratora baza podataka.

U ovom materijalu fokus će najviše biti na relacione baze podataka, i programskim jezikom za interakciju sa istima, s obzirom da su one najrasprostranjenije i na tu vrstu se prvenstveno misli kada se govori bazama podataka. Bit će prezentirana i jedna vrsta objekne baze podataka i interakcije sa istom.

[1] https://en.wikipedia.org/wiki/SQL
[2] https://en.wikipedia.org/wiki/List_of_relational_database_management_systems
[3] https://en.wikipedia.org/wiki/Object_database
[4] https://en.wikipedia.org/wiki/NoSQL

1-2 Predavanje

Pojam baze podataka i modeli

2-1 Ponavljanje gradiva

Šta su baze podataka?

Baze podataka su softverski i hardverski sistemi koji vrše organizovano skladištenje podataka. Baze podataka se također zovu DBMS (**D**ata**b**ase **M**anagement **S**ystems).

Tipovi baza podataka?

Najčešća podjela tipova baza podataka su baze podataka za struktuirane i nestruktuirane tipove podataka. Ovo se također zove SQL i NoSQL tipovi baza podataka. Ukoliko nije napomenuto, govori se o SQL bazama podataka. Struktuirane baze podataka se također nazivaju relacijskim bazama podataka.

Kako su podaci u SQL bazama podataka organizovani?

Podaci su organizovani u imenovane baze podataka – šeme – koje su dalje organizovane u tabele (također zvani entiteti). Tabele imaju kolone koje imaju predefinisani tip podataka. Tabele međusobno mogu biti povezane relacijama. Zato se SQL baze podataka također zovu relacione baze podataka.

Koje tipove podataka možemo pohranjivati u SQL baze podataka?

Integer, Float, Decimal, Text, Blob, Date, itd…

Kako instalirati i pokrenuti SQL bazu podataka?

Baza se može instalirati preko softverskog paketa “XAMPP” ili direktno instalacijom MySQL DBMSa. Postoje i druge baze podataka koje imaju svoju instalacionu proceduru.

Kako se vrši interacija sa SQL bazama podataka?

Interakcija sa DBMSom se odvija preko komandne linije ili programskog interfejsa koristeći SQL naredbe. SQL je deklarativni domenski jezik (DDL - data description language) koji se koristi za oblikovanje baze podataka i prenos podataka.

Komande za kreiranje baze podataka i tabela

Kreiranje baze:

```
CREATE DATABASE databasename;
```

Brisanje baze:

```
DROP DATABASE databasename;
```

Prikaz postojećih baza:

```
SHOW DATABASES;
```

Simbol za bazu podataka

Selektovanje baze podataka:

```
USE databasename;
```

Kreiranje tabele:

```
CREATE TABLE Studenti (
      StudentID INT,
      Prezime VARCHAR(255),
      Ime VARCHAR(255),
      Adresa VARCHAR(255),
      Grad VARCHAR(255),
      ZadnjaIzmjena DATETIME,
      Opis TEXT
);
```

Opis tabele:

```
DESC Studenti;
```

Izmjena tabele, dodavanje kolone:

```
ALTER TABLE Studenti
ADD Email varchar(255);
```

Izmjena tabele, brisanje kolone:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Šta je modeliranje podataka?

Modeliranje podataka je organizovanje baze podataka, tabela, kolona i tipova podataka za skladištenje i pretraživanje za određenu svrhu.

Šta je indeksiranje baze podataka?

Indeksiranje je proces prebacivanja određenog tipa podatka u strukturu podataka koja se brže pretražuje. Koristi se kako bi ubrzalo vrijeme pretraživanja podataka.

Koje vrste relacija postoje u relacijskim bazama podataka?

- Jedan-na-jedan: jedan red u jednoj tabeli je povezan sa jednim i samo jednim redom u drugoj tabeli
- Jedan-na-više: jedan red u jednoj tabel je povezan sa više redova u drugoj ili drugim tabelama
- Više-na-više: kad više redova iz iste tabele može biti povezano sa više redova u drugoj tabeli. Ovo se obično realizuje koristeći međutabelu.

Koja su ograničenja u relacijskom modelu podataka?

Spašavanja podataka u relacionim bazama podataka zahtjeva dijeljenje podataka u različite entitete. Ovo doprinosi manjoj veličini prilikom pohrane ali je kompleksnost održavanja veća. Relacijske baze podataka nude bolju podršku za transakcijsko procesiranje ali slabiju za analitičko procesiranje.

Šta je primarni i strani ključ?

Primarni ključ je jedinstveni identifikator reda u tabeli. Strani ključ je vrijednost primarnog ključa u drugoj, relacijski povezanoj, tabeli.

Šta je normalizacija podataka?

Normalizacija podataka je proces organizacije baze podataka, uključujući kreiranje tabela, organizovanje kolona kao i prebacivanje podataka u drugačiju strukturu podataka.

Šta su transakcije i ACID pravila?

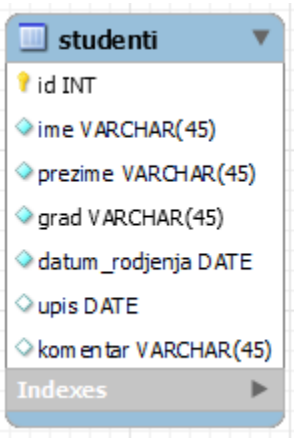
Transakcija je naziv za logičku grupu komandi koje baza podataka treba izvršiti u stilu ili sve ili ništa tj. ili će se sve komande iz transakcije izvršiti ili neće nijedna.
ACID pravila su pravila koje sistem za bazu podataka mora zadovoljiti kako bi garantovao transakcijske sposobnosti baze podataka. Tj. podaci će biti validni bez obzira na greške, nasilno gašenje itd… Ova pravila su:

- Atomičnost (Atomicity) - nedljeljivost komandi – ili / ili princip
- Konsistentnost (Consistency) - podaci mogu biti promijenjeni na jedini dozvoljeni način
- Izolacija (Isolation) - mogućnost da se transakcije odvijaju u izolovanom okruženju
- Trajnost (Durability) - sistem će preživjeti nasilne promjene poput gašenja računara

Ostale oblasti za ponoviti:

- SQL DDL iskazi `SELECT`, `INSERT`, `UPDATE` i `DELETE`
- Indeksiranje - kreiranje, brisanje
- Pogledi - kreiranje, brisanje
- Ograničenja (Constraints) - kreiranje, brisanje
- Zadatak:
    - Kreirati bazu podataka koja sadrži tri tabele
        - Studenti (ime, prezime, datum rođenja)
        - Razred (naslov, godina)
        - Upis (student id, razred id)
    - Voditi računa o ograničenjima, primarnim i stranim ključevima.
    - Napraviti upite za unos, modifikaciju i brisanje podataka u ovim tabelama.
    - Napraviti pogled za studente sa određenim imenom
    - Napraviti indeks za pretraživanje po godini razreda
    - Modifikovati tabelu studenti i dodati kolonu “Upis” – godina upisa u školu

2-2 SQL upiti



SQL upiti INSERT, DELETE, UPDATE

Nakon što se tabela kreira, moguće je unijeti podatke u tabelu. Recimo da imamo tabelu studenti i da želimo unijeti studente u tabelu. To se može uraditi na sljedeći način:

```
INSERT INTO studenti(id, ime, prezime)
VALUES (NULL, 'Alina', 'Alinic');
```

NULL je specijalna vrijednost koja predstavlja nedostatak vrijednosti. To znači da će baza podataka odlučiti koju će vrijednost unijeti, ili će jednostavno unijeti NULL. Ukoliko unosimo podatke u sve kolone u tabeli, ne moramo eksplicitno definisati nazive kolona, ali u ovom slučaju moramo definisati vrijednosti za sve kolone:

```
INSERT INTO studenti
VALUES(100,'AAlina','Alinic','Sarajevo','2002-01-02', '2020-02-02','');
```

Prilikom ubacivanja imena potkrala se greška, pa je slovo A duplirano. Da bi smo ovo izmijenili u bazi, koristimo UPDATE komandu. Npr.:

```
UPDATE studenti SET ime='Alina' WHERE id=100;
```

Izmjene će biti primjenjene na sve pronađene redove. Ukoliko se slučajno izostavi WHERE klauzula, onda će izmjene biti primjenjene na sve redove. Na sličan način se može izmijeniti vrijednost više atributa:

```
UPDATE studenti SET ime='Alina', prezime='Alinic' WHERE id=100;
```

Da se obirše neki red u tabeli, koristi se DELETE FROM komanda koja ima istu WHERE klauzulu. Ukoliko se izostavi WHERE klauzula, brisanje će biti primjenjeno na sve redove u tabeli. Npr.:

```
DELETE FROM studenti WHERE id=100;
```

SQL upiti SELECT, FROM, WHERE

Naredba SELECT se koristi za pregled podataka u tabeli. Podaci koji se dobiju se zovu rezultni skup (result-set). Za pregled svih unosa u ovoj tabeli koristit će se komanda SELECT:

```
SELECT * FROM studenti;
```

Slično kao i kod `UPDATE` i `DELETE`, koristimo `WHERE` da bliže označimo koje redove želimo:

```
SELECT * FROM studenti WHERE id=3;
```

Ovo u prevodu znači, selektuj sve redove iz tabele studenti gdje je vrijednost kolone id, 3. Pošto je *id* primarni ključ i time jedinstveni identifikator reda, rezultat će imati samo jedan red.

Zvjezdica nakon ključne riječi `SELECT` znači da se vrate sve kolone koje su dostupne. Umjesto zvjezdice možemo označiti specifične kolone koje želimo da dobijemo. Npr:

```
SELECT id, ime, prezime FROM studenti WHERE id=3;
```

Ovom komandom neće se dobiti vrijednosti za *`datum_rodjenja`* i *`upis`* za studenta pod `id=3`. Za sada, vrijedi spomenuti da se mogu koristiti operatori: `>, <, =, <=, >=` i `<>` za različito.

Generalizujući, sintaksa ove komande izgleda ovako:

```
SELECT column1, column2, ... FROM table_name WHERE condition;
```

Poslije ključne riječi `SELECT` ide lista kolona (ili zvjezdica za sve kolone) za selekciju za izlaz, kao i njihovih modifikatora. Nakon toga ide `FROM` sa tabelom ili listom tabela. Klauzula `WHERE` služi za dodatno filtriranje redova po određenim uslovu ili uslovima.

Pronalazak jedinstvenih vrijednosti uz `SELECT DISTINCT`

Poseban dodatak `SELECT` komandi nam omogućava da vratimo listu jedinstvenih imena. Npr.:

```
SELECT DISTINCT ime FROM studenti;
```

će selektovati sva različita imena tj. sve jedinstvene vrijednosti iz tabele studenti za kolonu *`ime`*. Ovo se također može kombinovati sa `WHERE` izrazom tako da će se `DISTINCT` primijeniti samo na redove filtrirane koristeći `WHERE` izraz.

Preimenovanje kolona

Koristeći `SELECT` upit, moguće je preimenovati izlaznu kolonu, ili izvršiti neku drugu vrstu manipulacije nad kolonom. Ovo se zove *alias* kolone, tj. dodijeljivanje nadimka koloni.

```
SELECT DISTINCT ime AS jedinstvena_imena FROM studenti;
```

Preimenovanje tabela

U SQL upitima moguće je preimenovati tabelu radi lakšeg korištenja. Prije toga, potrebno je reći da su sljedeće dvije komande ekvivalentne:

```
SELECT id, ime, prezime FROM studenti;
SELECT studenti.id, studenti.ime, studenti.prezime FROM studenti;
```

Tj. moguće je navesti naziv tabele prije naziva kolone kako bi se naznačila tabela koja se treba upotrijebiti. Ovo će biti posebno koristno prilikom spajanja tabela kasnije. Pošto je ovo dugo za napisati, moguće je dodijeliti nadimak tabeli. Npr. sljedeća komanda je jednaka prethodnim:

```
SELECT s.id, s.ime, s.prezime FROM studenti s;
```

Ograničavanje izlaznog skupa sa `LIMIT` i `OFFSET`

S obzirom da tabele mogu imati milione redova, ukoliko je potrebno ograničiti količinu redova koje nam baza vraća to se može učiniti korištenjem klauzule `LIMIT` kao dodatka `SELECT` upitu.

```
SELECT * FROM studenti LIMIT 30;
```

će vratiti 30 redova iz tabele studenti. Ova klazulua se stavlja na kraj upita.

Postoje situacije u kojima nije potrebno samo ograničiti broj izlaznih redova. Šta ukoliko imamo web stranicu koja ima listu artikala i ta lista ima svoje stranice. Tako prva stranica ima 10 artikala, druga, slijedećih 10, treća slijedećih 10, itd… U tom slučaju, može se postaviti `OFFSET` što predstavlja prvu poziciju od koje `LIMIT` počinje važiti. Npr.:

```
SELECT * FROM studenti LIMIT 10 OFFSET 20;
```

U ovom slučaju izlazni skup kreće od broja 21, dakle prve dvije stranice su preskočene. Za kraću verziju ovog upita, može se staviti:

```
SELECT * FROM studenti LIMIT 20, 10;
```

gdje par brojeva 20, 10 jednostavno znači: preskoči prvih 20 i vrati 10.

Prebrojavanje redova

Ukoliko je potrebno vratiti broj redova za određen upit, tj. prebrojati redove izlaza, može se koristiti agregacijska funkcija unutar `SELECT` klauzule. Npr.

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

Ovo znači da će baza primjeniti agregacijsku funkciju na izlazni skup podataka. Broj redova za prebrojavanje se može dodatno suziti koristeći `WHERE` klauzulu. Ili sa preimenovanom kolonom:

```
SELECT COUNT(*) AS ukupan_broj_studenata FROM studenti;
```

Vježba 2-2

Priprema:

1. Kreirati tabelu studenti sa zadatim kolonama
2. Unijeti testni skup podataka (5 redova)

Napisati upite za:

1. Ispis svih redova iz tabele studenti.
2. Promijeniti ime i prezime studenta pod id=2.
3. Obrisati sve studente koji se zovu Alisa.
4. Ispis studenta sa id=3.
5. Ispis studenata sa id manjim od 3.
6. Ispis svih studenata sa datumom rodjenja prije 2005.
7. Ispis broja studenata iz prethodnog upita.
8. Ispis svih jedinstvenih imena.
9. Ispis svih jedinstvenih imena i prezimena.
10. Ispis broja svih jedinstvenih imena studenata sa datumom rodjenja poslije 2005.
11. Ispis 3 studenata počevši od studenta pod id=5.
12. Ispis svih jedinstvenih imena studenata pod id manjim od 5.
13. Ispis treće stranice svih studenata, ako stranica ima dva studenta

Vježba 2-2 Rješenja

1. Ispis svih redova iz tabele studenti.
```
SELECT * FROM studenti;
```

2. Promijeniti ime i prezime studenta pod id=2.
```
UPDATE studenti SET ime='Novo', prezime='Prezime 2' WHERE id=2;
```

3. Obrisati sve studente koji se zovu Alisa.
```
DELETE FROM studenti WHERE ime='Alisa';
```

4. Ispis studenta sa id=3.
```
SELECT * FROM studenti WHERE id=3;
```

5. Ispis studenata sa id manjim od 3.
```
SELECT * FROM studenti WHERE id<3;
```

6. Ispis svih studenata sa datumom rodjenja prije 2005.
```
SELECT * FROM studenti WHERE datum_rodjenja < '2005-01-01';
```

7. Ispis broja studenata iz prethodnog upita.
```
SELECT COUNT(*) FROM studenti WHERE datum_rodjenja < '2005-01-01';
```

8. Ispis svih jedinstvenih imena.
```
SELECT DISTINCT ime FROM studenti;
```

9. Ispis svih jedinstvenih imena i prezimena.
```
SELECT DISTINCT ime, prezime FROM studenti;
```

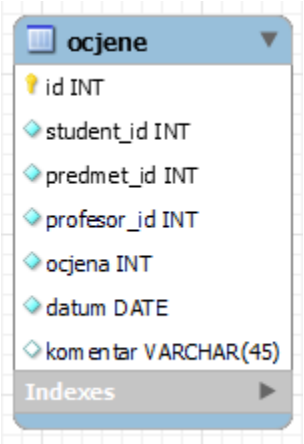
10. Ispis svih jedinstvenih imena studenata pod id manjim od 5.
```
SELECT COUNT(DISTINCT ime)
FROM studenti WHERE datum_rodjenja > '2005-12-31';
```

11. Ispis 3 studenata počevši od studenta pod id=5.
```
SELECT * FROM studenti WHERE id >= 5 LIMIT 3;
```

12. Ispis broja svih jedinstvenih imena studenata sa datumom rodjenja poslije 2005.
```
SELECT DISTINCT ime FROM studenti WHERE id < 5;
```

13. Ispis treće stranice svih studenata, ako stranica ima dva studenta
```
SELECT * FROM studenti LIMIT 2 OFFSET 4;
```

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 tako, međutim vrijednost *student_id* će biti nevezana za najvišu i najmanju ocjenu.

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 i paginacija.

Vježba 2-3

Priprema:

1. Unijeti testnu bazu

Napraviti upite za:

1. Prebrojati broj unesenih ocjena za studenta pod id=1
2. Prebrojati broj komentara za ocjene za studenta pod id=2
3. Pronaći najnižu i najvišu ocjenu na predmetu pod id=1
4. Pronaći najraniji i najkasniji unos ocjene za studenta pod id=2
5. Pronaći prosječnu ocjenu za svakog studenta
6. Pronaći najnižu i najvišu ocjenu za svakog studenta
7. Pronaći prosječnu ocjenu za svakog studenta kod svakog profesora
8. Pronaći prosječnu ocjenu za svakog studenta sortiranu po broju ocjena po studentu od najmanjeg broja ocjena po studentu do najvećeg
9. Ispisati sve studente, sortirati po prezimenu, po imenu i po gradu po abecednom redu
10. Pronaći grad koji ima najveći broj studenata, a ako ih više ima isti broj, vrati onog koji je zadnji po abecedi
11. Pronaći broj studenata rođenih poslije 2004. godini u svakom gradu

Vježba 2-3 Rješenja

1. Prebrojati broj unesenih ocjena za studenta pod id=1
```
SELECT COUNT(*) ocjena FROM ocjene WHERE student_id=1;
```

2. Prebrojati broj komentara za ocjene za studenta pod id=2
```
SELECT COUNT(komentar) FROM ocjene WHERE student_id=2;
```

3. Pronaći najnižu i najvišu ocjenu na predmetu pod id=1
```
SELECT MIN(ocjena), MAX(ocjena) FROM ocjene WHERE predmet_id=1;
```

4. Pronaći najraniji i najkasniji unos ocjene za studenta pod id=2
```
SELECT MIN(datum), MAX(datum) FROM ocjene WHERE student_id=2;
```

5. Pronaći prosječnu ocjenu za svakog studenta
```
SELECT AVG(ocjena) AS prosjek, student_id
FROM ocjene GROUP BY student_id;
```

6. Pronaći najnižu i najvišu ocjenu za svakog studenta
```
SELECT MIN(ocjena) AS najniza, MAX(ocjena) AS najvisa, student_id
FROM ocjene
GROUP BY student_id;
```

7. Pronaći prosječnu ocjenu za svakog studenta kod svakog profesora
```
SELECT AVG(ocjena) AS prosjecna, student_id, profesor_id
FROM ocjene
GROUP BY student_id, profesor_id;
```

8. Pronaći prosječnu ocjenu za svakog studenta sortiranu po broju ocjena po studentu od najmanjeg broja ocjena po studentu do najvećeg
```
SELECT AVG(ocjena) AS prosjecna, COUNT(*) AS brocjena, student_id
FROM ocjene
GROUP BY student_id
ORDER BY brocjena ASC;
```

9. Ispisati sve studente, sortirati po prezimenu, po imenu i po gradu po abecednom redu
```
SELECT * FROM studenti ORDER BY ime, prezime, grad;
```

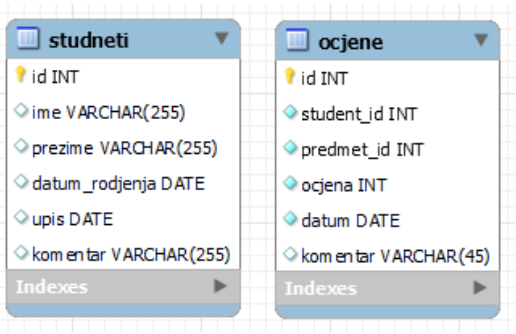
10. Pronaći grad koji ima najveći broj studenata, a ako ih više ima isti broj, vrati onog koji je zadnji po abecedi

```
SELECT grad, COUNT(*) AS br_studenata
FROM studenti
GROUP BY grad
ORDER BY br_studenata DESC, grad
LIMIT 1;
```

11. Pronaći broj studenata rođenih poslije 2004. godini u svakom gradu

```
SELECT COUNT(*) br_studenata, grad
FROM studenti
WHERE datum_rodjenja > '2004-12-31'
GROUP BY grad;
```

2-4 SQL Operatori



Operatori AND, OR, NOT

U prethodnim slučajevima, kada se selektovala vrijednost iz tabele, to se radilo po samo jednom uslovu. Npr. *sve ocjene za studenta pod id=3*. Ili npr. ako je potrebno vratiti sve studente sa ocjenom 5:

```
SELECT * FROM ocjene WHERE ocjena=5;
```

Međutim, nekad želimo da dobijemo sve unose gdje ocjena ***nije*** 5. Za tu svrhu se koristi operator `NOT` koji se stavlja prije uslova.

```
SELECT * FROM ocjene WHERE NOT ocjena=5;
```

Pored `NOT` operatora, postoji niz drugih operatora. `AND` i `OR` operatori daju mogućnost dodatnog filtriranja po jednoj ili više kolona. Evo nekoliko primjera:

```
SELECT * FROM ocjene WHERE ocjena>3 AND student_id=3;
SELECT * FROM ocjene WHERE ocjena=4 OR ocjena=5 AND predmet_id=1;
SELECT * FROM ocjene WHERE NOT ocjena=5 AND student_id=2;
SELECT COUNT(*) FROM ocjene WHERE ocjena=5 AND student_id=3 AND
predmet_id=2;
SELECT * FROM studenti WHERE ime='Alisa' AND NOT prezime='Alisovic';
```

U prevodu, ovi upiti će uraditi sljedeće:

1. Vrati sve ocjene veće od 3 za studenta pod id=3
2. Vrati sve petice i četvrtice na predmetu pod id=1
3. Vrati sve ocjene koje nisu petice za studenta pod id=2
4. Vrati broj petica za studenta pod id=3 na predmetu pod id=2
5. Vrati sve studente sa imenom Alisa ali koja nema prezime Alisovic

Na ovaj način se mogu dodatno filtrirati redovi iz tabele i dobiti željene informacije. Pošto logički operatori mogu biti zbunjujući, moguće je (kao u matematici) grupisati logičke cjeline koristeći zagrade. Zagrade imaju veći prioritet od samih operatora. Npr.:

```
SELECT * FROM studenti WHERE grad='Sarajevo' AND (ime='Alisa' OR ime='Bakir')
```

Ovaj upit će vratiti sve studente koji se zovu Alisa ili Bakir a koje za grad imaju Sarajevo.

Operatori BETWEEN i IN

Ako je potrebno vratiti sve ocjene između 1 i 5 onda možemo napisati:

```
SELECT * FROM ocjene WHERE NOT ocjena=1 AND NOT ocjena=5;
SELECT * FROM ocjene WHERE ocjena>1 AND ocjena<5;
SELECT * FROM ocjene WHERE ocjena>=2 AND ocjena<=4;
```

Svi ovi upiti će dati iste rezultate. Međutim, postoji operator BETWEEN koji se može ovako iskoristiti:

```
SELECT * FROM ocjene WHERE ocjena BETWEEN 2 AND 4;
```

Ovaj upit vraća unose između dvije vrijednosti (uključivo na oba kraja) i daje iste rezultate kao prethodna tri upita. BETWEEN operator selektuje vrijednosti između dva opsega i te vrijednosti mogu biti brojevi, tekst ili datumi, i može se koristiti u kombinaciji i sa drugim operatorima:

```
SELECT * FROM ocjene WHERE ocjena NOT BETWEEN 2 AND 4;
SELECT * FROM ocjene WHERE ocjena BETWEEN 2 AND 4 AND (student_id=3 OR
student_id=4);
```

Prvi upit vraća sve unose ocjene gdje ocjena nije 2, 3 ili 4. Drugi upit vraća sve unose za studenta pod id=3 ili 4 gdje su ocjene 2, 3 ili 4. Međutim, ovaj upit se može elegantnije napisati koristeći operator IN. Operator IN potvrđuje da li se traženi pojam nalazi u datom skupu. Tako, drugi upit se može napisati i kao:

```
SELECT * FROM ocjene WHERE ocjena BETWEEN 2 AND 4 AND student_id IN (3,4);
```

Na ovaj način, može se dodati mnogo veća lista studenata, sa mnogo manje teksta. Ovo radi na istom principu i sa nenumeričkim kolonama. Npr.:

```
SELECT * FROM studenti WHERE ime IN ('Alisa', 'Bakir');
```

Ovaj upit će vratiti sve studente koje za ime imaju jedno od imena u datom skupu. Na ovaj način ne moramo pisati duge OR izraze. Svi ovi operatori se mogu koristiti u kombinaciji jedni sa drugima, a pošto može postojati logička nepreciznost, mogu se koristiti zagrade kako bi se logički izrazi grupisali.

Operator LIKE

U prethodnim primjerima, pretraživani su unosi studenata koji za ime imaju Alisa ili Bakir. Međutim recimo da u tabeli postoje imena poput Alice, Amra, Admir, Bob, Bruno itd. Koristeći = u WHERE klauzuli, mogu se dobiti samo precizna podudaranja, tj. podaci moraju u potpunosti odgovarati traženom pojmu. Ali ako je potrebno izvući sve studente koji za prvo slovo imena imaju A onda se to ne može uraditi na taj način.

Tu može pomoći operator `LIKE`. `LIKE` operator je operator koji vrši šablonsko pretraživanje. Radi samo na tekstualnim podacima i može odgovoriti na pitanja, da li podatak počinje ili završava ili možda sadrži određen tekst. Npr.:

```
SELECT * FROM studenti WHERE ime LIKE 'A%';
SELECT * FROM studenti WHERE ime LIKE '%kir';
```

Prvi upit vraća sve studente čije ime počinje sa slovom A. Drugi vraća sve studente čije ime završava sa `'kir'`. Npr. Bakir. Znak % predstavlja nula, jedan ili više karaktera. Ako treba tražiti podudaranje samo jednog karaktera, onda se u tu svrhu može koristiti simbol _. Npr.:

```
SELECT * FROM studenti WHERE ime LIKE 'Ali__';
```

Ovdje je rečeno da se vrate svi studenti kojima ime počinje sa Ali a završava sa bilo koja, tačno dva karaktera. Pa će tako ovaj upit pronaći imena poput Alisa, Alice, Alina itd. Mogao se koristiti i simbol % npr. `'Ali%'` međutim ovo bi vratilo i neko ime, recimo, Ali, što nije traženo.

Ova dva simbola se mogu i kombinovati. Tako, ako je potrebno pronaći sve unose sa imenom koje ima slovo k na drugom mjestu rekli bi:

```
SELECT * FROM studenti WHERE ime LIKE '_k%';
```

I ovaj operator se može koristiti sa ostalim operatorima. Tako ako želimo npr. sve studente čija imena ne počinju sa Ali, možemo napisati:

```
SELECT * FROM studenti WHERE ime NOT LIKE 'Ali%';
```

ili ako želimo sve studente koji počinju sa A ili B:

```
SELECT * FROM studenti WHERE ime LIKE 'A%' OR ime LIKE 'B%';
```

Operator `IS [NOT] NULL`

Kako bi vratili podatke u tabeli za koje nema unijetih podataka (tj. neka je prilikom kreiranja tabele, dozvoljeno da podaci u koloni mogu budu prazni) koristimo `IS NULL`:

```
SELECT * FROM studenti WHERE komentar IS NULL;
```

Ova komanda će vratiti sve studente koji nemaju komentara. Slično, možemo pronaći samo studente koji imaju unesen komentar:

```
SELECT * FROM studenti WHERE komentar IS NOT NULL;
```

Vježba 2-4

Priprema:

1. Unijeti testnu bazu

Napisati upite za:

1. Pronaći sve ocjene gdje je ocjena 3
2. Pronaći sve ocjene gdje predmet_id nije između 2 i 5 osim ako je 3
3. Pronaći sve studente gdje je datum rođenja u martu 2002.
4. Pronaći prvih troje studenata po prosjeku. Ako ih više ima isti prosjek, vratiti prvih troje koji imaju najveći ID.
5. Pronaći prosjek svih ocjena unesenih poslije 01.09.2015. za studenta pod id=2
6. Pronaći sve studente koji imaju ime da počinje sa B i završava sa r.
7. Pronaći sve studente kojima ime ima 5 slova i počinje sa A.
8. Pronaći sve studente kojima prezime počinje sa B ili D.
9. Pronaći sume ocjena po studentu i profesoru koji je dao tu ocjenu, a da je data u 2017. godini i da je sortirano po sumi od najveće ka najmanjoj. Vratiti listing treće stranice ako stranica ima 3 reda.
10. Pronaći sve ocjene koje imaju komentar.
11. Pronaći sve studente koji nemaju komentar.

Vježba 2-4 Rješenja

1. Pronaći sve ocjene gdje je ocjena 3
```
SELECT * FROM ocjene WHERE ocjena=3;
```

2. Pronaći sve ocjene gdje predmet_id nije između 2 i 5 osim ako je 3
```
SELECT * FROM ocjene
WHERE predmet_id=3 OR predmet_id NOT BETWEEN 2 AND 5;
```

3. Pronaći sve studente gdje je datum rođenja u martu 2002.
```
SELECT * FROM studenti
WHERE datum_rodjenja BETWEEN '2002-03-01' AND '2002-03-31';
```

4. Pronaći prvih troje studenata po prosjeku. Ako ih više ima isti prosjek, vratiti prvih troje koji imaju najveći ID.
```
SELECT AVG(ocjena) AS prosjek, student_id
FROM ocjene
GROUP BY student_id
ORDER BY prosjek DESC, student_id DESC
LIMIT 3;
```

5. Pronaći prosjek svih ocjena unesenih poslije 01.09.2015. za studenta pod id=2
```
SELECT AVG(ocjena) FROM ocjene
WHERE datum > '2015-09-01' AND student_id=2;
```

6. Pronaći sve studente koji imaju ime da počinje sa B i završava sa r.
```
SELECT * FROM studenti WHERE ime LIKE 'B%r';
```

7. Pronaći sve studente kojima ime ima 5 slova i počinje sa A.
```
SELECT * FROM studenti WHERE ime LIKE 'A____';
```

8. Pronaći sve studente kojima prezime počinje sa B ili D.
```
SELECT * FROM studenti WHERE prezime LIKE 'B%' OR prezime LIKE 'D%';
```

9. Pronaći sume ocjena po studentu i profesoru koji je dao tu ocjenu, a da je data u 2017. godini i da je sortirano po sumi od najveće ka najmanjoj. Vratiti listing treće stranice ako stranica ima 3 reda.
```
SELECT SUM(ocjena) AS suma, student_id, profesor_id
FROM ocjene
WHERE datum BETWEEN '2017-01-01' AND '2017-12-31'
GROUP BY student_id, profesor_id
ORDER BY suma DESC
LIMIT 3 OFFSET 6;
```

10. Pronaći sve ocjene koje imaju komentar.
```
SELECT * FROM ocjene WHERE komentar IS NOT NULL;
```

11. Pronaći sve studente koji nemaju komentar.
```
SELECT * FROM studenti WHERE komentar IS NULL;
```

Test 1 Primjer

Ime i prezime: _____
Grupa ?

1. Šta je indeksiranje?

2. Koja je razlika između operatora `BETWEEN` i `IN`?

3. Potrebno je dizajnirati bazu podataka za vođenje evidencije o glumcima na projektu. Potrebno je evidentirati ime, prezime, datum zaposlenja, datum rođenja, visinu, težinu, vrstu angažmana koja može biti glavni, sporedni glumac ili statista, datum prestanka rada na projektu ako je prestao raditi i da li ima svoje auto ili ne. Potrebno je opisati, riječima i slikom, ograničenja, tipove i sve potrebne detalje.

4. Napisati upite za prethodno pitanje:
    a. Pronaći broj jedinstvenih imena glumaca
    b. Pronaći broj glumaca uposlenih u 2015. godini, po angažmanu
    c. Izmjenu vrste angažmana svih glumaca uposlenih u 2015. sa glavni na sporedni
    d. Pronaći prosječnu težinu glumaca istog prezimena

5. Potrebno je dizajnirati bazu podataka za vođenje evidencije o pacijentima. Potrebno je evidentirati ime i prezime pacijenta, razlog boravka, datum rođenja, datum prijema, datum odjave, i komentar ako je doktor unio. Potrebno je opisati, riječima i slikom, ograničenja, tipove i sve potrebne detalje.

6. Napisati upite za prethodno pitanje:
    a. Pronaći sve pacijente kojima prezime počinje sa K i ima 6 karaktera sortirano po imenu. Ispisati 3. Stranicu ako stranica ima 10 redova.
    b. Pronaći zadnjeg odjavljenog pacijenta za kojeg postoji komentar. Ako ih ima vise onda treba pronaći onog koji je prvi po abecedi prezimenom i imenom.

Test 1 Primjer Rješenje

1. Indeksiranje je proces koji omogućava brzo pretraživanje tabela po određenoj koloni.

2. `BETWEEN` je operator koji vrši selekciju nečega ukoliko se nalazi između dvije vrijednosti dok `IN` je operator koji vrši selekciju nečega ukoliko to ima vrijednost kao jedna od vrijednosti iz skupa.

3.

Glumci		
#	id	INT, AUTO_INC
*	ime	VARCHAR(100)
*	prezime	VARCHAR(100)
*	zaposlen	DATE
*	rodjen	DATE
*	visina	INT
*	tezina	INT
*	anganzman	CHAR, ‘G’ – glavni ‘S’ – sporedni ‘T’ – statista
+	prestanak	DATE
*	auto	CHAR, ‘D’ – da, ‘N’ – ne

4.

a.
```
SELECT COUNT(DISTINCT ime)
FROM glumci;
```

b.
```
SELECT COUNT(*), angazman
FROM glumci
WHERE zaposlen BETWEEN '2015-01-01' AND '2015-12-31'
GROUP BY angazman;
```

c.
```
UPDATE glumci
SET angazman = 'S'
WHERE angazman = 'G' AND zaposlen BETWEEN '2015-01-01' AND '2015-12-31';
```

d.
```
SELECT AVG(tezina) AS prosjek, prezime
FROM glumci
GROUP BY prezime;
```

5.

pacijenti		
#	id	INT, AUTO_INC
*	ime	VARCHAR(100)
*	prezime	VARCHAR(100)
*	razlog	VARCHAR(255)
*	rodjen	DATE
*	prijem	DATE
+	odjava	DATE
+	komentar	TEXT

6.

a.
```
SELECT *
FROM pacijenti
WHERE prezime LIKE 'K _ _ _ _ _'
ORDER BY ime
LIMIT 10 OFFSET 20;
```

b.
```
SELECT *
FROM pacijenti
WHERE komentar IS NOT NULL
ORDER BY odjava DESC, prezime, ime
LIMIT 1;
```

Zadaća 2-1

Ukoliko postoji baza podataka za vođenje evidencije o računima u banci:

računi		
#	id	INT AUTOINC
*	vlasnik_ime	VARCHAR(100)
*	vlasnik_prezime	VARCHAR(100)
*	stanje	DECIMAL (10,2)
*	tip	VARCHAR(100), “tekući” – tekući “štedni” – štedni
*	poslovnica_grad	TEXT
+	napomena	TEXT
*	kreiran	DATE

Napisati upite koji će:

1. Pronaći osobu koja najviše novca na štednom računu.
2. Pronaći koliko ima računa svih tipova.
3. Pronaći sve vlasnike koji su kreirali račune tokom marta 2021. i na kojima stoji napomena a čije prezime počinje sa K i završava sa ić.
4. Ispisati sva jedinstvena imena koja postoje u bazi koji su otvorili račun u maju 2020. i prezivaju se na D.
5. Brisanje svih računa kreiranih u poslovnici u Mostaru tokom 2019. godine a u napomeni se spominje „za brisati“.

Opisati riječima šta rade sljedeći upiti:

6.
```sql
SELECT poslovnica_grad, COUNT(*) broj
FROM računi
GROUP BY poslovnica_grad
ORDER BY poslovnica_grad;
```

7.
```sql
SELECT SUM(stanje) uk, tip
FROM računi
GROUP BY tip;
```

8.
```sql
SELECT DISTINCT vlasnik_ime, vlasnik_prezime
FROM računi;
```

9.
```sql
SELECT vlasnik_ime, vlasnik_prezime
FROM računi
WHERE poslovnica_grad='Tuzla' AND kreiran
   BETWEEN '2016-01-01' AND '2016-01-31' AND napomena IS NOT NULL;
```

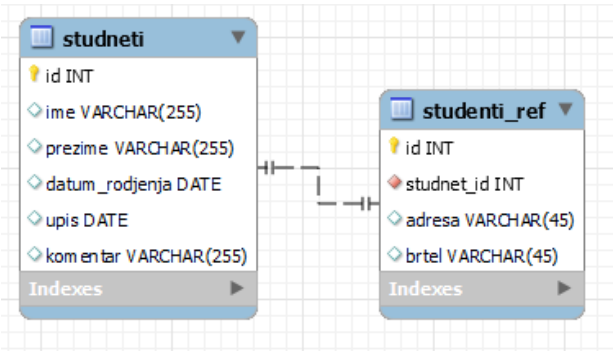
2-5 Veze između tabela

Tipovi veza

Prije detaljnog opisa spajanaj tabela, potrebno je opisati tipove veza koje mogu biti između tabela. Ti tipovi su:

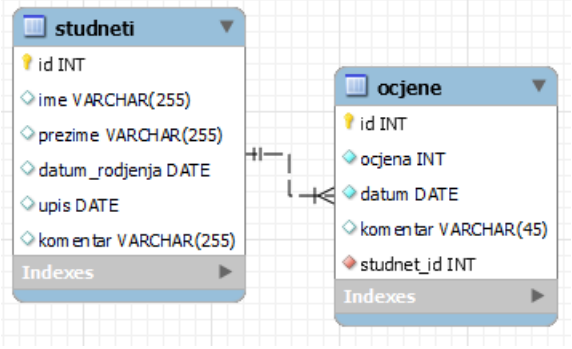
- Jedan na jedan
- Jedan na više
- Više na više

Recimo da imamo potrebu unijeti neke ostale referentne podatke za stuenta, poput, broj telefona, ulica, grad, poštanski broj, adresa, i još neke druge podatke koje ne želimo da pohranjujemo u tabelu studenti radi jednostavnosti. Ono što se može napraviti je tabela sa referentnim podacima, a koja se referira na tabelu studenti, tako označavajući kojem studentu referentni podaci pripadaju. Npr.:



Veza jedan-na-jedan

Ovo je vrsta veze jedan-na-jedan. Na slici je **ERD** (**E**ntity **R**elationship **D**iagram) diagram koji predstavlja vezu jedan-na-jedan i ona je predstavljena sa dvije uspravne crte na oba kraja veze. U praksi, ovo znači da će studenti_ref imati strani ključ *student_id* koji predstavlja studenta za kojeg su ostale vrijednosti vezane. Veze jedan-na-jedan nam mogu pomoći da proširimo skup i odvojimo nužne od opcionalnih podataka.

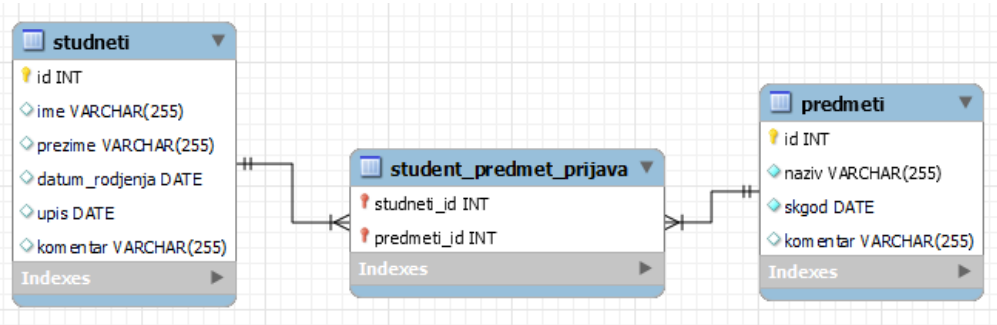


Veza jedan-na-više

Ukoliko je potrebno da jedan entitet ima više unosa za jedan element drugog entiteta, onda trebamo koristiti vezu jedan na više. Takva veza je prikazana na slici iznad i predstavlja potrebu za spašavanjem više ocjena za jednog studenta. Na ERD dijagramu ova veza se prikazuje sa crticama na jednoj strani i trokutom na drugoj strani i u tom smijeru čita se, jedan na više. Za suprotnu orijentaciju potrebno je zamijeniti trokut i crtice.

Ovo se tumači kao: jedan student može imati više ocijena. Potrebno je naglasiti da ova vrsta veze se implementira koristeći strani ključ. Pa je tako u tabeli ocjena, student predstavljen koristeći `student_id` i upravo to je strani ključ u tabeli `ocjene`. Vrijednost ove kolone se može ponavljati i upravo zato se ova veza zove jedan na više, jer jedan unos u tabeli studenti se može ponavljati više puta u tabeli `ocjene`.

Treća i zadnja veza koja postoji je više-na-više. Ove veze se prave kada je potrebno da se više unosa u jednoj tabeli spoji sa više unosa u drugoj tabeli. Recimo ako je potrebno modelirati prijavu studenata na nekom predmetu. U ovom slučaju, student može biti prijavljen na više predmeta ali istovremeno, jedan predmet može imati više studenata. Veza više-na-više se implementiraju koristeći dvije veze jedan-na-više koristeći međutabelu.



Veza više-na-više

Informaciju koji student je upisan na koji predmet se nalazi u tabeli `student_predmet_prijava`. Međutabela ne mora nužno imati samo strane ključeve. Moguće je dodati dodatna polja u ovu tabelu jer je ovo tabela kao i svaka druga.

Napomena oko tipova podataka

Spašavanje datoteka

Prilikom modeliranja baza podataka, potrebno je spasiti različite tipove podataka. Međutim oni tipovi koji postoje u bazama ne odgovaraju nužno potrebama u stvarnom svijetu.

Npr. ukoliko je potrebno spasiti neki dokument, sliku, video, kako bi to bilo najbolje uraditi? Poznajemo `BLOB` tip podataka za bilo koji niz nula i jedinica koji želimo spasiti. Međutim, detaljnijom analizom vidimo da `BLOB` može imati maksimalnu dužinu od 65535 bajti. Ovo znači da ne možemo spasiti ni veliku datoteku niti veliku sliku a pogotov ne video. Stoga, koji je najbolji način spasiti datoteku?

Najbolji način je zapravo da se datoteka ne spašava u bazu uopšte. Ono što se zapravo spasi je putanja do datoteke na disku, gdje se ona nalazi. Na ovaj način, baza podataka se ne opterećuje bespotrebnim podacima, a aplikacija zna gdje se datoteka nalazi, i korisnik može spasiti bilo koju veličinu dok god postoji dovoljno prostora na disku. Tako da se u bazi unosi samo `VARCHAR` ili `TEXT` polje koje označava putanju na kojoj se datoteka nalazi.

Razlika između `VARCHAR` i `TEXT` je što `VARCHAR` obično ima maksimalnu dužinu od 255 bajti, a `TEXT` od 65535 bajti ili karaktera. Još jedan tip podataka je `CHAR` koji isto tako ima maksimalnu dužinu od 255, međutim za razliku od `VARCHAR`-a, spašavanje `CHAR`-a zauzima tačno onoliko karaktera koliko je rezervisano dok `VARCHAR` može zauzimati i manje od rezervisanog broja.

Spašavanje šifri

Ukoliko želimo spasiti nečiju šifru u bazu podataka, prvi izbor bi bio vjerovatno `VARCHAR`. Taj izbor može biti u redu zavisno od onoga ŠTA se stavlja u taj `VARCHAR`. Ukoliko spasimo šifru, upravo onako kako ju je korisnik unio, tzv. cleartext, koji je problem sa kojim bi se tako dizajnirana baza podataka susrela?

Zapravo, problem nije tehničke prirode, već ljudske. Baze podataka bivaju hakovane te ovi podaci mogu doći u “ruke” nepovjerljivih osoba. Osim hakera, ove osobe mogu biti i administratori baza podataka, ili neko ko je hakovao njih. Postoji cijeli lanac nepovjerenja kao razlog zašto šifre ne bi trebale biti spašene u bazu podataka u izvornom obliku. Stoga se postavlja pitanje, kako onda spasiti šifru?

Za spašavanje šifri iskoristit će se jednosmjerna *hash* funkcija pod nazivom `SHA1`. Ova funkcija proizvodi *hash* na osnovu ulaznog stringa. Pretvaranje stringa u *hash* se zove *hash*-iranje. `SHA1` je samo jedna funkcija iz porodice *hash* funkcija. Karakteristike *hash* funkcija su da kreiraju *hash* iste dužine za ulazni tekst bilo koje dužine, tako da dva teksta različite dužine daju *hash* iste dužine. I druga karakteristika je da je šansa za kolizije veoma veoma mala, tj. ne postoje dva stringa bilo koje dužine koja mogu proizvesti isti *hash*. Treća karakteristika ovih funkcija je da nije moguće *hash* pretvoriti nazad u string, tj. ne postoji inverzna fukcija koja bi od *hash*-a vratila prvobitni string.

Posljedica ovih funkcija za šifre je to da zapravo u bazu možemo spasiti *hash* šifre, umjesto samu šifru. Zbog karakteristike jednosmjernosti, šifra se ne može povratiti nazad, a sama šifra bi se koristila samo tokom `INSERT` ili `UPDATE` komandi. Npr.:

```
UPDATE korisnici SET sifra = SHA1('sifra4321') WHERE id=5;
```

Na ovaj način smo zaštitili šifru od neželjenih pogleda. Ali kako će aplikacija provjeriti je li šifra tačna prilikom logovanja korisnika? Jednostavno će *hash*-irati šifru koji je korisnik unio i uporediti da li *hash* te šifre, odgovara spašenom *hash*-u u bazi. Npr.:

```
SELECT COUNT(*) FROM korisnici
WHERE email='test@email.com' AND sifra=SHA1('sifra');
```

Vježba 2-5

Priprema:

1. Instalirati MySQL Workbench
    a. https://cdn.mysql.com//Downloads/MySQLGUITools/mysql-workbench-community-8.0.28-winx64.msi
    b. To je alat koji omogućava crtanje ERD dijagrama
    c. Kao i konfigurisanje baze koristeći GUI
    d. Alternativa https://app.diagrams.net/

Zadatak:

1. Potrebno je modelirati sistem koji vrši evidenciju korisnika u školi (e-dnevnik). Ovi korisnici mogu biti studenti i profesori. Obje vrste korisnika imaju polja poput, imena, prezimena, datuma rodjenja. Profesori imaju datum početka rada. Studenti imaju datum upisa u školu. Osim korisnika, potrebno je vođenje evidencije predmetima. Na predmetu može biti više od jednog studenta, a svaki student može biti upisan na više predmeta. Isti predmet se ponavlja svake godine, stoga je potreban i datum početka nastave na predmetu. Predmet vodi jedan profesor. Potrebno je modelirati i evidenciju o ocjenama na predmetu. Tokom studija, profesor studenta ocjenjuje više puta. Potrebno je evidentirati i datum kraja predmeta.

2. Dizajnirati bazu podataka za aplikaciju za razmjenu poruka (kao Viber). Potrebno je evidentirati registraciju korisnika. Nakon registracije, korisnici mogu slati poruke jedni drugima. Osim ovoga, svaki korisnik može kreirati grupe u koje može dodati druge korisnike, i svi oni međusobno mogu razmijenjivati poruke. Isto tako, grupe mogu biti tzv. read-only tj. samo korisnik koji je kreirao grupu može da šalje poruke. Korisnici nemaju ograničenja u koliko grupa sudjeluju. Osim ovoga, potrebno je omogućiti blokiranje korisnika, tj. jedna osoba može blokirati jednu ili više osoba kako mu te osobe ne bi slale poruke.

3. Potrebno je modelirati bazu podataka za muzički studio. Potrebno je evidentirati osobe koje mogu biti producenti ili autori sa imenom i prezimenom bez dupliciranja podataka jer ista osoba može biti i producent i autor. Potrebno je evidentirati i pjesme koje moraju imati naziv i, opcionalno, datum objave. Pored toga, pjesma mora biti dio albuma tj. pjesma se može nalaziti u jednom i samo jednom albumu. Album mora posjedovati naziv i jednu pjesmu može producirati više producenata kao i autorisati više autora. Pjesma isto tako ima žanr koji mora imati naziv, i potrebno je izbjeći dupliciranje podataka.

4. Dizajnirati bazu podataka za sistem za upravljanje dokumentima. Pošto je web sistem, korisnici se trebaju registrovati da bi mu pristupili. Nakon toga, korisnici mogu kreirati ili upload-ovati bilo koji dokument, i pregledati ga ukoliko za to postoji mogućnost. Pored ovoga, potrebno je evidentirati mogućnost dijeljenja dokumenata. Samo jedan korisnik može kreirati dokument, međutim on može dati ovlasti čitanja ili mijenjanja tog dokumenta drugom korisniku. Osim ovoga, korisnici mogu slati poruke me, nevezano za dokument, ali moguće je i da poruka bude uvezi nekog dokumenta.

Zadaća 2-2

1. Dizajnirati bazu podataka za sistem za vođenje evidencije o web portalu za tv kanale. Na ovom portalu, korisnici mogu pregledati koji sve kanali postoje, a administratorski korisnici mogu dodavati nove kanale. Osim vođenja evidencije o tv kanalima, sistem treba da vodi evidenciju i o tv emisijama koje mogu biti serije, filmovi ili nekog drugog tipa. Sve tv emisije se mogu emitovati na jednom ili više kanala, te je potrebno korisnicima omogućiti uvid u to kada se koja emisija na kojem kanalu pojavljuje. Korisnici trebaju biti u mogućnosti evidentirati koje su im omiljene emisije kako bi im sistem mogao prikazati kada i na kojem kanalu će biti njihove omiljene emisije. Osim označavanja omiljenih emisija, korisnici mogu ocijenjivati emisije (sve emisije, serije, filmove i ostale), te mogu pisati komentare na emisije kako bi drugi korisnici mogli vidjeti iskustva drugih korisnika. Potrebno je nacrtati i opisati sve tabele, tipove, ograničenja i veze između tabela.

2. Dizajnirati bazu podataka za sistem za vođenje evidencije na turniru u tenisu. Na turniru se registruju takmičari, te učestvuju na mečevima. Da bi turnir bio profitabilan, potrebno je omogućiti registraciju sponzora. Sponzori mogu sponzorisati jedan ili više mečeva, a mogu ujedno biti sponzori jednog ili više takmičara istovremeno. Sponzorstva se mogu ponavljati te imaju svoj datum početka i kraja. Potrebno je nacrtati i opisati sve tabele, tipove, ograničenja i veze između tabela.

3. Dizajnirati bazu podataka za sistem za vođenje evidencije o aplikacijama (npr. AppStore ili PlayStore). Ovaj sistem treba omogućiti registraciju korisnika te dodavanje aplikacija u sistem od strane administratora. Svi korisnici mogu pretraživati aplikacije koje su dostupne te ih preuzeti. Sistem treba evidentirati koji je korisnik preuzeo koje aplikacije. Osim preuzimanja, sistem treba omogućiti korisnicima davanje ocijene aplikaciji, kao i pisanje komentara. Radi potvrde valjanosti komentara treba omogućiti lajkanje i dislajkanje komentara. Radi zaštite od spama, i drugih nepravilnosti, potrebno je omogućiti korisnicima da prijave neregularne komentare kako bi ih administratori mogli revidirati. Potrebno je nacrtati i opisati sve tabele, tipove, ograničenja i veze između tabela.

2-6 Spajanje tabela

JOIN

U vježbi 2-4, pod brojem 9. (u pojednostavljenoj varijanti) tražena su imena svih studenata sa ocjenom 5. Međutim, imena se ne nalaze u tabeli ocjene. Jedini način da se ovo uradi je da se na neki način povežu tabele studenti i ocjene. U tabeli ocjene, postoji kolona *student_id* koja zapravo predstavlja studenta sa kojim je ocjena data. Ovo se može iskoristiti na sljedeći način:

```
SELECT studenti.ime, studenti.prezime, ocjene.ocjena
FROM studenti, ocjene
WHERE studenti.id = ocjene.student_id AND ocjene.ocjena=5;
```

Za razliku od prethodnih upita, sada su korištene dvije tabele u `FROM` klauzuli. Kako bi se skratilo pisanje, može se koristiti aliasing:

```
SELECT s.ime, s.prezime, o.ocjena
FROM studenti s, ocjene o
WHERE s.id = o.student_id AND o.ocjena=5;
```

Ovo je jedna vrsta JOIN-a, ili povezivanja jedne ili više tabela (u ovom slučaju dvije). Ova vrsta povezivanja je implicitno lijevo unutrašnje povezivanje (*LEFT INNER JOIN*). Razlikuje se nekoliko tipova veza, unutrašnje i vanjsko (*INNER i OUTER*) i lijevo i desno (*LEFT i RIGHT*).

Potrebno je spomenuti da se ovi nazivi djelomično razlikuju od DBMS-a do DBMS-a međutim u ovom slučaju govorimo o MySQL ili MariaDB bazi podataka. U MySQL bazi postoje ove vrste veza:

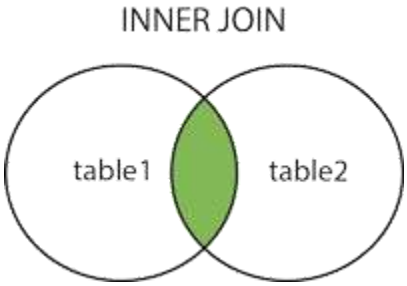
- `INNER JOIN`
- `LEFT OUTER JOIN` ili `LEFT JOIN`
- `RIGHT OUTER JOIN` ili `RIGHT JOIN`
- `FULL OUTER JOIN` ili `CROSS JOIN`
- `SELF JOIN`
- `UNION / ALL`

JOIN se koristi tako što se naznače ključne riječi koje predstavljaju vezu a potom kolone preko kojih se dvije tabele vežu. Npr.

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

INNER JOIN

Ova vrsta JOIN-a vraća redove koje imaju unose u obje tabele.



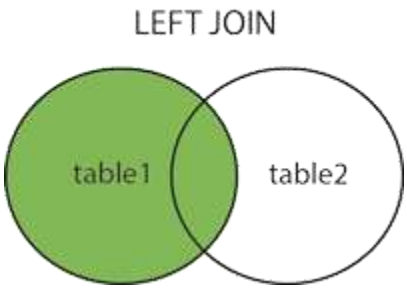
Npr. ako je potrebno pronaći sve učenike sa ocjenom 5, trebamo spojiti tabele studenti i ocjene

```
SELECT s.ime, s.prezime, o.ocjena
FROM studenti s INNER JOIN ocjene o ON s.id = o.student_id
WHERE o.ocjena=5;
```

Primjeti se da su rezultati ovog upita isti kao rezultati implicitnog upita, gdje su dvije tabele spojene definišući vezu preko `WHERE` klauzule. Potrebno je reći da je rezultat između ova dva upita isti međutim eksplicitno definisanje `INNER JOIN` pruža mogućnost bazi da bolje optimizuje izvršenje upita pa tako eksplicitni `INNER JOIN` ima bolje performanse.

LEFT JOIN

`LEFT JOIN` vraća sve rezultate iz lijeve tabele, i podudarajuće rezultate iz desne tabele.



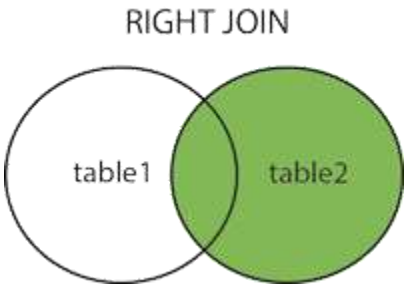
Ukoliko nema podudarajućih rezultata, na njihovom mjestu stajat će `NULL`. Npr. ukoliko je potrebno pronaći sve studente i njihove ocjene, uključujući i studente koji nisu ocijenjeni, onda se to može uraditi na sljedeći način:

```
SELECT s.id, s.ime, s.prezime, o.ocjena
FROM studenti s
LEFT JOIN ocjene o ON s.id = o.student_id;
```

Jednako isti rezultat bi bio dobiven ako bi se koristilo `LEFT OUTER JOIN`.

RIGHT JOIN

`RIGHT JOIN` je obratno od `LEFT JOIN` i vraća sve rezultate iz desne tabele i podudarajuće rezultate iz lijeve tabele.

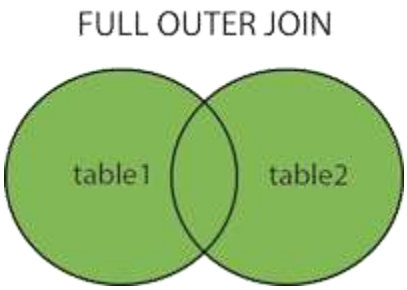


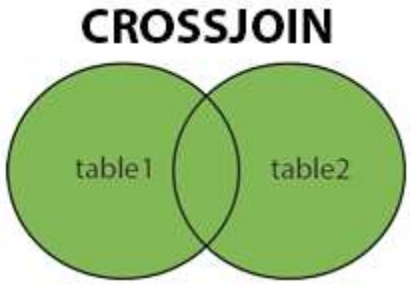
Npr. ako želimo vratiti sve ocjene i podudarajuće predmete, uključujući predmete koji nemaju ocjene, možemo reći:

```
SELECT o.ocjena, p.naziv
FROM ocjene o
RIGHT JOIN predmeti p ON p.id = o.predmet_id;
```

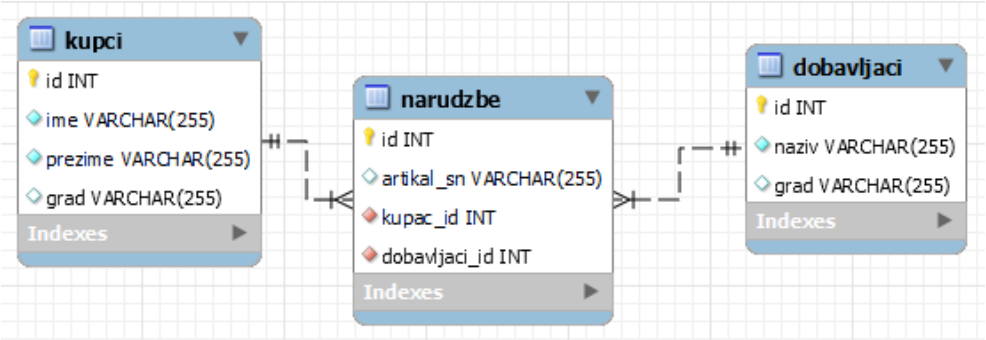
FULL JOIN ili `CROSS JOIN`

Puno spajanje dvije tabele u izlazu vraća rezultate iz obje tabele. Ovo se također zove `FULL OUTER JOIN`. U MySQL bazi podataka, ovo se zove `CROSS JOIN`:





Ovaj JOIN, vraća sve rezultate iz obje tabele, bez obzira ima li podudaranja ili ne. Npr. recimo da postoje tabele za kupce, narudžbe i dostavljače:



```
SELECT k.ime, k.prezime, n.artikal_id
FROM kupci k
CROSS JOIN narudzbe n ON k.id = n.kupac_id;
```

Pošto `CROSS JOIN` vraća vrijednosti iz obje tabele bez obzira ima li podudaranja ili ne, stoga će u izlazu biti kupci koji nemaju nikakvih narudžbi ali i narudžbe za koje nema podudarajućeg polja u tabeli kupci.

SELF JOIN

SELF JOIN je obični JOIN kao i svaki drugi, stim da se tabela spaja sama sa sobom. Na primjeru tabele kupci, koristeći SELF JOIN možemo vratiti sve kupce koji se nalaze u istom gradu:

```
SELECT a.ime, a.prezime, b.ime, b.prezime, a.grad
FROM kupci a, kupci b
WHERE a.id <> b.id AND a.grad = b.grad;
```

Ovaj upit će vratiti sve kupce koji se nalaze u istom gradu.

UNION / ALL

`UNION` operator se koristi da kombinira rezultate dva ili više `SELECT` izraza, međutim, pošto funkcioniše kao unija dva skupa, da bi funkcionisalo, mora biti ispunjeno par zahtjeva:

- Svaki `SELECT` upit mora imati isti broj kolona
- Sve kolone moraju imati podudarajuće tipove podataka
- Kolone u svakom `SELECT` izrazu moraju biti na istim pozicijama

Npr. ako želimo da pronađemo sve gradove iz tabele kupci i iz tabele dobavljači, možemo napisati:

```
SELECT grad FROM kupci
UNION
SELECT grad FROM dobavljaci;
```

Ovo će vratiti sva jedinstvena imena gradova iz dvije tabele, tj. bez duplikata. Ako želimo da vratimo sa duplikatima, možemo koristiti `UNION ALL`:

```
SELECT grad FROM kupci
UNION ALL
SELECT grad FROM dobavljaci;
```

Vježba 2-6

Priprema:

1. Obrisati i importovati testnu bazu

Napraviti upite za:

2. Ispisati sve studente imenom i prezimenom i njihove ocjene
3. Ispisati sve studente i njihove ocjene, kao i predmete na kojima je ta ocjena dobijena sortirano po nazivu predmeta od Z do A
4. Pronaći sve studente nazivom i njihove ocjene kao i profesore koji su ih dali i nazive predmeta na kojima je ta ocjena, sortirano po datumu
5. Pronaći sve studenti i njihove ocjene, na predmetu “Baze podataka”
6. Pronaći nazive studenata i broj ocjena koji oni imaju
7. Pronaći nazive studenata i prosječnu ocjenu po predmetu
8. Pronaći nazive studenata i kada mu je koji profesor na kojem predmetu dao najstariju ocjenu
9. Pronaći predmete i datume najstarije ocjene
10. Pronaći prosječnu ocjenu po svakom predmetu
11. Pronaći prosječnu ocjenu na predmetima čiji se naziv završava sa “ika”
12. Pronaći najstariju i najnoviju ocjenu profesora

2-7 SQL Aritmetika i funkcije

SQL Aritmetika

Prilikom pravljenja upita, moguće je koristiti i običnu aritmetiku. Pod ovim se misli na sabiranje (+), oduzimanje (-), dijeljenje (/), množenje (*) i ostatak cjelobrojnog dijeljenja (%). Npr.:

```
SELECT 2*3;
SELECT 32%3;
```

su potpuno validni upiti. Prvi će dati rezultat 6, a drugi ostatak cjelobrojnog dijeljenja sa 3. Ove operacije se mogu obaviti nad kolonama numeričkog tipa i mogu se koristiti u `SELECT`, `WHERE` i `UPDATE` klauzulama. Aritmetičke operacije se mogu primjeniti na jednu ili više kolona. Npr.:

```
SELECT ocjena + 5 FROM ocjene;
SELECT * FROM studenti WHERE id%2=0;
UPDATE statistika SET brstudenata = brstudenata + 30;
SELECT popust*cijena AS prava_cijena FROM artikli;
```

Prvi upit vraća sve ocjene uvećane za 5, drugi vraća studente sa parnim id brojem, a treći uvećava broj studenata u tabeli statistika. Četvrti upit prikazuje da se aritmetika može koristiti i na više od jedne kolone pa tako primjenjuje popust na cijenu za svaki artikal.

SQL Funkcije

Aritmetičke operacije su na neki način i funkcije specijalizovane za numeričke tipove podataka. Ponekad je potrebno izvršiti manipulaciju drugih tipova podataka. Već su bile prikazane određene vrste funkcija poput `SUM`, `COUNT`, `MIN`, `MAX` ali to su agregacijske funkcije i one se upotrebljavaju prilikom grupisanja podataka.

Funkcije nad stringovima

Tekstualni tipovi podataka u računarima se kolokvijalno zovu stringovi. Recimo da nam je traženi studenati sa komentarima kraćim od 20 znakova. Za tu svrhu, potrebno je znati dužinu komentara. Za tu svrhu nam može poslužiti funkcija `LENGTH`:

```
SELECT * FROM studenti WHERE LENGTH(komentar) < 20;
```

Od ostalih funkcija stringova, iako postoje i druge, fokusirat ćemo se na `REVERSE`, `TRIM`, `LTRIM`, `RTRIM`, `UPPER`, `LOWER`, `CONCAT`.

Naziv	Primjer	Izlaz	Opis
`LENGTH`	`LENGTH('  Bakir  ')`	9	Vraća dužinu stringa
`REVERSE`	`REVERSE('  Bakir  ')`	“ rikaB “	Obrće redoslijed karaktera u stringu
`TRIM`	`TRIM('  Bakir  ')`	"Bakir”	Eleminiše razmake na početku i kraju
`LTRIM`	`LTRIM('  Bakir  ')`	“Bakir “	Eleminiše razmake na početku
`RTRIM`	`RTRIM('  Bakir  ')`	“ Bakir”	Eleminiše razmake na kraju
`UPPER`	`UPPER('  Bakir  ')`	“ BAKIR “	Pretvara u velika slova
`LOWER`	`LOWER('  Bakir  ')`	“ bakir “	Pretvara u mala slova
`CONCAT`	`CONCAT('A','B','C')`	“ABC”	Sastavlja stringove

Funkcije je moguće i ugnijezditi, stoga je `UPPER(TRIM(ime))` potpuno validan izraz i rezultirat će trimovanim imenom sa velikim slovima.

Funkcije nad brojevima

Pored arithmetičkih, postoje i druge funkcije koje se mogu primjeniti nad brojevima. Neke od njih su: `POW`, `SIN`, `COS`, `ROUND`.

Naziv	Primjer	Izlaz	Opis
`POW`	`POW(2, 0)`	1	Stepenovanje, $2^0 = 1$
`SIN`	`SIN(0.5*3.14)`	0.99	Matematski sinus nad radijanima
`COS`	`COS(0)`	1	Matematski kosinus nad radijanima
`ROUND`	`ROUND(135.375)`	135.38	Zaokruživanje na dvije decimale

Npr. ako je potrebno zaokružiti prosječnu ocjenu iz tabele filmovi, možemo reći:

```
SELECT ROUND(AVG(ocjena), 2) FROM filmovi;
```

Sve ove funkcije se mogu koristiti i u `WHERE` klauzuli prilikom svih komandi gdje je ista dostupna.

Funkcije nad datumima

Kada bi smo htjeli dodati jedan dan na datum, ako bi vrijednost nekog datuma bila npr. `'2020-01-31'` i ako bi smo htjeli dodati jedan dan, ukoliko se napiše `'2020-01-31'+1` za rezultat se dobije `'20200132'` što je netačno jer taj datum ne postoji.

Za ovu svrhu postoje funkcije manipulacije datumima i neke od njih koje će biti pokazane su: `CURDATE`, `NOW`, `DAY`, `MONTH`, `YEAR`, `DAYOFWEEK`, `DAYOFMONTH`, `DAYOFYEAR`, `WEEKOFYEAR`, `DATE_ADD`, `DATE_SUB`.

Naziv	Primjer	Izlaz	Opis
`CURDATE`	`CURDATE()`	`'2022-02-03'`	Trenutni datum
`NOW`	`NOW()`	`'2022-02-03 10:46:41'`	Trenutni datum i vrijeme
`DAY`	`DAY('2022-03-22')`	22	Dan iz datuma (1-31)
`MONTH`	`MONTH('2022-03-22')`	3	Mjesec iz datuma (1-12)
`YEAR`	`YEAR('2022-03-22')`	2022	Godina iz datuma
`DAYOFWEEK`	`DAYOFWEEK('2022-03-22')`	3	Dan u sedmici (1-7) 1 = Nedjelja
`DAYOFMONTH`	`DAYOFMONTH('2022-03-22')`	22	Dan u mjesecu Isto što i `DAY()`
`DAYOFYEAR`	`DAYOFYEAR('2022-03-22')`	81	Dan u godini
`WEEKOFYEAR`	`WEEKOFYEAR('2022-03-22')`	12	Sedmica u godini

`DATE_ADD`
`DATE_SUB`
	`DATE_ADD("2017-06-15 09:34:21", INTERVAL 10 DAY);`
	`DATE_SUB("2017-06-15", INTERVAL 10 DAY);`
		`'2017-06-25 09:34:21'`
		
		`'2017-06-05'`
			Sabiranje i oduzimanje datuma kao i datum-vrijeme tipova

Posebno su korisne funkcije o datumima koje vraćaju trenutni datum, tako možemo praviti upite tipa: pronaći sve artikle kojima rok nije istekao:

```
SELECT * FROM artikli WHERE rok < CURDATE()
```

Ili

```
SELECT * FROM artikli WHERE rok < NOW()
```

Ili npr. broj artikala proizvedenih po mjesecu u 2020. godini:

```
SELECT MONTH(proizveden) AS mjesec_proizvodnje, COUNT(*) FROM artikli
WHERE YEAR(proizveden)=2020
GROUP BY mjesec_proizvodnje;
```

Vježba 2-7

Priprema:

1. Unijeti testnu bazu podataka

Napraviti upite za:

1. Pronaći sve studente, ime i prezime kao jedna kolona, i njegove ocjene uvećane za 5 tako da se 1 prevodi u 6 a 5 u 10.
2. Pronaći najmanji mjesec u kojem je data ocjena.
3. Pronaći sve studente koji su se upisali zadnjih 5 godina.
4. Pronaći prosječnu ocjenu svih martova zadnjih 5 godina.
5. Pronaći prosječnu ocjenu zadnjih 5 godina za svaki mjesec.
6. Pronaći prosječnu ocjenu za prvih 6 mjeseci svih godina.
7. Pronaći sve studente koji imaju istu dužinu imena.
8. Pronaći prosječnu ocjenu po svim studentima koji imaju istu dužinu imena.
9. Pronaći prosječnu dužinu naziva grada studenta zaokruženu na dvije cifre.
10. Pronaći prosječnu ocjenu studenata po predmetu, ali ime i prezime vratiti kao jednu kolonu.
11. Pronaći prosječnu ocjenu u prvih 10 dana svih mjeseci u 2017. godini.
12. Pronaći ime i prezime svih studenata i godine starosti.
13. Pronaći prosječnu starost studenata u svakom gradu.
14. Među studentima čije je ime kraće od naziva predmeta na kojem imaju ocijenu, pronaći iz kojeg grada dolaze studenti koji imaju najviše ocjena

Ako je data baza podataka za web foto galeriju:

Korisnici (id `int autoinc`, email `varchar`(100), sifra `text`,
tip `varchar`(20) default='korisnik', datum_reg `datetime`)
Foto (id `int autoinc`, naziv `text`, putanja `text`, datum `datetime`, korisnik_id `int`)
Albumi (id `int autoinc`, naziv `text`, datum `datetime`, korisnik_id `int`)
Albumi_Foto (id `int autoinc`, foto_id `int`, album_id `int`)
Komentari (id `int autoinc`, datum `datetime`, tekst `text`, foto_id `int`,
autor_korisnik_id `int`)

Za navedenu bazu:

15. Pronaći broj albuma u kojima se nalaze komentarisane slike.
16. Pronaći najstariji datum komentarisanja za svaki tip korisnika
17. Pronaći prosječan broj komentara na slikama za svakog korisnika.
18. Pronaći broj slika kreiranih na isti dan u sedmici kao i albumi u kojem se nalaze.

Test 2 Primjer

Ime i prezime: _____
Grupa ?

1. Šta je veza više-na-više i kada se koristi?

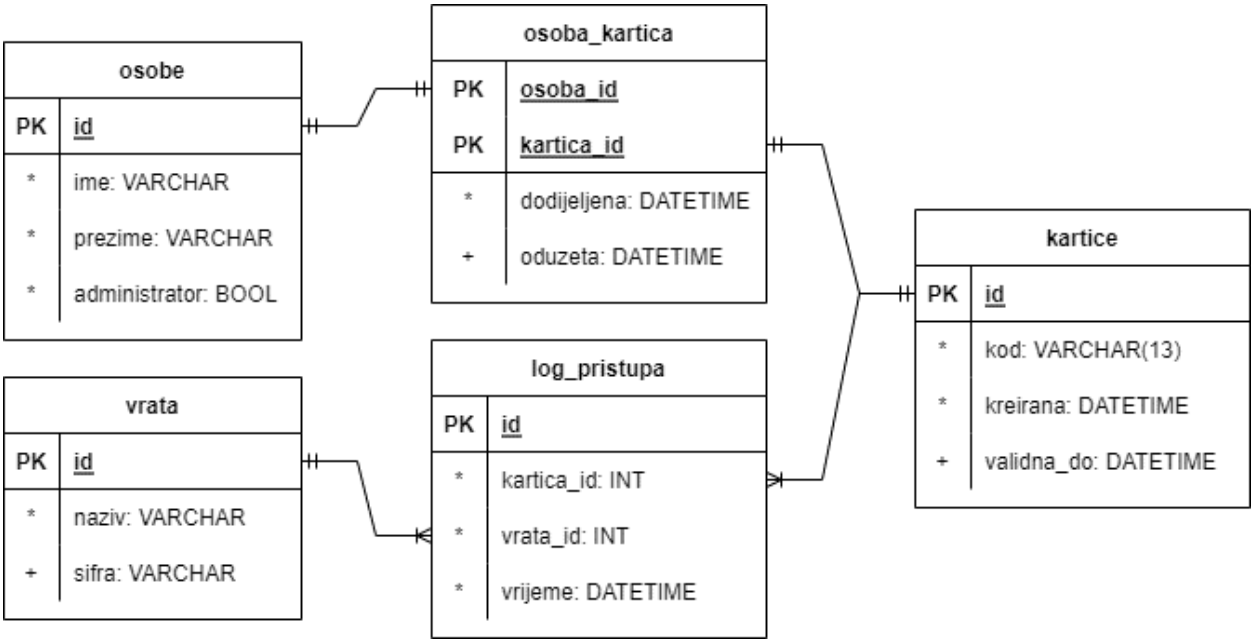
2. Koja je razlika između implicitnog i `INNER JOIN`-a?

3. Dizajnirati bazu podataka za umjetničku galeriju. Galerija posjeduje umjetnine (slike, skulpture i ostale radove), ali isto tako može posuditi umjetnine od druge galerije. Novac zarađuje od prodaje običnih i VIP ulaznica za svaku izložbu koju napravi, tako da je potrebno ovo evidentirati. Izložbe mogu biti tematske, te je to potrebno evidentirati, a isto tako i koje su umjetnine izložene na izložbi. Potrebno je omogućiti prodaju ulaznica preko web stranice, samo za registrovane korisnike.

4. Napisati upite za bazu podataka iz prethodnog pitanja:
    a. Pronaći broj izloženih umjetnina po svakoj tematici
    b. Pronaći zaradu od prodaje preko interneta za svaki tip ulaznice tokom 2017. godine
    c. Pronaći zaradu za svaku vrstu umjetnine izloženih u aprilu 2018. godine
    d. Pronaći najstarijeg registrovanog korisnika na svakoj izložbi
    e. Unijeti novog korisnika
    f. Pronaći zaradu od VIP ulaznica za svaki mjesec u kojem je održana izložba
    g. Pronaći broj prodatih ulaznica za svaku umjetninu
    h. Pronaći zaradu prodatih ulaznica za svaku umjetninu
    i. Pronaći broj prodatih ulaznica web korisnicima
    j. Pronaći prosječnu starost umjetnina koje galerija posjeduje

Zadaća 2-3

Data je šema baze podataka na slici:



Potrebno je:

1. Opisati svrhu, namjenu i mogućnosti ove baze podataka.

2. Napisati upite za:
   a. Pronalazak potpunog loga o tome ko je kada pristupio kojim vratima, i koju je karticu korisitio
   b. Broj pristupa svim vratima u tekućem mjesecu.

3. Opisati šta rade sljedeći upiti:
   a.
```
SELECT k.kod, COUNT(*), YEAR(v.vrijeme) AS godpristupa
FROM kartice k, log_pristupa lp
WHERE lp.kartica_id = k.id
GROUP BY godpristupa;
```

   b.
```
SELECT DISTINCT o.ime
FROM osobe o, osoba_kartica ok
WHERE o.id = ok.osoba_id
   AND oduzeta IS NOT NULL
   AND oduzeta < NOW();
```

2-8 Napredne teme

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 za koje je broj ocjena veći od 1 sortirano po broju ocjena:

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

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 izvodi SET saldo=saldo-100 WHERE user='Alisa';
UPDATE izvodi 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 izvodi SET saldo=saldo-100 WHERE korisnik='Alisa';
      UPDATE izvodi SET saldo=saldo+100 WHERE korisnik='Bakir';
COMMIT;
```

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 transakcije, ručno poništiti. Npr.:

```
START TRANSACTION;
      UPDATE izvodi SET saldo=saldo-100 WHERE korisnik='Alisa';
      UPDATE izvodi 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, ručno 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 Eminin iznos za 300

U ovom primjeru došla su dva zahtjeva za prenos sredstava sa Alisinog računa na Bakirov i Eminin račun. 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 100KM 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 100KM? 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 klauzulom `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;
```

Pogledi

Pogledi su koncept u bazama podataka koji omogućavaju tretiranje rezultata nekog `SELECT` upita kojem je izlaz tabela, upravo kao tabelu. Drugim riječima, imenujemo upit. Nad ovom tabelom mogu se raditi sve normalne `SELECT` operacije. Npr.:

```
SELECT * FROM studenti WHERE id BETWEEN 30 AND 300 AND grad='Sarajevo' AND
komentar IS NOT NULL AND prezime LIKE 'D%';
```

S obzirom da je ovaj upit komplikovan nije poželjno smišljati ga svaki put kada su potrebni ovi rezultati, stoga možemo kreirati *pogled*:

```
CREATE VIEW pretraga SELECT * FROM studenti WHERE id BETWEEN 30 AND 300 AND
grad='Sarajevo' AND komentar IS NOT NULL AND prezime LIKE 'D%';
```

Ovaj upit se može koristiti kao bilo koja druga imenovana tabela:

```
SELECT * FROM pretraga;
SELECT * FROM pretraga WHERE id=33;
```

Vježba 2-8

Priprema:

1. Otvoriti dva komandna prompta i pozicionirati se na testnu bazu u oba terminala
2. Kreirati tabelu `racuni(id INT AUTO_INC, naziv TEXT, iznos INT)` i unijeti iznose za Alisu 100, Bakira 100 i Alinu 300 KM.

Vježba:

1. Simulirati proces isplate novca sa Alininog računa na Alisin.
2. Simulirati proces isplate novca sa Alisinog računa na Alinin kroz transakciju.
3. Simulirati neuspjeli proces isplate novca sa Bakirovog računa na Alinin.
4. Simulirati proces isplate novca uz prethodnu provjeru dostupnog stanja bez zaključavanja redova.
5. Simulirati proces isplate novca uz prethodnu provjeru dostupnog stanja uz zaključavanje redova.
6. Kreirati okidače na `INSERT` i `DELETE`
7. Kreirati pogled

Predavanje 2-9

Objektne baze podataka

S obzirom da se interakcija sa bazama podataka najčešće vrši koristeći programerski interfejs, tako da krajnji korisnik ne koristi bazu podataka već koristi neki program koji preko programskog interfejsa spašava i vraća podatke u bazu. Kako bi se ova interakcija izvršila, programi koriste SQL upite kako bi unijeli i vratili podatke iz baze kao i imali i ostalu interakciju.

Pisanje SQL upita, te parsiranje rezultata koristeći programski jezik može biti vrlo komplikovano. Još jedan nedostatak relacionih baza podataka je potreba za modeliranjem komplikovanih veza između tabela. Kako je tehnologija napredovala, prostor za pohranu podataka se povećao mnogostruko u odnosu na period kada su relacione baze podataka izumljene.

Prilikom izrade nekih programa, postoji potreba da se sačuvaju objekti koje program koristi. Za ovu svrhu su napravljene objektne baze podataka. One se još nazivaju dokumentne baze podataka.

U objektnim bazama podataka, podaci su spašeni kao upravo isti objekti kao u programskim jezicima i potreba za strukturom je smanjena ili uklonjena. Tako da objektne baze nemaju tabele, već imaju objekte i kolekcije objekata. Jedna vrsta objektne ili dokumentne baze je MongoDB.

MongoDB

MongoDB[5] ili jednostavno Mongo, je objektna baza podataka koja objekte spašava u kolekcije. Objekti imaju atribute i njihove vrijednosti. Vrijednosti atributa objekta mogu ponovo biti objekti.

Primjer jednog objekta u Mongu je:

```
{
   "_id": ObjectId("507f191e810c19729de860ea"),
   "ime": "Alisa",
   "prezime": "Alisovic",
   "razred": 3,
   "upis": ISODate("2021-12-19T06:01:17.171Z"),
   "predmeti": ["Matematika", "Fizika", "Hemija"],
   "prosjek": 3.14,
   "izborni_predmet": null,
   "izgled": { oci: "plave", visina: 160, boja_kose: "crna" }
}
```

[5] https://www.mongodb.com/

Osnovna karakteristika objekata je da imaju naziv atributa, koji je string, i vrijednost atributa koji najčešće mogu biti brojevi, stringovi, datumi, null-vrijednosti, nizovi, objekti, koji će biti u fokusu ali mogu biti i drugi koji nisu spomenuti ovdje ali jesu u dokumentaciji ove baze podataka.

Interakcija sa MongoDB se može vršiti kroz komandnu liniju, programski, ili koristeći Robo 3T[6] aplikaciju koja omogućava pisanje komandi. U primjerima će biti korišten Robo 3T.

Kada se koristi Robo 3T, ne koristi se mongo shell kao interfejs, već MongoDB-ov upitni jezik. Osnovni objekat nad kojim se pozivaju metode se zove db. Tako npr.:

```
db.adminCommand( { listDatabases: 1 } );
```

će vratiti listu baza u našoj instalaciji baze podataka. `adminCommand()` je metoda za cijeli spektar administratorskih operacija za bau podataka. Osnovne interakcije poput kreiranja baze i kolekcija će se obavljati kroz Robo 3T GUI, tako da će glavni fokus biti na upitni jezik.

Tako npr. da dobavimo sve studente iz kolekcije, možemo napisati:

```
db.getCollection('studenti').find({});
db.studenti.find({});
```

U prvom slučaju koristimo metodu `getCollection()`, a u drugom jednostavno koristimo atribut nad objektom `db`. Ova dva pristupa su potpuno jednaka, i drugi je tu radi lakšeg korištenja.

Kolekcija je također objekat, nad kojim se može primjeniti niz funkcija ili metoda. Umjesto komandi i klauzula kao u SQL jeziku ovdje se pozivaju funkcije da obave neku funkciju. Tako za osnovne funkcije imamo `find insert` i `remove`. Tako npr. za ubacivanje studenta u kolekciju, koristi se:

```
db.studenti.insert({ ime: 'Denis', prezime: 'Prezime' });
```

Da bi vidjeli da je objekat zaista unesen u kolekciju, možemo koristiti metodu `find()`. Za pretragu se koristi objekat u kojem se naznače polja i njihove vrijednosti, a kao rezultat se dobiju svi objekti koji imaju ta polja koja imaju tu vrijednost.

```
db.studenti.find({ ime: 'Denis' });
```

Rezultat:

```
{ "_id" : ObjectId("6207920285b9ad817857d8d1"),
  "ime" : "Ime",
  "prezime" : "Prezime" }
```

[6] https://robomongo.org/

ObjectId

Primjećuje se da objekat ima dodatno polje u odnosu na objekat koji je unesen u bazu podataka. To polje je `"_id"` i predstavlja **primarni ključ**. Tip ovog ključa je ObjectId. U SQL bazama imali smo id koji je bio `INT` i uvećavao se za svaki novi unos. Međutim, u MongoDB, ObjectId je kombinacija nasumično odabranih brojeva i datuma kada je unos napravljen.

ObjectId se sastoji od 12 bajti, gdje prvih 4 označavaju broj sekundi od UNIX epohe[7] naredna 3 bajta označavaju ID mašine, a naredna 2 ID procesa. Ova kombinacija omogućava dovoljnu nasumičnost kako se ne bi desilo preklapanje. Dodatni plus je što nam prva 4 bajta daju vrijeme kada je objekat kreiran. Za vođenje evidencije kada je objekat modifikovan će biti potrebno koristiti neko drugi atribut.

Struktura

Kontrastno SQL bazama, objektne baze nemaju fiksiranu strukturu. To znači da jedan objekat može imati jednu listu atributa, a drugi objekat drugu listu atributa u istoj kolekciji. Sa organizacijske tačke gledišta imati objekte sa potpuno različitim atributima i nema smisla, ali imati objekte sa istom baznom skupinom atributa, gdje neki objekti mogu imati dodatne atribute različite, može biti vrlo korisno. Tako npr. možemo dodati objekat sa drugačijom listom atributa:

```
db.studenti.insert({ ime: 'Drugo ime', nadimak: 'dj drugi' });

db.studenti.find({});
{
    "_id" : ObjectId("6207920285b9ad817857d8d1"),
    "ime" : "Denis",
    "prezime" : "Prezime"
},
{
    "_id" : ObjectId("6207a14885b9ad817857d8d2"),
    "ime" : "Drugo ime",
    "nadimak" : "dj drugi"
}
```

Imajući više objekta ali sa različitim strukturama, eleminiše se potreba za organizacijom tabela i dizajniranjem baze tako detaljno kao što je rađeno u SQL bazama. Za brisanje objekta možemo koristiti metodu `remove()`:

```
db.studenti.remove({ "_id" : ObjectId("6207a14885b9ad817857d8d2") });
```

Nedostatak objektnih baza je loša podrška za transakcije, te iz tog razloga, prihvaćeno je dupliranje podataka kako bi se izbjegla potreba za transakcijama.

[7] Početak UNIX epohe 00:00:00 UTC on 1 January 1970

Vježba 2-9

Priprema:

1. Preuzeti i instalirati mongo bazu podataka
    a. https://www.mongodb.com/try/download/community?tck=docs_server
2. Preuzeti i instalirati Robo 3T
    a. https://download.studio3t.com/robomongo/windows/robo3t-1.4.4-windows-x86_64-e6ac9ec5.exe
3. Kreirati novu, testnu bazu podataka, kroz Robo 3T. Desni klik => Create Database
4. Kreirati novu kolekciju, studenti, kroz Robo 3T. Desni klik na bazu => Create Collection

Napisati upite za:

1. Unijeti nekoliko studenata u kolekciju studenti sa različitom strukturom, tj. atributima
2. Koristiti metodu find za pronalazak objekata
3. Koristiti metodu remove za brisanje objekata

2-10 Mongo upiti

Tipovi podataka

Kao i u SQL bazama podataka, tako i MongoDB atributi u objektima imaju tipove. Neki od najčešćih tipova su: null, ObjectId, number, double, string, array, binData, bool, date, int, long. U MongoDB dokumentaciji može se pronaći paralelno poređenje sa SQL jezikom i naredbama kao i tipovima u MongoDB: https://docs.mongodb.com/manual/reference/sql-comparison/

Primjer unosa objekta sa različitim tipovima podataka:

```
db.studenti.insert({
   ime: 'Alisa',
   prezime: 'Alisovic',
   datum_rodj: ISODate('2005-03-01'),
   osobine: {
      godine: 15,
      visina: 160,
      tezina: NumberInt(55)
   },
   koeficijent: Number(3.37),
   grupa: 'D'
});
```

Ovo je primjer komplikovanijeg objekta. Za pretraživanje objekta spomenuta je `find({})` metoda. Tako se može pretraživati na nekoliko načina.

```
1. db.studenti.find({ ime: 'Alisa' });
2. db.studenti.find({ prezime: 'Alisovic' });
3. db.studenti.find({ ime: 'Alisa', prezime: 'Alisovic' });
4. db.studenti.find({ datum_rodj: ISODate('2005-03-01 00:00:00.000Z') });
5. db.studenti.find({ 'osobine.godine': 15 });
```

Ovdje se mogu vidjeti primjeri pretraživanja na različite načine, uključujući i pretraživanje po atributima pod objekata. Za izmjenu podataka, koristi se update() metoda:

```
db.studenti.update({ ime: 'Denis' },
   { $set:
      { prezime: 'Izmjena', godine: NumberInt(20) }
   });
```

Na prvom mjestu se naznači objekat za pretraživanje, a na drugom mjestu objekat sa specijalnim operaterom `$set`, i objektom sa atributima koje želimo primjeniti. Efekat ove komande je da

nepoklapajući atributi u orginalnom objektu neće biti uklonjeni već će biti izmjenjeni samo podudarajući atributi ili dodani novi. Treba napomenuti da se `_id` atribut ne može mijenjati.

Recimo da želimo dodati niz objekata za studenta:

```
db.studenti.update({ ime: 'Denis' }, { $set: {
   hobiji: [
     { naziv: 'igre', godine: 3 },
     { naziv: 'ples', godine: 2 },
     { naziv: 'teretana', godine: 4 },
     { naziv: 'fudbal', godine: 8 }
   ]
});
```

Dodan je novi niz, liste hobija za sve studente kojima je ime `Danis`. Na isti način je mogao biti potražen dokument sa specifičnim `_id`. Da se pretraže studenti koji za prvi hobij imaju igre:

```
db.studenti.find({
    'hobiji.0.naziv': 'igre'
});
```

Operatori `$lt, $gt, $lte, $gte, $eq`

Ovi operatori omogućavaju pretraživanje poput manje (`$lt`), veće (`$gt`), manje ili jednako (`$lte`), veće ili jednako (`$gte`) i jednako (`$eq`). Ovi operatori se mogu primjeniti na sve tipove podataka.

```
db.studenti.find({
    godine: { $gte: 20 }
});
```

Operator `$in` i `$all`

Ukoliko objekat ima niz brojeva poput:

```
db.studenti.insert({
   ime: 'Kenan', prezime: 'Kenanovic',
   koeficijenti: [ 3, 8, 10, 11, 18, 21, 27 ]
});
```

I želimo da pretražimo sve objekte koji u atributu koeficijent imaju brojeve 11 ili 13, u te svrhe se može koristiti operator `$in`.

```
db.studenti.find({ koeficijenti: { $in: [11, 13] } });
```

Za razliku operatora `$in`, operator `$all` zahtjeva da atribut ima sve vrijednosti navedene u upitu. Npr. da smo koristili `$all` u prethodnom upitu, ne bi pronašao Kenana jer među koeficijentima ima samo 11 ali nema 13.

```
db.studenti.find({ koeficijenti: { $all: [11, 13] } });
```

Jedinstvene vrijednosti

Ukoliko želimo pronaći sva imena u kolekciji studenti, možemo koristiti `distinct()` funkciju, kojoj se proslijeđuje naziv atributa za kojeg je potrebno pronaći jedinstvene vrijednosti.

```
db.studenti.distinct('ime');
```

Ukoliko je potrebno tražiti jedinstvene vrijednosti na podskupu podataka, može se dodati upit kojim se sužava skup unutar kojeg se traži jedinstvena vrijednost.

```
db.studenti.distinct('ime', { razred: 3 }});
```

Tako da će ovaj upit pronaći jedintvena imena ali samo za studente za koje je razred=3.

Operatori `$and` i `$or`

Ovo su logički operatori koji mogu pomoći definisanje preciznijih zahtjeva za pretraživanjem. Npr. svi studenti koji su prvi ili treći razred:

```
db.studenti.find({ $or: [{ razred: 1 }, { razred: 3 }] });
```

Dakle, za ove operatore potrebno je definisati niz objekata koji predstavljaju uslove nad kojim se ovaj logički operator definiše. `$and` i `$or` se mogu kombinovati i sa ostalima pa npr traženje studenata koji imaju između 18 i 25 bi izgledalo ovako:

```
db.studenti.find({
      $and: [{ godine: { $gte: 18 } }, { godine: { $lt: 25 } } ]
});
```

Moguće je i kombinovati sve ove operatore. Npr. svi student koji su prvi ili treći razred a upisani su na matematiku i fiziku:

```
db.studenti.find({
      $and: [
                  { predmeti: { $all: ['Matematika', 'Fizika'] } },
                  { $or: [{ razred: 1 }, { razred: 3 }] }
      ]
});
```

Vježba 2-10

2-11 Ostale Mongo funkcije i operatori

Sortiranje

Da bi se sortirao objekat koristi se `sort()` funkcija nad objektima vraćenim koristeći `find()` metodu. Npr.:

```
db.studenti.find({}).sort({ ime: 1, prezime: -1});
```

Ovaj upit će vratiti sve studente sortirano abecedno po imenu i suprotno abecedno po prezimenu.

S obzirom da _id polje u sebi sadrži datum kreiranja dokumenta, interesantna posljedica ovoga je da se može koristiti prilikom sortiranja, tako da se svi studenti, sortirani od zadnje kreiranog mogu pronaći koristeći:

```
db.studenti.find({}).sort({ _id: -1 });
```

Ograničavanje i paginacija

Kako bi se pronašla određena stranica iz izlaznog skupa, mogu se koristiti metode `skip()` i `limit()`. Tako npr. šesta stranica studenata, ako stranica ima 10 elemenata, kombinovano sa sortiranjem:

```
db.studenti.find({}).sort({ ime: 1 }).skip(5*10).limit(10);
```

Tekstualna pretraga

Pošto je moguće imati mnogo tekstualnih polja, potrebno je moći ih pretraživati. Vidjeli smo da možemo jednostavno reći `find({ ime: 'Alisa' })` da pronađemo po imenu, ali ako želimo pretraživati tekstualne podatke po djelimičnom podudaranju. Tako npr. da nađemo sva imena koja počinju sa A, napisat ćemo:

```
db.studenti.find({ ime: /^A[a-z]+/i })
```

Ova komanda je zapravo dio Regex jezika za tekstualnu pretragu, kojim se zapravo definišu šabloni. Pravila ovog jezika su da se šablon po kojem se podudara tekst, definiše unutar dvije kose crte (`/`) nakon koje idu opcije. Tako da ovaj šablon ima sljedeće značenje:

`^` – kapica ili caret simbol, definiše početak teksta
`A` – govori da se očekuje slovo A i s obzirom na prethodni simbol, prvo slovo mora biti A
`[a-z]` – govori da se očekuje bilo koje slovo od a do z
`+` – symbol koji označava da se prethodni iskaz može podudarati jednom ili više puta

Zbog toga što se tekst „Alice“ ili „Alisa“ zadovoljavaju ovaj šablon, onda će biti pronađeni svi objekti gdje ime ima podudaranje sa ovim šablonom.
Neke od ostalih primjera će biti izlistani ali s obzirom da regex može biti komplikovan postoje alati sa kojima se može testirati podudaranje[8]. Npr.:

```
/^[0-9]{3} [0-9]{3}-[0-9]{3,4}$/
```

Šablon kojim se na početku (^) moraju naći tri ({3}) broja ([0-9]) onda mora biti razmak, onda moraju biti tri broja pa crta pa tri ili četri ({3,4}) broja opet i onda je kraj stringa. Ovo je korisno za podudaranje brojeva telefona, npr. 061 123-456 ili 062 123-4567, jer će svi netačno upisani brojevi pasti na testiranju šablona.

```
/^\D+ic$/
```

Šablon kojim se podudaraju svi stringovi koji od početka imaju 1 ili više (+) slova (\D), i završavaju ($) sa ic. Postoje mnogi drugi simboli, jer su ovo samo primjeri te se ovim jezikom mogu napraviti raznorazni šabloni za podudaranje stringova.

Projekcija

S obzirom da objekti mogu biti različite veličine, te imati nizove u svojim atributima, različitih dimenzija, pronalaženje objekata koji mogu biti veliki može predstavljati problem. Postoje situacije u kojima nisu neophodne informacije svakog atributa. Npr. ako tražimo studenta, a student imaju listu hobija, i ima objekat *osobine* sa dugom listom osobina, međutim nas ne interesuju ni osobine ni hobiji već samo *ime*, *prezime*, i *datum rodjenja*, onda možemo iskoristiti projekcije da vratimo samo određenu listu atributa.

```
db.studenti.find({}, { ime: 1, prezime: 1, datum_rodj: 1 });
```

Projekcija se defineše kao drugi argument u `find()` funkciji. Taj objekat ima popis atributa koje želimo da nam se vrate i 1 kao vrijednost ili 0 ako ne želimo te atribute.

Ukoliko je potrebno vratiti sve postojeće atribute, osim jednog onda možemo napisati:

```
db.studenti.find({}, { hobiji: 0 });
```

U ovom slučaju MongoDB će vratiti sve studente i sve njihove atribute osim atributa hobiji. Na ovaj način se mogu optimizovati upiti kako se nepotrebni podaci ne bi prenosili preko mreže i tako bi se ubrzao prenos i smanjili troškovi prenosa podataka koji se obično naplaćuju.

Objektne baze podataka (u slučaju MongoDB dokumentna) faju fleksibilnu mogućnost i fleksibilnu strukturu organizacije podataka. Nedostatak je što nemaju jaku podršku za transakcije te je zbog toga potrebno pažljivo odmjeriti potrebe sistema za koji se projektuje i bira baza podataka.

[8] https://regex101.com/#javascript

Indeksiranje

Kao i sve ostale baze podataka, i MongoDB ima indeksiranje koje omogućava brže pretraživanje po jednom ili više polja. Za razliku od MySQL baze, prilikom kreiranja indeksa, definiše se hoće li indeks biti uzlazni ili silazni. Da se pogledaju postojeći indeksi, koristi se getIndexes().

```
db.studenti.getIndexes();

[
    {
        "v" : 2,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_"
    }
]
```

Po defaultu, svaka kolekcija je indeksirana po _id polju. Međutim, moguće je napraviti indeks po bilo kojem drugom polju koristeći metodu `createIndex()`.

```
db.studenti.createIndex({ ime: 1 }, { name: 'Indeks po imenu' });
```

Na ovaj način smo kreirali indeksiranje po imenu i dali mu naziv. Vrijednost 1 za ime pri kreiranju indeksa znači da će biti kreiran uzlazno sortirani indeks, a ako je -1 taj indeks će biti silazno sortirann. Na isti način možemo kreirati indeks koristeći više atributa.

```
db.studenti.createIndex(
      { ime: 1, prezime: -1 },
      { name: 'Indeks po nazivu' }
);
```

Kako smo kreirali indeks, tako možemo i obrisati indeks nad kolekcijom, identifikovajući ga koristeći njegovo ime ili specifikaciju:

```
db.studenti.dropIndex({ ime: 1});
```

ili

```
db.studenti.dropIndex('Indeks po imenu');
```

Vježba 2-11

2-12 Napredne teme baza podataka

Sve situacije do sada su obuhvaćale bazu podataka ograničene veličine. Tj. nisu razmatrani slučajevi gdje baza podataka postane dovoljno velika da ne može stati na prostor dostupan u jednom računaru. U ovim slučajevima potrebno je podijeliti bazu podataka na više računara. Zbog ovih potreba postoje *data* centri, koji skladište podatke na hiljadama računara.

Npr. WhatsApp razmjenjuje 65 milijardi poruka dnevno, Facebook generiše 4000 TB podataka dnevno, služeći 2.3 milijarde korisnika[9]. Procjenjuje se da Facebook skladišti 300.000 TB podataka ili 300 PetaB podataka. Svrha skladištenja ove količine podataka je bolji uvid u načine na koji korisnici koriste neki sistem, kako bi se od toga izvukli korisni i profitabilni podaci.

Baze podataka koje skladište podatke na više računara, ne nužno na istoj fizičkoj lokaciji, zovu se distribuirane baze podataka. One su jedna vrsta distribuiranih sistema koje su posebno polje izučavanja u računarskim naukama i informatici. Osim zahtjeva za skladištenje, potrebno je zadovoljiti zahtjeve za pretragu svih tih podataka.

Distribuiranje sistema na više računara donosi niz izazova, kao što su potencijalni nestanci struje, nefunkcionisanje mreže kojom su računari povezani, problemi u radi mrežnih uređaja, problemi prestanka rada diskova, ili drugih komponenti računara. Npr. jedan primjer crkavanja diskova je 0.14% na godišnjoj bazi[10] stoga ako centar ima oko 150.000 diskova to znači da će se pokvariti oko 210 diskova na godišnjem nivou, što znači da će trebati mijenjati disk skoro svaki dan.

Ovo su primjeri različitih potreba i različitih problema. Zbog dinamičnosti razvoja sistema, postojanje strukture podataka kakva postoji u SQL bazama, donosi niz potencijalnih problema radi organizacije modela podataka. Spašavanje svakog klika ili pokreta miša korisnika, kao i svih podataka desetina ili stotina miliona korisnika, donosi jedinstvene izazove za svaki sistem. Za ove i ostale svrhe, napravljene su druge vrste podataka kako bi se mogli spasiti podaci sa dinamičkom strukturom.

Poseban problem je pretraživanje i analiziranje svih ovih podataka. Npr. analiziranje nekog skupa podataka obično se odvija nad istom kolonom. Kao npr. dobavljanje prosječne ocjene svih studenata. U tom slučaju nisu potrebne ostale kolone, ali SQL baze podataka spašavaju redove na disk u jednom nizu, tako da bi se dobavili podaci jedne kolone iz svih redova, baza podataka i disk, moraju preći sve dostupne redove. Da bi se ovaj problem riješio napravljene su baze podataka koje spašavaju podatke sekvencijalno po koloni, tako da su svi redovi jedne kolone spašeni u nizu na disku, te je pretraživanje podataka brže i habaju disk manje.

To je jedan primjer NoSQL baze podataka, mada se pojam NoSQL odnosi za sve baze podataka koje nemaju fiksnu strukturu. Dati primjer analize podataka se zove *Analitičko Procesiranje* ili

[9] https://medium.com/@srank2000/how-facebook-handles-the-4-petabyte-of-data-generated-per-day-ab86877956f4#:~:text=Hive%20is%20Facebook's%20data%20warehouse,map%2Dreduce%20jobs%20per%20day.
[10] https://www.backblaze.com/blog/backblaze-hard-drive-stats-q2-2020/

OLAP (*Online Analytical Processing*). Nije ni strano da se podaci dupliraju u dvije različite baze podataka, koje su optimizovane za različite svrhe. SQL baze podataka, podržavaju tzv. *Transakcijsko Procesiranje* ili OLTP (*Online Transactional Processing*), tj. garantovanje procesiranje transakcija koje zadovoljavaju ACID pravila, tj. garanciju konsistentnosti podataka. Međutim zadovoljavanje ovih uslova, onemogućava bazu podataka da brzo procesira podatke, pogotovo ne brzo onoliko kada je procesiranje potrebno milionama korisnika.

Još jedan naziv za OLAP tj. podnaziv je i *Data Mining* ili kopanje informacija, jer postoje osobe zadužene za pronalazak novih znanja u moru podataka. Tako raznorazne kompanije mogu pronaći navike kupca kao i korisnicima omiljene stvari. Npr. je li vam se ikad desilo da potražite na Googl-u ili Ebay-u neki proizvod a da bi naknadno vidjeli reklame za isti proizvod na potpuno drugoj stranici.

Kako bi se svi ovi podaci spasili u distribuirane baze podataka, postoje razni pristupi rješavanju ovog problema. Svi ovi pristupi pokušavaju zadovoljiti CAP teoremu. CAP teorema govori da bilo koja distribuirana baza podataka može zadovoljiti dva od naredna tri uslova: Konsistentnost (Consistency), Dostupnost (Availability) i Toleranciju na praticionisanje (Partition tolerance). Drugim riječima, distribuirana baza podataka ne može zadovoljiti absolutne zahtjeve, stoga će pokušati zadovoljiti neke od njih. Konsistentnost ovih sistema ne mora biti na prvom mjestu ali će se konsistentnost eventualno pokušati zadovoljiti.

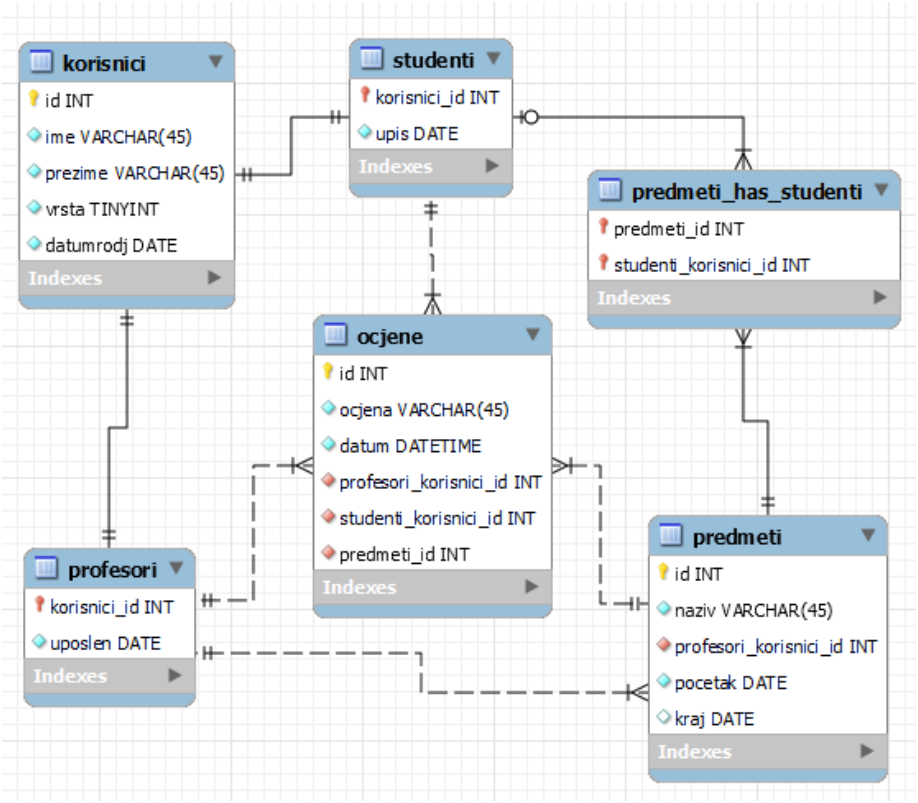
Jedan od načina korištenja distribuiranih baza podataka je *batch-processing* pristup. Ovim pristupom, vrši se procesiranje određene količine podataka, ili svih, tako što se pokrene program u zakazano vrijeme i obradi te podatke. Drugi, noviji, način je koristeći programa koji konstantno procesiraju beskonačni niz podataka, kako dolaze budu obrađeni i zove se *stream-processing*.

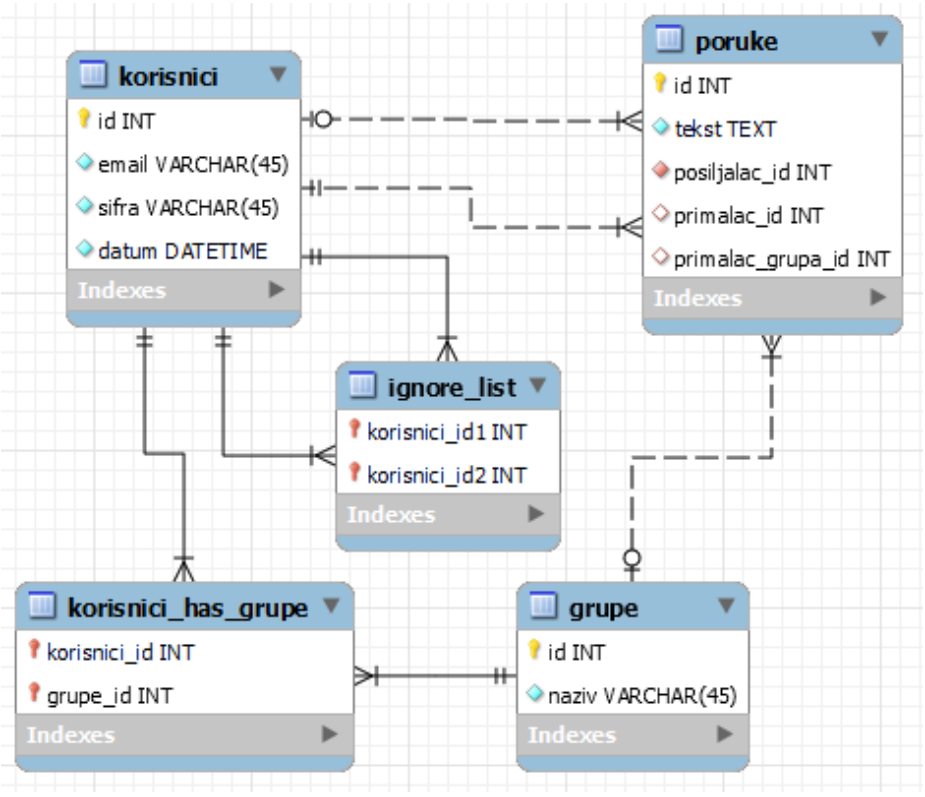
Osim ovih tipova baza podataka, postoje ostale specijalizovane baze podataka. Jedan primjer je baze podtaka za pretraživanje teksta i sličnosti teksta. Npr. potreba za pretragom artikala na oglasniku čak ni kada naziv ne podudara striktno, jer je možda neko napravio grešku u kucanju. Primjeri te vrste podataka su Apache Lucene i ElasticSearch. Još jedan primjer specijalizovane baze podataka su baze za Geografske Informacione Sisteme (GIS) gdje je potrebno brzo pretraživanje 2D ili 3D podataka tipa udaljenost od grada, ili raspon između GPS koordinata. Iako baze podataka poput MongoDB ili PostgreSQL imaju ove sposobnosti, za određene potrebe postoje specijalizovane baze podataka.

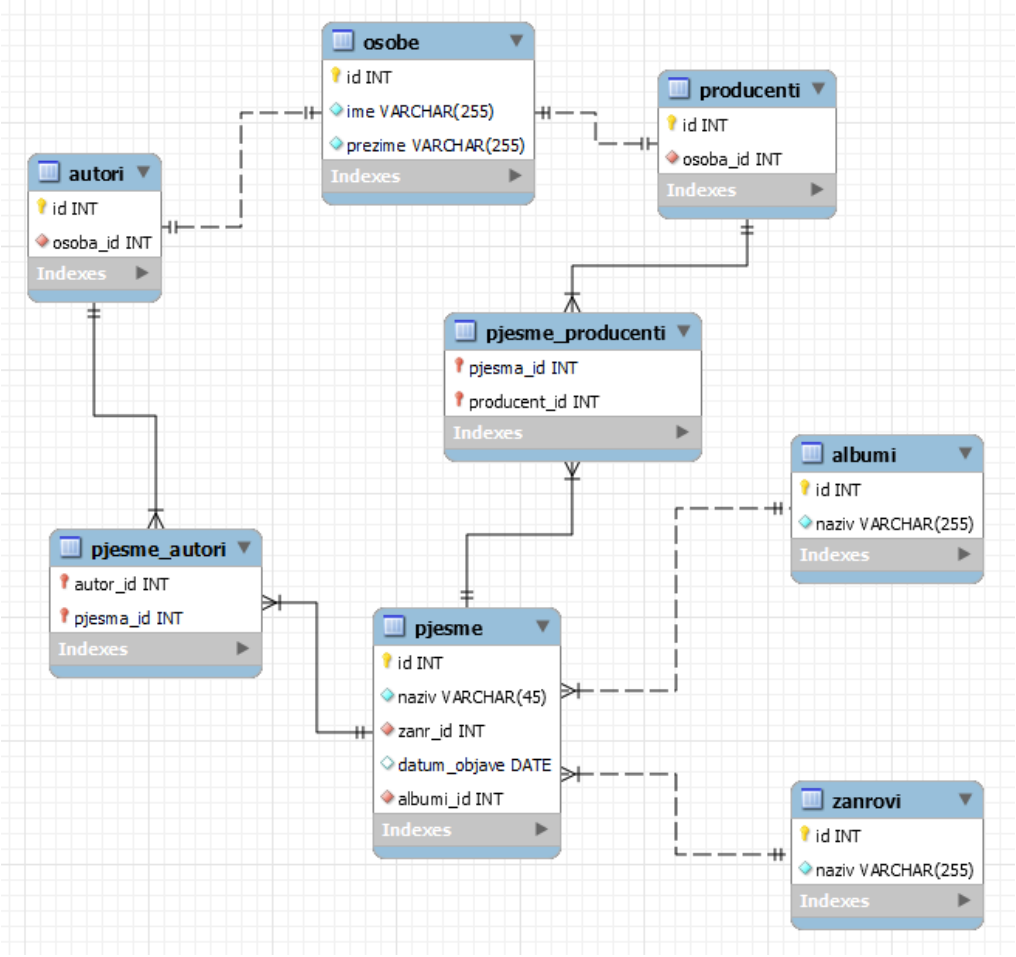
Postoji još mnogo tema za izučavanje baza podataka kao i sistema koji ih koriste poput: 2-phase-commit, replikacija, particionisanje, nivoi izolacije, skladišta podataka, key-value store, linearzibilnost, distribuirane transakcije itd, međutim ovdje se fokusiralo na osnovne fukcije SQL i objektnih baza podataka.

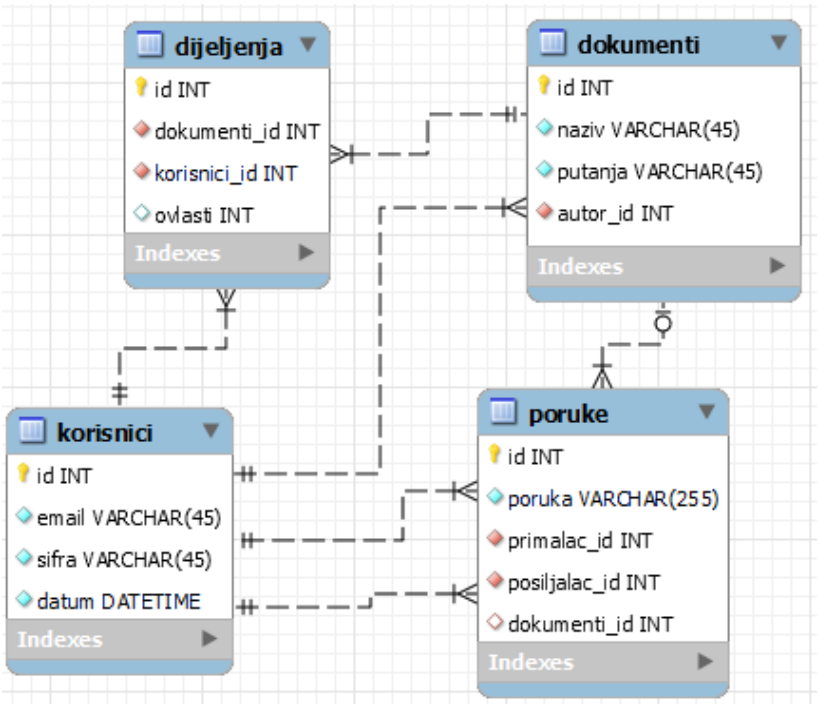
Za potrebe velikog broja jednostvnih sistema, ova dokumentacija može biti dovoljna podloga, a za sve ostale detalje tu su dokumentacije svake pojedinačne baze podataka koje detaljno opisuju sve dostupne funkcionalnosti.

Vježba 2-5 Rješenja






osobe
id INT
ime VARCHAR(255)
prezime VARCHAR(255)
Indexes
producenti
id INT
osoba_id INT
Indexes
autori
id INT
osoba_id INT
Indexes
pjesme_producenti
pjesma_id INT
producent_id INT
Indexes
albumi
id INT
naziv VARCHAR(255)
Indexes
pjesme_autori
autor_id INT
pjesma_id INT
Indexes
pjesme
id INT
naziv VARCHAR(45)
zanr_id INT
datum_objave DATE
albumi_id INT
Indexes
zanrovi
id INT
naziv VARCHAR(255)
Indexes


dijeljenja
id INT
dokumenti_id INT
korisnici_id INT
ovlasti INT
Indexes
dokumenti
id INT
naziv VARCHAR(45)
putanja VARCHAR(45)
autor_id INT
Indexes
korisnici
id INT
email VARCHAR(45)
sifra VARCHAR(45)
datum DATETIME
Indexes
poruke
id INT
poruka VARCHAR(255)
primalac_id INT
posiljalac_id INT
dokumenti_id INT
Indexes

Vježba 2-6 Rješenja

1. Ispisati sve studente imenom i prezimenom i njihove ocjene
```
SELECT s.ime, s.prezime, o.ocjena
FROM studenti s, ocjene o
WHERE s.id = o.student_id;
```

2. Ispisati sve studente i njihove ocjene, kao i predmete na kojima je ta ocjena dobijena sortirano po nazivu predmeta od Z do A
```
SELECT s.ime, s.prezime, o.ocjena, p.naziv
FROM studenti s, ocjene o, predmeti p
WHERE s.id = o.student_id AND o.predmet_id = p.id
ORDER BY p.naziv DESC;
```

3. Pronaći sve studente nazivom i njihove ocjene kao i profesore koji su ih dali i nazive predmeta na kojima je ta ocjena, sortirano po datumu
```
SELECT o.datum, s.ime, s.prezime, o.ocjena, p.naziv, pro.ime, pro.prezime
FROM studenti s, ocjene o, predmeti p, profesori pro
WHERE s.id = o.student_id AND o.predmet_id = p.id AND pro.id = o.profesor_id
ORDER BY o.datum;
```

4. Pronaći sve studenti i njihove ocjene, na predmetu “Baze podataka”
```
SELECT o.ocjena, s.ime, s.prezime
FROM studenti s, ocjene o, predmeti p
WHERE s.id = o.student_id AND o.predmet_id = p.id
   AND p.naziv = "Baze Podataka";
```

5. Pronaći nazive studenata i broj ocjena koji oni imaju
```
SELECT s.ime, s.prezime, COUNT(o.ocjena) br_ocjena
FROM studenti s, ocjene o
WHERE s.id = o.student_id
GROUP BY s.ime, s.prezime;
```

6. Pronaći nazive studenata i prosječnu ocjenu po predmetu
```
SELECT s.ime, s.prezime, AVG(o.ocjena) prosjek, p.naziv
FROM studenti s, ocjene o, predmeti p
WHERE s.id = o.student_id AND o.predmet_id = p.id
GROUP BY s.ime, s.prezime, p.naziv;
```

7. Pronaći nazive studenata i kada mu je koji profesor na kojem predmetu dao najstariju ocjenu
```
SELECT s.ime, s.prezime, MIN(o.datum) AS najstarija, p.naziv, o.profesor_id
FROM studenti s, ocjene o, predmeti p
WHERE s.id = o.student_id AND o.predmet_id = p.id
GROUP BY s.ime, s.prezime, o.profesor_id, p.naziv;
```

8. Pronaći predmete i datume najstarije ocjene
```
SELECT p.naziv, MIN(o.datum) AS najstarija
FROM ocjene o, predmeti p
WHERE o.predmet_id = p.id
GROUP BY p.naziv;
```

9. Pronaći prosječnu ocjenu po svakom predmetu
```
SELECT p.naziv, AVG(o.ocjena) AS prosjek
FROM ocjene o, predmeti p
WHERE o.predmet_id = p.id
GROUP BY p.naziv;
```

10. Pronaći prosječnu ocjenu na predmetima čiji se naziv završava sa “ika”
```
SELECT p.naziv, AVG(o.ocjena) AS prosjek
FROM ocjene o, predmeti p
WHERE o.predmet_id = p.id AND p.naziv LIKE '%ika'
GROUP BY p.naziv;
```

11. Pronaći najstariju i najnoviju ocjenu profesora
```
SELECT o.profesor_id, MIN(o.datum), MAX(o.datum)
FROM ocjene o
GROUP BY o.profesor_id;
```

Vježba 2-7 Rješenja

1. Pronaći sve studente, ime i prezime kao jedna kolona, i njegove ocjene uvećane za 5 tako da se 1 prevodi u 6 a 5 u 10.
```
SELECT CONCAT(s.ime, ' ', s.prezime) AS student, o.ocjena + 5 AS ocjena
FROM studenti s, ocjene o
WHERE s.id = o.student_id;
```

2. Pronaći najmanji mjesec u kojem je data ocjena.
```
SELECT MIN(MONTH(o.datum))
FROM ocjene o;
```

3. Pronaći sve studente koji su se upisali zadnjih 5 godina.
```
SELECT s.ime, s.prezime, s.upis
FROM studenti s
WHERE s.upis > DATE_SUB(NOW(), INTERVAL 5 YEAR);
```

4. Pronaći prosječnu ocjenu svih martova zadnjih 5 godina.
```
SELECT AVG(o.ocjena) AS prosjek
FROM ocjene o
WHERE o.datum > DATE_SUB(NOW(), INTERVAL 5 YEAR) AND MONTH(o.datum) = 3;
```

5. Pronaći prosječnu ocjenu zadnjih 5 godina za svaki mjesec.
```
SELECT MONTH(o.datum) AS mjesec, AVG(o.ocjena) AS prosjek
FROM ocjene o
WHERE o.datum > DATE_SUB(NOW(), INTERVAL 5 YEAR)
GROUP BY mjesec;
```

6. Pronaći prosječnu ocjenu za prvih 6 mjeseci svih godina.
```
SELECT MONTH(o.datum) AS mjesec, AVG(o.ocjena) AS prosjek
FROM ocjene o
WHERE MONTH(o.datum) <= 6
GROUP BY mjesec;
```

7. Pronaći sve studente koji imaju istu dužinu imena.
```
SELECT s1.id, s2.id, s1.ime, s1.prezime, s2.ime, s2.prezime
FROM studenti s1, studenti s2
WHERE LENGTH(s1.ime) = LENGTH(s2.ime) AND s1.id <> s2.id;
```

8. Pronaći prosječnu ocjenu po svim studentima koji imaju istu dužinu imena.
```
SELECT LENGTH(s.ime) AS duzina, AVG(o.ocjena) AS prosjek
FROM studenti s, ocjene o
WHERE s.id = o.student_id
GROUP BY duzina;
```

9. Pronaći prosječnu dužinu naziva grada studenta zaokruženu na dvije cifre.
```
SELECT ROUND(AVG(LENGTH(grad)), 2) FROM student;
```

10. Pronaći prosječnu ocjenu studenata po predmetu, ali ime i prezime vratiti kao jednu kolonu.
```
SELECT AVG(o.ocjena) AS prosjek,
CONCAT(s.ime, ' ', s.prezime) AS student, o.predmet_id
FROM studenti s, ocjene o
WHERE s.id = o.student_id
GROUP BY s.id, o.predmet_id;
```

11. Pronaći prosječnu ocjenu u prvih 10 dana svih mjeseci u 2017. godini.
```
SELECT AVG(o.ocjena) AS prosjek
FROM ocjene o
WHERE YEAR(o.datum) = 2017 AND DAY(o.datum) BETWEEN 1 AND 10;
```

12. Pronaći ime i prezime svih studenata i godine starosti.
```
SELECT ime, prezime, (YEAR(NOW()) – YEAR(datum_rodjenja) AS starost
FROM studenti;
```

13. Pronaći prosječnu starost studenata u svakom gradu.
```
SELECT AVG(YEAR(NOW()) - YEAR(datum_rodjenja)) AS prosjek_god, grad
FROM studenti
GROUP BY grad;
```

14. Među studentima čije je ime kraće od naziva predmeta na kojem imaju ocijenu, pronaći iz kojeg grada dolaze studenti koji imaju najviše ocjena
```
SELECT COUNT(*) broj, s.grad
FROM studenti s, ocjene o, predmeti p
WHERE o.student_id = s.id AND o.predmet_id = p.id
   AND LENGTH(s.ime) < LENGTH(p.naziv)
GROUP BY s.grad
ORDER BY broj DESC
LIMIT 1;
```

15. Pronaći broj albuma u kojima se nalaze komentarisane slike.
```
SELECT COUNT(DISTINCT a.id)
FROM albumi a, albumi_foto af, komentari kom, korisnici kor, foto f
WHERE kom.foto_id = f.id AND af.foto_id = f.id AND af.album_id = a.id;
```

16. Pronaći najstariji datum komentarisanja za svaki tip korisnika

```
SELECT MIN(kom.datum) najstariji, kor.id
FROM korisnici kor, komentari kom
WHERE kor.id = kom.autor_korisnik_id
GROUP BY kor.id;
```

17. Pronaći prosječan broj komentara na slikama za svakog korisnika.

```
SELECT COUNT(kom.id)/COUNT(DISTINCT f.id) AS prosjek, ko.id
FROM komentari kom, korisnici ko, foto f
WHERE ko.id = f.korisnik_id AND kom.foto_id = f.id
GROUP BY ko.id;
```

18. Pronaći broj slika kreiranih na isti dan u sedmici kao i albumi u kojem se nalaze.

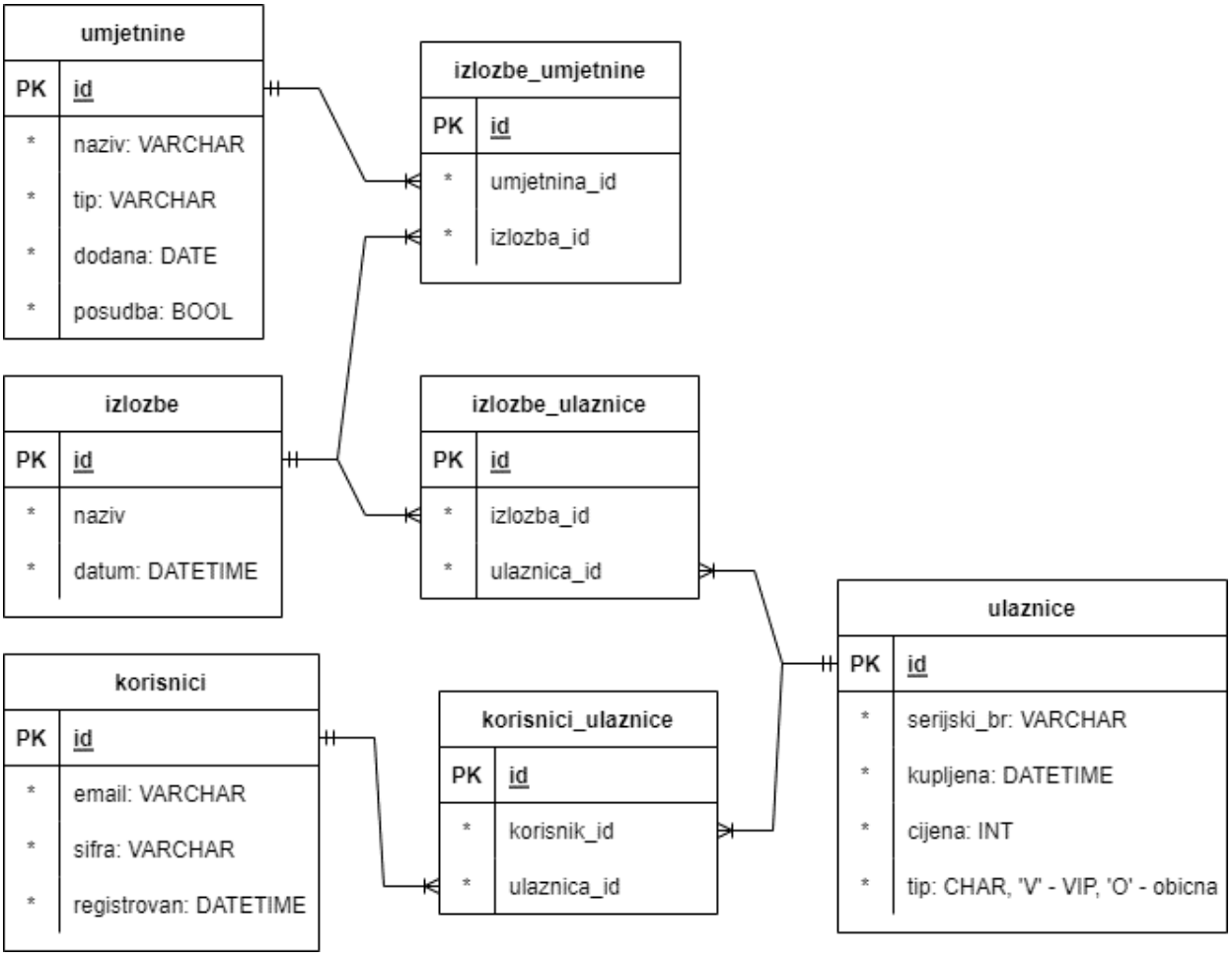
```
SELECT COUNT(DISTINCT f.id)
FROM foto f, albumi a, albumi_foto af
WHERE af.foto_id = f.id AND af.album_id = a.id
   AND DAYOFWEEK(f.datum) = DAYOFWEEK(a.datum);
```

Test 2 Primjer Rješenja

1. Veze više-na-više se koriste kada dva entiteta mogu međusobno imati višestruku pripadnost, npr. Jedna osoba može kupiti više filmova ali isto tako isti film može biti kupljen od strane više osoba. Realizuje se koristeći međutabelu.

2. Implicitni i `INNER JOIN` se ne razlikuju po rezultatima ali će baza podataka bolje optimizovati upit koji koristi `INNER JOIN` i time brže vratiti rezultat.

3.



   a. Pronaći broj izloženih umjetnina po svakoj tematici

```
SELECT iz.naziv, COUNT(*)
FROM umjetnine um, izlozbe_umjetnine ium, izlozbe iz
WHERE ium.izlozba_id = iz.id AND ium.umjetnina_id = um.id
GROUP BY iz.naziv;
```

b. Pronaći zaradu od prodaje preko interneta za svaki tip ulaznice tokom 2017. godine

```
SELECT ul.tip, SUM(ul.cijena)
FROM korisnici_ulaznice kul, ulaznice ul
WHERE kul.ulaznica_id = ul.id
   AND YEAR(ul.kupljena) = 2017
GROUP BY ul.tip;
```

c. Pronaći zaradu za svaku vrstu umjetnine izloženih u aprilu 2018. godine

```
SELECT um.tip, SUM(ul.cijena)
FROM umjetnine um, izlozbe_umjetnine ium,
      izlozbe iz, izlozbe_ulaznice iul, ulaznice ul
WHERE ul.id = iul_ulaznica_id AND iul.izlozba_id = iz.id
   AND ium.izlozba_id = iz.id AND ium.umjetnina_id = um.id
   AND YEAR(iz.datum) = 2018 AND MONTH(iz.datum) = 4
GROUP BY um.tip;
```

d. Pronaći najstarijeg registrovanog korisnika na svakoj izložbi

```
SELECT iz.naziv, MIN(kor.registrovan)
FROM korisnici k, korisnici_ulaznice kul,
      ulaznice ul, izlozbe_ulaznice iul, izlozbe iz
WHERE k.id = kul.korisnik_id AND kul.ulaznica_id = ul.id
   AND iul.ulaznica_id = ul.id AND iul.izlozba_id = iz.id
GROUP BY iz.naziv;
```

e. Unijeti novog korisnika

```
INSERT INTO korisnik VALUES(NULL, 'email@emial.com',
                            SHA1('sifra'), NOW());
```

f. Pronaći zaradu od VIP ulaznica za svaki mjesec u kojem je održana izložba

```
SELECT SUM(ul.cijena) zarada, MONTH(iz.datum) mjesec
FROM ulaznice ul, izlozbe_ulaznice iul, izlozbe iz
WHERE ul.id = iul.ulaznica_id AND iul.izlozba_id = iz.id
   AND ul.tip = 'V'
GROUP BY mjesec;
```

g. Pronaći broj prodatih ulaznica za svaku umjetninu

```
SELECT um.naziv, COUNT(*) AS broj
FROM umjetnine um, izlozbe_umjetnine ium,
      izlozbe iz, izlozbe_ulaznice iul, ulaznice ul
WHERE ul.id = iul_ulaznica_id AND iul.izlozba_id = iz.id
   AND ium.izlozba_id = iz.id AND ium.umjetnina_id = um.id
GROUP BY um.naziv;
```

h. Pronaći zaradu prodatih ulaznica za svaku umjetninu

```
SELECT um.naziv, SUM(ul.cijena) AS zarada
FROM umjetnine um, izlozbe_umjetnine ium,
      izlozbe iz, izlozbe_ulaznice iul, ulaznice ul
WHERE ul.id = iul_ulaznica_id AND iul.izlozba_id = iz.id
   AND ium.izlozba_id = iz.id AND ium.umjetnina_id = um.id
GROUP BY um.naziv;
```

i. Pronaći broj prodatih ulaznica web korisnicima

```
SELECT COUNT(*) broj
FROM korisnici_ulaznice kul, ulaznice ul
WHERE kul.ulaznica_id = ul.id;
```

j. Pronaći prosječnu starost umjetnina koje galerija posjeduje

```
SELECT AVG(YEAR(NOW()) - YEAR(datum)) prosjek
FROM umjetnine
WHERE posudba = FALSE;
```

10 Kontrolni zadaci

0,1,2 1; 3,4 2; 4,5,6 3; 7,8 4; 9,10 5;

[0-2) 1
[2-4) 2
[4-7) 3
[7-9) 4
[9-10] 5

Test 1A

Ime i prezime: _____

1. Kako su relacione baze podataka organizovane?

2. Šta je indeksiranje i zašto se koristi?

3. Potrebno je dizajnirati bazu podataka za vođenje evidencije o dokumentima. Dokumenti imaju svoj naziv i lokaciju, te ime i prezime vlasnika, broj osoba sa kojima je dokument podijeljen, kategoriju, datum kreiranja i datum zadnje modifikacije dokumenta. Osim ovoga dokument može imati datum uništavanja dokumenta kako bi ga sistem mogao automatski obrisati. Potrebno je opisati, riječima i slikom, tipove i sve potrebne detalje i napisati upite:

    a. Pronaći prosječan broj osoba sa kojima su dokumenti podijeljeni po kategoriji sortirano po nazivu kategorije
    b. Pronaći najstariji kreirani dokument. Ukoliko ih ima više odabrati onaj koji je najstariji po modifikaciji.
    c. Pronaći dokumente modifikovane tokom 2017. godine a koji se u nazivu imaju oznaku “vazno”.
    d. Pronaći listu kategorija bez duplikata

4. Potrebno je modelirati bazu za vođenje evidencije o mentorima. Za mentore potrebno je evidentirati ime, prezime, zvanje, br. telefona, datum položenog stručnog ispita, ocjena za položeni ispit koja može biti u rasponu od 1-10, spol, i opis. Potrebno je opisati, riječima i slikom, tipove i sve potrebne detalje i napisati upite:

    a. Pronaći prosjek ocjena mentora koji imaju zvanje bravar za svako prezime
    b. Pronaći sve mentore čije ime počinje sa A i imaju položen ispit najmanje 3 godine od sada, sortirano obrnuto abecedno po imenu

Test 1B

Ime i prezime: _____

1. Koje tipove podataka možemo pohranjivati u SQL baze podataka (nabrojati pet)?

2. Šta je strani ključ?

3. Potrebno je dizajnirati bazu podataka za vođenje evidencije o muzičarima. Za svakog muzičara potrebno je pratiti evidenciju o imenu, prezimenu, nadimku, broju objavljenih pjesama, datumu zadnje objavljene pjesme, kao i naziva produkcijske kuće za koju radi. Potrebno je opisati, riječima i slikom, ograničenja, tipove i sve potrebne detalje i napisati upite:

    a. Unijeti novog muzičara
    b. Pronaći broj muzičara u svakoj producentskoj kući abecedno sortirano po nazivu producentske kuće
    c. Pronaći prosječan broj objavljenih pjesama po producentskoj kući
    d. Pronaći sve muzičare koji se zovu Alisa, Bakir, Kenan ili Zana. Ispisati treću stranicu, ako jedna stranica ima 10 elemenata.

4. Potrebno je dizajnirati bazu za vođenje evidencije o kupcima. Za kupce je potrebno evidentirati, ime, prezime, adresu, grad, broj do sada kupljenih artikala, datum zadnje kupovine, količina potrošenog novca. Potrebno je opisati, riječima i slikom, ograničenja, tipove i sve potrebne detalje i napisati upite:

    a. Pronaći sve kupce čije prezime završava sa “ić” i ima ukupno 5 znakova
    b. Pronaći ukupan broj kupljenih artikala u svakom gradu sortirano po nazivu grada

Test 1C

Ime i prezime: _____

1. Šta je primarni ključ?

2. Koja je razlika između `UPDATE` i `DELETE` komandi?

3. Potrebno je dizajnirati bazu podataka za vođenje evidencije o uposlenicima bolnice. Uposlenici mogu biti doktori, sestre i administrativni radnici. Za svakog uposlenika evidentira se ime, prezime, kad je uposlen, pozicija, ukupan broj sati rada, spol, i komentar. Potrebno je opisati, riječima i slikom, ograničenja, tipove i sve potrebne detalje i napisati upite:

    a. Pronaći broj doktora muškaraca
    b. Pronaći sve uposlenike kojima je broj sati rada između 100 i 200. Ispisati petu stranicu ako jedna stranica ima 20 elemenata.
    c. Pronaći najstariji datum zaposlenja za svaku poziciju, sortirano po nazivu pozicije.
    d. Pronaći broj doktora muškaraca za koje postoji neki komentar

4. Potrebno je dizajnirati bazu podataka za vođenje evidencije o prodavnici artikala. Potrebno je evidentirati naziv artikla, šifru, datum proizvodnje, datum isteka roka, cijenu, trenutni popust, kategoriju, komentar i datum prodaje. Potrebno je opisati, riječima i slikom, ograničenja, tipove i sve potrebne detalje i napisati upite:

    a. Brisanje svih prodanih artikala proizvedenih u 2020. godini.
    b. Pronaći najstariji datum prodaje za svaku kategoriju za artikle sa komentarima sortirano obrnutno abecedno po nazivu kategorije

Test 1D

Ime i prezime: _____

1. Koja je razlika između primarnog i stranog ključa?

2. Koja je razlika između operatora `LIKE` i `AND`?

3. Potrebno je dizajnirati bazu podataka za vođenje evidencije o filmovima. Potrebno je evidentirati naziv filma, datum izdavanja, žanr, dužina trajanja, komentar, naziv režisera, naziv scenariste, budžet filma i zarada filma. Potrebno je opisati, riječima i slikom, ograničenja, tipove i sve potrebne detalje i napisati upite:

    a. Pronaći jedinstvene nazive filmova koji traju između 60 i 90 minuta
    b. Pronaći prosječnu zaradu filmova po žanru, iz 2015. godine koji su napravili profit tj. zaradili više od budžeta
    c. Pronaći sve filmove koji imaju komentar i ispisati treću stranicu ako jedna stranica ima 10 filmova
    d. Pronaći ukupnu zaradu za svaki žanr filma koji traje 90 minuta ili manje

4. Potrebno je dizajnirati bazu podataka za vođenje evidencije o knjigama. Knjige mogu imati naslov, naziv autora, datum izdavanja, žanr, broj stranica, cijenu. Potrebno je opisati, riječima i slikom, ograničenja, tipove i sve potrebne detalje i napisati upite:

    a. Pronaći najstariji daum izdavanja za svaki zanr
    b. Izmjenu cijene svih knjiga na 50 od autora čije prezime ne završava na „ic“

Test 2A

Ime i prezime: _____

1. Šta je veza više-na-više i kada se koristi?

2. Koja je razlika između `INNER` i `LEFT JOIN`?

3. Dizajnirati bazu podataka za računarsku web igru šaha. Samo registrovani igrači mogu igrati, i pored igrača treba omogućiti evidenciju o administratorima. Igrači mogu igrati koliko god igri žele, a jednu igru igraju dva igrača i potrebno je evidentirati ko je sve sa kim igrao. Svi potezi napravljeni tokom svake igre je potrebno pregledati poslije igre stoga su spašeni u jednostavnom tekstualnom formatu npr. “D3 na A2”. Ovo će drugi igrači koristiti poslije da pregledaju igru. Osim gledanja neke igre, igrači mogu i komentarisati igre. Osim ovoga, igrači mogu lajkati i dislajkati igre. Potrebno je nacrtati i opisati sve tabele i vezu između tabela i napisati upite:

    a. Pronaći prosječan broj igri odigranih tokom svakog sata u danu (0-23)
    b. Pronaći 10. stranicu odigranih partija između dva igrača tako da se partija označava sa “Prezime A vs Prezime B”, ako stranica ima 10 partija
    c. Pronaći sve igrače koji su komentarisali igre koje su igrali
    d. Pronaći broj dislajkova koji su igrači dali sortirano od najvećeg do najmanjeg i abecedno po prezimenu
    e. Pronaći prosječan broj odigranih partija igrača koji su ujedno i administratori, zaokruženo na dvije decimale

Test 2B

Ime i prezime: _____

1. Šta je veza jedan-na-više i kada se koristi?

2. Koja je razlika između `VARCHAR` i `TEXT` tipova?

3. Dizajnirati bazu podataka instagram-olike aplikacije. Potrebno je evidentirati korisnike i njihove podatke, a korisnici ujedno mogu biti i administratori. Svi korisnici mogu postaviti objave, koje mogu biti video ili audio uz tekst. Svi korisnici mogu lajkati objave svih korisnika i mogu komentarisati sve objave. Pored ovoga, korisnici međusobno mogu slati privatne poruke. Potrebno je nacrtati i opisati sve tabele i vezu između tabela i napisati upite:

    a. Pronaći broj objava u svakom danu u mjesecu u 2022. godini (1-31)
    b. Pronaći komentare svih korisnika koji su registrovani u neparnoj godini
    c. Pronaći sve privatne poruke tako da su pošiljalac i primalac vraćeni kao “Od A => B” gdje su A i B emailovi
    d. Pronaći broj lajkova od 2018. do 2020. godine, svih korisnika koji nisu administratori
    e. Pronaći prosječan broj komentara u zadnjih mjesec dana po emailu autora

Test 2C

Ime i prezime: _____

1. Koja je veza na slici?

2. Koja je razlika između `LEFT JOIN` i `RIGHT JOIN`?

3. Dizajnirati bazu podataka za novinarski web portal (npr. klix). Portal ima anonimne i registrovane korisnike, novinare i administratore. Novinari mogu objaviti članak u kojem se nalazi tekst sa slikama. Kada se članak objavljuje, mora proći kroz nekoliko faza, poput nacrta, prijedloga, provjere i objave. Na svakom članku, registrovani korisnici mogu pisati komentare. Pored komentara, članci mogu imati nula, jedan ili više tagova. Tako da posjetilac portala može pronaći sve članke po nekom određenom tagu. Potrebno je nacrtati i opisati sve tabele i vezu između tabela i napisati upite:

    a. Pronaći naslove i njihovu dužinu svih novinskih članaka objavljenih od strane administratora, sortiranih po naslovu
    b. Pronaći prosječan broj tagova po članku napisanih petkom
    c. Pronaći najstariji članak u fazi nacrta, od svih novinara čiji email završava sa “gmail.com”. Ispisati 10. stranicu ako stranica ima 10 redova
    d. Pronaći najstarije komentare po članku i autoru
    e. Pronaći nazive svih tagova i broj članaka koji su tagovani tim tagom

Test 2D

Ime i prezime: _____

1. Šta je veza jedan-na-jedan i kada se koristi?

2. Koja je razlika između implicitnog i `INNER JOIN`-a?

3. Dizajnirati bazu podataka za oglasnik artikala (kao olx). Potrebno je evidentirati korisnike, koji mogu biti i administratori. Svaki korisnik, koji je vlasnik artikla, može postaviti jedan ili više artikala na stranicu. Korisnik pregleda artikle, i ukoliko želi, može postaviti pitanja, tj. postoji mogućnost postavljanja pitanja o artiklu i davanja odgovora od strane vlasnika artikla. Osim pitanja, koja su javna, korisnici mogu razmijeniti privatne poruke. Kako bi korisnik mogao upratiti koji su mu artikli zanimljivi, on ih može spasiti kao spašene artikle za kasniji pregled. Potrebno je nacrtati i opisati sve tabele i vezu između tabela i napisati upite:

    a. Pronaći sve artikle koje je korisnik unio na isti dan u sedmici kao kada je korisnik registrovan
    b. Pronaći broj spašenih artikala spašenih tokom januara bilo koje godine, koje je svaki korisnik spasio. Ispisati 3. stranicu ako stranica ima 20 redova.
    c. Pronaći prosječan broj pitanja po artiklu za svakog vlasnika artikla, ali u obzir ulaze samo artikli kojima je naslov kraći od 50 karaktera
    d. Pronaći prosječan broj objavljenih artikala po satu dana (1-24)
    e. Pronaći broj poruka za svakog pošiljaoca čije ime ima do 10 karaktera sortirano po imenu korisnika obrnuto abecedno

Test 3A

Ime i prezime: _____

1. Za šta se koristi `distinct()` metoda?

2. Šta je `ObjectId` i koje su mu karakteristike?

3. Potrebno je dizajnirati bazu podataka za vođenje evidencije o filmovima. Potrebno je prikazati jedan primjer objekta iz takve kolekcije. Nakon toga, napisati upite za:
    a. Pronalazak svih filmova iz 2005. godine
    b. Pronalazak svih filmova akcionih komedija
    c. Pronalazak svih filmova koje je režirao Steven Spielberg a da ??????

Test 3B

Ime i prezime: _____

1. Koja je razlika između `remove()` i `find()` metoda?
2. Šta su kolekcije?
3.

Test 3C

Ime i prezime: _____

1. Koja je razlika između `insert()` i `find()` metoda?
2. Šta je projekcija i kako se koristi?
3.

Test 3D

Ime i prezime: _____

Grupa D

1. Koja je razlika između operatora `$all` i `$in`?
2. Šta je indeksiranje i kako se koristi?
3.

SQL Referenca

```
CREATE DATABASE dbname;      USE dbname;          DROP DATABASE dbname;        DESC table_name;

CREATE [OR REPLACE] TABLE [IF NOT EXISTS] table_name (
       col_name COL_TYPE [NOT NULL] [AUTO_INCREMENT],...
       [PRIMARY KEY(col_name,...)],
       [UNIQUE (col_name)],...
       [CHECK (condition)],...
       [CONSTRAINT constraint_name [CHECK(condition)]],...
       [CONSTRAINT constraint_name [UNIQUE(col_name,...)]],...
       [CONSTRAINT constraint_name FOREIGN KEY(col_name) REFERENCES foreign_table(col_name)],...
);

COL_TYPE = INT | SMALLINT | TINYINT | CHAR | VARCHAR | TEXT | BLOB | FLOAT | DOUBLE | DECIMAL | DATE | DATETIME

CREATE [OR REPLACE] TABLE [IF NOT EXISTS] table_name AS select_query;

ALTER TABLE table_name ADD col_name COL_TYPE;
ALTER TABLE table_name DROP COLUMN column_name;
ALTER TABLE table_name MODIFY COLUMN column_name COL_TYPE;

DROP TABLE table_name;                                  TRUNCATE TABLE table_name;

CREATE [OR REPLACE] INDEX [IF NOT EXISTS] index_name ON table_name (column1, column2,...);
CREATE [OR REPLACE] UNIQUE INDEX [IF NOT EXISTS] index_name ON table_name (column1, column2,...);
ALTER TABLE table_name DROP INDEX index_name;

INSERT INTO table_name [(column1, column2, column3,...)] VALUES (value1, value2, value3,...);
UPDATE table_name SET column1 = value1, column2 = value2,... [WHERE condition];
DELETE FROM table_name [WHERE where_condition];

SELECT [DISTINCT] column1 [AS alias1],... FROM table_name [table_alias],... [WHERE condition]
       [GROUP BY column_name(s)]
       [ORDER BY column_name(s) [ASC | DESC]]
       [LIMIT number [OFFSET number]]

... WHERE columnN [NOT] LIKE pattern;            pattern :=   % => 0,1,2+ chars   |   _ => exactly one
... WHERE condition1 [AND|OR] condition2 [AND|OR] condition3 ...;
... WHERE NOT condition;                                              operatori := >, <, =, <=, >=, <>
... WHERE column_name IS [NOT] NULL;
... WHERE column_name IN (value1, value2, ...);
... WHERE column_name BETWEEN value1 AND value2;

SELECT column_name(s) FROM table1 t1 [INNER|LEFT|RIGHT] JOIN table2 t2 ON t1.column=t2.column,...;
SELECT column_name(s) FROM table1 CROSS JOIN table2;
SELECT column_name(s) FROM table1 UNION [ALL] SELECT column_name(s) FROM table2;

CREATE [OR REPLACE] VIEW [IF NOT EXISTS] view_name AS select_statement;
DROP VIEW view_name;

MySQL Aggregation functions := SUM, MIN, MAX, COUNT, AVG
MySQL String functions := LENGTH, REVERSE, TRIM, LTRIM, RTRIM, UPPER, LOWER, CONCAT
MySQL Number functions := POW, SUM, SIN, COS, ROUND
MySQL Date functions := CURDATE, NOW, DAY, MONTH, YEAR, DAYOFWEEK, DAYOFMONTH, DAYOFYEAR, WEEKOFYEAR,
DATE_ADD(date, INTERVAL value addunit)
addunit := SECOND | MINUTE | HOUR | DAY | WEEK | MONTH | QUARTER | YEAR
```

10 Kontrolni zadaci

Ideje:

- Sistem za kontrolu pristupa
- Video portal ala youtube
- Ala instagram
- Ala facebook
- Ala forum
- Document management system
- Ala ebay
- Ala amazon
- Nekretnine
- Banka
- e-dnevnik škole
- Galerije
- Umjetnicka galerija
- Kino sala
- Stadion utakmice
- Sportski tim (sponzori, tim, treneri)
- E-sport, igraci tim sponzori turniri
- Igra igrac, achievement, klan, sponzorstvo, mečevi
- Chat aplikacija, lista kontakata, blokirani, poruke, grupe ala viber
- Objave notifikacije
- Dionice (notifikacije itd)
- Hotel management
- Tvprofil .net
- logistika shipping
- teams baza podataka
- darovanje krvi
- db publikacija
- baza za car crashes

DSS
Dizajnirati bazu podataka za sistem za upravljanje dokumentima. Pošto je web sistem, korisnici se trebaju registrovati da bi mu pristupili. Nakon toga, korisnici mogu kreirati ili upload-ovati bilo koji dokument, i pregledati ga ukoliko za to postoji mogućnost. Pored ovoga, potrebno je evidentirati mogućnost dijeljenja dokumenata. Samo jedan korisnik može kreirati dokument, međutim on može dati ovlasti čitanja ili mijenjanja tog dokumenta drugom korisniku. Osim ovoga, korisnici mogu slati poruke jedan drugom, nevezano za dokument, ali moguće je i da poruka bude uvezi nekog dokumenta.

Kornsici, dokumenti, dijeljenje, poruke

E-dnevnik
Dizajnirati bazu podataka za elektronski dnevnik. Potrebno je evidentirati korisnike, koji mogu biti učenici, nastavnici, roditelji i administratori. Učenici su dodijeljeni na više predmeta. Nastavnici se dodijeljuju na određeni predmet, i za svaki čas tog predmeta, evidentiraju prisustvo. Osim toga, nastavnici evidentiraju ocjene koje učenici dobijaju na predmetima.

Korisnici, predmeti, ocjene, casovi, prisustvo

Kino
Dizajnirati bazu podataka za kino. Kino ima 5 dvorana, i emituju se projekcije filmova u raznim terminima. Gosti mogu kupiti karte te je potrebno evidentirati karte za projekcije. Potrebno je evidentirati kada je karta kupljena, za koju projekciju i isto tako kada je karta iskorištena. Filmovi mogu biti projektovani u različitim terminima, te je ovo potrebno evidentirati kao i informacije o filmovima radi marketinga.

Galeriju slika
Dizajnirati bazu podataka za galeriju slika. Da bi korisnici postavili svoje slike, potrebno je da se registruju. Registrovani korisnici mogu upload-ovati sliku ili video na svoj račun. Osim ovoga, oni mogu kreirati albume u koje mogu spremati slike. Jedna slika ili video može biti dodijeljeno u više albuma. Albumi se mogu podijeliti sa drugim korisnicima, i ovo je potrebno evidentirati.

Korisnici, slike, albumi, albumi_share

Umjetnička galerija
Dizajnirati bazu podataka za umjetničku galeriju. Galerija posjeduje umjetnine (slike, skulpture i ostale radove), ali isto tako može posuditi umjetnine od druge galerije. Novac zarađuje od prodaje običnih i VIP ulaznica za svaku izložbu koju napravi, tako da je potrebno ovo evidentirati. Pošto izložbe mogu biti tematske, to znači da nisu sve umjetnine izložene na svakoj izložbi te je potrebno radi dokumentacije ovo evidentirati. Potrebno je omogućiti prodaju ulaznica preko web stranice, samo za registrovane korisnike.

Umjetnine, ulaznice, izložbe, izlozene_umjetnine, korisnici

Viber
Dizajnirati bazu podataka za aplikaciju za razmjenu poruka (kao Viber). Potrebno je evidentirati registraciju korisnika. Nakon registracije, koriscnici mogu slati poruke jedni drugima i mogu imati samo jedan prozor za razmjenu poruka međusobno. Osim ovoga, svaki korisnik može kreirati grupe u koje može dodati druge korisnike. Grupe mogu biti tzv. read-only tj. samo korisnik koji je kreirao grupu može da šalje poruke. Osim ovoga, potrebno je omogućiti blokiranje korisnika, tj. jedna osoba može blokirati jednu ili više osoba kako mu te osobe ne bi slale poruke.

Korisnici, poruke, grupe, grupe_korisnici, ignore

Facebook
Dizajnirati bazu podataka za aplikaciju poput Facebook-a.

Korisnici, objave, komentari, poruke,

Banka
Dizajnirati bazu podataka za banku. Banka ima klijente i uposlenike koji također mogu biti klijenti u banci. Uposlenici su uposeni na određenim pozicijama i ovo je potrebno evidentirati. Evidencija se vodi i o klijentima. Klijenti mogu imati više računa u banci i račun može biti tipa ili tekući ili štedni. Svaki račun ima svoj broj koji ima 13 alfanumeričkih znakova. Banka također vrši transakcije, koje ne moraju nužno biti od klijenata već …….

korisnici, racuni, transakcije, obavijesti,

Potrebno je modelirati sistem koji vrši evidenciju korisnika jednog web foruma. Web forum ima korisnike. Korisnici se mogu registrovati sa email-om, korisničkim imenom i šifrom. Ali samo jedan email po korisniku je dozvoljen. Forum ima teme, i svaki registrovani korisnik može kreirati temu, i napisati *post* na određenu temu. Svaka tema može biti moderirana od strane više moderatora. Ali moderatori ne mogu moderirati svaku temu, već samo određene teme. Osim moderatora, postoje administratori koji imaju sve moguće privilegije.

Potrebno je modelirati napredni sistem privilegija korisnika. Neki sistem može imati korisnike kao npr. web forum pod 2. Ovdje je potrebno napraviti sistem privilegija, gdje je svaka privilegija posebno definisana (npr. korisnik može kreirati post, korisnik može kreirati temu, korisnik može modifikovati tuđi post, korisnik može blokirati drugog korisnika). Svaka od ovih privilegija može biti dio grupnih privilegija (npr. moderator). Korisniku se mogu dodijeliti privilegije individualno ili grupno, ali ako se dodijele grupno, mogu se “prepisati”, tj. ako se korisniku dodijeli skup privilegija moderatora, može mu se oduzeti individualna privilegija modifikovanja tuđeg posta.

Teme za praktične radove

Ovo su teme za praktične radove. Za potrebe svakog od ovih sistema, potrebno je analizirati potrebe postojećih sistema i potreba korisnika te dizajnirati bazu podataka kako bi sistem evidentirao potrebne podatke. Osim ovoga, potrebno je osmisliti sve potrebne upite koje bi sistem potencijalno koristio te ih napisati. Počev od ubacivanja određenih podataka u bazu, modifikacije i brisanja, do izvlačenja ispisa, i statističkih informacija iz baze podataka. Između 5 i 10 tabela bi trebalo biti dovoljno za evidenciju podataka u ovim sistemima.

Dizajn baze napraviti u MySQL Workbenchu. Napraviti export baze iz phpmyadmina a upite napisati u pdf-u ili txt-u i eksportovati kao sliku. Sve zajedno poslati na email ili na teams ako je preveliko za email.

1. Potrebno je dizajnirati bazu podataka za sistem za vođenje evidencije o nekretninama. Specifičnosti nekretnina je što zemljišta imaju geometrijski opis, te GPS lokaciju.

2. Potrebno je dizajnirati bazu podataka za sistem poput Steam-olikoj platformi za igre. Specifičnost ovakve platforme su tzv. achievementi u igrama, same igre, novosti uvezi njih, te privatne poruke koji korisnici mogu da šalju međusobno.

3. Potrebno je dizajnirati bazu podataka za sistem poput YouTube-a. Specifičnost YouTuba je razmjena video klipova te opcije praćenja, komentarisanja i plejlisti.

4. Potrebno je dizajnirati bazu podataka za web forum. Specifičnosti foruma su da ima puno različitih vrsta podataka gdje su svi povezani.

5. Potrebno je dizajnirati bazu podataka za turističku agenciju. Ova agencija ima web portal i za uposlenike i za klijente.

6. Potrebno je dizajnirati bazu podataka za agenciju za registraciju auta. Ova agencija ima web portal i za uposlenike i za klijente.

7. Potrebno je dizajnirati bazu podataka za sistem za brokere koji trguju dionicama na web stranici. Specifičnost sistema je potreba spašavanja velikog broja podataka i izvještaja po različitim periodima.

8. Potrebno je dizajnirati bazu podataka za dostavljačku / kurir službu. Specifičnost ove aplikacije je geolokacija, kao i vremena i cijene dostava te popusti.

9. Potrebno je dizajnirati bazu podataka za menadžment hotela. Specifičnost ove aplikacije je vođenje računa o zauzetim i rezervisanim resursima (sobe, uposlenici itd...).

10. Potrebno je dizajnirati bazu podataka za sistem za glasanje.

11. Ili neki drugi sistem po slobodnom odabiru koji nije obrađen na predmetu.

Teme za referate

1. Indeksiranje — motivacija, svrha, implementacija
2. Normalizacija i normalne forme u bazi podataka
3. Unix time — spašavanje datuma i vremena kao Unix epoha u bazama podataka.
4. NoSQL baze podataka. Razlika između SQL i NoSQL baza podataka.
5. Blockchain kao bazu podataka
6. Ograničenja u bazama podataka (constraints) — motivacija, svrha i implementacija
7. Procedure u MySQL-u
8. Pretvaranje tipova u MySQL-u
9. Replikacija u MySQL-u
10. Migracija u MySQL-u
11. Pogledi u MySQL-u (obični, materijalizovani; obnavljanje pogleda itd)
12. Opisati PostgreSQL
13. Opisati CouchDB
14. Opisati HBase
15. Opisati ElasticSearch
16. Data Mining — opis, motivacija, svrha, metode i primjena
17. Ili neka druga tema po prijedlogu a da nijie obrađena na predmetu