

UNIVERZA V MARIBORU
FAKULTETA ZA NARAVOSLOVJE IN MATEMATIKO
Oddelek za matematiko in računalništvo

dr. Krista Rizman Žalik

Podatkovne baze
Zbirka nalog z rešitvami

Maribor, 2021

Predgovor

V današnji informacijski dobi je zbiranje in hranjenje informacij v podatkovnih bazah osrednjega pomena.

V tem e-gradivu so zbrane vaje in naloge z rešitvami, ki so primerne za izgradnjo znanja iz podatkovnih baz. Namenjene so predvsem študentom izobraževalnega računalništva pri predmetu Osnove podatkovnih baz. V posameznem poglavju je podanih tudi nekaj osnovnih pojmov. V pomoč pri zbranem gradivu pa so tudi obstoječi učbeniki, zbirke vaj in internetni viri.

Kazalo

1	Uvod	4
1.1	Struktura e-gradiva	4
2	Podatkovno modeliranje relacijskih baz z <i>ER</i> diagramom in ustvarjanje baze	5
2.1	Zbirka nalog	6
3	SQL	9
3.1	Vstavljanje podatkov	10
3.2	Preproste poizvedbe	10
3.2.1	Zbirka nalog	11
3.3	Pogoji z <i>AND</i> , <i>OR</i>	12
3.3.1	Zbirka nalog:	12
3.4	Poizvedbe z <i>IN</i> , <i>EXISTS</i>	13
3.4.1	Zbirka nalog	14
3.5	Funkcije: <i>COUNT</i> , <i>AVG</i> , <i>SUM</i> , <i>MIN</i> , <i>MAX</i>	15
3.5.1	Zbirka nalog	15
3.6	Brisanje in spreminjanje podatkov	17
3.6.1	Zbirka nalog	17
3.7	Stiki: <i>LEFT</i> , <i>RIGHT</i> , <i>FULL OUTER</i> , <i>INNER JOIN</i>	17
3.7.1	Zbirka nalog	18
3.8	Količnik in SQL	19
3.8.1	Zbirka nalog	19
4	Transakcije	23
4.1	Naloga	24
5	Prožilci	25
5.1	Zbirka nalog	26
6	Procedure	30
6.1	Zbirka nalog	30
7	MySQL in PHP	34
7.1	Zbirka nalog	34

8 *NoSQL* podatkovna baza *MongoDB* 38

8.1 Enostavne poizvedbe v MongoDB 38

8.1.1 Zbirka nalog 39

8.2 Kompleksne poizvedbe v MongoDB 45

8.2.1 Zbirka nalog 46

Literatura 52

1 Uvod

V današnji informacijski dobi je zbiranje in hranjenje informacij v podatkovnih bazah osrednjega pomena. Prednosti uporabe podatkovnih baz so: večja dostopnost podatkov in konsistentnost, izboljšana integriteta in varnost podatkov, lažje arhiviranje in obnova podatkov in lažje vzdrževanje aplikacij. Najpomembnejše pridobitve podatkovnih baz so podatkovna neodvisnost, učinkovit dostop do podatkov z nadzorom nad sočasnim dostopom do podatkov, centralizirana administracija in enostavno obnavljanje po nesrečah.

1.1 Struktura e-gradiva

E-gradivo je sestavljeno iz devetih poglavij. Uvodnemu poglavju sledi poglavje, ki opisuje modeliranje relacijskih podatkovnih baz z entitetno-relacijskimi diagrami. V tretjem poglavju spoznamo deklarativni programski jezik *SQL* (angl. *Structured Query Language*) za delo s podatki v relacijskih podatkovnih bazah, ki se je razvil med prvimi in je še veno najpogostejši. V četrtem poglavju spoznamo transakcije, v petem prožilce in v šestem procedure. V vsakem poglavju je zbirka nalog, ki jih delamo z orodjem *MySQL Workbench* v odprtokodni implementaciji relacijske baze *MySQL*. V sedmem poglavju si pogledamo, kako implementiramo vse osnovne operacije za delo z bazo (vrivanje podatkov, poizvedovanje, spreminjanje, in brisanje) v jeziku *PHP*. *PHP* je široko uporabljen, brezplačen strežniški skriptni jezik za ustvarjanje dinamičnih in interaktivnih spletnih strani. V osmem poglavju opišemo *NoSQL* podatkovno bazo *MongoDB*.

2 Podatkovno modeliranje relacijskih baz z *ER* diagramom in ustvarjanje baze

Relacijske baze, kot jih je definiral Codd [1] združujejo podatke v tabelah s stolpci in vrsticami, kjer so tabele medsebojno logično povezane s pomočjo enakih stolpcev tabel.

Načrtovanje podatkovne baze je postopek opredelitve in razvoja strukture *PB*. Za načrtovanje uporabimo entitetno relacijski diagram (*ER* diagram). Avtor *ER* diagrama je Peter Pin-Shan Chen (l. 1976) [2]. *ER* diagram omogoča pomensko predstavitev sveta. Je grafična predstavitev, ki vključuje entitete in povezave med njimi. Entitete so objekti iz realnega sveta o katerih zbiramo, hranimo in obdelujemo informacije. Povezava poveže dve entiteti in opiše razmerja med entitetama. *ER* diagram je potrebno preslikati v relacijski model, ker sistemi za upravljanje relacijskih baz ne podpirajo *ER* modela.

V vsakem projektu razvoja podatkovne baze je potrebno:

1. določiti zbirko vprašanj za izbrano domeno projekta,
2. oblikovati zahteve za podatke, ki jih zahteva aplikacija,
3. narisati *ER*-diagram, ki zajema te zahteve.

Aplikacija, ki obdeluje več podatkov, le-te hrani v podatkovni bazi. Primeri aplikacij podatkovnih baz:

- spletna trgovina,
- izposoja predmetov (smuči, strojev...)v
- kvizi,
- zbirka fotografij, slik, avtorjev,
- elektronska redovalnica,
- kuharski recepti,
- letalski prevozi,
- emisije CO₂ in vplivni faktorji (površina zgradb, zelenja, št. ur dela, podnebni pas),
- agencija za delavce,
- študentski servis.
- gibanje cen (delnic, nafte)
- kontakti: telefoni, maili, ljudje, skupine, dogodki,
- glasba, naslovi skladb, avtorji, viri, zvrsti, skupine,
- tekme, liga,
- obiskovalci spletne strani,
- zbirka filmov, serij, epizod, igralcev.

Za manjši projekt veljajo naslednja priporočila za izdelavo vprašanj in zahtev.

Zapišemo 10 tipičnih vprašanj zanimivih za domeno izbrane aplikacije Vprašanja zapišemo z besedilom in ne uporabimo poizvedovalnega jezika za podatkovne baze.

Ta vprašanja uporabimo za oblikovanje podatkovnih zahtev in preverbo popolnosti le teh. Podatkovne zahteve zapišemo v besedilu. Oblikovanje zahtev in izdelava *ER* modela lahko poteka vzporedno.

Manjši ER diagram vsebuje približno 15 entitet.

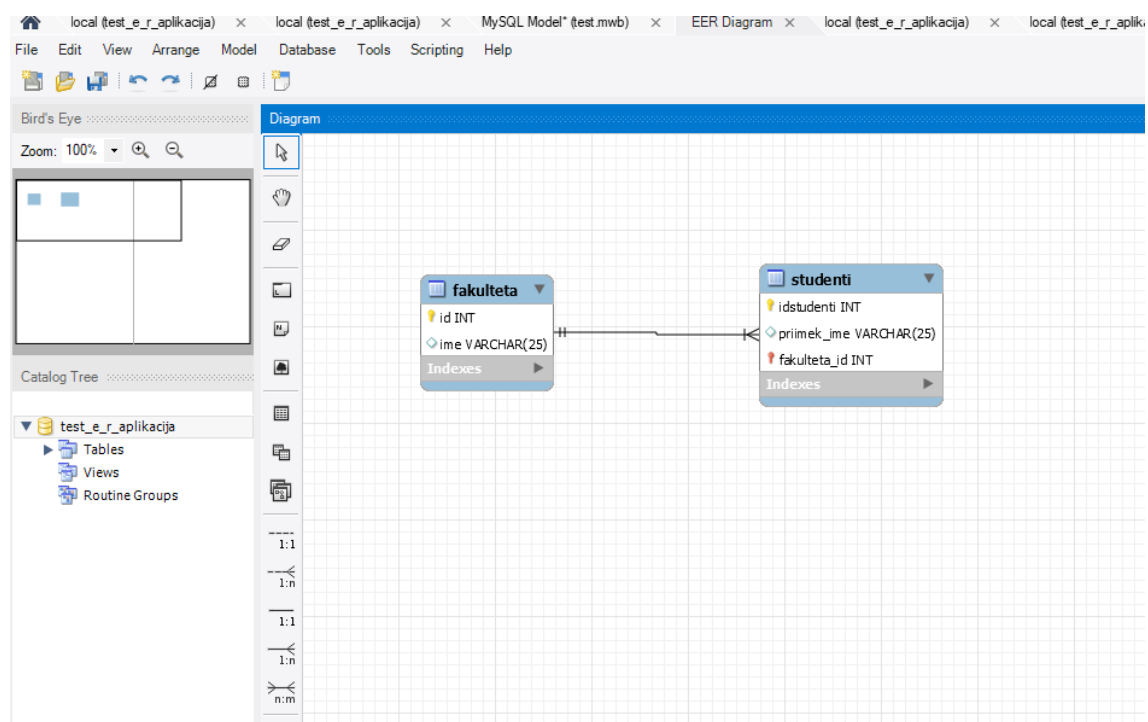
ER diagram izdelamo z uporabo orodja *MySQL Workbench* in z ukazom »Database>Forward Engineering« ustvarimo tudi tabele v bazi.

2.1 Zbirka nalog

- a) Izdelajmo preprost ER diagram za hranjenje študentov in fakultet ter ustvarimo bazo. Nato tabele napolnimo s testnimi podatki.

Uporabljamo *MySQL Workbench* 8.0 (verzija 8.0.25) in *UwAmp* verzijo 3.1.0. *UwAmp* je prosto razvojno okolje za razvoj spletnih aplikacij v okolju oken, ki vključuje spletni strežnik Apache in PHP ter podatkovno bazo *MySQL*. Najprej izdelamo ER diagram v okolju *MySQL Workbench* z ukazom »File>NewModel + AddDiagram«.

Primer ER diagrama:



Ustvarimo novo shemo v orodju *MySQL Workbench*. Poimenujmo jo: *test_e_r_aplikacije*.

Z miško izberemo ustvarjeno shemo in jo določimo kot privzeto z izbiro: »Set as Default Shema.«

Odpremo ER diagram in preimenujemo bazo enako kot smo poimenovali ustvarjeno shemo v katalogu (v spodnjem levem oknu na sliki zgoraj).

Ustvarimo novo povezavo na bazo (ang. connect string) in jo poimenujmo *localhost ER*. V zadnji vrstici opisa povezave navedimo shemo, ki smo jo ustvarili (*test_e_r_aplikacije*).

Izberemo »>Database > Forward Engineering« in vpišemo shranjeno povezavo (*localhost e-r*). Geslo za uporabnika *root* je *root*.

Vstavimo podatke:

```
INSERT INTO študenti VALUES (1,"1",1)
```

Pozorni smo, da ima primarni ključ le eno vrednost.

Najprej vnesemo zapise s primarnim ključem, nato zapise, ki vsebujejo tuje ključe, ki se sklicujejo na primarne.

Sicer dobimo naslednji izpis o napaki:

```
INSERT INTO študenti VALUES (1,"1",1)      Error Code: 1452. Cannot add or UPDATE a
child row: a foreign key constraint fails (`test_e_r_aplikacija`.`študenti`, CONSTRAINT
`fk_studenti_fakulteta` FOREIGN KEY (`fakulteta_id`) REFERENCES `fakulteta` (`id`) ON
DELETE NO ACTION ON UPDATE NO ACTION)      0.016 sec
```

Preverjanje tujih ključev lahko izklopimo:

```
SET FOREIGN_KEY_CHECKS=0;
```

V tem primeru vrstni red vnosov, najprej primarni in nato tuj ključ, ni pomemben. Sami pa moramo paziti, da za vsak tuj ključ vnesemo tudi primaren, sicer dobimo nekonsistentno bazo.

V bazo vnesemo vrstico tabele za vrstico, nato izpišemo podatke in vključimo preverjanje tujih ključev.

```
INSERT INTO študenti VALUES (1,"1",1);
SELECT * FROM študenti;
SELECT * FROM fakulteta;
SET FOREIGN_KEY_CHECKS=1;
```

b)Izdelajmo bazo, ki bo hranila podatke o študentih, revijah in rezervacijah zadnje številke revije. Študent lahko rezervira zadnji izvod revije, ostali so prosto dostopni v čitalnici.

```
CREATE TABLE študenti (
  id INT(11) NOT NULL,
  priimek VARCHAR(20) NULL,
  ocena INT(11) NULL,
  starost INT(11) NULL,
  PRIMARY KEY (id));
```

```
CREATE TABLE revije (
  id INT(11) NOT NULL,
```



```
naslov VARCHAR(20) NOT NULL,  
pogostost INT(3) NULL,  
PRIMARY KEY (id);
```

Študenti vedno rezervirajo zadnji izvod revije. Ostali izvodi so prosto dostopni. Ker študent lahko rezervira več revij in tudi posamezno revijo lahko rezervira več študentov ob različnih dnevih, naredimo presečno tabelo rezervacije:

```
CREATE TABLE rezervacije (  
  študent_id INT(11) NOT NULL,  
  revija_id INT(11) NOT NULL,  
  dan DATE NULL,  
  PRIMARY KEY (študent_id, revija_id),  
  CONSTRAINT `fk_revije`  
    FOREIGN KEY (`idnew_table`)  
    REFERENCES `test`.`revije` (`id`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION,  
  CONSTRAINT `fk_štud`  
    FOREIGN KEY (`idnew_table`)  
    REFERENCES `test`.`študenti` (`id`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION);
```

3 SQL

Strukturirani povpraševalni jezik *SQL* (angl. Standard Query Language) je standardni jezik za relacijske podatkovne baze. Določen je z *ANSI* standardom (angl. American National Standards Institut) iz 1986, in *ISO* (International Organization for Standardization) iz leta 1987. *SQL* je bil en izmed prvih komercialnih jezikov za delo z relacijskimi podatkovnimi bazami in je postal najpogostejše uporabljen jezik za delo s podatkovnimi bazami [3].

SQL [4,5] se uporablja za vstavljanje, iskanje, spreminjanje in brisanje zapisov v podatkovni bazi.

Najpogostejši ukazi jezika *SQL* so

- a) ukazi upravljanja s podatki, ki tvorijo jezik DML (angl. Data Manipulation Language):

SELECT izpiše podatke iz baze,

UPDATE spremeni podatke v podatkovni bazi,

DELETE zbriše podatke iz baze,

INSERT INTO stavek vrine nove podatke v bazo.

- a) ukazi za definiranje podatkov, ki tvorijo jezik DDL (angl. Data Definition Language):

CREATE DATABASE tvori novo bazo,

ALTER DATABASE spremeni podatkovno bazo,

CREATE TABLE ustvari novo tabelo,

ALTER TABLE spremeni tabelo,

DROP TABLE zbriše tabelo,

CREATE INDEX tvori indeks po ključu za iskanje,

DROP INDEX zbriše indeks.

SQL poizvedbe smo izvedli na shemi s tabelami študenti, revije in rezervacije.

```
CREATE TABLE študenti (  
  id INT(11) NOT NULL,  
  priimek VARCHAR(20) NULL,  
  ocena INT(11) NULL,  
  starost INT(11) NULL,  
  PRIMARY KEY (id));
```

```
CREATE TABLE rezervacije (  
  študent_id INT(11) NOT NULL,  
  revija_id INT(11) NOT NULL,  
  dan DATE NULL,  
  PRIMARY KEY (študent_id, revija_id));
```

```
CREATE TABLE revije (  
  id INT(11) NOT NULL,  
  naslov VARCHAR(20) NOT NULL,  
  pogostost INT(3) NULL,  
  PRIMARY KEY (id);
```

3.1 Vstavljanje podatkov

Stavek INSERT omogoča vrivanje podatkov v bazo.

Sintaksa stavka INSERT:

```
INSERT INTO ime_tabele (kolona1, kolona2, kolona3, ...)  
VALUES (vrednost1, vrednost2, vrednost3, ...);
```

Pri vrivanju podatkov je potrebno paziti, da najprej vnesemo primarne in nato tuje ključe, ker sicer dobimo napako ob vnosu. Lahko pa preverjanje izključimo pred vnosom z ukazom:

```
SET FOREIGN_KEY_CHECKS=0;  
in po vnosu preverjanje vključimo:  
SET FOREIGN_KEY_CHECKS=1;
```

V vse tri ustvarjene tabele vrinemo vsaj tri vrstice. Vrinili smo naslednje podatke:

```
INSERT INTO revije(id, naslov, pogostost) VALUES (1, 'Monitor',12);  
INSERT INTO revije(id, naslov, pogostost) VALUES (2, 'Uporabna informatika',4);  
INSERT INTO revije(id, naslov, pogostost)VALUES (3, 'Elektrotehnoški vestnik',5);
```

```
INSERT INTO študenti VALUES (1, 'Novak Miha', 9, 19);  
INSERT INTO študenti VALUES (2, 'Novak Nik', 8, 24);  
INSERT INTO študenti VALUES (3, 'Novak Nejc', 9, 27);  
INSERT INTO študenti VALUES (4, ' Novak Žiga', 9, 22);
```

```
INSERT INTO rezervacije (študent_id, revija_id) VALUES(1,1, '2021-5-20');  
INSERT INTO rezervacije (študent_id, revija_id) VALUES(1,2, '2021-5-20');  
INSERT INTO rezervacije (študent_id, revija_id) VALUES(2,3);
```

3.2 Preproste poizvedbe

Zapišimo osnovni blok za poizvedbe.

```
-- Osnovni blok poizvedbe:  
SELECT [distinct] seznam atributov  
FROM ime tabel  
WHERE pogoji  
ORDER BY atributi;
```

FROM opiše tabele.

WHERE izloči vrstice, ki ne zadostujejo pogoju.

DISTINCT izloči ponavljajoče vrstice.

ORDER BY uredi po atributih naraščajoče, če sledi ASC ali pa nič in padajoče, če atributom sledi besedica DESC.

Če poizvedba zahteva podatke iz več tabel, potem moramo zapisati pogoje združevanja podatkov iz več tabel.

Poizvedbe lahko tudi komentiramo. Komentar je lahko dolg eno vrstico ali več.

```
SELECT * FROM študenti ; --enovrstični komentar
```

```
/* Večvrstični kmentar: Naslednja poizvedbe izbere vse kolone  
in vse vrstice iz tabele študenti : */
```

```
SELECT * FROM študenti ;
```

3.2.1 Zbirka nalog

- a) Izpišimo različne ocene študentov in nato izpišimo še vse ocene študentov.

```
SELECT distinct ocena FROM študenti ;
```

```
SELECT id, ocena FROM študenti ;
```

- b) Izpišimo vse priimke in ocene študentov:

```
SELECT priimek, ocena FROM študenti;
```

- c) Izpišimo vse študente in revije, ki so jih rezervirali:

```
SELECT m.priimek, r.revija_id  
FROM študenti m, rezervacije r  
WHERE m.id=r.študent_id;
```

- d) Izpišimo vse pare študentov, tako da starejšemu sledi mlajši:

```
SELECT m1.priimek, m1.starost, m2.priimek, m2.starost  
FROM študenti m1, študenti m2  
WHERE m1.starost > m2.starost;
```

- e) Izpišimo vse pare študentov in uparimo vsakega študenta z vsakim. Dobimo kartezični produkt študentov:

Če v poizvedbi iz dveh tabel ni pogoja, dobimo kartezični produkt, ko je vsak zapis prve tabele povezan z vsakim zapisom iz druge tabele.

```
SELECT m1.priimek, m1.starost, m2.priimek, m2.starost
FROM študenti m1, študenti m2;
```

f) Izpišimo študente in starost pred 5 in 10 leti:

```
SELECT priimek, starost-5 as starost_pred_5, starost-10 as starost_pred_10
FROM študenti
WHERE priimek Like 'Novak %';
```

g) Izpišimo pare študentov, tako da ima drug študent pol manjšo oceno kot prvi.

```
SELECT m1.priimek, m2.priimek
FROM študenti m1, študenti m2
WHERE 2*m1.ocena= m2.ocena;
```

h) Izpišimo pare študentov, tako da ima drug študent za 1 manjšo oceno kot prvi.

```
SELECT m1.priimek, m2.priimek, m1.ocena, m2.ocena
FROM študenti m1, študenti m2
WHERE m1.ocena= m2.ocena-1;
```

i) Izpišimo vse študente s priimkom, ki se začne s črko N in konča z malo črko k.

_ nadomesti en znak
% nadomesti en ali več znakov

```
SELECT priimek
FROM študenti
WHERE priimek like 'N%k';
```

3.3 Pogoji z *AND*, *OR*

Več pogojev poizvedbe lahko združujemo z logičnima operatorjema in- *AND* in ali - *OR*.

3.3.1 Zbirka nalog:

a) Izpišimo *id* študenta, ki je rezerviral revijo, ki izide vsak mesec ali ima 4 izvode na leto.

```
SELECT r.študent_id
FROM revije l, rezervacije r
WHERE l.id=r.revija_id AND
(l.pogostost=4 or l.pogostost =12);
```

b) Izpišimo id študenta, ki je rezerviral revijo, ki izide vsak mesec in revijo, ki ima 4 izvode na leto.

```
SELECT DISTINCT r.študent_id
FROM revije l, rezervacije r
WHERE l.id=r.revija_id AND
(l.pogostost=4 and l.pogostost =12);
```

Zgornja poizvedba ne vrne nobenega zapisa, ker preverja pogostost v eni vrstici tabele revije in le-ta ne more biti hkrati 4 in 12!

V standardnem SQL jeziku lahko uporabimo *INTERSECT*.

```
SELECT DISTINCT r.študent_id
FROM revije l, rezervacije r
WHERE l.id=r.revija_id AND l.pogostost=4
INTERSECT
SELECT DISTINCT r1.študent_id
FROM revije l1, rezervacije r1
WHERE l1.id=r1.revija_id AND l1.pogostost=12;
```

MySQL ne pozna presečišča (*INTERSECT*) in je potrebno poizvedbo zapisati s stavkom *SELECT*, ki dvakrat poizveduje v istih tabelah revije in rezervacije.

```
SELECT DISTINCT r.študent_id
FROM revije l, rezervacije r, revije l1, rezervacije r1
WHERE l.id=r.revija_id AND l.pogostost=4 AND l1.id=r1.revija_id AND
l1.pogostost=12
AND r1.študent_id = r.študent_id;
```

3.4 Poizvedbe z *IN*, *EXISTS*

IN operator dovoljuje v pogoju (za *WHERE*) primerjanje z več vrednostmi. Zapišemo jih za *IN* operatorjem, ki opiše več pogojev (*OR*), od katerih mora biti izpolnjen vsaj en pogoj.

```
SELECT imena_kolon
FROM ime_tabele
WHERE ime_kolone IN (vrednost1, vrednost2, ...);
```

```
SELECT imena_kolon
FROM table_name
WHERE column_name IN (SELECT stavek);
```

Operator *EXISTS* testira obstoj vsaj enega zapisa v poizvedbi, ki sledi besedi *EXISTS*. Operator *EXISTS* vrne resnično (*angl. true*), če poizvedba za *EXISTS* vrne en ali več zapisov.

```

SELECT imena_kolon
FROM ime_tabele
WHERE EXISTS
(SELECT ime_kolone FROM ime_tabele WHERE pogoj);

```

3.4.1 Zbirka nalog

a) Izpišemo študente, ki niso rezervirali nobene revije. Standardni *SQL* pozna operator *EXCEPT*:

```

SELECT id FROM študenti
EXCEPT
SELECT id
FROM rezervacije r, študenti m
WHERE m.id = r.študent_id;

```

MySQL pa operatorja *EXCEPT* ne pozna. Potrebno je uporabiti operator *IN* ali *EXISTS*.

```

SELECT id FROM študenti
WHERE id NOT IN
( SELECT id
FROM rezervacije r, študenti m
WHERE m.id = r.študent_id);

```

b) Izpišemo priimke študentov, ki so si rezervirali revijo 1:

```

SELECT m.priimek
FROM študenti m, rezervacije r
WHERE r.revija_id=1 and r.študent_id= m.id;

```

Rešitev z uporabo *IN*:

```

SELECT m.priimek
FROM študenti m
WHERE m.id IN (
SELECT r.študent_id
FROM rezervacije r
WHERE r.revija_id =1);

```

Rešitev z uporabo *EXISTS*:

```

SELECT m.priimek
FROM študenti m
WHERE EXISTS ( SELECT *
FROM rezervacije r
WHERE r.revija_id=1 AND r.študent_id = m.id);

```

Pogoj za WHERE EXISTS je izpolnjen, če je vsaj ena rezervacija revije z *id* 1 študenta z *id* enakim *m.id*.

3.5 Funkcije: *COUNT*, *AVG*, *SUM*, *MIN*, *MAX*

Funkcije vrnejo:

- COUNT število vrstic,
- AVG povprečje,
- SUM vsoto,
- MIN, MAX: vrne najmanjšo, največjo vrednost vrednosti kolon v množici,
- DISTINCT lahko dodamo skupinske funkcije :COUNT, AVG, SUM, MIN, MAX za izračun samo različnih vrednosti.

K standardnemu bloku za poizvedbe lahko dodamo še *GROUP BY* in *HAVING*.

```
SELECT imena_kolon
FROM ime_tabele
WHERE pogoj
GROUP BY imena_kolon
HAVING pogoj
ORDER BY imena_kolon;
```

Če dodamo še kolono z *GROUP BY ime kolone*, vrednosti katere bomo uporabili za tvorjenje skupin, potem skupinske funkcije izračunajo vrednosti po posameznih skupinah.

Za pogoj, ki uporablja skupinsko funkcijo uporabimo *HAVING*.

3.5.1 Zbirka nalog

a) Koliko je študentov v tabeli?

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

b) Število rezervacij za vsako revijo, ki izide četrtletno?

```
SELECT l.id, COUNT(*) AS skupaj_rezervacij
FROM revije l, rezervacije r
WHERE r.revija_id=l.id AND l.pogostost=4
GROUP BY l.id;
```

c) Kakšna je povprečna starost vseh študentov z oceno 10?

```
SELECT AVG (m.starost)
FROM študenti m
WHERE m.ocena=10;
```


d) Kakšna je povprečje različnih starosti študentov z oceno 10?

```
SELECT AVG(DISTINCT m. starost)
FROM študenti m
WHERE m.ocena=10;
```

e) Poiščimo ime in priimek najstarejšega študenta:

```
SELECT priimek, MAX(starost)
FROM študenti ;
```

Poizvedba vrne napačen rezultat! Vrne prvi priimek v tabeli in največjo starost!

Poleg skupinske funkcije ne sme biti v poizvedbi še posamičen atribut, razen če je tabela urejena po skupinah po tem atributu s stavkom *GROUP BY*.

f) Za vsako oceno študenta, poiščimo starost najmlajšega študenta, ki je starejši od 18 let.

```
SELECT m.ocena, MIN(m.starost)
FROM študenti m
WHERE m.starost >= 18
GROUP BY m.ocena;
```

g) Poiščimo najstarejše študente:

Ker skupinskih funkcij ne moremo gnezditi v podatkovni bazi *MySQL*, izdelamo vgnezdenepoizvedbe.

```
SELECT m. priimek, m.starost
FROM študenti m
WHERE m.starost =
-- poišče max. starost
( SELECT MAX(m2.starost)
FROM študenti m2);
```

h) Poiščimo starost najmlajšega študenta s starostjo ≥ 18 , za vsako oceno z vsaj 2 študentoma:

```
SELECT m.ocena, MIN(m.starost), count(*) as število_študentov
FROM študenti m
WHERE m.starost >= 18
GROUP BY m.ocena
HAVING count(*) > 1;
```

i) Poiščimo največjo poprečno starost študentov, ki imajo enako oceno.

Ker skupinskih funkcij ne moremo gnezditi v podatkovni bazi *MySQL*, lahko uporabimo poglede (angl. view).

```
CREATE OR REPLACE VIEW poprecna_starost as
SELECT ocena, (AVG(starost)) as poprecje_starost
FROM študenti
GROUP BY ocena;
```

```
SELECT * from poprecna_starost;
```

```
SELECT MAX(poprecje_starost)
FROM poprecna_starost;
```

3.6 Brisanje in spreminjanje podatkov

Zapise v bazi podatkov lahko spremenimo z *UPDATE* in jih izbrišemo z *DELETE*.

```
UPDATE ime_tabele
SET kolona1 = vrednost1, kolona2 = vrednost2, ...
WHERE pogoji;
```

```
DELETE FROM ime_tabele WHERE pogoji;
```

3.6.1 Zbirka nalog

a) Spremenimo priimek študenta z id=3:

```
UPDATE študenti SET priimek = 'nov priimek' WHERE (id = '3');
```

b) Zbrišemo študente z oceno 9:

```
SET SQL_SAFE_UPDATES = 0;
DELETE FROM študenti
WHERE ocena=9;
```

3.7 Stiki: *LEFT, RIGHT, FULL OUTER, INNER JOIN*

Poznamo več vrst stikov (angl. joins) . Opišemo jih v bloku poizvedbe za imeni tabel.

```
SELECT seznam kolon
FROM tabela
[INNER | {LEFT | RIGHT | FULL } OUTER] JOIN tabla
ON kvalifikacijski seznam
WHERE pogoj;...
```

INNER JOIN je privzet stik.

LEFT OUTER JOIN vrne vse ujemajoče se vrstice, plus vse neuparjene vrstice iz tabele na levo od stika (od besedila *LEFT OUTER JOIN*) in v neuparjena polja vstavi vrednost *null*.

RIGHT OUTER JOIN vrne vse ujemajoče se vrstice, plus vse neuparjene vrstice iz tabele desno od stika in v neuparjena polja vstavi vrednost *null*.

FULL OUTER JOIN je samo v standardnem *SQL* in ga *MySQL* ne pozna. Vrne vse ujemajoče se vrstice, plus vse neuparjene vrstice iz tabele desno in levo od stika in LEVO od stika in v neuparjena polja vstavi vrednost *null*.

3.7.1 Zbirka nalog

a) Izpišimo študente in revije, ki so jih rezervirali:

```
SELECT m.id, m.priimek, r.revija_id
FROM študenti m INNER JOIN rezervacije r
ON m.id = r.študent_id;
```

b) Želimo izpis vseh študentov in revij, če so jih rezervirali. Torej želimo izpis tudi študentov, ki niso rezervirali nobene revije.

```
SELECT m.id, m.priimek, r.revija_id
FROM študenti m LEFT OUTER JOIN rezervacije r
ON m.id = r.študenti_id;
```

c) Izdelamo hierarhijo oddelkov.

Poleg imena oddelka, hranimo še podatke v kateri oddelek organizacijsko sodi. Tvorimo tabelo oddelki in nato tvorimo poizvedbo, ki za izbran oddelek prikaže še hierarhijo oddelkov v katere sodi.

```
CREATE TABLE oddelki (
  id INT(11) NOT NULL,
  ime VARCHAR(45) NULL,
  id_starša INT(11) NULL,
  PRIMARY KEY (id));
```

```
SELECT t1.ime AS lev1, t2.ime as lev2, t3.ime as lev3,
t4.ime as lev4
FROM oddelki AS t1
LEFT JOIN oddelki AS t2 ON t2.id_starša= t1.id
LEFT JOIN oddelki AS t3 ON t3.id_starša= t2.id
LEFT JOIN oddelki AS t4 ON t4.id_starša= t3.id
WHERE t1.ime= 'skriptarnica';
```

- d) Z uporabo *RIGHT OUTER JOIN* vrnimo vse študente in revije, ki so jih rezervirali. Če študent ni rezerviral nobene revije bo vrednost *r.revija_id* enaka null.

```
SELECT m.id, m.priimek, r.revija_id
FROM rezervacije r RIGHT OUTER JOIN študenti m
ON m.id = r.študenti_id;
```

- e) Izpišimo vse revije in vse rezervacije s standardnim jezikom *SQL*.

Če študent ni rezerviral nobene revije, je polje *r.revija_id* null.

```
SELECT m.id, m.priimek, r.revija_id
FROM študenti m FULL OUTER JOIN rezervacije r
ON m.id = r.študenti_id;
```

- f) Naredimo *FULL OUTER JOIN* v *MySQL*.

Ker podatkovna baza *MySQL* ne pozna *FULL OUTER JOIN* naredimo unijo poizvedb:

```
SELECT imena kolon
FROM tabela1
LEFT JOIN tabela2
ON tabela1.ime_kolone = tabela2.ime_kolone
UNION
SELECT imena kolon
FROM tabela1
RIGHT JOIN tabela2
ON tabela1.ime_kolone = tabela2.ime_kolone;
```

3.8 Količnik in SQL

Količnik je rezultat poizvedb, kjer poizvedujemo o vseh. Na primer:

- vsi študenti, ki so rezervirali revijo, ki izhaja mesečno in revijo, ki izhaja vsako četrtoletje,
- vsi študenti, ki so rezervirali revijo 2 in 3.

3.8.1 Zbirka nalog

- a) Izpišimo študente, ki so rezervirali vse revije. Lahko jih poiščemo s štejem rezervacij.

```
SELECT m.priimek
FROM študenti m, rezervacije r
WHERE m.id = r.študent_id
GROUP BY m.priimek, m.id
```

```
HAVING COUNT(DISTINCT r.revija_id) =
( SELECT COUNT(*) FROM revije);
```

b) Poiščimo študente, ki so rezervirali reviji z $id=2$ in z $id=3$? Rešitev izdelamo postopoma.

Rezultat zadnje poizvedbe sta študenta z $id=1$ in z $id=2$, če imamo v tabeli rezervacije naslednje zapise:

študent_id	revija_id	dan
1	1	2021-05-20
1	2	
1	3	
2	2	
2	3	
3	3	

Rešitev bomo izdelali postopoma.

Reviji 2 in 3 damo v tabelo *zahtevaneRevije*, ki jo tvorimo z naslednjim stavkom:

```
CREATE TABLE zahtevaneRevije
(id int(5));
```

```
INSERT INTO zahtevaneRevije VALUES(3);
INSERT INTO zahtevaneRevije VALUES(2);
SELECT * FROM zahtevaneRevije;
```

Najprej naredimo kartezični produkt dveh tabel: *študenti* in *zahtevaneRevije*. Kartezični produkt upari vsako vrstico prve tabele z vsako vrstico druge tabele.

```
CREATE OR REPLACE VIEW študentiInZahtevaneRevije AS
SELECT DISTINCT študenti.id as študent_id, zahtevaneRevije.id as revija_id
FROM študenti , zahtevaneRevije ;
```

```
SELECT * FROM študentiInZahtevaneRevije ;
```

Izpis:

_id	revija_id	id
1		3
1		2
2		3
2		2
3		3
3		2
4		3
4		2

Izpišemo vse vrstice *študent_id* in *revija_id* iz pogleda *študentiInZahtevaneRevij*, ki se ne pojavijo v rezervacijah.

```
CREATE OR REPLACE VIEW študentiInNevzeteZahtevaneRevije AS
SELECT * FROM študentiInZahtevaneRevije
WHERE NOT EXISTS (
  SELECT *
  FROM rezervacije r
  WHERE študentiInZahtevaneRevije .id = r.študent_id AND
  študentiInZahtevaneRevije .revija_id = r.revija_id);

SELECT * FROM študentiInNevzeteZahtevaneRevije;
```

Študenti, ki niso rezervirali zahtevanih revij:

```
SELECT id FROM študentiInNevzeteZahtevaneRevije;
```

Izpis:
študent_id
3
4
5
5

Dobimo rezultat za količnik. Izpišemo še vse študente, ki niso v pogledu *študentiInNevzeteZahtevaneRevije*.

```
SELECT id FROM študenti
WHERE NOT EXISTS
( SELECT študent_id FROM študentiInNevzeteZahtevaneRevije
  WHERE študenti.id= študentiInNevzeteZahtevaneRevije.študent_id);
```

Izpis:
id
1
2

- c) Kateri študenti so rezervirali reviji z *id*=2 in z *id*=3? Poiščimo rezultat z eno poizvedbo.

Torej izpišemo vsakega študenta, za katerega velja, da ne obstaja zahtevana revija, ki je ni rezerviral ta študent.

```
SELECT DISTINCT r.študent_id
FROM rezervacije AS r
WHERE NOT EXISTS (
```

```

SELECT *
FROM zahtevaneRevije AS y
WHERE NOT EXISTS (
  SELECT *
  FROM rezervacije AS z
  WHERE (z.študent_id=r.študent_id)
  AND (z.revija_id=y.id));

```

Izpis:

študent_id

1

2

- d) Izpišimo študente, ki so rezervirali reviji z $id=2$ in z $id=3$? Ker iščemo študente, ki so rezervirali le dve reviji, lahko zapišemo poizvedbo, ki dvakrat preišče rezervacije.

```

SELECT m.id , m.priimek
FROM študenti m , rezervacije r, rezervacije r1
WHERE r.revija_id=2 AND r.študenti_id =m.id AND
r1. revija_id=3 AND r1.študenti_id =m.id;

```

4 Transakcije

Transakcije so nedeljive - atomarne operacije nad bazo.

Transakcija je en stavek *SQL* ali zaporedje več stavkov *SQL*, ki privedejo bazo iz enega v drugo konsistentno stanje. Posamezna operacija nad podatki je lahko sestavljena iz več ukazov (npr. zmanjšanje vrednosti na enem računu in povečanje na drugem računu). Transakcije povzročijo več sprememb v eni ali več tabelah.

Transakcije so sestavljene iz več ukazov jezika za upravljanje s podatki *SQL DML* (angl. *data management language*) *SELECT*, *INSERT*, *UPDATE*, *DELETE*. Transakcije potrjuje stavek *COMMIT* in razveljavi stavek *ROLLBACK*.

Transakcijo pa sestavlja tudi posamezen ukaz jezika definiranja podatkov *SQL DDL* (angl. *data definition language*), kot sta na primer *CREATE TABLE* ali *ALTER TABLE*, ki je avtomatično potrjen in implicitno konča transakcijo.

Za transakcijo velja, da se izvedejo vse spremembe ali nobena in se mora vedno v celoti izvesti. Če pride do napake in se del sprememb ne izvede, se vse zavržejo.

Transakcija se začne, ko se začne izvajati prvi *DDL* ali *DML* ukaz in se konča, ko se zgodi en izmed naslednjih dogodkov:

1. Ukaz *COMMIT*, ki potrdi spremembe transakcije ali ukaz *ROLLBACK*, ki zavrže spremembe.
2. Konča se *DDL* ukaz.
3. Zgodi se mrtva zanka (angl. *deadlock*), ko se ne more nadaljevati nobena transakcija in vse čakajo, da se izvede potrditev v drugi transakciji.
4. Izhod iz urejevalnika *SQL* samodejno konča transakcijo in jo potrdi ali zavrže, tako kot je bilo določeno z ukazom *AUTOCOMMIT*. Če je *AUTOCOMMIT* 1, potem se potrditev izvede samodejno za vsakim ukazom, če pa je 0 se potrditev ne izvede samodejno.

```
SET AUTOCOMMIT=1;  
SET AUTOCOMMIT=0;
```

5. Napake: sistemske, prekinitev napajanja....

Postavimo lahko značko z ukazom *SAVEPOINT*, ki definira točko, na katero se lahko sklicujemo, ko želimo zavreči samo določene spremembe in ne vse spremembe.

4.1 Naloga

Uporabimo SAVEPOINT za vstavljanje in spreminjanje podatkov.

Tvorimo tabelo :

```
CREATE TABLE oddelki (  
  šifra_oddelka INT(11) NOT NULL,  
  ime_oddelka VARCHAR(20) NULL,  
  mesto VARCHAR(20) NULL,  
  PRIMARY KEY (id));  
INSERT INTO oddelki VALUES(50,'Oddelek za rač.','Maribor');
```

SAVEPOINT vrivanje_končano;

```
UPDATE oddelki  
SET ime_oddelka = 'Oddelek za računalništvo'  
WHERE šifra_oddelka=50;
```

ROLLBACK TO vrivanje_končano;

Zavržemo spremembe vseh stavkov nazaj do točke shranjevanja *vrivanje_končano*.
Ime oddelka 50 je *Oddelek za rač.* .

```
UPDATE oddelki  
SET ime_oddelka = 'Oddelek za računalništvo'  
WHERE šifra_oddelka=50;  
COMMIT;
```

Potrdili smo ukaz in ime oddelka 50 je *'Oddelek za računalništvo'*.

5 Prožilci

Prožilec je v *MySQL* zaporedje stavkov *SQL*, ki se nahajajo v sistemskem katalogu. Je vrsta shranjene procedure, ki se samodejno prikliče kot odziv na dogodek: vrivanje, ažuriranje ali brisanje. Vsak prožilec je povezan s tabelo, ki se aktivira ob stavku *SQL DML: INSERT, UPDATE ali DELETE*.

Prožilec izvede kodo zapisano v prožilcu med stavkoma *BEGIN* in *END*, takoj ko so izpolnjeni zapisani pogoji proženja pred ali po izvedenem stavku za vrivanje/spreminjanje/brisanje (*BEFORE/AFTER INSERT/UPDATE/DELETE*).

Poznamo dva tipa prožilcev: prožilce na nivoju vrstic in prožilce na nivoju stavkov.

Prožilci na nivoju vrstic se prožijo za vsako vrstico, ki je vrinjena, spremenjena ali izbrisana. Ob vnosu desetih vrstic podatkov v tabelo se prožilec desetkrat proži. Prednost vrstičnih prožilcev je, da izvedejo stavke prožilca samo za vrstice, ki so se zares spremenile. Pogosto moramo za vsako vrsto dogodka napisati svoj prožilec, ki pa je največkrat zelo podoben ostalim.

Prožilci na nivoju stavkov: Prožijo se enkrat za celotno transakcijo, ne glede na število vrinjenih, spremenjenih ali izbranih vrstic.

ISO standard:

```
CREATE TRIGGER
BEFORE | AFTER dogodek ON tabela
[REFERENCING sinonimi za stare ali nove vrednosti]
[FOR EACH ROW]
[WHEN (pogoj)] -- pogoj za vrstico (kot WHERE)
BEGIN
-- telo prožilca
END;
```

Za prožilce v podatkovni bazi *MySQL* velja naslednje.

Samo en dogodek je možen za vsak prožilec ob vrivanju, spremembi ali brisanju (*INSERT, UPDATE, DELETE*).

Spremeniti moramo ločilo za konec stavka, namesto podpičja lahko določimo novo ločilo, kot na primer // s stavkom *DELIMITER //*.

Lokalne spremenljivke moramo definirati znotraj *BEGIN/END* kot na primer celo število *x*:

```
DECLARE x INTEGER;
```

MySQL uporablja stavek *RETURNS* namesto *RETURN*.

MySQL ne pozna stavčnih prožilcev, ni sinonimov za *OLD* in *NEW*.

Ne uporabljamo dvopičja pred *OLD* in *NEW*.

MySQL ne pozna *WHEN* sklopa (lahko pa uporabimo proceduralni *IF ... END*).

Ovisno od vrste dogodka se lahko sklicujemo na:

– stare vrednosti pred spremembo (*OLD*) za prožilce, ki jih prožita stavka *DELETE* in *UPDATE*,

– nove vrednosti po spremembi (*NEW*) za prožilce, ki jih prožita stavka *INSERT* in *UPDATE*.

MySQL podpira samo prožilce na nivoju vrstic.

Prožilci omogočajo enostaven način za načrtovanje opravil. Lahko jih uporabimo za nadzor nad spremembami podatkov v tabelah ali izvedbo opravil ob napakah, ki se zgodijo na bazi. Prožilci nudijo tudi drug način za preverjanje integritete podatkov.

Slabosti prožilcev:

Prožilec lahko nudi le delno validacijo in ne celotne. Lahko upravlja naslednje omejitve: *NOT NULL*, *UNIQUE*, *CHECK* in omejitve na tujih ključih.

Prožilec lahko oteži odpravljanje težav, ker se izvede samodejno.

Prožilci povečajo delo na strežniku *MySQL*.

5.1 Zbirka nalog

- a) Izdelajmo prožilec, ki beleži vse spremembe v tabeli študenti.

Najprej naredimo novo tabelo, ki bo hranila podatke o spremembah:

```
CREATE TABLE študenti_nadzor (  
    aktivnost VARCHAR(15) NULL,  
    id INT(11) NULL,  
    priimek VARCHAR(20) NULL,  
    ocena INT(11) NULL,  
    starost INT(11) NULL,  
    datum_spremembe DATETIME(6) NULL);
```

Ob brisanju, vrivanju in ažuriranju se spremembe shranijo v tabelo *študenti_nadzor*.

Prožilec za hranjenje sprememb ob spremembi podatkov:

```
DELIMITER $$  
CREATE DEFINER = CURRENT_USER TRIGGER `test`.`študenti_BEFORE_UPDATE`  
BEFORE UPDATE ON `študenti` FOR EACH ROW  
BEGIN  
    INSERT INTO študenti_nadzor  
    SET aktivnost= 'update',  
        datum_spremembe= now(),  
        priimek= OLD.priimek,  
        ocena=OLD.ocena,  
        starost=OLD.starost;  
END$$  
DELIMITER ;
```

Preverimo delovanje prožilca:

```
SELECT * FROM študenti ;  
UPDATE študenti set priimek='d', ocena= 8, starost=35 WHERE id=1;  
SELECT * FROM študenti ;
```

```
SELECT * FROM študenti _nadzor;
```

Prožilec ne more spremeniti podatkov v tabeli nad katero dela. Če želimo v zgornjem prožilcu tudi vstaviti nov zapis v tabelo *študenti*, dobimo naslednjo napako:

```
09:24:19      INSERT INTO študenti VALUES (666, 'abc',7,35)      Error   Code:   1442.  
Can't UPDATE table 'študenti ' in stored function/trigger because it is already used by  
statement which invoked this stored function/trigger.      0.016 sec
```

Prožilec za hranjenje sprememb ob vrivanju:

```
DELIMITER $$  
CREATE DEFINER = CURRENT_USER TRIGGER `test`.`študenti _BEFORE_INSERT` BEFORE  
INSERT ON `študenti ` FOR EACH ROW  
BEGIN  
INSERT INTO študenti _nadzor  
SET aktivnost= 'insert',  
datum_spremembe= now(),  
priimek= NEW.priimek,  
ocena=NEW.ocena,  
starost=NEW.starost;  
if new.ocena=0 then  
set new.ocena=1;  
END IF;  
END$$  
DELIMITER ;
```

Preverimo delovanje prožilca:

```
SELECT * FROM študenti _nadzor;  
INSRT INTO študenti VALUES(12,'p',9,24);  
SELECT * FROM študenti _nadzor;
```

Prožilec za hranjenje sprememb ob brisanju:

```
DELIMITER $$  
CREATE DEFINER=`root`@`localhost` TRIGGER `test`.`študenti _BEFORE_DELETE`  
BEFORE DELETE ON `študenti ` FOR EACH ROW  
BEGIN  
INSERT INTO študenti _nadzor  
SET aktivnost= 'delete',  
datum_spremembe= now(),  
priimek= OLD.priimek,  
ocena=OLD.ocena,  
starost=OLD.starost;  
END$$  
DELIMITER ;
```

Preverimo delovanje prožilca:

```
SELECT * FROM študenti _nadzor;  
DELETE FROM študenti WHERE id=12;  
SELECT * FROM študenti _nadzor;
```

- b) Izdelamo prožilec, ki za vsako vrstico računa (postavko) za izpite poišče ceno za izpite in popravi vrednost celotnega računa. Različni izpiti imajo različne cene.

```
DROP TABLE računi;  
DROP TABLE postavke;
```

```
CREATE TABLE računi  
( id_rac INT(4),  
  vrednost INT(4));
```

```
CREATE TABLE postavke  
( id_rac INT(4),  
  vrednost INT(4),  
  izpit_id INT(4));
```

```
CREATE TABLE izpiti  
( id_izpita INT(4),  
  cena INT(4));
```

```
set AUTOCOMMIT=0;  
set SQL_SAFE_UPDATES=0;
```

```
DROP TRIGGER post_b;  
DELIMITER //  
CREATE TRIGGER post_b BEFORE INSERT ON postavke FOR EACH ROW  
begin  
SET NEW.vrednost= ( SELECT cena FROM izpiti WHERE id_izpita= NEW.izpit_id);  
UPDATE računi SET vrednost = vrednost +NEW. vrednost  
WHERE id_rac = NEW.id_rac;  
END; //
```

```
DELIMITER ;  
show errors;
```

```
DELETE FROM izpiti;  
DELETE FROM računi;  
DELETE FROM postavke;
```

```
INSERT INTO računi VALUES (1,0);  
INSERT INTO izpiti VALUES (1,5);
```

```
set SQL_SAFE_UPDATES=0;
INSERT INTO postavke VALUES (1,NULL,NULL,2,1);
```

```
SELECT * FROM postavke;
SELECT * FROM računi;
```

- c) Izdelamo prožilec, ki ob spremembi številke računa popravi tudi številke računov v vseh postavkah.

```
DELIMITER //
CREATE TRIGGER rac_UPDATE BEFORE UPDATE
ON računi
FOR EACH ROW BEGIN
UPDATE postavke SET
id_rac = NEW.id_rac
WHERE id_rac= OLD.id_rac;
END;//
DELIMITER ;
```

Preverimo delovanje prožilca:

```
SET SQL_SAFE_UPDATES = 0;
UPDATE računi SET id_rac=400 WHERE id_rac=1;
SELECT * FROM postavke;
```

6 Procedure

V bazi podatkov je potrebno večkrat izvajati operacije, kot so čiščenje baze podatkov, mesečne obdelava obračunov ali ustvarjanje nove entitete z več privzetimi vnosi. Takšna opravila vključujejo izvajanje več poizvedb, ki jih lahko združimo v eno nalogo in jo izvedemo. V ta namen lahko izdelamo shranjeno proceduro (angl. Stored Procedure). Shranjena procedura *MySQL* je sestavljena iz vnaprej sestavljene kode *SQL*, ki jo je mogoče izvesti za izvajanje več nalog skupaj z izvajanjem nekaterih logičnih operacij.

MySQL omogoča uporabo kazalcev znotraj shranjenih procedur. Kazalci omogočajo samo branje vrednosti in ne spreminjanja [4]. Kazalci omogočajo prehod po vrsticah tabele samo v eni smeri in ne morejo preskočiti vrstic.

Sintaksa je podobna kot vrinjen SQL. Kazalce je potrebno deklarirati:
DECLARE kazalec CURSOR FOR SELECT ...

Najprej deklariramo spremenljivke, nato deklariramo kazalce in nato upravljalec. Deklarirajmo še upravljalec, ki bo ob dogodku, ko kazalec ne bo več vrnil nobene vrstice postavil spremenljivko konec na resnično (TRUE):
DECLARE CONTINUE HANDLER FOR NOT FOUND SET konec = TRUE;

Na začetku je kazalec potrebno odpreti in na koncu zapreti:
OPEN kazalec;
CLOSE kazalec;

Kazalec vrne naslednjo vrstico s stavkom:
FETCH kazalec INTO spremenljivke;

6.1 Zbirka nalog

- a) Tvorimo proceduro za vnos privzetih vlog študentov.

Najprej ustvarimo tabelo:

```
CREATE TABLE vloge_študentov(  
  id INT NOT NULL,  
  vloga VARCHAR(15) NULL,  
  id_študenta INT NULL,  
  PRIMARY KEY (id));
```

Naredimo proceduro, ki uporabniku z *id vhodni_id*, dodeli vse privzete vloge, ki so študent, demonstrator in študent tutor.

Izdelajmo proceduro, ki dodeli privzete vloge za študenta z *id= vhodni_id*, ki je vhodni parameter.

```

DELIMITER //
CREATE OR REPLACE PROCEDURE vrini_privzete_vloge(vhodni_id int)
BEGIN
    DECLARE zadnji int;

    SELECT max(id) INTO zadnji FROM vloge_študentov ;
    INSERT INTO vloge_študentov VALUES
    (zadni+1, 'demonstrator', vhodni_id),
    (zadni+2, 'študent', vhodni_id);
    (zadni+3, 'študent tutor', vhodni_id);

    END //
DELIMITER ;

```

Klicanje procedure:

```
call vrini_privzete_vloge (1112222);
```

Brisanje procedure ni potrebno, če jo ustvarimo s *CREATE OR REPLACE PROCEDURE*. Z *DECLARE* definiramo spremenljivke. Proceduro zberemo s stavkom *DROP PROCEDURE*:

```
DROP PROCEDURE vrini_privzete_vloge;
```

b) Procedura vrne število študentov

Zapišimo proceduro, ki vrne število študentov, ki imajo *vlogo vhodna_vloga*.

```

DROP PROCEDURE štej_uporabnike;
DELIMITER //
CREATE PROCEDURE štej_uporabnike vhodna_vloga varchar(15))
BEGIN
    DECLARE števec int ;

    SELECT count(*) into števec from vloge_študentov where vloga = vhodna_vloga;
    SELECT števec;

    END //
DELIMITER ;

```

Klicanje procedure:

```
call štej_uporabnike ('demonstrator');
```

c) Procedura za izpis opisnih ocen

Najprej naredimo tabelo za hranjenje ocen in vrinimo podatke.


```

CREATE TABLE rezultati (
    študent_id INT NOT NULL,
    odstotek INT NULL,
    ocena VARCHAR(10) NULL,
    PRIMARY KEY (`študent_id`));

INSERT INTO rezultati VALUES (1,50,null), (2,60,null), (3,34,null), (4,70,null), (5,43,null);

CREATE PROCEDURE opravil(vhodni_id int)
BEGIN
    DECLARE todstotek int;

    SELECT odstotek into todstotek from rezultati where student_id = vhodni_id;;

    CASE
        WHEN todstotek <50 THEN SELECT 'negativno';
        WHEN todstotek <60 THEN SELECT 'zadostno';
        WHEN todstotek <70 THEN SELECT 'dobro';
        WHEN todstotek <90 THEN SELECT 'prav dobro';
        ELSE SELECT 'odlično';
    END CASE;

END;

```

d) Vrinimo rezultat preverjanja znanja v vse zapise v tabeli rezultati.

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `ocenjevanje`()
BEGIN
    DECLARE done int default false;
    DECLARE tštudent_id int;
    DECLARE todstotek int;

    DECLARE kazalec CURSOR FOR SELECT student_id, odstotek from rezultati;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN kazalec;

zanka: LOOP
    FETCH kazalec INTO tštudent_id, todstotek;
    IF done THEN LEAVE zanka; END IF;
    IF todstotek <50 THEN
        UPDATE rezultati SET ocena='ni opravil' WHERE student_id = tštudent_id;
    ELSEIF todstotek <60 THEN
        UPDATE rezultati SET ocena='zadostno' WHERE student_id = tštudent_id;
    END IF;
END LOOP;

```

```
ELSEIF todstotek <70 THEN
    UPDATE rezultati SET ocena='dobro' WHERE student_id = tstudent_id;
ELSEIF todstotek <90 THEN
    UPDATE rezultati SET ocena='prav dobro' WHERE student_id = tstudent_id;
ELSE
    UPDATE rezultati SET ocena='odlično' WHERE student_id = tstudent_id;
END IF;
END LOOP;

CLOSE kazalec;

END;
```

7 MySQL in PHP

PHP je strežniški skriptni jezik za ustvarjanje dinamičnih in interaktivnih spletnih strani. V *PHP* kodi se najprej prijavimo na bazo in nato delamo s podatki.

PHP kodo damo v datoteko s končnico *.php* in datoteko prenesemo na *www* direktorij *UWAMP*. *UWAMP* je spletno razvojno okolje v okolju oken, ki omogoča ustvarjanje spletnih aplikacij z *Apache2*, *PHP* in bazo podatkov *MySQL*.

Tudi v *PHP* je najprej potrebno vzpostaviti povezavo z bazo. Nato lahko naredimo več poizvedb vstavljanja podatkov, spreminjanja, brisanja in iskanja podatkov. Na koncu povezavo z bazo zapremo.

```
// Povezava z bazo
```

```
$povezava = new mysqli($streznik,$uporabnik,$geslo,$baza);
```

```
// Poizvedbe : vrivanje zapisa v bazo
```

```
$povezava->query("INSERT INTO ocene_testa(vpisna,ocena) VALUES ('" . $vpisna.  
"', '" . $ocena."');");
```

```
// Prekinemo povezavo z bazo
```

```
$povezava->close();
```

7.1 Zbirka nalog

- V *PHP* izdelajmo primer vseh operacij *CRUD* (angl. CREATE, RETRIEVE, UPDATE, DELETE) za vnos ocen študentov.

Najprej tvorimo tabelo ocen:

```
CREATE TABLE ocene_testa (  
  vpisna VARCHAR(11) NULL,  
  ocena VARCHAR(11) NULL);
```

Izdelamo *obrazec.html* in *baza.php*.

In obrazec poženemo s spletnega urejevalnika: <http://localhost/obrazec.html>

← → ↻ ⓘ localhost/obrazec.html

Vnesi podatke v polja za vpis in/ali spreminjanje in/ali brisanje.

Vnesi

Vpisna ocena

Spremeni

Vpisna ocena

Briši

Vpisna

Obrazec:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Obrazec</title>
</head>
Vnesi podatke v polja za vpis in/ali spreminjanje in/ali brisanje.
<body>
  <form action="baza.php" method="POST">
    <br>
    <h2>Vnesi </h2>
    <label for="Vpisna">Vpisna</label>
    <input type="text" name="vpisna" id="vpisna">
    <label for="ocena">ocena</label>
    <input type="text" name="ocena" id="ocena">
    <br>
    <h2>Spremeni</h2>
    <label for="Vpisna">Vpisna</label>
    <input type="text" name="vpisna1" id="vpisna1">
    <label for="ocena">ocena</label>
    <input type="text" name="ocena1" id="ocena1">
    <br>
    <h2>Briši</h2>
    <label for="Vpisna">Vpisna</label>
    <input type="text" name="vpisna2" id="vpisna2">
    <button type="submit">Pošlji</button>
  </form>
</body>
</html>
```

Zapišemo še kodo za vrivanje dobljenih podatkov ocen v bazo in izpis podatkov, brisanje in spreminjanje podatkov:

```

baza.php:
<?php
//CREATE TABLE `test`.`ocene_testa` (
// `vpisna` VARCHAR(11)) NOT NULL,
// `ocena` VARCHAR(11) NULL,
// PRIMARY KEY (`vpisna`));
$streznik = "localhost";
$uporabnik= "root";
$geslo = "root";
$baza ="test";
$vpisna = "";
$ocena = "";
$vpisna1 = "";
$vpisna2 = "";
// Povezava z bazo
$povezava = new mysqli($streznik,$uporabnik,$geslo,$baza);
// Preverimo povezavo
if ($povezava->connect_error) {
    die("Connection failed: " . $povezava->connect_error);
}
// VRIVANJE zapisa v bazo
if (isset($_POST["vpisna"])) {
    $vpisna = $_POST["vpisna"];
}
if (isset($_POST["ocena"])) {
    $ocena = $_POST["ocena"];
}
if ($vpisna && $ocena) {
    $povezava->query("INSERT INTO ocene_testa(vpisna,ocena) VALUES ('" . $vpisna. "',
    '" . $ocena."');");
}
// SPREMINJANJE zapisa
if (isset($_POST["vpisna1"])) {
    $vpisna1 = $_POST["vpisna1"];
}
if (isset($_POST["ocena1"])) {
    $ocena1 = $_POST["ocena1"];
}
if ($vpisna1 && $ocena1) {
    $povezava->query("UPDATE ocene_testa SET ocena= '" . $ocena1. "' WHERE vpisna= '" .
    $vpisna1. "'");
}
// BRISANJE zapisa
if (isset($_POST["vpisna2"])) {
    $vpisna2 = $_POST["vpisna2"];
}

```

```

if ($vpisna2) {
    $povezava->query("DELETE FROM ocene_testa WHEREvpisna =" . $vpisna2 . " "; );
}
// POIZVEDBA
$sql = "SELECT * FROM ocene_testa ORDER BY ocena;";
$odg = $povezava->query($sql);

if ($odg->num_rows > 0) {

print("<table><thead><tr><td><strong>Vpisna<strong></td><td><strong>Ocena</stro
ng></td></tr></thead>");
    foreach ($odg as $index => $od) {
        print("<tr>"); print("<td>" . $od["vpisna"] . "</td>");
        print("<td>" . $od["ocena"] . "</td>"); print("</tr>");
    }
    print("</table>");
} else {
    echo "0 results";
}
// Prekinemo povezavo z bazo
$povezava->close();
?>

```

8 *NoSQL* podatkovna baza *MongoDB*

NoSQL podatkovne baze ne vsebujejo relacijskega podatkovnega modela, podatki so nerelacijski in kot poizvedovalni jezik po poizvedbah ne uporabljamo *SQL*-ja. Rešiti poskušajo problematiko razširljivosti, prilagodljivosti in dostopnosti. *MongoDB* [6-10] je *NoSQL* podatkovna baza. *MongoDB* je dokumenti sistem za upravljanje podatkovnih baz (SUPB), ki ni relacijski in je napisan v *C++*.

Baza podatkov v *MongoDB* je vsebnik za zbirke dokumentov. Podatkovni strežnik *MongoDB* shranjuje več baz podatkov in vsaka baza podatkov dobi svoj nabor datotek. Baza podatkov vsebuje več zbirk (angl. *collection*). Pojem tabele v relacijskih bazah je ekvivalenten pojmu zbirka, pojem vrstica tabele pa *JSON/BSON* dokumentu in pojem kolone *JSON/BSON* polju v dokumentu.

Dokument - je zapis v zbirki *MongoDB*. Dokument je sestavljen iz imen polj in vrednosti.

Polje - je par ime-vrednost v dokumentu. Dokument ima nič ali več polj.

MongoDB ne uporablja stikov podatkov. Namesto stika omogoča gnezdenje dokumentov (angl. *embedded documents*) in stike v uporabniškem programu.

V *MongoDB* je prvi osnovni korak vzpostavitev baze podatkov (angl. *database*) in zbirke (angl. *collections*). Baza podatkov se uporablja za shranjevanje več zbirk, zbirka pa hrani vse dokumente (angl. *documents*). Dokumenti vsebujejo ustrezna imena polj in vrednosti polj.

Če primarnega ključa dokumenta (*_id*) ne dodamo, ga *MongoDB* samodejno ustvari. *_id* je primarni ključ dokumenta:

Enolični identifikator -id je 12 bitna šestnajstiško število za vsak dokument v zbirki. 12 bitov se sestavi po naslednji shemi:

- 4 bitna časovna značka,
- 3 bitna informacija o ID računalnika,
- 2 bitna informacija o ID procesa,
- 3 bitni prirastek.

Vsakemu dokumentu brez id *MongoDB* doda enolični identifikator.

8.1 Enostavne poizvedbe v *MongoDB*

Ustvarimo bazo študenti z ukazom *use študenti*. Če baza podatkov s tem imenom ne obstaja jo *MongoDB* ustvari, sicer vrne obstoječo bazo.

```
>use ime_baze
```

Prikažimo trenutno bazo:

```
> db
```

Prikaz vseh baz:

```
> show dbs
```

Zbrišemo bazo:

```
> use ime_baze  
> db.dropDatabase()
```

MongoDB naredi zbirko samodejno ob vrivanju dokumenta, če zbirka ne obstaja:

```
db.ime_zbirke.insert({dokument})
```

Prikažemo zbirke:

```
> show collections  
Studenti
```

Brisanje zbirke:

```
db.ime_zbirke.drop()
```

Izpišemo vse dokumente iz zbirke:

```
db.ime_zbirke.find().pretty()
```

Izpišemo samo en dokument:

```
db.ime_zbirke.findOne()
```

Prikažemo lahko samo določeno število dokumentov:

```
db.ime_zbirke.find().limit(NUMBER)
```

Preskočimo lahko določeno število dokumentov in nato prikažemo naslednje:

```
db.ime_zbirke.find().skip(NUMBER)
```

Prikazana polja dokumenta označimo z 1, neprikazana v izpisu pa z 0:

```
db.ime_zbirke.find({}, {polje1: 1, polje2: 0})
```

Urejanje izpisa naraščajoče:

```
db.ime_zbirke.find().sort({polje:1})
```

Urejanje izpisa padajoče:

```
db.ime_zbirke.find().sort({polje:-1})
```

Spremembo podatkov izvedemo z metodo update:

```
db.ime_zbirke.update(pogoji, spremenjeni_podatki)
```

8.1.1 Zbirka nalog

- a) Kreirajte bazo študenti in dokument študenti in izdelajte poizvedbe, spreminjanje in brisanje ter vrivanje dokumentov (angl. CRUD).

Uporabimo *MongoDB* v oblaku. Izberemo *MongoDB Atlas*. Delamo na podatkovni bazi *MongoDB* verzije 4.4.6 v oblaku na spletnem naslovu [11].

Za način povezave s podatkovno bazo MongoDB izberemo »*CONNECT with the mongo shell*« in uporabimo kar *Windows PowerShell*.

Tvorimo *Cluster* in dobimo spodnjo sliko:

Clusters

Find a cluster...

SANDBOX

Cluster0

Version 4.4.6

CONNECT METRICS COLLECTIONS ...

CLUSTER TIER

M0 Sandbox (General)

REGION

AWS / N. Virginia (us-east-1)

TYPE

Replica Set - 3 nodes

LINKED REALM APP

None Linked

Izberemo *Database Access*.


Dodamo uporabnika *test* in mu dodelimo privilegij: »*Read and write to any database*«.

Database Access

Database Users

Custom Roles

+ ADD NEW DATABASE USER

User Name ↕	Authentication Method ↕	MongoDB Roles	Resources	Actions
 test	SCRAM	readWriteAnyDatabase@admin	All Resources	<div><div>EDIT</div><div>DELETE</div></div>

Nato izberemo način povezave z *MongoDB* strežnikom *Network Access*. Izberemo »+Add IP Address« in če želimo prijavo iz poljubnega IP naslova izberemo: »**ALLOW ACCESS FROM ANYWHERE**«.

Network Access

IP Access List

Peering

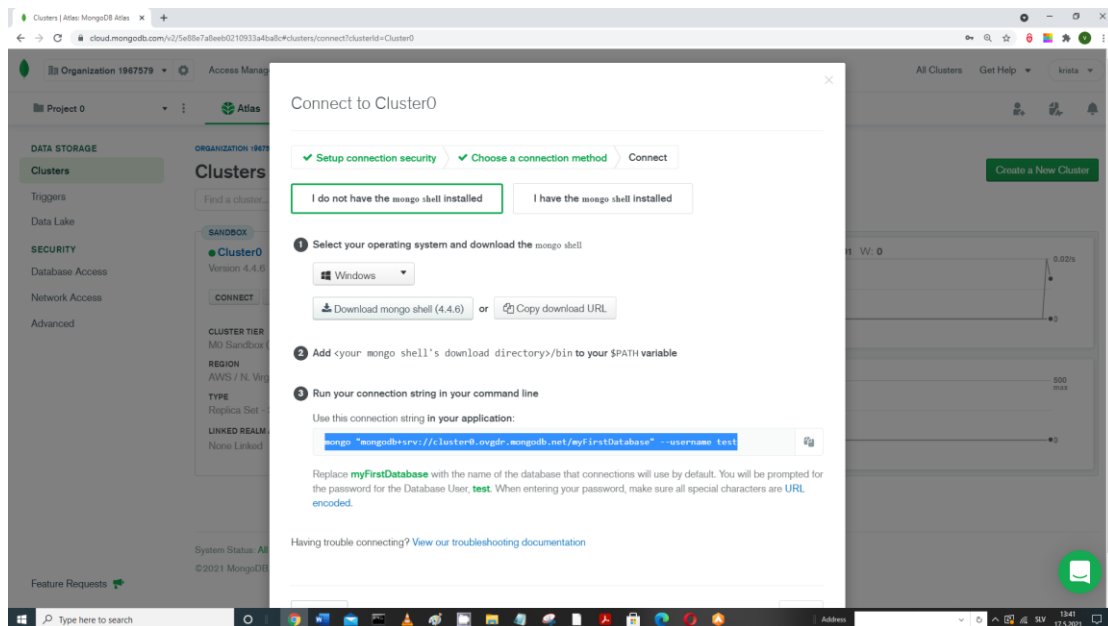
Private Endpoint

+ ADD IP ADDRESS

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
0.0.0.0/0 (includes your current IP address)		<div><div></div>Active</div>	<div><div>EDIT</div><div>DELETE</div></div>

Izberemo *CONNECT* in nato »*CONNECT with the mongo shell*«. Iz spleta si naložimo *Mongo shell 4.4.6* ter dodamo pot v okoljsko spremenljivko *\$PATH* v okolju oken.



Prekopiramo besedilo za povezavo:

mongo "mongodb+srv://cluster0.ovgdr.mongodb.net/MyFirstDatabase --username test

Poženemo Windows PowerShell in prekopiramo povezavo:

mongo "mongodb+srv://cluster0.ovgdr.mongodb.net/test" --username test

Ustvarimo bazo študenti z ukazom *use študenti*. Če baza podatkov s tem imenom ne obstaja jo MongoDB ustvari, sicer vrne obstoječo bazo.

>use študenti

switched to db študenti

Trenutno baze lahko prikažemo z ukazom *db*:

> db

študenti

Vse baze prikažemo z ukazom *show dbs*:

> show dbs

admin 0.000GB

local 3.873GB

nova 0.000GB

test1use 0.051GB

Če želimo zbrisati bazo:

> use študenti

switched to db študenti

> db.dropDatabase()

{ "dropped" : "studenti", "ok" : 1 }

- a) Tvorimo zbirko študenti in izvedimo osnovne operacije tvorjenja, iskanja, spreminjanje in brisanja dokumentov.

MongoDB naredi zbirko samodejno ob vrivanju dokumenta, če zbirka ne obstaja:

```
db.COLLECTION_NAME.insert({DOCUMENT})
```

Redko zbirke ustvarimo z ukazom:

```
db.createCollection(ime_zbirke, možnosti).
```

Možnosti opišejo konfiguracijo za zbirko in so izbirni predmet.

Vrinemo nove dokumente:

```
>db.študenti.insert({"ime":"Miha Novak"})
>db.študenti.insert({ime:"Drugi Novak",starost:25})
>db.študenti.insert({ime:"Tretji Novak", spol:"moški" })
>db.študenti.insert({ime:"četrty", spol:1})
>db.študenti.Mulinsert({ime:"Četrty Novak", spol:"moški"" })
```

Prikažemo zbirke:

```
> show collections
dtudent
študenti
```

Brisanje zbirke:

```
>db.dtudent.drop()
```

Izpišimo vse študente:

```
db.študenti.find().pretty()
```

Izpišemo samo en dokument:

```
db.študenti.findOne()
```

Projekcija izpiše samo imena študentov, zato je za ime 1 in ne izpišimo *id*. Za *id* damo 0.

```
db.študenti.find({}, {"ime":1, "_id":0})
```

Izpišimo samo 2 študenta in uporabimo *limit*:

```
db.ime_zbirke.find().limit(NUMBER)
```

```
db.Študenti.find().limit(2)
```

Izpišimo četrtega študenta:

```
db.študenti.find({}, {"ime":1,"_id":0}).limit(1).skip(3)
```

Izpišimo urejena imena študentov:

```
db.študenti.find({},{"ime":1,"_id":0}).sort({"ime":1})
```

Spremenimo podatke:

```
db.študenti.update({ime: "Četrty Novak"},{$set:{starost:24}})
```

Brisanje študentov:

```
db.študenti.deleteMany({ime: "Četrty Novak"})
```

```
db.študenti.deleteOne({ime: "Četrty Novak"})
```

b) Vrinimo popolne podatke dveh študentov v zbirko študenti.

Najpogostejši podatkovni tipi so *String*, *Integer*, *Boolean*, *Double*, *Arrays* (lista vrednosti), *Timestamp* (beleži čas oddaje ali spremembe dokumenta), *Date*, *Null* za opis praznega polja brez podatkov.

Tip *Object* se uporabi za gnezdenje dokumentov. Vsak dokument je v zavutih oklepajih.

```
db.študentiFakultete1.insert({
  student_id:"2021/1000",
  ime:"Prvi Novak",
  smer:"Računalništvo",
  izpiti:[
    {
      predmet:"Upor. PO",
      ocena:"10",
      datum:"4.4.2021"
    },
    {
      predmet:"Osnove PB",
      ocena:"10",
      datum:"4.4.2021"
    },
    {
      predmet:"Osnove IS",
      ocena:"10",
      datum:"4.4.2021"
    }
  ],
  naslov: {
    mesto:"Maribor",
    pošta:"2000",
    ulica:"Prva 1",
  }})
```

```
db.študentiFakultete1.insert({
  student_id:"2021/1001",
  ime:"DrugiNovak",
```

```

smer:"Računalništvo",
izpiti:[
  {
    predmet:"Upor. PO",
    ocena:"10",
    datum : "4.4.2021"
  },
  {
    predmet:"Osnove PB",
    ocena:"10",
    datum : "4.4.2021"
  },
  {
    predmet:"Osnove IS",
    ocena:"10",
    datum : "4.4.2021"
  }
],
naslov: {
  mesto:"Maribor",
  pošta:"2000",
  ulica:"Prva 111",
}})

```

c) Poizvedbe

Za poizvedbe uporabljamo metodo `find()`:

```
db.ime_zbirke.find({povpraševanje})
```

Pogoj enakosti zapišemo: { <ključ> : <vrednost> }

Lahko pa uporabimo operatorje:

manjše kot- \$lt,
 manjše ali enako kot- \$lte,
 večje kot- \$gt,
 večje ali enako kot- \$gte,
 različno- \$ne

Izpišimo študente, ki so starejši od 20 let.

```
db.studenti.find({ starost: { $gt: 20 } })
```

Izpišimo študente računalništva:

```
db.studenti.find({ smer: "Računalništvo" }).pretty()
```

Več pogojev lahko združujemo z \$and ali \$or.

```
db.ime_zbirke.find({ $and: [ {pogoj}, {pogoj}, .... ] })
```

Izpišimo študente s priimkom Novak, ki niso na smeri *Računalništvo*:

```
db.študenti.find({ $and: [{ime: /.Novak./ }, {smer: {$ne: "Računalništvo"}}] }).pretty()
```

Izpišimo študente smeri *Računalništvo*, ki so opravili izpit *Osnove IS*.

```
db.študenti.find({ $and: [{smer:"Računalništvo"}, {izpiti: { $elemMatch: {predmet: "Osnove IS"}}}] }).pretty()
```

Izpišemo samo imena študentov računalništva, ki so opravili izpit *Osnove IS*:

```
db.študentiFakultete1.find({ $and: [{smer:"Računalništvo"}, {izpiti: { $elemMatch: {predmet: "Osnove IS"}}}] } {"ime": 1, "_id": 0})
```

d) Tvorimo indekse

MongoDB ukazi za delo z indeksi so:

```
db.ime_zbirke.createIndex({ključ:vrednost, .... })
```

```
db.ime_zbirke.getIndexes()
```

```
db.ime_zbirke.dropindex({ključ:vrednost})
```

Indeksirajmo

```
db.student.createIndex({student_id: -1, smer: 1})
```

8.2 Kompleksne poizvedbe v MongoDB

Agregacije *MongoDB* modelira kot procesni cevovod. Dokumenti vstopajo v večstopenjski cevovod, ki preoblikuje dokumente do končnega rezultata.

Primer:

```
db.študenti.aggregate([  
  { $match: { smer: "Računalništvo" } },  
  { $group: { _id: "$cust_id", total: { $sum: "$amount" } } } ] )
```

Prva stopnja uporabi *\$match* stopnjo, ki preseje dokumente po smeri in posreduje naslednji stopnji (*\$group*;) dokumente študentov smeri *Računalništvo*.

Naslednja stopnja *\$group* stopnja naredi skupine dokumentov z istim poljem "*\$cust_id*", in izračuna vsoto vrednosti vsake skupine z istim "*\$cust_id*".

Za agregirane funkcije nudi *MongoDB* metodo *aggregate()*.

```
db.ime_zbirke.aggregate( [ { <stopnja> }, ... ] )
```

Stopen je lahko več in možne so naslednje *stopnje* :

\$project posreduje polja za naslednjo stopnjo.

\$match preseje dokumente in posreduje samo odgovarjajoče.

\$limit posreduje samo določeno število dokumentov, ki jih določa parameter.

\$group oblikuje dokumente v skupine.

\$sort dokumente uredi naraščajoče ali padajoče.

\$min, *\$max*, *\$avg*, *\$sum* izračuna najmanjšo, največjo, povprečno vrednost ali vsoto.

\$lookup izvede stik z leve (*angl. left out join*) z drugo zbirko znotraj iste baze.

```
{
  $lookup:
  {
    from: <zbirka dokumentov za stik >,
    localField: <polje iz dokumentov osnovne zbirke>,
    foreignField: <polje dokumentov iz zbirke za stik iz »from«>,
    as: <ime polja izhodne matrike>
  }
}
```

8.2.1 Zbirka nalog

Ustvarimo bazo univerza in vstavimo zbirki dokumentov fakultete in študenti.

```
mongo "mongodb+srv://cluster0.ovgdr.mongodb.net/test" --username test
```

use univeza

```
db.fakultete.insert([
  { ime : 'FERI',
    lokacija : {
      mesto : 'Maribor', naslov: 'Koroška cesta 46',
    },
    predmeti: [
      { ime : 'Podatkovne baze', letnik: 1, smer: 'Računalništvo'},
      { ime : 'Osnove svetovnega spleta', letnik: 1, smer: 'Računalništvo'},
      { ime : 'Uporabniška PO', letnik: 1, smer: 'Računalništvo'},
    ]
  },
  { ime : 'FF',
    lokacija : {
      mesto : 'Maribor', naslov: 'Koroška cesta 60',
    },
    predmeti: [
      { ime : 'Uvod v ZRD', letnik: 3, smer: 'Zgodovina'},
    ]
  }
])
```

```
db študenti.insert([
  { fakulteta : 'FERI', leto : 2018, vpisna : 24441, ime : 'Prvi Novak', smer:
    'Računalništvo', letnik: 4},
```

```

... { fakulteta : 'FERI', leto : 2018, vpisna : 24442, ime : 'Drugi Novak' ,smer:
'Računalništvo',letnik: 4},
... { fakulteta : 'FERI', leto : 2018, vpisna : 24444, ime : 'Tretji Novak' ,smer:
'Računalništvo',letnik: 4},
... { fakulteta : 'FERI', leto : 2019, vpisna : 23333, ime : 'Četrti Novak' ,smer:
'Računalništvo',letnik: 3},
... { fakulteta : 'FERI', leto : 2020, vpisna : 21111, ime : 'Peti Novak' ,smer:
'Računalništvo',letnik: 2},
... { fakulteta : 'FERI', leto : 2021, vpisna : 22222, ime : 'Šesti Novak' ,smer:
'Računalništvo',letnik: 1}
])

```

a) Izpišimo vse dokumente vseh fakultet iz Maribora.

```

db.fakultete.aggregate([
... { $match : { mesto : 'Maribor' } }
... ]).pretty()

```

Zgornja poizvedba nič ne izpiše, ker je mesto del gnezdenega dokumenta s poljem *lokacija* in vrednost dobimo z *lokacija.mesto*:

```

db.fakultete.aggregate([
... { $match : { 'lokacija.mesto' : 'Maribor' } }
... ]).pretty()

```

b) Izpišimo samo imena fakultet iz Maribora.

```

db.fakultete.aggregate([
{ $project : { _id : 0, country : 1, mesto : 1, ime : 1 } }
... ]).pretty()

```

Izpis:

```

{ "ime" : "FERI" }
{ "ime" : "FF" }

```

c) Izpišimo število dokumentov v zbirki fakultete.

```

db.fakultete.aggregate([
... { $group : { _id : '$ ime ', totaldocs : { $sum : 1 } } }
... ]).pretty()

```

Izpis:

```

{ "_id" : null, "totaldocs" : 2 }

```

d) Shranimo število dokumentov v zbirki fakultete v zbirko *dokumentRezultatov*:

```

db.fakultete.aggregate([

```



```
... { $group : { _id : '$ ime ', totaldocs : { $sum : 1 } } },
... { $out : 'dokumentRezultatov' }
... ])
```

> show collections

Izpis:

```
dokumentRezultatov
fakultete
študenti
```

```
db.dokumentRezultatov.find().pretty()
```

Izpis:

```
{ "_id" : null, "totaldocs" : 2 }
```

e) Za vsak predmet v zbirki dokumentov fakultete naredimo svoj dokument.

```
db.fakultete.aggregate([
... { $match : { ime : 'FERI' } },
... { $unwind : '$predmeti' }
... ]).pretty()
```

Izpis:

```
{
  "_id" : ObjectId("60bf56b132deb9325b29b392"),
  "ime" : "FERI",
  "lokacija" : {
    "mesto" : "Maribor",
    "naslov" : "Koroška cesta 46"
  },
  "predmeti" : {
    "ime" : "Podatkovne baze",
    "letnik" : 1,
    "smer" : "Računalništvo"
  }
}
{
  "_id" : ObjectId("60bf56b132deb9325b29b392"),
  "ime" : "FERI",
  "lokacija" : {
    "mesto" : "Maribor",
    "naslov" : "Koroška cesta 46"
  },
  "predmeti" : {
    "ime" : "Osnove svetovnega spleta",
    "letnik" : 1,

```

```

    "smer" : "Računalništvo"
  }
}
{
  "_id" : ObjectId("60bf56b132deb9325b29b392"),
  "ime" : "FERI",
  "lokacija" : {
    "mesto" : "Maribor",
    "naslov" : "Koroška cesta 46"
  },
  "predmeti" : {
    "ime" : "Uporabniška PO",
    "letnik" : 1,
    "smer" : "Računalništvo"
  }
}

```

f) Predmete prikažimo kot svoj dokument in izpišimo samo ime predmet in smer.

```

db.fakultete.aggregate([db.fakultete.aggregate([
... { $match : { ime : 'FERI' } },
... { $unwind : '$predmeti' },
... { $project : { _id : 0, 'predmeti.ime' : 1, 'predmeti.smer' : 1 } },
... { $sort : { 'predmeti.ime' : -1 } }
... ]).pretty()

```

Izpis:

```

{ "predmeti" : { "ime" : "Podatkovne baze", "smer" : "Računalništvo" } }
{ "predmeti" : { "ime" : "Osnove svetovnega spleta", "smer" : "Računalništvo" } }
{ "predmeti" : { "ime" : "Uporabniška PO", "smer" : "Računalništvo" } }

```

g) Če v prejšnji nalogi želimo samo 1 predmet uporabimo limit.

```

db.fakultete.aggregate([
... { $match : { ime : 'FERI' } },
... { $unwind : '$predmeti' },
... { $project : { _id : 0, 'predmeti.ime' : 1, 'predmeti.smer' : 1 } },
... { $sort : { 'predmeti.ime' : -1 } },
  { $limit : 1 }
... ]).pretty()

```

Izpis:

```

{ "predmeti" : { "ime" : "Podatkovne baze", "smer" : "Računalništvo" } }

```

h) Izpišimo število predmetov vseh fakultet:

```
db.fakultete.aggregate([
... { $unwind : '$predmeti' },
... { $count : 'total_documents' }
... ]).pretty()
```

Izpis:

```
{ "total_documents" : 4 }
```

i) Povežimo podatke o fakulteti s študenti in uporabimo obe zbirki dokumentov: fakultete in študenti. Uporabimo *\$lookup* in uparimo polje fakulteta v zbirki dokumentov študenti s poljem ime v zbirki dokumentov fakultete.

```
db.fakultete.aggregate([
... { $match : { ime : 'FERI' } },
... { $project : { _id : 0, ime : 1, lokacija:1 } },
... { $lookup : {
... from : 'študenti',
... localField : 'ime',
... foreignField : 'fakulteta',
... as : 'študenti'
... } }
... ]).pretty()
```

Izpis:

```
{
  "ime" : "FERI",
  "lokacija" : {
    "mesto" : "Maribor",
    "naslov" : "Koroška cesta 46"
  },
  "študenti" : [
    {
      "_id" : ObjectId("60bf56be32deb9325b29b394"),
      "fakulteta" : "FERI",
      "leto" : 2018,
      "vpisna" : 24441,
      "ime" : "Prvi Novak",
      "smer" : "Računalništvo",
      "letnik" : 4
    },
    {
      "_id" : ObjectId("60bf56be32deb9325b29b395"),
      "fakulteta" : "FERI",
      "leto" : 2018,
      "vpisna" : 24442,
```

```

    "ime" : "Drugi Novak",
    "smer" : "Računalništvo",
    "letnik" : 4
  },
  {
    "_id" : ObjectId("60bf56be32deb9325b29b396"),
    "fakulteta" : "FERI",
    "leto" : 2018,
    "vpisna" : 24444,
    "ime" : "Tretji Novak",
    "smer" : "Računalništvo",
    "letnik" : 4
  },
  {
    "_id" : ObjectId("60bf56be32deb9325b29b397"),
    "fakulteta" : "FERI",
    "leto" : 2019,
    "vpisna" : 23333,
    "ime" : "Četrti Novak",
    "smer" : "Računalništvo",
    "letnik" : 3
  },
  {
    "_id" : ObjectId("60bf56be32deb9325b29b398"),
    "fakulteta" : "FERI",
    "leto" : 2020,
    "vpisna" : 21111,
    "ime" : "Peti Novak",
    "smer" : "Računalništvo",
    "letnik" : 2
  },
  {
    "_id" : ObjectId("60bf56be32deb9325b29b399"),
    "fakulteta" : "FERI",
    "leto" : 2021,
    "vpisna" : 22222,
    "ime" : "Šesti Novak",
    "smer" : "Računalništvo",
    "letnik" : 1
  }
]
}

```

Literatura

- [1] Codd, E. F. (1970). A Relational Tedel of Data for Large Shared Data Banks. Communications of the ACM. 13 (6): 377–387. doi:10.1145/362384.362685I.
- [2] Chen, Peter Pin-Shan (1976). The Entity–Relationship Model – Toward A Unified View of Data. ACM Transactions on Database Systems. 1 (1): 9–36. CiteSeerX 10.1.1.523.6679. doi:10.1145/320434.320440. S2CID 52801746. ISSN 0362-5915.
- [3] Chapple, Mike. SQL Fundamentals. Databases. About.com. Dostopano: 2021-05-14.
- [4] MySQL 8.0 Reference Manual.
<https://dev.mysql.com/doc/refman/8.0/en/> Dostopano: 2021-05-14.
- [5] SQL Tutorial, <https://www.w3schools.com/sql/default.asp> Dostopano: 2021-05-14.
- [6] MongoDB manual, <https://docs.mongodb.com/manual/> Dostopano: 2021-05-14.
- [7] Karl Seguin: The Little MongoDB Book. <http://github.com/karlseguin/the-littlemongodb-book>, 2012.
- [8] MongoDB university: <https://university.mongodb.com/> Dostopano: 2021-05-14.
- [9] MongoDB tuturiel: <https://www.tutorialspoint.com/mongodb/index.htm>
- [10] MongoDB tuturiel for beginners: <https://www.guru99.com/mongodb-tutorials.html>
- [11] The database for modern applications; <https://www.mongodb.com/> (Get started free: <https://www.mongodb.com/cloud/atlas/register>). Dostopano: 2021-05-20.