RAČUNALNIŠTVO

NAPREDNA UPORABA PODATKOVNIH BAZ - NUB

Uroš Sterle

www.bodiprofi.si

munus 2

Uvajanje novih izobraževalnih programov v srednjem poklicnem in strokovnem izobraževanju s področja tehnike za obdobje 2008-2012.
Operacijo delno sofinancira Evropska unija iz Evropskega socialnega sklada in Ministrstvo za šolstvo in šport.

SPLOŠNE INFORMACIJE O GRADIVU

Izobraževalni program
Tehnik računalništva

Ime modula
Napredna uporaba podatkovnih baz 4 – NUB

Naslov učnih tem ali kompetenc, ki jih obravnava učno gradivo
Jezik za rokovanje s podatki (DML – Data Manipulation Language). Jezik za definiranje podatkov (DDL – Data Definition Language). Shranjene procedure, funkcije, prožilci in dogodki. Jezik za nadzor nad podatki (DCL – Data Control Language). Jezik za nadzor nad transakcijami (TCL - Transaction Control Language). Arhiviranje in restavracija podatkovne baze. Izdelava aktivne spletne strani.

Avtor: Uroš Sterle
Recenzent: Andrej Arh
Lektor: Milena Ilić
Datum: avgust 2012

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

POVZETEK/PREDGOVOR

Gradivo ***Napredna uporaba podatkovnih baz*** je namenjeno dijakom 4. letnika SSI – Tehnik računalništva in dijakom 1. letnika PTI - Tehnik računalništva. Pokriva vsebinski del, naveden v katalogu znanja, pri čemer sem kot izbrani sistem za upravljanje s podatkovnimi bazami izbral MySQL. Podatkovne zbirke so danes nepogrešljive na različnih področjih. So tako del poslovnih aplikacij, kot osnova dinamičnih spletnih strani, tako da predstavljajo pomemben kamenček v mozaiku znanja modernega računalnikarja.

Gradivo je nastalo na osnovi zapiskov in dolgoletnih izkušenj avtorja z delom s podatkovnimi bazami.

V besedilu so poleg teorije tudi številni zgledi, na koncu poglavij pa tudi vaje. Rešitve vaj, ki so zbrane na koncu celotnega gradiva, predstavljajo le eno izmed možnih rešitev, mogoče niti ne najboljšo in najkrajšo. Ker se poizvedovalnega jezika SQL seveda ne da naučiti le s prepisovanjem tujih skript, pričakujem, da bodo dijaki poleg skrbnega študija zgledov in rešitev pisali kodo tudi sami. Zato jim predlagam, da rešijo naloge, ki so objavljene v številnih, na spletu dosegljivih, zbirkah nalog. Dijake, ki bi o posamezni tematiki radi izvedeli več, vabim, da si ogledajo tudi gradivo, ki je navedeno v literaturi.

Bralcem, ki bodo uporabljati ta učbenik pri svojem predmetu, priporočam tudi uporabo spletne učilnice, za katero menim, da je postala nepogrešljiv pripomoček pri poučevanju programiranja. Gradivo ***Napredna uporaba podatkovnih baz*** opisuje: Jezik za rokovanje s podatki (DML – Data Manipulation Language), jezik za definiranje podatkov (DDL – Data Definition Language), shranjene procedure, funkcije, prožilce in dogodke, jezik za nadzor nad podatki (DCL – Data Control Language), jezik za nadzor nad transakcijami (TCL - Transaction Control Language), arhiviranje in restavracijo podatkovne baze ter izdelava aktivne spletne strani, kot zaključek oz. praktično uporabo podatkovnih zbirk.

Ključne besede: SQL, podatkovne zbirke, tabele, poizvedbe, ažuriranje, podatkovni tipi, indeksi, pogledi, procedure, funkcije, prožilci, dogodki, nadzor nad podatki, transakcije, arhiviranje, restavracija aktivna spletna stran.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO VSEBINE

KAZALO SLIK

KAZALO TABEL

1 Uvod in priprava delovnega okolja

V tem gradivu se bomo posvetili napredni uporabi SQL-a. Najprej se bomo seznanili z nekaterimi termini in si pripravili okolje, v katerem bomo delali.

1.1 SQL

SQL ali strukturirani povpraševalni jezik za delo s podatkovnimi bazami (angl. Structured Query Language) je najbolj razširjen in standardiziran povpraševalni jezik za delo s podatkovnimi zbirkami, s programskimi stavki, ki posnemajo ukaze v naravnem jeziku. Določen je z ANSI/ISO SQL standardom. SQL standard se je razvijal od leta 1986 in danes obstaja več različic. Oznaka SQL-92 se navezuje na standard, izdan v letu 1992, SQL:1999 se navezuje na standard, izdan leta 1999, SQL:2003 se navezuje na različico iz leta 2003 in tako naprej. Izraz SQL standard uporabljamo za poimenovanje trenutne različice SQL standarda v vsakem časovnem obdobju.

1.2 SUPB

Sistem za upravljanje s podatkovno bazo (okrajšano SUPB) je množica programov, namenjena kreiranju, vzdrževanju in nadzoru dostopa do podatkov v podatkovni bazi.

1.2.1 Najbolj znani SUPB

Najbolj znani SUPB so Oracle, Microsoft SQL Server, MySQL, Microsoft Access, Paradox, Firebird, PostgreSQL … Delali bomo z MySQL-om zaradi treh preprostih razlogov: je zastonj, zelo razširjen in zelo preprost za namestitev. Bralec, ki bo želel prestopiti k drugemu SUPB-ju, bo to brez posebnih težav lahko storil, obratno prav tako.

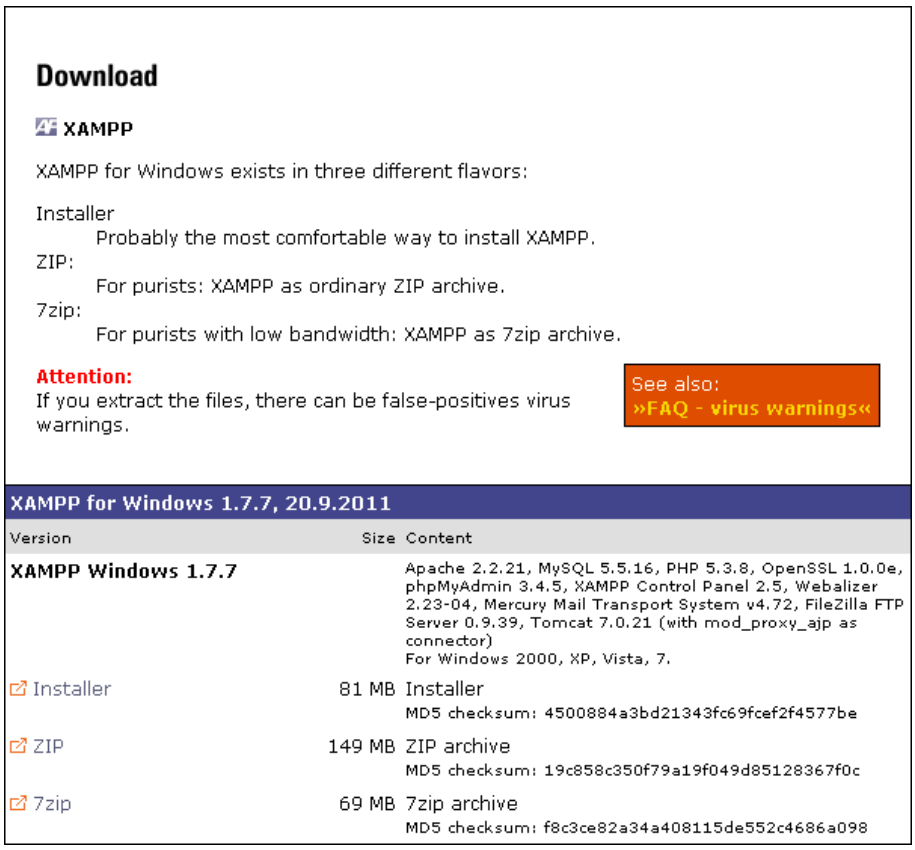
1.3 XAMPP

Xampp je skupek programov, ki med drugim vsebuje tudi MySQL. Ostale pomembnejše aplikacije, ki jih vsebuje, so spletni strežnik Apache, skriptni jezik PHP, vizualno orodje za delo z MySQL-om phpMyAdmin in FTP strežnik FileZilla.

1.3.1 Namestitev spletnega strežnika XAMPP

1. Prenos XAMPP

S spletne strani apachefriends.org prenesemo XAMPP za vaš operacijski sistem. Glede na to, da bomo uporabljali Windows, bomo prenesli "XAMPP for Windows".

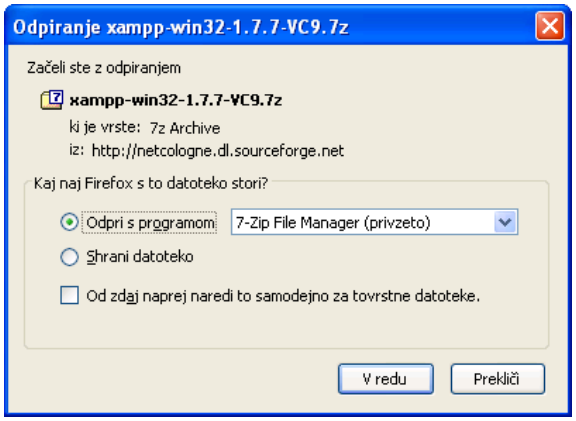


Slika 1: Spletna stran z datotekami XAMPP-a.

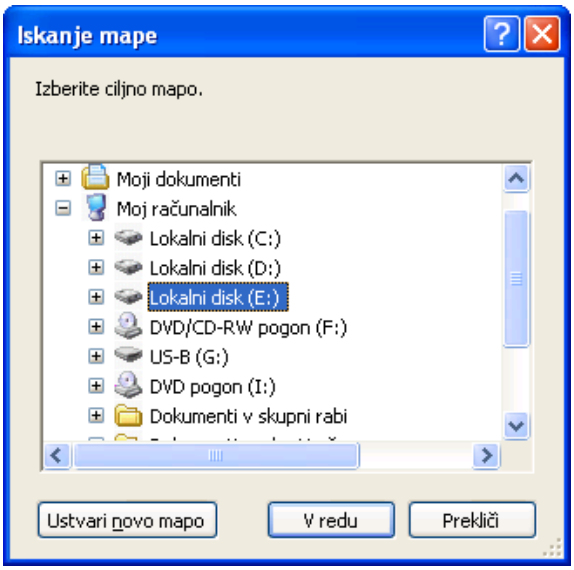
Izberemo lahko namestitveno datoteko ali pa prenosno različico, zapakirano v ZIP ali 7z datoteko. Izbrali bomo 7z datoteko, saj jo lahko namestimo tudi na USB ključek in jo imamo vedno pri roki.

2. Namestitev XAMPP

Preneseno 7z različico razpakiramo v korensko mapo enega od trdih diskov ali na USB ključek.



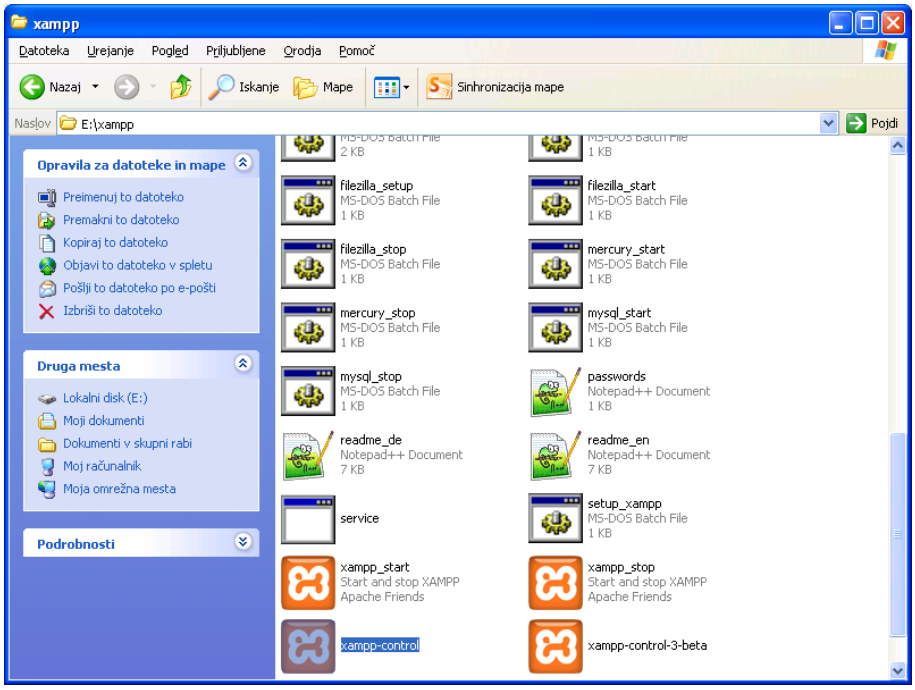
Slika 2: Okno s prenosom XAMPP-ja.



Slika 3: Okno za izbiro ciljne mape.

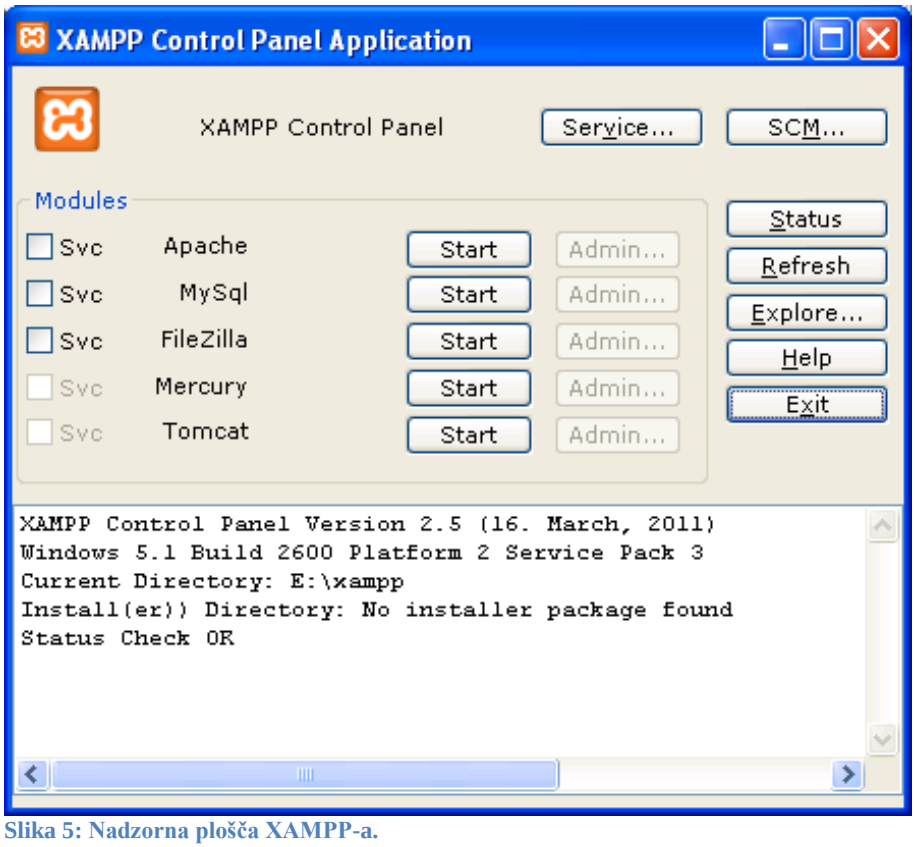
3. Zagon strežnika Apache in SUPB MySQL

V mapi xampp zaženemo datoteko "xampp-control.exe".



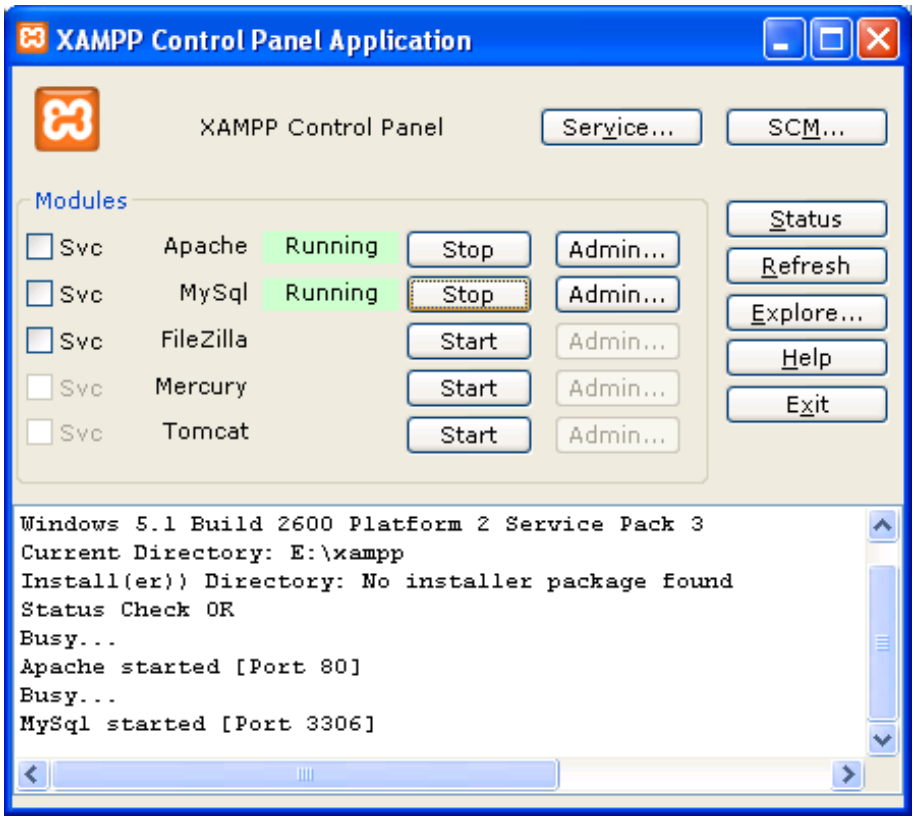
Slika 4: Vsebina mape XAMPP-a.

Kliknemo na gumba start ob oznakah Apache in MySql.



Slika 5: Nadzorna plošča XAMPP-a.

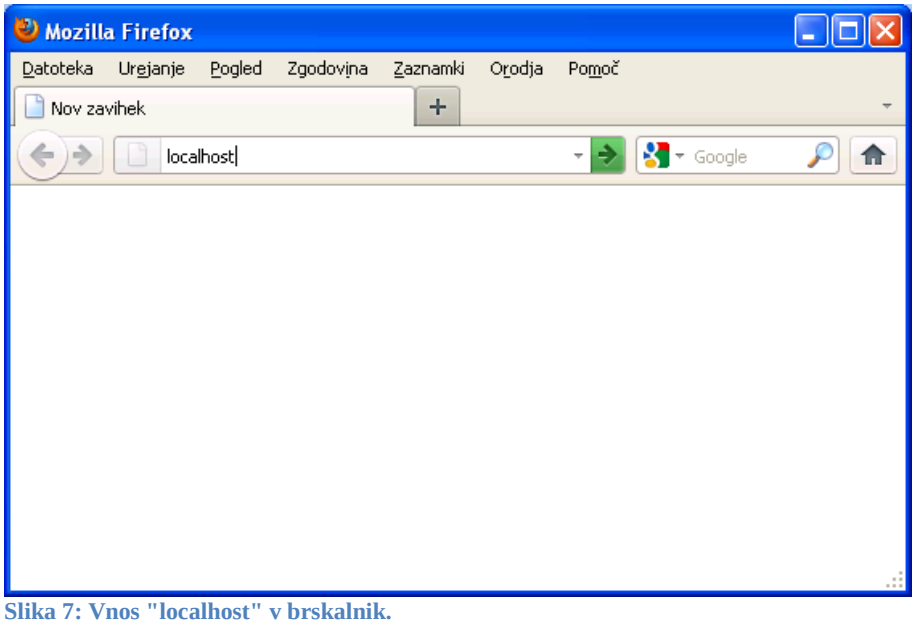
Dobiti moramo sledeče stanje.



Slika 6: Nadzorna plošča XAMPP-a in delujoča Apache in MySQL.

4. Uporaba XAMPP

Spletni brskalnik usmerimo na http://localhost oz. na http://127.0.0.1.

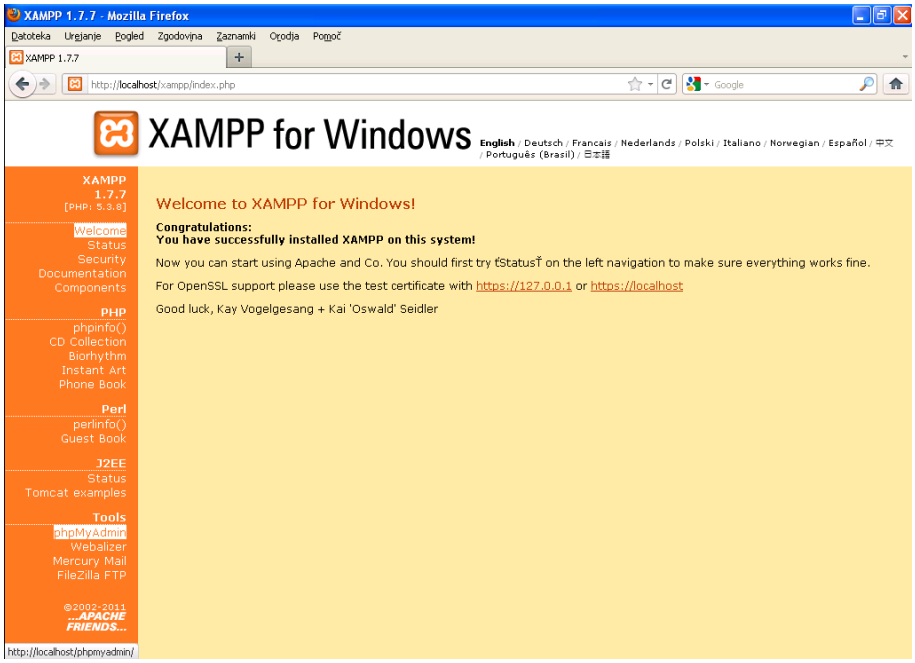


Slika 7: Vnos "localhost" v brskalnik.

Pojavi se začetno okno XAMPP-a.

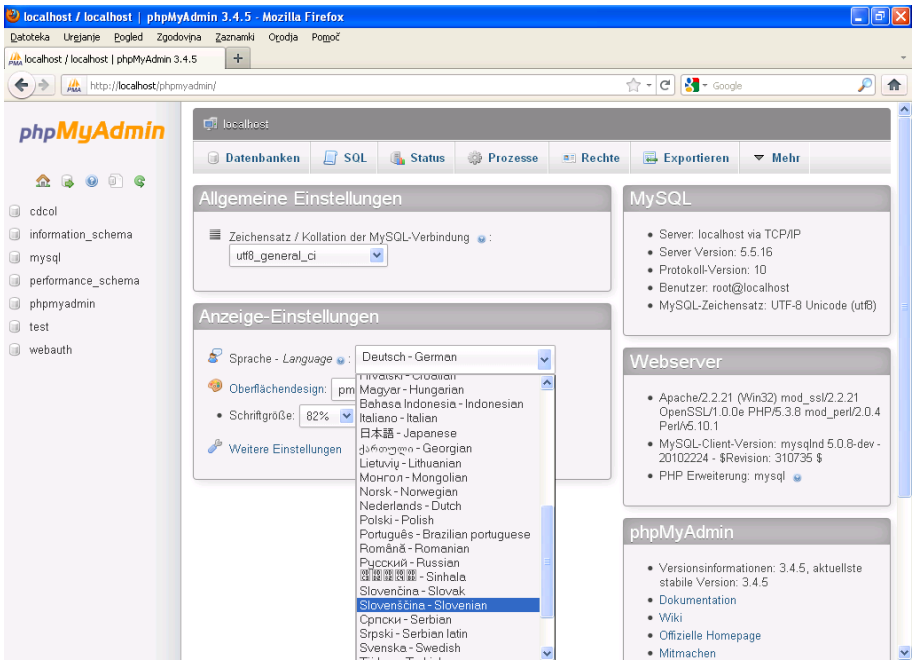
Slika 8: Začetno okno XAMPP-a.

Izberemo jezik, ki ga najbolj poznamo, in pojavi se pozdravno okno XAMPP-a. Spodaj levo izberemo phpMyAdmin.



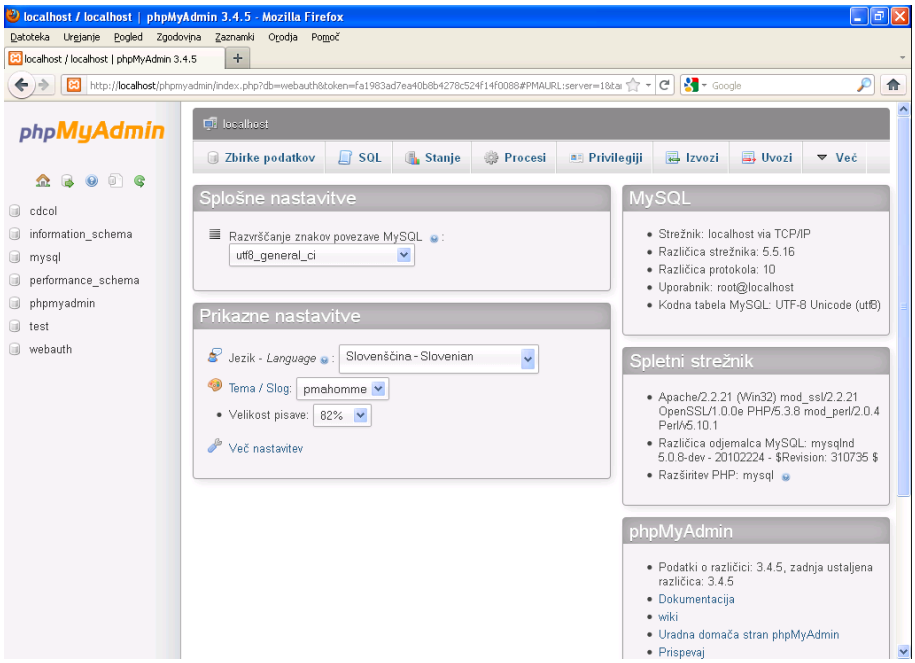
Slika 9: Pozdravno okno XAMPP-a.

Izberemo še jezik okolja phpMyAdmin ...



Slika 10: Izbira jezika okolja phpMyAdmin.

... in okolje je pripravljeno za delo.



Slika 11: Okolje pripravljeno za delo.

1.4 MySQL

1.4.1 Skupine SQL ukazov (DDL, DML, DCL, TCL)

Prva skupina je skupina za definiranje podatkov (DDL – Data Definition Language). Z njimi lahko ustvarimo, spreminjamo in brišemo zbirke podatkov, tabele, poglede, indekse in prožilce.
V drugo skupino (DML – data manipulation language) spadajo ukaz za vračanje podatkov iz tabel oz. za poizvedbe ter ukazi za brisanje, vstavljanje in spreminjanje.
DCL – Data Control Language iz tretje skupine pa omogoča vse v zvezi z zaščito podatkovne baze.
TCL - Transaction Control Language je jezik za nadzor nad transakcijami. Ta jezik nadzoruje potrjevanje podatkov.

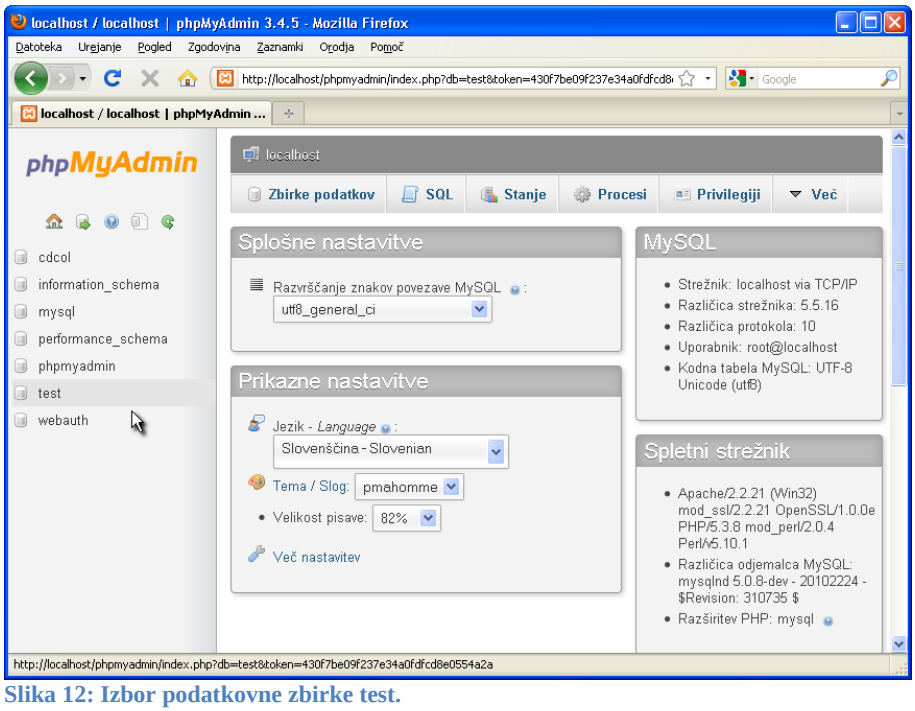
2 Jezik za rokovanje s podatki (DML – Data Manipulation Language)

Jezik za rokovanje s podatki obsega ukaz za vračanje podatkov iz tabel oz. za poizvedbe (SELECT). Podatke spreminjamo z ukazi za brisanje (DELETE), vstavljanje (INSERT) in spreminjanje (UPDATE).

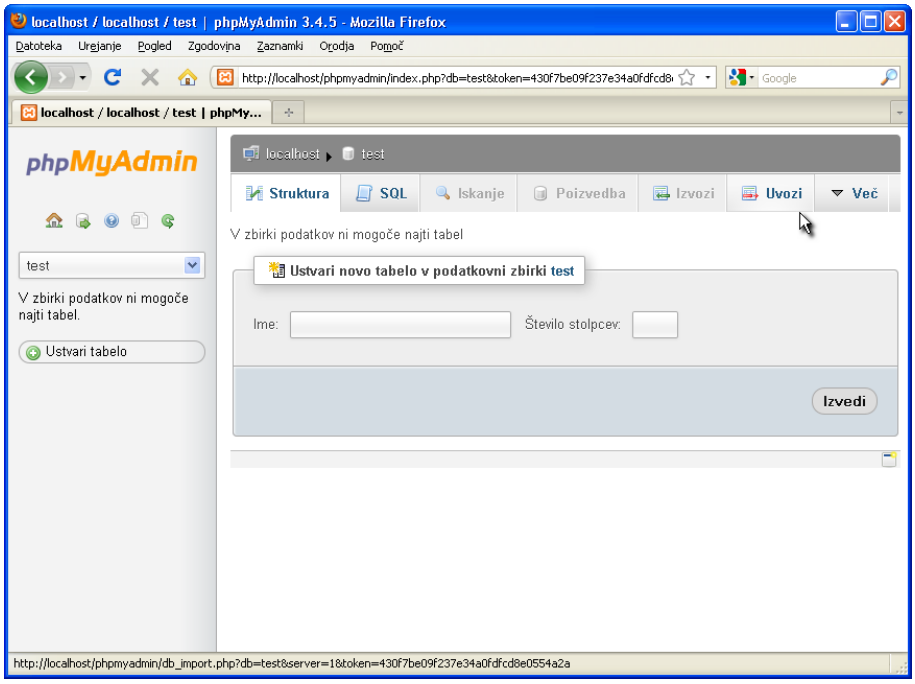
2.1 Uvoz podatkov

V tem trenutku še ne znamo ustvarjati tabel, znamo pa narediti kakšno poizvedbo.

V okolju phpMyAdmin izberemo podatkovno zbirko test in uvozimo podatke.

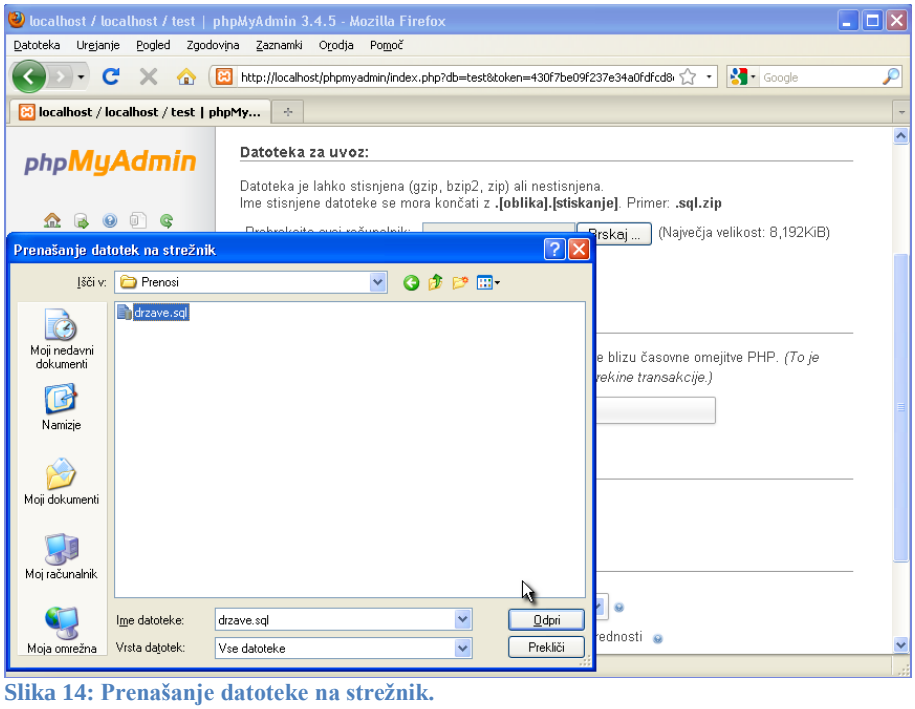


Slika 12: Izbor podatkovne zbirke test.



Slika 13: Stanje po izboru.

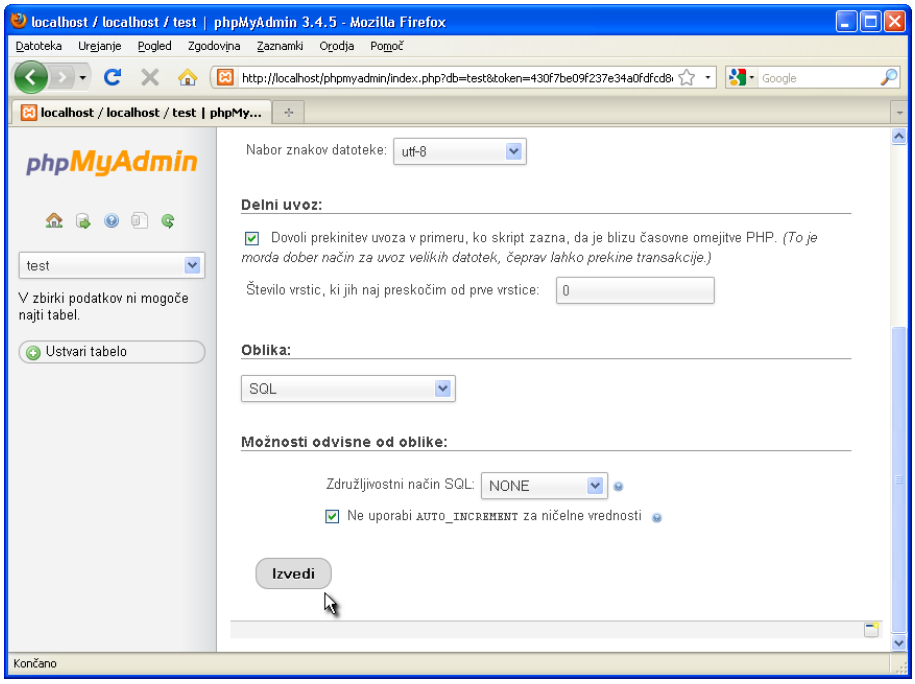
Izberemo datoteko »drzave.sql«[1], ki smo jo že prej prenesli s spleta in shranili na namizje. Vse datoteke, uporabljene v tem gradivu, lahko najdemo v http://sterle.tsckr.si/NUB/prenosi/.



Slika 14: Prenašanje datoteke na strežnik.

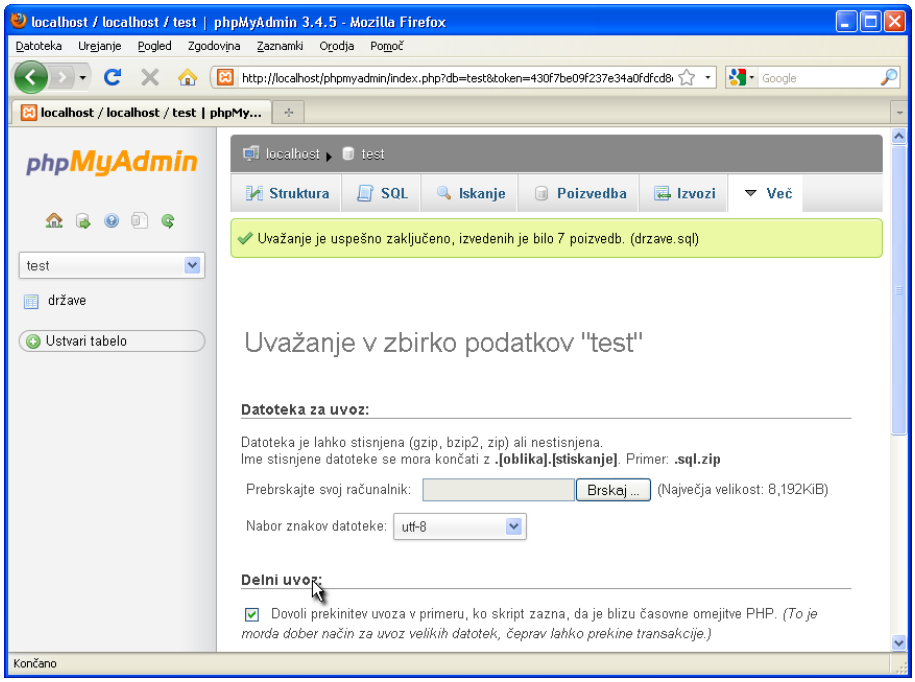
[1] http://sterle.tsckr.si/NUB/prenosi/drzave.sql (Vse datoteke, uporabljene v tem gradivu, lahko najdemo v http://sterle.tsckr.si/NUB/prenosi/.)

Kliknemo gumb "Izvedi" na dnu strani.



Slika 15: Gumb "Izvedi".

In tabela "države" je uvožena. Vidimo jo na levi strani.



Slika 16: Uvožena tabela.

S tem smo pripravili okolje za nadaljnje delo.

2.2 Osnovna oblika SELECT stavka

Ukaz SELECT nam vrne podatke iz tabel. Splošna oblika ukaza je:

```
SELECT seznam_polj
  FROM seznam_tabel
 WHERE pogoj;
```

`seznam_polj` je lahko spisek stolpcev ali *, ki pomeni »vsi stolpci«.
`seznam_tabel` označuje tabelo, iz katere črpamo podatke.
`pogoj` pa določi enega ali več pogojev, ki morajo biti izpolnjeni. WHERE stavek lahko izpustimo.

Stavke sicer ni potrebno zaključevati s podpičjem, kot je prikazano zgoraj, razen če delamo v ukaznem načinu oz. če pišemo zaporedje več ukazov.

2.2.1 VELIKE in male črke

Občutljivost na velike in male črke je odvisna od več faktorjev, med drugim od vrste elementov in operacijskega sistema. Tako so ključne besede in aliasi za tabele v vseh primerih neobčutljivi na velikost znakov, medtem ko so bazni objekti tabele in podatkovne zbirke občutljivi na UNIX-u. Na Windowsih niso.

Naslednji stavki so tako ekvivalentni:

```
SELECT version();
select version();
SeLeCt version();
```

Funkcija version() nam vrne oznako različice MySQL-a, ki ga uporabljamo:

```
+-----------+
| version() |
+-----------+
| 5.5.16    |
+-----------+
```

V zadnjem primeru smo videli SELECT stavek brez ključne besede FROM. To lahko storimo v primeru, da tabele ne potrebujemo. Uporabimo lahko tudi trivialno oz. prazno tabelo z imenom DUAL.

```
SELECT version()
  FROM DUAL;
```

2.2.2 Aritmetični operatorji

V seznamu polj lahko uporabimo tudi aritmetične operatorje (+, -, * in /) in oklepaje

```
SELECT 5+7;
SELECT cena*(1+ddv/100)
  FROM cenik;
```

2.3 Splošna sintaksa SELECT stavka

```
SELECT [ALL | DISTINCT | DISTINCTROW ] polje_1 [AS] alias_1 [, polje_2
...]
 [FROM ime_tabele
 [WHERE pogoj]
 [GROUP BY {polje | izraz | mesto} [ASC | DESC], ... [WITH ROLLUP]]
 [HAVING pogoj]
 [ORDER BY {polje | izraz | mesto} [ASC | DESC], ...]
 [LIMIT {[odmik,] število_vrstic | število_vrstic OFFSET odmik}]
 [INTO OUTFILE 'ime_datoteke'
 [CHARACTER SET nabor_znakov] izvozne_možnosti
 | INTO DUMPFILE 'ime_datoteke'
 | INTO ime_spremenljivke [, ime_spremenljivke]]
```

Oglejmo si SELECT stavek bolj podrobno.

Najprej povejmo še nekaj o oznakah.

Pokončna črta "|" pomeni, da uporabimo eno izmed možnosti (ASC | DESC pomeni, da napišemo ali ASC ali DESC).

Oglati okepaji "[]" pomenijo neobvezno vsebino ([WHERE pogoj] pomeni, da vsebino oglatih oklepajev, tj. "WHERE pogoj", lahko izpustimo) .

Zaviti oklepaji "{}" pomenijo obvezno vsebino ({polje | izraz | mesto} pomeni, da moramo vnesti ali polje ali izraz ali mesto).

Kaj pomeni torej vrstica:

```
[ORDER BY {polje | izraz | mesto} [ASC | DESC], ...]
```

Zunanji oglati oklepaj pomeni, da lahko vse skupaj izpustimo. {polje | izraz | mesto} pomeni, da je obvezno vpisati ali polje ali izraz ali mesto. [ASC | DESC] pomeni, da lahko pišemo ali ASC ali DESC ali pa vse skupaj izpustimo. Vejica in ... (, ...) pa pomeni, da lahko vnesemo več takih zaporedij, torej da lahko razvrščamo po večih stoplcih.

```
ALL | DISTINCT | DISTINCTROW
```

ALL je privzeto in ponavadi ne pišemo, pomeni pa, da vrne vse vrstice, tudi podvojene, DISTINCT izloči podvojene vrstice. DISTINCTROW je sinonim za DISTINCT.

```
SELECT DISTINCT ime
  FROM dijaki;
```

Nam vrne samo različna imena dijakov.

```
AS (alias)
```

Alias je prislov, ki pomeni drugače povedano, z drugim imenom, po domače, kot nam razloži SSKJ. V SQL-u s pomočjo aliasov lahko preimenujemo stolpce. Besedico AS lahko tudi izpustimo.

```
SELECT DISTINCT ime AS različna_imena
  FROM dijaki;
```

Besedico AS lahko tudi izpustimo. Alias določi že presledek za stolpcem.

```
SELECT DISTINCT ime različna_imena
  FROM dijaki;
```

Z aliasom preimenovan stolpec lahko uporabimo v stavkih GROUP BY, ORDER BY in HAVING.

```
FROM
```

Za besedico FROM sledijo imena ene ali več tabel.

```
SELECT *
  FROM dijaki, krožki;
```

Tudi tabelam lahko določimo aliase.

```
SELECT *
  FROM dijaki d, krožki k;
```

Običajno za aliase tabel uporabimo kar začetne črke tabel.

2.4 Agregacijske funkcije

Včasih informacija, ki jo potrebujemo, ni direktno zapisana v tabeli, jo pa lahko izračunamo iz shranjenih podatkov.

Primer: Imamo neko naročilo, ki vsebuje ime izdelka, vrsto izdelka, število kosov in ceno. Zanima pa nas, koliko je vseh izdelkov, kolikšna je skupna cena, koliko je različnih izdelkov, najdražji izdelek, najcenejši izdelek, povprečna vrednost izdelka in skupna cena za posamezno vrsto izdelka. Teh podatkov nimamo, jih pa seveda preprosto izračunamo s pomočjo agregacijskih funkcij.

Znamo našteti funkcije, ki jih potrebujemo za izračun zgornjega primera? Štetje, seštevanje, povprečna vrednost, maksimum, minimum …

Nadaljujmo z našim primerom. Podano imamo tabelo naročilo z naslednjimi podatki:

ime_izdelka	vrsta_izdelka	kosi	cena
Jogurt	Mlečni izdelki	2	0,50
Rogljiček	Sladice	5	1,00
Salama	Mesni izdelki	1	4,99
Mleko	Mlečni izdelki	3	0,80
Sir	Mlečni izdelki	1	2,00
Klobasa	Mesni izdelki	2	3,00
Torta	Sladice	1	9,50

Tabelo naročilo si lahko uvozite s pomočjo datoteke »narocilo.sql«[2].

2.4.1 COUNT

Prva agregacijska funkcija, ki si jo bomo ogledali je COUNT. Postavimo si nekaj vprašanj in nanje odgovorimo.

1. Koliko je zapisov v tabeli?
2. Koliko je različnih vrst izdelkov?
3. Koliko je izdelkov v vsaki vrsti izdelka?

Glede na to, da je naša tabela majhna, to lahko storimo na pamet. Seveda pri velikih tabelah to ne bi imelo nobenega pomena in zato imamo tudi funkcijo COUNT(). Vse agregacijske funkcije kot argument sprejmejo stolpec tabele. V primeru COUNT pa za stolpec lahko uporabimo kar *, in sicer v primeru, ko hočemo prešteti vse vrstice, ki ustrezajo poizvedbi.

1. Koliko je zapisov v tabeli?

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

[2] http://sterle.tsckr.si/NUB/prenosi/narocilo.sql (Vse datoteke, uporabljene v tem gradivu, lahko najdemo v http://sterle.tsckr.si/NUB/prenosi/.)

Rezultat:

```
+----------+
| COUNT(*) |
+----------+
|        7 |
+----------+
```

2. Koliko je različnih vrst izdelkov?

```
SELECT COUNT(vrsta_izdelka)
FROM naročilo;
```

Rezultat:

```
+----------+
| COUNT(*) |
+----------+
|        7 |
+----------+
```

To pa ni to, kar smo želeli doseči. Želeli smo število različnih vrst izdelkov, a moramo to tudi posebej povedati z ukazom DISTINCT.

```
SELECT COUNT(DISTINCT vrsta_izdelka)
FROM naročilo;
```

Rezultat:

```
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
```

Običajno agregacijske funkcije uporabljamo v kombinaciji z GROUP BY in HAVING stavkoma. In ravno GROUP BY potrebujemo za zadnji odgovor.

3. Koliko je izdelkov v vsaki vrsti izdelka?

```
SELECT vrsta_izdelka, COUNT(*) AS 'število izdelkov'
FROM naročilo
GROUP BY vrsta_izdelka;
```

Rezultat:

```
+----------------+------------------+
| vrsta_izdelka  | število izdelkov |
+----------------+------------------+
| Mesni izdelki  |                2 |
| Mlečni izdelki |                3 |
| Sladice        |                2 |
+----------------+------------------+
```

2.4.2 SUM

Funkcija SUM() sešteje vrednosti v stolpcu, ki ustrezajo poizvedbi.

Vprašanja:

1. Koliko je vseh kosov v naročilu?

```
SELECT SUM(kosi)
FROM naročilo;
```

Rezultat:

```
+-----------+
| SUM(kosi) |
+-----------+
|        15 |
+-----------+
```

2. Kolikšna je skupna vrednost vseh kosov posameznega izdelka?

```
SELECT ime_izdelka, SUM(kosi*cena)
FROM naročilo
GROUP BY ime_izdelka;
```

Rezultat:

```
+-------------+----------------+
| ime_izdelka | SUM(kosi*cena) |
+-------------+----------------+
| Jogurt      |           1.00 |
| Klobasa     |           6.00 |
| Mleko       |           2.40 |
| Rogljiček   |           5.00 |
| Salama      |           4.99 |
| Sir         |           2.00 |
| Torta       |           9.50 |
+-------------+----------------+
```

3. Kolikšna je skupna vrednost naročila?

```
SELECT SUM(kosi*cena)
FROM naročilo;
```

Rezultat:

```
+----------------+
| SUM(kosi*cena) |
+----------------+
|          30.89 |
+----------------+
```

2.4.3 AVG

Funkcija AVG() izračuna povprečno vrednost vrednosti v stolpcu, ki ustrezajo poizvedbi.

Poiščimo povprečno ceno za posamezno vrsto izdelkov!

```
SELECT vrsta_izdelka, AVG(cena)
FROM naročilo
GROUP BY vrsta_izdelka;
```

Rezultat:

```
+----------------+-----------+
| vrsta_izdelka  | AVG(cena) |
+----------------+-----------+
| Mesni izdelki  |  3.995000 |
| Mlečni izdelki |  1.100000 |
| Sladice        |  5.250000 |
+----------------+-----------+
```

2.4.4 MIN

Funkcija MIN() poišče najmanjšo vrednost v stolpcu, ki ustreza poizvedbi.

Poiščimo najmanjšo ceno!

```
SELECT MIN(cena)
FROM naročilo;
```

Rezultat:

```
+-----------+
| MIN(cena) |
+-----------+
|      0.50 |
+-----------+
```

2.4.5 MAX

Funkcija MAX() poišče največjo vrednost v stolpcu, ki ustrezajo poizvedbi.

Poiščimo izdelek z največ kosi!

```
SELECT ime_izdelka
FROM naročilo
WHERE kosi = (SELECT MAX(kosi)
                FROM naročilo);
```

Rezultat:

```
+-------------+
| ime_izdelka |
+-------------+
| Rogljiček   |
+-------------+
```

2.5 Vaje - SELECT

Pripravljeno imate datoteko za uvoz in naloge. Kako uvozite podatke, imate opisano v prejšnjem poglavju. Naloge skrbno preberite, delajte jih po vrsti, saj si logično sledijo. Velikokrat je potreben samo majhen popravek, da rešite naslednjo nalogo.

1. Izpišite vse podatke iz tabele države.
2. Izpišite vse podatke za Slovenijo.
3. Izpišite vse podatke za Slovenijo in Avstrijo.
4. Izpišite vse podatke za Slovenijo, Nizozemsko, Belgijo, Norveško in Španijo.
5. Izpišite imena vseh držav, ki so velike med 10.000 in 30.000 km^2. Spisek uredite po abecednem vrstnem redu naraščajoče.
6. Izpišite imena vseh evropskih držav v bazi. Spisek uredite po abecednem vrstnem redu padajoče.
7. Izpišite vse podatke o državah, katerih ime se prične s črko H.
8. Izpišite vse podatke o državah, ki imajo v regiji besedo Amerika (Južna Amerika, Severne Amerika, Srednja Amerika).
9. Prikažite ime države ter število prebivalcev na km^2 površine države. Spisek uredite enkrat padajoče in enkrat naraščajoče po gostoti prebivalstva. Stolpec, ki prikazuje število prebivalcev na km^2, poimenujte »Gostota prebivalstva«.
10. Prikažite gostoto prebivalstva za Slovenijo. V glavi stolpca naj piše: »Gostota prebivalstva za Slovenijo«.
11. Prikažite ime države in BDP na prebivalca. Spisek naj bo urejen naraščajoče po BDP-ju na prebivalca. Stolpec, v katerem je prikazana izračunana vrednost, naj se imenuje: »BDP na prebivalca«.
12. Stolpcu dajte ime »Število evropskih držav v bazi« in izpišite podatke o številu držav, pri katerih je kot regija navedena Evropa.
13. Koliko je držav z več kot 50 milijoni prebivalcev?
14. Koliko je skupno število prebivalcev v državah, ki so v bazi?
15. Koliko je skupno število prebivalcev v evropski regiji?
16. Prikažite skupno število prebivalcev za posamezno regijo. Seznam naj bo urejen padajoče po skupnem številu prebivalcev.
17. Prikažite vse regije, ki imajo več kot 500.000.000 prebivalcev.
18. Prikažite skupno število prebivalcev in gostoto prebivalstva na km^2 za posamezno regijo. Seznam naj bo urejen padajoče po skupni gostoti prebivalcev.
19. Prikažite imena držav in BDP na prebivalca za prvih 60 najbogatejših držav.
20. Pokažite imena držav in BDP na prebivalca za 10 % najbogatejših držav. Najprej izračunajte, koliko je 10 % vseh držav, in rezultat uporabite pri poizvedbi.
21. Prikažite povprečni BDP na prebivalca po regijah. Spisek naj bo urejen naraščajoče po povprečnem BDP.
22. Izpišite ime regije in površino najmanjše države za Evropo, Azijo, Afriko in Oceanijo.
23. Izpišite države in njihove površine za vse države v bazi, pri katerih je površina med polovico in dvakratnikom površine Slovenije.

Rešitve na strani 118.

2.6 Povezovanje tabel

SQL omogoča povezovanje dveh ali več tabel. Pogledali si bomo načine povezovanja dveh tabel. Na podoben način lahko povežemo tudi več tabel.

2.6.1 Povezovanje z enačajem

Najbolj običajno povezovanje izvedemo s pomočjo enačaja.

Sintaksa:

```
SELECT stolpec1, stolpec2, ...
FROM tabela1 t1, tabela2 t2
WHERE t1.stolpec1 = t2.stolpec2;
```

Primer:

```
SELECT *
FROM stranke s, pošte p
WHERE s.poštna_št=p.poštna_št;
```

Rezultat:

```
+---------+-----------+-----------+-------------+
| ime     | poštna_št | poštna_št | pošta       |
+---------+-----------+-----------+-------------+
| Janez   |      2000 |      2000 | Maribor     |
| Nina    |      3000 |      3000 | Celje       |
| Anton   |      4000 |      4000 | Kranj       |
| Anja    |      6000 |      6000 | Koper       |
| Janja   |      4000 |      4000 | Kranj       |
| Janja   |      1000 |      1000 | Ljubljana   |
| Timotej |      4000 |      4000 | Kranj       |
| NULL    |      5000 |      5000 | Nova Gorica |
+---------+-----------+-----------+-------------+
```

2.6.2 Povezovanje z ukazom JOIN

Do enakega rezultata kot pri povezovanju z enačajem pridemo s pomočjo ukaza JOIN.

Sintaksa:

```
SELECT stolpec1, stolpec2, ...
FROM tabela1 t1
JOIN tabela2 t2
ON t1.stolpec1 = t2.stolpec2;
```

Primer:

```
SELECT *
FROM stranke s
JOIN pošte p
ON s.poštna_št=p.poštna_št;
```

2.6.3 Uporaba ukaza USING

Podoben rezultat kot pri prejšnjih dveh načinih dosežemo s pomočjo ukaza USING.

Sintaksa:

```
SELECT stolpec1, stolpec2, ...
FROM tabela1 t1
JOIN tabela2 t2
USING (stolpec1);
```

Primer:

```
SELECT *
FROM stranke s
JOIN pošte p
USING (poštna_št);
```

Rezultat:

```
+-----------+---------+-------------+
| poštna_št | ime     | pošta       |
+-----------+---------+-------------+
|      2000 | Janez   | Maribor     |
|      3000 | Nina    | Celje       |
|      4000 | Anton   | Kranj       |
|      6000 | Anja    | Koper       |
|      4000 | Janja   | Kranj       |
|      1000 | Janja   | Ljubljana   |
|      4000 | Timotej | Kranj       |
|      5000 | NULL    | Nova Gorica |
+-----------+---------+-------------+
```

2.6.4 Uporaba ukaza LEFT JOIN

LEFT JOIN nam vrne tudi tiste zapise iz »leve« tabele, ki nimajo povezave z »desno« tabelo. Leva tabela pomeni levo od ukaza LEFT JOIN.

Sintaksa:

```
SELECT stolpec1, stolpec2, ...
FROM tabela1 t1
LEFT JOIN tabela2 t2
ON t1.stolpec1 = t2.stolpec2;
```

Primer:

```
SELECT *
FROM stranke s
LEFT JOIN pošte p
ON s.poštna_št=p.poštna_št;
```

Rezultat:

```
+---------+-----------+-----------+-------------+
| ime     | poštna_št | poštna_št | pošta       |
+---------+-----------+-----------+-------------+
| Janez   |      2000 |      2000 | Maribor     |
| Nina    |      3000 |      3000 | Celje       |
| Janko   |      NULL |      NULL | NULL        |
| Anton   |      4000 |      4000 | Kranj       |
| Anja    |      6000 |      6000 | Koper       |
| Janja   |      4000 |      4000 | Kranj       |
| Janja   |      1000 |      1000 | Ljubljana   |
| Timotej |      4000 |      4000 | Kranj       |
| NULL    |      5000 |      5000 | Nova Gorica |
+---------+-----------+-----------+-------------+
```

2.6.5 Uporaba ukaza RIGHT JOIN

Analogno LEFT JOIN ukazu nam RIGHT JOIN vrne tudi tiste zapise iz »desne« tabele, ki nimajo povezave z »levo« tabelo.

Sintaksa:

```
SELECT stolpec1, stolpec2, ...
FROM tabela1 t1
RIGHT JOIN tabela2 t2
ON t1.stolpec1 = t2.stolpec2;
```

Primer:

```
SELECT *
FROM stranke s
RIGHT JOIN pošte p
ON s.poštna_št=p.poštna_št;
```

Rezultat:

```
+---------+-----------+-----------+-------------+
| ime     | poštna_št | poštna_št | pošta       |
+---------+-----------+-----------+-------------+
| Janja   |      1000 |      1000 | Ljubljana   |
| Janez   |      2000 |      2000 | Maribor     |
| Nina    |      3000 |      3000 | Celje       |
| Anton   |      4000 |      4000 | Kranj       |
| Janja   |      4000 |      4000 | Kranj       |
| Timotej |      4000 |      4000 | Kranj       |
| NULL    |      5000 |      5000 | Nova Gorica |
| Anja    |      6000 |      6000 | Koper       |
| NULL    |      NULL |      7000 | NULL        |
| NULL    |      NULL |      NULL | Brežice     |
+---------+-----------+-----------+-------------+
```

2.6.6 Uporaba ukaza LEFT JOIN namesto RIGHT JOIN

Če zamenjamo vrstni red tabel, lahko prevedemo RIGHT JOIN v LEFT JOIN in obratno.

Sintaksa:

```
SELECT stolpec1, stolpec2, ...
FROM tabela1 t1
LEFT JOIN tabela2 t2
ON t1.stolpec1 = t2.stolpec2;
```

je enakovredna

```
SELECT stolpec1, stolpec2, ...
FROM tabela2 t2
RIGHT JOIN tabela1 t1
ON t1.stolpec1 = t2.stolpec2;
```

Primer:

```
SELECT *
FROM pošte p
LEFT JOIN stranke s
ON s.poštna_št=p.poštna_št;
```

Rezultat:

```
+-----------+-------------+---------+-----------+
| poštna_št | pošta       | ime     | poštna_št |
+-----------+-------------+---------+-----------+
|      1000 | Ljubljana   | Janja   |      1000 |
|      2000 | Maribor     | Janez   |      2000 |
|      3000 | Celje       | Nina    |      3000 |
|      4000 | Kranj       | Anton   |      4000 |
|      4000 | Kranj       | Janja   |      4000 |
|      4000 | Kranj       | Timotej |      4000 |
|      5000 | Nova Gorica | NULL    |      5000 |
|      6000 | Koper       | Anja    |      6000 |
|      7000 | NULL        | NULL    |      NULL |
|      NULL | Brežice     | NULL    |      NULL |
+-----------+-------------+---------+-----------+
```

2.7 Vgrajene funkcije

Za večino programskih jezikov je značilno, da imajo nekatere pogosto uporabljane funkcije že vgrajene in tudi MySQL ni izjema. Razdelili jih bomo glede na podatkovne tipe, na katerih funkcije delujejo. Tako poznamo številske, znakovne, časovne, dvojiške in še nekatere druge funkcije.

2.7.1 Časovne funkcije

Kot že ime pove, časovne funkcije manipulirajo s časom. Gre za aritmetične operacije nad časovnimi vrednostmi, pridobivanje trenutnega časa in datuma, pretvarjanjem iz ene oblike v drugo, izluščevanje dela časovne vrednosti oz. sestavljanje vrednosti iz posameznih enot, če naštejemo samo najbolj pogosto uporabljane.

Vseh časovnih funkcij ne bomo obravnavali posamezno, saj jih je preveč in bi vzelo preveč časa. Za nekaj najbolj uporabnih bomo naredili primere, nekaj pa jih bomo spoznali preko vaj, ki sledijo na koncu razdelka.

Spodaj je tabela vgrajenih časovnih funkcij, tako da bo bralec sam lahko preskusil tiste, ki jih ne bomo skupaj.

Ime	**Opis**
ADDDATE (datum, INTERVAL vrednost časovna_enota), ADDDATE (datum, dni)	Prišteje časovno vrednost (interval) datumu.
ADDTIME (izraz1, izraz2)	Vrne izraz1+izraz2, kjer je izraz1 tipa time ali datetime, izraz2 pa tipa time.
CONVERT_TZ (datum_čas, časovni_pas_1, časovni_pas_2)	Pretvori datum_čas tipa datetime iz časovnega pasa časovni_pas_1 v časovni pas časovni_pas_2.

CURDATE()	Vrne trenutni datum.
CURRENT_DATE(), CURRENT_DATE	Sinonima za CURDATE().
CURRENT_TIME(), CURRENT_TIME	Sinonima za CURTIME().
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Sinonim za NOW().
CURTIME()	Vrne trenutni čas.
DATE_FORMAT (datum, format)	Oblikuje datum, kot je navedeno.
DATE_ADD (datum, INTERVAL vrednost časovna_enota), DATE_SUB (datum, INTERVAL vrednost časovna_enota)	Prišteje ali odšteje časovno vrednost (interval) datumu.
DATE (datum)	Izlušči datumski del iz datetime izraza.
DATEDIFF (datum1, datum2)	Odšteje: datum1 - datum2.
DAY (datum)	Sinonim za DAYOFMONTH().
DAYNAME (datum)	Vrne ime dneva v tednu v odvisnosti od sistemske spremenljivke lc_time_names. Za slovenske dneve nastavite: SET lc_time_names = 'sl_SI';
DAYOFMONTH (datum)	Vrne dan v mesecu (0–31).
DAYOFWEEK (datum)	Vrne dan v tednu (1 = nedelja, 2 = ponedeljek, ..., 7 = sobota).
DAYOFYEAR (datum)	Vrne dan v letu (1–366).
EXTRACT (enota FROM datum)	Izlušči del datuma v izbrani enoti.
FROM_DAYS(x)	Pretvori dan v obliki števila v tip date.
FROM_UNIXTIME(x)	Pretvori datum iz UNIX-ove oblike v tip date.
GET_FORMAT({DATE I TIME | DATETIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})	Vrne niz, ki določa obliko datuma. Uporabno predvsem v kombinaciji z DATE_FORMAT() in STR_TO_DATE() funkcijama.
HOUR (čas)	Izlušči uro iz časa.
LAST_DAY (datum)	Vrne zadnji dan izbranega meseca oz. NULL, če datum ni veljaven.
LOCALTIME(), LOCALTIME	Sinonim za NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Sinonim za NOW()
MAKEDATE (leto, dan_v_letu)	Vrne datum za določen dan v letu. Dan v letu mora biti večji od 0, sicer vrne NULL.
MAKETIME (ure, minute, sekunde)	Iz danih podatkov vrne časovno vrednost tipa time.
MICROSECOND()	Vrne mikrosekunde podanega časa.
MINUTE()	Izlušči minute iz časa.
MONTH()	Vrne mesec podanega argumenta.
MONTHNAME()	Vrne ime meseca v odvisnosti od sistemske spremenljivke lc_time_names. Za slovenske mesece nastavite:

	SET lc_time_names = 'sl_SI';
NOW()	Vrne trenutni datum in čas.
PERIOD_ADD (obdobje, meseci)	Obdobju prišteje podano število mesecev. Obdobje ni časovna vrednost.
PERIOD_DIFF (obdobje1, obdobje2)	Vrne, koliko mesecev je med podanima obdobjema.
QUARTER (datum)	Vrne četrtletje podane vrednosti (1–4).
SEC_TO_TIME (sekunde)	Pretvori sekunde v obliko 'HH:MM:SS'.
SECOND (čas)	Izlušči sekunde iz časa (0–59).
STR_TO_DATE (niz, oblika)	Pretvori niz v čas.
SUBDATE (datum, INTERVAL vrednost časovna_enota), SUBDATE (datum, dni)	Odšteje časovno vrednost (interval) od datuma.
SUBTIME (časovna_enota, čas)	Odšteje drugi argument od prvega.
SYSDATE()	Vrne trenutni čas.
TIME_FORMAT (čas, format)	Oblikuje čas, kot je navedeno.
TIME_TO_SEC (čas)	Pretvori čas v sekunde.
TIME (časovna_enota)	Izlušči čas iz argumenta.
TIMEDIFF (časovna_enota1, časovna_enota2)	Odšteje (časovna_enota1 - časovna_enota2). Argumenta morata biti istega podatkovnega tipa.
TIMESTAMP (časovna_enota), TIMESTAMP (časovna_enota1, časovna_enota2)	Z enim argumentom vrne vrednost tipa datetime; z dvema argumentoma pa vsoto prav tako tipa datetime.
TIMESTAMPADD (enota, interval, časovna_enota)	Prišteje interval datetime izrazu v podani enoti.
TIMESTAMPDIFF (enota, časovna_enota1, časovna_enota2)	Odšteje dva datetime izraza. Rezultat vrne v podani enoti.
TO_DAYS (datum)	Pretvoti datum v dneve (število dni od leta 0).
TO_SECONDS()	Pretvoti datum v sekunde (število sekund od leta 0).
UNIX_TIMESTAMP(), UNIX_TIMESTAMP (datum_čas)	Brez argumenta vrne število sekund od '1970-01-01 00:00:00' UTC, z argumetom število sekund od '1970-01-01 00:00:00' UTC do vrednosti argumenta.
UTC_DATE, UTC_DATE()	Vrne trenutni UTC datum v obliki 'YYYY-MM-DD' ali YYYYMMD, odvisno od tega, ali je uporabljen v besedilnem ali številskem kontekstu.
UTC_TIME, UTC_TIME()	Vrne trenutni UTC čas v obliki 'HH:MM:SS' ali HHMMSS.uuuuuu, odvisno od tega, ali je uporabljen v besedilnem ali številskem kontekstu.
UTC_TIMESTAMP, UTC_TIMESTAMP()	Vrne trenutni UTC datum in čas v obliki 'YYYY-MM-DD HH:MM:SS' ali YYYYMMDDHHMMSS.uuuuuu, odvisno od

	tega, ali sta uporabljena v besedilnem ali številskem kontekstu.
WEEK (datum[, način])	Vrne teden v letu, odvisno od načina (0–7). Privzeta vrednost za način je 0.
WEEKDAY (datum)	Vrne indeks dneva v tednu (0 = ponedeljek, 1 = torek, ..., 6 = nedelja).
WEEKOFYEAR (datum)	Vrne teden v letu (0–53). Sinonim za WEEK (datum, 3).
YEAR (datum)	Izlušči leto iz datuma (1000–9999).
YEARWEEK (datum)	Vrne leto in teden.

Tabela 1: Tabela vgrajenih časovnih funkcij.

2.7.1.1 Časovne oznake, enote in oblike ter način številčenja tednov v letu

Pri pretvarjanju in izluščevanju uporabljamo časovne oznake, enote in oblike. Časovne oblike so različni načini prikaza časovnih enot. Tako lahko na primer mesec marec prikažemo številčno, številčno z začetno ničlo, z okrajšavo ali s celo besedo v različnih jezikih. Enote so osnovne merske enote za merjenje časa, kot so leto, dan, ura ,in tudi sestavljene, kot sta datum in čas. Oblike se nanašajo predvsem na mednarodne zapise časa, npr. evropski ali ameriški zapis za datum. Način številčenja tednov v letu pa določa, kako začnemo s tem številčenjem. Poglejmo vse to na primerih.

Z uporabo funkcije DATE_FORMAT bomo izluščili mesec iz rojstnega dneva Michelangela (6. 3. 1475). Privzeta oblika vnašanja časa je v skladu z ISO standardi, torej v obliki YYYY-MM-DD (leto-mesec-dan). Michelangelov rojstni dan v ISO obliki je '1475-03-06'. Seveda lahko vnašamo tudi v drugih oblikah, a to po vsej verjetnosti nima prav posebnega pomena, saj imamo funkcije, ki nam pretvarjajo datume v želene oblike. Drugače je seveda pri izpisih.

Primer:

```
SELECT DATE_FORMAT('1475-03-06', '%b') b,
DATE_FORMAT('1475-03-06', '%c') c,
DATE_FORMAT('1475-03-06', '%M') M,
DATE_FORMAT('1475-03-06', '%m') m;
```

Rezultat:

```
+------+------+-------+------+
| b    | c    | M     | m    |
+------+------+-------+------+
| Mar  | 3    | March | 03   |
+------+------+-------+------+
```

Ker nismo zadovoljni z angleškim izpisom, bomo s pomočjo prirejanja vrednosti sistemski spremenljivki lc_time_names določili, da bodo časovna imena izpisana v slovenščini.

```
SET lc_time_names = 'sl_SI';
```

Ob ponovni izvedbi prejšnjega stavka dobimo naslednji rezultat:

```
+------+------+-------+------+
| b    | c    | M     | m    |
+------+------+-------+------+
| mar  | 3    | marec | 03   |
+------+------+-------+------+
```

Sledijo zgoraj omenjene tabele.

Časovne oznake

Oznaka	**Opis**
%a	Okrajšano ime dneva (Sun … Sat)
%b	Okrajšano ime meseca (Jan … Dec)
%c	Mesec, številčno (0 … 12)
%D	Dan v mesecu z angleško pripono (0th, 1st, 2nd, 3rd …)
%d	Dan v mesecu, številčno (00 … 31)
%e	Dan v mesecu, številčno (0 ... 31)
%f	Mikrosekunde (000000 … 999999)
%H	Ura (00 ... 23)
%h	Ura (01 ... 12)
%I	Ura (01 ... 12)
%i	Minute, številčno (00 … 59)
%j	Dan v letu (001 … 366)
%k	Ura (0 … 23)
%l	Ura (1 … 12)
%M	Ime meseca (Januar … December)
%m	Mesec, številčno (00 … 12)
%p	AM ali PM
%r	Čas, 12-urno (hh:mm:ss skupaj z AM ali PM)
%S	Sekunde (00 … 59)
%s	Sekunde (00 … 59)
%T	Čas, 24-urno (hh:mm:ss)
%U	Teden (00 … 53), kjer je nedelja prvi dan v tednu
%u	Teden (00 ... 53), kjer je ponedeljek prvi dan v tednu
%V	Teden (01 ... 53), kjer je nedelja prvi dan v tednu; v kombinaciji z %X
%v	Teden (01 ... 53), kjer je ponedeljek prvi dan v tednu; v kombinaciji z %x
%W	Ime dneva v tednu (nedelja ... sobota)
%w	Dan v tednu (0=nedelja ... 6=sobota)
%X	Leto za teden, kjer je nedelja prvi dan v tednu, numeric, four digits; v kombinaciji z %V
%x	Leto za teden, kjer je ponedeljek prvi dan v tednu, numeric, four digits; v kombinaciji z %v

%Y	Leto, številčno, štiri številke
%y	Leto, številčno, dve številki
%%	Znak "%"
%x	x, for any "x" not listed above

Tabela 2: Časovne oznake.

Časovne enote

Enota	Pričakovana oblika izraza
MICROSECOND	mikrosekunde
SECOND	sekunde
MINUTE	minute
HOUR	ure
DAY	dnevi
WEEK	tedni
MONTH	meseci
QUARTER	četrtletja
YEAR	leta
SECOND_MICROSECOND	'sekunde.mikrosekunde'
MINUTE_MICROSECOND	'minute:sekunde.mikrosekunde'
MINUTE_SECOND	'minute:sekunde'
HOUR_MICROSECOND	'ure:minute:sekunde.mikrosekunde'
HOUR_SECOND	'ure:minute:sekunde'
HOUR_MINUTE	'ure:minute'
DAY_MICROSECOND	'dnevi ure:minute:sekunde.mikrosekunde'
DAY_SECOND	'dnevi ure:minute:sekunde'
DAY_MINUTE	'dnevi ure:minute'
DAY_HOUR	'dnevi ure'
YEAR_MONTH	'leta-meseci'

Tabela 3: Časovne enote.

Časovne oblike

Klic funkcije	Rezultat
GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'

GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%s'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

Tabela 4: Časovne oblike.

Način številčenja tednov v letu

Način	Prvi dan v tednu	Interval	Teden št. 1 je prvi teden ...
0	Nedelja	0-53	... z nedeljo v tem letu.
1	Ponedeljek	0-53	... z več kot tremi dnevi v tem letu.
2	Nedelja	1-53	... z nedeljo v tem letu.
3	Ponedeljek	1-53	... z več kot tremi dnevi v tem letu.
4	Nedelja	0-53	... z več kot tremi dnevi v tem letu.
5	Ponedeljek	0-53	... s ponedeljkom v tem letu.
6	Nedelja	1-53	... z več kot tremi dnevi v tem letu.
7	Ponedeljek	1-53	... s ponedeljkom v tem letu.

Tabela 5: Način številčenja tednov v letu.

2.7.2 Vaje – Časovne funkcije

Uvozite tabelo s podatki »rojstni_dnevi.sql«[3] in izvedite naslednje poizvedbe:

1. Razvrstite osebe od najstarejše do najmlajše.
2. Izpišite dve najmlajši ženski in dva najstarejša moška.
3. Izpišite vse, ki bodo imeli rojstni dan v naslednjih 90 dneh.
4. Poiščite vse, ki so stari med 10000 in 20000 dnevi.
5. Poiščite vse, ki so rojeni med drugo svetovno vojno in niso rojeni meseca septembra. (1939-09-01 do 1945-09-02)
6. Koliko oseb je rojenih v posameznem mesecu?
7. Izpišite mesece, v katerih je bilo rojeno vsaj sedem oseb.
8. Izpišite vse pare oseb, ki so bili rojeni manj kot 15 dni narazen.
9. Izpišite vse pare oseb, katerih rojstni dnevi se razlikujejo za manj kot 5 dni.
10. Koliko dni ste stari vi?
11. Izpišite vse, ki so bili rojeni v petek trinajstega.

Rešitve na strani 122.

2.7.3 Številske funkcije

Številske funkcije operirajo s števili. Gre predvsem za matematične funkcije, pretvorbo med številskimi sistemi, zaokroževanja in generiranje naključnih števil. Funkcija PI() pa nam vrne vrednost Ludolfovega števila na šest decimalnih mest, hrani pa ga v dvojni natančnosti (podatkovni tip DOUBLE).

[3] http://sterle.tsckr.si/NUB/prenosi/rojstni_dnevi.sql (Vse datoteke, uporabljene v tem gradivu, lahko najdemo v http://sterle.tsckr.si/NUB/prenosi/.)

Številske funkcije

Ime	Opis
ABS(x)	Vrne absolutno vrednost.
ACOS(x)	Vrne arkus kosinus.
ASIN(x)	Vrne arkus sinus.
ATAN2(y, x), ATAN(y, x)	Vrne arkus tangens dveh argumentov.
ATAN(x)	Vrne arkus tangens.
CEIL(x), CEILING(x)	Vrne najmanjše celo število, ki ni manjše od argumenta. Zaokroži navzgor.
CONV(x, iz_baze, v_bazo)	Pretvarja števila med številskimi sistemi. Največja možna baza je 36.
COS(x)	Vrne kosinus.
COT(x)	Vrne kotangens.
CRC32(izraz)	Izračuna CRC vrednost.
DEGREES(x)	Pretvori radiane v stopinje.
EXP(x)	Vrne e^x.
FLOOR(x)	Vrne največje celo število, ki ni večje od argumenta. Zaokroži navzdol.
LN(x), LOG(x)	Vrne naravni logaritem argumenta.
LOG10()	Vrne logaritem z osnovo 10 danega argumenta.
LOG2()	Vrne logaritem z osnovo 2 danega argumenta.
LOG(b, x)	Vrne logaritem z osnovo b argumenta x.
MOD(m, n)	Vrne ostanek pri celoštevilskem deljenju m-ja z n.
OCT(x)	Vrne osmiško vrednost desetiškega števila.
PI()	Vrne število π.
POW(x, y), POWER(x, y)	Vrne x^y.
RADIANS(x)	Pretvori stopinje v radiane.
RAND(), RAND(x)	Vrne naključno število tipa float, večje ali enako 0 in manjše od 1. Parameter pomeni seme, ki poraja ponavljajoča zaporedja.
ROUND(x), ROUND(x, d)	Zaokroži argument na d decimalnih mest. Če d spustimo, prevzame privzeto vrednost 0, kar pomeni zaokroževanje na celo število. d je lahko tudi negativen. To povzroči, da d števk levo od decimalnega ločila postane 0.
SIGN(x)	Vrne -1, 0 ali 1, odvisno od tega, če je argument negativen, enak nič ali pozitiven.
SIN(x)	Vrne sinus argumenta.
SQRT(x)	Vrne kvadratni koren argumenta.
TAN(x)	Vrne tangens argumenta.
TRUNCATE(x, d)	Odreže število x na d decimalnih mest. d je lahko tudi negativen. To povzroči, da d števk levo od decimalnega ločila postane 0.

Tabela 6: Številske funkcije.

Oglejmo si izpis števila π, izračun cos(0,5) in zaokroževanje.

Primer:

```
SELECT PI() pi, PI()+0.000000000000000 pi_dvojno,
 cos(0.5) kosinus, ROUND(cos(0.5)) zaokr;
```

Rezultat:

```
+----------+-------------------+--------------------+-------+
| pi       | pi_dvojno         | kosinus            | zaokr |
+----------+-------------------+--------------------+-------+
| 3.141593 | 3.141592653589793 | 0.8775825618903728 |     1 |
+----------+-------------------+--------------------+-------+
```

2.7.4 Znakovne funkcije

Znakovne funkcije nam olajšajo delo z nizi. Tako z njimi lahko združujemo nize, izberemo samo del niza, iščemo ali zamenjamo podnize z drugimi, jih pretvarjamo, oblikujemo, primerjamo, primerjamo z vzorci in celo z regularnimi izrazi.

Ime	**Opis**
ASCII(niz)	Vrne numerično vrednost znaka **niz**a, ki je najbolj levo.
BIN()	Pretvori **n** v binarno število in ga vrne kot niz. n je največ tipa BIGINT. Ekvivalentno CONV(n,10,2).
BIT_LENGTH(niz)	Vrne dolžino **niz**a v bitih.
CHAR_LENGTH(niz)	Vrne število znakov **niz**a.
CHAR(N,... [USING nabor_znakov])	Vrne znak za vsako število.
CHARACTER_LENGTH(niz)	Sinonim za CHAR_LENGTH(niz).
CONCAT_WS(ločilo, niz1, niz2, ...)	Zlepi **nize** skupaj z **ločilom**.
CONCAT(niz1, niz2, ...)	Zlepi **nize**.
ELT(n, niz1, niz2, ...)	Vrne **niz** s podano zaporedno številko (indeksom).
EXPORT_SET(biti, vključen, izključen[, ločilo[, število_bitov]])	Vrne niz bitov argumenta **biti,** ločenih z ločilom **ločilo**. Namesto enk imamo znak **vključen**, namesto ničle **izključen**. Lahko podamo še **število_bitov**, ki jih želimo imeti v nizu. Biti naraščajo od leve proti desni.
FIELD(niz1, niz2, ...)	Vrne indeks (mesto) prvega argumenta med ostalimi.
FIND_IN_SET(niz1, nizi)	Vrne indeks (mesto) prvega argumenta v drugem.
FORMAT(x, d[, lokalno])	Vrne število **x** kot niz v obliki '#,###,###.##' na **d** decimalnih mest v odvisnosti od lc_time_names. Za slovenski zapis nastavite: SET lc_time_names = 'sl_SI';

FROM_BASE64(niz)	Pretvori **niz** v kodiranje base-64 in vrne niz. Obratno kot TO_BASE64().
HEX(niz), HEX	Vrne šestnajstiško predstavitev argumenta. Za vsak znak dva šestnajstiška znaka. Obratno kot UNHEX().
INSERT(niz, mesto, dolžina, nov_niz)	Vstavi **nov_niz** v **niz** na **mesto** in nadomesti največ **dolžina** znakov.
INSTR(niz, podniz)	Vrne mesto prvega mesta **podniz**a v **niz**u.
LCASE(niz)	Sinonim za LOWER()
LEFT(niz, n)	Vrne prvih **n** znakov iz **niz**a.
LENGTH(niz)	Vrne dolžino **niz**a v bajtih.
LIKE	Preprost način primerjanja z vzorci
LOAD_FILE(datoteka)	Prenese datoteko na strežnik. Navesti morate celotno pot do datoteke.
LOCATE(podniz, niz[, mesto])	Vrne mesto prvega mesta **podniz**a v **niz**u od **mesta** naprej. Če podniza ni v nizu, vrne 0.
LOWER(niz)	Pretvori **niz** v male črke.
LPAD(niz, dolžina, polnilo)	**niz** podaljša na **dolžina** znakov. Z leve zapolni s **polnilo**m. Če je dolžina krajša od niza, ga skrajša.
LTRIM(niz)	Odstrani vodilne presledke.
MAKE_SET(biti, niz1, niz2, …)	Vrne množico tistih izmed naštetih nizov, ki ustrezajo mestom, ki jih določajo biti.
MATCH	Iskanje po celotnem besedilu.
MID(niz,mesto,dolžina)	Sinonim za SUBSTRING (niz,mesto,dolžina).
NOT LIKE	Negacija preprostega primerjanja vzorca.
NOT REGEXP	Negacija REGEXP
OCTET_LENGTH()	Sinonim za LENGTH()
ORD(niz)	Vrne kodo prvega znaka **niz**a. Če je znak sestavljen iz več bajtov, vrne vrednost po naslednji formuli: (koda prvega bajta) + (koda drugega bajta * 256) + (koda tretjega bajta * 256^2) ...
POSITION()	Sinonim za LOCATE()
QUOTE(niz)	Ustvari niz, ki ustreza vnosu. Upošteva tudi posebne znake.
REGEXP	Primerjanje z vzorcem s pomočjo regularnih izrazov.
REPEAT(niz, n)	Ponovi **niz** **n**-krat.
REPLACE(niz, stari_niz, novi_niz)	V **niz**u zamenja vse pojavitve starega niza z novim nizom. Funkcija je občutljiva na velikost črk.
REVERSE(niz)	Zamenja vrstni red znakov v **niz**u.
RIGHT(niz, n)	Vrne **zadnjih n** znakov iz **niz**a.
RLIKE	Sinonim za REGEXP.
RPAD(niz, dolžina, polnilo)	**niz** podaljša na **dolžina** znakov. Z desne zapolni

	s **polnilo**m. Če je dolžina krajša od niza, ga skrajša.
RTRIM(niz)	Odstrani zaključne presledke.
SOUNDEX(niz)	Vrne niz po soundex algoritmu.
izraz1 SOUNDS LIKE izraz2	Sinonim za SOUNDEX(izraz1) = SOUNDEX(izraz2).
SPACE	Vrne niz dolžine **n** samih presledkov.
STRCMP(niz1, niz2)	Primerja dva niza. Vrne 0, če sta enaka, -1, če je prvi manjši od drugega, in 1 sicer.
SUBSTR()	Sinonim za SUBSTRING().
SUBSTRING_INDEX(niz, ločilo, n)	Vrne podniz **niz**a do mesta, kjer se **n**-tič pojavi **ločilo**.
SUBSTRING(niz,mesto), SUBSTRING(niz FROM mesto), SUBSTRING(niz,mesto,dolžina), SUBSTRING(niz FROM mesto FOR dolžina)	Vrne podniz.
TO_BASE64()	Pretvori **niz** iz kodiranja base-64 in vrne niz. Obratno kot FROM_BASE64().
TRIM()	Odstrani vodilne in zaključne presledke.
UCASE()	Sinonim za UPPER()
UNHEX()	Pretvori vsak šestnajstiški par v znak.
UPPER()	Pretvori v velike črke.
WEIGHT_STRING()	Vrne "težo" niza. Uporablja se za testiranje v odvisnosti od naborov znakov.

Tabela 7: Znakovne funkcije.

2.7.5 Vaje – Številske in znakovne funkcije

Uvozite tabelo s podatki »rojstni_dnevi.sql«[4] in izvedite naslednje poizvedbe:

1. Tabeli rojstni_dnevi dodajte stolpec začetnice in ga napolnite z začetnicami priimka in imena.
2. V tabeli rojstni_dnevi trem naključnim zapisom dodajte po dva presledka na začetek in na konec.
3. Izpišite te zapise.
4. Posodobite tabelo, tako da teh presledkov ne bo več.
5. Iz tabele rojstni_dnevi izpišite vse podatke za osebe, katerih ime se začne na isto črko, kot se konča priimek.
6. Iz tabele rojstni_dnevi izpišite vse podatke za osebe, katerih mesto v abecedi prve črke priimka je enako dnevu rojstva.
7. Poiščite tiste osebe, ki imajo ime enako dolgo priimku.
8. Preverite, ali je stavek palindrom, tj. stavek, ki se prebere naprej in nazaj enako.

[4] http://sterle.tsckr.si/NUB/prenosi/rojstni_dnevi.sql (Vse datoteke, uporabljene v tem gradivu, lahko najdemo v http://sterle.tsckr.si/NUB/prenosi/.)

Primera: Perica reže raci rep. Ali se bo Gordana na drog obesila?

a) V poizvedbi napišite stavek s presledki, z velikimi in malimi črkami, vendar brez ločil. Vrne naj vrednost: 1 je palindrom oz. 0 ni palindrom.
b) Kaj pa, če imamo napisana še vsa ločila?

Rešitve na strani 124.

2.8 Vstavljanje podatkov (INSERT)

INSERT stavek doda nov zapis v tabelo. V najpreprostejši obliki ključni besedi INSERT sledijo ime tabele, ključna beseda VALUES in seznam vrednosti, ločenih z vejico, ki pripadajo posameznim poljem tabele. Seveda moramo v tem primeru vnesti vrednosti za vsa polja tabele.

```
INSERT INTO ime_tabele
VALUES (vrednost_1, vrednost_2, ...,vrednost_n);
```

Primer:

```
INSERT INTO države
VALUES (207, 'Iskra', 'TŠC', 100, 700, 0);
```

INTO lahko izpustimo, a ga ponavadi pišemo zaradi boljše berljivosti.

Lahko vnesemo samo nekaj polj, seveda ostala ne smejo imeti atributa NOT NULL.

```
INSERT INTO ime_tabele (polje_1, polje_2, ...,polje_n)
VALUES (vrednost_1, vrednost_2, ...,vrednost_n);
```

Primer:

```
INSERT INTO države (št, država)
VALUES (208, 'Indija Koromandija');
```

Lahko vnesemo tudi več vrstic naenkrat. Sintaksa za vnos M vrstic naenkrat:

```
INSERT INTO ime_tabele (polje_1, polje_2, ...,polje_n)
VALUES (vrednost_1A, vrednost_2A, ...,vrednost_nA),
(vrednost_1B, vrednost_2B, ...,vrednost_nB),
...
(vrednost_1M, vrednost_2M, ...,vrednost_nM) ;
```

Primer:

```
INSERT INTO države (št, država)
VALUES (209, '4.Ra'), (210, '4.Rb');
```

Če imamo oba seznama polj in vrednosti prazna, nam vstavi privzete vrednosti.

```
INSERT INTO ime_tabele () VALUES();
```

Primer:

```
INSERT INTO države ()
VALUES ();
```

Zelo uporabna je tudi povezava INSERT in SELECT stavkov. V tem primeru lahko zelo preprosto vstavimo podatke iz ene ali več tabel v drugo.

```
INSERT INTO ime_tabele
SELECT ...;
```

ali

```
INSERT INTO ime_tabele (polje_1, polje_2, ...,polje_n)
SELECT ...;
```

Prvo sintakso uporabimo v primeru, da vnašamo vsa polja tabele, drugo pa, ko želimo vnašati samo določene stolpce.

Primer:

```
INSERT INTO države
SELECT *
 FROM države;
```

Uganete, kaj naredi zadnji SQL stavek?

2.9 Brisanje podatkov (DELETE)

Stavek DELETE se uporablja za brisanje vrstic iz tabel. Sintaksa je podobna SELECT stavku, saj besedi FROM sledi ime tabele, besedi WHERE pa pogoj, ki filtrira vrstice, ki jih želimo zbrisati.

Brez WHERE pogoja DELETE zbriše celo tabelo, zato je potrebna previdnost.

```
DELETE FROM ime_tabele WHERE pogoj;
```

Primer:

```
DELETE FROM države WHERE št=210;
```

2.10 Posodabljanje podatkov (UPDATE)

Stavek UPDATE se uporablja za popravljanje podatkov tabel. Ključni besedi UPDATE sledi ime tabele, nato pa za besedico SET določamo polja in njihove nove vrednosti, sledi WHERE pogoj, ki filtrira vrstice, ki jih želimo popraviti.

Brez WHERE pogoja UPDATE popravi vsako vrstico tabele, zato je tudi tu potrebna previdnost.

```
UPDATE ime_tabele
   SET polje_1={vrednost_1|DEFAULT} [, polje_2={vrednost_2|DEFAULT}]
...
   [WHERE pogoj]
   [ORDER BY ...]
   [LIMIT število_vrstic];
```

Primer:

```
UPDATE države
   SET BDP=10
   WHERE država = 'Iskra';
```

2.11 Vaje - INSERT, DELETE in UPDATE

1. V podatkovni zbirki ustvarite novo tabelo z imenom "nove_države" in vanjo prepišite vse podatke za evropske države iz tabele "države".
2. V tabelo nove_države dodajte državi "Srbija" in "Črna gora".
3. Poiščite največjo vrednost v polju "št" in novima državama dodelite naslednji dve zaporedni številki.
4. Na spletu poiščite ostale podatke za ti dve državi in popravite podatke.
5. Zbrišite državo "Srbija in Črna gora".
6. V tabeli nove_države pretvorite površino iz kvadratnih kilometrov v kvadratne milje. (1 kvadratni kilometer = 0.386102159 kvadratne milje, obratno 2.58998811).
7. V tabelo nove_države dodajte državo "Indija Koromandija" iz regije "Nije", ki ima eno kvadratno miljo in pet prebivalcev.
8. V tabelo nove_države dodajte vse države iz tabele države, imajo površino manjšo kot 20000 kvadratnih kilometrov. Hkrati pretvorite kvadratne kilometre v kvadratne milje.
9. V tabeli nove_države za 100000 znižajte BDP vsem državam, ki imajo manj kot 3000000 prebivalcev.
10. V tabeli nove_države povečajte prebivalstvo za 10 % državam, ki se začnejo na črko A.
11. V tabeli nove_države odstranite države, ki imajo površino med 4000 in 5000 kvadratnih milj.
12. V tabeli nove_države odstranite vse države, ki imajo BDP večji kot 1000000000.
13. V tabeli nove_države odstranite države, ki nimajo znanega BDP-ja.

Rešitve na strani 126.

3 Jezik za definiranje podatkov (DDL – Data Definition Language)

Jezik za definiranje podatkov (DDL – Data Definition Language) obsega naslednje ukaze:

- CREATE (ustvarjanje),
- ALTER (spreminjanje),
- DROP (brisanje).

Z njimi lahko definiramo zbirke podatkov (DATABASE), tabele (TABLE), indekse (INDEX), poglede (VIEW), prožilce (TRIGGER).

3.1 Izdelava, uporaba in brisanje podatkovne zbirke

Podatkovno zbirko ustvarimo z naslednjim ukazom:

```
CREATE DATABASE ime_podatkovne_zbirke;
```

Imamo lahko več podatkovnih zbirk in nas zanima, katere so le-te. To ugotovimo z ukazom:

```
SHOW DATABASES;
```

Izbira določene podatkovne zbirke se izvede z ukazom:

```
USE ime_podatkovne_zbirke;
```

Ko določene podatkovne zbirke ne potrebujemo več, jo zbrišemo z ukazom:

```
DROP DATABASE ime_podatkovne_zbirke;
```

Primer:

```
CREATE DATABASE ladjedelnica;
USE ladjedelnica;
DROP DATABASE ladjedelnica;
```

3.2 Tabele (TABLE)

3.2.1 Izdelava tabel CREATE TABLE

Podatkovna zbirka brez tabel skoraj nima nobenega smisla. Torej? Ukaz za ustvarjanje tabel:

```
CREATE TABLE ime_tabele
(
  stolpec_1 podatkovni_tip_1,
  stolpec_2 podatkovni_tip_2,
  ...
  stolpec_n podatkovni_tip_n
);
```

Primer:

```
CREATE TABLE ladja (
  ID int, ime varchar(30), tip varchar(30),
  moč int, nosilnost int, cena int
) DEFAULT CHARSET='utf8';
```

Zadnja vrstica primera zagotavlja pravilno hranjenje in prikaz šumnikov, zato ta dodatek uporabimo pri vsaki tabeli, kjer jih bomo uporabljali.

3.2.2 Spreminjanje tabel ALTER TABLE

ALTER TABLE je namenjen dodajanju, brisanju ali spreminjanju stolpcev v tabelah.

Sintaksa za dodajanje stolpca:

```
ALTER TABLE ime_tabele
ADD ime_stolpca podatkovni_tip_stolpca;
```

Primer:

```
ALTER TABLE ladja
ADD proizvajalec varchar(30);
```

Sintaksa za brisanje stolpca:

```
ALTER TABLE ime_tabele
DROP COLUMN ime_stolpca;
```

Primer:

```
ALTER TABLE ladja
DROP COLUMN tip;
```

Sintaksa za spreminjanje stolpca:

```
ALTER TABLE ime_tabele
  CHANGE [COLUMN] ime_starega_stolpca ime_novega_stolpca
podatkovni_tip_stolpca;
```

Primer:

```
ALTER TABLE ladja
  CHANGE ime ime varchar(50);
```

Primer:

```
ALTER TABLE ladja
  CHANGE ime naziv varchar(50);
```

3.2.3 Brisanje tabel DROP TABLE

DROP TABLE je namenjen brisanju tabel.

Sintaksa za brisanje tabel:

```
DROP TABLE ime_tabele;
```

Primer:

```
DROP TABLE ladja;
```

3.3 Podatkovni tipi

Naučili smo se že ustvariti, spremeniti in zbrisati tabelo. Do sedaj smo uporabili samo dva podatkovna tipa, in sicer INT in VARCHAR(x). Poglejmo še ostale podatkovne tipe, ki jih lahko uporabljamo.

Osnovni tipi, ki jih uporabljamo, so:

- številski,
- časovni,
- besedilni.

3.3.1 Številski podatkovni tipi

Najprej bomo pregledali numerične vrednosti. Vsak numerični tip INT je lahko tudi UNSIGNED (nepredznačen), AUTO_INCREMENT (samoštevilo) ali ZEROFILL. Če je UNSIGNED, lahko zasede le pozitivne vrednosti, atribut AUTO_INCREMENT avtomatsko povečuje numerično

vrednost natanko enemu polju v tabeli, ki mora imeti vsaj enega od atributov NOT NULL, PRIMARY KEY ali UNIQUE, ZEROFILL pa zapolni začetek števila z ničlami in privzame še atribut UNSIGNED. To je uporabno npr. pri telefonskih številkah. Če bi imeli podatkovni tip INT(9) in bi vnesli telefonsko številko 041123456, bi se dejansko vneslo 41123456. Za pravilno shranjevanje zato uporabimo INT(9) ZEROFILL. Oznaka M pomeni največje število števk, D pa število mest za decimalno vejico. Poleg vrednosti INT lahko uporabljamo še sledeče tipe:

```
BIT[(M)]
```

Bitno polje. M pomeni število bitov v polju med 1 in 64. Privzeta vrednost za M je 1.

```
TINYINT[(M)] [UNSIGNED] [ZEROFILL]
```

(1 bajt) - uporabljamo ga, kadar zasede argument majhne vrednost. Lahko hrani števila med 0 in 255 (28–1), če je unsigned, ali od -128 do 127. BOOL in BOOLEAN sta sinonima za TINYINT(1), kjer 0 pomeni narobe (false), neničelne vrednosti pa prav (true). Vrednost TRUE pomeni 1, FALSE pa 0.

```
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]
```

(2 bajta) - primeren za števila od 0 do 65535 (216–1), če je unsigned ali od -32768 do 32767

```
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]
```

(3 bajte) - zasede vrednosti od 0 do 16777215 (224–1), obratno niti ne bomo gledali, ker ni tako pomembno.

```
INT[(M)] [UNSIGNED] [ZEROFILL]
```

(4 bajte) - od 0 do 4294967295 (232–1). INTEGER je sinonim za INT.

```
BIGINT[(M)] [UNSIGNED] [ZEROFILL]
```

(8 bajtov) - razpon 0 do (264-1), se pa uporablja za ogromne vrednosti.

```
SERIAL
```

je alias za BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.

```
FLOAT[(M, D)] [UNSIGNED] [ZEROFILL]
```

(4 bajte) - realno število s plavajočo vejico, enojna natančnost. Npr. FLOAT(7,4) pomeni, da bodo števila prikazana v obliki -999.9999. MySQL izvede zaokroževanje pri shranjevanju podatkov, tako bi v našem primeru vnos števila 999.00009 zaokrožil na 999.0001.

```
DOUBLE(M, D) [UNSIGNED] [ZEROFILL]
```

(8 bajtov) - realno število s plavajočo vejico, dvojna natančnost. DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL] in REAL[(M,D)] [UNSIGNED] [ZEROFILL] sta sinonima za DOUBLE.

```
DECIMAL[(M[, D])] [UNSIGNED] [ZEROFILL]
```

(M+1 bajt ali M+2 bajta) - realno število, shranjeno kot niz, z določeno decimalno vejico. DEC[(M[,D])] [UNSIGNED] [ZEROFILL], NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL], FIXED[(M[,D])] [UNSIGNED] [ZEROFILL] so sinonimi za DECIMAL.

3.3.2 Časovni podatkovni tipi

Sedaj pa si poglejmo še tipe za shranjevanja časa.

```
DATE
```

Zasede 3 bajte v obliki YYYY-MM-DD: 2011-10-09 (Leto-Mesec-Dan).

```
TIME
```

Zasede 3 bajte v obliki HH:MM:SS: 19:20:21 (Ura-Minuta-Sekunda).

```
DATETIME
```

Zasede 8 bajtov v obliki YYYY-MM-DD HH:MM:SS: 2011-10-09 19:20:21 (Leto-Mesec-Dan Ura-Minuta-Sekunda).

```
TIMESTAMP
```

Zasede 4 bajte v obliki YYYYMMDDHHMMSS; velja do konca leta 2037.: 20111009192021 (brez ločnic zapisan datum in čas).

```
YEAR[(2|4)]
```

Zasede 1 ali 2 bajta v obliki YYYY ali YY: 2011 ali 11 (leto zapisano s štirimi števkami ima dovoljene vrednosti med 1901 in 2155 ter vrednost 0000 ali dvema števkama dovoljene vrednosti med 70 in 69 in predstavljajo leta med 1970 in 2069).

Primer:

```
SELECT CURDATE(), CURRENT_DATE;
```

Vrne:

```
CURDATE()  CURRENT_DATE
2011-11-14      2011-11-14
```

Primer:

```
SELECT CURTIME(), CURRENT_TIME;
```

Vrne:

```
CURTIME()  CURRENT_TIME
09:37:34   09:37:34
```

Primer:

```
SELECT CURRENT_TIMESTAMP, SYSDATE(), NOW();
```

Vrne:

```
CURRENT_TIMESTAMP     SYSDATE()        NOW()
2011-11-14 09:22:16   2011-11-14 09:22:16   2011-11-14 09:22:16
```

Zgornji primeri nam prikazuje uporabo funkcij, ki nam vrnejo trenutni datum in/ali čas. Kot vidimo, imamo več možnosti za dosego istega rezultata.

3.3.3 Znakovni podatkovni tipi

Naslednji, ki pridejo na vrsto, so tipi za shranjevanje nizov oziroma teksta.

```
CHAR(x), VARCHAR(x)
```

Sta podobna tipa podatkov za shranjevanje nizov znakov. Razlikujeta se v načinu shranjevanja in vračanja podatkov. Dolžina CHAR podatkovnega tipa je nespremenljiva in hrani toliko znakov, kot smo jih določili ob deklaraciji. Če nismo uporabili vseh znakov, se dopolni s presledki. Ko podatek tipa CHAR vrnemo, pa le-ta nima končnih presledkov. Tip VARCHAR pa hrani in vrača toliko znakov, kot smo jih uporabili. VARCHAR potrebuje še en ali dva bajta za informacijo o dolžini niza. Tipa zahtevata parameter x, ki pomeni maksimalno število znakov, ki jih lahko hranimo, lahko je od 0 do 255 za CHAR in od 0 do 65535 za VARCHAR.

Primer:

Vrednost	CHAR(4)	Poraba pomnilnika	VARCHAR(4)	Poraba pomnilnika
''	' '	4 bajte	''	1 bajt
'ab'	'ab '	4 bajte	'ab'	3 bajte
'abcč'	'abcč'	4 bajte	'abcč'	5 bajtov
'abcčdefg'	'abcč'	4 bajte	'abcč'	5 bajtov

Tabela 8: Razlika med podatkovnima tipoma CHAR in VARCHAR.

```
BINARY(x), VARBINARY(x)
```

Sta podobna tipoma CHAR in VARCHAR, le da vsebujeta dvojiške nize znakov namesto znakovnih. Podobnost je tudi v načinu shranjevanja in vračanja podatkov, le da namesto presledkov BINARY tip dopolnimo z "0x00" oz. "\0". Parameter x, ki pomeni maksimalno število bajtov in ne znakov kot pri CHAR in VARCHAR, ki jih lahko hranimo, lahko je od 0 do 255 za BINARY in od 0 do 65535 za VARBINARY.

Primer:

```
CREATE TABLE bin_test (c BINARY(3));

INSERT INTO bin_test
VALUES ('a');

SELECT HEX(c), c = 'a', c = 'a\0\0'
 FROM bin_test;
```

Vrne:

```
HEX(c) c = 'a' c = 'a\0\0'
610000 0 1
```

Zgornji primer potrjuje, kar smo napisali zgoraj. Potem ko smo v polje s podatkovnim tipom BINARY(3) vnesli 'a', se je to polje dopolnilo na 'a\0\0'.

```
TEXT, BLOB
```

TEXT in BLOB tipi so zelo podobni VARCHAR in VARBINARY tipoma, le da ne moreta imeti privzetih (DEFAULT) vrednosti. TEXT torej shranjuje znakovne nize, BLOB pa dvojiške vrednosti. BLOB pomeni [B]inary [L]arge [OB]jects, torej veliki dvojiški objekti. BLOB tipi so primerni za shranjevanje vseh vrst datotek.
Poznamo štiri vrste TEXT in BLOB tipov:
TINYTEXT in TINYBLOB lahko hranita do 28=256 znakov oz. bajtov. TEXT in BLOB do 216=65 536 znakov oz. bajtov, kar je 64 kilobajtov. MEDIUMTEXTin MEDIUMBLOB 224=16 777 216 znakov oz. bajtov, kar je 16 megabajtov. LONGTEXT in LONGBLOB pa 232=4 294 967 296 znakov oz. bajtov, kar znaša 4 gigabajte.

```
ENUM
```

ENUM (enumeration oz. oštevilčenje) uporabljamo, kadar želimo stolpcu prirediti eno izmed vnaprej določenih vrednosti (npr. velikost).

Primer:

```
CREATE TABLE enum_test
(
 velikost ENUM('majhno', 'srednje', 'veliko')
);
INSERT INTO enum_test (velikost)
VALUES ('majhno'), ('veliko');
```

Vsako oštevilčenje ima svoj indeks oz. mesto, za katerega velja, da se številčenje vrednosti začne s številko 1, prazna vrednost ('') ima številko 0, polje brez vrednosti (NULL) pa NULL. Indeks pri oštevilčenju nima nobene povezave z indeksi tabel. V našem primeru:

vrednost	**indeks**
NULL	NULL
''	0
'majhno'	1
'srednje'	2
'veliko'	3

Tabela 9: Vrednosti in indeksi v primeru ENUM.

```
SET
```

SET (množica) je znakovno polje, ki lahko vsebuje nič ali več elementov iz vnaprej določene množice vrednosti, ki jih ločimo z vejico.

Primer:

```
CREATE TABLE set_test ( korak SET('ena', 'dva') );

INSERT INTO set_test (korak) VALUES ('');
INSERT INTO set_test (korak) VALUES ('ena');
INSERT INTO set_test (korak) VALUES ('dva');
INSERT INTO set_test (korak) VALUES ('ena', 'dva');
```

Dan primer prikazuje tudi vse možne vrednosti polja korak.

3.4 Uvoz in izvoz podatkov

3.4.1 Uvoz podatkov (LOAD DATA)

Že v prvem poglavju smo videli, kako lahko uvozimo podatke s pomočjo grafičnega vmesnika phpMyAdmin. Tokrat bomo naredili korak naprej in se naučili uvažati podatke preko LOAD ukaza.

Sintaksa:

```
LOAD DATA INFILE ime_datoteke
INTO TABLE ime_tabele
[CHARACTER SET nabor_znakov]
[FIELDS TERMINATED BY 'ločilo polj']
[LINES TERMINATED BY 'ločilo vrstic']
[IGNORE število LINES];
```

Za ime datoteke moramo napisati celotno pot do datoteke. To lahko naredimo na dva načina:

```
'D:\\vaje\\podatki.txt'
```

oziroma:

```
'D:/vaje/podatki.txt'
```

Za nabor znakov bomo uporabljali utf8

Ločilo_polj in ločilo_vrstic bosta odvisna od datoteke, ki jo bomo uvozili. Običajno uporabljamo vejico, podpičje ali tabulator (\t) za ločilo polj in novo vrstico (\n) za ločilo vrstic.

IGNORE bomo uporabili, kadar bomo npr. imeli datoteko, ki bo vsebovala tudi imena stolpcev. V tem primeru bomo napisali IGNORE 1 LINES

Poizkusimo uvoziti podatke v tabelo države iz datoteke drzave.txt, ki smo jo shranili na disk D v mapo vaje:

```
LOAD DATA INFILE 'D:\\vaje\\drzave.txt'
INTO TABLE države
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

V primeru, da ne navedemo poti, se mora datoteka nahajti v mapi. D:\xampp\mysql\data\ime_podatkovne_zbirke. V našem primeru, ko uporabljamo podatkovno zbirko test, v mapi D:\xampp\mysql\data\test.

Primer:

```
LOAD DATA INFILE 'drzave.txt'
INTO TABLE države
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

3.4.2 Izvoz podatkov (SELECT ... INTO OUTFILE)

SELECT stavek smo že obravnavali, a se ga bomo na tem mestu še enkrat spomnili. V prejšnjem poglavju smo si ogledali uvoz podatkov s pomočjo stavka LOAD DATA in SELECT ... INTO OUTFILE je njegovo nasprotje, torej gre za izvoz podatkov.

Sintaksa:

```
SELECT *
FROM ime_tabele
INTO OUTFILE ime_datoteke
[CHARACTER SET nabor_znakov]
[FIELDS TERMINATED BY 'ločilo polj']
[LINES TERMINATED BY 'ločilo vrstic'];
```

Paziti moramo, da napišemo celo pot datoteke in da mapa, v katero zapisujemo datoteko, obstaja, sicer nam bo MySQL vrnil napako.

Primer:

```
SELECT *
FROM države
INTO OUTFILE 'D:\\vaje\\drzave-izvoz.txt'
CHARACTER SET 'utf8'
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n';
```

V primeru, da ne navedemo poti, nam bo datoteko shranil v mapo. `D:\xampp\mysql\data\ime_podatkovne_zbirke`. V našem primeru, ko uporabljamo podatkovno zbirko test, bo datoteka shranjena v mapi `D:\xampp\mysql\data\test`.

Primer:

```
SELECT *
FROM države
INTO OUTFILE 'drzave-izvoz.txt'
CHARACTER SET 'utf8'
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n';
```

3.5 Uporaba BLOB podatkovnih tipov (Shranjevanje datotek v MySQL)

Najprej ustvarimo tabelo, v katero bomo shranjevali datoteke. Uporabili bomo naslednja polja:

1. id: identifikacijsko samoštevilo
2. ime: ime datoteke
3. tip: tip datoteke
4. velikost: velikost datoteke
5. datoteka: datoteka

Stolpec datoteka bo podatkovnega tipa BLOB. BLOB pomeni velika dvojiška datoteka ([B]inary [L]arge [OB]ject), ki lahko hrani različne količine podatkov. MySQL ima štiri BLOB podatkovne tipa, ki so:

- TINYBLOB,
- BLOB,
- MEDIUMBLOB,
- LONGBLOB.

Ker je BLOB omejen na 64 kilobajtov, bomo uporabili MEDIUMBLOB, ki lahko hrani do 16 megabajtov velike datoteke.

```
CREATE TABLE shramba (
 id INT NOT NULL AUTO_INCREMENT,
 ime VARCHAR(30) NOT NULL,
 tip VARCHAR(30) NOT NULL,
 velikost VARCHAR(10) NOT NULL,
 datoteka MEDIUMBLOB NOT NULL,
 PRIMARY KEY(id)
);
```

Shranjevanje datoteke v MySQL obsega dva koraka. Najprej moramo shraniti datoteko na strežnik in jo nato vstaviti v MySQL. Za shranjevanje datoteke na strežnik bomo uporabili funkcijo load_file(), za vstavljanje v MySQL pa navadni INSERT stavek. Uporabili bomo datoteki zima.jpg in MySQL-QuickRef.pdf.

Primer:

```
INSERT INTO shramba (ime, velikost, tip, datoteka )
VALUES ('Zima', '104KB', 'jpg', load_file("D:/zima.jpg"));
```

```
INSERT INTO shramba (ime, velikost, tip, datoteka )
VALUES ('MySQL-QR', '33KB', 'pdf',
load_file("D:/MySQL_QuickRef.pdf"));
```

Če hočemo dobiti neke podatke iz tabele, uporabimo SELECT stavek in tako bo tudi z vračanjem datotek iz tabel.
Edina razlika je v tem, da določimo mesto in ime datoteke, v katero bomo shranili datoteko iz tabele.

Primer:

```
SELECT datoteka
  INTO DUMPFILE 'D:/zima2.jpg'
  FROM shramba
 WHERE ime='Zima';
```

```
SELECT datoteka
  INTO DUMPFILE 'D:/MySQL_QuickRef2.pdf'
  FROM shramba
```

```
WHERE ime='MySQL-QR';
```

3.6 Vaje – DDL in DML

Naslednji sklop vaj obsega upravljanje s podatkovnimi zbirkami, tabelami, uvoz in izvoz podatkov ter tudi poizvedovanje in ažuriranje podatkov. Obsega torej vso snov do tega mesta. Ker je bila naloga uporabljena za domačo nalogo oz. seminarsko nalogo, so podana originalna navodila skupaj s točkovanjem.

S spletne strani http://ljubljanskimaraton.si/sl/result/history ljubljanskega maratona uvozite rezultate dveh zaporednih tekov, kot je navedeno v tabeli spodaj. Lahko pa izberete svojo kombinacijo dveh tekov. Vaše tabele ustvarite tako, da bodo ustrezale podatkom. Za delo uporabite sistem za upravljanje podatkovnih baz MySQL.

1.	in	2.	maraton	ženske
2.	in	3.	polmaraton	moški
3.	in	4.	rekreativni tek	ženske
4.	in	5.	maraton	moški
5.	in	6.	polmaraton	ženske
6.	in	7.	rekreativni tek	moški
7.	in	8.	maraton	ženske
8.	in	9.	polmaraton	moški
9.	in	10.	rekreativni tek	ženske
10.	in	11.	maraton	moški
11.	in	12.	polmaraton	ženske
12.	in	13.	rekreativni tek	moški
13.	in	14.	maraton	ženske
14.	in	15.	polmaraton	moški

Tabela 10: Možne kombinacije dveh zaporednih tekov.

Naloge:

1. **Ustvarjanje tabel. [10 točk]**
   Ustvarite in uporabite podatkovno zbirko z imenom oblike »številka_prvega_teka-in-številka_drugega_teka-tip_teka-spol« npr. »1-in-2-maraton-moški«. S pomočjo ukaza CREATE TABLE ustvarite tabeli za vaša dva teka. Tabeli poimenujte na način »številka_teka-tip_teka-spol«. Če imate npr. 1. in 2. maraton za moške, ju poimenujte »1-maraton-moški« in »2-maraton-moški«. Uporabite smiselna imena stolpcev in podatkovne tipe. Kodo shranite v datoteko 1.txt.
2. **Shranjevanje podatkov s spleta. [10 točk]**
   V vašem brskalniku najdite ustrezen tek in z miško izberite celotno tabelo. Pritisnite Ctrl+C oz. kopiraj. Odprite beležnico in pritisnite Ctrl+V oz. prilepi. Shranite datoteko z imenom na enak način kot v prvi nalogi, le da ji dodate končnico .txt. Pazite na to, da datoteko shranite v utf-8 kodiranju in na to, da datoteke za uvoz ne smejo imeti šumnikov. Ponovite proceduro za drugi tek.
3. **Uvoz podatkov. [10 točk]**
   S pomočjo ukaza LOAD DATA uvozite podatke z obeh prej ustvarjenih datotek v

ustrezni tabeli. Podatki v tako ustvarjenih datotekah so ločni s tabulatorjem. Kodo shranite v datoteko 3.txt.

4. **Poizvedbe. [40 točk]**
   Rešite naslednje poizvedbe in rezultate izvozite v datoteke z imeni 4-1.txt, 4-2.txt … (Vsaka izvožena datoteka prinese 1 točko.) Kodo vseh ustrezno označenih poizvedb shranite v datoteko 4.txt.:
   1. Koliko je bilo vseh tekmovalcev na posameznem teku? [1+1 točka]
   2. Izpiši najboljših 10 tekmovalcev z obeh tekov! [2+1 točka]
   3. Izpiši rezultate vseh tekmovalcev z obeh tekov, ki imajo priimek, enak vašemu. [3+1 točka]
   4. Poiščite vse tekmovalce, ki so tekmovali v obeh tekih. [3+1 točka]
   5. Poiščite vse tekmovalce, ki so tekmovali v obeh tekih in so izboljšali rezultat. [4+1 točka]
   6. Kakšen je bil povprečni rezultat za oba teka? [3+1 točka]
   7. Koliko je bilo tekmovalcev po državah z obeh tekov? [4+1 točka]
   8. Izpišite vse skupine s po vsaj tremi tekmovalci, ki so imeli enake rezultate. [4+1 točka]
   9. Napišite poljubno smiselno poizvedbo, ki bo vsebovala agregacijski operator. [3+1 točka]
   10. Napišite poljubno smiselno poizvedbo, ki bo vsebovala LEFT JOIN. [3+1 točka]
5. **Ažuriranje tabel in podatkov. [20 točk]**
   Rešite še naslednje naloge. Kodo vseh ustrezno označenih nalog shranite v datoteko 5.txt.:
   1. Prvi vaši tabeli dodajte stolpec z imenom slika, ki naj ima tak tip, da boste vanj lahko shranili slikovno datoteko. [5 točk]
   2. Vstavite vrstico z vašimi podatki. Vsebuje naj ime, priimek, letnico rojstva. [5 točk]
   3. Popravite to vrstico tako, da ji dodate še sliko. [5 točk]
   4. Zbrišite vrstico tekmovalca, ki je dosegel mesto enako zaporedni številki ob vašem imenu v zgornji tabeli. [5 točk]
6. **Izvoz podatkov v datoteko. [10 točk]**
   S pomočjo ukaza SELECT ... INTO OUTFILE izvozite tabelo, ki ste jo spreminjali, v datoteko z imenom v obliki »številka_teka-tip_teka-spol-izvoz.txt«. Npr. »2-maraton-moški-izvoz.txt«. Kodo shranite v datoteko 6.txt.

Kriterij ocenjevanja: 0-49 točk - nzd (1), 50-59 točk - zd (2), 60-74 točk - db (3), 75-87 točk - pdb (4), 88-100 točk - odl (5).

Vse datoteke zapakirajte v arhivsko datoteko z imenom »Razred-Priimek-Ime.7z«. Lahko tudi v zip ali rar datoteko. Npr. »4.Ra-Novak-Janez.7z«. Datoteko oddate v spletno učilnico.

Rešitve na strani 129.

3.7 Indeksi (INDEX)

Indeksi pospešijo delo s podatkovno bazo. Običajno indekse dodamo tistim poljem, preko katerih pogosto iščemo podatke, in poljem, ki jih uporabljamo za povezavo z drugimi tabelami

(tuj ključ). SQL-2003 indeksov ne določa, zato so v domeni proizvajalcev posameznih SUPB. Z indeksi ne gre pretiravati, saj preveliko število indeksov upočasni delo s podatkovno bazo.

3.7.1 Kreiranje indeksov

Sintaksa:

```
CREATE INDEX ime_indeksa
  ON ime_tabele (stolpec1, stolpec2, ...);
```

Primer:

```
CREATE INDEX indeks_države
  ON države (št);
```

Primer:

```
CREATE INDEX indeks_rd_1
  ON rojstni_dnevi (id);
```

Primer:

```
CREATE INDEX indeks_rd_2
  ON rojstni_dnevi (ime, priimek);
```

3.7.2 Spreminjanje indeksov

MySQL ne pozna ukaza za spreminjanje indeksov. Tako moramo indeks najprej zbrisati in potem ponovno narediti.

3.7.3 Brisanje indeksov

Sintaksa:

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

Primer:

```
DROP INDEX indeks_države
  ON države;
```

Primer:

```
DROP INDEX indeks_rd_1
  ON rojstni_dnevi;
```

3.7.4 Podatki o indeksih

Včasih nas zanima, katere indekse sploh imamo definirane na določenih tabelah. To izvemo s pomočjo ukaza SHOW:

Sintaksa:

```
SHOW {INDEX | INDEXES | KEYS}
    {FROM | IN} ime_tabele
    [{FROM | IN} ime_podatkovne_zbirke];
```

Primer:

```
SHOW INDEX
 FROM države
 FROM test;
```

Lahko uporabimo tudi obliko **ime_podatkovne_zbirke.ime_tabele** kot alternativno obliko za **ime_tabele FROM ime_podatkovne_zbirke**.

Naslednji primer je torej ekvivalenten zgornjemu:

```
SHOW INDEX
 FROM test.države;
```

3.7.5 Vaje – Indeksi

1. Ustvarite indeks na tabeli države na poljih št. in država ter indeks v tabeli kratice_držav na polju država.
2. Popravite prvi indeks tako, da mu dodate polje regija.
3. Izpišite vse indekse na tabelah države in kratice_držav.

Rešitve na strani 134.

3.8 Pogledi (VIEW)

Pogled predstavlja logično tabelo, ki temelji na tabeli ali drugem pogledu. Pogled sam po sebi ne vsebuje nobenega podatka. Lahko bi rekli, da pogled predstavlja okno, skozi katerega uporabnik vidi podatke v eni ali več tabelah. V bazi je pogled shranjen kot SQL-stavek.

3.8.1 Razlogi za uporabo pogledov:

- omejitev dostopa do podatkov (s pogledom se prikažejo le izbrani stolpci in izbrane vrstice),

- skrivanje kompleksnosti podatkov (pogled se obnaša kot ena tabela, čeprav so podatki v njem iz več tabel),
- zagotavljanje podatkovne neodvisnosti in
- za predstavitev različnih pogledov na podatke (pogled predstavlja način za preimenovanje stolpcev, ne da bi dejansko spremenili definicijo osnovne tabele).

3.8.2 Ustvarjanje pogleda

Sintaksa:

```
CREATE [OR REPLACE] VIEW ime_pogleda [(nov_stolpec_1, ...,
nov_stolpec_n)]
  AS SELECT ...;
```

Primer:

```
CREATE VIEW rd_1
  AS SELECT * FROM rojstni_dnevi;
```

Primer:

```
CREATE VIEW birthday (id, name, surname, birthday, gender)
  AS SELECT * FROM rojstni_dnevi;
```

Pri tem je potrebno paziti, da ima SELECT stavek, ki sledi za besedico AS, ravno toliko stolpcev, kot jih je našteto zgoraj.

3.8.3 Spreminjanje pogleda

Sintaksa:

```
ALTER VIEW ime_pogleda [(nov_stolpec_1, ..., nov_stolpec_n)]
  AS SELECT ...;
```

Primer:

```
ALTER VIEW birthday (name, surname, gender)
  AS SELECT ime, priimek, rojstni_dan FROM rojstni_dnevi;
```

3.8.4 Brisanje pogleda

Sintaksa:

```
DROP VIEW ime_pogleda_1[, ime_pogleda_2, ...];
```

Primer:

```
DROP VIEW rd_1;
```

Podatki o pogledih

Če hočemo poiskati poglede iz določenih podatkovnih zbirk, uporabimo ukaz SHOW:

Sintaksa:

```
SHOW [FULL] TABLES [{FROM | IN} ime_podatkovne_zbirke]
    [LIKE 'vzorec' | WHERE izraz];
```

Verjetno ste pričakovali ukaz SHOW VIEWS, a se pogledi obnašajo zelo podobno kot tabele, tako da imamo samo en ukaz za oba bazna objekta.

Primer:

```
SHOW TABLES
 FROM test;
```

S tem dobimo spisek vseh tabel in pogledov. Če želimo poiskati samo poglede, uporabimo ključno besedico FULL. Ta nam v drugem stolpcu (Table_type) pove, ali je v prvem tabela "BASE TABLE" ali pogled "VIEW".

Primer:

```
SHOW FULL TABLES FROM test
WHERE Table_type = 'VIEW';
```

3.8.5 Vaje – Pogledi

1. Ustvarite pogled z imenom "oznake_držav", ki bo vseboval ime države, regijo, dvočrkovno (A2), tročrkovno (A3) in numerično (N3) kodo po standardu ISO 3166, kodo mednarodnega olimpijskega komiteja (MOK), mednarodno avtomobilsko oznako (DS) in dvočrkovno internetno domeno (IANA). Pri tem uporabite podatke iz tabel države in kratice_držav. Tabelo kratice držav lahko uvozite s pomočjo datoteke »kratice_drzav.sql«[5].
2. Popravite pogled tako, da bo vseboval tudi države, ki nimajo zapisov v tabeli "države".
3. Izpišite države, katerih regija ni določena v popravljenem pogledu.
4. Ustvarite pogled z imenom "internetne_oznake_držav", ki bo vseboval ime države in dvočrkovno internetno domeno (IANA).
5. Izpišite vse države, ki imajo enaki črki v internetni oznaki.
6. Izpišite vse poglede iz podatkovne zbirke test.

Rešitve na strani 134.

[5] http://sterle.tsckr.si/NUB/prenosi/kratice_drzav.sql (Vse datoteke, uporabljene v tem gradivu, lahko najdemo v http://sterle.tsckr.si/NUB/prenosi/.)

4 Shranjene procedure, funkcije, prožilci in dogodki

Shranjene procedure, funkcije, prožilci in dogodki so načini hranjenja ukazov na podatkovnem strežniku. Ti načini omogočajo različnim aplikacijam dostop do istih postopkov.

4.1 Primerjava shranjenih procedur, funkcij, prožilcev in dogodkov

MySQL omogoča štiri tipe shranjenih postopkov: procedure, funkcije, prožilci in dogodki.

Proceduro kličemo ročno s pomočjo ukaza CALL. Lahko ima nič ali več parametrov ter lahko vrne rezultate preko izhodnih spremenljivk. Tudi funkcijo kličemo ročno z nič ali več parametri, vrne pa skalarno vrednost. Prožilec se izvede, ko se spremeni zapis v določeni tabeli, dogodek pa ob določenem času in je lahko enkraten ali ponavljajoč.

4.2 Sestavljeni stavki

Ta del opisuje zgradbo sestavljenih stavkov, ki se uporabljajo kot telo shranjenih programov: procedur, funkcij, prožilcev in dogodkov. To so pravzaprav deli SQL kode, shranjene na strežniku, ki jo lahko kadarkoli uporabimo.

Sestavljeni stavek je blok, sestavljen iz drugih blokov, deklaracij spremenljivk, uravnalcev in kazalcev ter stavkov, ki omogočajo kontrolo nad potekom, kot so zanke in pogojni stavki.

4.2.1 Komentarji

MySQL pozna tri vrste komentarjev:

```
#vrstični komentar
```

```
-- vrstični komentar (za znakoma -- mora biti vsaj en presledek)
```

```
/*
večvrstični
komentar
*/
```

4.2.2 Bloki

Sintaksa:

```
[začetna_oznaka:] BEGIN
[stavki]
END [končna_oznaka]
```

Bloke uporabljamo za pisanje sestavljenih stavkov, ki se lahko pojavijo v funkcijah, prožilcih in dogodkih. Stavki se morajo končati s podpičjem (;). Podpičje pa je ločilo, ki izvede ukaz. Če

hočemo v programu uporabiti več stavkov, zaključenih s podpičjem, moramo spremeniti ločilo. To storimo z ukazom "DELIMITER". Seveda pa moramo vse skupaj vseeno izvesti, zato na koncu kode dodamo še ločilo za izvedbo, tj. tisto, ki smo ga določili na začetku. Na koncu zopet postavimo ločilo na podpičje. Stavke lahko izpustimo, dovoljen je torej tudi prazen blok.

Bloke lahko tudi označimo. Upoštevati moramo naslednja pravila:

- Začetna_oznaka se mora zaključiti z dvopičjem (:).
- Začetna oznaka ne potrebuje končne oznake. Če imamo končno oznako, mora biti enaka kot začetna.
- Končna_oznaka ne more biti brez začetne.
- Oznake na isti stopnji gnezdenja morajo biti različne.
- Oznake so lahko dolge največ 16 znakov.

Do oznake lahko dostopamo preko ITERATE in LEAVE stavkov. Naslednji primer prikazuje uporabo teh stavkov za nadaljevanje in prenehanje izvajanja zanke:

```
DELIMITER //

CREATE PROCEDURE ponavljaj(p1 INT)
BEGIN
  oznaka1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE oznaka1; END IF;
    LEAVE oznaka1;
  END LOOP oznaka1;
END;

//

DELIMITER ;
```

4.2.3 Lokalne spremenljivke

DECLARE se uporablja za deklariranje nekaj različnih lokalnih objektov. Mi ga bomo uporabili za deklariranje spremenljivk, uravnalcev (handlerjev) in kazalcev (kurzorjev).

DECLARE lahko uporabimo samo znotraj BEGIN ... END blokov in to samo na začetku bloka.

Sintaksa:

```
DECLARE ime_spremenljivke [, ime_spremenljivke] ... podatkovni_tip
[DEFAULT vrednost]
```

Če DEFAULT izpustimo, je privzeta vrednost NULL.

4.2.4 Uporabniško definirane spremenljivke

Vrednosti lahko shranjujemo tudi zunaj blokov. Take lahko uporabimo kasneje v drugih stavkih. Spremenljivkam te vrste bi drugje rekli globalne, a to v našem primeru ni čisto res. So pravzaprav odvisne od povezave, tj. spremenljivke, ki jih definira en klient, jih drugi ne morejo videti. Sprostijo se, ko se klient odjavi.

Spremenljive označujemo v obliki @ime_spremenljivke, kjer ime_spremenljivke lahko vsebuje alfanumerične znake, ".", "_", and "$". Lahko pa vsebuje tudi druge znake, če je označena kot niz znakov ('@moja-spremenljivka', "@moja-spremenljivka") ali indentifikator (`@moja-spremenljivka`).

Prvi način določanja vrednosti spremenljivkam je s pomočjo SET stavka.

```
SET @ime_spremenljivke1=izraz1 [, @ime_spremenljivke2=izraz2] ...
```

Kot operator prirejanja v stavku SET lahko uporabimo "=" ali ":=".

Spremenljivkam lahko določimo vrednosti tudi izven SET stavkov. V tem primeru moramo za operator prirejanja uporabiti ":=" saj je "=" mišljen kot primerjalni operator.

Primer:

```
SET @t1=1, @t2=2, @t3:=4;
SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;
SELECT @t4;
```

S stavkom "SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;" poleg izpisa izvedemo tudi prirejanje vrednosti spremenljivki "@t4".

4.3 Procedure

Procedura je na strežniku shranjen postopek oz. zaporedje ukazov, ki ima lahko nič ali več parametrov ter lahko vrne rezultate preko izhodnih spremenljivk.

Ustvarjanje procedure

Sintaksa:

```
CREATE PROCEDURE ime_procedure ([[ IN | OUT | INOUT ]parameter1
podatkovni_tip[,...]])
  [COMMENT 'opis procedure']
  BEGIN
    telo_procedure;
  END;
```

Kombinaciji ključnih besed CREATE PROCEDURE sledi ime procedure in v oklepaju našteti parametri skupaj s podatkovnim tipom. Pred parametrom lahko dodamo še tip parametra, ki je lahko IN (vhodni parameter), OUT (izhodni parameter) in INOUT (vhodno izhodni parameter). Privzet tip parametra je IN. To pomeni, da njegova sprememba znotraj procedure ne vpliva zunaj nje, kar pa ne velja za INOUT parameter. OUT parametru seveda določimo vrednost znotraj procedure.

Primer brez parametrov:

```
DELIMITER |

CREATE PROCEDURE stevilo_drzav()
BEGIN
  SELECT 'Število držav:', COUNT(*)
  FROM države;
END;

|

DELIMITER ;
```

Uporaba:

```
CALL stevilo_drzav();
```

Primer z dvema vhodnima in izhodnim parametrom:

```
DELIMITER $

CREATE PROCEDURE vsota (IN a INT, IN b INT, OUT c INT)
BEGIN
  SET c=a+b;
END;

$

DELIMITER ;
```

Uporaba:

```
SET @st=0;
SELECT @st;
CALL vsota(3, 4, @st);
SELECT @st;
```

Primer z vhodno-izhodnim parametrom:

```
DELIMITER $

CREATE PROCEDURE kvadrat (INOUT x INT)
BEGIN
  SET x=x*x;
END;

$

DELIMITER ;
```

Uporaba:

```
SET @st=4;
SELECT @st;
CALL kvadrat(@st);
SELECT @st;
```

4.4 Funkcije

Funkcija je, podobno kot procedura, postopek, ki ga kličemo ročno z nič ali večparametri, vrne pa skalarno vrednost.
Ustvarjanje funkcije

Sintaksa:

```
CREATE FUNCTION ime_funkcije ([parameter1[,...]])
  RETURNS podatkovni_tip
  [COMMENT 'opis funkcije']
  BEGIN
    telo_funkcije;
  RETURN skalarna_vrednost;
END;
```

Kombinaciji ključnih besed CREATE FUNCTION sledi ime funkcije in v oklepaju našteti parametri skupaj s podatkovnim tipom. Tip parametra je lahko samo IN (vhodni parameter) in ga zato izpustimo. Ker funkcija vrne neko vrednost, moramo določiti, kakšnega podatkovnega tipa bo le-ta. To storimo za ključno besedo RETURNS. Na koncu bloka (RETURN) moramo še določiti vrednost, ki naj jo funkcija vrne.

Primer:

```
DELIMITER |

CREATE FUNCTION stevilo_drzav_2()
```

```
RETURNS INT
BEGIN
  DECLARE st INT;
  SELECT COUNT(*)
  INTO st
  FROM države;
  RETURN st;
END;

|

DELIMITER ;
```

Uporaba:

```
SELECT stevilo_drzav_2();
```

4.5 Zanke in pogojni stavki

MySQL podpira zanke in pogojne stavke, ki kontrolirajo potek znotraj shranjenih procedur.

Zanke so LOOP, WHILE in REPEAT, znotraj teh pa uporabljamo še stavka ITERATE in LEAVE.
Pogojna stavka pa sta IF in CASE.

```
MySQL ne podpira FOR zanke.
```

4.5.1 IF stavek

IF je osnovni pogojni stavek.

Sintaksa:

```
IF pogoj1 THEN stavki
    [ELSEIF pogoj2 THEN stavki] ...
    [ELSE stavki]
END IF
```

Primer:

```
DELIMITER //

CREATE FUNCTION primerjaj(n INT, m INT)
  RETURNS VARCHAR(20)

  BEGIN
    DECLARE s VARCHAR(20);
```

```
    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);

    RETURN s;
  END //

DELIMITER ;
```

Uporaba:

```
SELECT primerjaj(3,7), primerjaj(3,3), primerjaj(3,-7);
```

Primer:

```
DELIMITER //

CREATE FUNCTION predznak(n INT)
  RETURNS CHAR(1)

  BEGIN
    IF n < 0 THEN RETURN '-';
    ELSEIF n = 0 THEN RETURN '0';
    ELSE RETURN '+';
    END IF;
  END //

DELIMITER ;
```

Uporaba:

```
SELECT predznak(-101), predznak(-0), predznak(101);
```

4.5.2 CASE stavek

CASE je napredni pogojni stavek.

Sintaksa:

```
CASE vrednost
    WHEN izraz1 THEN stavki
    [WHEN izraz1 THEN stavki] ...
    [ELSE stavki]
```

```
END CASE
```

ali

```
CASE
    WHEN pogoj1 THEN stavki
    [WHEN pogoj2 THEN stavki] ...
    [ELSE stavki]
END CASE
```

Primer:

```
DELIMITER //

CREATE FUNCTION dan_v_tednu(n INT)
  RETURNS VARCHAR(10)

  BEGIN
    CASE n
    WHEN '1' THEN RETURN 'ponedeljek';
    WHEN '2' THEN RETURN 'torek';
    WHEN '3' THEN RETURN 'sreda';
    WHEN '4' THEN RETURN 'četrtek';
    WHEN '5' THEN RETURN 'petek';
    WHEN '6' THEN RETURN 'sobota';
    WHEN '7' THEN RETURN 'nedelja';
    ELSE RETURN 'napaka';
    END CASE;
  END //

DELIMITER ;
```

Uporaba:

```
SELECT dan_v_tednu(3);
```

4.5.3 WHILE stavek

Stavki znotraj WHILE stavka se ponavljajo, dokler je pogoj izpolnjen.

WHILE zanko lahko tudi označimo.

Sintaksa:

```
[začetna_oznaka:] WHILE pogoj DO
    stavki
END WHILE [končna_oznaka]
```

Primer:

```
DELIMITER //

CREATE PROCEDURE test_while (IN stej_do INT)
BEGIN
    DECLARE st INT DEFAULT 0;

    WHILE st < stej_do DO
        SET st = st + 1;
        SELECT st;
    END WHILE;

END;

//

DELIMITER ;
```

Uporaba:

```
CALL test_while(10);
```

Primer:

```
DELIMITER $$

CREATE PROCEDURE ascii_znaki()
BEGIN
  DECLARE i INT DEFAULT 1;

  CREATE TEMPORARY TABLE ascii_tabela
    (ascii_koda INT, ascii_znak CHAR(1));

  WHILE (i<=128) DO
    INSERT INTO ascii_tabela VALUES(i, CHAR(i));
    SET i=i+1;
  END WHILE;

  SELECT * FROM ascii_tabela;

  DROP TABLE ascii_tabela;

END$$

DELIMITER ;
```

Uporaba:

```
CALL ascii_znaki();
```

4.5.4 REPEAT stavek

Stavki znotraj REPEAT stavka se ponavljajo, dokler se pogoj ne izpolni. Izvede se vedno vsaj enkrat.

REPEAT zanko lahko tudi označimo.

Sintaksa:

```
[začetna_oznaka:] REPEAT
    stavki
UNTIL pogoj
END REPEAT [končna_oznaka]
```

Primer:

```
DELIMITER //

CREATE PROCEDURE test_repeat (IN stej_do INT)
BEGIN
     DECLARE st INT default 0;

     povecaj: repeat
          SET st = st + 1;
          SELECT st;
          UNTIL st > stej_do
     END REPEAT povecaj;
END;

//

DELIMITER ;
```

Uporaba:

```
CALL test_repeat(10);
```

4.5.5 LOOP stavek

LOOP je preprosta zanka, ki omogoča ponavljanje enega ali več stavkov. Izvaja se toliko časa, dokler ne pridemo do LEAVE stavka. V funkcijah lahko izstopimo iz zanke tudi z RETURN stavkom. Če zanka nima stavka za izhod, dobimo neskončno zanko.

LOOP zanko lahko tudi označimo.

Sintaksa:

```
[začetna_oznaka:] LOOP
    stavki
END LOOP [končna_oznaka];
```

Primer:

```
DELIMITER //

CREATE PROCEDURE ponavljaj(p1 INT)
BEGIN
  oznaka1: LOOP
    SET p1 = p1 - 1;
    IF p1 >= 0 THEN
    BEGIN
      SELECT p1;
      ITERATE oznaka1;
       END;
    END IF;
    LEAVE oznaka1;
  END LOOP oznaka1;
END;

//

DELIMITER ;
```

Uporaba:

```
CALL ponavljaj(5);
```

4.6 Prožilci

Prožilec (trigger) je bazni objekt, ki je pripet na tabelo. Izgleda kot procedura ali funkcija. Pravzaprav je posebne vrste procedura. Glavna razlika med prožilci in procedurami je v tem, da se prožilec "sproži" natanko takrat (prej ali potem), ko se izvede INSERT, UPDATE ali DELETE stavek.

4.6.1 OLD in NEW

S pomočjo ključnih besed OLD in NEW pa lahko dostopamo do podatkov pred in po spremembi v tabeli.

Sintaksa:

```
CREATE TRIGGER ime_prožilca
[BEFORE|AFTER] [INSERT|UPDATE|DELETE] ON ime_tabele
FOR EACH ROW {stavek}
```

Primer:

Najprej ustvarimo tabelo dnevnik_sprememb, kamor bomo vnašali podatke o spremembah v tabeli.

```
CREATE TABLE dnevnik_sprememb(
  id INT AUTO_INCREMENT PRIMARY KEY,
  čas TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  čas_dogodka ENUM ('pred', 'po'),
  ukaz ENUM ('INSERT', 'UPDATE', 'DELETE'),
  uporabnik VARCHAR(50),
  opombe VARCHAR(50)
) DEFAULT CHARSET='utf8';
```

Ustvarimo vse možne prožilce {BEFORE | AFTER} {INSERT | UPDATE | DELETE}:

```
CREATE TRIGGER pred_vnosom
  BEFORE INSERT
  ON države
  FOR EACH ROW
  INSERT INTO dnevnik_sprememb (čas_dogodka, ukaz, uporabnik, opombe)
  VALUES ('pred', 'INSERT', current_user(), 'Uporaba prožilca
PRED_VNOSOM');
```

```
CREATE TRIGGER po_vnosu
  AFTER INSERT
  ON države
  FOR EACH ROW
  INSERT INTO dnevnik_sprememb (čas_dogodka, ukaz, uporabnik, opombe)
  VALUES ('po', 'INSERT', current_user(), CONCAT('Vnesli smo državo ',
NEW.država, '.'));
```

```
CREATE TRIGGER pred_spremembo
  BEFORE UPDATE
  ON države
  FOR EACH ROW
  INSERT INTO dnevnik_sprememb (čas_dogodka, ukaz, uporabnik, opombe)
  VALUES ('pred', 'UPDATE', current_user(), CONCAT(OLD.država, ' smo
preimenovali v ', NEW.država, '.'));
```

```
CREATE TRIGGER po_spremembi
  AFTER UPDATE
  ON države
  FOR EACH ROW
  INSERT INTO dnevnik_sprememb (čas_dogodka, ukaz, uporabnik, opombe)
  VALUES ('po', 'UPDATE', current_user(), 'Uporaba prožilca
PO_spremembi.');
```

```
CREATE TRIGGER pred_brisanjem
  BEFORE DELETE
  ON države
  FOR EACH ROW
  INSERT INTO dnevnik_sprememb (čas_dogodka, ukaz, uporabnik, opombe)
  VALUES ('pred', 'DELETE', current_user(), CONCAT(OLD.država, ' je
šla v koš.'));
```

```
CREATE TRIGGER po_brisanju
  AFTER DELETE
  ON države
  FOR EACH ROW
  INSERT INTO dnevnik_sprememb (čas_dogodka, ukaz, uporabnik, opombe)
  VALUES ('pred', 'DELETE', current_user(), CONCAT(OLD.država, ' je
šla v koš.'));
```

Uporaba:

```
SELECT 'Pred INSERT-om';

INSERT države (država, regija)
VALUES ('Gorenjska', 'Slovenija');

SELECT 'Po INSERT-u';
```

```
SELECT 'Pred UPDATE-om';

UPDATE države
SET država = 'Dolenjska'
WHERE država = 'Gorenjska';

SELECT 'Po UPDATE-u';
```

```
SELECT 'Pred DELETE-om';

DELETE FROM države
WHERE država='Dolenjska';

SELECT 'Po DELETE-u';
```

4.6.2 Spreminjanje prožilcev

MySQL ne pozna ukaza za spreminjanje prožilcev. Tako moramo prožilec najprej zbrisati in potem ponovno narediti.

4.6.3 Brisanje prožilcev

Sintaksa:

```
DROP TRIGGER ime_prožilca;
```

Primer:

```
DROP TRIGGER po_brisanju;
```

4.6.4 Podatki o prožilcih

Za pregled prožilcev uporabimo ukaz:

```
SHOW TRIGGERS FROM ime_podatkovne zbirke;
```

V našem primeru:

```
SHOW TRIGGERS FROM test;
```

4.7 Dogodki

Dogodek (event) je bazni objekt, ki je pripet na tabelo. Izgleda kot procedura ali funkcija.

Dogodki so podobno kot prožilci shranjena zaporedja ukazov, le da se izvedejo v točno določenem času. Lahko so enkratni ali ponavljajoči.

4.7.1 Zagon koledarja dogodkov v MySQL-u

Dogodki v MySQL-u se izvajajo s pomočjo posebnega koledarja dogodkov. Ta koledar je privzeto izklopljen. Kako pa ugotovimo, če koledar teče?

Z ukazom:

```
SHOW PROCESSLIST;
```

prikažemo delujoče procese in če teče koledar, bi se morali pokazati vsaj dve vrstici, kjer bi pri eni v stolpcu "user" pisalo "event_scheduler". Če dobimo samo eno vrstico, koledar ne teče in dogodki se ne bodo izvedli.

Koledar dogodkov zaženemo s tem, da spremenljivki event_scheduler določimo vrednost ON.

Sintaksa:

```
SET GLOBAL event_scheduler = ON;
```

4.7.2 Ustvarjanje dogodkov

Poglejmo, kako lahko naredimo dogodek.

Sintaksa:

```
CREATE EVENT ime_dogodka
ON SCHEDULE
AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'komentar']
DO BEGIN
    -- telo dogodka
END;
```

Najprej ustvarimo tabelo, v katero bomo zapisovali dnevnik izvajanja dogodkov.

```
CREATE TABLE dnevnik_dogodkov
       (ime_dogodka VARCHAR(20) NOT NULL,
        začetek_dogodka TIMESTAMP NOT NULL);
```

4.7.2.1 Enkratni dogodki - AT

Ustvarimo dogodek, ki bo v našo tabelo zapisal pozdrav čez pet minut.

```
CREATE EVENT lep_pozdrav
    ON SCHEDULE AT TIMESTAMP(NOW() + INTERVAL 5 MINUTE)
    DO INSERT INTO dnevnik_dogodkov VALUES ('Lep pozdrav!', NOW());
```

4.7.2.2 Ponavljajoči dogodki - EVERY

Ustvarimo dogodek, ki bo v našo tabelo petkrat zapisal pozdrav, prvič točno opoldne. Pozdravi naj si sledijo vsako minuto po enkrat.

```
CREATE EVENT petkrat_opoldne
ON SCHEDULE EVERY 1 MINUTE
   STARTS TIMESTAMP(CURDATE() + INTERVAL 1 DAY,'12:00:00')
   ENDS   TIMESTAMP(CURDATE() + INTERVAL 1 DAY,'12:00:00')
          + INTERVAL 4 MINUTE
DO INSERT INTO dnevnik_dogodkov
```

```
   VALUES ('Opoldanski pozdrav!', NOW());
```

4.7.3 Popravljanje dogodkov

Ker pa ne želimo čakati pet minut, da bi videli, če dogodek deluje, ga popravimo tako, da bo v našo tabelo zapisal pozdrav že čez dve minuti.

Sintaksa:

```
ALTER EVENT ime_dogodka
ON SCHEDULE
AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]
[ON COMPLETION [NOT] PRESERVE]
[RENAME TO novo_ime_dogodka]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'komentar']
DO BEGIN
    -- telo dogodka
END;
```

Spreminjanje dogodka spremeni eno ali več lastnosti, ne da bi bilo potrebno ponovno ustvariti dogodek. Sicer je sintaksa podobna CREATE stavku, le da je dodana možnost preimenovanja.

Spremenimo dogodku lep_pozdrav čas izvedbe.

```
ALTER EVENT lep_pozdrav
    ON SCHEDULE AT TIMESTAMP(NOW() + INTERVAL 1 MINUTE);
```

Dogodek lahko samo onemogočimo ...

```
ALTER EVENT lep_pozdrav
    DISABLE;
```

... in seveda tudi omogočimo.

```
ALTER EVENT lep_pozdrav
    ENABLE;
```

Preimenovanje dogodkov vidimo na naslednjem primeru.

```
ALTER EVENT lep_pozdrav
    RENAME TO najlepsi_pozdrav;
```

Brisanje dogodkov

Podobno kot pri ostalih baznih objektih izvajamo brisanje z ukazom DROP.

Sintaksa:

```
DROP EVENT ime_dogodka;
```

Primer:

```
DROP EVENT najlepsi_pozdrav;
```

4.7.4 Podatki o dogodkih

Za pregled dogodkov uporabimo ukaz:

```
SHOW EVENTS [{FROM | IN} ime_podatkovne_zbirke]
    [LIKE 'vzorec' | WHERE izraz];
```

V našem primeru:

```
SHOW EVENTS FROM test;
```

4.8 Vaje – Shranjene procedure, funkcije prožilci in dogodki

Pri naslednjem sklopu vaj bomo uporabili podatkovno zbirko podjetje, ki vsebuje štiri tabele. Podatkovno zbirko podjetje smo spoznali že v prejšnjem poglavju o arhiviranju. Celotna skripta za uvoz podatkovne zbirke skupaj s strukturo tabel in podatki si lahko snamete s spleta, vseeno pa zapišimo celotno kodo:

```
CREATE DATABASE podjetje;

USE podjetje;

CREATE TABLE usluzbenci
     (st_usluzbenca INTEGER NOT NULL,
     ime_usluzbenca VARCHAR(20) NOT NULL,
     priimek_usluzbenca VARCHAR(20) NOT NULL,
     st_oddelka CHAR(4) NULL
     ) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE oddelki
     (st_oddelka CHAR(4) NOT NULL,
     naziv_oddelka VARCHAR(25) NOT NULL,
     lokacija_oddleka VARCHAR(30) NULL
     ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
CREATE TABLE projekt
     (st_projekta VARCHAR(4) NOT NULL,
     ime_projekta VARCHAR(15) NOT NULL,
     sredstva FLOAT NULL
     ) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE delovno_mesto
     (st_usluzbenca INTEGER NOT NULL,
     st_projekta VARCHAR(4) NOT NULL,
     poklic VARCHAR (15) NULL,
     datum_zacetka DATE NULL
     ) ENGINE=MyISAM DEFAULT CHARSET=utf8;

INSERT INTO usluzbenci VALUES(25348, 'Matej', 'Kovač','d3');
INSERT INTO usluzbenci VALUES(10102, 'Ana', 'Novak','d3');
INSERT INTO usluzbenci VALUES(18316, 'Janez', 'Noč', 'd1');
INSERT INTO usluzbenci VALUES(29346, 'Jakob', 'Pretnar', 'd2');
INSERT INTO usluzbenci VALUES(9031, 'Eva', 'Prešern', 'd2');
INSERT INTO usluzbenci VALUES(2581, 'Mojca', 'Bernik', 'd2');
INSERT INTO usluzbenci VALUES(28559, 'Silvija', 'Komar', 'd1');
INSERT INTO oddelki VALUES ('d1', 'raziskovalni','Kranj');
INSERT INTO oddelki VALUES ('d2', 'računovodstvo', 'Jesenice');
INSERT INTO oddelki VALUES ('d3', 'marketing', 'Kranj');
INSERT INTO projekt VALUES ('p1', 'Triglav', 120000.00);
INSERT INTO projekt VALUES ('p2', 'Stol', 95000.00);
INSERT INTO projekt VALUES ('p3', 'Kepa', 186500.00);
INSERT INTO delovno_mesto VALUES (10102,'p1', 'analitik',
'2006.10.1');
INSERT INTO delovno_mesto VALUES (10102, 'p3', 'manager', '2008.1.1');
INSERT INTO delovno_mesto VALUES (25348, 'p2', 'uslužbenec',
'2007.2.15');
INSERT INTO delovno_mesto VALUES (18316, 'p2', NULL, '2007.6.1');
INSERT INTO delovno_mesto VALUES (29346, 'p2', NULL, '2006.12.15');
INSERT INTO delovno_mesto VALUES (2581, 'p3', 'analitik',
'2007.10.15');
INSERT INTO delovno_mesto VALUES (9031, 'p1', 'manager', '2007.4.15');
INSERT INTO delovno_mesto VALUES (28559, 'p1', NULL, '2007.8.1');
INSERT INTO delovno_mesto VALUES (28559, 'p2', 'uslužbenec',
'2008.2.1');
INSERT INTO delovno_mesto VALUES (9031, 'p3', 'uslužbenec',
'2006.11.15');
INSERT INTO delovno_mesto VALUES (29346, 'p1','uslužbenec',
'2007.1.4');
```

4.8.1 Procedure

1. Ustvarite proceduro z imenom vsota, ki sešteje dve realni števili. Z uporabo ustvarjene procedure izračunajte 123,45+12,345.
2. Ustvarite proceduro z imenom stevilo_usluzbencev, ki izpiše število zapisov tabele usluzbenci. Uporabite to proceduro.
3. Ustvarite proceduro z imenom vnos_usluzbencev, ki v tabelo usluzbenci zapiše toliko vrstic z vašim imenom in priimkom, kolikor je podano v parametru procedure.
4. Ustvarite proceduro, ki poveča sredstva v tabeli projekt za 10 odstotkov.
5. Ustvarite proceduro, ki podvoji vse zapise tabele projekt.

Rešitve na strani 137.

4.8.2 Funkcije

1. Ustvarite funkcijo z imenom potenca, ki izračuna x^y. Izračunajte 5^3.
2. Ustvarite funkcijo za približno računanje sinusa s pomočjo formule: sin(x) = x - x^3/ 3! + x^5/ 5! Z uporabo ustvarjene funkcije izračunajte sin(pi/6).
3. Ustvarite funkcijo za približno računanje arkus tangensa s pomočjo formule: arc tan(x) = x - x^3/ 3 + x^5/ 5. Z uporabo ustvarjene funkcije izračunajte arctan(1).
4. Ustvarite funkcijo z imenom besede, ki vrne število besed vstavljenega niza znakov. Z uporabo ustvarjene funkcije preštejte besede stavka »Danes je lep dan.«
5. Ustvarite funkcijo z imenom st_usluzbencev, ki izpiše število zapisov tabele usluzbenci. Uporabite to funkcijo.

Rešitve na strani 139.

4.8.3 Prožilci

1. Ustvarite prožilec z imenom pozitivna_sredstva, ki ob vnosu negativne vrednosti v polje sredstva tabele projekt vnese vrednost 0. Napišite SQL stavek, ki sproži ustvarjeni prožilec.
2. Ustvarite prožilec popravi_oddelek, ki ob spremembi vrednosti polja st_oddelka tabele oddelki popravi ustrezne zapise tudi v tabeli usluzbenci. Napišite SQL stavek, ki sproži ustvarjeni prožilec.
3. Ustvarite prožilec usluzbenci_v_arhiv, ki ob brisanju zapisa iz tabele usluzbenci le-tega zapiše v tabelo usluzbenci_arhiv, ki ima enako strukturo kot tabela usluzbenci. Napišite SQL stavek, ki sproži ustvarjeni prožilec.
4. Izpišite vse prožilce tabele uslužbenci.
5. Izbrišite prožilec usluzbenci_v_arhiv.

Rešitve na strani 141.

4.8.4 Dogodki

1. Ustvarite dogodek z imenom izprazni_tabelo_usluzbenci, ki bo čez eno uro izpraznil tabelo uslužbenci. Po izvedbi naj se dogodek ne zbriše.
2. Onemogočite in ponovno omogočite dogodek izprazni_tabelo_usluzbenci.
3. Preimenujte dogodek izprazni_tabelo_usluzbenci v izprazni_usluzbence.
4. Izbrišite dogodek izprazni_usluzbence.

Rešitve na strani 142.

5 Jezik za nadzor nad podatki (DCL – Data Control Language)

Jezik za nadzor nad podatki (DCL – Data Control Language) nam omogoča dodeljevanje in odvzemanje različnih pravic uporabnikom.

5.1 Upravljanje z uporabniškimi računi (USER)

Informacije o uporabniških računih so shranjene v tabelah podatkovne zbirke MySQL. O tej podatkovni zbirki in o njenih tabelah izvemo več o poglavju o administraciji.

5.1.1 Ustvarjanje uporabniških računov

Čeprav je ustvarjanje uporabniških imen del DDL jezika, bomo sintakso in primere obravnavali na tem mestu.

Na preprost način lahko ustvarimo uporabniški račun z geslom. Geslo lahko izpustimo, a to ni priporočljivo.

Sintaksa:

```
CREATE USER uporabniško_ime
    [IDENTIFIED BY [PASSWORD] 'geslo'];
```

Primer:

```
CREATE USER trgovec
    IDENTIFIED BY 'trgovec';
```

Če se želimo izogniti vnašanju golega gesla, uporabimo besedo PASSWORD. V tem primeru za geslo vnesemo prekodirano geslo, ki ga vrne funkcija PASSWORD().

Primer:

```
SELECT PASSWORD('trgovec');
```

Vrne:

```
+-------------------------------------------+
| PASSWORD('trgovec')                       |
+-------------------------------------------+
| *EB7ACFE29F860C5DCC7D67A49ECA9897A41FB64B |
+-------------------------------------------+
```

V tem primeru ustvarjanje uporabnika zgleda takole:

```
CREATE USER trgovec
 IDENTIFIED BY PASSWORD '*EB7ACFE29F860C5DCC7D67A49ECA9897A41FB64B ';
```

Ustvarili smo uporabniško ime trgovec z geslom trgovec.

5.1.2 Brisanje uporabniških računov

Še bolj preprosto je brisanje uporabniških računov. Zbrišemo jih lahko več na enkrat. Imena ločimo z vejico.

Sintaksa:

```
DROP USER uporabniško_ime_1 [, uporabniško_ime_2] ...;
```

Primer:

```
DROP USER trgovec;
```

5.2 Dodeljevanje pravic uporabnikom (GRANT)

Zdaj že znamo delati s podatkovnimi zbirkami in tabelami. Znamo ustvariti tudi uporabnike. Prišel je čas, da določimo, kdo lahko kaj počne: kdo lahko bere in kdo piše v tabele. To naredimo z dodeljevanje pravic uporabnikom.

Poenostavljena sintaksa stavka GRANT se glasi:

```
GRANT vrsta_pravice [(stolpci)]
ON {TABLE | FUNCTION | PROCEDURE}
   {*
  | *.*
  | ime_podatkovne_zbirke.*
  | ime_podatkovne_zbirke.ime_tabele
  | ime_tabele
  | ime_podatkovne_zbirke.ime_funkcije
  | ime_podatkovne_zbirke.ime_procedure}
TO uporabnik
[IDENTIFIED BY 'geslo']
[WITH GRANT OPTION];
```

Največkrat dodelimo vse privilegije (ALL PRIVILEGES) lokalnemu uporabniku za podatkovno zbirko, ki ima geslo za dostop do le-te (v našem primeru test). Ostale vrste pravic so opisane v spodnji tabeli.

5.2.1 Vrste pravic

Vrsta pravice	Opis
ALL [PRIVILEGES]	Dodeli vse pravice, razen GRANT OPTION.
ALTER	Omogoči uporabo ALTER TABLE.
ALTER ROUTINE	Omogoči spreminjanje in brisanje shranjenih procedur in funkcij.
CREATE	Omogoči ustvarjanje podatkovnih zbirk in tabel.
CREATE ROUTINE	Omogoči ustvarjanje shranjenih procedur in funkcij.
CREATE TEMPORARY TABLES	Omogoči uporabo CREATE TEMPORARY TABLE.
CREATE USER	Omogoči uporabo CREATE USER, DROP USER, RENAME USER in REVOKE ALL PRIVILEGES.
CREATE VIEW	Omogoči ustvarjanje in spreminjanje pogledov.
DELETE	Omogoči uporabo DELETE stavka.
DROP	Omogoči brisanje podatkovnih zbirk, tabel in pogledov.
EVENT	Omogoči uporabo dogodkov.
EXECUTE	Omogoči uporabniku izvajanje shranjenih procedur in funkcij.
FILE	Omogoči uporabniku, da od strežnika zahteva branje in shranjevanje datotek.
GRANT OPTION	Omogoči dodajanje in odvzemanje pravic drugim uporabnikom.
INDEX	Omogoči ustvarjanje in brisanje indeksov.
INSERT	Omogoči uporabo INSERT stavka.
LOCK TABLES	Omogoči uporabo LOCK TABLES stavka na tabelah s pravico izvajanja SELECT stavka.
PROCESS	Omogoči uporabniku vpogled v vse procese s pomočjo stavka SHOW PROCESSLIST.
SELECT	Omogoči uporabo SELECT stavka.
SHOW DATABASES	Omogoči uporabo SHOW DATABASES za prikaz podatkovnik zbirk.
SHOW VIEW	Omogoči uporabo SHOW CREATE VIEW stavka.
TRIGGER	Omogoči prožilce.
UPDATE	Omogoči uporabo UPDATE stavka.
USAGE	Sinonim za “brez pravic”.

Tabela 11: Vrste pravic.

Dodelimo vse pravice uporabniku z imenom uporabnik na lokalnem strežniku.

```
GRANT ALL PRIVILEGES
ON test.*
TO uporabnik@localhost
IDENTIFIED BY 'geslo';
```

Namesto za omejitev na objekte podatkovne zbirke, npr. tabele, se lahko odločimo za omejitev na jezik za manipuliranje (DML). Uporabimo naslednji primer:

```
GRANT SELECT,INSERT,UPDATE,DELETE
ON test.*
TO uporabnik@localhost
IDENTIFIED BY 'geslo';
```

Tako lahko uporabnik spreminja podatke samo z uporabo SELECT, INSERT, UPDATE ali DELETE stavkov. Če želimo dati pravice nelokalnim uporabnikom, lahko določimo IP-je, iz katerih se lahko dostopa do podatkovnih zbirk.

```
GRANT ALL PRIVILEGES
ON test.*
TO uporabnik@192.168.0.2
IDENTIFIED BY 'geslo';
```

Tako se lahko uporabnik z naslova '192.168.0.2' poveže na podatkovno zbirko. Če dopustimo uporabniku, da dostopa od koderkoli, uporabimo nadomestni znak '%'

```
GRANT ALL PRIVILEGES
ON test.*
TO uporabnik@'%'
IDENTIFIED BY 'geslo';
```

Lahko se celo odločimo, da uporabnik sploh ne potrebuje gesla, če se poveže z določenega mesta.

```
GRANT ALL PRIVILEGES
ON test.*
TO uporabnik@192.168.0.2
```

Vseeno je bolje uporabljati gesla. Poglejmo še pogoj WITH GRANT OPTION. Ta omogoča uporabniku dodeljevanje pravic drugim:

```
GRANT ALL PRIVILEGES
ON test.*
TO uporabnik@localhost
IDENTIFIED BY 'geslo'
WITH GRANT OPTION;
```

To omogoči uporabniku prijavo v podatkovno zbirko in dodeljevanje pravic uporabniku z imenom »prijatelj« za izvajanje SELECT, INSERT, UPDATE ali DELETE stavka.

```
GRANT SELECT,INSERT,UPDATE,DELETE
ON test.*
TO prijatelj@localhost
IDENTIFIED BY 'prijateljevo_geslo';
```

WITH GRANT OPTION ponavadi označuje lastništvo, čeprav s tem ne dobimo nobenih dodatnih pravic.

5.3 Odvzemanje pravic (REVOKE)

Odvzemanje pravic je skoraj identično dodeljevanju, le da zamenjamo REVOKE ... FROM z GRANT ... TO in izpustimo geslo ter ostale možnosti.

Za primer odvzemimo vse pravice uporabniku 'vohun'

```
REVOKE ALL PRIVILEGES
ON test.*
FROM vohun@localhost;
```

Lahko pa mu samo odvzamemo pravico izvajanja stavkov UPDATE, INSERT in DELETE, tako da ne more spreminjati podatkov.

```
REVOKE INSERT,UPDATE,DELETE
ON test.*
FROM vohun@localhost;
```

6 Jezik za nadzor nad transakcijami (TCL - Transaction Control Language)

Gre za jezik, ki nadzoruje potrjevanje podatkov.

6.1 Opredelitev transakcije

Transakcija predstavlja skupek ažuriranj, ki jih izvede transakcijski program. Z vidika SUPB predstavlja transakcija osnovno enoto spremembe, kar med drugim pomeni, da se transakcija mora izvesti v celoti ali pa sploh ne.
Dve pomembni nalogi SUPB pri izvajanju transakcij:

- zagotavljanje sočasnosti pri izvajanju transakcij,
- obnavljanje PB po transakcijskih in sistemskih nesrečah (razveljavljanje, ponavljanje transakcij ...).

6.2 Autocommit

MySQL samodejno potrjuje vse stavke, ki jih izvedemo. To določa sistemska spremenljivka @@autocommit, ki je privzeto nastavljena na 1.

```
SELECT @@autocommit;
```

Nam vrne vrednost 1, kar pomeni, da takoj po izvedbi INSERT, UPDATE oz. DELETE stavka spremembe vidijo vsi uporabniki.

To stanje lahko spremenimo z ukazom:

```
SET autocommit=0;
```

Primer:

Pokažimo, kako deluje autocommit.

Privzeto je spremenljivka autocommit nastavljena na 1. Preverimo:

```
SELECT @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
```

Ustvarimo preprosto tabelo, ki naj uporablja InnoDB mehanizem za shranjevanje podatkov, saj le-ta podpira transakcije. MyISAM mezanizem transakcij ne podpira.

```
CREATE TABLE Test(število INT) engine=InnoDB;
```

Vstavimo tri zapise v tabelo. Ti zapisi so takoj potrjeni.

```
INSERT INTO Test VALUES (1), (2), (3);
```

```
SELECT * FROM Test;
```

```
+---------+
| število |
+---------+
|       1 |
|       2 |
|       3 |
+---------+
```

Izklopimo samodejno potrjevanje s tem, da postavimo vrednost spremenljivke autocommit na 0:

```
SET autocommit=0;
SELECT @@autocommit;
```

```
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
```

Vstavimo še dve vrstici v tabelo:

```
INSERT INTO Test VALUES (4), (5);
```

Tako imamo v tabeli pet zapisov:

```
SELECT * FROM Test;
```

```
+---------+
| število |
+---------+
|       1 |
|       2 |
|       3 |
|       4 |
|       5 |
+---------+
```

Vendar podatki niso trajno zapisani v tabeli. Z ukazom ROLLBACK jih vzamemo nazaj.

```
ROLLBACK;
```

```
SELECT * FROM Test;
```

```
+---------+
| število |
+---------+
|       1 |
|       2 |
|       3 |
+---------+
```

Še enkrat vstavimo števili 4 in 5. Tokrat ju potrdimo z ukazom COMMIT. ROLLBACK stavek, ki sledi, tako nima nobenega učinka:

```
INSERT INTO Test VALUES (4), (5);
```

```
COMMIT;
```

```
ROLLBACK;
```

```
SELECT * FROM Test;
```

```
+---------+
| število |
+---------+
|       1 |
|       2 |
|       3 |
|       4 |
|       5 |
+---------+
```

6.3 Ukazi za delo s transakcijami

Z omogočenim samopotrjevanjem, tj. vrednost spremenljivke autocommit je 1, je vsak SQL stavek, ki kaj spreminja, svoja transakcija. Če želimo začeti svojo transakcijo, to storimo z ukazom START TRANSACTION. Ta transakcija je kasneje potrjena s COMMIT stavkom ali ovržena s stavkom ROLLBACK. Znotraj transakcije imamo lahko več stavkov, ki so vsi potrjeni ali ovrženi kot ena enota.

Primer:

Uporabili bomo isto tabelo kot prej. Najprej jo izpraznimo:

```
TRUNCATE Test;
```

Spodnja koda ustvari transakcijo in vnese štiri zapise v tabelo, a podatki še niso potrjeni.

```
START TRANSACTION;
```

```
INSERT INTO Test VALUES (1), (2);
```

```
INSERT INTO Test VALUES (3), (4);
```

Iz trenutne povezave podatke seveda vidimo:

```
SELECT * FROM Test;
```

```
+-----+
| Num |
+-----+
|   1 |
|   2 |
|   3 |
|   4 |
+-----+
```

Vendar preko neke druge povezave vidimo tabelo Test prazno. Povežimo se še enkrat:

```
mysql –u root –p
```

```
SELECT * FROM test.Test;
```

Nam vrne:

```
Empty set
```

To je druga povezava do podatkovne zbirke. Iz te povezave podatki še niso vidni.

Potrditev na prvi povezavi zagotovi, da so podatki vidni z obeh povezav.

```
COMMIT;
```

Začnimo s še eno transakcijo. Tokrat bomo podatke zavrteli nazaj. Vidimo, da vstavljeni štirje zapisi po ukazu ROLLBACK ne obstajajo več.

```
START TRANSACTION;

INSERT INTO Test VALUES (5), (6);
INSERT INTO Test VALUES (7), (8);

ROLLBACK;

SELECT * FROM Test;
```

```
+---------+
| število |
+---------+
|       1 |
|       2 |
|       3 |
|       4 |
+---------+
```

6.4 Mesta vrnitve (SAVEPOINT)

Do sedaj smo si pogledali transakcije, ki smo jih lahko potrdili oz. ovrgli na enem samem mestu. S pomočjo stavka SAVEPOINT se lahko vrnemo na vsako mesto, kjer smo določili mesto vrnitve.

Primer:

Še enkrat uporabimo našo tabelo Test. Za začetek jo spraznimo.

```
TRUNCATE Test;

START TRANSACTION;

INSERT INTO Test VALUES (1), (2);

SAVEPOINT mesto_vrnitve_1;

INSERT INTO Test VALUES (3), (4);

ROLLBACK TO SAVEPOINT mesto_vrnitve_1;

INSERT INTO Test VALUES (5), (6);

COMMIT;

SELECT * FROM Test;
```

Zadnja poizvedba nam vrne naslednji rezultat:

```
+---------+
| število |
+---------+
|       1 |
|       2 |
|       5 |
|       6 |
+---------+
```

6.5 Ukazi, ki samodejno potrdijo transakcijo

Nekateri ukazi sami sprožijo potrditev odprte transakcije. To so predvsem DDL ukazi. V bistvu ti ukazi najprej potrdijo transakcijo in se potem izvedejo. Spodaj imamo naštete te ukaze.

ALTER TABLE	Spremeni strukturo tabele.
CREATE INDEX	Ustvari indeks na tabeli.
DROP DATABASE	Odstrani podatkovno zbirko iz MySQLovega strežnika.
DROP INDEX	Zbriše indeks, pripet na tabelo.
DROP TABLE	Zbriše tabelo s podatkovne zbirke.
LOCK TABLES	Prepreči sočasni dostop do tabele.
RENAME TABLES	Preimenuje tabelo.
SET AUTOCOMMIT=1	Vklopi spremenljivko autocommit.
START TRANSACTION	Začne transakcijo.
TRUNCATE TABLE	Izprazni tabelo.
UNLOCK TABLES	Odklene zaklenjene tabele.

Tabela 12: Ukazi, ki samodejno potrdijo transakcijo.

6.6 Vaje – Transakcije

1. Ustvarite tabelo imena s stolpcema id, ki je samoštevilo in primarni ključ in ime znakovnega tipa. Tabela naj podpira šumnike.
2. Vstavite imeni Maks in Eva ter izpišite vsebino tabele.
3. Začnite transakcijo.
4. Posodobite ime v Nace, kjer je id enak 1.
5. Ustvarite mesto vrnitve.
6. Posodobite ime v Mica, kjer je id enak 2 ter izpišite vsebino tabele.
7. Zavrzite spremembe do ustvarjenega mesta vrnitve.
8. Izpišite vsebino tabele in zaključite transakcijo.
9. Kateri dve imeni sta na koncu v tabeli?

Rešitve na strani 135.

7 Arhiviranje in restavracija podatkovne baze

Arhiviranje in restavracija podatkovne baze sta dve zelo pomembni dejanji. O pomembnosti podatkov na tem mestu ne bi zgubljali časa, pojdimo k dejanjem.

7.1 Arhiviranje podatkovne baze

Arhiviranje podatkovne baze bomo izvedli s programom mysqldump, ki se nahaja v mapi X:\xampp\mysql\bin, če x zamenjamo z imenom diska, na katerem imamo nameščen xampp. Program mysqldump je zelo učinkovito orodje za arhiviranje MySQLovih podatkovnih zbirk. Ustvari datoteko oblike *.sql, ki vsebuje DROP TABLE, CREATE TABLE in INSERT INTO stavke, s katero natančno opiše izbrane podatkovne zbirke.

Sintaksa:

```
mysqldump -u [uporabniško_ime] -p[geslo] [ime_podatkovne_zbirke] >
[arhivska_datoteka.sql]
```

Primer:

Ustvarimo si preprosto podatkovno zbirko z eno tabelo, z enim stolpcem in tremi zapisi.

```
CREATE DATABASE arhiv;
USE arhiv;
```

```
CREATE TABLE tabela (stolpec INT);
```

```
INSERT INTO tabela VALUES (1), (10), (100);
```

Arhivirajmo našo podatkovno zbirko:

```
mysqldump -u root -proot arhiv > arhiv.sql
```

To, da se geslo, ki je v našem primeru root, drži stikala –p, ni napaka, ampak tako mora biti. Datoteko arhiv.sql je arhivski program ustvaril v isti mapi, kjer se nahaja program. Lahko pa napišemo celotno pot, če želimo arhivsko datoteko shraniti drugam.

Primer:

```
mysqldump -u root -proot arhiv > d:\arhiv.sql
```

Poglejmo še, kaj vsebuje ustvarjena datoteka.

```
-- MySQL dump 10.13  Distrib 5.5.16, for Win32 (x86)
--
-- Host: localhost    Database: arhiv
```

```
-- ------------------------------------------------------
-- Server version    5.5.16

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `tabela`
--

DROP TABLE IF EXISTS `tabela`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `tabela` (
  `stolpec` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `tabela`
--

LOCK TABLES `tabela` WRITE;
/*!40000 ALTER TABLE `tabela` DISABLE KEYS */;
INSERT INTO `tabela` VALUES (1),(10),(100);
/*!40000 ALTER TABLE `tabela` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

```
-- Dump completed on 2012-07-18 21:52:48
```

Vse kode seveda ne poznamo in je tudi ne bomo študirali. Poznamo pa kodo, ki je v odebeljenem tisku. To so ravno stavki, ki smo jih omenili v uvodu in se ujemajo z našimi podatki.

mysqldump poleg –u za uporabniško ime in –p za geslo pozna še nekaj dodatnih stikal. Mi jih bomo omenili samo nekaj.

`--databases db1 db2 …` uporabimo, ko želimo arhivirati več podatkovnih zbirk naenkrat oz. kadar želimo, da mysqldump ustvari tudi podatkovno zbirko.
`--all-databases` arhivira vse podatkovne zbirke.
`--events` arhivira dogodke, privzeto `--skip-events`, ki dogodkov ne arhivira.
`--routines` arhivira shranjene funkcije in procedure, privzeto `--skip-routines`, ki tega ne stori.
`--triggers` ob arhiviranju shrani tudi prožilce, kar je tudi privzeta možnost, če želimo prožilce spustiti, uporabimo stikalo `--skip-triggers`.
`--no-data` ne shrani podatkov.
`--no-create-info` ne shrani strukture tabel. Zadnji dve možnosti uporabimo, če želimo posebej shraniti strukturo tabele in posebej same podatke.

Primer:

```
mysqldump -u root -proot --no-data arhiv > d:\arhiv1.sql
mysqldump -u root -proot --no-create-info arhiv > d:\arhiv2.sql
```

V datoteki arhiv1.sql je samo CREATE stavek brez podatkov, v datoteki arhiv2.sql pa samo INSERT stavek brez strukture tabele.

Lahko arhiviramo samo določene tabele izbranih podatkovnih zbirk. To storimo tako, da za imenom podatkovne zbirke naštejemo želene tabele.

Sintaksa:

```
mysqldump -u [uporabniško_ime] -p[geslo] [ime_podatkovne_zbirke]
[tabela1 tabela2 …] > [arhivska_datoteka.sql]
```

Primer:

```
mysqldump -u root -proot podjetje oddelki > d:\arhiv.sql
```

Zgornji primer nam arhivira samo tabelo oddelki podatkovne zbirke podjetje. Za uvoz podatkovne zbirke podjetje uporabite datoteko »podjetje.sql«[6].

7.2 Restavracija podatkovne baze

[6] http://sterle.tsckr.si/NUB/prenosi/podjetje.sql (Vse datoteke, uporabljene v tem gradivu, lahko najdemo v http://sterle.tsckr.si/NUB/prenosi/.)

Restavracijo podatkovne zbirke izvedemo zelo preprosto s pomočjo arhivske datoteke.

Sintaksa:

```
mysql -u [uporabniško_ime] -p[geslo] [ime_podatkovne_zbirke] <
[arhivska_datoteka.sql]
```

Primer:

```
mysql -u root -proot arhiv < d:\arhiv.sql
```

Seveda v primeru, ko arhiviramo celotno podatkovno zbirko skupaj s CREATE DATABASE stavkom, restavracija prav tako ustvari podatkovno zbirko.

7.3 Arhiviranje in restavracija s pomočjo aplikacije phpMyAdmin

Z aplikacijo phpMyAdmin smo se že srečali, tako da uporaba ne bi smela biti problematična.

7.3.1 Arhiviranje s phpMyAdmin-om.

Odpremo phpMyAdmin.
Izberemo podatkovno zbirko s klikom na njeno ime na seznamu na levem delu okna.
Kliknemo povezavo (gumb) Izvozi. Pokaže se nam novo okno z napisom Izvažanje tabel iz zbirke podatkov "izbrana_podatkovna_zbirka".
Prikažeta se nam dva načina izvoza:

- Hitro - prikaži kar najmanj možnosti.
- Po meri - prikaži vse mogoče možnosti.

Sledi še spustni seznam, ki določi obliko.
Lahko pustimo privzete izbore in kliknemo Izvedi.
Prikaže se nam pogovorno okno, ki nas povpraša po mestu hranjenja datoteke, in ob potrditvi shranimo arhivsko datoteko.

7.3.2 Restavracija s phpMyAdmin-om.

Odpremo phpMyAdmin.
Ustvarimo podatkovno zbirko, v katero bomo uvozili strukturo in podatke, oz. izberemo že obstoječo podatkovno zbirko.
Kliknemo povezavo Uvozi. Prikaže se nam novo okno z napisom Uvažanje v zbirko podatkov "izbrana_podatkovna_zbirka".
Izberemo datoteko za uvoz, jo prenesemo na strežnik in s klikom na izvedi poženemo zaporedje ukazov za uvoz podatkov.

7.4 Vaje – Arhiviranje in restavracija podatkovnih zbirk

1. Naredite kopijo poljubne podatkovne zbirke.
2. Arhivirajte podatkovno zbirko test skupaj s procedurami, funkcijami in dogodki, prožilce izpustite.
3. Iz podatkovne zbirke podjetje arhivirajte tabeli usluzbenci in oddelki. V eni izvozni datoteki naj bo struktura tabel, v drugi pa podatki.

Rešitve na strani 135.

8 Izdelava aktivne spletne strani

Podatkovne zbirke, ki so del aktivnih spletnih strani, smo že dodobra spoznali. Na tem mestu imamo več možnosti, kako izdelati aktivno spletno stran. S pomočjo skriptnega jezika PHP in HTML-ja ter povezavo z MySQL-om bi lahko ustvarili dinamično spletno stran, a je PHP vsebina drugega predmeta.

Mi bomo spletno stran ustvarili s pomočjo Wordpress-a, uporabili pa bomo tudi nekaj znanja MySQL-a.

8.1 WordPress

WordPress je moderna semantična platforma za objavljanje s poudarkom na estetiki, spletnih standardih in uporabnosti. WordPress je prosta in neprecenljiva programska oprema.

Jedro je zgrajeno s pomočjo več sto prostovoljcev iz skupnosti in če želite več, je na voljo na tisoče vtičnikov in tem, ki bodo spremenile vašo spletno mesto, v skoraj karkoli si lahko prestavljate. Prek 25 milijonov ljudi si je izbralo WordPress za njihovo mesto na spletu – pridružite se nam.

Ste pripravljeni?

Tako je napisano na slovenski strani enega najbolj priljubljenih sistemov za upravljanje z vsebinami oz. krajše CMS-jem (angl. Content Management System). Je verjetno tudi najpreprostejši za uporabo, kar je verjetno eden glavnih razlogov za popularnost. Drugi razlog je seveda cena, saj je zastonj.

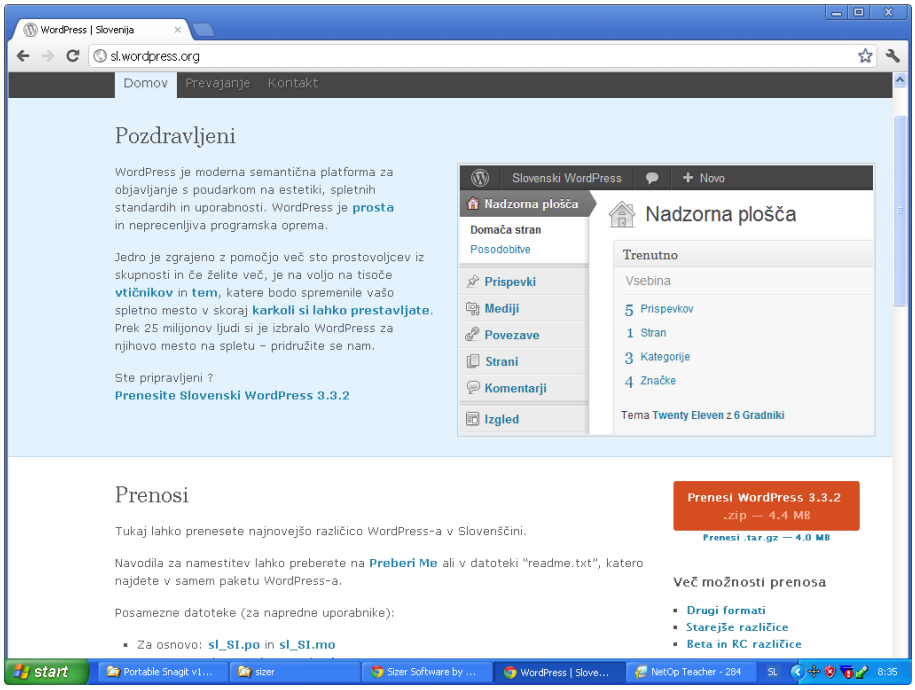
Mi bomo WordPress namestili na lokalnem strežniku, ki ga že dobro poznamo. Prehod na pravi strežnik pa ne bi smel predstavljati težav vsakemu, ki mu bo to uspelo na lokalnem strežniku.

Ne bomo se poglabljali v oblikovanje in vsebino, to bo vsak uporabnik storil sam. Poglobili se bomo samo v namestitev.

8.1.1 Namestitev Wordpress-a na lokalnem strežniku

1. Prenos Wordpress-a

S spletne strani http://sl.wordpress.org/ prenesete zadnjo različico Wordpress-a.



Slika 17: Slovenska spletna stran Wordpress-a.

To storite s klikom na gumb:

Slika 18: Gumb za prenos Wordpress-a.

2. Namestitev Wordpress-a

Arhivsko datoteko razširimo v podmapo htdocs mape xampp.

Poženemo XAMPP nadzorno ploščo in zaženemo Apache in MySQL. V naslovno vrstico izbranega brskalnika vnesemo localhost (odpremo phpMyAdmin ali HeidiSQL).

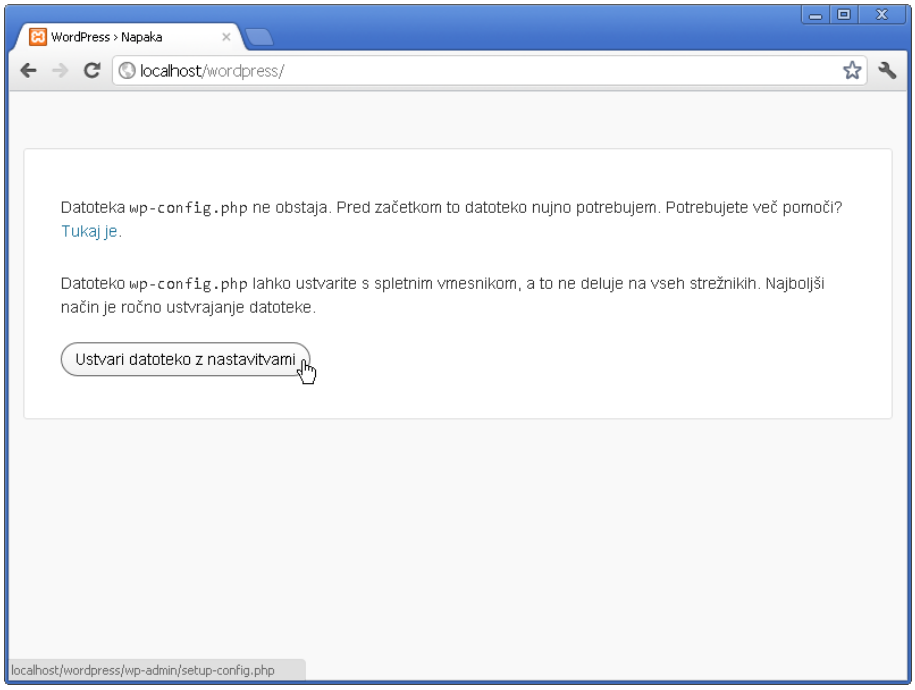
Ustvarimo podatkovno zbirko wordpress, uporabnika wp z enakim geslom in na koncu temu uporabniku damo vse pravice na lokalnem strežniku.

```
CREATE DATABASE wordpress;

CREATE USER wp
IDENTIFIED BY 'wp';
```

```
GRANT ALL PRIVILEGES
ON wordpress.*
TO wp@localhost
IDENTIFIED BY 'wp';
```

V naslovno vrstico izbranega brskalnika vnesemo localhost/wordpress in kliknemo na gumb »Ustvari datoteko z nastavitvami«.



Slika 19: Ustvarjanje namestitvene datoteke.

Pojavi se okno, ki zahteva podatkovno zbirko in uporabnika, kar smo storili že prej.

Slika 20: Nekateri podatki, ki jih potrebujemo za namestitev Wordpress-a.

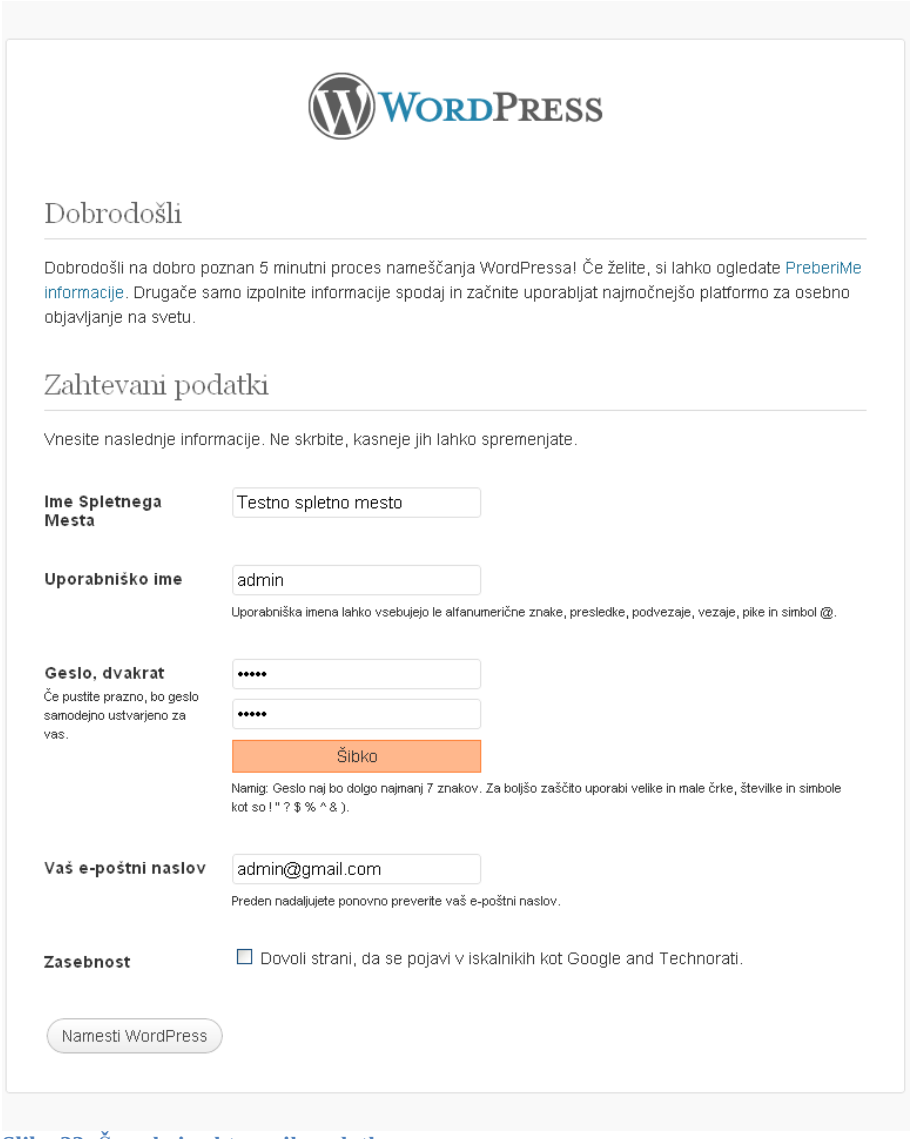
Klik na »Let's go!« nas pripelje do zaslona, ki zahteva podatke o podatkovni zbirki in uporabniku, ki smo ju ustvarili prej.

Slika 21: Vnos podatkovne zbirke in uporabniških podatkov.

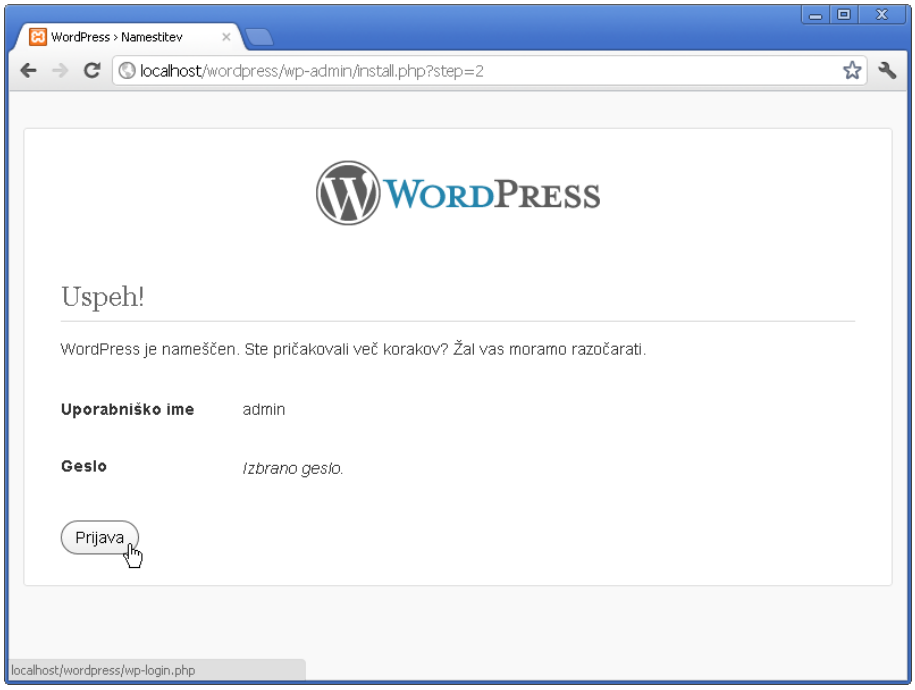
Gumb »Submit« in že smo skoraj na koncu.

Slika 22: Zagon namestitve.

Klik na »Run the install« in že smo pred zadnjim korakom. Vnesemo podatke in kliknemo na »Namesti WordPress«.



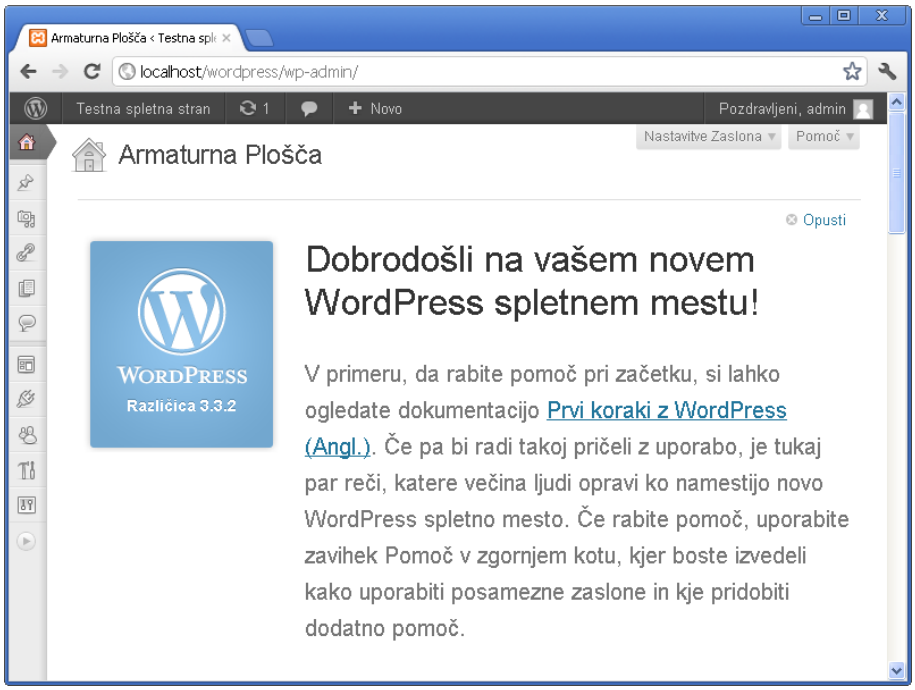
Slika 23: Še nekaj zahtevanih podatkov.



Slika 24: Potrditev uspešne namestitve.

3. Prijava v nadzorno ploščo

V nadzorno ploščo se prijavimo z uporabniškim imenom in geslom, ki smo si ga izbrali ob namestitvi. To storimo na naslovu http://localhost/wordpress/wp-login.php. Po pritisku na gumb »prijavi« se nam pokaže nadzorna ali armaturna plošča.

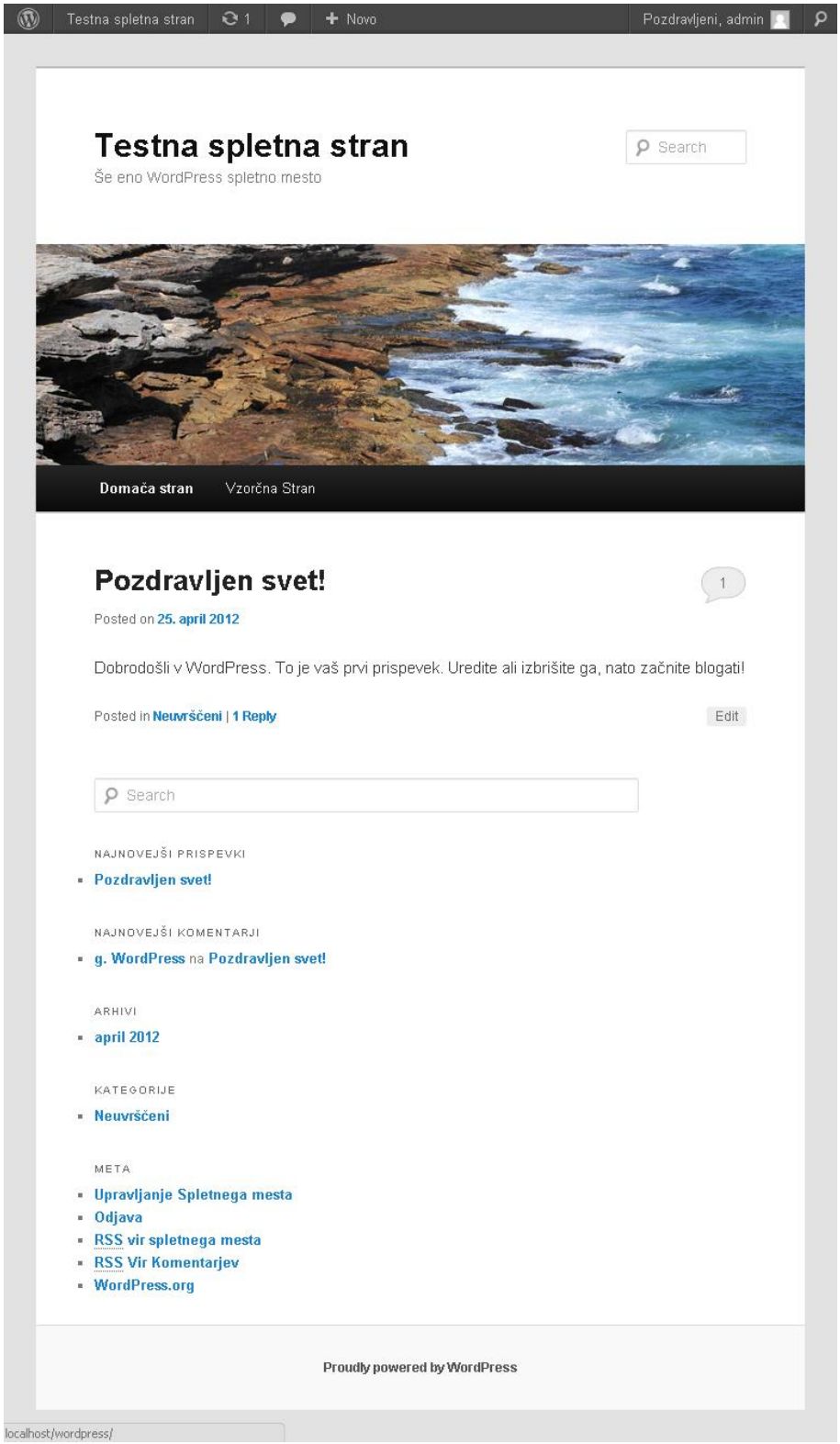


Slika 25: Nadzorna ali armaturna plošča.

Če hočemo obiskati našo spletno stran, se postavimo na ime naše spletne strani, v našem primeru »Testna spletna stran« in kliknemo na »Obišči Spletno mesto« .

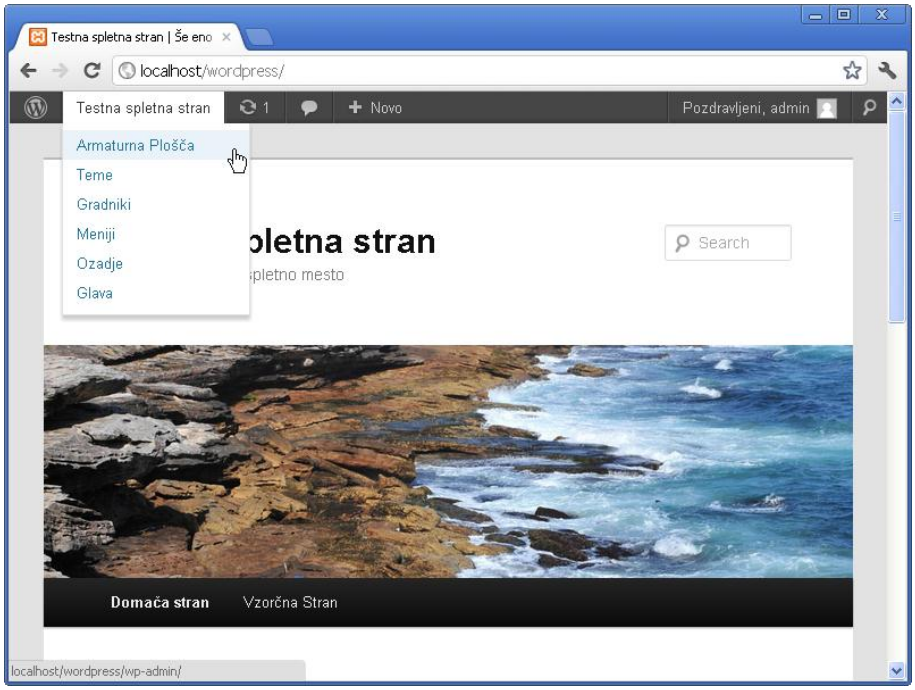
Prikaže se nam čisto prava delujoča spletna stran na lokalnem strežniku.

Do istega rezultata pridemo, če v naslovno vrstico brskalnika vpišemo http://localhost/wordpress/. Na tem naslovu se torej nahaja naša spletna stran.



Slika 26: Prikaz testne spletne strani.

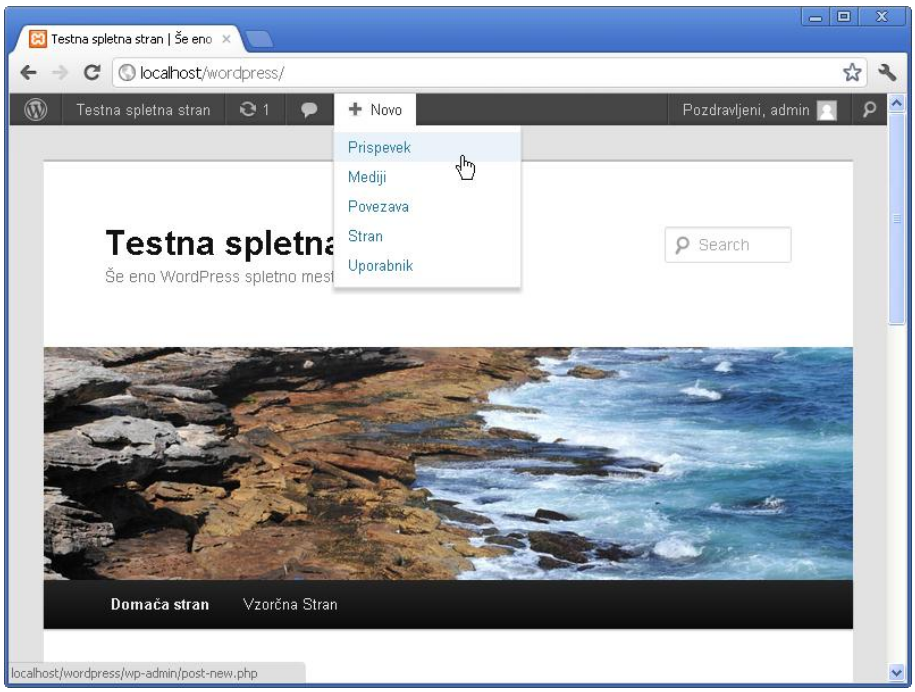
Če želimo nazaj na armaturno ploščo, se zopet postavimo na gumb »Testna spletna stran« in kliknemo na »Armaturna plošča«.



Slika 27: Nazaj na armaturno ploščo.

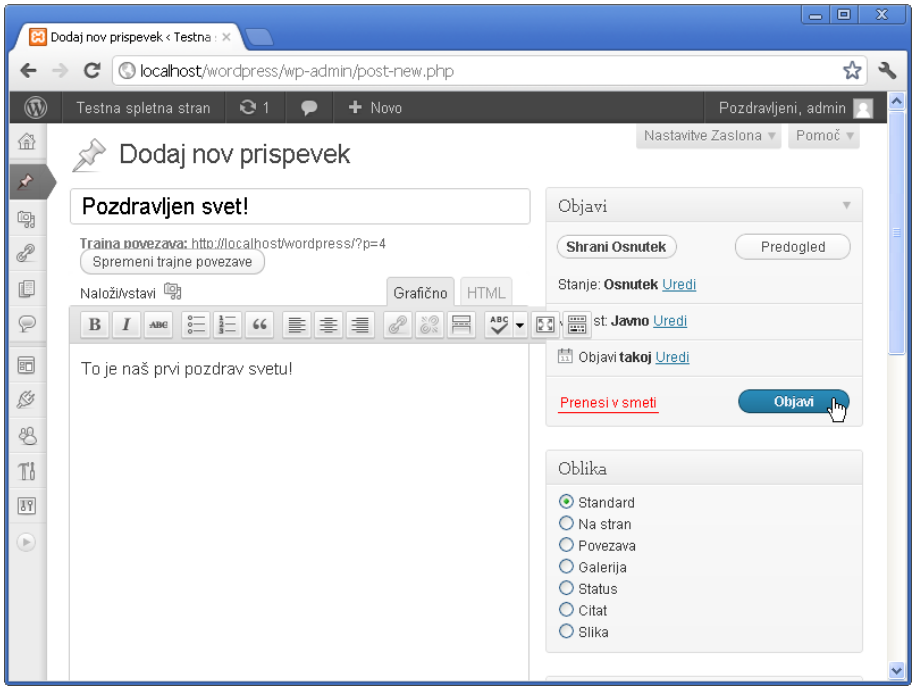
4. Izdelava spletne strani

Vsebino dodamo s postavitvijo na gumb »+ Novo« in klikom na »Prispevek«.



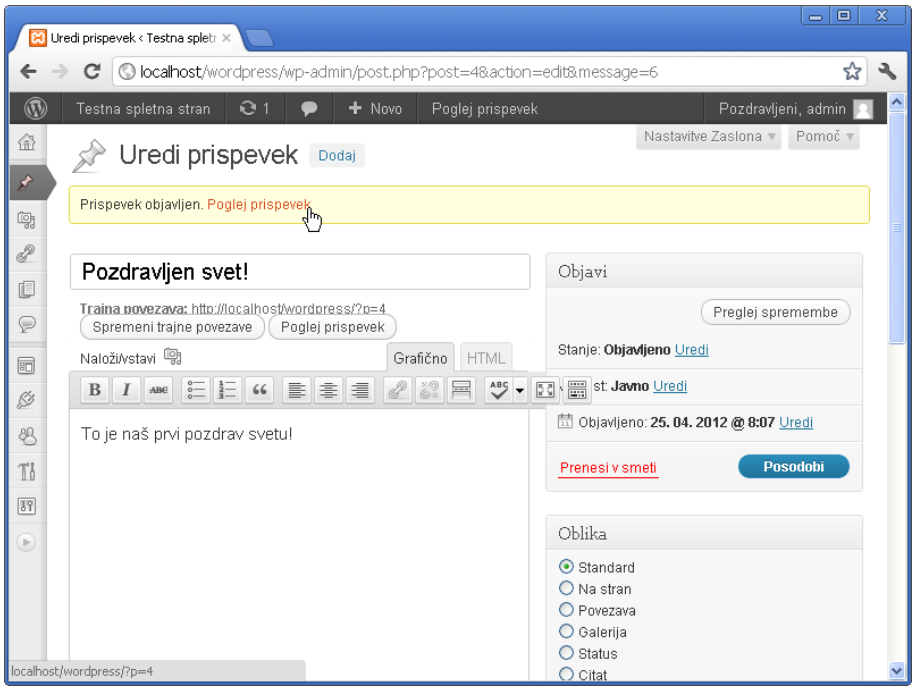
Slika 28: Dodajanje prispevka.

Pojavi se nam urejevalnik prispevkov:



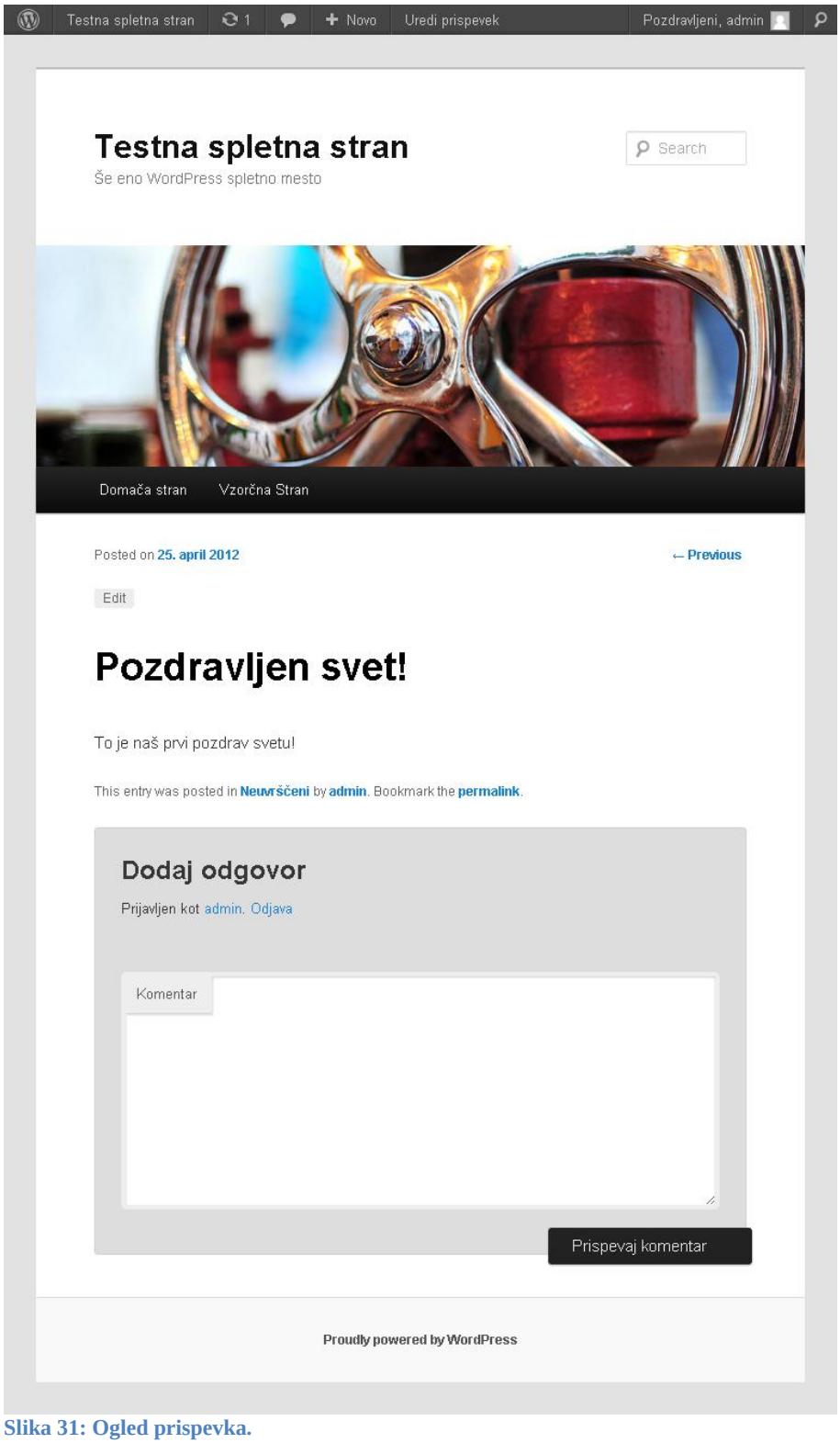
Slika 29: Objava prispevka.

Vnesemo naslov »Pozdravljen svet!« in kliknemo na gumb »Objavi«. S tem smo že objavili svoj prvi prispevek.



Slika 30: Potrditev objave prispevka.

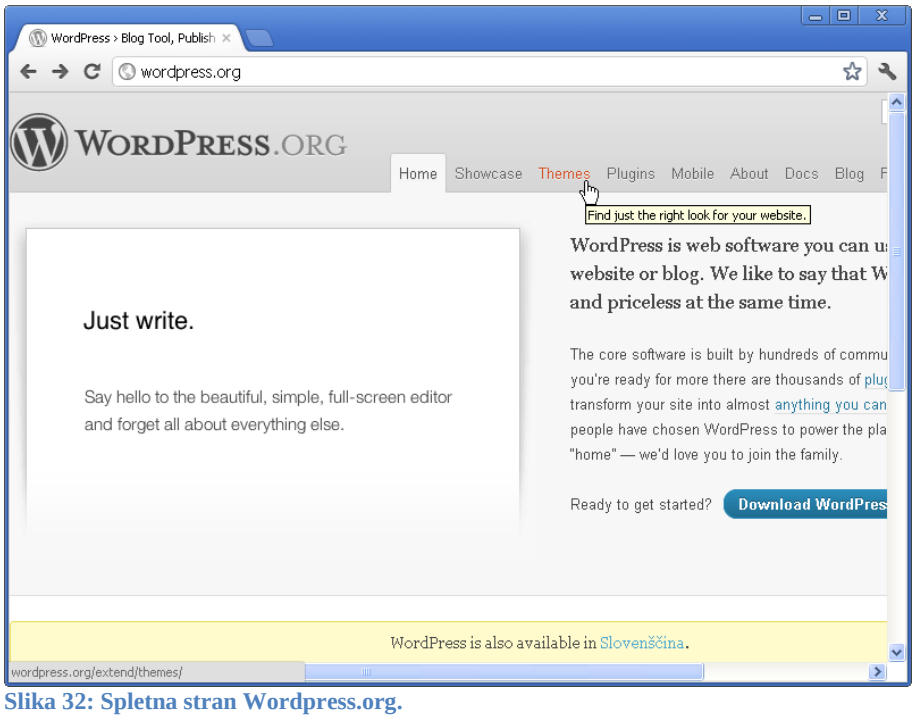
S klikom na »Poglej prispevek« si naše delo lahko tudi ogledamo.



Slika 31: Ogled prispevka.

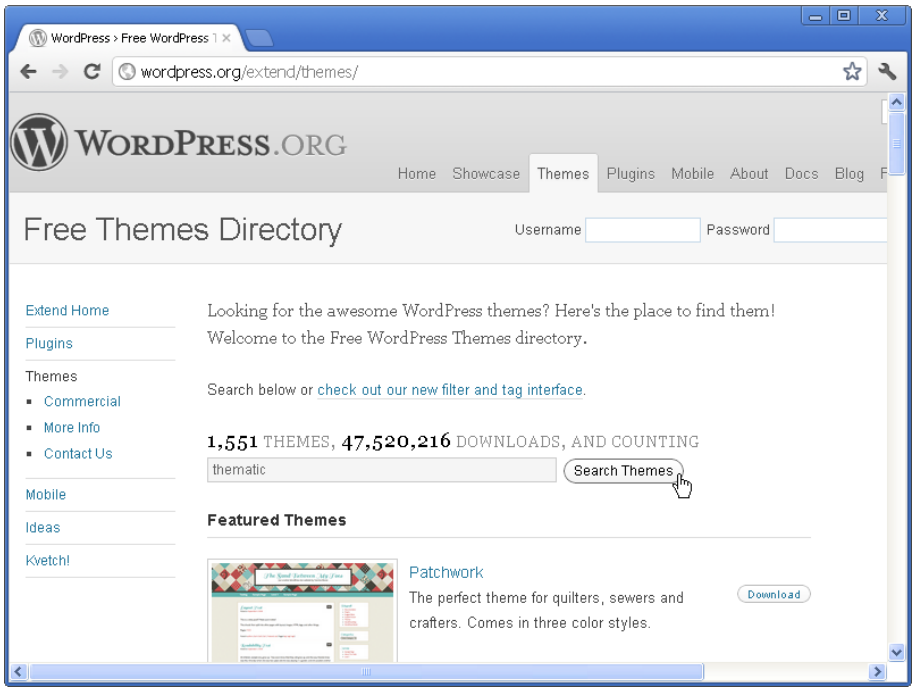
5. Namestitev teme

Na spletni strani http://wordpress.org/ kliknemo na zavihek »Themes«.



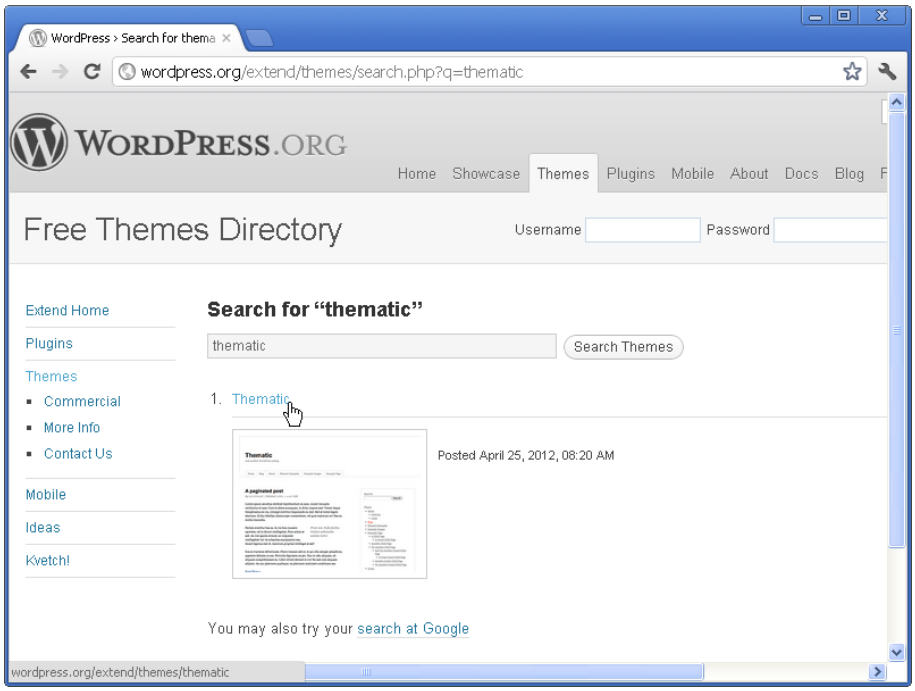
Slika 32: Spletna stran Wordpress.org.

V iskalnem polju poleg gumba »Search themes« vtipkamo »Thematic« in kliknemo omenjeni gumb.



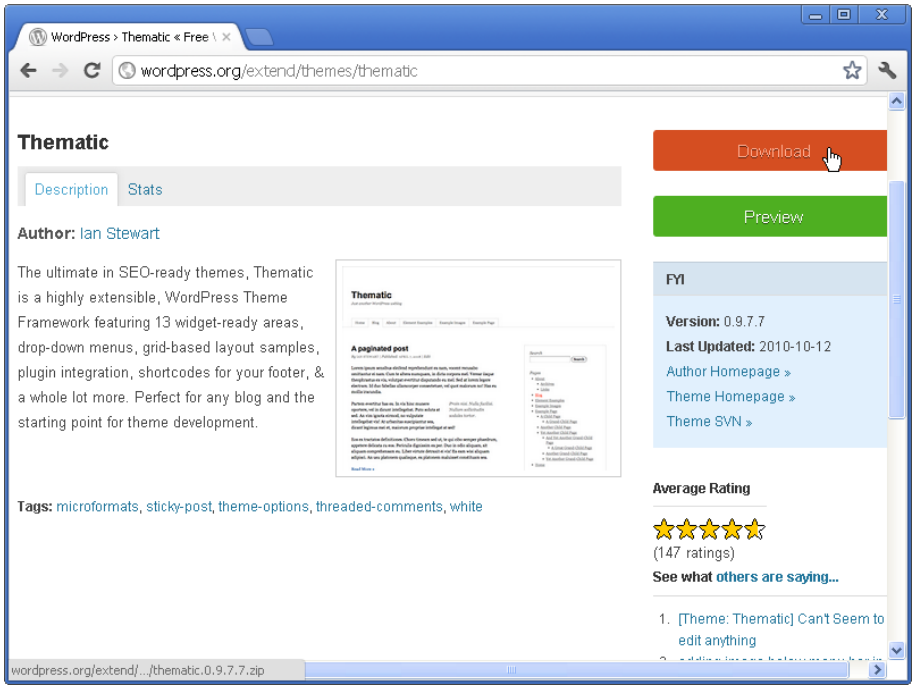
Slika 33: Iskanje tem.

Po končanem iskanju kliknemo na povezavo »Thematic«.



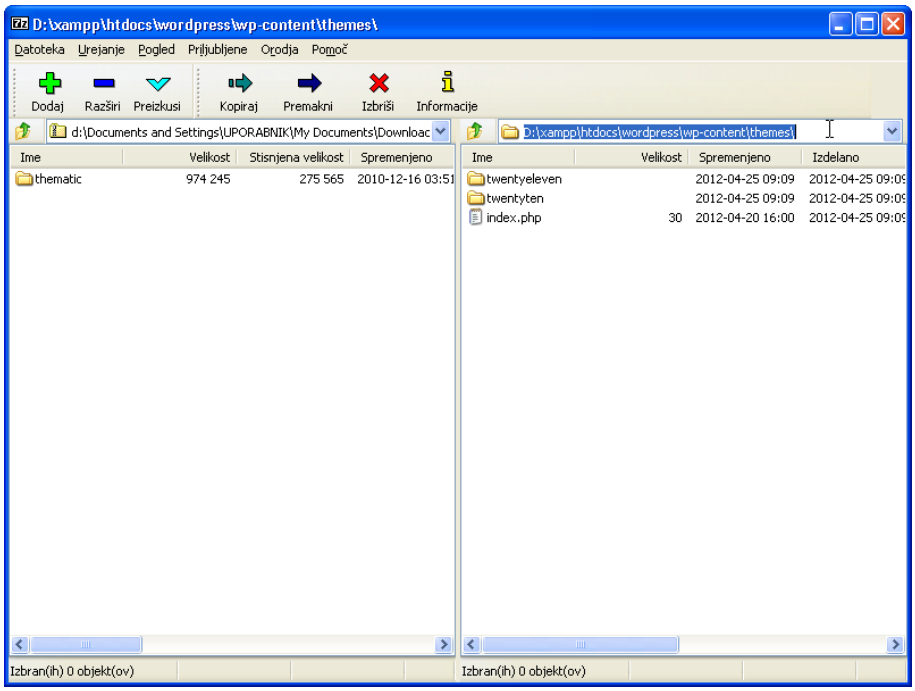
Slika 34: Izbor teme "Thematic".

In še na gumb »Download«, da prenesemo arhivsko datoteko, ki vsebuje vse datoteke izbrane teme.



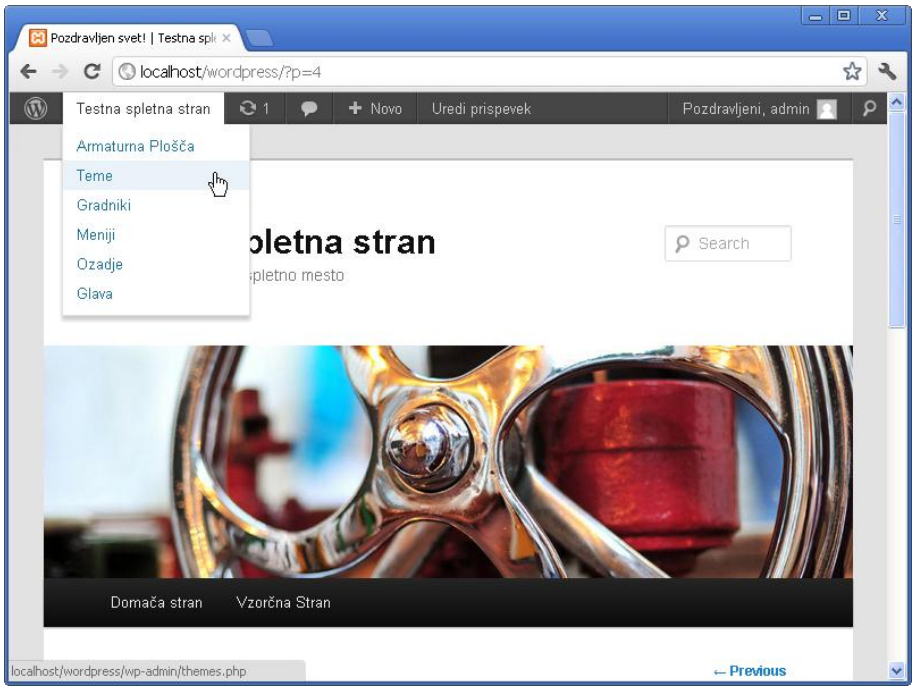
Slika 35: Prenos teme "Thematic".

Po prenosu odpremo datoteko in razpakiramo vsebino v mapo `X:\xampp\htdocs\wordpress\wp-content\themes`, kjer je X ime vašega diska.



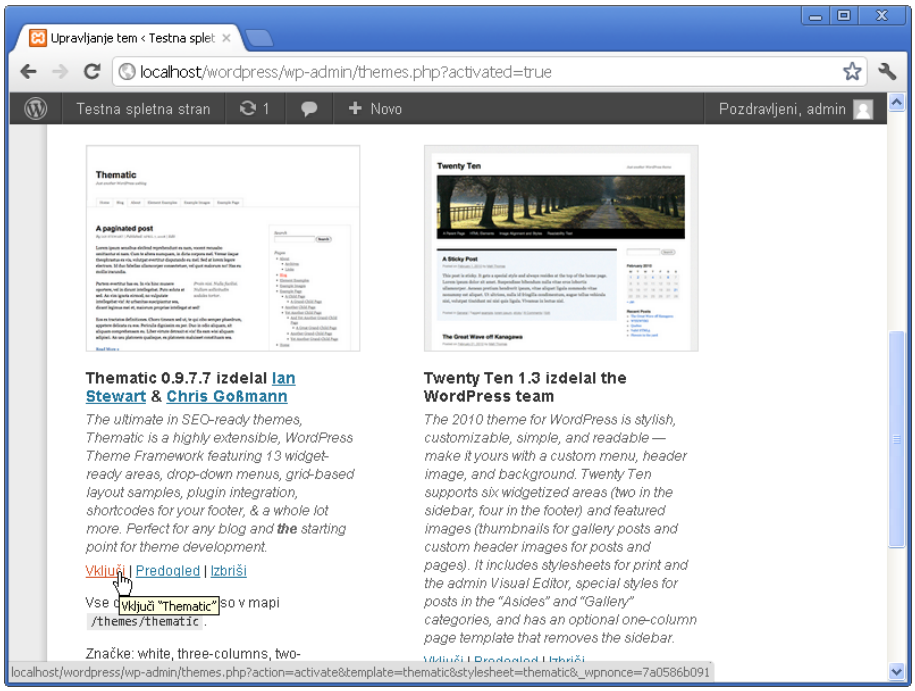
Slika 36: Okno aplikacije 7-zip za stiskanje datotek.

V WordPressu se postavimo na gumb »Testna spletna stran« in kliknemo na gumb »Teme«.



Slika 37: Povezava do tem.

Poiščemo nameščeno temo in kliknemo povezavo »Vključi«.



Slika 38: Vključitev teme.

Tako smo na zelo preprost način ustvarili spletno stran, ji dodali vsebino in spremenili izgled. Tudi dodajanje drugih gradnikov je tako preprosto kot nameščanje nove teme. Za izdelavo lastne teme pa je potrebno nekoliko več truda.

9 Rešitve vaj

9.1 Rešitve vaj s strani 23 (Vaje - SELECT)

1. Izpišite vse podatke iz tabele države.

```
SELECT *
  FROM države;
```

2. Izpišite vse podatke za Slovenijo.

```
SELECT *
  FROM države
 WHERE država='Slovenija';
```

3. Izpišite vse podatke za Slovenijo in Avstrijo.

```
SELECT *
  FROM države
 WHERE država='Slovenija'
    OR država='Avstrija';
```

-- ali

```
SELECT *
  FROM države
 WHERE država IN ('Slovenija', 'Avstrija');
```

4. Izpišite vse podatke za Slovenijo, Nizozemsko, Belgijo, Norveško in Španijo.

```
SELECT *
  FROM države
 WHERE država IN ('Slovenija', 'Nizozemska', 'Belgija', 'Norveška',
'Španija');
```

5. Izpišite imena vseh držav, ki so velike med 10.000 in 30.000 km2. Spisek uredite po abecednem vrstnem redu naraščajoče.

```
SELECT država
  FROM države
 WHERE površina BETWEEN 10000 AND 30000;
```

6. Izpišite imena vseh evropskih držav v bazi. Spisek uredite po abecednem vrstnem redu padajoče.

```
  SELECT država
    FROM države
   WHERE regija='Evropa'
ORDER BY država DESC;
```

7. Izpišite vse podatke o državah, katerih ime se prične s črko H.

```
SELECT *
  FROM države
 WHERE država LIKE 'H%';
```

8. Izpišite vse podatke o državah, ki imajo v regiji besedo Amerika. (Južna Amerika, Severne Amerika, Srednja Amerika).

```
SELECT *
  FROM države
 WHERE regija LIKE '%Amerika%';
```

9. Prikažite ime države ter število prebivalcev na km^2 površine države. Spisek uredite enkrat padajoče in enkrat naraščajoče po gostoti prebivalstva. Stolpec, ki prikazuje število prebivalcev na km^2, poimenujte Gostota prebivalstva.

```
SELECT država, prebivalstvo/površina AS 'Gostota prebivalstva'
  FROM države
ORDER BY prebivalstvo/površina DESC;
```

```
SELECT država, prebivalstvo/površina AS 'Gostota prebivalstva'
  FROM države
ORDER BY prebivalstvo/površina ASC;
```

10. Prikažite gostoto prebivalstva za Slovenijo. V glavi stolpca naj piše: Gostota prebivalstva za Slovenijo.

```
SELECT prebivalstvo/površina AS 'Gostota prebivalstva za Slovenijo'
  FROM države
 WHERE država = 'Slovenija';
```

11. Prikažite ime države in BDP na prebivalca. Spisek naj bo urejen naraščajoče po BDP na prebivalca. Stolpec, v katerem je prikazana izračunana vrednost, naj se imenuje BDP na prebivalca.

```
SELECT država, BDP/prebivalstvo AS 'BDP na prebivalca'
  FROM države
ORDER BY BDP/prebivalstvo;
```

12. Stolpcu dajte ime Število evropskih držav v bazi in izpišite podatke o številu držav, pri katerih je kot regija navedena Evropa.

```
SELECT COUNT(*) 'Število Evropskih držav v bazi'
  FROM države
 WHERE regija='Evropa';
```

13. Koliko je držav z več kot 50 milijoni prebivalcev?

```
SELECT COUNT(*) 'Število držav z več kot 50 milijoni prebivalcev'
  FROM države
 WHERE prebivalstvo > 50000000;
```

14. Kolikšno je skupno število prebivalcev v državah, ki so v bazi?

```
SELECT SUM(prebivalstvo) 'Skupno število prebivalcev'
  FROM države;
```

15. Kolikšno je skupno število prebivalcev v evropski regiji?

```
SELECT SUM(prebivalstvo) 'Skupno število evropskih prebivalcev'
  FROM države;
 WHERE regija='Evropa';
```

16. Prikažite skupno število prebivalcev za posamezno regijo. Seznam naj bo urejen padajoče po skupnem številu prebivalcev.

```
  SELECT regija, SUM(prebivalstvo) 'število prebivalcev'
    FROM države
GROUP BY regija
ORDER BY SUM(prebivalstvo) DESC;
```

17. Prikažite vse regije, ki imajo več kot 500.000.000 prebivalcev.

```
  SELECT regija, SUM(prebivalstvo) 'število prebivalcev'
    FROM države
GROUP BY regija
  HAVING SUM(prebivalstvo) > 500000000;
```

18. Prikažite skupno število prebivalcev in gostoto prebivalstva na km^2 za posamezno regijo. Seznam naj bo urejen padajoče po skupni gostoti prebivalcev.

```
  SELECT regija, SUM(prebivalstvo), SUM(prebivalstvo)/SUM(površina)
'gostota prebivalstva'
    FROM države
GROUP BY regija
ORDER BY SUM(prebivalstvo)/SUM(površina) DESC;
```

19. Prikažite imena držav in BDP na prebivalca za prvih 60 najbogatejših držav.

```
SELECT država, BDP/prebivalstvo AS 'BDP na prebivalca'
  FROM države
ORDER BY BDP/prebivalstvo DESC
LIMIT 60;
```

20. Pokažite imena držav in BDP na prebivalca za 10 % najbogatejših držav. Najprej izračunajte, koliko je 10 % vseh držav, in rezultat uporabite pri poizvedbi.

```
SELECT ROUND(COUNT(*)*(10/100))
  FROM države;

SELECT država, BDP/prebivalstvo
FROM države
ORDER BY BDP/prebivalstvo DESC
LIMIT tu_vnesete_rezultat_prve_poizvedbe; // v našem primeru »21«
```

21. Prikažite povprečni BDP na prebivalca po regijah. Spisek naj bo urejen naraščajoče po povprečnem BDP.

```
SELECT regija, AVG(BDP/prebivalstvo) AS 'BDP na prebivalca'
  FROM države
GROUP BY regija
ORDER BY AVG(BDP/prebivalstvo);
```

22. Izpišite ime regije in površino najmanjše države za Evropo, Azijo, Afriko in Oceanijo.

```
SELECT *
FROM države
WHERE (regija, površina) IN -- (('Afrika', 259), ('Azija', 300),
('Evropa', 2), ('Oceanija', 10))
(SELECT regija, MIN(površina)
FROM države
WHERE regija IN ('Evropa', 'Azija', 'Afrika', 'Oceanija')
GROUP BY regija);
```

23. Izpišite države in njihove površine za vse države v bazi, pri katerih je površina med polovico in dvakratnikom površine Slovenije.

```
SELECT država, površina
FROM države
WHERE površina BETWEEN (SELECT površina/2
                          FROM države
                         WHERE država='Slovenija')
                   AND
                       (SELECT površina*2
                          FROM države
                         WHERE država='Slovenija');
```

9.2 Rešitve vaj s strani 39 (Vaje – Časovne funkcije)

1. Razvrstite osebe od najstarejše do najmlajše.

```
SELECT *
  FROM rojstni_dnevi
 ORDER BY rojstni_dan
```

--preveri podvojenost

```
 SELECT ime, priimek, COUNT(*) št
  FROM rojstni_dnevi
GROUP BY ime, priimek
HAVING št > 1
```

2. Izpišite dve najmlajši ženski in dva najstarejša moška.

```
(SELECT *
  FROM rojstni_dnevi
 WHERE spol='Ž'
 ORDER BY rojstni_dan DESC
 LIMIT 2)
 UNION
(SELECT *
  FROM rojstni_dnevi
 WHERE spol='M'
 ORDER BY rojstni_dan
 LIMIT 2)
```

3. Izpišite vse, ki bodo imeli rojstni dan v naslednjih 90 dneh.

```
SELECT ime, priimek, rojstni_dan, spol, DATEDIFF(DATE_ADD(rojstni_dan,
INTERVAL(YEAR(CURDATE()) - YEAR(rojstni_dan)) YEAR), CURDATE())
razlika_do_rd,
DATE_ADD(rojstni_dan, INTERVAL(YEAR(CURDATE()) - YEAR(rojstni_dan))
YEAR) rd_letos, rojstni_dan, CURDATE()
FROM `rojstni_dnevi`
WHERE DATEDIFF(DATE_ADD(rojstni_dan, INTERVAL(YEAR(CURDATE()) -
YEAR(rojstni_dan)) YEAR), CURDATE()) BETWEEN 0 AND 30
```

4. Poiščite vse, ki so stari med 10000 in 20000 dnevi.

```
SELECT *, DATEDIFF(CURDATE(),rojstni_dan)
FROM rojstni_dnevi
WHERE DATEDIFF(CURDATE(),rojstni_dan) BETWEEN 10000 AND 20000
```

5. Poiščite vse, ki so rojeni med drugo svetovno vojno in niso rojeni meseca septembra. (1939-09-01 do 1945-09-02)

```
SELECT *
FROM rojstni_dnevi
WHERE rojstni_dan BETWEEN 1939-09-01 AND 1945-09-02
  AND MONTH(rojstni_dan) <> 9;
```

6. Koliko oseb je rojenih v posameznem mesecu?

```
SELECT MONTH(rojstni_dan) mesec, COUNT(*) "število oseb"
FROM rojstni_dnevi
GROUP BY MONTH(rojstni_dan);
```

7. Izpišite mesece, v katerih je bilo rojeno vsaj sedem oseb.

```
SELECT MONTH(rojstni_dan) mesec, COUNT(*) "število oseb"
FROM rojstni_dnevi
GROUP BY MONTH(rojstni_dan)
HAVING COUNT(*)>=7;
```

8. Izpišite vse pare oseb, ki so bili rojeni manj kot 15 dni narazen.

```
SELECT *
  FROM rojstni_dnevi r1, rojstni_dnevi r2
 WHERE ABS (DATEDIFF (r1.rojstni_dan, r2.rojstni_dan)) < 15
 AND r1.id < r2.id
```

9. Izpišite vse pare oseb, katerih rojstni dnevi se razlikujejo za manj kot 5 dni.

```
SELECT *, ABS(DAYOFYEAR(r1.rojstni_dan)-DAYOFYEAR(r2.rojstni_dan))
razlika
  FROM rojstni_dnevi r1, rojstni_dnevi r2
WHERE r1.id < r2.id
 AND (ABS (DAYOFYEAR(r1.rojstni_dan)-DAYOFYEAR(r2.rojstni_dan)) <  5
OR
      ABS (DAYOFYEAR(r1.rojstni_dan)-DAYOFYEAR(r2.rojstni_dan)) >
360)
```

10. Koliko dni ste stari vi?

```
SELECT DATEDIFF(CURDATE(), '1974-03-06')
```

11. Izpišite vse, ki so bili rojeni v petek trinajstega.

```
SELECT *, DAYOFWEEK(rojstni_dan)
FROM rojstni_dnevi
WHERE DAYOFWEEK(rojstni_dan)=6
  AND DAY(rojstni_dan)=13;
```

Izpis časovnih spremenljivk v slovenščini:

```
SET lc_time_names = 'sl_SI';
SELECT DAYNAME(rojstni_dan), MONTHNAME(rojstni_dan)
from rojstni_dnevi
```

9.3 Rešitve vaj s strani 43 (Vaje – Številske in znakovne funkcije)

1. Tabeli rojstni_dnevi dodajte stolpec začetnice in ga napolnite z začetnicami priimka in imena.

```
ALTER TABLE rojstni_dnevi
ADD COLUMN začetnice CHAR(2);
```

```
UPDATE rojstni_dnevi
SET `začetnice`=CONCAT(SUBSTRING(priimek,1,1),SUBSTRING(ime,1,1));
```

2. V tabeli rojstni_dnevi trem naključnim zapisom dodajte po dva presledka na začetek in na konec.

```
UPDATE `rojstni_dnevi`
SET `ime`=CONCAT('  ', ime, '  ')
WHERE `id`=ROUND(RAND()*(SELECT COUNT(*)FROM rojstni_dnevi)+0.5)
LIMIT 1;
```

```
#TESTNA POIZVEDBA
     #Naključno število med 1 in številom zapisov
     SELECT ROUND(RAND()*(SELECT COUNT(*)FROM rojstni_dnevi)+0.5)

     #Naljučna vrstica
     SELECT *
     FROM rojstni_dnevi
     WHERE id=ROUND(RAND()*76+0.5);
```

3. Izpišite te zapise.

```
SELECT *
FROM rojstni_dnevi
WHERE ime!=TRIM(ime);
```

4. Posodobite tabelo, tako da teh presledkov ne bo več.

```
UPDATE rojstni_dnevi
SET ime=TRIM(ime);
```

5. Iz tabele rojstni_dnevi izpišite vse podatke za osebe, katerih ime se začne na isto črko, kot se konča priimek.

```
SELECT *
FROM rojstni_dnevi
WHERE
ORD(LOWER(SUBSTRING(ime,1,1)))=ORD(LOWER(SUBSTRING(priimek,CHAR_LENGTH
(priimek),1)))
```

```
#TESTNA POIZVEDBA
     SELECT *, SUBSTRING(ime,1,1) prva_ime,
SUBSTRING(priimek,CHAR_LENGTH(priimek),1) zadnja_priimek,
       SUBSTRING(priimek,LENGTH(priimek),1) zadnja_priimek_narobe,

STRCMP(SUBSTRING(ime,1,1),SUBSTRING(priimek,CHAR_LENGTH(priimek),1))
primerjava1,

ORD(LOWER(SUBSTRING(ime,1,1)))=ORD(LOWER(SUBSTRING(priimek,CHAR_LENGTH
(priimek),1))) primerjava2
     FROM rojstni_dnevi
     WHERE
SUBSTRING(ime,1,1)=SUBSTRING(priimek,CHAR_LENGTH(priimek),1);
```

6. Iz tabele rojstni_dnevi izpišite vse podatke za osebe, katerih mesto v abecedi prve črke priimka je enaka dnevu rojstva.

```
SELECT *, ORD(priimek)-64, ASCII(priimek)-64, DAY(rojstni_dan)
FROM rojstni_dnevi
WHERE ASCII(priimek)-64=DAY(rojstni_dan);
```

```
#TESTNA POIZVEDBA
     SELECT *, ORD(priimek)-64, ASCII(priimek)-64, DAY(rojstni_dan)
     FROM rojstni_dnevi;
```

7. Poiščite tiste osebe, ki imajo ime enako dolgo priimku.

```
SELECT *
FROM rojstni_dnevi
WHERE CHAR_LENGTH(ime)=CHAR_LENGTH(priimek);
```

```
#NAROBE, ČE IMAMO VEČBAJTNE ZNAKE
SELECT *
FROM rojstni_dnevi
WHERE LENGTH(ime)=LENGTH(priimek);
```

8. Preverite, ali je stavek palindrom, tj. stavek, ki se prebere naprej in nazaj enako.

Primera: Perica reže raci rep. Ali se bo Gordana na drog obesila?

a) V poizvedbi napišite stavek s presledki, z velikimi in malimi črkami, vendar brez ločil. Vrne naj vrednost 1 - je palindrom oz. 0 ni palindrom.

```
SELECT STRCMP(REVERSE(REPLACE('Perica reže raci rep.', ' ', '')),
REPLACE('Perica reže raci rep.', ' ', ''))
```

```
#TESTNA POIZVEDBA
     SELECT REVERSE(REPLACE('Perica reže raci rep', ' ', '')),
REPLACE('Perica reže raci rep', ' ', '')
     SELECT REVERSE(REPLACE('Perica reže raci rep.', ' ', '')),
REPLACE('Perica reže raci rep.', ' ', '')
```

9.4 Rešitve vaj s strani 46 (Vaje - INSERT, DELETE in UPDATE)

V podatkovni zbirki ustvarite novo tabelo z imenom "nove_države" in vanjo prepišite vse podatke za evropske države iz tabele "države".

```
CREATE TABLE nove_države
(
  št int,
  država varchar(50),
  regija varchar(50),
  površina int,
  prebivalstvo int,
  BDP bigint
) DEFAULT CHARSET=utf8;
```

```
INSERT INTO nove_države
SELECT *
  FROM države
 WHERE regija='Evropa';
```

V tabelo nove_države dodajte državi "Srbija" in "Črna gora".

```
INSERT INTO nove_države (država)
VALUES ('Srbija'), ('Črna gora');
```

Poiščite največjo vrednost v polju "št" in novima državama dodelite naslednji dve zaporedni številki.

```
SELECT MAX(št)
  FROM nove_države;
```

```
UPDATE nove_države
   SET št=205+1
WHERE država='Srbija';
```

```
UPDATE nove_države
   SET št=205+2
WHERE država='Črna gora';
```

ne gre:

```
UPDATE nove_države
   SET št=(SELECT MAX(št) FROM nove_države)+1
WHERE država='Srbija';
```

gre pa:

```
UPDATE nove_države SET št = ( SELECT maksimum +1
FROM (
  SELECT MAX( št ) AS maksimum
  FROM nove_države ) AS trenutna_tabela
)
WHERE država = 'Srbija'
```

in tudi:

```
UPDATE nove_države AS n,
(
SELECT MAX( št ) +1 AS maksimum
FROM nove_države
) AS n2
SET n.št = n2.maksimum WHERE država = 'Črna gora';
```

```
UPDATE nove_države AS n, (SELECT MAX( št )+1 AS maksimum FROM
nove_države) AS n2
SET n.št = n2.maksimum
WHERE država = 'Črna gora';
```

Na spletu poiščite ostale podatke za ti dve državi in popravite podatke.

```
UPDATE nove_države
   SET regija='Evropa', površina=88361, prebivalstvo=10147398,
BDP=54310000000000
 WHERE država='Srbija';
```

```
UPDATE nove_države
   SET regija='Evropa', površina=13812, prebivalstvo=620029,
BDP=17600000000
 WHERE država='Črna gora';
```

Zbrišite državo "Srbija in Črna gora".

```
DELETE FROM nove_države
 WHERE država='Srbija in Črna gora';
```

V tabeli nove_države pretvorite površino iz kvadratnih kilometrov v kvadratne milje. (1 kvadratni kilometer = 0.386102159 kvadratne milje, obratno 2.58998811)

```
UPDATE nove_države
   SET površina = površina * 0.386102159;
```

```
UPDATE nove_države
   SET površina = površina * 2.58998811;
```

V tabelo nove_države dodajte državo "Indija Koromandija" iz regije "Nije", ki ima eno kvadratno miljo in pet prebivalcev.

```
INSERT INTO nove_države (država, regija, površina, prebivalstvo)
VALUES ('Indija Koromandija', 'Nije', 1, 5);
```

V tabelo nove_države dodajte vse države iz tabele države, ki imajo površino manjšo kot 20000 kvadratnih kilometrov. Hkrati pretvorite kvadratne kilometre v kvadratne milje.

```
INSERT INTO nove_države
SELECT št, država, regija, površina * 0.386102159, prebivalstvo, BDP
FROM države
WHERE površina<20000;
```

V tabeli nove_države za 100000 znižajte BDP vsem državam, ki imajo manj kot 3000000 prebivalcev.

```
UPDATE nove_države
   SET BDP=BDP-100000
 WHERE prebivalstvo<3000000;
```

V tabeli nove_države povečajte prebivalstvo za 10 % državam, ki se začnejo na črko A.

```
UPDATE nove_države
   SET prebivalstvo=prebivalstvo*1.1
 WHERE država LIKE 'A%';
```

V tabeli nove_države odstranite države, ki imajo površino med 4000 in 5000 kvadratnih milj.

```
DELETE FROM nove_države
 WHERE površina BETWEEN 4000 AND 5000;
```

V tabeli nove_države odstranite vse države, ki imajo BDP večji kot 1000000000.

```
DELETE FROM nove_države
 WHERE BDP>1000000000;
```

V tabeli nove_države odstranite države, ki nimajo znanega BDP-ja.

```
DELETE FROM nove_države
 WHERE BDP IS NULL;
```

9.5 Rešitve vaj s strani 58 (Vaje – DDL in DML)

V rešitvah smo izbrali 13. in 14. maraton za ženske.

1. **Ustvarjanje tabel. [10 točk]**
   Ustvari in uporabi podatkovno zbirko z imenom oblike »številka_prvega_teka-in-številka_drugega_teka-tip_teka-spol« npr. »1-in-2-maraton-moški«. S pomočjo ukaza CREATE TABLE ustvarite tabeli za vaša dva teka. Tabeli poimenujte na način »številka_teka-tip_teka-spol«. Če imate npr. 1. in 2. maraton za moške, jih poimenujte »1-maraton-moški« in »2-maraton-moški«. Uporabite smiselna imena stolpcev in podatkovne tipe. Kodo shranite v datoteko 1.txt.

```
CREATE DATABASE `13-in-14-maraton-ženske`;
USE `13-in-14-maraton-ženske`;

CREATE TABLE `13-maraton-ženske` (
  mesto int,
  št int,
  ime varchar(50),
  letnica_rojstva int,
  država varchar(5),
  neto time,
  rezultat time,
  kat varchar(3),
  mesto_v_kat int
) DEFAULT CHARSET='utf8';

CREATE TABLE `14-maraton-ženske`(
  mesto int,
  št int,
  ime varchar(50),
  letnica_rojstva int,
  država varchar(5),
  neto time,
  rezultat time,
  kat varchar(3),
  mesto_v_kat int
) DEFAULT CHARSET='utf8';
```

2. **Shranjevanje podatkov s spleta. [10 točk]**
   V vašem brskalniku najdite ustrezen tek in z miško izberite celotno tabelo. Pritisnite Ctrl+C oz. kopiraj. Odprite beležnico in pritisnite Ctrl+V oz. prilepi. Shranite datoteko z imenom na enak način kot v prvi nalogi, le da ji dodate končnico .txt. Pazite na to, da datoteko shranite v utf-8 kodiranju in na to da datoteke za uvoz ne smejo imeti šumnikov. Ponovite proceduro za drugi tek.

13. maraton ženske (samo prvih deset tekmovalk)

```
#      Št.    Ime    LR     Država       netto        Rezultat     Kat.   # v Kat.
1      29     MESENTSEVA TATYANA 72     UKR    2:37:13      2:37:13      C      1
2      27     TRILINSKAYA NADEZDA       79     RUS    2:37:25      2:37:26      A      1
3      25     GEBRE ZEBENAY      85     ETH    2:39:20      2:39:21      A      2
4      24     VASILEVSKAYA LIDIA 73     RUS    2:41:50      2:41:51      C      2
5      26     RONO GEORGINA      80     KEN    2:43:09      2:43:10      A      3
6      30     BERG KAJSA   79    SWE    2:44:17      2:44:18      A      4
7      518    ŠUŠTARŠIČ MATEJA   73     SLO    2:57:15      2:57:17      C      3
8      347    MRAVLJE NEŽA       79     SLO    3:03:20      3:03:33      A      5
9      737    FORTIN ALEKSADRA   72     SLO    3:08:25      3:08:29      C      4
10     613    BLATNIK DAMJANA    80     SLO    3:14:23      3:14:26      A      6
```

14. maraton ženske (samo prvih deset tekmovalk)

```
#      Št.    Ime    LR     Država       netto        Rezultat     Kat.   # v Kat.
1      26     CHEPTONUI KILEL CAROLINE  81     KEN    2:25:24      2:25:24      A      1
2      25     FILONYUK TETYANA   84     UKR    2:26:54      2:26:55      A      2
3      28     GETNET KASSA SELOMIE      86     ETH    2:31:14      2:31:15      A      3
4      30     TOLA GELETO FATE   87     ETH    2:35:21      2:35:22      A      4
5      32     TRILINSKAYA NADEZDA       79     RUS    2:36:54      2:36:55      B      1
6      27     ROSSEEVA OLGA      82     RUS    2:37:53      2:37:54      A      5
7      33     RUBAN JULIA  83    UKR    2:39:06      2:39:07      A      6
8      31     MEKONNEN TIRUWORK  86     ETH    2:42:18      2:42:19      A      7
9      29     CHELIMO SELINA CHEMUNGE   82     KEN    2:44:57      2:44:58      A      8
10     723    LEHONKOVA NATALIYA 82     UKR    2:48:29      2:48:30      A      9
```

3. **Uvoz podatkov. [10 točk]**
   S pomočjo ukaza LOAD DATA uvozite podatke z obeh prej ustvarjeni datotek v ustrezni tabeli. Podatki v tako ustvarjenih datotekah so ločni s tabulatorjem. Kodo shranite v datoteko 3.txt.

```
LOAD DATA INFILE 'D:/13-maraton-zenske.txt'
INTO TABLE `13-maraton-ženske`
CHARACTER SET 'utf8'
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

```
LOAD DATA INFILE 'D:/14-maraton-zenske.txt'
INTO TABLE `14-maraton-ženske`
CHARACTER SET 'utf8'
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

V našem primeru sta datoteki shranjeni v korenski mapi diska D. Pazite na pot do datotek.

4. **Poizvedbe. [40 točk]**
   Rešite naslednje poizvedbe in rezultate izvozite v datoteke z imeni 4-1.txt, 4-2.txt, … (Vsaka izvožena datoteka prinese 1 točko.) Kodo vseh ustrezno označenih poizvedb shranite v datoteko 4.txt.:

1. Koliko je bilo vseh tekmovalcev na posameznem teku? [1+1 točka]

```
SELECT COUNT(*) 'št. tekmovalcev'
FROM `13-maraton-ženske`
UNION
SELECT COUNT(*) 'št. tekmovalcev'
FROM `14-maraton-ženske`
```

2. Izpiši najboljših 10 tekmovalcev z obeh tekih! [2+1 točka]

```
(SELECT *
FROM `13-maraton-ženske`
LIMIT 10)
UNION
(SELECT *
FROM `14-maraton-ženske`
LIMIT 10)
```

3. Izpiši rezultate vseh tekmovalcev z obeh tekov s priimkom, ki je enako vašemu. [3+1 točka]

```
(SELECT *
FROM `13-maraton-ženske`
WHERE ime LIKE '% STERLE')
UNION
(SELECT *
FROM `14-maraton-ženske`
WHERE ime LIKE '% STERLE')
```

4. Poiščite vse tekmovalce, ki so tekmovali v obeh tekih. [3+1 točka]

```
SELECT *
FROM `13-maraton-ženske` a, `14-maraton-ženske` b
WHERE a.ime=b.ime;
```

5. Poiščite vse tekmovalce, ki so tekmovali v obeh tekih in so izboljšali rezultat. [4+1 točka]

```
SELECT *
FROM `13-maraton-ženske` a, `14-maraton-ženske` b
WHERE a.ime=b.ime
AND a.rezultat > b.rezultat;
```

6. Kakšen je bil povprečni rezultat za oba teka? [3+1 točka]

```
SELECT SEC_TO_TIME(AVG(TIME_TO_SEC(a.rezultat)))
FROM `13-maraton-ženske` a
```

```
UNION
SELECT SEC_TO_TIME(AVG(TIME_TO_SEC(a.rezultat)))
FROM `14-maraton-ženske` a
```

7. Koliko je bilo tekmovalcev po državah z obeh tekov? [4+1 točka]

```
SELECT COUNT(*), a.država
FROM `13-maraton-ženske` a
GROUP BY a.država
UNION
SELECT COUNT(*), a.država
FROM `14-maraton-ženske` a
GROUP BY a.država
```

8. Izpišite vse skupine s po vsaj tremi tekmovalci, ki so imeli enake rezultate. [4+1 točka]

```
SELECT COUNT(a.rezultat), a.rezultat
FROM `13-maraton-ženske` a
GROUP BY a.rezultat
HAVING COUNT(a.rezultat)>2
UNION
SELECT COUNT(a.rezultat), a.rezultat
FROM `14-maraton-ženske` a
GROUP BY a.rezultat
HAVING COUNT(a.rezultat)>2
```

9. Napišite, poljubno smiselno poizvedbo, ki bo vsebovala agregacijski operator. [3+1 točka]

```
SELECT COUNT(a.rezultat), a.rezultat
FROM `13-maraton-ženske` a
GROUP BY a.rezultat
HAVING COUNT(a.rezultat)>1
```

ali pa

```
SELECT MIN(a.rezultat), a.država
FROM `13-maraton-ženske` a
GROUP BY a.država
```

10. Napišite, poljubno smiselno poizvedbo, ki bo vsebovala LEFT JOIN. [3+1 točka]

```
SELECT a.ime, b.ime
FROM `13-maraton-ženske` a
LEFT JOIN `14-maraton-ženske` b
USING (ime);
```

ali

```
SELECT a.ime, b.ime
FROM `13-maraton-ženske` a
LEFT JOIN `14-maraton-ženske` b
ON a.ime = b.ime
```

5. **Ažuriranje tabel in podatkov. [20 točk]**
   Rešite še naslednje naloge. Kodo vseh ustrezno označenih nalog shranite v datoteko 5.txt.:
   1. Prvi vaši tabeli dodajte stolpec z imenom slika, ki naj ima tak tip, da boste vanj lahko shranili slikovno datoteko. [5 točk]

```
ALTER TABLE `13-maraton-ženske`
ADD COLUMN slika MEDIUMBLOB;
```

   2. Vstavite vrstico z vašimi podatki. Vsebuje naj ime, priimek, letnico rojstva. [5 točk]

```
INSERT INTO `13-maraton-ženske` (ime, letnica_rojstva)
VALUES ('Uroš Sterle', 74);
```

   3. Popravite to vrstico tako, da ji dodate še sliko. [5 točk]

```
UPDATE `13-maraton-ženske`
SET slika = load_file('D:\\ Marathon_Man.png')
WHERE ime = 'Uroš Sterle';
```

   4. Zbrišite poljubno vrstico. [5 točk]

```
DELETE FROM `13-maraton-ženske`
WHERE mesto=13;
```

6. **Izvoz podatkov v datoteko. [10 točk]**
   S pomočjo ukaza SELECT ... INTO OUTFILE izvozite tabelo, ki ste jo spreminjali, v datoteko z imenom v obliki »številka_teka-tip_teka-spol-izvoz.txt«. Npr. »2-maraton-moški-izvoz.txt«. Kodo shranite v datoteko 6.txt.

```
SELECT *
FROM `13-maraton-ženske`
INTO OUTFILE 'D:\\13-maraton-zenske-izvoz.txt'
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\r\n';
```

9.6 Rešitve vaj s strani 61 (Vaje – Indeksi)

1. Ustvarite indeks na tabeli države na poljih št in država ter indeks v tabeli kratice_držav na polju država.

```
CREATE INDEX indeks_države_1
ON države (št, država);
```

```
CREATE INDEX indeks_kratice_držav_1
ON kratice_držav (država);
```

2. Popravite prvi indeks tako, da mu dodate polje regija.

```
DROP INDEX indeks_države_1
ON države;
```

```
CREATE INDEX indeks_države_1
ON države (št, država, regija);
```

3. Izpišite vse indekse na tabelah države in kratice_držav.

```
SHOW INDEXES
FROM države
FROM test;
```

```
SHOW INDEXES
FROM kratice_držav
FROM test;
```

9.7 Rešitve vaj s strani 64 (Vaje – Pogledi)

1. Ustvarite pogled z imenom "oznake_držav", ki bo vseboval ime države, regijo, dvočrkovno (A2), tročrkovno (A3) in numerično (N3) kodo po standardu ISO 3166, kodo mednarodnega olimpijskega komiteja (MOK), mednarodno avtomobilsko oznako (DS) in dvočrkovno internetno domeno (IANA). Pri tem uporabite podatke iz tabel države in kratice_držav.

```
CREATE VIEW oznake_držav
AS SELECT k.država, d.regija, k.A2, k.A3, k.N3, k.MOK, k.DS, k.IANA
FROM države d, kratice_držav k
WHERE d.država=k.država;
```

2. Popravite pogled tako, da bo vseboval tudi države, ki nimajo zapisov v tabeli "države".

```
ALTER VIEW oznake_držav
AS SELECT k.država, d.regija, k.A2, k.A3, k.N3, k.MOK, k.DS, k.IANA
FROM države d RIGHT JOIN kratice_držav k
ON d.država=k.država;
```

3. Izpišite države, katerih regija ni določena v popravljenem pogledu.

```
SELECT *
FROM oznake_držav
WHERE regija IS NULL;
```

4. Ustvarite pogled z imenom "internetne_oznake_držav", ki bo vseboval ime države in dvočrkovno internetno domeno (IANA).

```
CREATE VIEW internetne_oznake_držav
AS SELECT država, IANA
FROM oznake_držav;
```

5. Izpišite vse države, ki imajo enaki črki v internetni oznaki.

```
SELECT *
FROM internetne_oznake_držav
WHERE SUBSTR(IANA,2,1)=SUBSTR(IANA,3,1);
```

6. Izpišite vse poglede iz podatkovne zbirke test.

```
SHOW FULL TABLES
FROM test
WHERE Table_type='VIEW';
```

9.8 Rešitve vaj s strani 96 (Vaje – Transakcije)

1. Ustvarite tabelo imena s stolpcema id, ki je samoštevilo in primarni ključ in ime znakovnega tipa. Tabela naj podpira šumnike.

```
CREATE TABLE imena (
  id INT PRIMARY KEY AUTO_INCREMENT,
  ime VARCHAR(8)
) DEFAULT CHARSET='utf8';
```

2. Vstavite imeni Maks in Eva ter izpišite vsebino tabele.

```
INSERT INTO imena (ime)
VALUES ('Maks'),('Eva');
```

```
SELECT id, ime
FROM imena;
```

3. Začnite transakcijo.

```
START TRANSACTION;
```

4. Posodobite ime v Nace, kjer je id enak 1.

```
UPDATE imena
SET ime='Nace'
WHERE id = 1;
```

5. Ustvarite mesto vrnitve.

```
SAVEPOINT mesto_vrnitve;
```

6. Posodobite ime v Mica, kjer je id enak 2 ter izpišite vsebino tabele.

```
UPDATE imena
SET ime='Mica'
WHERE id = 2;
```

```
SELECT id, ime
FROM imena;
```

7. Zavrzite spremembe do ustvarjenega mesta vrnitve.

```
ROLLBACK TO SAVEPOINT mesto_vrnitve;
```

8. Izpišite vsebino tabele in zaključite transakcijo.

```
SELECT id, ime
FROM imena;
```

```
COMMIT;
```

9. Kateri dve imeni sta na koncu v tabeli?

```
Nace in Eva.
```

9.9 Rešitve vaj s strani 100 (Vaje – Arhiviranje in restavracija podatkovnih zbirk)

1. Naredite kopijo poljubne podatkovne zbirke.

```
mysqldump -u root -proot arhiv > arhiv.sql
mysqladmin -u root -proot create arhiv2
mysql -u root -proot arhiv2 < arhiv.sql
```

2. Arhivirajte podatkovno zbirko test skupaj s procedurami, funkcijami in dogodki, prožilce izpustite.

```
mysqldump -u root -proot --events --routines --skip-trigger test >
test.sql
```

3. Iz podatkovne zbirke podjetje arhivirajte tabeli usluzbenci in oddelki. V eni izvozni datoteki naj bo struktura tabel, v drugi pa podatki.

```
mysqldump -u root -proot --no-data podjetje usluzbenci oddelki >
d:\podjetje1.sql
mysqldump -u root -proot --no-create-info podjetje usluzbenci oddelki
> d:\podjetje2.sql
```

9.10 Rešitve vaj s strani 81 (Vaje – Shranjene procedure, funkcije, prožilci in dogodki)

9.10.1 PROCEDURE

1. Ustvarite proceduro z imenom vsota, ki sešteje dve realni števili. Z uporabo ustvarjene procedure izračunajte 123,45+12,345.

```
DELIMITER $

CREATE PROCEDURE vsota (IN a FLOAT, IN b FLOAT, OUT c FLOAT)
BEGIN
  SET c=a+b;
END;
$
DELIMITER ;

SET @st=0;
SELECT @st;
CALL vsota(123.45, 12.345, @st);
SELECT @st;
```

2. Ustvarite proceduro z imenom stevilo_usluzbencev, ki izpiše število zapisov tabele usluzbenci. Uporabite to proceduro.

```
DELIMITER &

CREATE PROCEDURE stevilo_usluzbencev()
BEGIN
  SELECT COUNT(*) FROM usluzbenci;
END;
&

DELIMITER ;

CALL stevilo_usluzbencev;
```

3. Ustvarite proceduro z imenom vnos_usluzbencev, ki v tabelo usluzbenci zapiše toliko vrstic z vašim imenom in priimkom, kolikor jih je podano v parametru procedure.

```
DELIMITER &

CREATE PROCEDURE vnos_usluzbencev(st INT, ime VARCHAR(20), priimek
VARCHAR(20))
BEGIN
  WHILE st>0 DO
    INSERT INTO usluzbenci (ime_usluzbenca, priimek_usluzbenca) VALUES
(ime, priimek);
     SET st=st-1;
  END WHILE;
END;
&

DELIMITER ;

CALL vnos_usluzbencev(5, 'Uroš', 'Sterle');
```

4. Ustvarite proceduro, ki poveča sredstva v tabeli projekt za 10 odstotkov.

```
DELIMITER &

CREATE PROCEDURE povecaj_sredstva()
BEGIN
  UPDATE projekt
  SET sredstva=sredstva*1.1;
END;
&

DELIMITER ;

CALL povecaj_sredstva();
```

5. Ustvarite proceduro, ki podvoji vse zapise tabele projekt.

```
DELIMITER &

CREATE PROCEDURE podvoji_projekt()
BEGIN
  INSERT INTO projekt
  SELECT * FROM projekt;
END;
&

DELIMITER ;
```

```
CALL podvoji_projekt();
```

9.10.2 FUNKCIJE

1. Ustvarite funkcijo z imenom potenca, ki izračuna x^y. Izračunajte 5^3.

```
DELIMITER &
CREATE FUNCTION potenca (x INT, y INT)
RETURNS INT
BEGIN
  DECLARE p INT DEFAULT 1;
  WHILE y >0 DO
    SET p = p * x;
    SET y = y - 1;
  END WHILE;
  RETURN p;
END;
&
DELIMITER ;

SELECT potenca(5, 3);
```

2. Ustvarite funkcijo za približno računanje sinusa s pomočjo formule:
   sin(x) = x - x^3/ 3! + x^5/ 5! Z uporabo ustvarjene funkcije izračunajte sin(pi/6).

```
DELIMITER &
CREATE FUNCTION sinus (x FLOAT)
RETURNS FLOAT
BEGIN
  RETURN x - x*x*x /(1*2*3) + x*x*x*x*x /(1*2*3*4*5);
END;
&
DELIMITER ;

SELECT sinus(pi()/6);
```

3. Ustvarite funkcijo za približno računanje arkus tangensa s pomočjo formule:
   arc tan(x) = x - x^3/ 3 + x^5/ 5. Z uporabo ustvarjene funkcije izračunajte arctan(1).

```
DELIMITER &
CREATE FUNCTION arkus_tangens (x FLOAT)
RETURNS FLOAT
BEGIN
  RETURN x - x*x*x / 3 + x*x*x*x*x /5;
END;
&
```

```
DELIMITER ;

SELECT arkus_tangens(1);
```

4. Ustvarite funkcijo z imenom besede, ki vrne število besed vstavljenega niza znakov. Z uporabo ustvarjene funkcije preštejte besede stavka »Danes je lep dan.«

```
DELIMITER &
CREATE FUNCTION besede (niz VARCHAR(100))
RETURNS INT
BEGIN
  DECLARE st_besed INT DEFAULT 0;
  DECLARE i INT DEFAULT 1;
  WHILE i < CHAR_LENGTH(niz) DO
    IF SUBSTR(niz, i, 1)=' ' AND SUBSTR(niz, i+1, 1)<>' ' THEN
        SET st_besed=st_besed+1;
    END IF;
      SET i=i+1;
  END WHILE;
  RETURN st_besed;
END;
&
DELIMITER ;

SELECT besede(" Perica  reže   raci rep.  ");
```

5. Ustvarite funkcijo z imenom st_usluzbencev, ki izpiše število zapisov tabele usluzbenci. Uporabite to funkcijo.

```
DELIMITER &
CREATE FUNCTION st_usluzbencev()
RETURNS INT
BEGIN
  DECLARE st INT DEFAULT 0;
  SELECT COUNT(*) INTO st
  FROM usluzbenci;
  RETURN st;
END;
&
DELIMITER ;

SELECT st_usluzbencev();
```

9.10.3 PROŽILCI

1. Ustvarite prožilec z imenom pozitivna_sredstva, ki ob vnosu negativne vrednosti v polje sredstva tabele projekt vnese vrednost 0. Napišite SQL stavek, ki sproži ustvarjeni prožilec.

```
DELIMITER &

CREATE TRIGGER pozitivna_sredstva
BEFORE INSERT
ON projekt
FOR EACH ROW
BEGIN
  IF NEW.sredstva<0 THEN
    SET NEW.sredstva=0;
  END IF;
END;
&

DELIMITER ;

INSERT INTO projekt VALUES ('p4', 'Pohorje', -100);
```

2. Ustvarite prožilec popravi_oddelek, ki ob spremembi vrednosti polja st_oddelka tabele oddelki popravi ustrezne zapise tudi v tabeli usluzbenci. Napišite SQL stavek, ki sproži ustvarjeni prožilec.

```
DELIMITER &

CREATE TRIGGER popravi_oddelek
AFTER UPDATE
ON oddelki
FOR EACH ROW
BEGIN
  IF NEW.st_oddelka <> OLD.st_oddelka THEN
    UPDATE usluzbenci
      SET st_oddelka=NEW.st_oddelka
      WHERE st_oddelka=OLD.st_oddelka;
  END IF;
END;
&

DELIMITER ;

UPDATE oddelki
SET st_oddelka='d7'
WHERE st_oddelka='d1';
```

3. Ustvarite prožilec usluzbenci_v_arhiv, ki ob brisanju zapisa iz tabele usluzbenci le-tega zapiše v tabelo usluzbenci_arhiv, ki ima enako strukturo kot tabela usluzbenci. Napišite SQL stavek, ki sproži ustvarjeni prožilec.

```
CREATE TABLE usluzbenci_arhiv
     (st_usluzbenca INTEGER NOT NULL,
     ime_usluzbenca VARCHAR(20) NOT NULL,
     priimek_usluzbenca VARCHAR(20) NOT NULL,
     st_oddelka CHAR(4) NULL
     ) ENGINE=MyISAM DEFAULT CHARSET=utf8;

DELIMITER &

CREATE TRIGGER usluzbenci_v_arhiv
BEFORE DELETE
ON usluzbenci
FOR EACH ROW
BEGIN
  INSERT INTO usluzbenci_arhiv
  VALUES (OLD.st_usluzbenca, OLD.ime_usluzbenca,
OLD.priimek_usluzbenca, OLD.st_oddelka);
END;
&

DELIMITER ;

DELETE FROM usluzbenci
WHERE priimek_usluzbenca='Novak';
```

4. Izpišite vse prožilce tabele uslužbenci.

```
SHOW TRIGGERS
FROM podjetje
WHERE `Table`='usluzbenci';
```

5. Izbrišite prožilec usluzbenci_v_arhiv.

```
DROP TRIGGER usluzbenci_v_arhiv;
```

9.10.4 DOGODKI

```
SET GLOBAL event_scheduler = ON;
```

1. Ustvarite dogodek z imenom izprazni_tabelo_usluzbenci, ki bo čez eno uro izpraznil tabelo uslužbenci. Po izvedbi naj se dogodek ne zbriše.

```
CREATE EVENT izprazni_tabelo_usluzbenci
ON SCHEDULE
AT TIMESTAMP(NOW() + INTERVAL 1 MINUTE)
ON COMPLETION PRESERVE
  DO TRUNCATE TABLE usluzbenci;
```

2. Onemogočite in ponovno omogočite dogodek izprazni_tabelo_usluzbenci.

```
ALTER EVENT izprazni_tabelo_usluzbenci
  DISABLE;

ALTER EVENT izprazni_tabelo_usluzbenci
  ENABLE;
```

3. Preimenujte dogodek izprazni_tabelo_usluzbenci v izprazni_usluzbence.

```
ALTER EVENT izprazni_tabelo_usluzbenci
    RENAME TO izprazni_usluzbence;
```

4. Izbrišite dogodek izprazni_usluzbence.

```
DROP EVENT izprazni_usluzbence;
```

10 Literatura

- Cabral S. K., Murphy K. (2011). MySQL Administrator's Bible. Indianapolis: John Wiley & Sons.
- Drenovec J. (2004). Podatkovne baze. Pridobljeno 1.9.2011 iz http://drenovec.tsckr.si/.
- Kofler M. (2005). The Definitive Guide to MySQL 5, Third Edition. Berkeley: Apress.
- Kai 'Oswald' Seidler (2011). XAMPP. Pridobljeno 21.9.2011 iz http://www.apachefriends.org/en/xampp.html.
- Oracle (2012). MySQL 5.5 Reference Manual. Pridobljeno 1.9.2011 iz http://dev.mysql.com/doc/refman/5.5/en/index.html.
- Refsnes Data (2011). W3Schools SQL Tutorial. Pridobljeno 1.9.2011 iz http://www.w3schools.com/sql/default.asp.
- Wikipedija, Prosta enciklopedija (2012). Pridobljeno 2.2.2012 iz http://sl.wikipedia.org/wiki/Glavna_stran.
- Wordpress Slovenija (2012). Pridobljeno 2.4.2012 iz http://sl.wordpress.org/.