UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Ines Frelih

Spletni sistem za vaje iz jezika SQL

Diplomska naloga

Ljubljana, 2011

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Zahvala

Zahvalila bi se rada vsem, ki so mi kakorkoli pomagali pri študiju in nastanku diplomske naloge. Za pomoč in nasvete pri ustvarjanju diplome bi se posebej zahvalila mentorju mag. Matiji Lokarju. Hvala tudi mojemu fantu in družini, ki so me podpirali v vseh letih študija.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Kazalo

ZAHVALA ... **2**
KAZALO ... **3**
KAZALO SLIK ... **5**
PROGRAM DIPLOMSKE NALOGE ... **6**
POVZETEK ... **7**
ABSTRACT ... **7**
1 PODATKI ... **8**
1.1 SHEMA ŠOLA ... 8
1.2 SHEMA ROKOMET ... 9
1.3 SHEMA NAROČANJE ... 10
1.4 SHEMA VIKTORJI ... 11
2 NAJNUJNEJŠE O PODATKOVNIH BAZAH ... **13**
2.1 JEZIK SQL ... 13
3 STAVEK SELECT ... **15**
3.1 ENOSTAVNA OBLIKA ... 15
3.1.1 Poizvedba po vseh podatkih ... *15*
3.1.2 Poizvedba po enem ali več stolpcih ... *16*
3.1.3 Poimenovanje stolpcev v dobljeni tabeli ... *17*
3.1.4 Poizvedba po različnih vrsticah ... *18*
3.2 WHERE ... 19
3.2.1 Enostavni relacijski operatorji ... *19*
3.2.2 Primerjanje različnih podatkovnih tipov ... *20*
3.2.3 Logični operatorji ... *22*
3.2.3.1 AND ... 22
3.2.3.2 OR ... 23
3.2.3.3 NOT ... 23
3.2.4 Posebni relacijski operatorji ... *24*
3.2.4.1 BETWEEN ... 24
3.2.4.2 IN ... 25
3.2.4.3 LIKE ... 25
3.2.4.4 IS NULL ... 27
3.2.5 Sestavljene poizvedbe ... *28*
3.3 ZDRUŽEVALNE FUNKCIJE ... 30
3.3.1 Funkcija MAX ... *30*
3.3.2 Funkcija MIN ... *31*
3.3.3 Funkcija SUM ... *31*
3.3.4 Funkcija AVG ... *32*
3.3.5 Funkcija COUNT ... *32*
3.4 POIZVEDBE S PODPOIZVEDBAMI ... 34
3.5 ORDER BY ... 36
3.6 GROUP BY ... 39
3.7 HAVING ... 42
3.8 ZDRUŽEVANJE TABEL ... 44
3.8.1 Ključi ... *46*
3.8.1.1 Primarni ključ ... 46

3.8.1.2 Tuji ključ 47
3.8.2 Relacije *47*
3.8.3 INNER JOIN *49*
3.8.4 LEFT OUTER JOIN *50*
3.8.5 RIGHT OUTER JOIN *51*
3.8.6 FULL JOIN *52*
3.8.7 Združevanje treh ali več tabel *52*

4 PROGRAM **55**

4.1 DELOVANJE PROGRAMA 55
4.2 UPORABNIKOVA STRAN 58
4.3 ADMINISTRATORSKA STRAN 62
4.4 RAZVOJNA ORODJA 66
4.5 NEKAJ ZANIMIVIH TOČK RAZVOJA 66
4.5.1 Primerjanje uporabnikovega stavka SELECT s pravilnim stavkom SELECT *66*
4.5.2 Tabele *66*
4.5.3 Vrstni red *67*
4.5.4 Prikazano *68*
4.5.5 Uporaba šumnikov *68*

ZAKLJUČEK **69**

LITERATURA **70**

SPLETNI VIRI **70**
DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Kazalo slik

Slika 1: naloge 55
Slika 2: kategorije 56
Slika 3: uporabnikova stran 58
Slika 4: posamezna naloga 59
Slika 5: gumb Preveri 60
Slika 6: gumb Namig 60
Slika 7: gumb Pravilni rezultati 61
Slika 8: gumb Pravilna poizvedba 61
Slika 9: pravilni rezultati 62
Slika 10: seznam nalog 63
Slika 11: dodajanje naloge 64
Slika 12: seznam kategorij 65
Slika 13: dodaj kategorijo 65
Slika 14: imena tabel 67
Slika 15: prvoten vrstni red 67
Slika 16: vrstni red po preoblikovanju 68

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Program diplomske naloge

V okviru diplomske naloge sestavite spletni sistem, ki bo omogočal vadbo poizvedovalnih stavkov SELECT ter predstavite sam stavek SELECT. Sestavljen naj bo iz uporabniškega dela, ki omogoča izvajanje stavkov in kontrolo pravilnosti poizved. Ima naj tudi administrativni del, kjer je možno urejati vprašanja in izbirati, katera vprašanja so prikazana v uporabniškem delu. V teoretičnem delu opišite stavek SELECT. Pri tem se omejite le na tiste značilnosti jezika, ki jih bo pokrival vaš spletni sistem.

mentor

mag. Matija Lokar

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Povzetek

Diplomska naloga je namenjena študentom Fakultete za matematiko in fiziko kot pomoč pri učenju uporabe stavka SELECT v jeziku SQL.

Diplomska naloga je sestavljeno iz pisnega dela in spletne strani. V prvem poglavju diplome je na kratko opisano najnujnejše o podatkovnih bazah in jeziku SQL. Nato je postopoma obravnavan stavek SELECT, od enostavne oblike pa vse do zapletene uporabe večih tabel. Na koncu je opisano delovanje spletne aplikacije.

Praktični del diplomske naloge predstavlja spletna stran z nalogami, ki jih lahko študenti rešujejo in tako usvojijo znanje uporabe stavka SELECT. Na spletni strani so naloge, ki jih obravnavam v pisnem delu diplomske naloge. Poleg teh nalog je še nekaj nalog iz vseh obravnavanih tem. V diplomski nalogi se sklicujem na naloge iz spletne strani, tako da lahko študenti pri učenju kombinirajo med teorijo in nalogami s spletne strani.

Poleg strani z nalogami obstaja še administratorska stran za upravljanje z nalogami. Administrator lahko popravlja in dodaja nove naloge ter kategorije. Vsaki nalogi pripada ime, besedilo, pravilen SQL, namig, imena tabel, sklic na diplomsko nalogo in kategorija.

Abstract

This thesis is aimed towards students of Faculty of Mathematics and Physics as an aid in learning the use of the statement SELECT of the SQL language.

The thesis consists of a written part and a web page. The first chapter outlines the essentials of RDMS and the SQL language. SELECT statement is then gradually described, from the basic form to the complex usage of multiple tables. Finally, the functionality of the web page is described.

The practical part of this thesis represents a web page with the tasks students can solve and acquire the knowledge of the SELECT statement. The tasks on the web site are also used in the written part. In addition to these tasks, there are also some tasks from all of the discussed theory. In the thesis I refer to the tasks on the web page so that students can combine learning theory and the tasks from the web page.

In addition to the web page with the tasks, there is an administrator page with management functions. The administrator can revise and add new tasks and categories. Each task has a name, text, correct SQL, tip, table names, a reference to the thesis and a category.

Math. Subj. Class. (2011): 68M11, 68N01, 68N15, 68P01, 68P15

ComputingReviewClass. System (1998): A.1, D.3.0, D.3.1, D.3.3, E.1, H.2.1, H.2.3, H.2.4, K.3.1, K.3.2

Ključne besede: PostgreSQL, SQL, SELECT, WHERE, JOIN, tabela, stolpec, vrstica, celica, podatek, primarni ključ, tuji ključ, relacije, podatkovna baza

Keywords: PostgreSQL, SQL, SELECT, WHERE, JOIN, table, column, row, cell, data, primarykey, foreignkey, relation, database

1 Podatki

Preko spletne strani http://uciSeSQL.fmf.uni-lj.si/Nivo1 dostopamo do nalog iz snovi, o kateri govori diplomska naloga. Tam je tudi baza podatkov `InesDiplomska`, kjer imamo zbrane različne podatke. To bazo bomo uporabljali pri razlagi snovi v pisnem delu naloge.

Bazo `InesDiplomska` smo razdelili na sheme. Naša baza je sestavljena iz shem `narocanje`, `rokomet`, `sola in viktorji`. V posamezni shemi so shranjene tiste tabele, ki so si tematsko podobne. Več shem (tem) imamo zato, da bomo snov lahko razlagali na več različnih prime

rih. Z uporabo shem smo se izognili potrebi, da bi morali na spletni strani uporabljati različne baze.

Podrobneje bomo o tem, kaj so tabele in kakšni so njihovi sestavni deli, spregovorili v razdelku 2. Tu le na kratko opišimo uporabljene sheme in pripadajoče tabele.

1.1 Shema šola

V shemi `sola` hranimo podatke o dijakih, razredih in razrednikih na neki šoli. Dijak obiskuje natanko en razred. V posameznem razredu je lahko več dijakov, vsak razred pa ima natanko enega razrednika. Zaradi pomanjkanja učiteljev na tej šoli je lahko vsak učitelj razrednik večim razredom.

V tabeli `dijaki` so shranjeni podatki o identifikacijski številki dijaka, imenu in priimku ter oznaki razreda, ki ga dijak obiskuje.

Tabela 1: dijaki

dijaki			
id_dijaka [integer]	ime [character varying (255)]	priimek [character varying (255)]	id_razreda [integer]
2500	Rok	Bizjak	1
2501	Miha	Petek	1
2502	Ksenja	Jerman	3
2503	Jaka	Zajc	1
2504	Marko	Sever	6
2505	Boris	Kuhar	5
2506	Mojca	Jerman	4
2507	Renata	Rupnik	4
2508	Rok	Breznik	5

Vsak stolpec ima svoj tip vrednosti. Če gre za niz (tip character varying), določimo še dolžino najdaljšega možnega niza. Ta dolžina je zapisana v oklepajih poleg tipa vrednosti.

V tabelo `razredniki` shranjujemo imena razrednikov, identifikacijske številke razrednikov in imena predmetov, ki jih poučujejo.

Tabela 2: razredniki

razredniki		
razrednik [character varying (255)]	id_razrednika [integer]	predmet [character varying (255)]
Mlakar	1	matematika
Zupanc	2	športna vzgoja
Majcen	3	slovenščina
Zorko	4	matematika
Jereb	5	fizika
Males	6	zgodovina

V tabeli `razredi` so shranjeni številka razrednika, ime razreda in povprečna ocena vseh dijakov v razredu .

Tabela 3: razredi

razredi			
stevilka_razreda [integer]	stevilka_razrednika [integer]	povprecna_ocena [numeric (10, 2)]	razred [character varying (255)]
1	1	3.45	1.a
2	2	3.02	1.b
3	3	4.10	2.a
4	4	3.67	2.b
5	5	3.86	3.a
6	6	3.22	3.b
7	4	3.55	3.c

1.2 Shema rokomet

V shemi `rokomet` so shranjeni podatki o evropskih prvenstvih v rokometu. Vsaki dve leti poteka evropsko prvenstvo v rokometu, ki je vsakič organizirano v neki državi. Po koncu prvenstva dobimo uvrstitve posameznih reprezentanc držav.

V tabeli `drzave` so shranjena imena držav, ki so nastopala na prvenstvih, njihove kratice in število prebivalcev .

Tabela 4: drzave

drzave		
kratica [character varying (3)]	naziv [character varying (255)]	stevilo_prebivalcev [integer]
A	Avstrija	8192880
D	Nemčija	82422299
SLO	Slovenija	2038733
⋮	⋮	⋮

V tabeli `financni_podatki_drzav` so shranjeni podatki o bruto domačih proizvodih (BDP) tistih držav, ki so nastopale na prvenstvih. V stolpcu kratica so shranjene kratice držav, ki so nastopale na prvenstvih. V stolpcu `bdp` imamo zabeležene bruto domače proizvode držav.

Tabela 5: financni_podatki_drzav

financni_podatki_drzav	
kratica [character varying (255)]	bdp [integer]
A	376841
CH	523772
D	3315643
F	2582527
⋮	⋮

V tabeli `prvenstva` so shranjeni podatki o letu prvenstva in kratice tiste države, kjer je prvenstvo potekalo.

Tabela 6: prvenstva

prvenstva	
leto [integer]	kratica_drzave_gostiteljice [character varying (3)]
1994	P
1996	E
1998	I
2000	HR
2002	S
2004	SLO
⋮	⋮

V tabeli `rezultati_prvenstev` imamo podatke o rezultatih evropskih prvenstev v rokometu. Tabelo sestavljajo stolpci: `id`, `leto`, `kratica_drzave` in `uvrstitev`.

Tabela 7: rezultati_prvenstev

rezultati_prvenstev			
id [serial]	leto [integer]	kratica_drzave [character varying (3)]	uvrstitev [integer]
1	2010	F	1
2	2010	HR	2
7	2008	DK	1
⋮	⋮	⋮	⋮

1.3 Shema naročanje

V neki trgovini si beležimo podatke o naročilih in strankah, ki so ta naročila opravile.

V shemi `naročanje` imamo tabeli naročnikov in naročil. V tabeli `narocnik` so shranjeni identifikacijska številka naročnika, ime in priimek naročnika, ter naslov, ki ga sestavljata ulica in mesto.

Tabela 8: narocnik

narocnik				
id_narocnik [integer]	ime [character varying (255)]	priimek [character varying (255)]	naslov [character varying (255)]	mesto [character varying (255)]
1	Janez	Novak	Novi dom 32	Trbovlje
2	Miha	Jazbec	Stari trg 42	Ljubljana
3	Mojca	Smodic	Trg revolucije 18	Maribor
4	Ana	Kolar	Opekarna 66	Ljubljana
5	Anja			
6	Janez	Vesel	Leskoškova 55	Trbovlje

V tabeli `narocila` so shranjeni podatki o številki naročila, številki naročnika, številu naročenih artiklov in skupna vrednost naročila.

Tabela 9: narocila

narocila				
id_narocila [integer]	stevilka_narocila [integer]	stevilka_narocnika [integer]	stevilo_artiklov [integer]	vrednost_narocila [numeric (10,2)]
1	50121	1	5	1020.50
2	50122	1	2	955.33
3	50123	3	7	4680.12
4	50124	4	3	1574.08
5	50125		5	3211.99
6	50126	4	4	5723.40
7	50127	2	1	500.68
8	50128	1	10	7100.60
9	50129	3	2	1730.64
10	50129	3	1	382.55

Določeni podatki v tabelah namerno manjkajo. Prazne celice imajo vrednost NULL. Več o tem si lahko preberete v razdelku 3.2.4.4.

1.4 Shema viktorji

V shemi `viktorji` so shranjeni podatki o rezultatih podelitve viktorjev. Vsako leto podelijo več nagrad. Nagrajenec lahko dobi več različnih nagrad v istem letu.

V tabeli `seznam_nagrad` so shranjena imena nagrad in identifikacijske številke teh nagrad.

Tabela 10: seznam_nagrad

seznam_nagrad	
nagrada [character varying (255)]	nagrada_id [serial]
radijska osebnost	1
radijska postaja	2
glasbeni izvajalec	3
televizijska osebnost	4
televizijska oddaja	5
⋮	⋮

V tabeli `seznam_nagrajencev` so shranjena imena nagrajencev in njihove identifikacijske številke.

Tabela 11: seznam_nagrajencev

seznam_nagrajencev	
nagrajenec_id [serial]	nagrajenec [character varying (255)]
1	Denis Avdić
2	Radio Center
3	Siddharta
4	Boštjan Romih
5	Na zdravje
⋮	⋮

Tabela `razglasitev` je sestavljena iz 4 stolpcev. Vsaka razglasitev posamezne nagrade v letu ima svojo identifikacijsko številko (`id`). V tabeli so poleg nje še podatki o identifikacijskih številkah nagrade in nagrajenca ter o letu podelitve nagrade. Tabela nam torej pove, katerega leta je določen nagrajenec prejel neko nagrado.

Tabela 12: razglasitev

razglasitev			
id [serial]	nagrada [integer]	nagrajenec [integer]	leto [integer]
1	1	1	2009
2	2	2	2009
15	3	14	2008
18	7	6	2008
21	10	16	2008
⋮	⋮	⋮	⋮

2 Najnujnejše o podatkovnih bazah

Podatke pogosto hranimo v podatkovnih bazah. Podatkovna baza je zbirka pomensko povezanih podatkov. Olajša nam delo s podatki. Operacije nad podatkovno bazo nam omogočajo poizvedovanje, spreminjanje, brisanje, branje in pisanje podatkov. Vsi podatki so shranjeni na enem mestu in do njih lahko dostopa več uporabnikov hkrati.

Sistem za upravljanje podatkovne baze je zbirka programov, ki se uporabljajo za upravljanje podatkov v podatkovni bazi. V uporabi so večinoma sistemi za upravljanje z relacijskimi podatkovnimi bazami **RDBMS** (Relational database management system). Primeri takšnih sistemov so: MySQL, PostgreSQL, DB2, Oracle ...

V diplomski nalogi bomo uporabljali sistem PostgreSQL.

Podatki so v relacijski bazi shranjeni v dvodimenzionalnih tabelah. Podatke v tabelah obdelujemo s pomočjo ustreznega jezika, v našem primeru bo to jezik SQL. Kaj več bomo o jeziku SQL povedali v razdelku 2.1.

Tabele so sestavljene iz stolpcev in vrstic. Vrstica predstavlja zapis o objektu. Stolpec določa lastnosti in tip lastnosti objektov. Presek med vrstico in stolpcem imenujemo celica. V celici je vrednost objekta - podatek. Če celica nima podatka, pravimo, da ima vrednost NULL.

V stolpcu torej določimo podatkovni tip objektov. Naštejmo nekaj tipov, ki smo jih uporabili v diplomski nalogi:

- character varying – zaporedje znakov (niz), ki ne sme presegati določene maksimalne dolžine niza,
- integer – celo število,
- numeric – decimalno število,
- serial – sistem sam določi enolično celo število,
- date – datum,
- boolean – logične vrednosti (FALSE, TRUE),
- text– besedilo
- ...

2.1 Jezik SQL

Do vsebine tabel dostopamo s pomočjo poizvedovalnih jezikov. Najbolj razširjen poizvedovalni jezik je SQL (Structured Query Language). Uveljavil se je predvsem zaradi učinkovite in preproste uporabe. Za izvajanje ukazov, zapisanih v jeziku SQL, poskrbi RDBMS. V jeziku SQL obstajajo štirje osnovni stavki za obdelovanje podatkov:

- SELECT – povpraševanje po podatkih,
- INSERT – vstavljanje podatkov,
- UPDATE – posodabljanje podatkov,
- DELETE – brisanje podatkov.

V razdelku 3 si bomo podrobneje ogledali stavek SELECT.

Jezik SQL ne razlikuje med velikimi in malimi črkami. Dogovor pa je, da vse ključne besede pišemo z velikimi črkami, ostalo pa z malimi črkami. Odvečne presledke sistem prezre, tako da lahko poizvedbe raztegnemo čez več vrstic.

Različni RDBMS-ji ukaze v jeziku SQL interpretirajo različno. Razlike običajno niso bistvene, vendar včasih določen ukaz, napisan v eni različici SQL, ne deluje v drugem sistemu. Vse povedano v tej diplomski nalogi bo veljalo za SQL, kot ga uporabljamo s sistemom PostgreSQL, različica 9.0.1.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

3 Stavek SELECT

S pomočjo stavka **SELECT** lahko naredimo poizvedbo po podatkih v tabelah. Stavek **SELECT** kot rezultat vrne tabelo podatkov. Če ukaz uporabimo kot samostojen stavek v ukaznem načinu dela z bazo (kjer neposredno povprašujemo po podatkih v bazi), se rezultat poizvedbe (dobljena tabela) izpiše na zaslon.

Za izvedbo stavka **SELECT** potrebujemo ime tabele, v kateri so podatki, imena stolpcev v tabeli ter pogoje, s katerimi določimo, katere podatke želimo videti.

Osnovna struktura stavka **SELECT** je sledeča:

```
SELECT seznam podatkov, ki jih želimo videti
FROM tabele, kjer bomo podatke našli
WHERE pogoji, ki določajo, katere podatke želimo videti
```

Najpogosteje je tabela s podatki ena sama, seznam podatkov pa sestavljajo le z vejicami ločena imena stolpcev, zato ukaz uporabimo kot

```
SELECT ime_stolpca1, ime_stolpca2,…, ime_stolpcaN
FROM ime_tabele WHERE pogoj
```

3.1 Enostavna oblika

3.1.1 Poizvedba po vseh podatkih

Iz tabele želimo dobiti vse podatke. To lahko naredimo tako, da v stavek SELECT vpišemo vsa imena stolpcev.

```
SELECT stolpec1, stolpec2, stolpec3,..., stolpecN FROM tabela
```

Pri večjem številu stolpcev je bolj uporaben naslednji način.

```
SELECT * FROM tabela
```

Primer (Tabela 4: drzave)

Oglejmo si primer, kjer pridobimo vse podatke iz tabele držav.

1. način: *SELECT kratica, naziv, stevilo_prebivalcev FROM drzave*
2. način: *SELECT * FROM drzave*

Rezultat je v obeh primerih enak:

kratica	naziv	stevilo_prebivalcev
A	Avstrija	8192880
D	Nemčija	82422299
SLO	Slovenija	2038733
⋮	⋮	⋮

Način z uporabo * nam ne zagotavlja, da bomo stolpce dobili v določenem vrstnem redu. Zato takrat, kadar nas zanima točno določen vrstni red stolpcev, uporabimo zapis z naštevanjem stolpcev.

Zgornja stavka SELECT si lahko na spletni strani ogledate v kategoriji `enostavna oblika` pri 1. in 2. nalogi.

Za utrditev te oblike stavka SELECT so na spletni strani na voljo 46., 80., in 88. naloga.

3.1.2 Poizvedba po enem ali več stolpcih

Večinoma nas celotna tabela ne zanima. Uporabno je narediti poizvedbo, ki nam vrne le določen stolpec oziroma določene stolpce v tabeli.

Za poizvedbo po več stolpcih bomo uporabili naslednji stavek:

```
SELECT stolpec1, stolpec2, stolpec3,..., stolpecN FROM tabela
```

Vrstni red, v katerem naštejemo stolpce, je pomemben, zato naslednja stavka nista enakovredna in vrneta različne rezultate.

```
SELECT kratica, naziv FROM drzave
```

```
SELECT naziv, kratica FROM drzave
```

Če nas pri tej poizvedbi vrstni red stolpcev ne zanima, je seveda vseeno, katerega izmed omenjenih stavkov SELECT uporabimo. Drugače pa moramo na vrstni red stolpcev paziti. Tudi pri uporabi spletne aplikacije moramo paziti. Kadar naloga predpisuje vrstni red stolpcev, moramo uporabiti poizvedbo, ki bo vrnila prav tak vrstni red stolpcev.

Primer (Tabela 4: drzave)

Denimo, da nas v tabeli s podatki o državah zanimata ime države in njena kratica. Glede na želeni vrstni red stolpcev lahko uporabimo eno od naslednjih oblik:

```
SELECT kratica, naziv FROM drzave
```

kratica	naziv
A	Avstrija
D	Nemčija
SLO	Slovenija
⋮	⋮

```
SELECT naziv, kratica FROM drzave
```

naziv	kratica
Avstrija	A
Nemčija	D
Slovenija	SLO
⋮	⋮

Zgornja stavka SELECT si lahko na spletni strani ogledate v kategoriji `enostavna oblika` pri 4. in 131. nalogi.

Na spletni strani so za utrditev te oblike stavka SELECT na voljo naslednje naloge: 82, 13, 47, 51.

3.1.3 Poimenovanje stolpcev v dobljeni tabeli

Tabela rezultatov, ki jo vrne stavek SELECT, ima stolpce privzeto poimenovane tako kot tabela (tabele), iz katere izbiramo podatke. Lahko pa stolpce izhodne tabele preimenujemo. Pri tem nam pomaga ključna beseda **AS**.

```
SELECT stolpec AS "novo ime stolpca" FROM tabela
```

Primer (Tabela 4: drzave):

Potrebujemo le imena držav. V izhodni tabeli naj bo namesto `naziv` stolpcu ime `drzava`.

```
SELECT naziv AS "drzava" FROM drzave
```

Rezultat

drzava
Avstrija
Švica
Danska
Madžarska
Hrvaška
⋮

Seveda lahko določene stolpce pustimo poimenovane z originalnim imenom. Tako ukaz

```
SELECT naziv AS "drzava", kratica FROM drzave
```

vrne tabelo

drzava	kratica
Avstrija	A
Švica	CH
Danska	DK
Madžarska	H
Hrvaška	HR
⋮	⋮

Stavka SELECT, ki smo jih uporabili v tem razdelku, najdete na spletni strani v kategoriji `enostavna oblika` pri 3. in 50. nalogi.

Če želite preveriti svoje znanje poimenovanja stolpcev, lahko rešite naloge 48, 89 in 90.

3.1.4 Poizvedba po različnih vrsticah

V izhodni tabeli lahko dobimo dve ali več enakih vrstic. Z uporabo **SELECT DISTINCT** dosežemo, da so izhodne vrstice enolične. To pomeni, da se razlikujejo vsaj v vrednosti v enem stolpcu.

```
SELECT DISTINCT stolpec FROM tabela
```

Primer (Tabela 8: narocnik)

Denimo, da nas zanima, iz katerih krajev prihajajo naročniki. Če uporabimo kar

```
SELECT mesto FROM narocnik
```

dobimo

mesto
Trbovlje
Ljubljana
Maribor
Ljubljana
⋮

Vidimo, da se določeni kraji podvajajo. To so tisti kraji, iz katerih imamo več naročnikov. Ker pa nas zanimajo samo različni kraji, uporabimo

```
SELECT DISTINCT mesto FROM narocnik
```

in dobimo

mesto
Trbovlje
Ljubljana
Maribor

Če pa v poizvedbo vključimo še ime naročnika

```
SELECT DISTINCT mesto, ime FROM narocnik
```

dobimo rezultat

mesto	ime
Ljubljana	Miha
Trbovlje	Janez
	Anja
Maribor	Mojca
Ljubljana	Ana

Poizvedba vrne vrstice, ki so si različne v kombinaciji obeh stolpcev. V tabeli (Tabela 8: narocnik) vidimo, da imamo dva naročnika z imenom Janez, ki prihajata iz Trbovelj. Z ukazom **SELECT DISTINCT** dosežemo, da dobimo kot rezultat samo en zapis Janeza iz Trbovelj.

Pri **SELECT DISTINCT** dobimo v rezultatu samo eno ponovitev kombinacije podatkov, pri uporabi oblike brez **DISTINCT** pa se kombinacije podatkov lahko večkrat ponovijo. Torej pri **SELECT DISTINCT** dobimo kvečjemu toliko (rezultatov) vrstic kot pri SELECT brez uporabe **DISTINCT**.

Primer (Tabela 8: narocnik)

Če brez uporabe **DISTINCT** izpišemo stolpec `ime` iz tabele naročnikov, se bodo imena ponavljala, oblika z **DISTINCT** pa nam bo povedala, katera so vsa različna imena v tabeli `narocnik`.

SELECT ime FROM narocnik

ime
Miha
Anja
Mojca
Ana
Janez
Janez

SELECT DISTINCT ime FROM narocnik

ime
Miha
Anja
Mojca
Ana
Janez

Zgornje stavke SELECT si lahko ogledate v kategoriji `enostavna oblika` pri nalogah 13, 14, 15, 80 in 81.

Na spletni strani so za utrditev te oblike stavka SELECT na voljo naslednje naloge: 52, 71, 115 in 91.

3.2 Where

Le redko nas zanimajo prav vsi podatki v tabeli. Če stavku **SELECT** dodamo del **WHERE**, dobimo v rezultatu samo tiste vrstice, kjer podatki ustrezajo danim pogojem.

Osnovna struktura te oblike stavka SELECT je

```
SELECT stolpec FROM tabela WHERE pogoji
```

Pri sestavi izhodne tabele bodo upoštevane le tiste vrstice v tabeli `tabela`, ki zadoščajo pogojem.

Pri sestavljanju pogojev lahko uporabimo relacijske in logične operatorje. Velikokrat lahko do pravilnega rezultata pridemo preko različne uporabe operatorjev.

3.2.1 Enostavni relacijski operatorji

Pomen operatorjev =, < in > se ne razlikuje od tega, kar smo navajeni. Njihovo uporabo si bomo ogledali kar preko zgledov spodaj.

Če nas na primer zanimajo le tiste vrstice, kjer ima stolpec določeno vrednost, bi napisali

```
SELECT seznam_stolpcev FROM tabela WHERE stolpec = vrednost
```

Primer (Tabela 4: drzave)

Denimo, da nas zanimata ime in priimek dijaka, ki ima identifikacijsko številko enako 2504. V drugem primeru pa želimo pridobiti podatke o tistih dijakih, ki imajo identifikacijske številke manjše ali enake 2505.

```
SELECT ime, priimek FROM dijaki WHERE id_dijaka = 2504
```

ime	priimek
Marko	Sever

```
SELECT ime, priimek FROM dijaki WHERE id_dijaka <= 2505
```

ime	priimek
Rok	Bizjak
Miha	Petek
Jaka	Zajc
Marko	Sever
Boris	Kuhar
Ksenja	Jerman

V zgornjem primeru smo kot vrednost uporabili število, zato nismo uporabili narekovajev. Če pa bi kot vrednost zapisali niz, potem bi okrog niza izpisali še enojne narekovaje.

Zgled takšne uporabe vrednosti je

```
SELECT ime, priimek FROM dijaki WHERE ime = 'Jaka'
```

ime	priimek
Jaka	Zajc

Obstajajo še operatorji <=, >=, != katerih pomen je prav tako očiten. Omenimo le, da je vrstni red znakov v teh sestavljenih operatorjih predpisan (ne moremo napisati npr. =>) in da vmes praviloma ne sme biti presledka (torej zapis < = ni pravilen). Tu imamo primer, kako različni RDBMS-ji različno tolmačijo sintakso jezika SQL. Tako prej povedano (»prepoved« presledkov, vrstni red znakov) velja za sistem PostgreSQL in tudi za MySQL. Če torej pri teh operatorjih uporabimo presledek, omenjena sistema javita napako. Po drugi strani pa sistem Oracle presledek ignorira in stavek izvede.

Pri primerjanju nizov operatorji ločijo med velikimi in malimi črkami. Če bi kot vrednost enkrat zapisali malo črko j, drugič pa veliko črko J, bi dobili različne rezultate (`ime = 'Jaka'` ni isto kot `ime = 'jaka'`).

Uporabljene stavke SELECT lahko najdete na spletni strani v kategoriji `relacijski operatorji v` 17., 84., in 85. nalogi.

Za vajo lahko rešite še naloge 5, 60 in 92.

3.2.2 Primerjanje različnih podatkovnih tipov

Na primeru tabele `narocila` si bomo ogledali, kaj se zgodi, če stolpec primerjamo z različnimi podatkovnimi tipi.

Primer (Tabela 9: narocila)

Naredimo poizvedbo, ki vrne vse podatke tistih naročil, ki imajo vrednost 1020.50.

Pravilen stavek SELECT bi bil naslednji:

```
SELECT * FROM narocila WHERE vrednost_narocila = 1020.50
```

V rezultatu dobimo

id_narocila	stevilka_narocila	stevilka_narocnika	stevilo_artiklov	vrednost_narocila
1	50121	1	5	1020.50

Tokrat bomo število 1020.50 zapisali kot niz

```
SELECT * FROM narocila WHERE vrednost_narocila = '1020.50'
```

Stolpec `vrednost_narocila` je podatkovnega tipa numeric (glej razdelek 2), primerjamo pa ga z nizom. Sistem zna vseeno primerjati pravilno, tako da dobimo enak rezultat kot zgoraj.

Primer (Tabela 9: narocila)

Naredimo poizvedbo, ki vrne vse podatke naročila številka 50122.

Stavki SELECT, ki vrnejo pravilno rešitev:

```
SELECT * FROM narocila WHERE stevilka_narocila = 50122
```

```
SELECT * FROM narocila WHERE stevilka_narocila = '50122'
```

```
SELECT * FROM narocila WHERE stevilka_narocila = 50122.0
```

id_narocila	stevilka_narocila	stevilka_narocnika	stevilo_artiklov	vrednost_narocila
2	50122	1	2	955.33

Če pa bi uporabili naslednji stavek

```
SELECT * FROM narocila WHERE stevilka_narocila = '50122.0'
```

sistem vhodne vrednosti ne more pravilno pretvoriti v ciljni podatkovni tip, zato vrne napako.

V našem primeru je `stevilka_narocila` tipa celo število (Integer), `'50122.0'` pa je tipa niz (String). Sistem poizkuša niz pretvoriti v število, vendar neuspešno.

Cela števila, zapisana v nizih, zna sistem pretvarjati v števila, pri decimalnih številih pa vrne napako.

Primer (Tabela 4: drzave)

Iz tabele bi radi izpisali tiste naročnike, ki ne prihajajo iz Ljubljane. Uporabili bomo operator != (`mesto != 'Ljubljana'`).

```
SELECT ime, mesto FROM narocnik WHERE mesto != 'Ljubljana'
```

ime	mesto
Mojca	Maribor
Janez	Trbovlje
Janez	Trbovlje

V rezultatu nismo dobili vrstice, kjer v stolpcu `mesto` ni podatka. Kako bi izpisali tudi to vrstico, si bomo ogledali v razdelku 3.2.4.4.

Kot smo omenili že v uvodu in opazili v prejšnjem zgledu, včasih v tabelah določene vrednosti v posameznem stolpcu nimamo. Takrat je na tem mestu v tabeli zapisana vrednost NULL. Povejmo pa, da vrednosti stolpca ne moremo primerjati z **NULL**. Denimo, da poskusimo z

```
SELECT ime, mesto FROM narocnik WHERE mesto = NULL
```

V tem primeru ne dobimo pravilnega rezultata. Primerjanje z NULL je posebnost, ki si jo bomo podrobneje ogledali v razdelku 3.2.4.4.

Na spletni strani lahko najdete zgornje stavke v kategoriji `relacijski operatorji` pri 18., 86. in 87. nalogi.

Za utrditev te oblike stavka SELECT so na voljo naslednje naloge: 59, 93, 129.

3.2.3 Logični operatorji

Za združevanje pogojev in zanikanje pogoja uporabljamo logične operatorje **AND**, **OR** in **NOT**.

3.2.3.1 AND

Logični operator **AND** poveže dva pogoja. Pogoj je izpolnjen, če sta izpolnjena oba pogoja, ki ju veže.

```
SELECT stolpec FROM tabela WHERE pogoj1 AND pogoj2
```

Primer (Tabela 8: narocnik)

Radi bi našli podatke o naročnikih s priimkom Jazbec, ki prihajajo iz Ljubljane.

Izpolnjena bosta morala biti oba pogoja:

- `priimek = 'Jazbec'`
- `mesto = 'Ljubljana'`

Pogoja povežemo z operatorjem **AND** in zapišemo v stavku SELECT

```
SELECT ime, priimek, id_narocnik, mesto FROM narocnik
WHERE mesto = 'Ljubljana' AND priimek = 'Jazbec'
```

Našli smo naročnika, ki ustreza pogojema. Izpišemo njegove podatke

ime	priimek	id_narocnik	mesto
Miha	Jazbec	2	Ljubljana

Zgornji primer si lahko na spletni strani ogledate v kategoriji `logični operatorji` pri nalogi 24.

Če želiti preveriti svoje znanje uporabe operatorja AND, lahko rešite 9., 94., in 95. nalogo.

3.2.3.2 OR

Tudi operator **OR** poveže dva pogoja. Tu ni potrebno, da sta oba pogoja izpolnjena. Dovolj je že, da je izpolnjen eden od pogojev.

```
SELECT stolpec FROM tabela WHERE pogoj1 OR pogoj2
```

Primer (Tabela 8: narocnik)

Radi bi našli vse naročnike v tabeli, ki jim je ime Janez ali pa je njihov priimek Kolar.

Veljati mora vsaj eden od pogojev. Prvi pogoj je ta, da je ime enako Janez (`ime = 'Janez'`). Drugi pogoj pa pravi, da naj bo priimek enak Kolar (`priimek = 'Kolar'`).

Pogoja povežemo z operatorjem **OR** in dobimo naslednji stavek

```
SELECT ime, priimek FROM narocnik
WHERE ime = 'Janez' OR priimek = 'Kolar'
```

Kot rezultat dobimo dva naročnika

ime	priimek
Ana	Kolar
Janez	Novak

Zgornji primer si lahko na spletni strani ogledate v kategoriji `logični operatorji pri` 25. nalogi.

Svoje znanje uporabe operatorja OR lahko preverite pri 58 in 96. nalogi.

3.2.3.3 NOT

Z logičnim operatorjem **NOT** zanikamo pogoj. Kot rezultat dobimo vrstice, za katere naš pogoj ne velja.

```
SELECT stolpec FROM tabela WHERE NOT pogoj
```

Primer (Tabela 8: narocnik)

Iz tabele želimo dobiti imena vseh naročnikov, ki niso Janezi.

```
SELECT ime FROM narocnik
WHERE NOT ime = 'Janez'
```

V rezultatu dobimo zapisana 4 imena

ime
Miha
Anja
Mojca
Ana

Zgornji stavek SELECT najdete na spletni strani v kategoriji `logični operatorji` pri 26. nalogi.

Če želite svoje znanje utrditi, rešite še naloge 27, 97 in 98.

3.2.4 Posebni relacijski operatorji

Operatoriji **BETWEEN, IN, LIKE, IS NULL** so posebnost jezika SQL. Njihovo razlago in primere si bomo podrobneje ogledali v naslednji razdelkih.

3.2.4.1 BETWEEN

Kadar želimo preveriti, ali so podatki med dvema mejama, uporabimo operator **BETWEEN**.

```
SELECT seznam_stolpcev FROM tabela
WHERE stolpec BETWEEN vrednost1 AND vrednost2
```

Primer (Tabela 1: dijaki):

Želimo narediti poizvedbo, ki nam vrne imena in priimke dijakov, ki imajo identifikacijsko številko med 2501 in 2506.

```
SELECT ime, priimek FROM dijaki
WHERE id_dijaka BETWEEN 2501 AND 2506
```

ime	priimek
Miha	Petek
Ksenja	Jerman
Jaka	Zajc
Marko	Sever
Boris	Kuhar
Mojca	Jerman

Opozoriti velja, da operator **BETWEEN** ne deluje v vseh sistemih enako. V nekaterih sistemih bo operator pri primerjanju upošteval obe meji in vse vrednosti med njima. V drugih pa bomo kot rezultat dobili le vrednosti strogo med mejama. Pri sistemu PostgreSQL, ki ga uporabljamo pri naši bazi na spletni strani, se pri primerjavi upoštevata obe meji in vse vrednosti med njima.

V kategoriji `relacijski operatorji` pri nalogi 19 najdete zgornji primer uporabe operatorja BETWEEN.

Za utrditev znanja poskusite z nalogami 6, 99 in 100.

3.2.4.2 IN

Včasih želimo, da bi pri pogoju povedali, da lahko določen stolpec zavzame nekaj različnih vrednosti. Sicer bi takrat pogosto lahko uporabili več pogojev primerjanja s posameznimi vrednostmi, ki bi jih združili z AND, vendar to ni pregledno. Poleg tega bomo pri sestavljenih stavkih (razdelek 3.2.5) videli, da včasih posameznih vrednosti sploh ne poznamo vnaprej. Takrat uporabimo operator **IN.**

```
SELECT seznam_stolpcev FROM tabela
WHERE stolpec IN (vrednost1, vrednost2, vrednost3)
```

Torej, če ima vrstica v stolpcu stolpec vrednost enako vrednostim `vrednost1`, `vrednost2` ali `vrednost3`, jo stavek SELECT upošteva pri sestavljanju rezultata.

Primer (Tabela 4: drzave)

Radi bi naredili poizvedbo, ki vrne imena in priimke tistih dijakov, ki so v razredu z identifikacijsko številko 3oziroma 4. Zato napišemo

```
SELECT ime, priimek FROM dijaki
WHERE id_razreda IN (3,4)
```

in dobimo samo tiste dijake, ki so v razredih, ki imata identifikacijsko število 3 ali 4.

Enake rezultate bi dobili z uporabo operatorja OR.

```
SELECT ime, priimek FROM dijaki
WHERE id_razreda = 3 OR id_razreda = 4
```

V obeh primerih dobimo enak rezultat

ime	priimek
Mojca	Jerman
Renata	Rupnik
Ksenja	Jerman

Zgornji primer si lahko na spletni strani ogledate v kategoriji `relacijski operatorji` pri 10. nalogi.

Svoje znanje uporabe operatorja IN lahko preverite pri nalogah 49 in 70.

3.2.4.3 LIKE

Operator **LIKE** uporabljamo takrat, kadar želimo, da se vrednost v stolpcu ujema z določenim vzorcem. Pri tem z znakom % označimo manjkajoči del vzorca, z _ pa manjkajoči znak.

```
SELECT seznam_stolpcev FROM tabela WHERE stolpec2 LIKE vzorec
```

Če želimo naprimer izpisati priimke, ki se začenjajo s črko P, bomo uporabili **LIKE** (`'P%'`). Torej desno od P-ja lahko stoji katerokoli zaporedje znakov poljubne dolžine (Perk, Pan, Pavšič) tudi prazno zaporedje (vzorcu ustreza torej tudi P).

S podčrtajem v vzorcu (npr. `LIKE('_erk'))`nadomestimo natanko eno črko. Vzorcu ustrezajo nizi, kjer je namesto podčrtaja lahko katerikoli znak (Perk, Gerk, Ferk). Priimek `'erk'` ne ustreza vzorcu, saj znak _ pove, da tam mora biti obvezno nek znak.

Primer (Tabela 4: drzave)

Radi bi naredili poizvedbo, s katero bomo dobili vsa imena dijakov z začetnico M. Uporabimo torej vzorec, ki ima kot prvi znak črko M. Manjkajoči del vzorca je desno od M-ja.

```
SELECT ime FROM dijaki WHERE ime LIKE ('M%')
```

ime
Miha
Marko
Mojca

Izpisali bi radi imena in priimke dijakov, katerih priimki se zaključijo s `k`. Manjkajoči del vzorca bo tokrat levo od `k`-ja.

```
SELECT ime, priimek FROM dijaki WHERE priimek LIKE ('%k')
```

ime	priimek
Rok	Bizjak
Miha	Petek
Renata	Rupnik
Rok	Breznik

Včasih želimo, da je naš vzorec kjerkoli v nizu. Tako vzorec postavimo med znaka %. Želimo narediti poizvedbo po tistih imenih dijakov, ki v imenu vsebujejo podniz `'en'`.

```
SELECT ime FROM dijaki WHERE priimek LIKE ('%en%')
```

ime
Renata
Ksenja

Tako pri Renati kot tudi pri Ksenji lahko v imenu najdemo vzorec `'en'`.

Uporabo obeh nadomestnih znakov (% in _) hkrati si bomo ogledali v naslednjem primeru.

Primer (Tabela 8: narocnik)

Denimo, da želimo najti tista imena naročnikov, ki imajo na drugem mestu črko `n`. Ker začetnice ne poznamo, bomo na njenem mestu uporabili podčrtaj. Desno od `n` pa imamo lahko poljubno zaporedje znakov in bomo zato uporabili %.

Stavek SELECT bo sledeč

```
SELECT ime FROM narocnik WHERE ime LIKE ('_n%')
```

V tabeli sta dve imeni, ki ustrezata temu vzorcu.

ime
Anja
Ana

Zgornje stavke SELECT najdete v kategoriji `relacijski operatorji` pri 8., 20., 21. in 22. nalogi.

Svoje znanje uporabe operatorja LIKE lahko preverite pri nalogi 72, 73 in 101.

3.2.4.4 IS NULL

V tabeli imamo občasno tudi prazne celice. Pravimo, da imajo te celice vrednost **NULL**. Če želimo poiskati te celice, uporabimo operator **IS NULL**.

```
SELECT seznam_stolpcev FROM tabela WHERE stolpec IS NULL
```

Primer (Tabela 8: narocnik)

Zanima nas, če imamo v tabeli naslove vseh naročnikov. Zato bomo izpisali imena naročnikov, ki imajo v stolpcu `naslov` vrednost NULL. Uporabili bomo operator IS NULL.

```
SELECT ime FROM narocnik WHERE naslov IS NULL
```

ime	naslov
Anja	

Ugotovili smo, da nam manjka naslov naročnice Anje.

V povezavi z relacijskimi operatorji velikokrat uporabljamo operator **NOT** (NOT IN, NOT LIKE, NOT BETWEEN). Uporabimo ga, kadar želimo 'obrniti' pogoj, kot ga določajo omenjeni operatorji.

V zgornjem primeru smo izpisali naročnike, čigar naslovov ne poznamo. Sedaj pa izpišimo imena in priimke naročnikov, katerih naslove poznamo. To bomo naredili s pomočjo operatorja **IS NOT NULL**.

```
SELECT ime, priimek FROM narocnik WHERE naslov IS NOT NULL
```

Kot rezultat dobimo naslednje naročnike

ime	priimek
Miha	Jazbec
Mojca	Smodic
Ana	Kolar
Janez	Novak
Janez	Vesel

Primer (Tabela 3: razredi)

V razdelku 3.2.2.2 smo našli dijake, ki obiskujejo razreda z identifikacijskim številom 3 ali 4. Tokrat pa nas zanimajo dijaki, ki ne obiskujejo teh razredov.

```
SELECT ime, priimek FROM dijaki
WHERE id_razreda NOT IN (3,4)
```

Kot rezultat dobimo naslednje dijake:

ime	priimek
Rok	Bizjak
Miha	Petek
Jaka	Zajc
Marko	Sever
Boris	Kuhar
Rok	Breznik

Oglejmo si še primer, pri katerem bomo izpisali dijake, ki nimajo identifikacijskih številk med `2501` in `2506`. Stavek bo podoben kot pri razdelku 3.2.2.1, le da bomo pred **BETWEEN** dodali besedo **NOT**.

```
SELECT ime, priimek FROM dijaki
WHERE id_dijaka NOT BETWEEN 2501 and 2506
```

Naslednji dijaki imajo identifikacijske številke, ki niso med 2501 in 2506

ime	priimek
Rok	Bizjak
Renata	Rupnik
Rok	Breznik

Stavka SELECT z uporabo operatorja NOT si lahko na spletni strani ogledate v kategoriji `relacijski operatorji` pri 23. in 28. nalogi.

Za utrditev so na voljo naloge 61, 74, in 102.

3.2.5 Sestavljene poizvedbe

Za zaključek razdelka o ukazu **WHERE** si oglejmo nekaj sestavljenih poizvedb. Pri njih bomo uporabili več operatorjev hkrati.

Primer (Tabela 1: dijaki)

Radi bi našli dijake, ki imajo identifikacijske številke manjše od `2505` in obiskujejo razrede z identifikacijskim številom `3` ali `4`.

Zapisati bo potrebno več pogojev:

- 1. pogoj `id_dijaka < 2505` identifikacijska številka dijaka je manjša od 2505
- 2. pogoj `id_razreda = 3` identifikacijska številka razreda je enaka 3
- 3. pogoj `id_razreda = 4` identifikacijska številka razreda je enaka 4

Prvi pogoj mora veljati v vsakem primeru. Pri drugem in tretjem pogoju pa velja, da mora biti pravilen vsaj eden izmed njiju.

Pogoje povežemo na naslednji način

```
SELECT ime, priimek FROM dijaki
WHERE id_dijaka < 2505 AND (id_razreda = 3 OR id_razreda = 4)
```

Kot vedno se problemov lahko lotimo na različne načine, z uporabo različnih operatorjev in funkcij.

Zgornjega primera bi se lahko lotili tudi z uporabo operatorja **IN** (glej razdelek 3.2.2.2).

Stavek bi bil sedaj tak

```
SELECT ime, priimek FROM dijaki
WHERE id_dijaka < 2505 AND id_razreda IN (3,4)
```

Rezultat je v obeh primerih enak. Tak dijak (oziroma dijakinja) je v obstoječi bazi le en:

ime	priimek
Ksenja	Jerman

Poglejmo si, kaj bi se zgodilo, če pri prvi obliki ne bi uporabili oklepajev.

```
SELECT ime, priimek FROM dijaki
WHERE id_dijaka < 2505 AND id_razreda = 3 OR id_razreda = 4
```

Če ne postavimo oklepajev, ima operator AND prednost pred operatorjem OR. Torej bi v tabeli najprej našli dijake z identifikacijsko številko manjšo od `2505` in obiskujejo razred z identifikacijsko številko `3`, nato pa bi tem dijakom dodali še dijake iz razreda z identifikacijsko številko `4`.

ime	priimek
Ksenja	Jerman
Mojca	Jerman
Renata	Rupnik

Primer (Tabela 8: narocnik)

V razdelku 3.2.2 smo si ogledali primer poizvedbe po naročnikih, ki ne prihajajo iz Ljubljane. Sedaj pa želimo uvrstiti med tiste, ki niso iz Ljubljane, tudi tiste, katerih mest ne poznamo.

```
SELECT ime, mesto FROM narocnik
WHERE mesto != 'Ljubljana' OR mesto IS NULL
```

Naročniki, ki ustrezajo pogojem so:

ime	mesto
Anja	
Mojca	Maribor
Janez	Trbovlje
Janez	Trbovlje

Zgornji primer najdete v kategoriji `logični operatorji` pri 29. nalogi.

Svoje znanje lahko utrdite z nalogami 7, 103 in 104.

3.3 Združevalne funkcije

Združevalne funkcije so funkcije, s katerimi združimo različne vrednosti posameznih stolpcev v en sam podatek. Z združevalnimi funkcijami lahko na enostaven način pridemo do povprečne, maksimalne, minimalne vrednosti v stolpcu, seštejemo vrednosti ali pa dobimo število (različnih ali vseh) neničelnih vrednostiv stolpcu.

- **MIN (stolpec)** – minimalna vrednost
- **SUM (stolpec)** – vsota vrednosti
- **AVG (stolpec)** – povprečna vrednost
- **COUNT (stolpec)** – število vrstic z neničelnim podatkom v tem stolpcu

Pri določanju vrednosti se upoštevajo le neničelne vrednosti v stolpcih. Funkcije lahko uporabljamo le v prvem delu stavka SELECT– kjer navajajo vrednosti, ki jih želimo dobiti. Rezultatov, ki jih vrnejo te funkcije, ne moremo neposredno uporabljati v pogojih v WHERE, razen če uporabimo gnezdene stavke. Kako lahko rezultate teh funkcij v posebnih primerih uporabimo v pogojih, si bomo ogledali v razdelku 3.7. Sedaj si podrobneje oglejmo posamezno funkcijo.

3.3.1 Funkcija MAX

Funkcija **MAX** izračuna maksimalno vrednost v stolpcu.

```
SELECT MAX(stolpec) FROM tabela
```

Primer (Tabela 9: narocila)

Zanima nas, kolikšna je največja vrednost, ki jo je doseglo posamezno naročilo.

```
SELECT MAX(vrednost_narocila) FROM narocila
```

max
7100.60

Zgornji stavek SELECT najdete na spletni strani v kategoriji `združevalne funkcije` pri nalogi 33.

Svoje znanje uporabe funkcije MAX lahko preverite pri 53., 105. in 106. nalogi.

3.3.2 Funkcija MIN

S funkcijo **MIN** poiščemo minimalno vrednost v stolpcu.

```
SELECT MIN(stolpec) FROM tabela
```

Primer (Tabela 9: narocila)

Zanima nas, kolikšna je najmanjša vrednost naročila.

```
SELECT MIN(vrednost_narocila) FROM narocila
```

min
382.55

Zgornji primer si lahko na spletni strani ogledate v kategoriji `združevalne funkcije` pri 34. nalogi.

Svoje znanje uporabe funkcije MIN lahko preverite pri 54., 107. in 108. nalogi.

3.3.3 Funkcija SUM

Če bi radi izračunali vsoto vseh vrednosti v stolpcu, uporabimo funkcijo **SUM**.

```
SELECT SUM(stolpec) FROM tabela
```

Primer (Tabela 4: drzave)

Naredimo poizvedbo, ki vrne skupno število prebivalcev vseh držav.

```
SELECT SUM(stevilo_prebivalcev) FROM drzave
```

Rezultat je sledeč

sum
600905866

Stavek SELECT, ki smo ga uporabili zgoraj, najdete v kategoriji `združevalne funkcije` pri 30. nalogi.

Svoje znanje uporabe funkcije SUM lahko preverite pri 62., 109. in 110. nalogi.

3.3.4 Funkcija AVG

Funkcija **AVG** nam izračuna povprečno vrednost v stolpcu.

```
SELECT AVG(stolpec) FROM tabela
```

Primer (Tabela 4: drzave)

Zanima nas povprečno število prebivalcev držav, ki so nastopala na evropskih prvenstvih.

```
SELECT AVG(stevilo_prebivalcev) FROM drzave
```

Ugotovimo, da je povprečno število prebivalcev držav iz tabele malo nad 27 milijoni.

avg
27313903

Povprečno vrednost bi lahko izračunali tudi s pomočjo SUM in COUNT funkcij, vendar je uporaba **AVG** preprostejša.

```
SELECT SUM(stevilo_prebivalcev) / COUNT(stevilo_prebivalcev) FROM drzave
```

Dobimo enak rezultat kot zgoraj, le stolpec je drugače poimenovan.

column
27313903

Stavek SELECT z uporabo funkcije AVG si lahko ogledate v kategoriji `združevalne funkcije` pri 31. nalogi.

Na spletni strani so za utrditev te oblike stavka SELECT na voljo naslednje naloge: 55, 111, 112, 114.

3.3.5 Funkcija COUNT

Funkcija **COUNT** vrne število vrstic z neničelno vrednostjo v stolpcu.

```
SELECT COUNT(stolpec) FROM tabela
```

Primer (Tabela 1: dijaki)

Radi bi ugotovili, koliko dijakov je zapisanih v tabeli. Zato uporabimo kar COUNT(*).

```
SELECT COUNT(*) FROM dijaki
```

Rezultat

count
10

Uporabimo lahko tudi drugi način, če vemo, da ima vsak dijak svojo identifikacijsko številko.

```
SELECT COUNT(id_dijaka) FROM dijaki
```

Dobimo enak rezultat kot zgoraj. Rezultat bi se od prejšnjega razlikoval le, če bi obstajal dijak, ki v tabeli ne bi imel vpisane identifikacijske številke.

Razlika v ugotavljanju števila vrstic tabele s COUNT(*) in COUNT(stolpec) je torej v tem, da pri prvi obliki zagotovo vemo, da bomo dobili število vrstic v tabeli. Pri drugi obliki pa je to število lahko manjše, če v navedenem stolpcu kakšna vrednost manjka (je NULL).

Razlikovati moramo med uporabo DISTINCT pri SELECT in pri COUNT. Primerjajmo rezultate poizvedb nekaterih na videz podobnih poizvedb. Če uporabimo COUNT (stolpec) dobimo število podatkov v stolpcu, pri uporabi COUNT(DISTINCT stolpec) pa dobimo število unikatnih podatkov v stolpcu. DISTINCT takoj za SELECT pa deluje šele na rezultatu in poskrbi, da se izločijo podvojene vrstice tabele, ki je rezultat poizvedbe.

Primer (Tabela 8: narocnik)

Ukaz

```
SELECT COUNT(mesto) FROM narocnik
```

nam pove, da je v tabeli `narocnik` zapisanih 5 mest.

count
5

Zanima pa nas, koliko mest v tabeli je različnih. Do rezultata bomo prišli s pomočjo DISTINCT.

```
SELECT COUNT(DISTINCT mesto) FROM narocnik
```

Rezultat

count
3

V rezultatu dobimo število vrstic, ki imajo v stolpcu `mesto` podatek. COUNT ne upošteva vrstic z vrednostjo NULL v stolpcu `mesto`. Kaj pa, če COUNT in DISTINCT zamenjata mesti?

```
SELECT DISTINCT COUNT (mesto) FROM narocnik
```

count
5

Glede na vrstni red operacije najprej izvede COUNT, šele nato DISTINCT. S COUNT dobimo eno vrstico, DISTINCT pa vrne unikatne vrstice. Ker smo pri COUNT dobili samo eno vrstico, seveda DISTINCT »nima dela« in je v rezultatu ta edina vrstica.

```
SELECT DISTINCT COUNT (DISTINCT mesto) FROM narocnik
```

count
3

Najprej se izvede DISTINCT znotraj oklepajev, nato COUNT in nato še tisti DISTINCT takoj za SELECT. Z DISTINCT `mesto` dobimo tabelo, v kateri so sama unikatna mesta. Na tej tabeli se nato izvede COUNT. Ker je DISTINCT vrnil 4 vrstice v rezultatu, ena od njih je bila NULL, COUNT vrne rezultat 3. Dobimo torej eno vrstico. Na tem rezultatu se izvede DISTINCT, ki nima dela in vrne to edino vrstico.

Primer (Tabela 4: drzave)

Želimo izvedeti povprečno število prebivalcev Avstrije, Slovenije in Jugoslavije.

Iz tabele je razvidno, da imajo države naslednje število prebivalcev

naziv	stevilo_prebivalcev
Avstrija	8192880
Slovenija	2038733
Jugoslavija	

Povprečje dobimo tako, da seštejemo prebivalce vseh držav in delimo s 3.

```
SELECT SUM(stevilo_prebivalcev) / 3 FROM drzave
WHERE naziv IN ('Avstrija', 'Slovenija', 'Jugoslavija')
```

Torej dobimo povprečno število prebivalcev je 3410537.

Sedaj pa uporabimo naslednji stavek SELECT

```
SELECT AVG(stevilo_prebivalcev) FROM drzave
WHERE naziv IN ('Avstrija','Slovenija','Jugoslavija')
```

in dobimo rezultat 5115806.

avg
5115806

Do razlike pride, ker funkcija AVG ni upoštevala vrednosti NULL pri Jugoslaviji.

Stavke SELECT, ki smo jih uporabili pri zgornjih primerih, najdete v kategoriji `združevalne funkcije` pri nalogah: 32., 36., 113. in 114.

Uporabo funkcije COUNT lahko utrdite pri 56. in 57. nalogi.

3.4 Poizvedbe s podpoizvedbami

Kot določeno vrednost v izrazih lahko uporabimo tudi rezultate stavka SELECT. Naredimo poizvedbo s podpoizvedbo. Za podpoizvedbo mora veljati, da vrne le en stolpec z eno vrednostjo, razen kadar njen rezultat kombiniramo z operatorjem IN.

Primer (Tabela 4: drzave):

V razdelku 3.3.4 smo že dobili povprečno število prebivalcev držav, udeleženk rokometnih prvenstev, ki je 27313903. Recimo, da nas zanimajo imena držav, ki imajo število prebivalcev večje od povprečja.

Povprečje (avg = 27313903) uporabimo v spodnjem stavku

```
SELECT naziv FROM drzave
WHERE stevilo_prebivalcev > 27313903
```

naziv
Nemčija
Španija
Francija
Italija
Poljska
Rusija
Ukrajina

Do rezultata smo torej prišli tako, da smo uporabili dva stavka SELECT. S prvim smo prišli do povprečja, v drugem pa smo potem to dobljeno povprečje uporabili. Lahko pa uporabimo gnezdene stavke, tako da naredi poizvedbo v podpoizvedbi.

Stavka združimo v skupni stavek in dobimo

```
SELECT naziv FROM drzave
WHERE stevilo_prebivalcev > (SELECT AVG(stevilo_prebivalcev)
      FROM drzave)
```

Kot vrednost v izrazu smo uporabili rezultat stavka SELECT. Sistem najprej ustrezno izvede podpoizvedbo in njen rezultat vključi v poizvedbo.

Dobimo enak rezultat kot prej.

Primer (Tabela 7: rezultati_prvenstev)

Zanima nas, katerega leta je bila zmagovalka prvenstva Francija in takrat Hrvaška ni dosegla vsaj 3. mesta.

V podpoizvedbi bomo poiskali vsa tista leta, ko Hrvaška ni osvojila 2. ali 3. mesta. Če vemo, da je kratica Hrvaške `'HR'`, bomo v ta namen napisali

```
SELECT leto FROM rezultati_prvenstev
WHERE kratica_drzave = 'HR' AND uvrstitev NOT IN(2,3)
```

Ta stavek vrne tabelo z enim stolpcem (vsa tista leta, ko Hrvaška ni bila niti druga in niti tretja). Rezultat tega stavka lahko uporabimo skupaj z operatorjem IN (z NOT IN). Sedaj zapišemo poizvedbo

```
SELECT leto FROM rezultati_prvenstev
WHERE kratica_drzave = 'F' AND uvrstitev = 1 AND
leto NOT IN(SELECT leto FROM rezultati_prvenstev
      WHERE kratica_drzave = 'HR' AND uvrstitev NOT IN(2,3))
```

In v rezultatu dobimo

leto
2006

V zgornjem primeru smo uporabili dejstvo, da vemo, da je kratica Francije niz `'F'` in kratica Hrvaške niz `'HR'`. V primeru, da kratice za Francijo ne poznamo, bomo v zgornjo poizvedbo vpeljali še eno podpoizvedbo.

Kratico države bomo dobili iz tabele države, kjer so zapisani nazivi držav in njihove kratice. Do kratice Francije bomo prišli z naslednjo poizvedbo

```
SELECT kratica FROM drzave WHERE naziv = 'Francija'
```

To poizvedbo vstavimo v prejšno poizvedbo namesto niza `'F'` in dobimo

```
SELECT leto FROM rezultati_prvenstev
WHERE kratica_drzave=(SELECT kratica FROM drzave
      WHERE naziv = 'Francija')
AND uvrstitev = 1
AND leto NOT IN (SELECT leto FROM rezultati_prvenstev
      WHERE kratica_drzave = 'HR' AND uvrstitev NOT IN(2,3))
```

Dobimo enak rezultat kot zgoraj. In če bi »pozabili« še kratico za Hrvaško, bi ustrezna poizvedba bila

```
SELECT leto FROM rezultati_prvenstev
WHERE kratica_drzave = (SELECT kratica FROM drzave
      WHERE naziv = 'Francija')
AND uvrstitev = 1
AND leto NOT IN (SELECT leto FROM rezultati_prvenstev
      WHERE kratica_drzave = (SELECT kratica FROM drzave
            WHERE naziv = 'Hrvaška')
      AND uvrstitev NOT IN(2,3))
```

Dobili smo kar zapleteno poizvedbo. Običajno takrat, kadar v poizvedbi potrebujemo podatke iz več tabel, uporabljamo združevanje tabel. Tak način si bomo ogledali v razdelku 3.8.

Zgornje stavke SELECT si lahko na spletni strani ogledate v kategoriji `podpoizvedbe` pri naslednjih nalogah: 35 , 116, 117 in 118.

Če želiti preveriti znanje uporabe podpoizvedb, rešite še naloge 132, 133 in 134.

3.5 ORDER BY

Kako bodo razvrščene vrstice v tabeli rezultatov, je odvisno od sistema. Če pa želimo na vrstni red vplivati, na koncu stavka SELECT uporabimo **ORDER BY**. Za **ORDER BY** zapišemo stolpec po katerem želimo razvrstiti podatke. Če želimo naraščajoči vrstni red, za imenom stolpca uporabimo ukaz **ASC**. Za padajoči vrstni red pa uporabimo ukaz **DESC**.

Primer (Tabela 4: drzave)

Želimo narediti poizvedbo, ki bo izpisala imena države in število njihovih prebivalcev naraščajoče po številu prebivalcev.

```
SELECT naziv, stevilo_prebivalcev FROM drzave
ORDER BY stevilo_prebivalcev ASC
```

naziv	stevilo_prebivalcev
Islandija	299388
Slovenija	2038733
Hrvaška	4494749
⋮	⋮
Francija	60876136
Nemčija	82422299
Rusija	142893540
Jugoslavija	

V zgornjem zgledu lahko vidimo, da pri uporabi ASC vrstice, ki imajo v ustreznem stolpcu vrednosti NULL, dobimo na koncu rezultata. Pri DESC pa je ravno obratno, vrstice z vrednostimi NULL dobimo na začetku rezultata.

Sedaj pa bi radi obratni izpis: od največjega proti najmanjšemu številu prebivalcev.

```
SELECT naziv, stevilo_prebivalcev FROM drzave
ORDER BY stevilo_prebivalcev DESC
```

naziv	stevilo_prebivalcev
Jugoslavija	
Rusija	142893540
Nemčija	82422299
Francija	60876136
⋮	⋮
Hrvaška	4494749
Slovenija	2038733
Islandija	299388

Naslednji stavek nam bo uredil države padajoče po nazivu (imenu).

```
SELECT naziv FROM drzave
ORDER BY naziv DESC
```

naziv
Ukrajina
Švica
Švedska
Srbija
⋮
Češka
Belorusija
Avstrija

Omenimo še, da takrat, kadar ne uporabimo ne ASC in ne DESC, ne moremo vedeti, v kakšnem vrstnem redu bodo naši rezultati. Vrstni red bo vsak sistem načeloma določil po svoje – nekateri bodo uporabili naraščajočega, drugi padajočega.

Včasih bi želeli podatke urediti po več ključih. Denimo, da bi dijake radi razvrstili po razredih, znotraj posameznega razreda pa po priimku po obratnem abecednem vrstnem redu.

```
SELECT priimek, id_razreda FROM dijaki
ORDER BY id_razreda ASC, priimek DESC
```

priimek	id_razreda
Zajc	1
Petek	1
Bizjak	1
Jerman	3
Rupnik	4
Jerman	4
Kuhar	5
Breznik	5
Sever	6

Za razvrščanje lahko uporabimo poljubne izraze in ne samo stolpce. Poljubni izrazi so lahko funkcije, podpoizvedbe (glej razdelek 3.4) in podobno.

Primer (Tabela 1: dijaki)

Radi bi naredili poizvedbo, ki vrne priimke dijakov urejene po njihovi dolžini.

```
SELECT priimek FROM dijaki
ORDER BY length(priimek)
```

V rezultatu dobimo

priimek
Zajc
Sever
Kuhar
⋮
Rupnik
Breznik

Primer (Tabela 3: razredi)

Želimo razvrstiti razrede po črkah razredov. Najprej naj bodo izpisani vsi a razredi, nato b razredi in tako dalje.

Ker so podatki v stolpcu `razred` sestavljeni iz številke in črke (npr. 1.a), moramo uporabiti funkcijo `substr()`.

```
SELECT * FROM razredi
ORDER BY substr(razred, 2)
```

Rezultat je sledeč

stevilka_razrednika	povprecna_ocena	razred	stevilka_razreda
1	3.45	1.a	1
5	3.86	3.a	5
3	4.10	2.a	3
4	3.67	2.b	4
2	3.02	1.b	2
6	3.22	3.b	6
4	3.55	3.c	7

Razredi so urejeni po črkah. Sedaj pa želimo, da so znotraj posamezne črke urejeni še po številki.

```
SELECT * FROM razredi
ORDER BY substr(razred, 2), razred
```

In dobimo

stevilka_razrednika	povprecna_ocena	razred	stevilka_razreda
1	3.45	1.a	1
3	4.10	2.a	3
5	3.86	3.a	5
2	3.02	1.b	2
4	3.67	2.b	4
6	3.22	3.b	6
4	3.55	3.c	7

Zgornje stavke SELECT si lahko na spletni strani ogledate v kategoriji `Uporaba ORDER BY, GROUP BY, HAVING` pri naslednjih nalogah: 37, 38, 39, 119, 120, 121 in 122.

Svoje znanje razvrščanja podatkov lahko preverite pri 63., 64. in 123. nalogi.

3.6 GROUP BY

Z uporabo **GROUP BY** rezultate poizvedbe združimo v skupine. **GROUP BY** torej združi vrstice, ki imajo enak podatek v stolpcu. **GROUP BY** vedno stoji na koncu v stavku SELECT in deluje v kombinaciji z združevalnimi funkcijami. Če namreč uporabimo **GROUP BY,** bomo v rezultatu za vsako skupino dobili eno vrstico. Z združevalno funkcijo pa povemo, kakšna naj bo ta vrstica.

Primer (Tabela 8: narocnik)

Radi bi sestavili poizvedbo, ki vrne vsa mesta in število imen naročnikov, ki prihajajo iz tega mesta.

```
SELECT mesto, COUNT(ime)FROM narocnik
```

Program javi napako. Poglejmo si, kaj bi pravzaprav dobili z zgornjo poizvedbo.

mesto	count
Trbovlje	6
Ljubljana	6
Maribor	6
Ljubljana	6
	6
Trbovlje	6

Pri vsakem mestu iz tabele bi izpisali še število vseh imen v tabeli. V našem primeru nismo želeli takšnega rezultata. Kako bo izgledala pravilna poizvedba z uporabo GROUP BY, si bomo ogledali spodaj.

Podatke bo najprej potrebno združiti. V tabeli si kot tistega po katerem bomo združevali, izberemo stolpec `mesto`. Dobimo 4 skupine podatkov.

id_narocnik	ime	priimek	naslov	mesto	
2	Miha	Jazbec	Stari trg 42	Ljubljana	Skupina 1
4	Ana	Kolar	Opekarna 66	Ljubljana	
1	Janez	Novak	Novi dom 32	Trbovlje	Skupina 2
6	Janez	Vesel	Leskoškova 55	Trbovlje	
3	Mojca	Smodic	Trg revolucije 18	Maribor	Skupina 3
5	Anja				Skupina 4

Združevalna funkcija se uporabi na vsaki skupini posebej. V našem primeru imamo 4 skupine, na katerih bomo izvedli funkcijo COUNT.

```
SELECT COUNT(ime) FROM narocnik
GROUP BY mesto
```

Rezultat

count
1
2
2
1

Ker to ni preveč informativno, bi radi dobili še imena skupin.. Ker vemo, da imajo vse vrstice v posamezni skupini skupni podatek mesta, bo skupino lahko predstavljal ta stolpec

```
SELECT mesto FROM narocnik
GROUP BY mesto
```

In dobimo

mesto
Trbovlje
Ljubljana
Maribor

Vidimo tudi, da imamo v tabeli vrstice, kjer podatka o mestu ni.

Sedaj lahko to dvoje združimo v

```
SELECT COUNT(ime), mesto FROM narocnik
GROUP BY mesto
```

In dobimo

count	mesto
1	
2	Trbovlje
2	Ljubljana
1	Maribor

Iz tega je sedaj jasno, da imena mesta nismo vnesli le pri eni vrstici.

Stavek za vsako skupino izpiše eno vrstico. Tako stavek

```
SELECT ime, mesto FROM narocnik
GROUP BY mesto
```

ni pravilen, ker potrebujemo združevalne funkcije pri stolpcu `ime`. GROUP BY mestu ustvari skupine

ime	mesto	
Miha	Ljubljana	Skupina 1
Ana	Ljubljana	

Janez	Trbovlje	Skupina 2
Janez	Trbovlje	

Mojca	Maribor	Skupina 3

Anja		Skupina 4

Sedaj sistem ne ve, katero ime izpisati pri posamezni skupini (npr. ali pri skupini 1 Miha ali Ana), zato pri imenu potrebujemo združevalno funkcijo.

Primer (Tabela 7: rezultati_prvenstev)

Denimo, da želimo prešteti, kolikokrat je posamezna država osvojila naslov prvaka v rokometu.

- Iz tabele bomo najprej izločili vse zapise, kjer država ni prvak (`uvrstitev = 1`).
- Dobljene vrstice bomo združili v skupine po državah (`GROUP BY kratica_drzave`).
- S `COUNT(uvrstitev)` bomo prešteli število vrstic v posamezni skupini.

In dobimo

```
SELECT kratica_drzave, COUNT(uvrstitev)FROM rezultati_prvenstev
WHERE uvrstitev = 1
GROUP BY kratica_drzave
```

V rezultatu dobimo

kratica_drzave	count
S	4
DK	1
D	1
F	2
RUS	1

Pri izločanju vrstic (uporabi WHERE) bi pogosto radi pri samem pogoju uporabili združevalne funkcije.

Primer (Tabela 9: narocila)

Radi bi ugotovili, kateri naročniki imajo skupno vrednost naročil večjo od 7000.

```
SELECT stevilka_narocnikaFROM narocila
WHERE SUM(vrednost_narocila) > 7000
GROUP BY stevilka_narocnika
```

Program nam javi napako, ker nam RDBMS ne dovoli uporabe združevalnih funkcij v WHERE. Če bi radi dobili samo vrstice, pogojene z združevalnimi funkcijami, moramo uporabiti HAVING. Uporabo HAVING si bomo ogledali v razdelku 3.7.

Zgornja primera si lahko na spletni strani ogledate v kategoriji `Uporaba ORDER BY, GROUP BY, HAVING` pri 124. in 125. nalogi.

Svoje znanje združevanja podatkov lahko preverite pri 40., 65. in 75. nalogi.

3.7 HAVING

Včasih bi radi upoštevali le tiste skupine, katerih združena vrednost zadošča določenemu pogoju. Na ta način bomo rešili problem, s katerim smo zaključili prejšnji razdelek.

Primer (Tabela 8: narocnik)

Denimo da bi radi ugotovili, kateri naročniki imajo skupno vrednost naročil večjo od 7000. Kot vemo že od prej, združevalnih funkcij ne moremo uporabljati pri izločanju vrstic z WHERE.

Če poskusimo z

```
SELECT stevilka_narocnika FROM narocila
WHERE SUM(vrednost_narocila) > 7000
GROUP BY stevilka_narocnika
```

dobimo napako. Pa tudi če bi to šlo, bi bil rezultat napačen, saj WHERE deluje pred GROUP BY. Torej se najprej izločijo ustrezne vrstice, šele potem pa se jih grupira.

Rešitev zgornjega problema nam omogoča uporaba ukaza **HAVING**. Ta omogoča, da najprej združimo podatke v skupini, nato pa postavimo pogoj.

V našem primeru bomo torej najprej razdelili tabelo `narocnik` na podtabele po stevilki naročnika.

id_narocila	stevilka_narocila	stevilka_narocnika	stevilo_artiklov	vrednost_narocila
1	50121	1	5	1020.50
2	50122	1	2	955.33
8	50128	1	10	7100.60
7	50127	2	1	500.68
3	50123	3	7	4680.12
9	50129	3	2	1730.64
10	50129	3	1	382.55
4	50124	4	3	1574.08
6	50126	4	4	5723.40
5	50125		5	3211.99

Potem bomo sešteli vrednosti naročil posameznega naročnika (posamezne skupine).

stevilka_narocnika	vrednost_narocila
1	9076.43
2	500.68
3	6793.31
4	7297.48
	3211.99

Na koncu pa še vrnemo stevilke naročnikov, ki imajo skupno vrednost naročil večjo od 7000.

```
SELECT stevilka_narocnika FROM narocila
GROUP BY stevilka_narocnika
HAVING SUM(vrednost_narocila)>7000
```

Naročnika 1 in 4 sta nakupila za več kot 7000.

stevilka_narocnika
4
1

Primer (Tabela 7: rezultati_prvenstev)

Oglejmo si še kak zgled, kjer s pridom uporabimo ukaz HAVING. Radi bi naredili poizvedbo, ki vrne tiste države, ki so na rokometnih prvenstvih zmagale več kot enkrat.

Tu bomo potrebovali tako izločevanje s pomočjo WHERE kot tudi kasnejšo izbiro ustreznih skupin s HAVING.

- Ker nas zanimajo le države zmagovalke, v WHERE postavimo `uvrstitev = 1.`
- `GROUP BY kratica_drzave` vrstice združi v skupine (v posamezni skupini so vsa tista leta, ko je posamezna država zmagala).
- S `HAVING COUNT(uvrstitev) > 1` pa upoštevamo le skupine, kjer je več kot eno prvenstvo, na katerem je država zmagala.

Stavek bo naslednji

```
SELECT COUNT(uvrstitev), kratica_drzave FROM rezultati_prvenstev
WHERE uvrstitev = 1
GROUP BY kratica_drzave
HAVING COUNT(uvrstitev) > 1
```

V rezultatu dobimo dve državi, ki sta do sedaj zmagali na prvenstvih v rokometu več kot enkrat.

count	kratica_drzave
2	F
4	S

Stavka SELECT, ki smo ju uporabili v tem razdelku, najdete v kategoriji `Uporaba ORDER BY, GROUP BY, HAVING` pri 41. in 42. nalogi.

Znanje uporabe HAVING lahko utrdite pri 138., 139. in 140. nalogi.

3.8 Združevanje tabel

Baze so večinoma sestavljene iz večih tabel. Večkrat želimo združiti podatke iz dveh ali več tabel. Spomnimo se zgleda iz razdelka 3.4, kjer smo uporabili podpoizvedbo za kratico Francije (in za kratico Hrvaške). Tam smo potrebovali tako tabelo `drzave`, ker imamo v njej imena držav, kot tudi tabelo `rezultati_prvenstev`, kjer imamo rezultate prvenstev.

Združevanje pomeni, da dobimo novo tabelo, katere vrstice so sestavljene iz stolpcev prve in druge tabele.

Združitev naredimo tako, da ustvarimo novo tabelo s podatki iz obeh tabel. Pri tem dejansko naredimo kartezični produkt velikosti n x m, kjer vsako vrstico prve tabele združimo z vsako vrstico druge tabele. Pri tem je n število vrstic prve tabele, m pa predstavlja število vrstic drug tabele. Pravimo, da je

takšna združitev implicitna. Potem pa s pomočjo WHERE izločimo le »smiselne vrstice« in v rezultatu vrnemo samo ujemajoče vrstice.

Tabelo `drzave` in tabelo `rezultati_prvenstev`

drzave		
kratica	naziv	stevilo_prebivalcev
A	Avstrija	8192880
D	Nemčija	82422299
SLO	Slovenija	2038733
⋮	⋮	⋮

rezultati_prvenstev			
id	leto	kratica_drzave	uvrstitev
1	2010	F	1
2	2010	HR	2
7	2008	DK	1
⋮	⋮	⋮	⋮

bi lahko združili tako, da bi upoštevali le vrstice, kjer se ujemata podatka v stolpcu `kratica` v tabeli `drzave` in v stolpcu `kratica_drzave` v tabeli `rezultati_prvenstev` (stolpca za združitev sta obarvana z modro barvo).

Ustrezna poizvedba je

```
SELECT * FROM drzave, rezultati_prvenstev
WHERE kratica = kratica_drzave
```

V našem primeru najprej dobimo kartezični produkt 23 x 51 = 1173 vrstic. Nato pa s smiselno postavitvijo pogojev dobimo tabelo z 51 vrsticami.

kratica	naziv	stevilo_prebivalcev	id	leto	kratica_drzave	uvrstitev
RUS	Rusija	142893540	23	2004	RUS	5
F	Francija	60876136	1	2010	F	1
HR	Hrvaška	4494749	2	2010	HR	2
IS	Islandija	299388	3	2010	IS	3
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Primer (Tabela 4: drzave, Tabela 7: rezultati_prvenstev)

Primer iz razdelka 3.4, kjer smo uporabili podpoizvedbo za kratico Francije, bomo tokrat rešili z združevanjem tabel.

```
SELECT leto FROM rezultati_prvenstev, drzave
WHERE kratica_drzave = kratica AND naziv = 'Francija'
AND uvrstitev = 1
AND leto NOT IN (SELECT leto FROM rezultati_prvenstev
      WHERE kratica_drzave = 'HR' AND uvrstitev IN(2,3))
```

Tabeli smo združili s povezavo `kratica_drzave = kratica` in postavili pogoj `naziv = 'Francija'`(v razdelku 3.4 smo namesto tega uporabili podpoizvedbo *`SELECT kratica FROM drzave WHERE naziv = 'Francija'`*).

Lahko se zgodi, da želimo združiti dve tabeli, ki imata kakšen stolpec enako poimenovan. Sistem takrat ne ve, iz katere tabele vzeti stolpec, uporabljen v poizvedbi. Zato moramo takrat v stavku SELECT poleg imena stolpca navesti še ime tabele.

drzave		
kratica	naziv	stevilo_prebivalcev

financni_podatki_drzav		
kratica	bdp	bdp_na_prebivalca

Pred imenom stolpca zapišemo ime tabele, kjer se nahaja stolpec.

- `drzave.kratica` (stolpec iz leve tabele)
- `financni_podatki_drzav.kratica` (stolpec iz desne tabele)

Omenimo še, da bi to obliko (`imeTabele.imeStolpca`) lahko uporabil vedno, pri vsaki poizvedbi in pri vsakem stolpcu.

Oblika združevanja je lahko implicitna ali pa eksplicitna. Implicitni tabeli združujemo tako, da v del FROM navedemo obe tabeli. Pri eksplicitni obliki pa tabeli združimo z uporabo JOIN.

Eksplicitno združevanje je lahko notranje ali zunanje. Pri notranjem združevanju združimo tabele tako, da združimo tiste vrstice obeh tabel, ki imajo določen podatek (ali podatke) tak, da je enak v obeh tabelah. Če je torej posamezna vrstica v tabeli taka, da se ne ujema z nobeno vrstico druge tabele, se njeni podatki v rezultatu ne bodo pojavili. Pri zunanjem združevanju pa dobimo lahko tudi vrstice, ki jih sestavljajo le podatki ene tabele, tam, kjer pa bi morali biti podatki iz druge tabele, pa so vrednosti NULL.

Da bi se izognili podvajanju vrstic v tabelah, ponavadi uvedemo primarni ključ. Ta nam enolično predstavlja vrstico. Uporabo ključev si bomo ogledali v razdelku 3.8.1.

3.8.1 Ključi

Ključi nam pomagajo pri razlikovanju zapisov v tabelah in pri združevanju tabel. Tabele imajo primarne in tuje ključe. Več o ključih si bomo ogledali v naslednjih razdelkih.

3.8.1.1 Primarni ključ

Vsaka tabela ima svoj primarni ključ (angleško primary key). To je tisti stolpec ali kombinacija več stolpcev, ki enolično določajo vrstico. Dve vrstici se torej razlikujeta vsaj v primarnem ključu. Primer takšnega primarnega ključa je matična številka občana, saj vemo, da dve osebi ne morata imeti enaki matični številki. Primeri iz vsakdanjega življenja so tudi davčna številka, serijska številka avtomobila, vpisna številka študenta ... Velikokrat za primarni ključ določimo stolpec, ki ga sistem zapolni z zaporednimi številkami (1, 2, 3 ...).

V tabeli `drzave` bi bil primeren primarni ključ stolpec `kratica` (obarvan z modro barvo), saj vemo da ima vsaka država svojo kratico. Stolpec `stevilo_prebivalcev` ne moremo vzeti kot primarni ključ, ker je možno, da imata dve državi enako število prebivalcev.

drzave		
kratica	naziv	stevilo_prebivalcev
A	Avstrija	8192880
D	Nemčija	82422299
SLO	Slovenija	2038733
⋮	⋮	⋮

V tabeli `rezultati_prvenstev` imamo tudi kratice držav in sicer v stolpcu `kratica_drzave.` Vendar le ta ni primeren za primarni ključ, saj je določena država lahko v rezultatih zapisana večkrat .

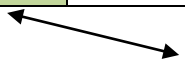
3.8.1.2 Tuji ključ

Tuji ključi nam pomagajo pri povezovanju tabel. Tuji ali drugače zunanji ključ (angleško foreign key) je stolpec ali kombinacija stolpcev v tabeli, ki je v neki drugi tabeli primarni ključ. Tabela ima lahko več tujih ključev. Določen stolpec (ali kombinacija stolpcev) je lahko hkrati primarni in tuji ključ.

Pri povezovanju tabel (Tabela 4: drzave in Tabela 7: rezultati_prvenstev) smo že ugotovili, da bi tabeli povezali preko kratice. Primarni ključ v tabeli `drzave`, to je stolpec `kratica` (zelena barva), bi povezali s stolpcem tabele `rezultati_prvenstev`, to je stolpec `kratica_drzave` (modra barva), in s tem ustvarili tuji ključ.

Povezava: `kratica = kratica_drzave`

drzave		
kratica	naziv	stevilo_prebivalcev
A	Avstrija	8192880
D	Nemčija	82422299
SLO	Slovenija	2038733
⋮	⋮	⋮



rezultati_prvenstev			
id	leto	kratica_drzave	Uvrstitev
1	2010	F	1
2	2010	HR	2
7	2008	DK	1
⋮	⋮	⋮	⋮

3.8.2 Relacije

Tabele povezujemo tako, da vrstici iz prve tabele ustreza vrstica iz druge tabele. Lahko pa tudi več vrstic iz ene tabele povežemo z večimi vrsticami druge tabele. Te povezave imenujemo relacije. Opisujejo nam odnose med podatki.

Poznamo več tipov relacij:

- **1:1**
  Vsaka vrstica prve tabele ustreza natanko eni vrstici iz druge tabele in vsaka vrstica druge tabele ustreza natanko eni vrstici iz prve tabele.
  Primer takega tipa relacije je relacija med tabelama `drzave` in `financni_podatki_drzav`. Vsaki državi iz tabele `drzave` pripada natanko ena vrstica iz tabele `financni_podatki_drzav` in vsaka vrstica iz tabele `financni_podatki_drzav` pripada natanko eni vrstici iz tabele `drzave`.

drzave		
kratica	naziv	stevilo_prebivalcev
A	Avstrija	8192880
F	Francija	60876136
DK	Danska	5450661
HR	Hrvaška	4494749
D	Nemčija	82422299
⋮	⋮	⋮

financni_podatki_drzav	
kratica	bdp
A	376841
D	3315643
DK	310760
F	2582527
HR	59917
⋮	⋮

- **1:n**
  Vrstice prve tabele so lahko povezane z večjim številom vrstic iz druge tabele.
  Primer takega tipa relacije je relacija med tabelama `drzave` in `rezultati_prvenstev` v shemi `rokomet`. Države lahko na prvenstvih nastopajo več let, tako da imajo v tabeli `rezultati_prvenstev` zapisanih več uvrstitev. Neko uvrstitev na enem prvenstvu pa lahko doseže le ena država.

drzave		
kratica	naziv	stevilo_prebivalcev
A	Avstrija	8192880
F	Francija	60876136
DK	Danska	5450661
HR	Hrvaška	4494749
D	Nemčija	82422299
⋮	⋮	⋮

rezultati_prvenstev			
id	leto	kratica_drzave	uvrstitev
1	2010	F	1
5	2010	DK	5
9	2008	F	3
13	2006	F	1
22	2004	HR	4
⋮	⋮	⋮	⋮

  Francija je v letih 2010 in 2006 osvojila 1. mesto, leta 2008 pa je bila 3.

- **n:m**
  Vrstice prve tabele so lahko povezane z večjim številom vrstic iz druge tabele in vrstice druge tabele so lahko povezane z večjim število vrstic iz prve tabele.
  Takšna relacija je med nagradami in nagrajenci v shemi `viktorji`. Nagrado podelijo večkrat (vsako leto) in nagrajenec lahko dobi več kot eno nagrado.
  Takšna povezava v jeziku SQL ni možna, zato je potrebno takšno povezavo realizirati drugače. To naredimo z uvedbo vmesne tabele. S prvo tabelo je povezana z relacijo n:1 in z drugo z 1:m.

vmesna tabela

seznam_nagrad	
nagrada	nagrada_id
1	Radijska osebnost
5	Televizijska oddaja
7	Zabavna TV-oddaja
10	Voditelj informativne TV-oddaje
⋮	⋮

razglasitev			
id	nagrada	nagrajenec	leto
1	1	1	2009
13	1	13	2008
17	5	6	2008
18	7	6	2008
29	5	24	2007
⋮	⋮	⋮	⋮

seznam_nagrajencev	
nagrajenec_id	nagrajenec
1	Denis Avdić
6	As ti tud not padu
13	Andrej Karoli
24	Avantura
⋮	⋮

3.8.3 INNER JOIN

Tabele lahko združujemo tudi z uporabo ukaza **INNER JOIN**. Temu združevanju pravimo notranje združevanje. Lastnost takšnega združevanja je, da v rezultatih dobimo le vrstice, ki vsebujejo ujemajoče vrednosti iz obeh tabel. Če vrstica prve tabele nima ujemajočih vrstic v drugi tabeli, ali obratno, ta vrstica prve oz druge tabele v rezultatih ni vključena .

```
SELECT seznam_stolpcev FROM tabela1
INNERJOIN tabela2 ON tabela1.stolpec1 = tabela2.stolpec2
```

Stolpca `stolpec1` in `stolpec2` sta ponavadi primarni in tuji ključ.
Ogledali si bomo primer z uporabo tabel `narocila` in `narocnik`.

Primer (Tabela 9: narocila,Tabela 8: narocnik)

Želimo narediti poizvedbo, ki nam bo vrnila imena in priimek naročnikov ter njihove številke naročil.

Iz tabele `narocnik` bomo potrebovali stolpca `ime` in `priimek`, iz tabele `narocila` pa stolpec `stevilka_narocila`. V obeh tabelah imamo identifikacijsko številko naročnika, tako da bo to naš ključ povezave.

Takole zapišemo stavek SELECT

```
SELECT narocnik.ime, narocnik.priimek, narocila.stevilka_narocila
FROM narocnik
INNER JOIN narocila ON
(narocnik.id_narocnik = narocila.stevilka_narocnika)
```

Ker stolpci v tabelah nimajo enakih imen, lahko pri navajanju stolpcev imena tabel izpustimo in dobimo poizvedbo

```
SELECT ime, priimek, stevilka_narocila FROM narocnik
INNER JOIN narocila ON (id_narocnik = stevilka_narocnika)
```

Dobljena tabela je

ime	priimek	stevilka_narocila
Janez	Novak	50121
Janez	Novak	50122
Mojca	Smodic	50123
Ana	Kolar	50124
Ana	Kolar	50126

Besedo **INNER** lahko v nekaterih sistemih izpustimo. To velja med drugim tudi za sistem PostgreSQL, ki ga uporabljamo, tako da bomo od sedaj naprej pisali le **JOIN**.

```
SELECT ime, priimek, stevilka_narocila FROM narocnik
JOIN narocila ON (id_narocnik= stevilka_narocnika)
```

Zgornji primer si lahko na spletni strani ogledate v kategoriji `Uporaba JOIN` pri nalogi 43.

Če želite utrditi svoje vedenje o tem, kako se uporablja INNER JOIN, so na spletni strani na voljo naloge 76, 77, 78 in 79.

3.8.4 LEFT OUTER JOIN

Če želimo, da so podatki iz prve (leve) tabele vključeni v rezultatih ne glede na to, ali v drugi tabeli obstaja vrstica, ki ima ujemajoč se podatek, uporabimo **LEFT OUTER JOIN**. Če za vrstico prve tabele obstaja ujemajoča se vrstica v drugi tabeli, v izhodni tabeli dobimo vrstico, ki jo sestavljajo podatki obeh vrstic. Če pa določena vrstica nima ustrezne vrstice v drugi tabeli, v izhodni tabeli dobimo vrstico, ki ima v stolpcih, kjer so sicer vrednosti iz druge tabele, vrednosti NULL. Temu združevanju pravimo levo zunanje združevanje. Naš sistem in še nekateri drugi delujejo tako, da besedo **OUTER** lahko izpustimo.

```
SELECT seznam_stolpcev FROM tabela1
LEFT JOIN tabela2 ON tabela1.stolpec1 = tabela2.stolpec2
```

Primer (Tabela 9: narocila, Tabela 8: narocnik)

Če uporabimo isti zgled kot prej pri notranjem združevanju (razdelek 3.8.3), le da pred JOIN dodamo besedo **LEFT** in napišemo

```
SELECT ime, priimek, stevilka_narocila FROM narocnik
LEFT JOIN narocila ON (id_narocnik = stevilka_narocnika)
```

Dobimo

ime	priimek	stevilka_narocila
Miha	Jazbec	
Anja		
Mojca	Smodic	50123
Ana	Kolar	50126
Ana	Kolar	50124
Janez	Novak	50122
Janez	Novak	50121
Janez	Vesel	

Vidimo, da v rezultatu dobimo vse tiste vrstice kot pri notranjem združevanju (obarvane so z modro barvo), kot tudi tiste vrstice iz leve (prve) tabele (v našem primeru tabela `narocnik`), ki nimajo ustreznih ujemajočih se podatkov v drugi tabeli.

Zgornji primer najdete v kategoriji `Uporaba JOIN` pri naslednji 67. nalogi.

3.8.5 RIGHT OUTER JOIN

Včasih želimo, da so v izhodni tabeli zagotovo vsi podatki iz desne tabele. Tu bo prav prišla uporaba združevanja z **RIGHT OUTER JOIN**. Tudi tu lahko podobno kot pri LEFT OUTER JOIN izpustimo besedo **OUTER** in dobimo **RIGHT JOIN**.

```
SELECT seznam_stolpcev FROM tabela1
RIGHT JOIN tabela2 ON tabela1.stolpec1 = tabela2.stolpec2
```

Kot rezultat dobimo vse ujemajoče se vrstice (kot pri notranjem združevanju) in vse tiste vrstice iz desne tabele (`tabela2`), ki nimajo ujemajočega podatka v levi tabeli (`tabela1`). Takrat namesto manjkajočih podatkov dobimo vrednosti NULL.

Primer (Tabela 9: narocila, Tabela 8: narocnik)

RIGHT JOIN bomo uporabili na istem primeru kot pri LEFT JOIN (razdelek 3.8.4).

```
SELECT ime, priimek, stevilka_narocila FROM narocnik
RIGHT JOIN narocila ON (id_narocnik = stevilka_narocnika)
```

Tokrat dobimo sledečo tabelo:

ime	priimek	stevilka_narocila
Janez	Novak	50121
Janez	Novak	50122
Mojca	Smodic	50123
Ana	Kolar	50124
		50125
Ana	Kolar	50126

V rezultatu dobimo vse tiste vrstice kot pri notranjem združevanju (obarvane so z modro barvo) in vrstice iz desne tabele , ki niso imele ustreznih podatkov v drugi tabeli.

Zgornji primer si lahko na spletni strani ogledate v kategoriji `Uporaba JOIN` pri 141. nalogi.

3.8.6 FULL JOIN

Če želimo, da so podatki iz prve in druge tabele vključeni v rezultatih, ne glede na to, ali obstajajo ujemajoče se vrstice, uporabimo **FULL JOIN**. **FULL JOIN** deluje, kot če bi istočasno uporabili LEFT OUTER JOIN in RIGHT OUTER JOIN. Kot rezultat dobimo vse ujemajoče se vrstice (kot pri notranjem združevanju) in vse tiste vrstice iz prve ali druge tabele, kjer v drugi (oziroma prvi) tabeli ni bilo ujemajočih se podatkov. Namesto manjkajočih podatkov dobimo vrednost NULL.

```
SELECT seznam_stolpcev FROM tabela1
FULL JOIN tabela2 ON tabela1.stolpec1 = tabela2.stolpec2
```

Primer (Tabela 9: narocila,Tabela 8: narocnik)

Uporabili bomo isti zgled kot pri prejšnjih združevanjih (razdelek 3.8.3), le da bomo uporabili FULL JOIN.

```
SELECT ime, priimek, stevilka_narocila FROM narocnik
FULL JOIN narocila ON (id_narocnik = stevilka_narocnika)
```

ime	priimek	stevilka_narocila
Janez	Novak	50121
Janez	Novak	50122
Miha	Jazbec	
Mojca	Smodic	50123
Ana	Kolar	50124
Ana	Kolar	50126
		50125
Anja		
Janez	Vesel	

Dobili smo tiste vrstice, kjer so se podatki prve tabele ujemali s podatki druge tabele (modra barva), vse tiste vrstice prve tabele, ki nimajo ujemajočega se podatka v drugi tabeli (rdeča barva) in vse vrstice druge tabele, ki nimajo ujemajočega se podatka v prvi tabeli (zelena barva).

Zgornji primer si lahko na spletni strani ogledate v kategoriji `Uporaba JOIN` pri 142. nalogi.

3.8.7 Združevanje treh ali več tabel

Do sedaj smo združevali le po dve tabeli naenkrat. Včasih potrebujemo podatke iz večih tabel. Kot primer bomo vzeli shemo o šoli.

Primer (Tabela 1: dijaki, Tabela 2: razredniki, Tabela 3: razredi)

Naredimo poizvedbo, ki nam vrne ime razrednika in povprečno oceno v tistem razredu, ki ga obiskuje Rok Breznik.

Ime razrednika bomo našli v tabeli `razrednik`. Povprečna ocena razreda je shranjena v tabeli `razredi`. Podatke o dijaku pa bomo našli v tabeli `dijaki`. Torej moramo zato, da pridemo do ustrezne poizvedbe, povezati vse tri tabele.

Tabeli `dijaki` in `razredi` lahko povežemo preko številke oziroma identifikacijsko številko razreda. Tabeli `razredi` in `razredniki` pa lahko združimo preko številke oziroma identifikacijske številke razrednika.

Tabeli bomo povezali na sledeč način:

`dijaki.id_razreda = razredi.stevilka_razreda`

`razredi.stevilka_razrednika = razredniki.id_razrednika`

Tabele združimo preko zgornjih povezav z ukazom JOIN, v pogoj pa postavimo ime in priimek dijaka. Prvi JOIN poveže tabelo `dijaki` in `razredi`. Združeno tabelo pa povežemo še s tabelo `razredniki`.

Stavek SELECT je sledeč

```
SELECT razrednik, povprecna_ocena FROMdijaki
 JOIN razredi ON dijaki.id_razreda = razredi.stevilka_razreda
JOIN razredniki ON razredi.stevilka_razrednika = razredniki.id_razrednika
 WHERE ime = 'Rok' AND priimek = 'Breznik'
```

Kot rezultat dobimo

razrednik	povprecna_ocena
Jereb	3.86

V razredu, kjer je Rok Breznik, imajo dijaki torej povprečno oceno 3.86, njihov razrednik pa se piše Jereb.

Primer (Tabela 6: prvenstva, Tabela 4: drzave, Tabela 7: rezultati_prvenstev)

Zanima nas, kdaj je bilo rokometno prvenstvo v Sloveniji in katero mesto je takrat Slovenija dosegla. Podatek o letu prvenstva bomo našli v tabeli `prvenstva`, uvrstitev Slovenije pa se nahaja v tabeli `rezultati_prvenstev`. V tabelah `prvenstva` in `rezultati_prvenstev` so samo kratice držav, tako da bomo do imena države prišli preko tabele `drzave`.

Najprej moramo vzpostaviti povezave med tabelami:

- *`drzave.kratica = prvenstva.kratica_drzave_gostiteljice`*
- *`drzave.kratica = rezultati_prvenstev.kratica_drzave`*

Preko tabele `drzave` bomo povezali tabeli `prvenstva` in `rezultati_prvenstev`. Tabele združimo z ukazom JOIN

```
prvenstva JOIN drzave ON drzave.kratica =
prvenstva.kratica_drzave_gostiteljice
JOIN rezultati_ prvenstev ON drzave.kratica =
rezultati_prvenstev.kratica_drzave
```

Potrebujemo še pogoje in sicer tega, da je naziv države Slovenija in da je leto prvenstva enako letu v rezultatih prvenstev.

SELECT prvenstva.leto, drzave.naziv, rezultati_prvenstev.uvrstitev FROM prvenstva
JOIN drzave ON drzave.kratica = prvenstva.kratica_drzave_gostiteljice
JOIN rezultati_prvenstev ON drzave.kratica = rezultati_prvenstev.kratica_drzave
WHERE drzave.naziv = 'Slovenija' AND prvenstva.leto = rezultati_prvenstev.leto

Slovenija je na domačem igrišču osvojila 2. mesto.

leto	naziv	uvrstitev
2004	Slovenija	2

Zgornja primera si lahko na spletni strani ogledate v kategoriji `Uporaba JOIN` pri 126. in 127. nalogi.

Združevanje večih tabel lahko utrdite pri 135., 136. in 137. nalogi.

4 Program

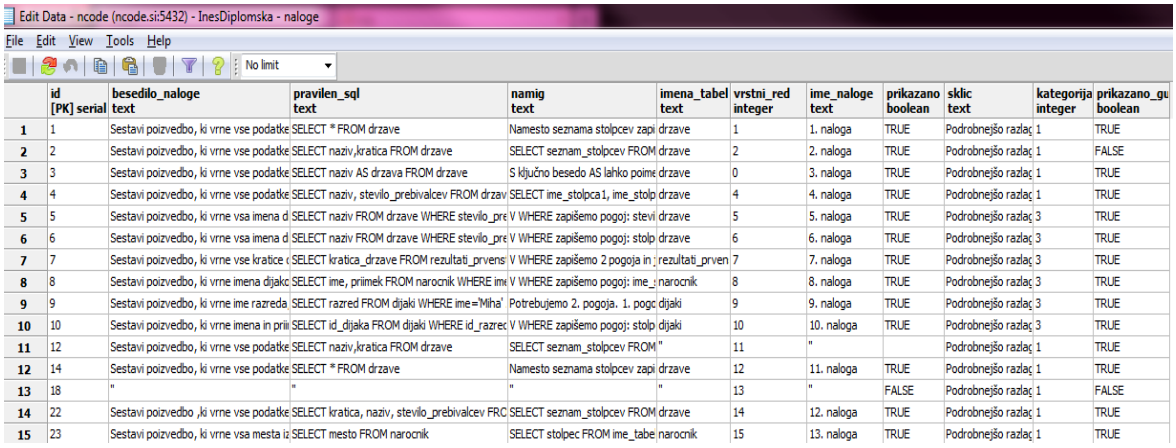
V tem razdelku si bomo ogledali, kako je bil razvit program za učenje stavkov SELECT. Program je sestavljen iz dela, ki je zadolžen za delo s spletno stranjo, na kateri so naloge razvrščene po kategorijah. Drugi del je administracijska stran, kjer lahko administrator ureja naloge in kategorije.

Struktura spletne strani je napisana v jeziku **HTML** (Hyper Text Markup Lanuguage). Čeprav je **HTML** standardiziran, si ga brskalniki včasih razlagajo različno. Zaradi tega pogosto pride do težav.

Pri oblikovanju spletne strani sem si pomagala s stili **CSS** (Cascading Style Sheets). **CSS** nam omogoča prilagoditev lastnosti elementov **HTML** (npr. pisava, barva, razmiki, poravnava ...). Z jezikom **JavaScript** sem še dodatno dopolnila funkcionalnost spletne strani. V ta namen sem uporabila dve knjižnici **jQuery** (verzija 1.6.2) in **jQuery** UI (1.8.14).

4.1 Delovanje programa

V bazi imamo tabelo `naloge`, v kateri so shranjene vse naloge in tabelo `kategorije`, kjer so shranjene vse kategorije. Ko v administrativnem delu dodamo novo nalogo, se ta shrani v bazi v tabelo `naloge` in dobi svojo identifikacijsko številko (`id`). Enako velja pri shranjevanju kategorije, le da se ta shrani v tabelo `kategorije`.


Edit Data - ncode (ncode.si:5432) - InesDiplomska - naloge

File Edit View Tools Help

No limit

	id [PK] serial	besedilo_naloge text	pravilen_sql text	namig text	imena_tabel text	vrstni_red integer	ime_naloge text	prikazano boolean	sklic text	kategorija integer	prikazano_gu boolean
1	1	Sestavi poizvedbo, ki vrne vse podatke	SELECT * FROM drzave	Namesto seznama stolpcev zapi	drzave	1	1. naloga	TRUE	Podrobnejšo razlag	1	TRUE
2	2	Sestavi poizvedbo, ki vrne vse podatke	SELECT naziv,kratica FROM drzave	SELECT seznam_stolpcev FROM	drzave	2	2. naloga	TRUE	Podrobnejšo razlag	1	FALSE
3	3	Sestavi poizvedbo, ki vrne vse podatke	SELECT naziv AS drzava FROM drzave	S ključno besedo AS lahko poime	drzave	0	3. naloga	TRUE	Podrobnejšo razlag	1	TRUE
4	4	Sestavi poizvedbo, ki vrne vse podatke	SELECT naziv, stevilo_prebivalcev FROM drzav	SELECT ime_stolpca1, ime_stolp	drzave	4	4. naloga	TRUE	Podrobnejšo razlag	1	TRUE
5	5	Sestavi poizvedbo, ki vrne vsa imena d	SELECT naziv FROM drzave WHERE stevilo_pre	V WHERE zapišemo pogoj: stevi	drzave	5	5. naloga	TRUE	Podrobnejšo razlag	3	TRUE
6	6	Sestavi poizvedbo, ki vrne vsa imena d	SELECT naziv FROM drzave WHERE stevilo_pre	V WHERE zapišemo pogoj: stolp	drzave	6	6. naloga	TRUE	Podrobnejšo razlag	3	TRUE
7	7	Sestavi poizvedbo, ki vrne vse kratice	SELECT kratica_drzave FROM rezultati_prvens	V WHERE zapišemo 2 pogoja in	rezultati_prven	7	7. naloga	TRUE	Podrobnejšo razlag	3	TRUE
8	8	Sestavi poizvedbo, ki vrne imena dijakc	SELECT ime, priimek FROM narocnik WHERE ime	V WHERE zapišemo pogoj: ime_	narocnik	8	8. naloga	TRUE	Podrobnejšo razlag	3	TRUE
9	9	Sestavi poizvedbo, ki vrne ime razreda	SELECT razred FROM dijaki WHERE ime='Miha'	Potrebujemo 2. pogoja. 1. pogo	dijaki	9	9. naloga	TRUE	Podrobnejšo razlag	3	TRUE
10	10	Sestavi poizvedbo, ki vrne imena in prii	SELECT id_dijaka FROM dijaki WHERE id_razrec	V WHERE zapišemo pogoj: stolp	dijaki	10	10. naloga	TRUE	Podrobnejšo razlag	3	TRUE
11	12	Sestavi poizvedbo, ki vrne vse podatke	SELECT naziv,kratica FROM drzave	SELECT seznam_stolpcev FROM	''	11	''		Podrobnejšo razlag	1	TRUE
12	14	Sestavi poizvedbo, ki vrne vse podatke	SELECT * FROM drzave	Namesto seznama stolpcev zapi	drzave	12	11. naloga	TRUE	Podrobnejšo razlag	1	TRUE
13	18	''	''	''	''	13	''	FALSE	Podrobnejšo razlag	1	FALSE
14	22	Sestavi poizvedbo ,ki vrne vse podatke	SELECT kratica, naziv, stevilo_prebivalcev FRC	SELECT seznam_stolpcev FROM	drzave	14	12. naloga	TRUE	Podrobnejšo razlag	1	TRUE
15	23	Sestavi poizvedbo, ki vrne vsa mesta iz	SELECT mesto FROM narocnik	SELECT stolpec FROM ime_tabe	narocnik	15	13. naloga	TRUE	Podrobnejšo razlag	1	TRUE

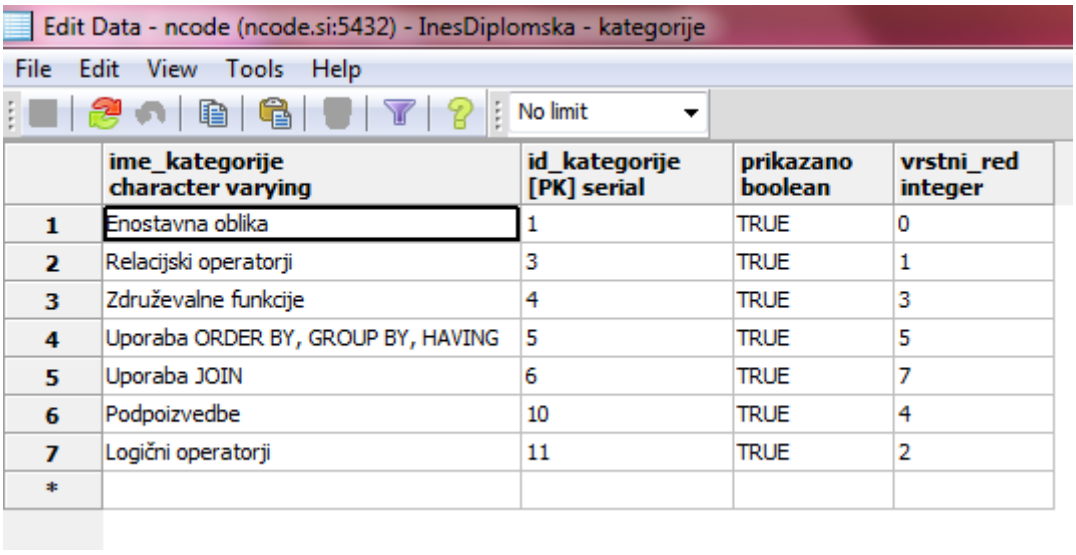
Slika 1: naloge

Sestavni deli vsake naloge so:

- `id` [serial]
- `besedilo_naloge` [text]
- `pravilen_sql` [text]
- `namig` [text]
- `imena_tabel` [text]

- `vrstni_red` [integer]
- `ime_naloge` [text]
- `prikazano` [boolean]
- `sklic` [text]
- `kategorija` [integer]
- `prikazano_gumb` [boolean]

`id` je tipa `serial`, kar pomeni, da RDBMS sam dodeli nalogi enolično identifikacijsko število. V stolpcu `besedilo_naloge` je zapisano besedilo naloge, ki jo uporabnik rešuje. V `pravilen_sql` je zapisan stavek SELECT, ki pravilno reši nalogo. V `namig` je zapisanih nekaj besed o tem, kako lažje rešiti nalogo. V `imena_tabel` so shranjena imena tabel, ki jih mora uporabnik uporabiti pri pravilni rešitvi naloge. Vsaka naloga ima svoj enolično določen `vrstni_red`, s katerim si pomagamo pri razvrščanju nalog. Vsaka naloga ima tudi svoje ime, ki je shranjeno v `ime_naloge`. `prikazano` je tipa `boolean`, zato je vrednost tega stolpca lahko TRUE ali FALSE. Če je vrednost enaka TRUE, potem je naloga prikazana na spletni strani, v nasprotnem primeru je na spletni strani ni. V `sklic` je zapisano, kje v diplomski nalogi si lahko uporabnik prebere kaj več teorije, ki mu bo pomagala pri reševanju konkretne naloge. V `kategorija` je zapisano, v katero kategorijo spada naloga. `Prikazano_gumb` je bodisi vrednosti TRUE bodisi FALSE. Če je vrednost enaka TRUE, potem se pri nalogi na spletni strani prikaže gumb `Pravilna poizvedba`, drugače tega gumba ni.



	ime_kategorije character varying	id_kategorije [PK] serial	prikazano boolean	vrstni_red integer
1	Enostavna oblika	1	TRUE	0
2	Relacijski operatorji	3	TRUE	1
3	Združevalne funkcije	4	TRUE	3
4	Uporaba ORDER BY, GROUP BY, HAVING	5	TRUE	5
5	Uporaba JOIN	6	TRUE	7
6	Podpoizvedbe	10	TRUE	4
7	Logični operatorji	11	TRUE	2
*				

Slika 2: kategorije

Sestavni deli vsake kategorije so:

- `ime_kategorije` [charactervarying]
- `id_kategorije` [serial]
- `prikazano` [boolean]
- `vrstni_red` [integer]

Vsaka kategorija ima svoje ime, ki je shranjeno v stolpcu `ime_kategorije`. `id_kategorije` je tipa `serial`, kar pomeni da RDBMS sam doda zaporedno številko pri vsaki novo dodani nalogi.

`prikazano` je bodisi TRUE bodisi FALSE. Če je vrednost TRUE, potem je kategorija izpisana na spletni strani, drugače je na spletni strani ni. Vsaka kategorija ima enoličen vrstni red v stolpcu `vrstni_red`. Ta nam pomaga pri zamenjavi vrstnega reda kategorij na spletni strani.

V datotekah `SQL.java`, `Naloga.java` in `Kategorija.java` so zapisane različne metode za upravljanje s temi nalogami in kategorijami v bazi.

V `SQL.java` so zapisane naslednje metode:

- statična metoda `izvediSQL(sql)`
  Ta metoda sprejme parameter `sql` in ga izvede, ter vrne rezultate.

- statična metoda `izpisiTabelo(ime tabele)`
  Vsaka naloga se nanaša na eno ali več tabel. Metoda `izpisiTabelo()`sprejme parameter `ime tabele` in jo izpiše. Izpiše stolpce in 3 vrstice.

- statična metoda `vrniTabele()`
  Metoda vrne imena vseh tabel v bazi, ki jih potrebujemo za reševanje nalog.

- statična metoda `vrniSheme()`
  Metoda vrne imena vseh shem v bazi.

`V datoteki Naloga.java` imamo razred `Naloga`, ki je sestavljen iz naslednjih metod:

- statična metoda `vrniNalogo(id naloge)`
  Metoda sprejme parameter `id naloge` in vrne ustrezno nalogo.

- statična metoda `vrniNaloge()`
  Metoda vrne vse naloge in baze.

- statična metoda `vrniNalogeZaKategorijo(id kategorije, samo prikazane)`
  Metoda sprejme 2 parametra – id kategorije in samo prikazane. Če je parameter `samo prikazane` nastavljen na FALSE, potem metoda vrne vse naloge, ki pripadajo dani kategoriji. V nasprotnem primeru vrne samo tiste naloge, ki imajo v stolpcu `prikazano` vrednost TRUE in pripadajo dani kategoriji.

- objektna metoda `shrani()`
  Metoda shrani v bazo novo nalogo ali pa posodobi že obstoječo.

- objektna metoda `brisi()`
  Metoda izbriše nalogo iz baze.

- statična metoda `vrniMaxVrstniRed()`
  Metoda vrne največje število v tabeli naloge v stolpcu `vrstni_red`. To število potrebujemo pri dodajanju nove naloge.

V `datoteki Kategorija.java` imamo razred `Kategorija`, ki je sestavljen iz naslednjih metod:

- statična metoda `vrniKategorijo(id kategorije)`
  Metoda sprejme parameter `id kategorije` in vrne ustrezno kategorijo.

- statična metoda `vrniKategorije()`
  Metoda nam vrne vse kategorije v bazi.

- objektna metoda `shrani()`
  Metoda shrani v bazo novo kategorijo ali pa posodobi že obstoječo.

- objektna metoda `brisi()`
  Metoda izbriše kategorijo iz baze.

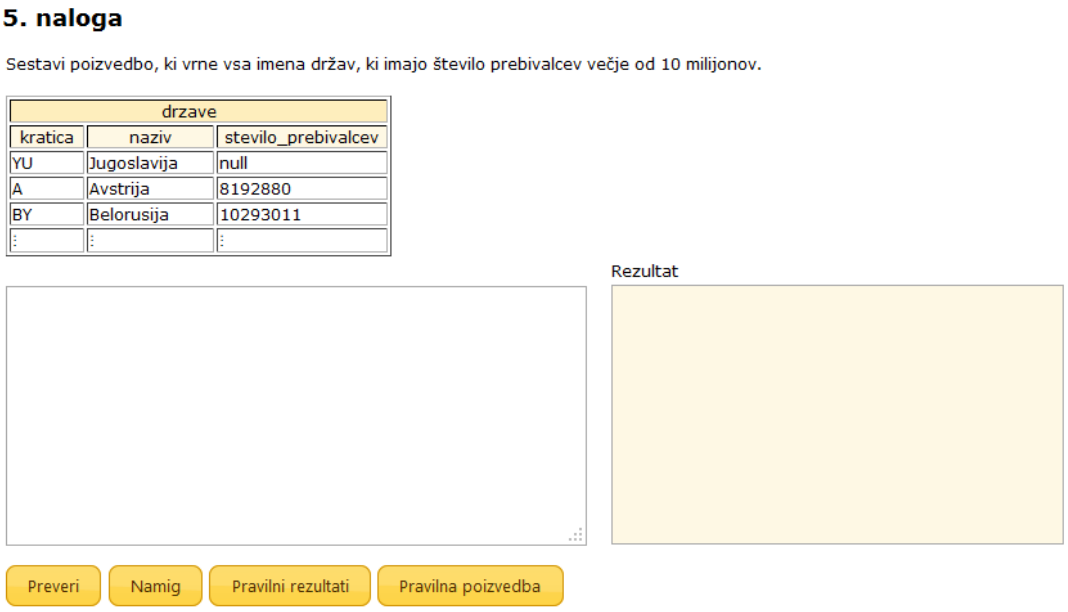
Spletna stran je sestavljena iz večih podstrani. Na teh podstraneh uporabljamo metode opisane zgoraj.

4.2 Uporabnikova stran

Osnovna stran uporabnikove strani ima na levi strani meni, kjer so zapisane kategorije. S klikom na eno izmed kategorij se prikaže seznam nalog iz te kategorije. Ob kliku na kategorijo se prikaže prva naloga v tej kategoriji.

Slika 3: uporabnikova stran

Pogledali si bomo, kako je videti posamezna naloga.

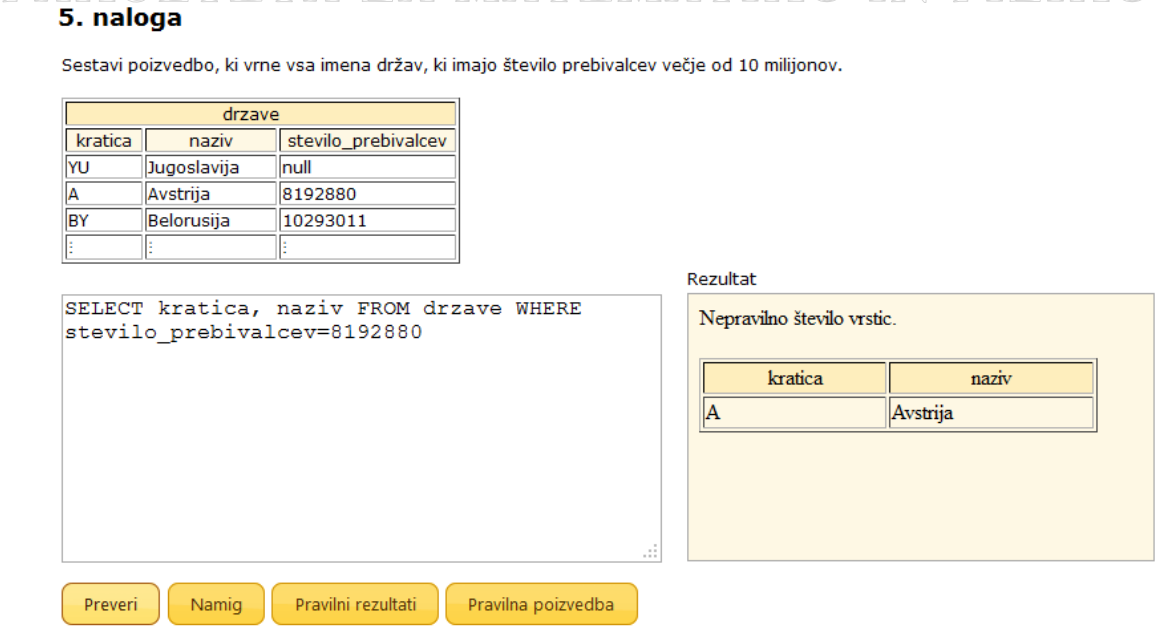


Slika 4: posamezna naloga

Na vrhu se izpiše ime naloge, besedilo in prvih nekaj vrstic tabel, ki jih uporabnik potrebuje za reševanje naloge. Pod tabelami sta dve okni. Levo okno je vnosno in vanj uporabnik zapiše rešitve naloge. V desnem oknu pa se ob pritisku na gumba `Preveri` ali `Pravilni rezultati` izpišejo rezultati stavkov SELECT.

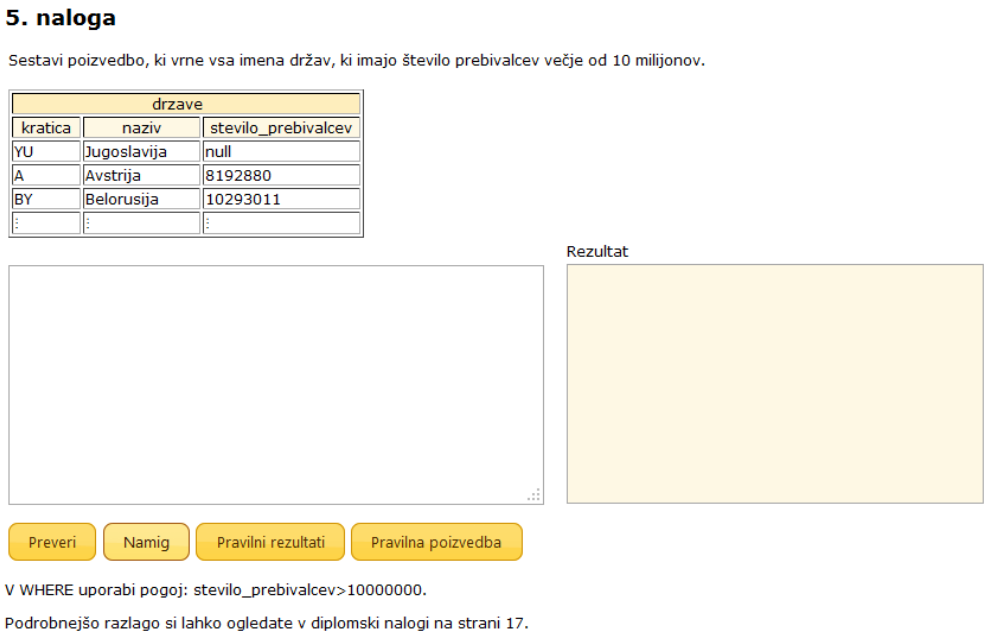
Na sliki (Slika 5: gumb Preveri) vidimo, da smo našemu primeru zapisali nepravilen, a sintaktično korekten stavek SELECT. Zato nam ob pritisku na gumb `Preveri` v desnem oknu izpiše napako »Nepravilno število vrstic«. Kakšne napake so možne, si bomo ogledali v razdelku 4.5.1. Pod obvestilom izpiše rezultate uporabnikovega stavka SELECT.

Omenimo, da se v vsakem primeru ob kliku na gumb `Preveri` na spletni strani izpišejo rezultati uporabnikovega stavka, ne glede na to, ali stavek vrača pravilne rezultate. Če pa je poizvedba sintaktično nepravilna, se izpiše »SQL napaka.«



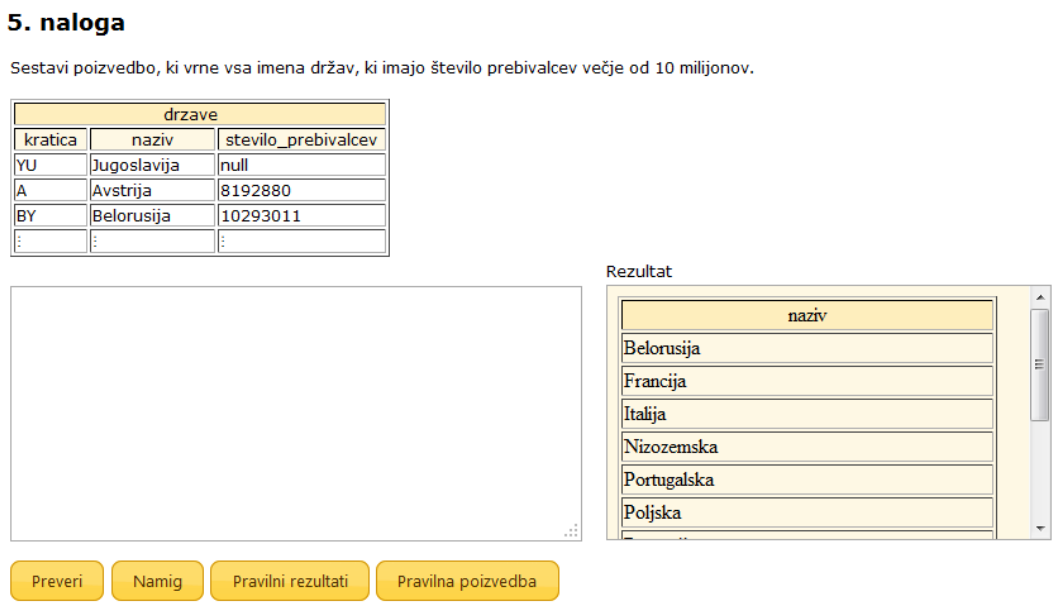
Slika 5: gumb Preveri

S pritiskom na gumb `Namig` se pod gumbi izpiše namig in številka strani v diplomski nalogi, kar uporabniku lahko pomaga pri reševanju naloge.



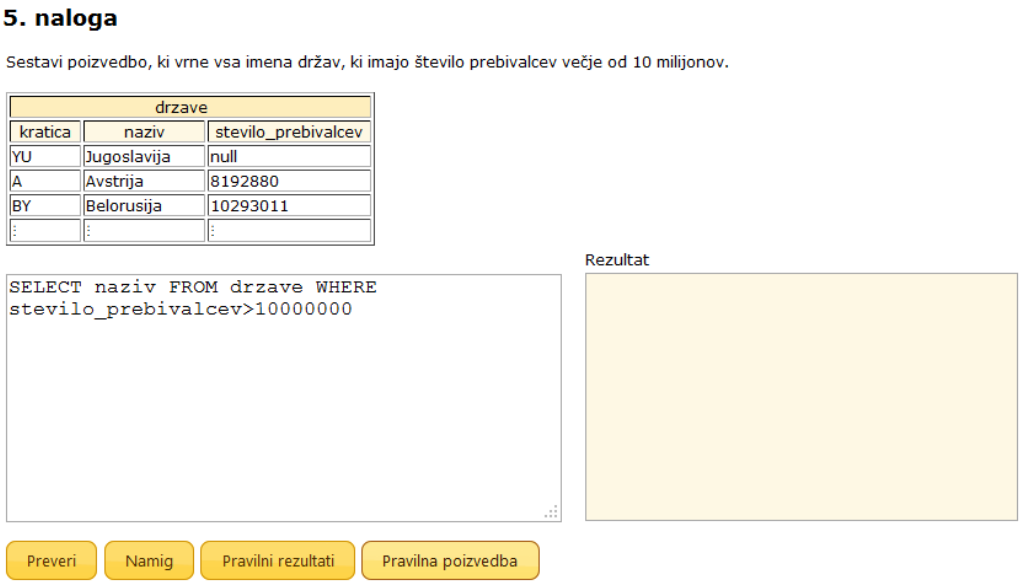
Slika 6: gumb Namig

Če želi uporabnik izvedeti, kakšni so pravilni rezultati, klikne na gumb `Pravilni rezultati` in v desnem oknu se izpišejo pravilni rezultati.



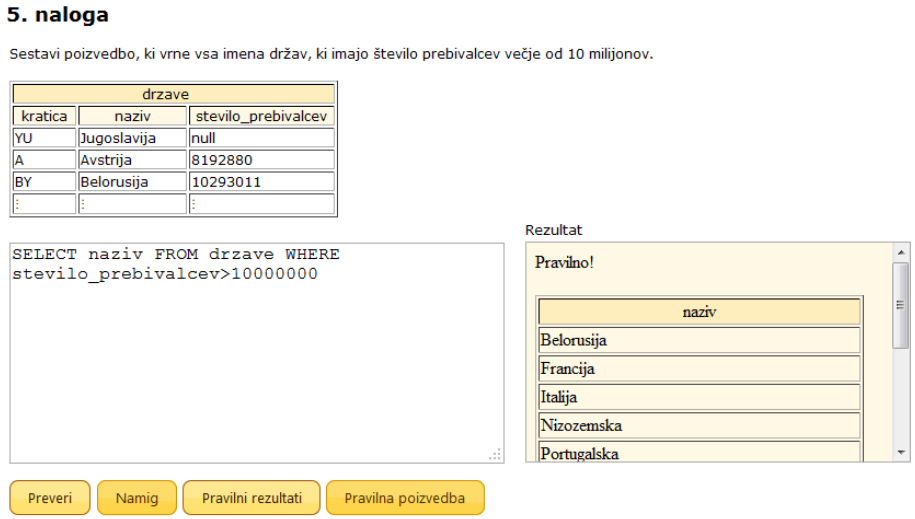
Slika 7: gumb Pravilni rezultati

Ob pritisku na gumb `Pravilna poizvedba` se v vnosnem oknu pojavi pravilen stavek SQL.



Slika 8: gumb Pravilna poizvedba

V primeru, da uporabnik zapiše pravilen stavek SELECT in klikne gumb `Preveri`, se v desnem oknu izpiše »Pravilno!« in pravilni rezultati.



Slika 9: pravilni rezultati

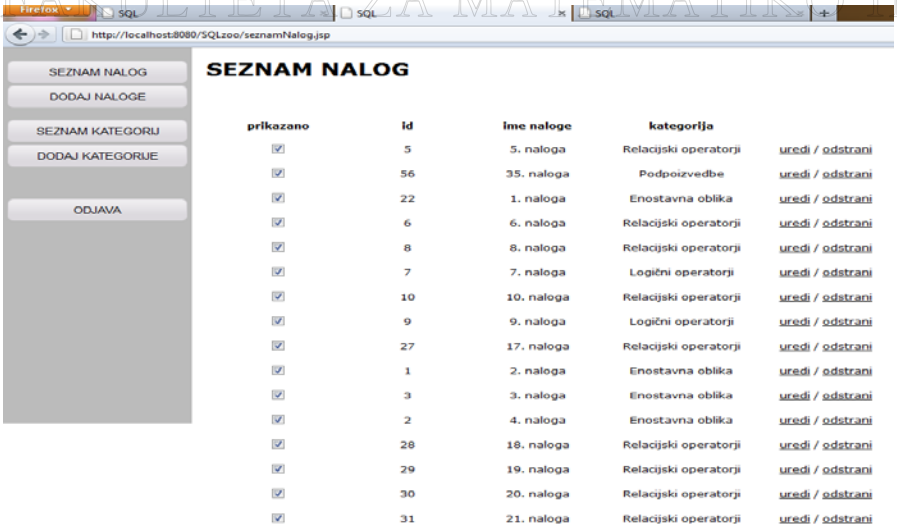
4.3 Administratorska stran

Administratorska stran ima na levi strani meni za urejanje nalog in kategorij.

Sestavni deli menija:

- `seznam nalog,`
- `dodaj naloge,`
- `seznam kategorij,`
- `dodaj kategorije.`

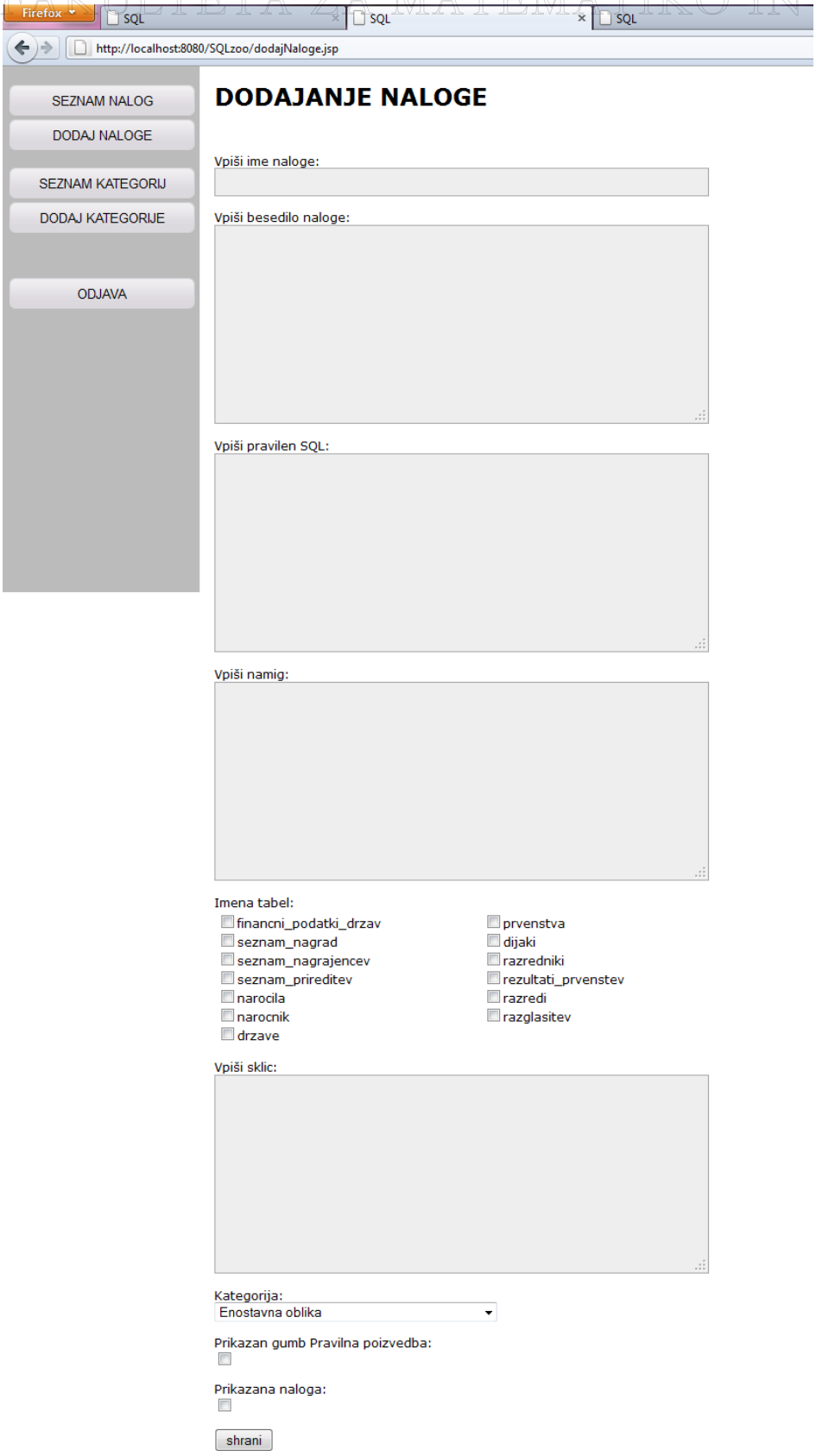
S klikom na `seznam nalog` se izpišejo vse naloge v bazi. S klikom na `uredi` pri posamezni nalogi lahko nalogo urejamo, ali pa jo izbrišemo s klikom na `odstrani`.



Slika 10: seznam nalog

V `dodaj naloge` lahko dodamo novo nalogo. V vnosna okna vpišemo vse potrebne podatke in s pritiskom na gumb `shrani` se naloga shrani v bazo.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

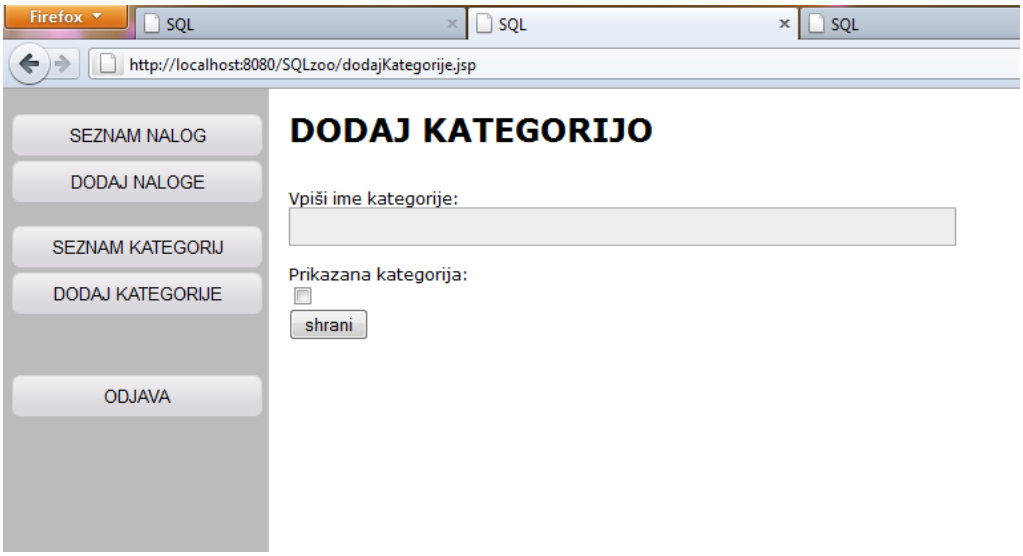


Slika 11: dodajanje naloge

S klikom na `seznam kategorij` se prikažejo vse kategorije v bazi. S klikom na `seznam nalog` pri eni od kategorij se prikažejo naloge v tej kategoriji. Te naloge lahko urejamo ali odstranimo. Kategorije lahko prav tako urejamo ali zbrišemo. Kategorije ne moremo izbrisati, če so v njej naloge. Naloge iz te kategorije je potrebno najprej odstraniti ali pa premakniti v drugo kategorijo. Šele ko v kategoriji ni nalog, jo lahko izbrišemo.

Slika 12: seznam kategorij

V `dodaj kategorije` lahko dodamo novo kategorijo. V vnosno okno vpišemo ime kategorije in označimo, ali želimo, da se kategorija prikaže na spletni strani. Kategorija se bo prikazala šele, ko bo vsebovala vsaj eno nalogo. S pritiskom na gumb `shrani` se kategorija shrani v bazo.



Slika 13: dodaj kategorijo

4.4 Razvojna orodja

Pri razvoju sem uporabila več orodij:

- `Eclipse`
  Za večino razvoja sem uporabila razvojno okolje Eclipse. V njem sem napisala javansko kodo, kodo HTML, CSS in JavaScript.
- `pgAdmin III`
  Za upravljanje z RDBMS-jem (PostgreSQL) sem uporabila pgAdmin III. Z njim sem ustvarjala in spreminjala tabele, urejala podatke, itd..

- `Notepad`
  Z njegovo pomočjo sem vse datoteke shranila v kodni tabeli UTF-8.

- `Firefox + Firebug`
  Celotno testiranje sem opravila v brskalniku Firefox. Še posebej sem si pomagala z dodatkom Firebug, ki mi je omogočil hitrejši razvoj. Istočasno sem testiranje opravljala tudi v brskalniku Chrome.

4.5 Nekaj zanimivih točk razvoja

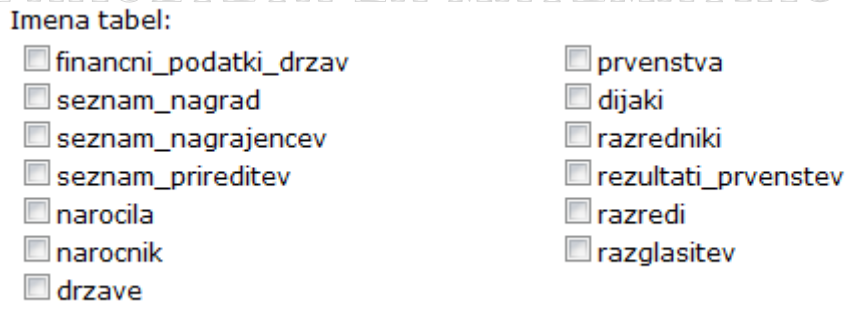
4.5.1 Primerjanje uporabnikovega stavka SELECT s pravilnim stavkom SELECT

Primerjavo med stavkoma naredimo na podstrani `izvediSQL.jsp`. Najprej primerjamo število vrstic med pravilnimi rezultati in uporabnikovimi rezultati. Če število vrstic ni enako, pomeni, da tudi rezultati niso enaki. Zato uporabnikov stavek SELECT ni pravilen. Program izpiše napako »Nepravilno število vrstic.«. Če pa je število vrstic enako, primerjamo še število stolpcev med rezultati. Če število ni enako, program izpiše napako »Nepravilno število stolpcev.«. Če je število enako, primerjamo imena stolpcev. Če imena niso enaka, program izpiše napako »Stolpci niso pravilni.«. Če se imena ujemajo, primerjamo še vse vrednosti v rezultatih. Če so vrednosti enake, kaže, da je uporabnikov stavek SELECT pravilen in program izpiše »Pravilno!«. Če vrednosti niso enake, uporabnikov stavek ni pravilen in program izpiše napako »Vrednosti v tabeli so nepravilne.«.

4.5.2 Tabele

Pri nalogah se lahko zgodi, da uporabnik potrebuje več kot eno tabelo. Problema, kako dodeliti eni nalogi več tabel, sem se lotila tako, da sem v bazi v tabeli naloge dodala stolpec `imena_tabel`. Stolpec je tipa `text`. Imena tabel so v stolpcu zapisana v nizu, ločena z vejicami. V `Naloga.java` imamo zapisano metodo `getImenaTabelLoceno()`, ki niz loči po vejicah in dobimo tabelo imen, v kateri so zapisana imena tabel posamezne naloge. Vsako ime predstavlja eno celico v tabeli.

V administraciji imamo za urejanje ali dodajanje naloge podana tudi imena tabel, ki jih potrebuje uporabnik pri nalogi.



Slika 14: imena tabel

V administraciji lahko pri posamezni nalogi kliknemo več potrditvenih polj (checkobox) in pri tej nalogi se izpišejo vse potrjene tabele.

4.5.3 Vrstni red

Vrstni red je pomemben pri razvrščanju nalog in kategorij. Vsaka na novo dodana naloga dobi svoj podatek v stolpcu `vrstni_red`. Z metodo `vrniMaxVrstniRed()` iz tabele naloge dobimo največje število v stolpcu `vrstni_red`. Nova naloga dobi v stolpcu `vrstni_red` vrednost, ki smo jo dobili z metodo `vrniMaxVrstniRed()+1`. Tako je nova naloga razvrščena na zadnje mesto. Vrstni red nalog lahko spreminjamo na posamezni kategoriji. S pritiskom na puščici gor ali dol pomaknemo nalogo za eno mesto gor ali dol. `spremeniVrstniRed.jsp` zamenja dve nalogi. Če pritisnemo puščico gor, se vrstni red zgornje in sedanje naloge zamenja. Če pritisnemo puščico dol, se sedanja naloga zamenja s spodnjo nalogo. Podobno velja za kategorije.

Oglejmo si primer zamenjave vrstnega reda.

Slika 15: prvoten vrstni red

Pri zgornji sliki bomo pritisnili na pušico gor pri nalogi z imenom 1. naloga in dobimo naslednji seznam.

prikazano	id	ime naloge	kategorija			
☑	1	1. naloga	Enostavna oblika	▲	▼	uredi / odstrani
☑	3	3. naloga	Enostavna oblika	▲	▼	uredi / odstrani
☑	2	2. naloga	Enostavna oblika	▲	▼	uredi / odstrani
☑	4	4. naloga	Enostavna oblika	▲	▼	uredi / odstrani
☑	14	11. naloga	Enostavna oblika	▲	▼	uredi / odstrani

Slika 16: vrstni red po preoblikovanju

4.5.4 Prikazano

V bazi imamo veliko različnih nalog. Za nekatere naloge ne želimo, da so vidne na spletni strani. V bazi sem v tabeli `naloge` dodala stolpec `prikazano` tipa `boolean`.Urejanje stolpca poteka preko administracije. Zaradi lažjega urejanja sem uvedla rešitev s pomočjo knjižnice jQuery in tehnologije AJAX. Ob kliku na potrditveno polje (checkbox) se izvede klic AJAX (klic brez osvežitve strani), ki spremeni stanje tega stolpca. Podobno velja tudi za kategorije.

4.5.5 Uporaba šumnikov

Pri uporabi šumnikov sem imela težave pri prikazu in njihovem vnosu. Pri prikazu sem imela težave predvsem zaradi orodja Eclipse, saj ta datotek ni shranjeval v kodni tabeli UTF-8. Zato sem si morala pomagati z orodjem Notepad. Pri vhodnih podatkih Java teh podatkov ni avtomatično pretvorila v UTF-8, zato sem morala to storiti ročno. Primer pretvorbe: `Stringsql = newString(vhod.getBytes('' 8859_1''), ''UTF8'').`

Zaključek

V diplomski nalogi sem se dotaknila osnov stavkov SELECT v jeziku SQL. Nekaj znanja o tem sem pridobila že tekom študija pri predmetu Računalništvo 2. To znanje pa sem še poglobila pri izdelavi diplomske naloge. Kot bistven vir informacij sem uporabila internet, kjub temu da je večina spletnih strani napisanih v angleškem jeziku. Upam, da bo moja diplomska naloga komu v pomoč tudi zato, ker je napisana v slovenščini in se bralcu ne bo potrebno ukvarjati s težavami pri prevajanju.

Uporabnik si lahko v diplomi prebere osnove različne uporabe stavkov SELECT. To teorijo lahko nadgradi z uporabo spletne strani, na kateri so naloge z uporabo stavkov SELECT. Naloge so razvrščene po kategorijah, tako da se lahko študent loti učenja po sklopih. Pri vsaki nalogi je označeno, v katero poglavje v diplomski nalogi spada. Če je naloga pretežka, si uporabnik lahko enostavno prebere nekaj osnov v diplomski nalogi. Administrator pa lahko te naloge ureja na administratorski strani.

Upam, da bo moja diplomska naloga pomagala tako profesorjem pri razlagi študentom kot tudi samim študentom pri pripravi na kolokvije in izpite.

Literatura

1. Peter Mrhar, *Uvod v SQL*, Nova Gorica, Založba Flamingo, 2002
2. Tomaž Mohorič,*Podatkovne baze*, Fakulteta za računalništvo in informatiko, 2002

Spletni viri

1. Spletna učilnica FMF pri predmetu podatkovne baze
   http://ucilnica.fmf.uni-lj.si/course/view.php?name=PRM-PB (zadnji dostop 28.7.11)
2. Oblikovanje besedila s pomočjo CSS
   http://www.w3schools.com/css/css_text.asp (zadnji dostop 2.7.11)
3. Oblikovanje tabel s pomočjo CSS
   http://www.w3schools.com/css/css_table.asp (zadnji dostop 2.7.11)
4. Delovanje spletnih povezav s pomočjo HTML
   http://www.w3schools.com/html/html_links.asp (zadnji dostop 15.6.11)
5. Uporaba seznamov s pomočjo HTML
   http://www.w3schools.com/html/html_lists.asp (zadnji dostop 15.6.11)
6. Uporaba input v HTML
   http://www.w3schools.com/tags/tag_input.asp (zadnji dostop 5.7.11)
7. Uporaba HAVING v stavkih SELECT
   http://www.w3schools.com/sql/sql_having.asp (zadnji dostop 20.7.11)
8. Tuji ključi pri tabelah v jeziku SQL
   http://www.w3schools.com/sql/sql_foreignkey.asp (zadnji dostop 20.7.11)
9. Splošno o jeziku SQL
   http://www.sql.org/ (zadnji dostop 2.7.11)
10. Splošno oPostgreSQL
    http://www.postgresql.org/ (zadnji dostop 10.7.11)
11. Uporaba LIMIT v stavkih SELECT
    http://www.postgresql.org/docs/9.0/static/sql-select.html#SQL-DISTINCT (zadnji dostop 5.7.11)
12. Uporaba shem v PostgreSQL
    http://www.network-theory.co.uk/docs/postgresql/vol1/TheSchemaSearchPath.html (zadnji dostop 20.7.11)
13. Tipi števil v PostgreSQL
    http://www.postgresql.org/docs/9.0/interactive/datatype-numeric.html (zadnji dostop 28.6.11)

14. Funkcije za delo z nizi v PostgreSQL
http://www.postgresql.org/docs/9.0/static/functions-string.html (zadnji dostop 28.6.11)
15. Delovanje nalog
http://sqlzoo.net/ (zadnji dostop 15.7.11)
16. Podatki o bruto domačih proizvodov držav na prebivalca
http://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)_per_capita (zadnji dostop 2.7.11)
17. Podatki o bruto domačih proizvodov držav
http://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal) (zadnji dostop 2.7.11)
18. Podatki o rezultatih Viktorjev
http://www.viktorji.si/index.php (zadnji dostop 2.7.11)
19. Uporaba jQuery
http://api.jquery.com/jQuery.get/ (zadnji dostop 19.7.11)
20. Uporaba AJAX
http://api.jquery.com/jQuery.ajax/ (zadnji dostop 19.7.11)
21. Uporaba sej
http://www.jsptut.com/Sessions.jsp (zadnji dostop 21.8.11)
22. Uporaba preusmeritve strani
http://www.exampledepot.com/egs/javax.servlet.jsp/redirect.html (zadnji dostop 21.8.11)
23. Pravice uporabnikov v PostgreSQL
http://www.postgresql.org/docs/8.3/static/sql-grant.html (zadnji dostop 21.8.11)

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO