Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у инжењерству

Предмет: Базе података

Број индекса: 2017/2014

Милош З. Димитријевић

Базе података у Веб окружењу

завршни рад

Комисија за преглед и одбрану:

1. **Проф. Др Милан Ерић - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру овог завршног рада кандидат треба да:

Проучи литературу и презентира основе концепата приступања базама података на Вебу. Завршни рад треба да обухвати уводна разматрања о развоју база података, основне карактеристике Веб окружења и основне концепте приступања базама података на Вебу као и конкретан показни пример.

Препоручена литература:

1. Лазаревић Б.: ***Базе података***, ФОН Београд, Београд 2003.
2. Павловић-Лажетић Г.: ***Основе релационих база података***, Математички факултет, Београд, 2000.
3. Полишчук Ј.: ***Пројектовање информационих система***, Електротехнички факултет, Подгорица, 2007.
4. Buyens, Jim : ***Развој база података на web-у***, ЦЕТ, Београд, 2000.

Крагујевац, 20.9.2016. Ментор:

Проф. Др. Милан Ерић

Изјава о самосталној изради рада

Изјављујем да сам овај завршни рад урадио самостално, служећи се стеченим знањем и искуством као и наведеном литературом.

2017/2014 Милош Димитријевић

_____

(потпис)

РЕЗИМЕ

У овом раду, најпре ће бити речи о основама података и информација, затим основама база података, а након тога биће представљене основе програмирања у Веб окружењу уз коришћење база података као сталног складишта и њихове предности и недостатке у односу на коришћење класичних врста складишта као што су текст фајлови и радне променљиве у програмском коду.

Такође, рад ће представити и неколико практичних примера коришћења база података у Веб окружењу, од којих ће један бити детаљно обрађен, како би читалац у потпуности био упознат са начином рада и имплементације база података у оквиру програмирања класичних Веб страница.

КЉУЧНЕ РЕЧИ

- База података
- Веб База података
- Имплементација База података у Веб окружењу

SUMMARY

In this thesis, first there will be words about basics of data and information, after which it will be discussed about basics of databases as well as the fundamentals of programming in Web environment while using databases as a permanent data storage, as well as their advantages and disadvantages relative to usage of the classical storages like text files and runtime variables in programming code

Also, the thesis will introduce the reader with a couple of practical examples, out of which one will be studied thoroughly, in order to ensure that the reader would be entirely acquainted with the method of work and implementation of databases in terms of programming classical Web pages.

KEYWORDS

- Database
- Web database
- Implementing database in Web environment

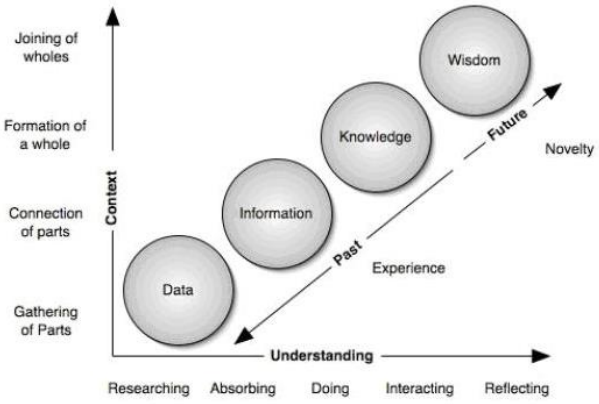
САДРЖАЈ:

УВОД

-О информацији-

Разликовање података од информација кључни је услов за управљање знањем. Информација представља један од основних појмова у информатици и сама њена заснованост и интерпретација припада информатичкој науци. Она је постала релевантан појам за све науке које се баве симболичком комуникацијом у распону од математике до рачунарске науке, од друштвених наука до медицине, од логике до лингвистике, итд. То је информацији дало интердисциплинарну димензију, јер свака наука покушава да протумачи тај комплексан појам.

Информација је реч латинског порекла *in formare* и изворно је значила стављање у одређену форму, односно, давање облика нечему.



Слика 1 – Од податка до мудрости

Слика 2 – Пирамида знања

По својим својствима информација у модерном информатичком добу могла би се интерпретирати као скуп података уобличених тако да имају неки смисао (слика 1). Даљом разрадом, можемо закључити да се креирањем целине добијених из систематизованих информација формира знање, а затим спајањем целина мудрост. Подаци, информације и знање могу се подвести под искуство, док мудрост, темељена на пређашњем искуству производи новине у одређеној области. Овај појам, назива се хијерархија знања и упрошћено, може се објаснити пирамидом знања, *слика 2.*

Хијерархија знања приказује ток знања. На почетку, скупимо податке које се састоје од чињеница, симбола, тј. можемо рећи да податак нема контекст. Када податке ставимо у контекст, односно дамо им форму, добијемо информације. То би значило да информације представљају процесуиране податке, како би ти подаци постали корисни. Питања која постављамо како бисмо добили информације од података јесу: "Ко?", "Шта?", "Када?" и "Где?". Након тога следи интерпретација информација, односно апликација података и информација како бисмо добили одређено знање. Да би се од информација добило знање, потребно је поставити питање "Како?". Након што одговоримо на ово питање и постигнемо одређено знање, потребно је све у потпуности разумети и повезати са искуством. Када остваримо овај циљ, дошли смо до највише тачке пирамиде, односно мудрости.

-Уопштено о базама података

База података представља организовану збирку података. Сам термин је настао развојем рачунарске индустрије, а затим се проширио и на неелектронске базе података.

Најчешћа дефиниција база података гласи да је то збирка записа сачуваних у рачунару, тако да јој се рачунарски програм може обратити приликом одговора на задати проблем. Предмети враћени одговором на упите постају информације, које се могу даље користити за генерисање одлука, које би иначе биле много теже или практично немогуће за стварање. Рачунарски програм који се користи ради управљања и испитивања базе података, назива се Систем управљања базом података.

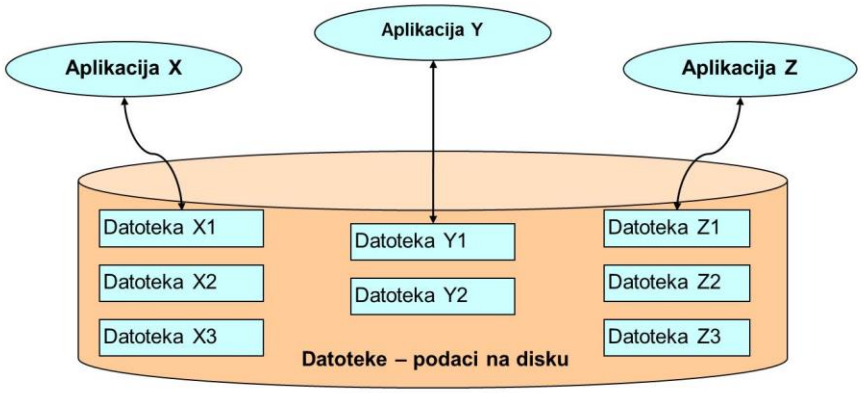
За неку базу података, најчешће се може направити структурни опис врсте података садржаних у тој бази, и тај опис се зове шема. Шема описује како предмете приказане у бази података, тако и саме односе међу њима. Постоје различити начини организовања шема, односно моделирања структуре базе података. Они се називају моделима база података или моделима података. Најраспрострањенији модел база података данас јесте релациони модел, који све информације приказује у облику вишеструких релационих табела од којих се сваки састоји од врста и колона. Овакав модел базе података приказује односе употребом вредности које су заједничке за више од једне табеле.

Назив "База података" се строго односи на збирку записа, а на софтвер којим се врше упити и управља базом података треба се односити као на "Систем управљања базом података". Међутим, када је контекст недвосмислен, многи администратори и програмери база података ипак користе термин "База података" у оба случаја.

-Примери обраде података

1. Класична обрада података

Код класичне обраде података (слика 3), за чување и обраду веће количине података, користи се систем датотека (фајл) код кога се подаци чувају, обрађују и претражују у скупу неповезаних података.



Слика 3 – Класичан систем обраде података

Подаци неповезани међу собом доводе до низа недостатака које ова врста обраде са собом повлачи. Неки од њих су и:

1) Зависност између програма и датотека (сваки програм мора да познаје детаљан опис датотека) – уписивање новог поља у запис захтева и промену у коду програма за обраду ове датотеке, чак иако програм тај нови запис не користи;

2) Редуданса података (појава истих података у различитим датотекама) – ова појава може довести до проблема при ажурирању база података, пошто се евентуалне измене података морају вршити на сваком месту на ком се исти подаци чувају, као и до повећања потребног меморијског простора;

3) Ограниченост дељења података;

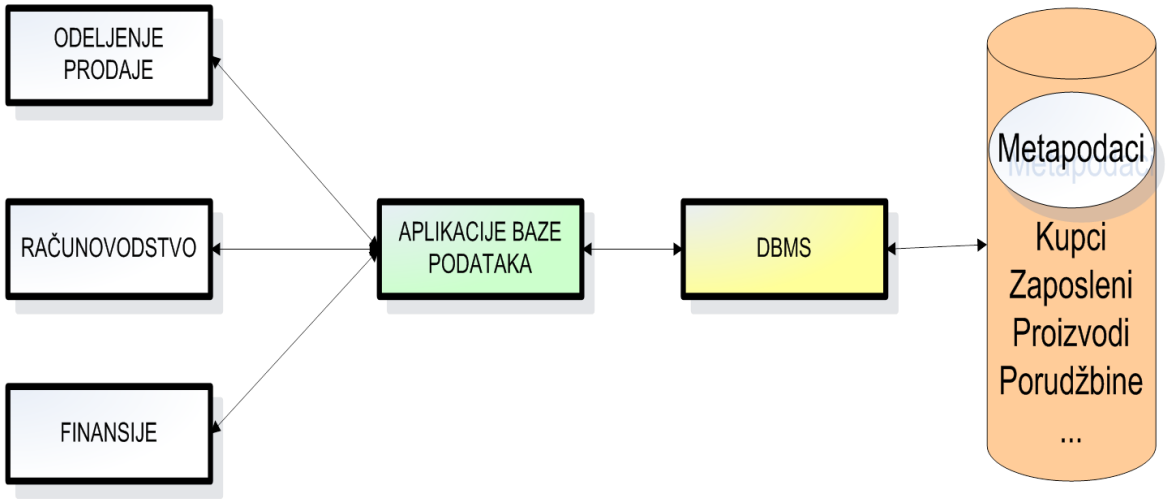
4) Дуго време за развој (нема наставка развоја – најлакше је почети из почетка);

5) Отежано одржавање програма (80% буџета се троши на одржавање) – односи се на ниску продуктивност у развоју информационог система (уколико се подаци налазе у различитим датотекама, или чак на различитим медијумима, потребни су велики програмерски напори како би се задовољио и веома једноставан информациони захтев)

1. Систем за управљање базом података (СУБП) – Database management system (ДБМС)

Представља софтверски систем ефикасан за унос, чување, претраживање и одржавање огромних количина података. Потенцира интеграцију и дељење података између свих одељења једне организације. Овај систем захтева потпуну промену у начину размишљања на свим нивоима управљања, јер подаци који су претходно чувани у више различитих датотека, сада су интегрисани у једну јединствену базу података.

Као и систем датотека, систем за управљање базом података такође омогућава чување великих количина података, независно од процеса који те податке користе, али, осим тога, он омогућава већу флексибилност коришћењем структура података који подржавају ефикаснији приступ веома великим количинама података.



Слика 4: ДБМС (Database management system) приступ базама података

Предности оваквог система јесу:

1) Независност између програма и података

- Одвајање метаподатака од апликација које користе податке
- Омогућен пренос података организације на друге рачунарске системе без потребе за променом програма

2) Минимална редуданса података

- Подаци су интегрисани у јединствену логичку целину
- Сваки податак се налази само једном у бази података

3) Побољшана конзистентност података

- Не постоји редуданса података и смањене су грешке

4) Побољшана размена података

- База података је ресурс целе организације
- Корисници имају различите погледе на јединствену базу података

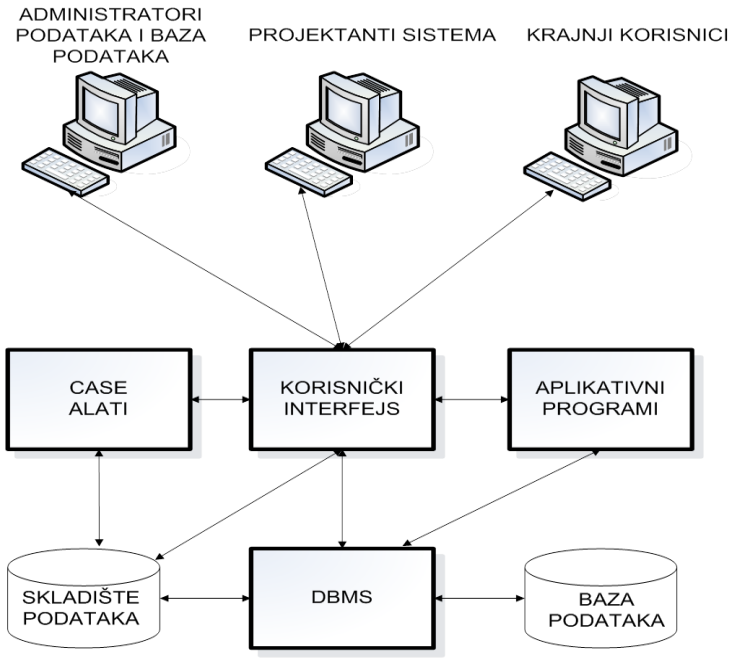
5) Повећана продуктивност у развоју апликација

- Смањење трошкова за развој нових апликација
- Програмери размишљају о функцијама, а не о детаљима описа података или имплементацији

6) Смањена потреба за одржавањем програма

- Могуће је независно променити формат података или апликацију.

7) Могућност опоравка базе података креирањем бекап копија.



Слика 5 – Типично окружење база података

-Практични примери база података

Под примером база података, могли бисмо навести систем за изнајмљивање филмова у видео клубу. База података би могла да садржи информације о филмовима, нпр. жанр, годину издавања филма, оригинални назив, назив преведен на српски језик, интерни редни број филма, тачно место на полици где се филм налази, тренутну доступност, медијум на ком се филм налази, као и информације о корисницима, односно члановима видео клуба: име, презиме, адреса, број телефона, итд. Систем би требало да обезбеди да се одређени филм може изнајмити само онолико пута колико копија филма има на стању, одговарајућу поруку уколико филм није доступан или уколико унесени корисник не постоји у бази података, или нпр. доступност информација о плаћеним и неплаћеним чланаринама за текући месец. Систем би такође требало да спречи губитак података услед нестанка електричне енергије или пада система.

Остали примери база података могу бити системи за резервацију авионских карата, банкарски, корпорацијски системи, као и системи за рад образовних установа.

-Развој база података

Развој база података се одвија у два потпуно супротна смера:

- Мањи системи, који имају тенденцију смањивања, како би се могли имплементирати у кућним рацунарима. Раније су сви системи база података захтевали велике рацунаре, јер су и меморијске јединице биле физички велике, док је данас могуце сацувати огромну колицину података и на обицном хард диску.
- Већи системи, који могу заузимати и неколико стотина гигабајта података, па чак и реда терабајта – то су системи великих компанија, у којима се не чувају само текстуални и нумерички записи, већ и слике, аудио и видео материјал, који могу заузимати огромне количине меморије, у зависности од потребе за квалитетом самог записа (уколико се ради рецимо о сателитским снимцима, један сат оваквог видео материјала може заузимати чак и више десетина гигабајта меморије, због потребе да видео материјал буде у високој резолуцији и јако високог квалитета, што наравно захтева и боље перформансе самих рачунара при обради толике количине података).

-Меморијске јединице за складиштење (storage units)

Примарна меморија је RAM (Random access memory). Међутим, као основна меморија за складиштење података, користе се секундарни меморијски уређаји – хард дискови, због потребе да се подаци сачувају и након евентуалног отказа система, из разлога што РАМ модули имају тај недостатак да се услед, рецимо, губитка електричне енергије, сви подаци изгубе због саме природе тих јединица (служе само за привремено чување података док се користе, и вишеструко су брже од хард диск-ова, због бржег приступа подацима који су тренутно у употреби). Веће базе података захтевају и већи меморијски простор, за шта се могу употребити и терцијалне меморијске јединице – CD и DVD медији и у последње време Blu-Ray медији, на којима се на физички малом простору могу сместити и више десетина гигабајта података. Терцијалне меморијске јединице се такође користе и за бекап читавих база, да би се у случају отказа појединих секундардних јединица могло повратити највећи део изгубљених података. Мана ових меморијских јединица је та што им је приступ доста спорији и од примарних и од секундарних јединица.

Наравно, сем великих складишних простора, некада, у циљу убрзавања обраде великих количина података, није довољан чак ни један процесор, већ се врши паралелни рад више рачунара (или процесора) на једном проблему – паралелно процесирање.

Код савремених система, најчешће се користи приступ клијент-сервер. Код једноставнијих система, клијент врши упит, помоћу неког од qуеру језика (најчешће SQL), док сервер, на ком се налази читав систем управљања базом, обрађује упите и враћа клијенту одговоре.

На сложенијим системима, код којих упите истовремено врши велики број клијената, главни сервер је повезан вишеслојним апликацијама са помоћним, извршним серверима, који се брину о повезивању са базом, трансакцијама, као и повезивању са клијентским рачунарима и обради њихових упита и прослеђивању њихових упита главном серверу (бази).

-Модели база података

Базе података логички су организоване неким од модела података. Данашњи ДБМС углавном подржавају неки од наведених логичких модела:

- Релациони модел – илити односни модел је заснован на скупу табела које описују систем. И подаци и везе између података приказују се правоуганим табелама. Лако се математички описује, па без обзира што је релативно сиромашан и једноставан, најчешће се користи и најлакше се имплементира на рачунару.
- Мрежни модел – структура података овог модела може се описати дијаграмом структуре података. Састоји се од слогова, међусобно повезаних физичким везама. Сваки слог се састоји од скупа поља, која одговарају атрибутима, док свако поље садржи један податак (вредност атрибута).
- Хијерархијски модел – ово је специјални случај мрежног модела података. База је хијерархијски приказана стаблом или скупом стабала, чворови приказују типове записа, док однос надређени-подређени приказује везе између типова записа.
- Објектни модел података – представља модел инспирисан објектно-орјентисаним програмским језицима. База се састоји од објеката, који се састоје од сопствених података и операција за руковање тим подацима. Сваки објекат припада некој класи, а између класа постоје везе наслеђивања или међусобног коришћења операција.

Мрежни и хијерархијски модел су били у употреби у 60-тим и 70-тим годинама прошлог века, док је почетком 80-тих година почео да преовладава релациони модел, који је и дан данас највише у употреби. Прелаз са релационог на објектно-орјентисани модел се још увек није догодио, тако да се и данашње базе могу поистоветити са релационим моделом података.

-Језици за рад са базама података

Постоје различите категорије језика за комуникацију између Датабасе манагемент система и крајњег корисника, тј. апликације. Неке од тих категорија су:

- Језици за опис података (DML – Дата Десцриптион Лангуаге) – овим језицима се дефинишу подаци и везе између података. Служе пројектанту базе или самом администратору ради записивања шеме или погледа. Наредбе ових језика подсећају на наредбе за дефинисање сложених типова података у језицима као што су Пасцал, ЦОБОЛ, итд.
- Језици за манипулисање подацима (DML – Data Manipulation Language) – служи за успостављање везе између базе података и апликације. ДМЛ наредбе омогућују лако маневрисање по бази података и међу њих спадају и наредбе за упис, брисање, промена или читање података из базе.
- Језици за постављање упита (QL – Query Language) – они служе крајњем кориснику за претраживање базе по задатим критеријумима. QL језици подсећају на говорни енглески језик, њихове наредбе само одређују резултат који ћемо добити, а не и сам поступак за добијање резултата.

Оваква подела на 3 језика у модерно доба база података већ готово и да не постоји. Све више се користе језици који у себи имају интегрисана сва претходно три наведена језика за манипулисање базама података. Један од данас најчешће коришћених језика је SQL (Structured Query Language), који служи како за креирање упита, тако и за опис и манипулисање подацима.

Сви горе поменути језици нису програмски језици. Они нам јесу потребни за повезивање са базом података, али нису довољни како би се креирала апликација, која би могла нешто извршавати користећи податке из базе.

80-тих година прошлог века, постали су јако популарни језици који су служили искључиво за развој апликација које би радиле у спрези за базама података. То су такозвани 4ГЛ језици (4th Generation Languages) или језици четврте генерације. Били су доста продуктивнији од класичних програмских језика, али им је велика мана била то што су били нестандардизовани, из разлога што је сваки од њих био део неког софтверског пакета за рад са базама података.

Данас, развој највећег броја апликација за рад са базама података се врши у стандардним програмским језицима: Ц++, Јава, итд., и за интеракције са базом се користе унапред припремљене класе објеката, па су због овога довољно продуктивни, а резултујућа апликација се лако уклапа у веће системе и преноси са једне базе података на другу.

Готово сви данашњи софтверски пакети подржавају SQL и релациони модел база података, и углавном се испоручују готово за све оперативне системе, те се стога могу интегрисати у практично било коју платформу.

БАЗЕ ПОДАТАКА У ВЕБ ОКРУЖЕЊУ

Сам развој Веб-а у протекле две деценије довео је до развоја најразличитијих услуга доступних на Веб-у. Многе од Веб услуга се управо заснивају на подацима које се смештају у базама података. Неки од примера где би се овакве базе података могле користити биле би рецимо електронске продавнице или новинске куће, које пружају приступ великој количини ускладиштених података или чак производи за подршку директној комуникацији између два пословна субјекта (енг. B2B – business to business).

Апликације које се служе базама података постоје већ више од 30 година, и настале су давно пре настанка Веб-а. Многе од ових апликација су засноване на мрежним технологијама које су биле познате давно пре него што је Веб уопште и постојао. Такозвани *point-of-service* системи који омогућавају подизање новца из банке управо су класичан пример оваквих првобитних умрежених система за рад са базама података: у експозитурама банака постоје тзв. АТМ машине (или терминали), док се централној бази података приступа преко широкопојасне мреже. Прве овакве системе могле су да плате једино веће организације које су имале новца за специјализовану терминалску опрему или, у појединим случајевима, за изградњу сопствене мрежне инфраструктуре.

Са друге стране, данашњи Веб пружа веома јефтин, лако доступан и стандардизован начин умрежавања. Постоји јако велики број корисника Веб-а са стандардизованим Веб претраживачима који могу радити на различитим моделима персоналних рачунара, док је пројектантима доступан софтвер за Веб сервере који може да испуни захтеве и за учитавањем докумената и за извршавањем програма. Ради испуњавања оваквих захтева, прилагођено је, или пројектовано више програмских језика који користе Веб сервере и Веб протоколе за писање скрипти.

Овај рад бавиће се углавном повезивањем Веб-а са базама података. Стандардне апликације које раде са базама података углавном се повезују преко три апликациона логичка слоја:

- Први слој представља основу система и састоји се из система за управљање базама података – ДБМС (енг. *Database Management System*) и саме базе података
- Други слој је састављен од највећег дела логике апликације, која се најчешће реализује скриптама, које са једне стране комуницирају са системом за управљање базама података на страни Веб сервера, док са друге стране могу да декодирају и генеришу ХТМЛ код потребан за приказ података у корисниковом Веб претраживачу.
- На трећем слоју, на самом врху, налази се корисников Веб претраживач, који уједно служи и као интерфејс апликације.

Како се трослојна архитектура користи у великом броју Веб апликација, у раду ће се започети њеним разматрањем, након тога, исти ће се бавити основним Веб протоколима, а затим и сваким слојем појединачно као и њиховим компонентама. Такође, у раду ће бити поменута и апликација *"Hugh and Dave's Online Wines"* (енг. Хју и Дејв онлајн вина).

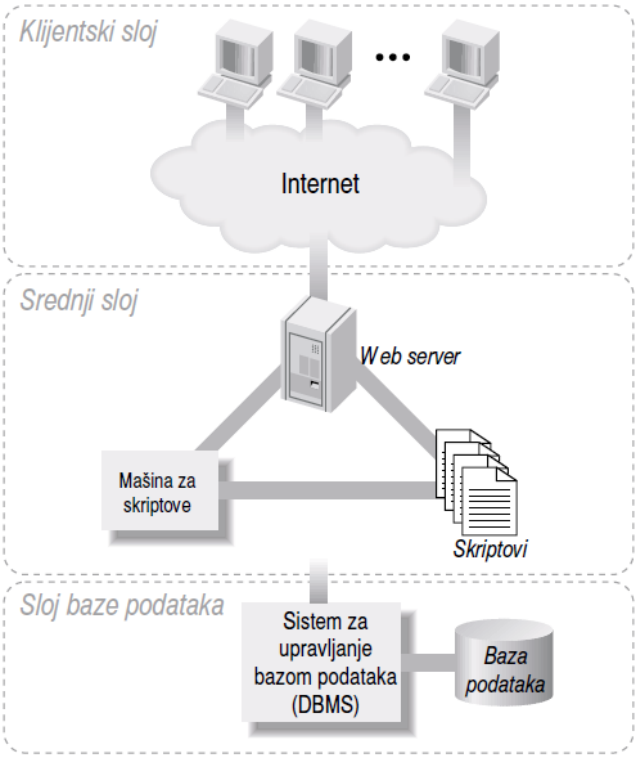
-Трослојна архитектура

Овај рад ће се бавити углавном Веб апликацијама које су дизајниране на принципу трослојне архитектуре.

На најнижем нивоу, налази се слој базе података. У његовом саставу су систем за управљање базама података ДБМС који управља базом чије податке корисници претражују, уносе, бришу или мењају (*слика 6*).

Изнад слоја базе података, налази се сложени средњи слој, који садржи највећи део апликационе логике и служи за пренос података између остала два слоја. На овом слоју се заснива комуникација и пренос података између система за управљање базама података на серверу и корисничке апликације (у овом случају Веб претраживача) на клијентском рачунару. На овом слоју се врши декодирање и генерисање XTML кода потребног за испис презентације на клијентском рачунару, односно потребног за пренос упита са клијентског рачунара на ДБМС.

На врху, односно, на трећем слоју се налази корисничка апликација, најчешће Веб претраживач, који представља интерфејс између корисника и апликације.



Слика 6 – Архитектура трослојног модела ВЕБ апликације повезане са базом података

Веб апликације које се могу описати трослојном архитектуром морају да имају могућност повезивања различитих врста софтвера и протокола, што се управо чини на другом слоју, због чега ће се највећи део овог рада посветити управо разматрању средњег слоја у Веб

апликацијама и апликационе логике која повезује клијента и суштински различите слојеве база података.

Коришћење речи Веб првенствено се односи на три основна, засебна стандарда на којима је он и заснован:

- ХТМЛ (енг. *Hyper-text markup language*) – језик за означавање хипертекста;
- ХТТП (енг. *Hyper-text transfer protocol*) – протокол за пренос хипертекста;
- ТЦП/ИП(енг. *Transmission control protocol/Internet protocol*) – пакет мрежних протокола

ХТМЛ је веома погодан за структуирање и приказивање података на Веб претраживачима. ТЦП/ИП протокол је ефикасан мрежни протокол који се бави преносом података између апликација на Интернету и има врло мало улоге у пројектовању Веб апликација. Прави проблем у ствари представља комуникација коришћењем ХТТП протокола са класичним базама података које раде на Веб-у. Да би се комуникација успешно одвијала и апликација сарађивала са базом без проблема, потребна је изузетно развијена и сложена апликациона логика.

-Протокол за пренос хипертекста (ХТТП)

Коришћење трослојне архитеткуре у изради Веб апликација обезбеђује суштински темељ за приступ тих апликација базама података, док сам Веб омогућује мрежу и протоколе који повезују трећи и средњи слој (клијентски слој и слој на ком се налази Веб сервер), односно, обезбеђује везу измрећу Веб претраживача на клијентском рачунару и Веб сервера. ХТТП компонента повезује све слојеве у трослојној архитектури. За програмере је од суштинске важности да разумеју какве проблеме ХТТП протокол може изазвати при пројектовању Веб апликација које раде са базама података. Веб претраживачи шаљу захтеве за ресурсима користећи ХТТП протокол, док Веб сервери шаљу одговоре на те захтеве.

ХТТП протокол омогућава дељење и прослеђивање ресурса на Веб-у. Посматрајући ситуацију из преспективе мреже, ХТТП представља апликациони слој, који се налази непосредно изнад ТЦП/ИП мрежних протокола. Већина данашњих Веб претраживача и Веб сервера користи верзију ХТТП/1.1. Поједини претраживачи и сервери још увек користе старију верзију ХТТП/1.0, али су углавном све апликације које подржавају ХТТП/1.1 компатибилне и са ХТТП/1.0.

Комуникације помоћу ХТТП протокола чине велику већину свих врста комуникација на Интернету. Давне 1997. године, помоћу ХТТП протокола се одвијало готово 75% укупног саобраћаја на Интернету, док се сматра да је тај проценат чак и порастао с обзиром на пораст броја апликација заснованих на ХТТП-у (нпр. услуге бесплатне електронске поште).

ХТТП је концептуално веома једноставан. Заснива се на томе да клијент пошаље серверу захтев за неким ресурсом, на шта сервер шаље одговор. ХТТП као одговор преноси захтевани ресурс – ХТМЛ документ, слику или излазне податке неког корисничког програма назад на интерфејс Веб претраживача (*слика 7*).

Слика 7 – ВЕБ претраживач шаље захтев на шта сервер одговара траженим ресурсом

ХТТП захтев који клијент шаље серверу би изгледа слично следећем примеру:

```
GET /index.html HTTP/1.0
From: milos@computer.org (Milos Dimitrijevic)
User-agent: Milos-fake-browser/version-1.0
Accept: text/plain, text/html
```

Наведени пример се састоји од текстуалног описа ресурса и додатних информација која се зову заглавља (енг. *headers*). У овом примеру се *GET* методом захтева страница *index.html* коришћењем ХТТП/1.0 верзије протокола, док се помоћу остала три реда заглавља врши идентификација корисника, Веб претраживача, као и типови података које читач (претраживач) може да прихвати. Захтев може да садржи и друга заглавља.

ХТТП одговор садржи код одговора и поруку, додатна заглавља, или чак захтевани ресурс. Пример одговора на претходни захтев, могао би изгледати слично следећем:

```
HTTP/1.0 200 OK
Date: Sun, 20 Jan 2016 13:44:25 GMT+1
Server: Apache/1.3.20
Content-type: text/html
Content/lenght: 88
Last-modified: Fri, 2 Nov 2015 11:40:03 GMT+1

<html><head>
<title>Test stranica</title></head>
<body>
<ht1>Ovo radi!</h1>
</body></html>
```

У првом реду одговора, сервер одговара да прихвата коришћење ХТТП/1.0 верзије протокола и потврђује да је успешно примио захтев исписивањем кода 200 и поруке ОК. Други чест одговор би могао бити *„404 Not Found”*, односно другим речима да ресурс није пронађен. У наредним редовима, приказани су тренутни датум и време, врста софтвера коју Веб сервер користи, тип послатих података, дужина одговора (мерена у бајтовима – енг. *byte*), као и подаци када је тражени ресурс последњи пут измењен, па затим, након једног празног реда, сам ресурс. У овом примеру је тражени ресурс ХТМЛ документ *index.html*.

Класичне апликације које раде са базама података чувају стање: корисници се прво пријављују, затим извршавају одговарајуће трансакције, и на крају, када обаве посао, одјављују се. На пример, апликација за банку захтева да се оператер на шалтеру прво пријави, затим да користи апликацију помоћу низа менија док опслужује захтеве текућег коминтента, да би се на крају дана одјавио. Апликација за банку има више стања: када се благајник одјави, он не може више да користи апликацију. ХТТП не чува стање. Кад кажемо да се стање на чува, то значи да је свака интеракција између читача Веб-а и Веб сервера независна од било које друге интеракције. Сваки ХТТП захтев читача Веб-а садржи идентичне информације у заглављу као што су идентификациони подаци корисника, типови страница које читач може да прихвати и инструкције за форматирање одговора. Чињеница да се стање не чува има и својих предности: најважнија је уштеда ресурса, пошто нема потребе да се на Веб серверу одржавају информације помоћу којих би се пратиле активности корисника, као и флексибилност која корисницима омогућава да се крећу између неповезаних страница или ресурса.

Пошто ХТТП не чува текуће стање, врло је тешко пројектовати Веб апликације које би чувале текуће стање. Неопходан је начин за одржавање страница у ХТТП-у тако да подаци протичу и може да се наметне одређена структура при употреби апликације. Често решење је размена жетона (енг. *token*) између читача Веб-а и Веб сервера помоћу којих се на јединствен начин идентификује корисник и његова сесија. Сваки пут када читач упути захтев за одређеним ресурсом, он шаље жетон, и сваки пут када се Веб сервер одазове, он враћа жетон читачу Веб-а. Софтвер у средишњем слоју помоћу жетона обнавља податке о кориснику од његовог претходног захтева (нпр. ком је менију у апликацији последњи пут приступано). Размена жетона омогућава да се апликацији додају структуре за које је неопходно чување стања, као што су менији, кораци и праћење процеса рада.

- **Лагани клијенти**

Пошто знамо да Веб апликације трослојне архитектуре по природи ствари нису погодне за ХТТП, могуће је запитати се чему онда уопште употреба таквог модела. Одговор првенствено треба тражити у предностима које пружају лагани клијенти. Читачи Веб-а су врло лагани клијенти: врло мали део логике апликације је укључен у клијенски слој. Читач само шаље ХТТП захтеве за ресурсима и приказује одговоре, који махом садрже ХТМЛ документе.

Трослојни модел значи да не морате да правите, инсталирате или подешавате клијентски слој. Сваки посетилац који има читач Веб-а може да користи Веб апликацију која

чита податке из базе, обично без потребе да инсталира додатни софтвер, да користи специфичан оперативни систем, или одређену хардверску платформу. То значи да апликација може да опслужи било који број различитих, географски расејаних корисника.

Алтернатива лаганом клијенту је наменски написан Јава аплет. Пример тежег клијента који се ипак може уклопити у трослојни модел је: корисник преузима аплет и извршава више апликационе логике на својој платформи него у случају лаганог клијента. Аплет и даље комуницира са средњим слојем који, пак представља интерфејс ка слоју базе података. У оваквом случају, предност је прилагођеност одређеној намени: уместо да се користи генерички читач, користи се наменско решење које елиминише многе проблеме око чувања стања, безбедности и недостатка флексибилности Веб-а. Аплет уопште не мора да користи ХТТП протокол за комуницирање са логиком апликације у средњем слоју.

Тежак клијент је део класичног двослојног решења, које је такође познато под називом клијент/сервер архитектура. Већина класичних апликација које приступају базама података (попут оних у банкама) има само два слоја. Клијенски слој садржи највећи део логике апликације, док је серверски слој, заправо, само ДБМС. Предност овакве архитектуре је то што могу да се пројектују наменска решења која потпуно задовољавају специфичне апликационе захтеве, без икаквих компромиса. Мане су недостатак флексибилности по питању хардвера и оперативног система, као и потреба да се сваком кориснику инсталира софтвер.

- **Клијентски слој**

Клијенски слој у моделу трослојне архитектуре обично је читач Веб-а. Читач Веб-а обрађује и приказује ХТМЛ ресурсе, шаље ХТТП захтеве за ресурсима и обрађује ХТТП одговоре. Као што је већ поменуто, употреба читача Веб-а као танког клијентског слоја пружа значајне предности, међу којима су једноставан развој и подршка за широк опсег различитих платформи.

На располагању је велики број читача Веб-а и сваки од њих има другачије особине. Најпопуларнији читач заснован на окружењу са прозорима јесте *„Internet Explorer”*. Нећемо описивати све могућности читача Веб-а, поменућемо само њихове заједничке одлике.

- Сви читачи Веб-а су ХТТП клијенти који шаљу захтеве и приказују одговоре добијене од Веб сервера (обично у неком графичком окружењу)
- Сви читачи тумаче садржај страница са ХТМЛ кодом када приказују страницу, тј., они корисницима приказују заглавља, слике, хиперлинкове, итд.
- Неки читачи приказују слике, репродукују филмове и звук и визуелизују друге типове објеката
- Многи читачи могу да извршавају *“Java Script”* код уграђен у ХТМЛ странице. *“Java Script”* се користи за нпр., проверу исправности података у ХТМЛ обрасцу *<form>* или промену изгледа и садржаја странице у зависности од корисникових апликација.

- Неки читачи Веб-а могу да покрећу компоненте развијене помоћу програмских језика *"Java"* и *"ActiveX"*. Те компоненте често дају додатне анимације, алатке које није могуће израдити помоћу ХТМЛ-а или друге, још сложеније могућности.

- Неколико читача може да примењује каскадне листе стилова (енг. *Cascading Style Sheets, CSS*) на ХТМЛ страницама како би управљали приказивањем ХТМЛ елемената.

Постоје мање (а некад и не баш тако мале) разлике у начинима на које поједини читачи Веб-а приказују ХТМЛ странице. Поједини читачи приказују искључиво текст а не приказују слике нити извршавају *"Java Script"* код, док су поједини у могућности да текст исписан на страници репродукују као звук, што омогућава приступ Веб-у особама са оштећеним видом.

Читачи Веб-а су најочигледнији пример корисничког агента, софтверског клијента, који захтева ресурсе од Веб сервера. У друге корисничке агенте спадају Веб Спајдер-и (аутоматизовани софтвер који крстари Веб-ом и проналази Веб странице) и посредничке оставе (кеш), софтверски системи који учитавају Веб странице и локално их складиште како би оне биле на располагању другим корисничким агентима.

- **Средњи слој**

У већини трослојних система база података на Веб-у, највећи део логике апликације налази се у средњем слоју. Клијентски слој приказује податке кориснику и прихвата податке које он уноси; слој базе података складишти и учитава податке. Средњи слој обавља већину преосталих задатака помоћу којих се обједињују остали слојеви: управља структуром и садржајем података који се приказују кориснику и обрађује податке добијене од корисника, претварајући их у упите за учитавање или уписивање у базу података. Такође, овај слој омогућава управљање стањем уз употребу ХТТП протокола. Логика апликације уграђена у средњи слој интегрише Веб са системом за управљање базом података (ДБМС).

У овом раду биће приказан апликациони модел у коме су компоненте средњег слоја Веб сервер, програмски језик за писање Веб скриптова и машина за извршавање скриптова (енг. *engine*). Веб сервер обрађује ХТТП захтеве и формулише одговоре. У случају Веб апликација, ти захтеви су обично упућени програмима који комуницирају са основним системом за управљање базом података. Као Веб сервер користиће се *"Apache HTTP server"*, производ организаије *"Apache Software Foundation"*. То је Веб сервер отвореног кода и користи се на више од 60% рачунара који су стално повезани на Интернет.

За писање скриптова који раде у средњем слоју користимо програмски језик *"PHP"*. Будући да је *"PHP"* настао као пројекат отвореног кода иза кога стоји *"Apache Software Foundation"*, не изненађује што је то најпопуларнији додатни модул за *"Apache HTTP server"*, са око 40% *"Apache HTTP server"*-а који подржавају *"PHP"*. *"PHP"* је изузетно прикладан за Веб апликације захваљујући алаткама за интеграцију Веб-а и окружења базе података. Изузетна флексибилност скриптова уграђених у ХТМЛ странице омогућава лаку интеграцију

са клијентским слојем. Подршка за интеграцију са слојем базе података такође је одлична захваљујући више од 15 библиотека које омогућавају повезивање са скоро свим познатим системима за управљање базама података.

- **Веб сервери**

Веб сервери се често називају и ХТТП серверима. Израз ХТТП сервер је добар сажетак функције коју обављају: њихов основни задатак је да открију пристизање ХТТП захтева из мреже, примају ХТТП захтеве које шаљу кориснички агенти (обично читачи Веб-а), обрађују те захтеве и враћају ХТТП одговоре који садрже тражене ресурсе.

У суштини, постоје два типа захтева које се шаљу Веб серверу: први, да се захтева слање одређене датотеке (обично статичке ХТМЛ Веб странице или слике) и други, када се захтева покретање програма и слање излазних података тих програма корисничком агенту.

Захтеви за покретање Веб скриптова који приступају бази података примери су ХТТП захтева када је неопходно да сервер покрене одређени програм. Помоћу софтвера који се користи у овом раду, упућиваћемо ХТТП захтеве за ресурсима типа *"PHP"* скриптова, након чега се покреће *"PHP Zend"* машина, која учитава и извршава скрипт, а потом прихвата његове излазне податке.

- **Употреба *"PHP"*-а за Веб скриптове**

"PHP" се наметнуо као компонента многих динамичких Веб апликација средњег и великог обима. То не значи да други језици за скриптове немају сопствене добре карактеристике због којих имају својих предности. Међутим, постоје многи разлози због којих је *"PHP"* добар избор, укључујући следеће:

- *"PHP"* је софтвер отвореног кода, што значи да је потпуно бесплатан. Због тог разлога, настојања програмске заједнице да га унапреди и одржава нису угрожена комерцијалним потребама

- У статичке ХТМЛ датотеке може се уградити један или више *"PHP"* скриптова, што значајно поједностављује интеграцију са клијентским слојем. Са друге стране, то може довести до мешања скриптова са презентацијом резултата; ипак, применом техника са шаблонима, може се решити већина наведених проблема.

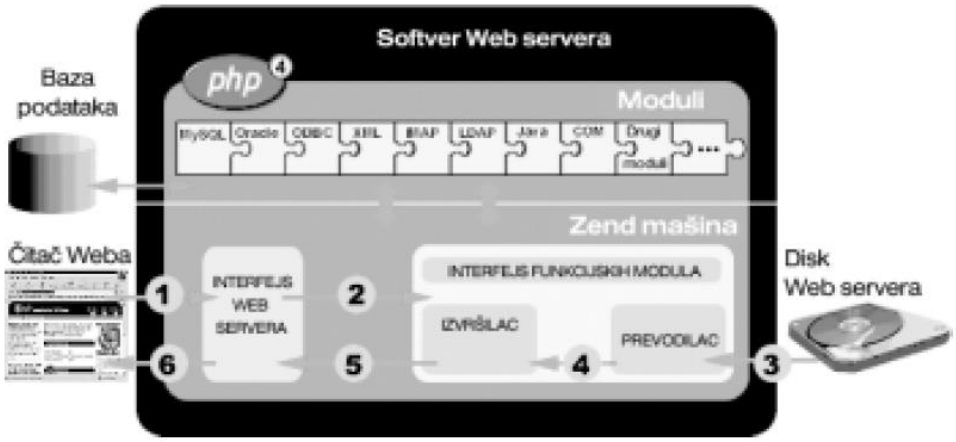
- Постоји преко 15 библиотека за основни, брз приступ слоју базе података.

- Брзо извршавање скриптова. Са новим побољшањима *„Zend"* механизма за обраду скриптова, извршавање је брзо и све компоненте раде унутар главног меморијског

простора *"PHP"*-а (за разлику од других окружења за извршавање скриптова, у којима су компоненте засебни модули). Емпиријски докази сведоче да *"PHP"* обавља задатке средње сложености брже од других познатих окружења за скриптове.

- Флексибилност по питању платформи и оперативних система. *"Apache"* ради на различитим платформама и на неколико оперативних система; *"PHP"* ради на свему томе, али и на многим другим када се интегрише са другим Веб серверима.

- *"PHP"* је подесан за развој сложених система. То је потпун програмски језик са преко 50 функцијских библиотека.



Слика 8 – Структура "PHP" окружења за извршавање скриптова

Када кориснички агент упути захтев Веб серверу за *"PHP"* скриптом, процес се одвија у следећих 6 корака:

1) Веб претраживач прослеђује захтев до Веб сервера интерфејсом *"Zend"* машине.
2) Интерфејс Веб сервера позива *"Zend"* машину и прослеђује јој параметре.
3) Машина преузима *"PHP"* скрипт са диска.
4) Преводилац преводи скрипте.
5) Извршилац покреће преведени код који може да садржи позиве функцијских модула. Излазни подаци извршиоца прослеђују се интерфејсу Веб сервера.
6) Интерфејс Веб сервера прослеђује излазне податке Веб серверу (који, на крају, прослеђује излазне податке корисничком агенту о облику ХТТП одговора).

Начин којим се управља и на који функционише *"PHP"* машина за извршавање скриптова зависи од начина на који је *"PHP"* модул укључен током инсталације *"Apache WEB server"*-а. Библиотека *"PHP"* модула је статички повезана са бинарном извршном датотеком *„httpd"*. То значи да се *"PHP"* машина за извршавање скрпитова учитава сваки пут када се покрене *"Apache"*, што као крајњу последицу има бржи рад *"PHP"* машине. Недостаци су то што *"Apache"* са статичком *"PHP"* библиотеком окупира више меморије него када се модул очитава динамички, као и то што је поступак надоградње модула мање флексибилан.

- **Слој базе података**

Слој базе података је основна Веб апликација која ради са базама података. Разумевање системских захтева, одабир софтвера за слој базе података, пројектовање базе података и израда слоја први су кораци успешног пројектовања апликација. Усредсредићемо се на компоненте слоја базе података и упознати са софтвером база података упоређујући га са другим техникама складиштења података.

У апликацији са трослојном архитектуром, слој базе података задужен је за управљање подацима. Управљање подацима обично подразумева складиштење и учитавање података, као и управљање податком ажурирања, при чему више процеса из средњег слоја може симултано, тј. истовремено да приступа бази, затим, заштиту података, очување њиховог интегритета и обезбеђивање услуга за подршку, попут израде резервних копија. У многим базама података за Веб све те послове обавља РДБМС и подаци бивају смештени у релационој бази података.

Управљање релационим подацима у трећем слоју захтева сложени РДБМС софтвер. На срећу, већина РДМБС-ова је пројектована тако да се сложеност софтвера не примећује. Да би се ефикасно користио ДБМС, потребно је да пројектант зна да пројектује базу података и да формулише команде и упите за ДБМС. У већини ДБМС-ова, језик за упите је *"SQL"*.

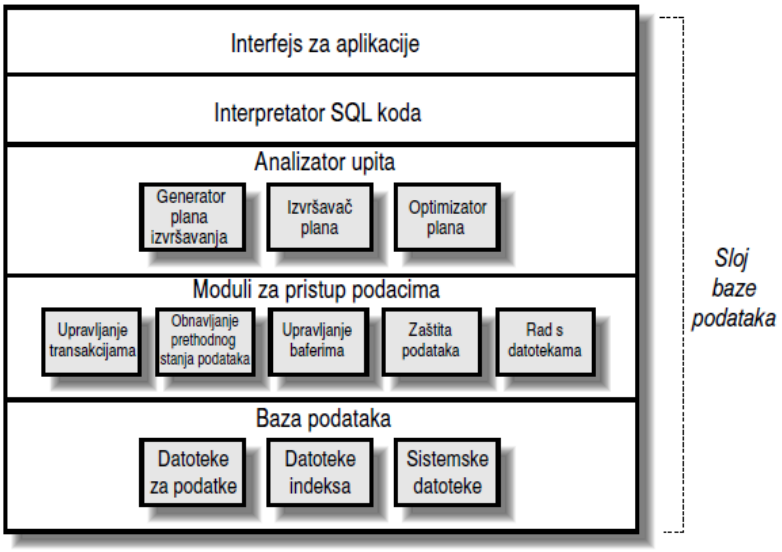
Већини корисника није потребно да разумеју унутрашњу архитектуру ДБМС-ова. За управљање подацима, у овом раду ће се користити *"MySQL RDBMS"*. Попут расправа о томе који програмски језик треба користити за скриптове у средњем слоју, расправља се и о томе који је ДБМС најприкладнији за одређену апликацију. *"MySQL"* има потпуно заслужену добру репутацију о брзини и посебно је добро пројектован за апликације где се подаци много чешће учитавају него што се ажурирају и за случајеве где су мања, једноставнија ажурирања уобичајена врста измене.. То су типичне карактеристике већине апликација које раде са базама података. Осим тога, попут *"PHP"*-а и *"Apache"*-а, *"MySQL"* спада у софтвер отвореног кода.

Постоје и друга нерационална ДБМС софтверска решења за складиштење података у слоју базе података. ту спадају машине за претраживање, системи за управљање документима и једноставнији мрежни сервиси, попут софтвера за ел. пошту.

ДБМС се састоји од неколико компонената:

1) Интерфејс за апликације – библиотеке за комуникацију са ДБМС-ом. Већина ДБМС-ова има једноставан преводилац у облику командне линије који често користи те библиотеке за преношење захтева унетих са тастатуре до ДБМС-а и за приказивање одговора. У Веб апликацијама заснованим на коришћењу база података, преводилац у облику командне линије обично је замењен функцијском библиотеком која је део језика за скриптове средњег слоја.

2) Интерпретатор *"SQL"* кода – синтаксни анализатор проверава синтаксу прослеђених упита и преводи их у интерни приказ (*слика 9*)



Слика 9 – Структура типичног ДБМС-а

3) Анализатор упита – генерише разне планове извршавања упита на основу статистике базе података и њених својстава, одабира један од планова и преводи га у акције које извршава на ниском нивоу.

4) Приступ подацима – у модуле који управљају приступом подацима ускладиштеним на диску спадају модул за управљање трансакцијама, модул за управљање поступком обнављања стања, модул за управљање бафером главне меморије, модул за заштиту података и модул за управљање приступом датотекама.

5) База података – то су заправо, само подаци физички смештени у датотеке. Подаци поред тога обухватају и индексне датотеке за брз приступ, као и статистичке податке о бази података и систему, који се првенствено користе за генерисање и оптимизовање планова за извршавање упита.

Пројектантима база података за Веб, битне компоненте су база података и апликациони интерфејс. Обично није потребно познавање и конфигурисање осталих компоненти ДБМС-а, сем када се ради о обимној апликацији.

- **Зашто треба користити ДБМС?**

Питање које се често поставља гласи: зашто користити сложени ДБМС за управљање подацима? Постоји неколико разлога. Они се могу објаснити поређењем базе података са програмима за унакрсне прорачуне, једноставном текстуалном датотеком или наменски изграђеним методом складиштења података.

Ако се као пример узме програм за унакрсне прорачуне, радни листови тог програма обично су дизајнирани за одређене апликације. Ако два корисника складиште имена и адресе, врло је вероватно да ће организовати податке на различит начин (зависно од потребе), и развити методе за сумирање и премештање података. У оваквој ситуацији програм и подаци нису независни; премештање колоне може значити и преписивање преко макроа или формуле, а размена података између апликација са којима раде та два корисника може бити сложена. Насупрот томе, ДБМС и база података обезбеђује независност датотеке и програма, где метода складиштења, редослед ускладиштених података и начин управљања подацима на диску не зависе од софтвера који им приступа.

Управљање сложеним везама је прилично тешко у прорачунским табелама или текстуалним датотекама. На пример, узевши продавницу вина у обзир: ако је неопходно складиштити податке о купцима, може се одвојити неколико колона у табели да би се сачувала кућна адреса сваког купца. Уколико је неопходно да се дода адреса на послу или поштанска адреса, биће неопходно још колона и сложена обрада да би се, на пример, послало писмо купцима. Да би се ускладили подаци о куповини коју су обавили купци вина, прорачунска табела биће шира и појавиће се проблеми. Тешко је, на пример, одредити максималан број колона потребних за складиштење поруџбина и пројектовати метод за њихову обраду и генерисање извештаја.

Прорачунске табеле и текстуалне датотеке не функционишу добро када су ускладиштени подаци спојени или повезани. Насупрот томе, ДБМС-ови су пројектовани за управљање сложеним релационим подацима. ДБМС-ови такође представљају комплетно решење: ако се користи ДБМС, неће бити потребе за решењима са наменски пројектованим прорачунским табелама или датотекама. Методе приступа подацима (најчешће помоћу језика за упите *"SQL"*) не зависе од врсте физичког складиштења и начина обраде података.

ДБМС обично дозвољава вишекориснички рад. ДБМС-ови средњег и великог обима могу системски контролисати уписивање података више различитих корисника. Насупрот томе, само један корисник може да отвори прорачунску табелу и да у њу уписује податке. Ако неки други корисник отвори ту исту табелу, неће моћи да види измене које у то време уноси први корисник. У најбољем случају, прорачунске табеле или текстуалне датотеке са дељеним приступом дозвољавају врло ограничен истовремени приступ.

Додатна предност ДБМС-а јесте његова брзина. Није у потпуности тачна тврдња да база података омогућава много брже претраживање података у односу на програме за унакрсне прорачуне или наменски систем датотека. У многим случајевима, претраживање у прорачунској табели или датотеци посебне намене може бити савршено прихватљиво или чак брже ако су оне пажљиво пројектоване, а количина података мала. Међутим, за управљање великом количином података, фундаментална структура ДБМС-а за претрагу обезбеђује брзо претраживање, а ако су потребе за подацима комплексне, ДБМС може да оптимизује методу њиховог проналажења.

Постоје и друге предности ДБМС-а, укључујући и заштиту података и права приступа у зависности од корисника, софтвер за администрирање, и подршку за рестаурацију података. Практична добит је у скраћеном времену развоја апликација: систем је већ изграђен, а потребни су само подаци и упити за приступ подацима.

- **Примери када треба користити ДБМС**

ДМБС треба користити за управљање подацима у следећим случајевима:

- Ако постоји више корисника који морају да приступају подацима у исто време;

- Ако постоји барем осредња количина података. На пример, у примеру у овом раду ће бити потребе за одржавањем података о неколико стотина купаца;

- Ако постоје везе између ускладиштених ставки. На пример, у продавници на Интернету могу постојати подаци о купцима, поруџбинама, попису робе и други подаци;

- Ако за податке постоје ограничења која се морају стриктно поштовати, попут дужине поља, типова поља, јединствених бројева купаца, итд.;

- Ако постоји потреба да се нови или збирни подаци изводе из основних, сродних података; то значи да подаци морају да се филтрирају да би се изводили извештаји и резултати;

- Ако постоји велика количина података која се мора брзо претраживати;

- Ако је важна безбедност, тј., ако постоји потреба да се задају правила ко може да приступа подацима;

- Ако је додавање, брисање или измена података сложен поступак.

- **Примери када не треба користити ДБМС**

Постоје ситуације када релациони ДБМС није потребан или је неподесан. Ево примера:

- Постоји само један тип ставки података и подаци се не претражују. На пример, ако се евидентира када се корисник пријави на систем и одјави са њега; додавање записа на крај једноставне текстуалне датотеке сасвим је довољно.

- Управљање подацима је једноставно. У том случају, подаци се могу убацити у Веб скрипт из средњег слоја, уместо да се систем оптерећује приступањем бази података сваки пут када су подаци потребни.

- Подаци захтевају сложену анализу. Програмски пакет за унакрсне прорачуне или статистички софтвер могу бити далеко погоднији за анализу.

- ***"MySQL"* ДБМС**

"MySQL" је ДБМС средњег обима, са већином одлика које имају системи великог обима и могућношћу да управља великом количином података. Пројектован је тако да је савршено погодан за управљање базама података које се обично користе у динамичким Веб апликацијама.

Разлика између *"MySQL"*-а и неких других система јесте у томе што *"MySQL"* не подржава поједина филтрирања и пружа ограничене могућности истовременог управљања подацима у вишекорисничком окружењу. То значи да десетине процеса из средњег слоја могу да приступају бази података у исто време, али не и стотине процеса.

Ограничења *"MySQL"*-а обично врло мало утичу на развој Веб апликација. Ипак, за системе са великим протоком података, великим бројем истовремених корисника или апликацијама које често ажурирају базу података, најбоље је размотрити коришћење других система.

- ***"SQL"***

"SQL" је стандардан језик за интеракцију са релационим базама података. Скоро сви системи релационих база података, укључујући и *"MySQL"* подржавају и *"SQL"* као алатку за израду, заштиту и претраживање базе података, као и управљање њоме. *"SQL"* је много више од обичног језика за упите; то је у потпуности усавршена алатка за све аспекте одржавња базе података.

"SQL" је имао компликован живот. Настао је раних седамдесетих година у *"IBM"*-овој истраживачкој лабораторији у Сан Хозеу, где је био познат под називом *"Sequel"*; неки корисници и даље га тако зову, иако је исправније користити акроним од три слова *"SQL"*. Након скоро 16 година развоја и неусаглашених реализавија, организације за стандарде *"ANSI"* и *"ISO"* установиле су 1986. године *"SQL"* стандард. Годину дана касније, *"IBM"* је установио другачији стандард.

Од средине осамдесетих, *"ANSI"* и *"ISO"* су установили три наредна стандарда. Први, *"SQL-89"*, најраспрострањенији је и у потпуности имплементиран *"SQL"* стандард у већини познатих база података. Многи системи имплементирају само неке могућности наредних издања, *"SQL-2"* или *"SQL-92"*, а скоро ниједан систем нема реализоване могућности најновије верзије стандарда, *"SQL-99"* или *"SQL-3"*.

Овај рад ће бити усредсређен на могућности на којима је заснован *"MySQL"* ДБМС. *"MySQL"* подржава основни ниво стандарда *"SQL-92"*.

- ***"SQL"* компоненте**

"SQL" се састоји из четири основне компоненте:

- Језик за дефинисање података (енг. *"DDL"*) - *"DDL"* је скуп *"SQL"* команди које формирају и бришу базу података, додају и уклањају табеле, форматирају индексе и уносе измене на сваком од побројаних елемената. *"DDL"* команде се углавном користе само током израде базе података. Индекси су структуре за брз приступ подацима и за њихово брзо ажурирање.

- Језик за манипулисање подацима (енг. *"DML"*) - *"DML"* је скуп команди које раде са ДБМС-ом и базом података. У *"DML"* команде спадају команде за претраживање, уметање и брисање података. Те команде служе за интеракцију са базом података током њеног уобичајеног коришћења.

- Управљање трансакцијама - *"SQL"* садржи команде за означавање групе команди као целине или трансакције. Коришћењем ових алатки, трансакције се могу опозвати или поништити.

- Напредне могућности - *"DML"* и *"DDL"* пружају напредне могућности за уграђивање *"SQL"*-а у програмске језике опште намене (у многоме налик на начин на који се *"SQL"* команде уграђују у *"PHP"*) и дефинисање приказа постојећих података за специјалне намене, као и додељивање и одузимање права приступа ДБМС-у и бази података. Они такође садрже команде за обезбеђивање интегритета, тј., команде које обезбеђују исправност података и поштовање релационих ограничења.

Од ове четири компоненте, у примеру датом у овом раду ће се користити свега две: *"DML"* и *"DDL"*.

Основни принципи на којима се заснивају базе података, практично су приказани апликацијом *„Hugh and Dave's Online Wines"*. У даљем тексту, ова апликација биће називана *продавницом вина.*

Апликација *продавница вина* има многе компоненте типичне за Веб апликације које раде са базама података, укључујући:

- Веб странице испуњене подацима из базе;

- претраживање помоћу упита, при чему корисник задаје ограничавајуће параметре за претраживање базе података;

- унос података и проверу њихове исправности; ХТМЛ обрасци прихватају податке, док *"Java Script"* код на клијентској страни и *"PHP"* скриптови на серверској страни проверавају њихову исправност;

- праћење корисника, тј., употребу техника за управање сесијом које протоколу ХТТП-а додају чување стања;

- проверу идентитета корисника и управљање њиховим активностима;

- извештавање.

- ***"Hugh and Dave's Online Wines"***

"Hugh and Dave's Online Wines" представља фиктивну продавницу вина на Интернету. У овом одељку ће укратко бити описани циљеви и функције продавнице вина, а потом ће бити размотрени системски захтеви изведени на основу тога. Такође, биће представљене и техничке компоненте продавнице вина.

Продавница вина је отворена за јавност: корисници имају ограничен приступ систему, а да би могли да поручују робу, морају претходно да се учлане. Циљ Веб апликације је да буде привлачна, једноставна и употребљива; ипак пошто су је дизајнирала два компјутерска научника, није им пошло за руком да је учине привлачном. Технички циљлеви су много боље испуњени; продавница вина управља са преко 1000 врсти вина, информацијама о залихама и базом података са око 1000 купаца и њихових поруџбина.

Сваки корисник са читачем Веб-а, може да приступи Веб локацији, прегледа и претражује вина која постоје у залихама и погледа податке у њима. У податке о винима спадају име, година бербе, тип вина, сорта грожђа и у неких случајевима рецензија експерата. Анонимни корисник може да дода одабрана вина у своју корпу за куповину. Корисници апликације могу да буду већ учлањени, а приликом учлањења дају податке о себи као и у већини других мрежних продавница.

Да би купили вина, корисници морају да се пријаве тако што дају податке о чланству. Ако се корисник тек учланио, он аутоматски бива пријављен. Када изабере вино, корисник може да га поручи. Поруџбина се сместа отпрема а потврда се шаље е-поштом.

У позадини, систем омогућава пословођама магацина да у базу података додају нове испоруке вина. Администратор Веб локације такође може да у продавницу вина додаје нова вина, винарије, виноградарске регионе и остале податке. Постоје и ограничене могућности за израду извештаја.

- **Системски захтеви**

Следећи захтеви се обично могу сачинити из докумената о планирању, разговора са купцима, итд. Али наравно, у овом раду се не описује поступак софтверског инжењеринга, већ се приказују општи захтеви који чине основу поменутих практичних примера. Неки аспекти системских захтева су упрошћени, одређени аспекти трговинских радњи су изостављени, а неки детаљи су опширно описани и верно одражавају реалне ситуације.

Побројани захтеви су дати само као опште тезе. Комерцијална апликација у реалној ситуацији била би детаљно описана функционалним системским захтевима. Апликацији професионалног квалитета такође би била придружена пројектна документација у којој би била описана структура базе података, изглед екрана и токови података.

Ево прегледа функционалних и системских захтева:

- Продавница вина на Интернету првенствено је намењена за е-пословање, чији је циљ продаја вина;

- Систем не омогућава књиговођство, управљање залихама, обрачун плата, наручивање и сличне друге послове;

- Корисници могу да одаберу вина и да их додају у корпу за куповину. Корисници могу да купе ставке из својих корпи најкасније један дан након што су у корпу додали прву ставку. Корисници имају само једну корпу за куповину и могу је испразнити када год пожеле;

- Корисници Веб локације могу бити анонимни све до тренутка када пристану да купе ставке из своје корпе;

- Да би купили ставке из корпе, корисници морају да се пријаве на систем. Да би се пријавио, корисник мора да има налог. Да би имао налог, корисник мора да пружи следеће податке о себи: презиме, име, адресу (минимум један ред), град, поштански број, државу, датум рођења, адресу е-поште и шифру. Адреса е-поште служи као корисничко име за пријављивање на систем. Корисник може да пружи додатне податке о средњем слову, титули, затим ту су два додатна поља за адресу, као и места за податке о делу државе (покрајина, регион), број телефона и број факса;

- Када корисник купи вина, његова поруџбина се архивира;

- Корисник може да добије попуст на вредност поруџбине. Попуст се може одобрити одређеног дана, на одређену минималну количину, или редовном купцу;

- У поруџбину може бити укључена цена доставе, која се одређује зависно од адресе корисника и начина доставе. У начине доставе спадају достава поморским саобраћајем, обичном поштом и експрес поштом. Поруџбина може да садржи и напомену која је намењена компанији за доставе;

- Вина су по општој подели класификована на црна, бела, пенушава, слатка и ојачана. Вина поред тога имају и име, годину бербер и опис. Описи су необавезни текстови слободне форме у којима се обично оцењује вино, слично опису на етикети;

- Вина се праве од различитих сорти грожђа, као што су Шардоне, Семијон, Мерло, итд. Једно вино се може правити од више сорти, а редослед навођења сорти је битан. На пример, вино направљено од две сорте грожђа, Каберне и Мерло, различито је од Мерло-Каберне-а;

- Корисници могу да прегледају вина према врсти или пореклу;

- Одређено вино производи одређена винарија;

- Винарије имају описе, који се обично састоје од мишљења стручњака, као и број телефона и факса;

- Винарије се налазе у одређеним крајевима. Крајеви се налазе у областима. На пример, долина Бароса у јужној Аустралији – као и свака област –има опис и по могућству слику или мапу;

- Корпа за куповину је незавршена поруџбина која садржи ставке. Она може постати завршена поруџбина након што се корисник пријави на систем. Свака ставка у поруџбини садржи одређено вино, количину тог вина и цену по боци. Цена вина је увек цена боце вина, која је првобитно стављена у корпу за куповину, што је уједно и увек најнижа могућа цена на залихама;

- Корисник може да мења количину вина у корпи за куповину и ставке могу да се уклоне из корпе;

- Вина која се продају налазе се на залихама робе. Сваки запис залиха садржи датум приспећа и односи се на одређено вино. Залихе садрже количине разврстане по цени боце и цени гајбе. Може постојати више записа залиха за одређено вино; тада се они односе на различите испоруке које су у продавницу вина стигле различитог датума или имају различите цене;

- Корисницима се увек нуди најнижа цена са залиха сваког вина. Када корисник дода вино у корпу за куповину, њему је гарантована та цена;

- Корисник може да купи само она вина која су на залихама;

- Када корисник претвори своју корпу за куповину у поруџбину, проверава се да ли на залихама постоји довољно робе да се она може поручити. Ако нема довољно вина, корисник бива обавештен и количина у његовој корпи се ажурира; до тога може доћи ако корисник дода у корпу више вина него што има на залихама;

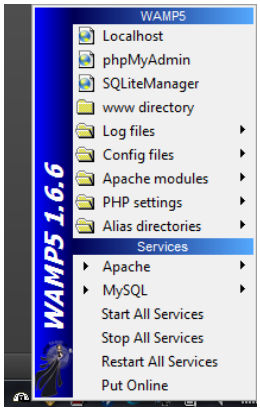
- Када постоји довољно робе да се обави поруџбина, количина вина на залихама умањује се након што се порруџбина оконча. На залихама се увек умањује најстарији запис одређеног вина.

ПРАКТИЧАН ПРИМЕР КОРИШЋЕЊА БАЗЕ ПОДАТАКА НА ВЕБ-У

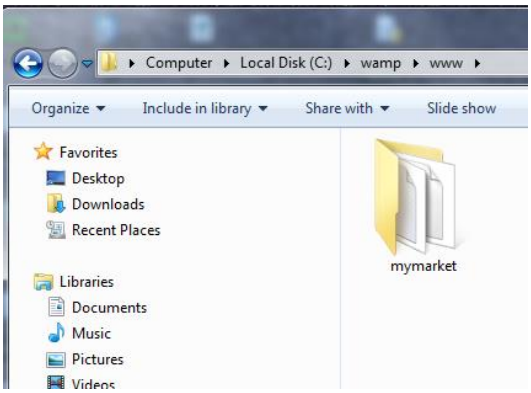
За овај рад, припремљен је као прилог и практичан пример коришћења базе података на Веб-у. Пример је урађен по угледу на винарију, налик на е-продавницу и бави се онлине продајеом беле технике. Веб сајт је рађен у *"PHP"*-у и користи *"MySQL"* базу података. У овом раду, неће се превише пажње давати на *"PHP"* код, из разлога што није тема самог рада, већ ће се више пажње обратити на креирање базе и интегрисање базе са *"PHP"*-ом у једну целину. За креирање овог примера, као и тестирање, коришћен је *"Wamp"* сервер, који служи за покретање *"PHP"* страница у локалу (енг. *localhost*) и може послужити за креирање *"MySQL"* базе података коју касније користимо у *"PHP"* коду.

- Инсталација *"Wamp"* сервера

Да би се овај рад привео крају и да би продавница могла да функционише ван мреже, потребно је инсталирати *"Wamp"* сервер. Коришћена је верзија *"Wamp"* сервера 1.6.6. Сама инсталација је интуитивна и треба само пратити инструкције на екрану како би се инсталација привела крају.



Слика 10 – "Wamp" сервер панел

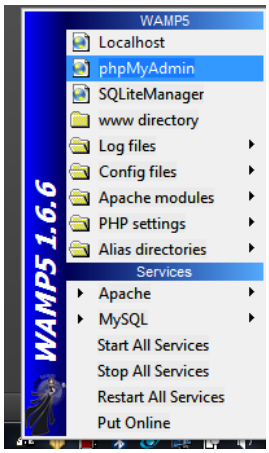


Слика 11– Директоријум у коме је смештена ВЕБ страна

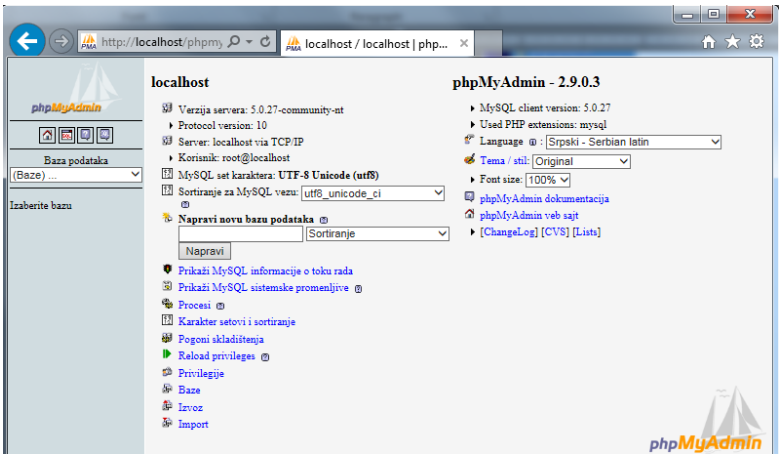
Уколико је при инсталацији за *"root"* директоријум Веб страна остављен *www* директоријум у директоријуму *"Wamp"*, ту ће бити смештен и комплетан пројекат који је обрађен у овом раду (у овом случају, директоријум – пројекат се зове *„mymarket"*).

- **Креирање базе података на *"Wamp"* серверу за Веб страну**

Први корак за креирање (односно, у овом случају *"import"*-овање) базе података јесте покретање *"Wamp"* сервера. Када се *"Wamp"* сервер покрене, појави се иконица у *"system tray"*-у. Потребно је кликнути на њу левим кликом, а затим отићи на *"phpMyAdmin"*(*слика 12*), и на тај начин се у Веб претраживачу отвори прозор за креирање привремен базе података (*слика 13*). Ово је неопходно урадити након инсталације *„Wamp"* сервера на било ком рачунару на ком Веб страна није пчокренута или након креирања нове *"MySQL"* базе (или потпуно нове Веб стране).

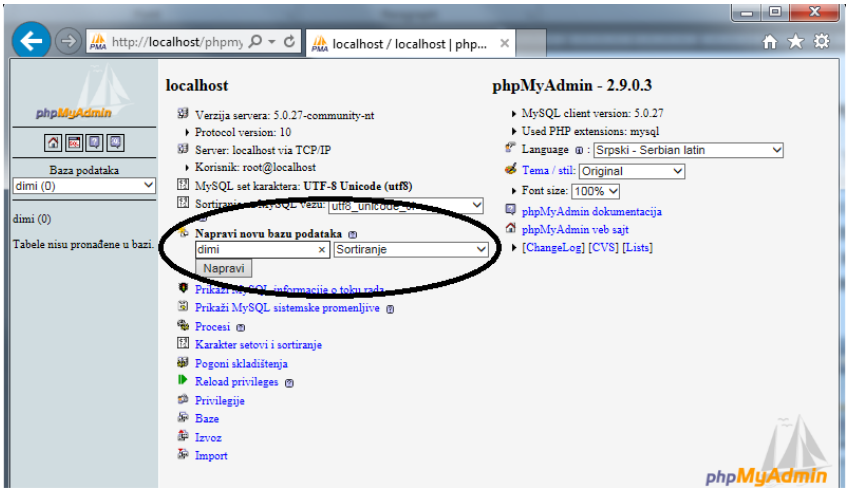


Слика 12 – "Wamp – phpMyAdmin"



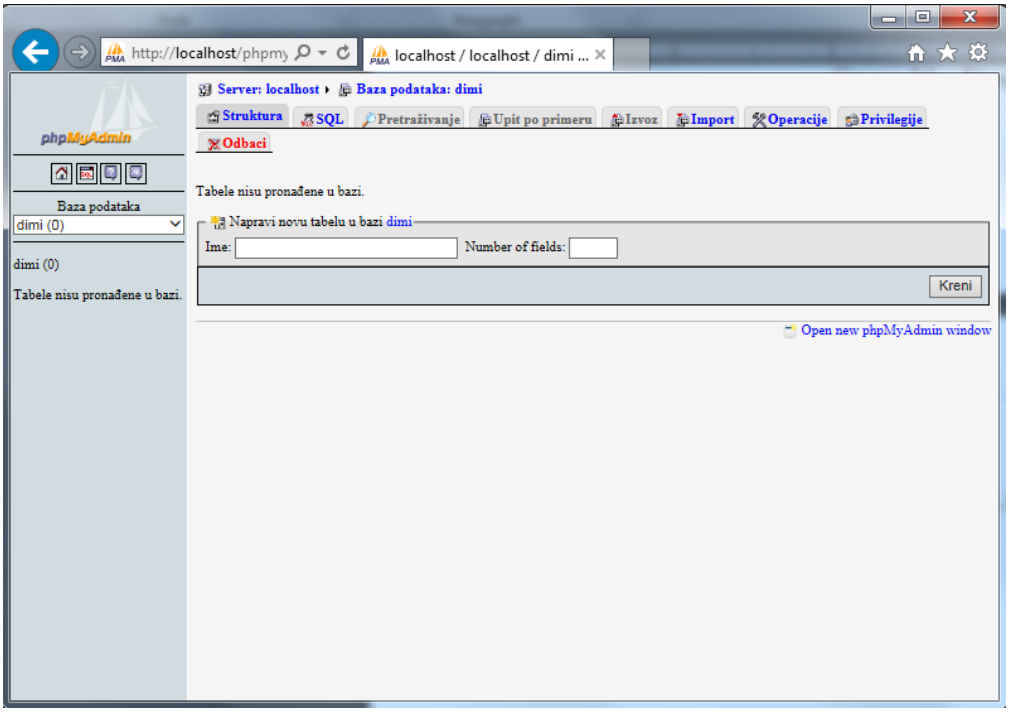
Слика 13 – Прозор "phpMyAdmin"

Са десне стране, у менију *"Language"* могуће је одабрати језик на ком ће страна *"phpMyAdmin"* бити исписана. Након одабира језика, у поље *„Napravi bazu podataka"* треба уписати име тренутне базе (у овом примеру користиће се база *"dimi"*, о којој ће бити више речи у каснијем тексту), а затим је потребно кликнути на дугме *„Napravi"*(*слика 14*).



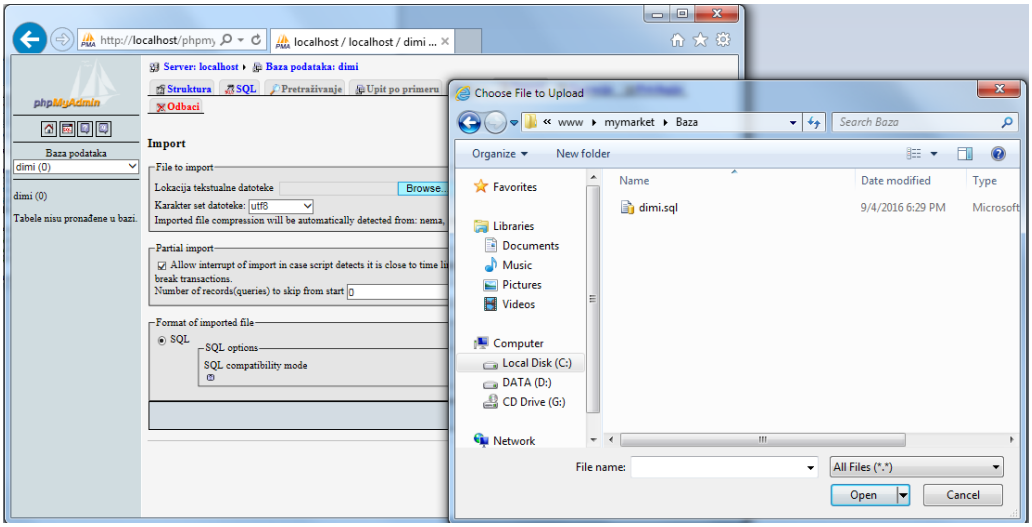
Слика 14 – Креирање базе података

Када се кликне на дугме *„Napravi“*, појављује се следећа страна:



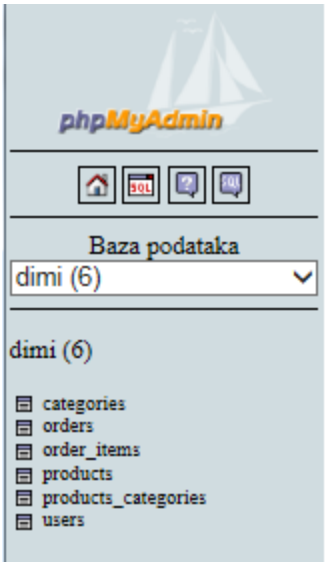
Слика 15 – Креирана база података

на којој је потребно кликнути на таб *“Import”*, одакле је кликом на *“Browse”* могуће *“Import”*-овати базу *“dimi”* која је креирана у току израде Веб стране и налази се на путањи “*root\Wamp\www\mymarket\Baza\dimi.sql*” (*слика 16)*:



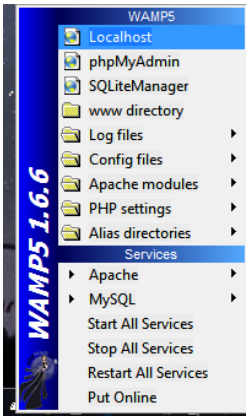
Слика 16 – “Import”-овање базе “dimi.sql” у тренутну “PhpMyAdmin” базу

а затим, кликом на дугме *"Kreni"*, *"MySQL"* база се I*"import"*-yje у тренутну базу са којом ради апликација. Овим се успешно креира база *"Dimi"*, што се може видети на следећој слици (односно на левој страни *"PhpMyAdmin"* Веб стране):



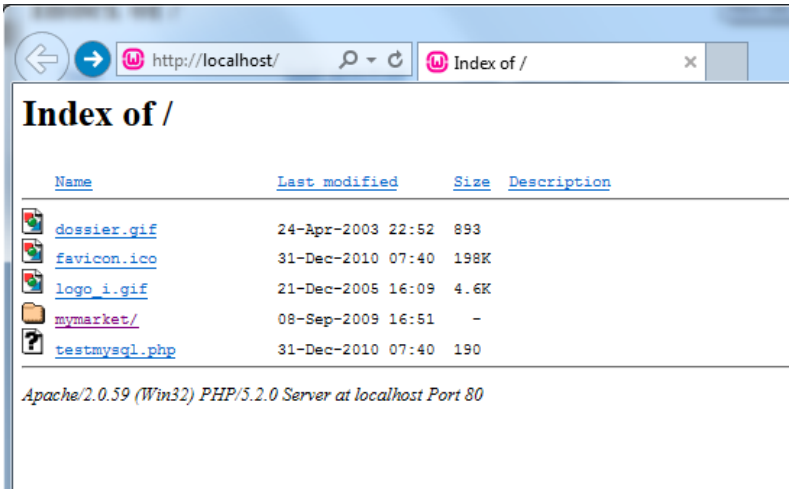
Слика 17 – Успешно креирана база "dimi" на "PhpMyAdmin" страни

Након успешно креиране базе података у *"PhpMyAdmin"*-у, поново је потребно кликнути на иконицу *"Wamp"* у *"system tray"*-у, а затим на *"Localhost"* (или, довољно је у навигационом менију у претраживачу откуцати *http://localhost/*) – слика 18:



Слика 18 – Отварање локалне ВЕБ странице

Следећи прозор који се отвара јесте Веб страна *"Wamp"* сервера ван мреже:



Слика 19 – "Localhost" страна

где, се са леве стране налази директоријум *"mymarket"* на који треба кликнути како би се отворила страна *"Bela Tehnika"*, чији код овај рад неће обрађивати јер није тема истог:

Слика 20 – ВЕБ страна "Bela tehnika.com"

- **Креирање *"SQL"* базе коришћене при развоју електронске продавнице**

У суштини, као и свака друга база података, и Веб база података која је креирана за потребе ове Веб стране, састоји се од табела. За потребе електронске продавнице беле технике, креирано је више табела, коришћењем *"SQL"* функција *"Create Table"*:

```
CREATE TABLE `categories` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `parent_id` int(11) NOT NULL DEFAULT '1',
  `name` varchar(25) NOT NULL DEFAULT '',
  `description` varchar(255) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `parent_id` (`parent_id`),
  KEY `name` (`name`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=7 ;
```

Користећи <CREATE TABLE 'categories'>, креирана је прва табела са називом *"categories"*, након чега се у загради декларишу промењљиве које представљају називе колона у табели, као што су *"id"*, *"parent_id"*, *"name"*, *"name"*, *"description"*. Након декларације сваке од ових „колона", неопходно је декларисати и тип уноса као и број карактера. Рецимо, за колону *"name"*, декларисано је да се ради о уносима типа „карактери", односно *"varchar"* (variable character), док унос у заградама (у случају колоне *"name"*, вредност је 25) означава максималну величину, односно максимални број карактера коришћених при уносу вредности у колону *"name"*. Као и за ову, и за све остале промењљиве, односно колоне, неопходно је декларисати како тип уноса, тако и максималну величину, односно број карактера и евентуално подразумеване (енг. *default*) вредности.

Такође, на крају, неопходно је декларисати и тзв. *"PRIMARY KEY"*, односно специфичну врсту индекса. *"PRIMARY KEY"* у свакој табели је обавезно јединствен за сваки унос у базу, тако да је за потребе ове табеле као примарни кључ одабран *"id"*, јер је при декларацији колоне *"id"* постављено да вредност уноса не сме бити *"NULL"* (што је неопходно да би се неко ограничење декларисало као примарни кључ), као и да се ради аутоматска инкрементација, тј., повећање сваког следећег уноса у ову колону за вредност 1, што значи да ће за сваки унос у ову табелу, вредност у колони *"id"* бити различит у односу на сваки претходни или наредни, што осигурава да за сваки одвојен унос података у базу постоји јединствен члан, на основу ког се идентификује сваки ред (односно унос) у табели.

Сем тога, након декларисања примарног кључа – а, декларисаће се и колоне које ће означавати индексе, односно кључеве. Индекси се користе за касније убрзање филтрирања, односно претраживања табела. Када вршимо упит базе ради проналажења поља која садрже експлицитно одређене вредности, она ће погледати у индексе како би пронашла жељену вредност. Овакво претраживање је веома брзо, јер су уноси сортирани по индексу, што

омогућава бинарно претраживање. Без индекса, база мора ишчитати сваки ред у табели како би пронашла жељену вредност, што прилично успорава процес претраживања.

Практично, неопходно је додати индекс свакој колони по којој се касније може јавити потреба за вршењем претраживања, односно филтрирања табеле. Са друге стране, међутим, уколико постоји сувише различитих индекса, без обзира колико корисно може бити при извршавању упита над базом, то може јако успорити креирање и брисање уноса у табелу, тако да је неопходно пазити да се креира онолико индекса колико је заиста неопходно – нешто што је потребно имати на уму уколико је планирано да се база користи за сталан упис, односно брисање уноса.

Након завршеног декларисања табеле, неопходно је навести који ће *"Engine"* база користити за складиштење, што је у овом случају *"InnoDB"*, коју *"MySQL 5.5*" и каснији користе подразумевано. Такође, потребно је означити и сет карактера који ће се користити за упис (у овом случају *latin1*).

Сличан поступак ће се користити и за креирање свих осталих неопходних табела, као што су:

Табела *"products_categories"*:

```
CREATE TABLE IF NOT EXISTS `products_categories` (
  `product_id` int(11) NOT NULL DEFAULT '0',
  `category_id` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`product_id`,`category_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Табела *"users"*:

```
CREATE TABLE `users` (
  `username` varchar(16) NOT NULL DEFAULT '',
  `password` varchar(32) NOT NULL DEFAULT '',
  `priv` varchar(5) NOT NULL DEFAULT '',
  `firstname` varchar(64) NOT NULL DEFAULT '',
  `lastname` varchar(64) NOT NULL DEFAULT '',
  `email` varchar(128) NOT NULL DEFAULT '',
  `phone` varchar(32) NOT NULL DEFAULT '',
  `address` varchar(255) NOT NULL DEFAULT '',
  PRIMARY KEY (`username`),
  UNIQUE KEY `email` (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Као што се може видети из декларације табеле *"products_categories"*, могуће је креирати и два примарна кључа у једној табели уколико је то неопходно. Рецимо, уколико *"category_id"* има једнаку вредност за два уноса, редови у табели ће се идентификовати помоћу *"product_id"* кључа и обрнуто. Практично, оваква конфигурација се користи у случају када је потребан композитни примарни кључ, тј., кључ који се састоји од више кључева који у пару никада нису исти за два различита уноса у табелу, док појединачни уноси једног од ова два кључа могу бити исти. Сликовит пример се може приказати на следећи начин:

manufacturer	***product_id (primary key)***	***category_id (primary_key)***
Gorenje	R6295W	Frizider
LG	3992SL	Frizider
Samsung	3992SL	Televizor

Табела 1 – Пример композитног примарног кључа

На овом хипотетичком примеру (*Табела 1*), рецимо, може се видети да је уносима треће колоне првог реда и треће колоне другог реда вредност *category_id* једнака, док је вредност друге колоне другог реда и друге колоне трећег реда једнака. Међутим, уколико се користи композитни примарни кључ који ће садржати и *product_id* и *category_id*, овакав конфликт се може избећи, јер се види да се у случају првог и другог реда, део кључа који представља *product_id* разликује један у односу на други, док у случају другог и трећег реда, јединствени део уноса представља вредност *category_id* кључа.

Даље, након декларисања самих табела и њених колона, неопходно је унети податке у табеле, и то коришћењем синтаксе:

```
INSERT INTO '<naziv tabele>' (<kolona1>, <kolona2>, <kolona3>, ...., <kolonan>) VALUES

(kolona1_vrednost1, kolona2_vrednost1, kolona3_vrednost1, ....., kolonan_vrenost1),

(kolona1_vrednost2, kolona2_vrednost2, kolona3_vrednost2, ....., kolonan_vrenost2),

(kolona1_vrednost3, kolona2_vrednost3, kolona3_vrednost3, ....., kolonan_vrenost3),

.

.

.

(kolona1_vrednostn, kolona2_vrednostn, kolona3_vrednostn, ....., kolonan_vrenostn)
```

Пошто је за поједине табеле унос података превише велика количина текста, може се издвојити из *"SQL"* кода базе *"Dimi"* која је креирана за потребе Веб стране електронске продавнице, свега пар примера практичног уношења података у табелу коришћењем *"SQL"* кода:

```
INSERT INTO `categories` (`id`, `parent_id`, `name`, `description`) VALUES
(1, 0, 'root', ''),
(2, 1, 'Frizideri', ''),
(3, 1, 'Masine za pranje vesa', ''),
(4, 1, 'Masine za susenje vesa', ''),
(5, 1, 'Masine za pranje sudova', ''),
(6, 1, 'Sporeti', ''),
(7, 1, 'Mikrotalasne rerne', ''),
(8, 1, 'Bojleri', ''),
(9, 1, 'Aspiratori', ''),
(10, 1, 'Dodatna oprema', '');
```

Као што се може видети из претходно наведеног примера, веома је битно водити рачуна о синтакси. Рецимо, када се додају вредности у колоне које су декларисане као *"Integer"* (што је у овом примеру декларисано при креирању табела), довољно је само уписати целобројну вредност. Међутим, уколико је колона декларисана као *"Varchar"*, важно је водити рачуна да се вредност, односно карактери уписују између једноструких наводника (*'vrednost'*).

Очигледно је са горњег примера на који се начин упис података врши, а то је ред по ред, односно, потребно је унети вредности сваке колоне појединачно, пре него што се пређе у следећи ред.

Други пример би могао бити унос података у табелу са наруџбеницама (*"order_items"*):

```
INSERT INTO `order_items` (`order_id`, `product_id`, `price`, `qty`) VALUES
(1, 1, 60.00, 2),
(2, 1, 60.00, 4),
(3, 6, 135.00, 1),
(4, 6, 135.00, 1),
(5, 18, 614.38, 1),
(6, 8, 186.00, 1),
(7, 21, 828.00, 1),
(7, 30, 290.00, 1),
(8, 21, 828.00, 1);
```

где колоне представљају:

- ID наруџбенице
- ID производа

- цена производа
- количина наручених производа

Упитом овакве табеле интегрисањем *"SQL"* кода, у рецимо *"PHP"*, могу се издвојити подаци о појединачним наруџбеницама уколико је потребно, на пример, исправити количину наручених производа, где би сам *"PHP"* вршио упит базе повезивањем на исту и то функцијом *"sqlsrv_connect"*:

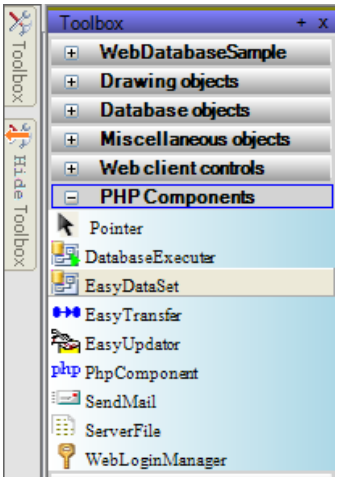
```
$serverName = "(local)";

$connectionOptions = array("Database"=>"ime_baze");

/* Connect using Windows Authentication. */

$conn = sqlsrv_connect($imeServera, $opcijeKonekcije);

if ($conn === false)

{ die( FormatErrors( sqlsrv_errors())); }
```

- ***"Limnor Studio"* – практичан пример**

Уколико за програмирање свог пројекта користимо *"Limnor Studio"*, повезивање на базу података може се вршити коришћењем "*SQL"* скрипте:

```
CREATE TABLE IF NOT EXISTS `student` (

  `StudentId` int(11) NOT NULL AUTO_INCREMENT,

  `Firstname` varchar(30) CHARACTER SET utf8 DEFAULT NULL,

  `Lastname` varchar(30) CHARACTER SET utf8 DEFAULT NULL,

  `BirthDate` datetime DEFAULT NULL,

  `Grade` int(11) DEFAULT NULL,

  PRIMARY KEY (`StudentId`)

)
```

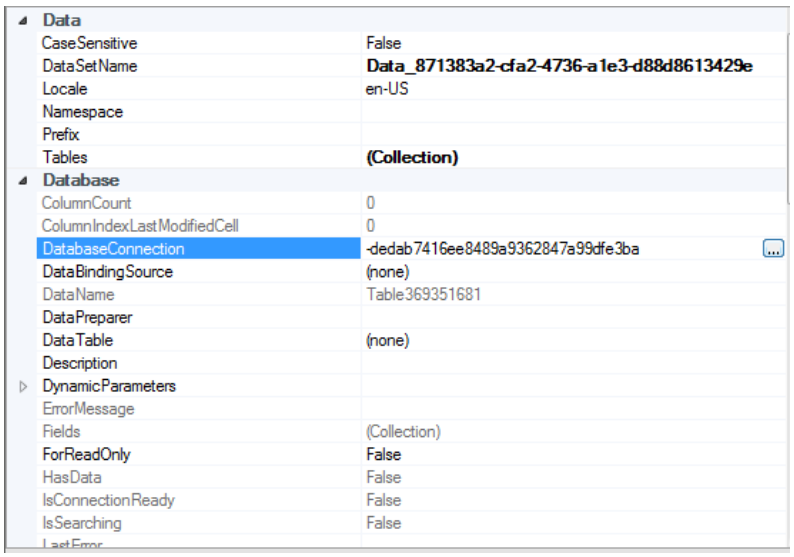
или простим коришћењем компоненте *"EasyDataSet"* из *"Toolbox"*-а (*слика 21*)*:*



Слика 21 – "EasyDataSet"

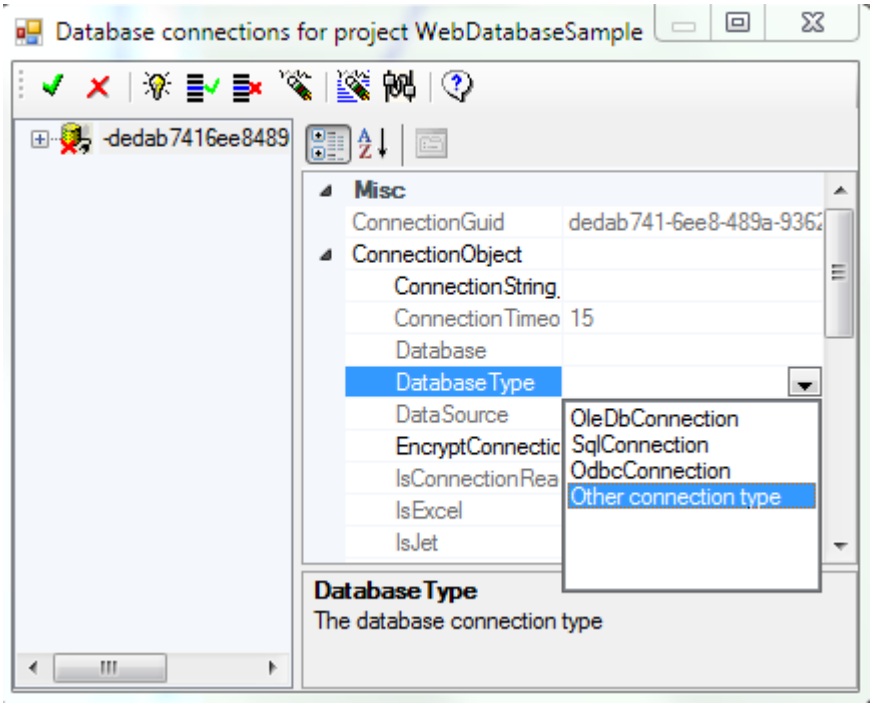
Неопходно је разумети да се *"EasyDataSet"* може користити у свим врстама апликација: како у *"Windows Forms"*, тако и у *"PHP"*, тако и у *".NET"* Веб апликацијама. Конфигурација ове компоненте је потпуно иста, без обзира о каквој врсти пројекта се ради.

Након одабира *EasyDataSet* компоненте, неопходно је сетовати *"Database connection"* својство (*слика 22)*:

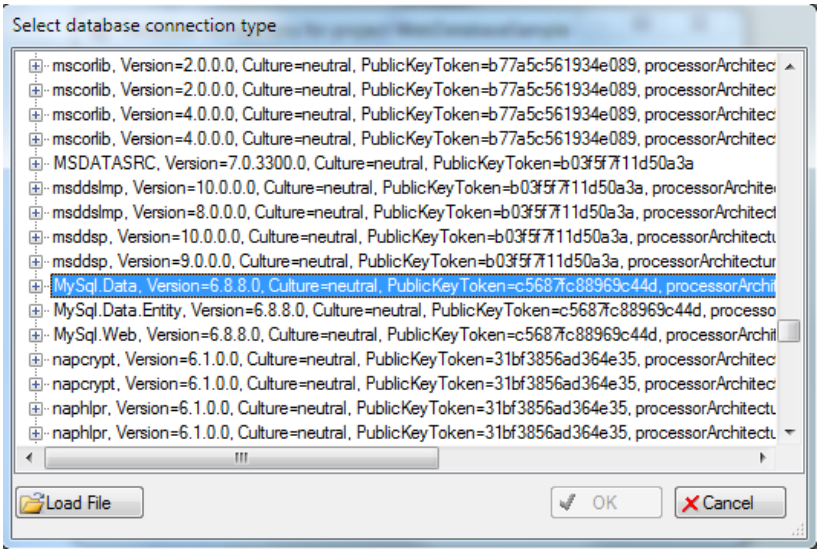


Слика 22 – "EasyDataSet" својства

након чега треба изабрати као тип конекције на базу *"Other connection type"*:



Слика 23 – "EasyDataSet – Database type"

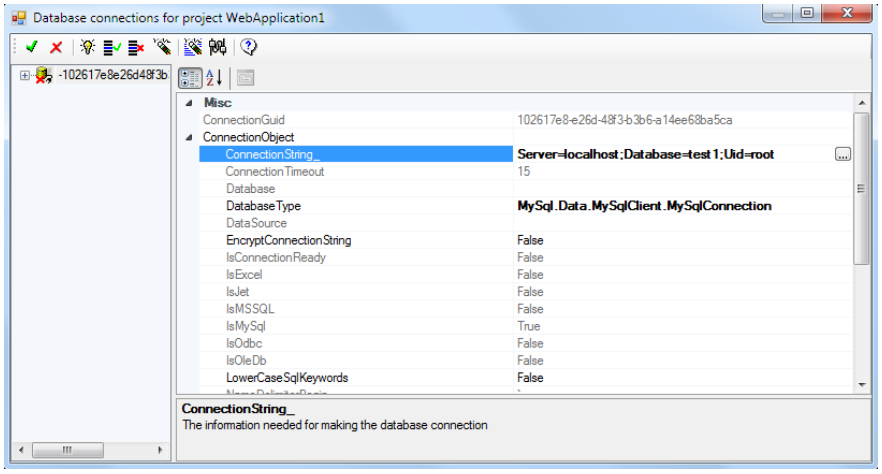


Слика 24 – "EasyDataSet – Database type – Database connetion type"

Када се одабере тип конекције, неопходно је подесити *"Connection string"* својство тако што се упише име *"SQL"* сервера, име базе која се користи, као и корисничко име и шифру који се користе за повезивање са базом података. Уколико се претпостави да ће се *"SQL"* сервер налазити на истом рачунару као и апликација која се пројектује, као име сервер се може користити просто *localhost*, док са друге стране, уколико "*SQL Server"* сервис већ постоји

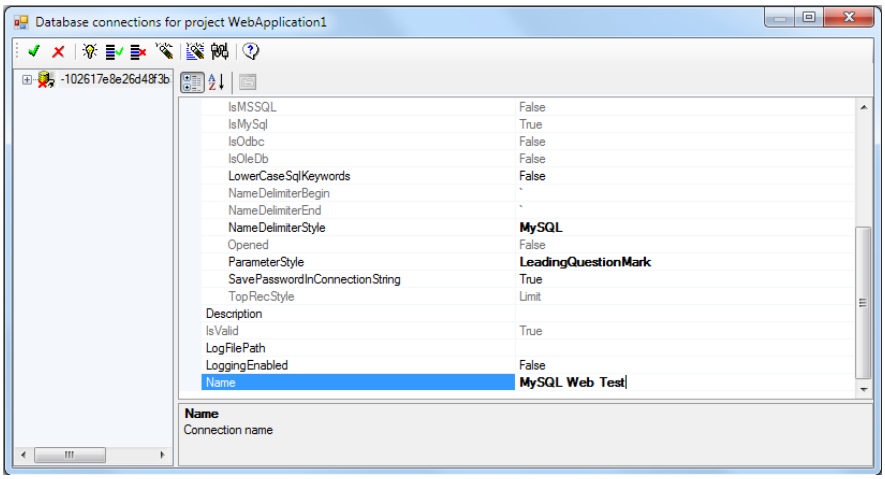
инсталиран на неком удаљеном рачунару, као име сервера може се користити *IP* адреса тог рачунара на мрежи.

Ако се претпостави да ће *"SQL Server"* сервис бити инсталиран на истом рачунару као и апликација на којој се ради и да постоји креирана база која се зове *"test1"*, као и да постоји корисничко име *"root"*, коме је додељена шифра *"pass"*, а који има дозволу приступа бази *"test1"*, у том случају, *"Connection string"* својство се може подесити на следећи начин:



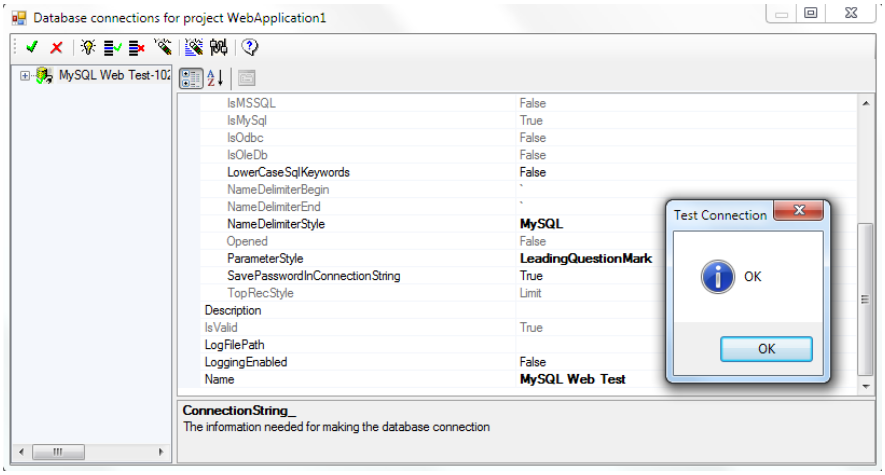
Слика 25 – "Connection string" подешавања

Након извршених претходних подешавања, потребно је још конекцији дати назив, на пример, у овом случају, назваћемо је *"MySQL Web Test"* (слика 26).



Слика 26 – Подешавање назива конекције

Можемо проверити статус конекције (да ли су сви унети параметри тачни) притиском на дугме  :



Слика 27 – Провера статуса конекције

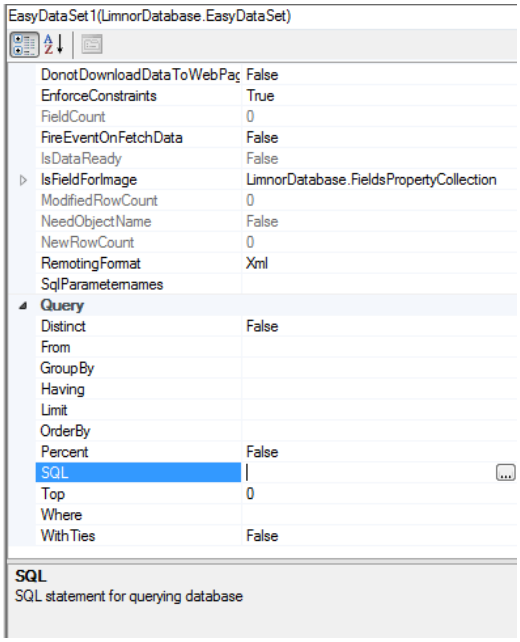
Потребно је запамтити да се на овај начин на базу података повезујемо користећи *"ADO.NET"* управљачки програм (енг. *driver*). Ово је случај само за време пројектовања апликације. Са друге стране, када се *"PHP"* Веб апликација, покрене, *"ADO.NET"* управљачки програм се не користи, из разлога што на пример, на *"GNU/Linux"* базираним машинама, он може бити недоступан. За време рада апликације, користиће се *"PHP"* библиотека за *"MySQL"*. За време рада апликације, све се ово дешава „иза завесе", док је пројектантима битно то да је потпуно свеједно развијати *"Web based"* ili *"Desktop based"* апликацију. У оба случаја, конекција на базе података ради ишчитавања или уписа података, извршава се на потпуно исти начин. Преводилац (енг. *compiler*) ће у сваком случају генерисати код за пројектанта.

- **Коришћење *"Query builder"*-а за креирање упита ка бази**

Да би се креирао упит, може се користити алат интегрисан у *"Limnor Studio"*, који се назива *"Query Builder"*.

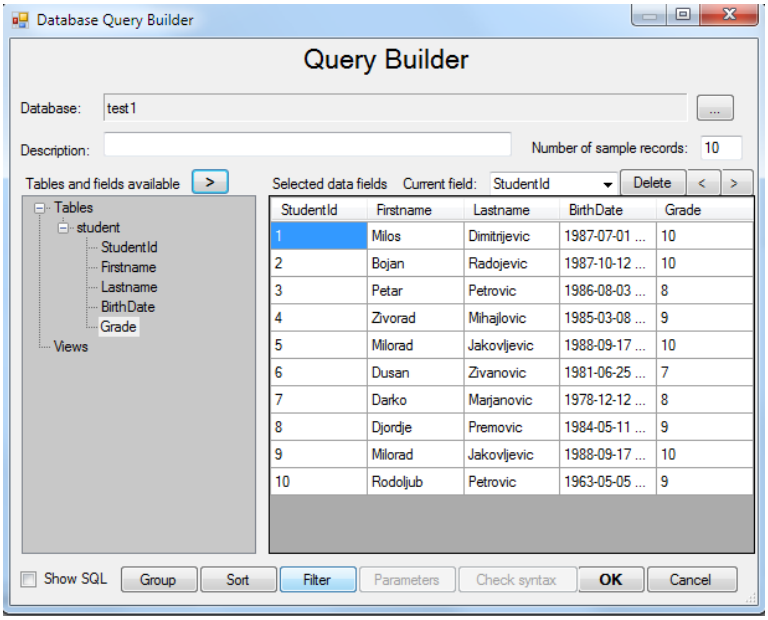
Претпоставиће се у даљем тексту да у бази *test1* постоји креирана табела *Student*, која садржи колоне *StudentID, FirstName, LastName, BirthDate* и *Grade*.

Да би се подаци могли ишчитати, потребно је подесити *"SQL"* својство *EasyDataSet*-а:



Слика 28 – Подешавање SQL својства

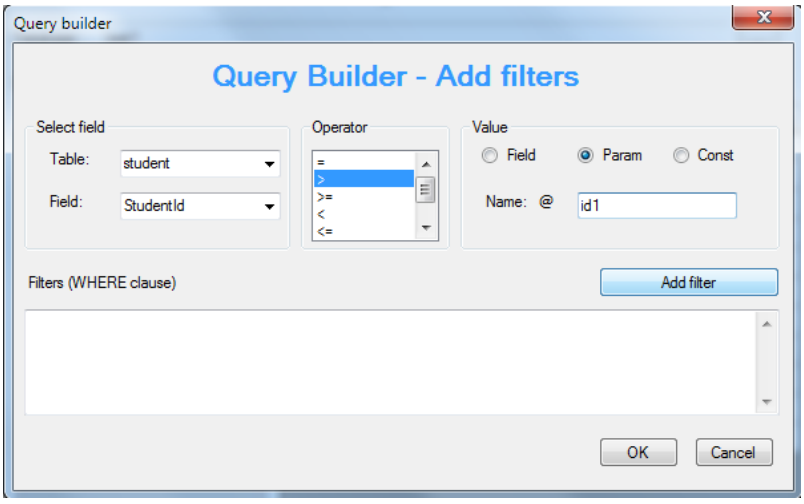
а затим у упит треба додати сва поља која се кориснику желе представити на Веб страни. Да би се ограничио распон података који ће се приказивати кориснику, неопходно је подесити одговарајуће филтере у *Query Builder*-у:



Слика 29 – Подешавање филтера Query Builder-а

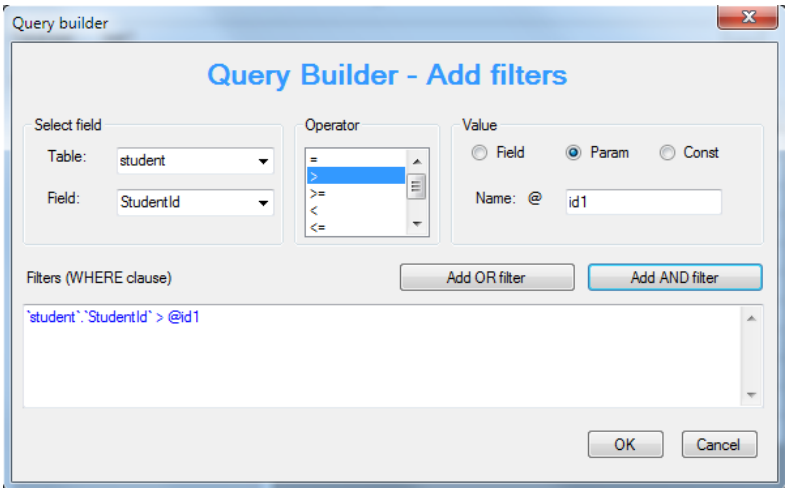
За овај пример, користиће се одређени распон *"StudentID"* промењиве као филтер. Биће дозвољено корисницима да унесу распон *ID*-ева студената са Веб стране.

Да би се ово омогућило, неопходно је подесити параметре филтера. Како би се коректно подесила *"StudentID"* променљива као филтер, потребно је у *"Select Field"* изабрати као табелу *"Student"*, а као *"Field"*, ону променљиву коју желимо као филтер (у овом случају *StudentID)*. Под *"Operator"*, биће изабран *">"*, и биће дато име филтеру *"id1"*, након чега је само неопходно кликнути на дугме *"Add Filter"* (слика 30, 31).



Слика 30 – Креирање новог филтера "id1"

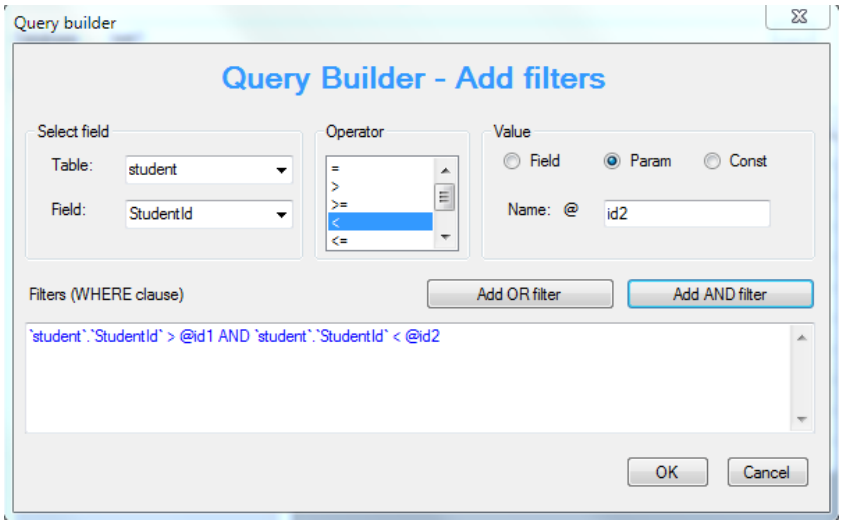
након чега се филтер појављује:



Слика 31 – Додавање филтера – WHERE клаузула

Након сетовања минималне вредности промењиве *"id1"*, неопходно је сетовати и максималну вредност, како би касније кориснику на Веб страни омогућили да подеси распон у ком жели да филтрира *"StudentID"*.

Поново је потребно подесити *"StudentID"* променљиву као филтер, потребно је у *"Select Field"* изабрати као табелу *"Student"*, а као *"Field"*, *"StudentID"*. Под *"Operator"* је овога пута неопходно изабрати *"<"*, и дати име филтеру *"id2"*, након чега је поново потребно кликнути на дугме *"Add AND Filter" (слика 32)*.



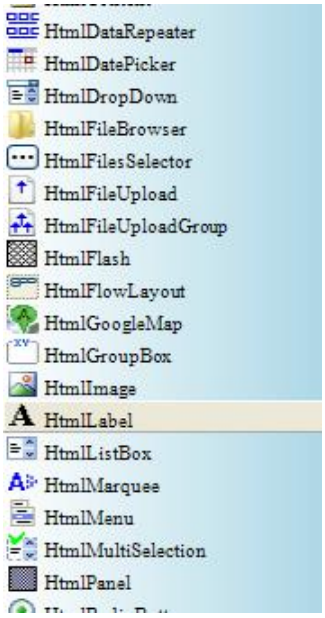
Слика 31 – Додавање филтера – WHERE клаузула

За овај пример, сасвим ће довољно бити имати ова два филтера.

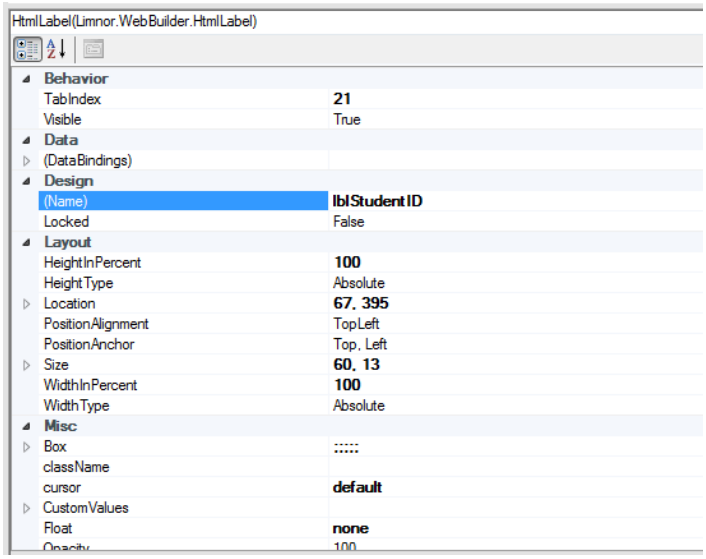
- **Повезивање података са одређеним пољима**

Да би се кориснику омогућио унос података, може се користити *"TextBox"* контрола, док је само за приказ података кориснику довољна и контрола *"Label"*.

Потребно је креирати контролу *"Label"* дуплим кликом на њен назив у *"Toolbox"*-у и назвати је *lblStudentID* (слика 32, 33):

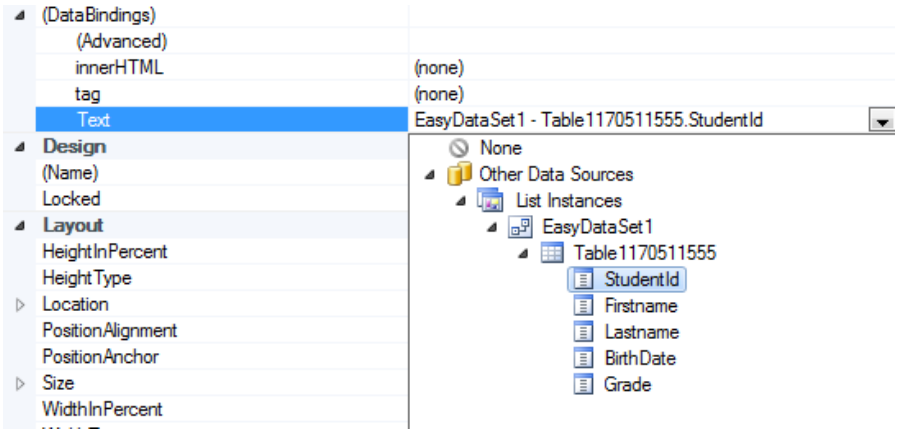


Слика 32 – Додавање контроле "Label"



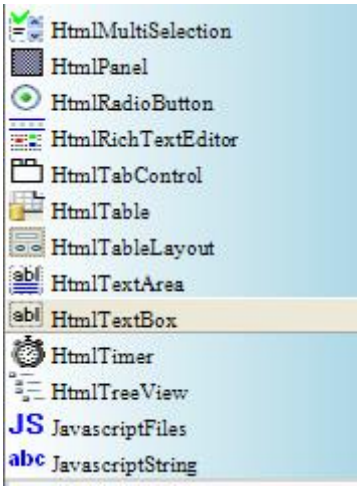
Слика 33 – Промена назива контроле "Label"

"DataBindings" својство треба подесити тако да повезује свој текст са *"StudentID"* колоном (слика 34):



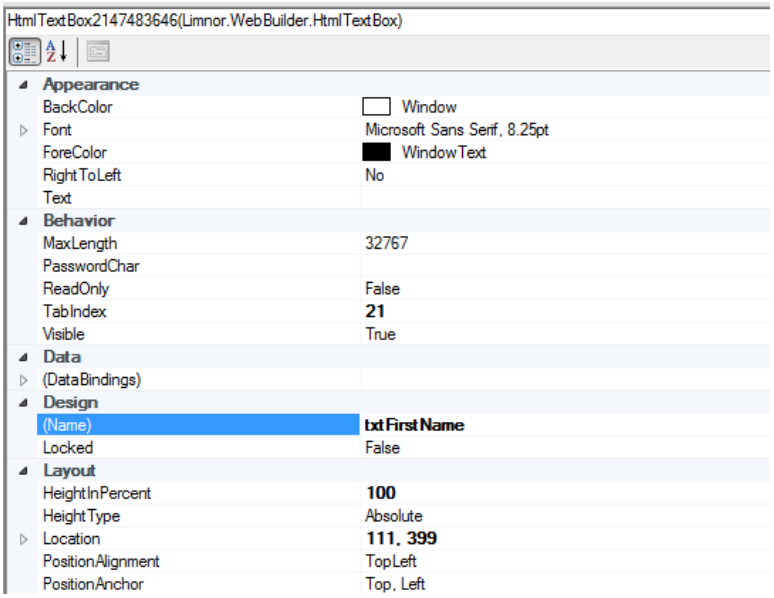
Слика 34 – "DataBindings" својство

Након извршених подешавања својстава контроле *Label*, додаћемо и контролу *TextBox* (слика 35):



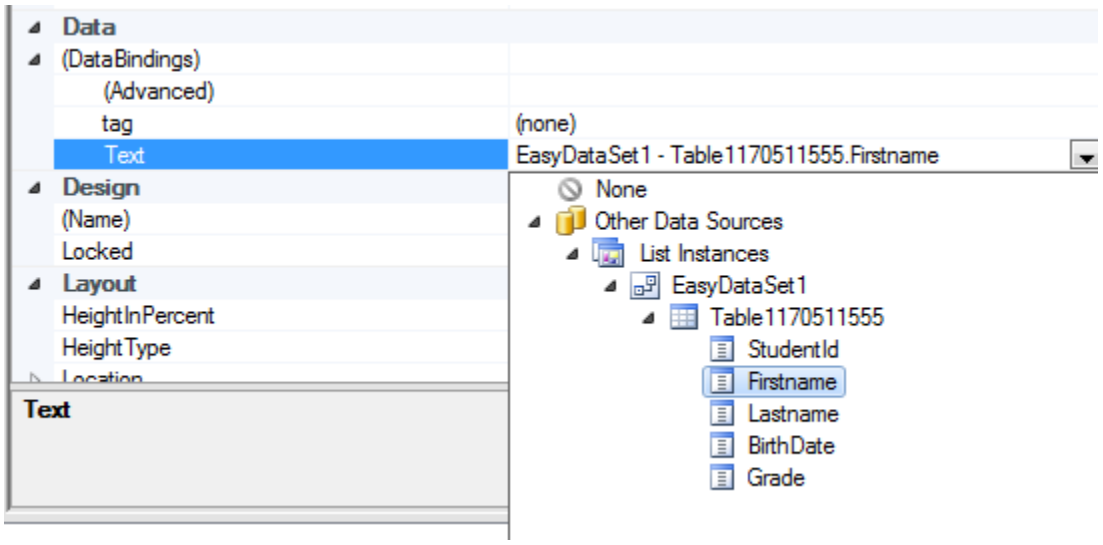
Слика 35 – Додавање контроле "TextBox"

а затим је преименовати у *"txtFirstName"* (слика 36):



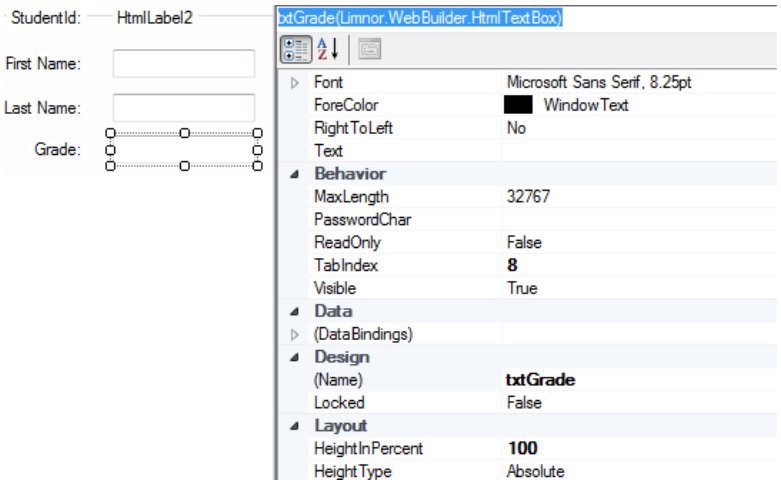
Слика 36 – Својства контроле "TextBox"

За ову контролу, својство *"DataBindings"* треба подедити тако да контрола свој текст прилагођава колони *"FirstName"* табеле *"Student"* (слика 37):



Слика 37 – Подешавања "DataBindings" контроле "TextBox"

На исти начин, неопходно је креирати *"TextBox"* контроле и за остале типове података из табеле и повезати их са одговарајућим колонама (слика 38):



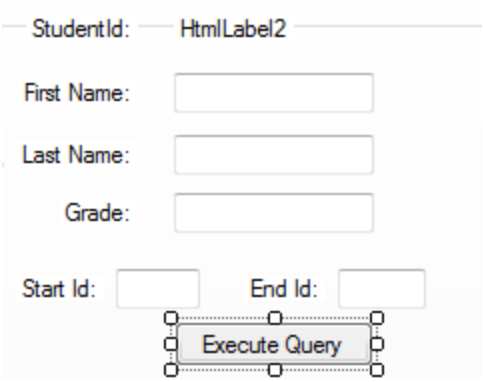
Слика 38 – Подешавања "DataBindings" "Textbox" контрола за колоне "Last Name" и "Grade"

- **Упит података**

Да би се ишчитали подаци из базе, неопходно је извршити упит. Пошто се користе параметри подешени у филтерима, мора се креирати кориснички интерфејс да би се кориснику омогућио унос вредности ових параметара. У ове сврхе, неопходно је креирати две *"TextBox"* контроле за унос вредности *id1* и *id2* (слика 39):

Слика 39 – Додавање "TextBox" контрола за упис вредности id1 и id2

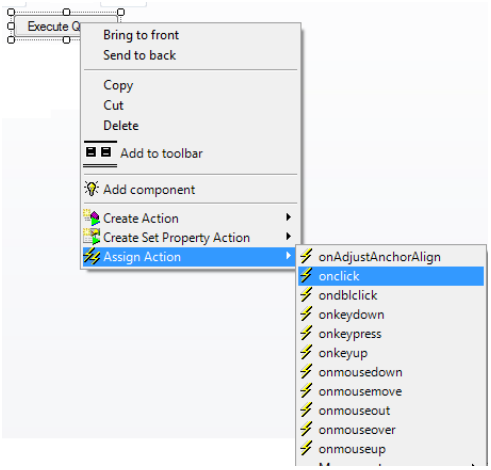
Наравно, неопходно је додати и једну контролу *"Button"* да би се њоме покренуо код и њу је могуће назвати нпр. *btQuery* (слика 40):



Слика 40 – Дугме за „окидање" "PHP" кода

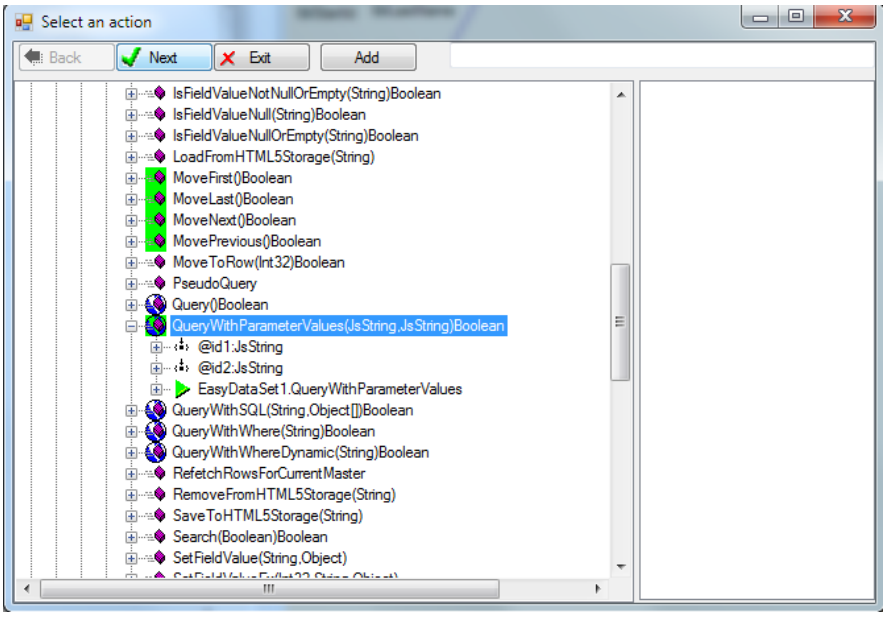
Као што је и очигледно, оно што треба постићи је да када корисник кликне на дугме, изврши се упит, користећи параметре из *"TextBox"* контрола.

Десним кликом на дугме треба кликнути на опцију *"Assign Action"*, а као врсту окидача (догађаја) изабрати *"onclick"* (слика 41):



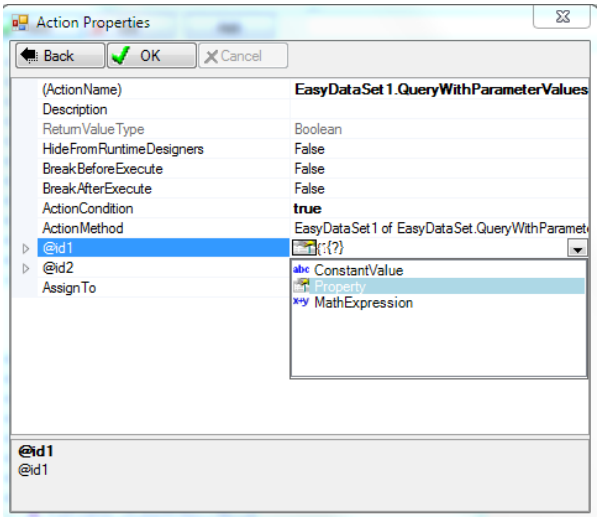
Слика 41 – Додељивање акције дугмету

затим изабрати *QueryWithParameterValues* из *EasyDataSet*-а, а затим кликнути на *Next* (слика 42):



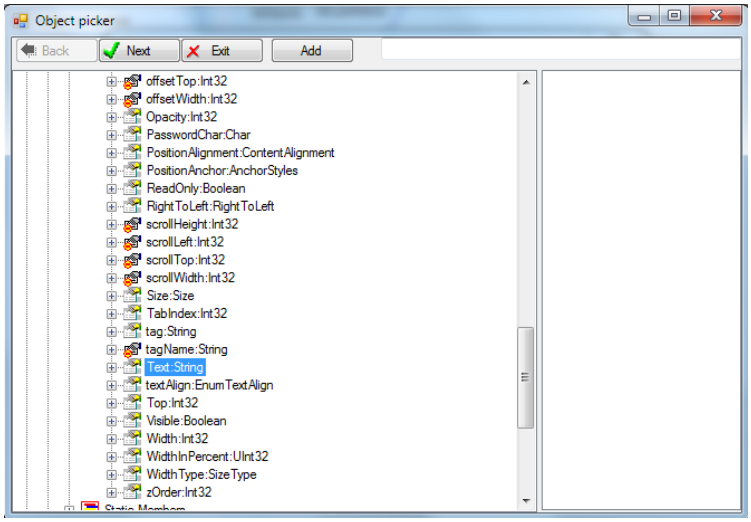
Слика 42 – Одабир догађаја "QueryWithParameterValues"

изабрати *"Properties"* за *"@id1"* (слика 43):



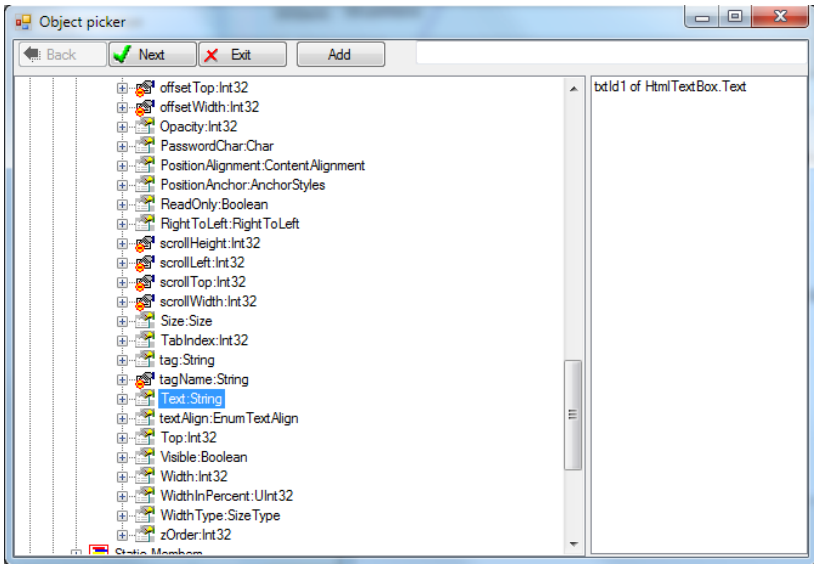
Слика 43 – "Properties" за "@id1"

а затим селектовати својство *Text* за почетни *"id"* (слика 44):



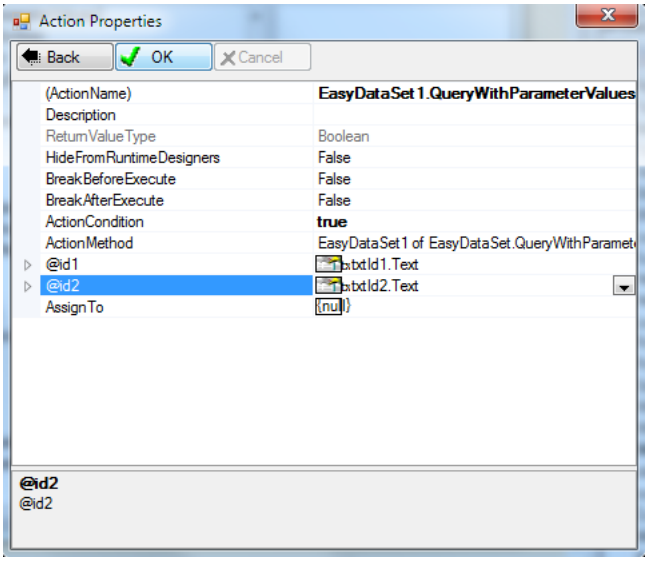
Слика 44 – Својство "Text" за "StartId"

За *"@id2"* селектовати *"Text"* својство *"TextBox"*-а за *"EndId"* (слика 45)::



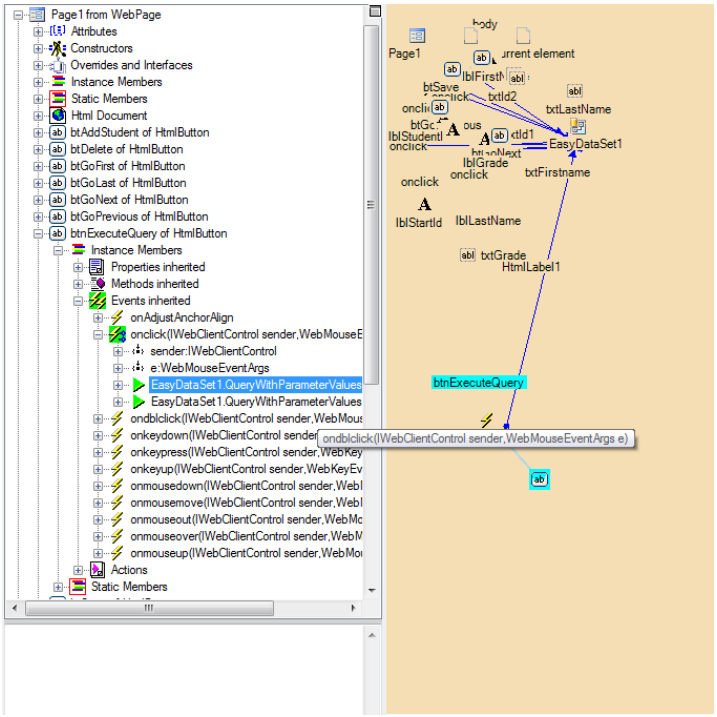
Слика 45 – Својство "Text" за "EndId"

Кликнути *"OK"* за завршетак креирања упита (слика 46):



Слика 46 – Завршетак креирања упита

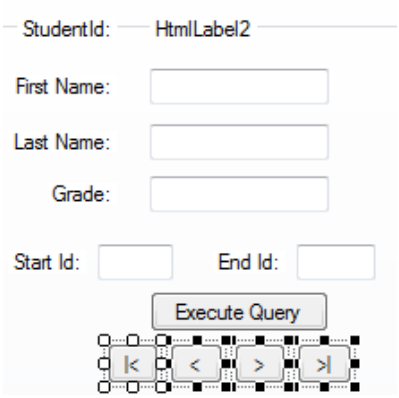
Овим је завршено креирање догађаја притиском на дугме, који је видљив у *"Event Map"*-у и у *"Object Explorer"*-у (слика 47):



Слика 47 – "Event Map" и "Object Explorer"

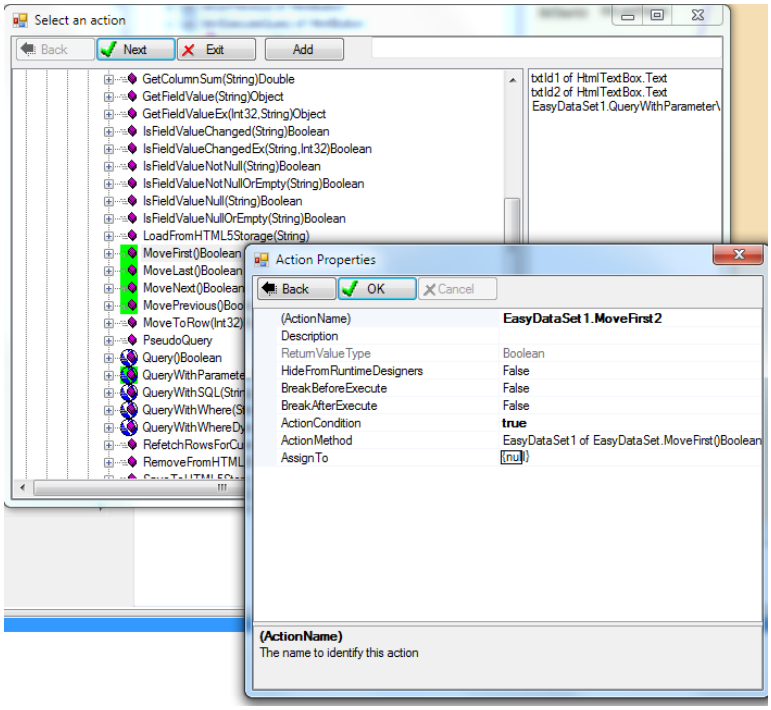
- **Навигација кроз податке**

Када корисник упише опсег *"StudentID"*, могуће је да ће бити потребно да има могућност навигације кроз добијене податке. У ове сврхе, креираћемо четири нова дугмета *btGoFirst, btGoPrevious, btGoNext, btGoLast* (слика 48)*:*



Слика 48 – Дугмад: "btGoFirst", "btGoPrevious", "btGoNext", "btGoLast"

Десним кликом на дугме *"btGoFirst"*, изабрати *"OnClick"* догађај, изабрати *"MoveFirst"* из *"EasyDataSet"*-а, и кликнути на *"Next"* (слика 49):



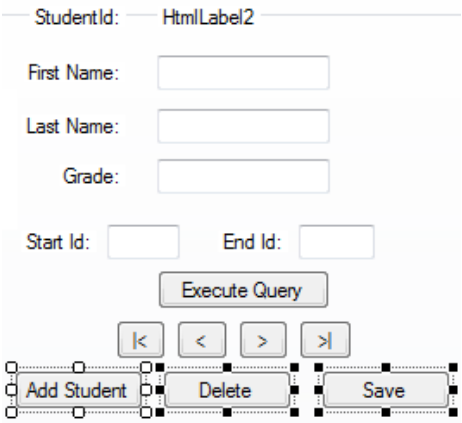
Слика 49 – Подешавање догађаја за дугме "btGoFirst"

Потребно је поновити овај поступак и за преостлаа три дугмета, с тим што је уместо *"MoveFirst"* неопходно изабрати *"MovePrevious", "MoveNext", "MoveLast"*, у зависности од тога о ком се дугмету ради.

- **Модификаиција и чување података**

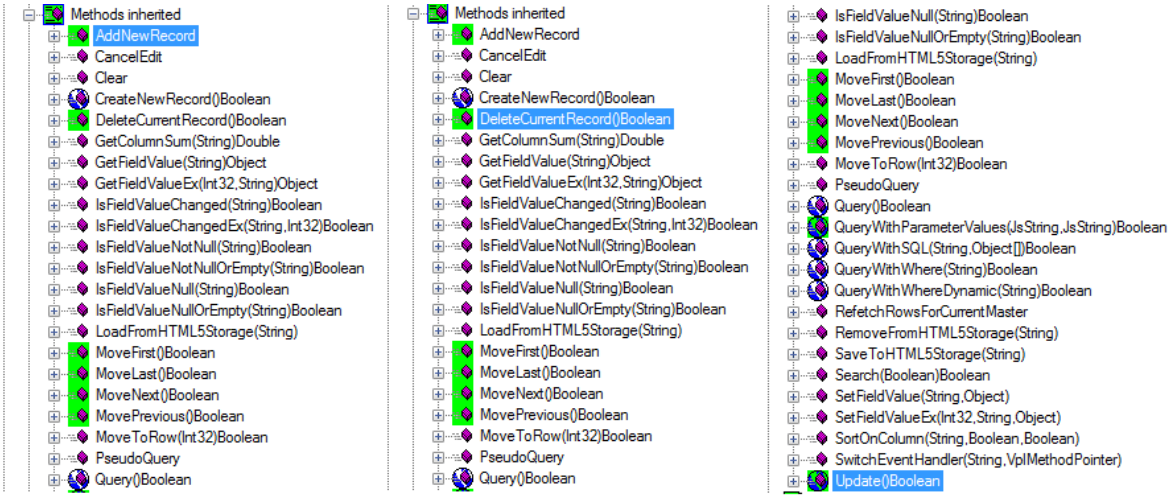
Такође, корисник може пожелети да дода нови упис у базу, као и да обрише неки постојећи упис из исте.

За ове сврхе, треба додати дугмад *"Add Student", "Delete"* и *"Save"* (слика 50):



Слика 50 – Дугмаd "Add Student", "Delete" и "Save"

За дугме *"Add Student"*, треба изабрати догађај *"AddNewRecord"* у *"EasyDataSet1"*. За остала двоја дугмад, неопходно је додати догађаје *"DeleteCurrentRecord"* и *"Update()Boolean"* (слика 51):



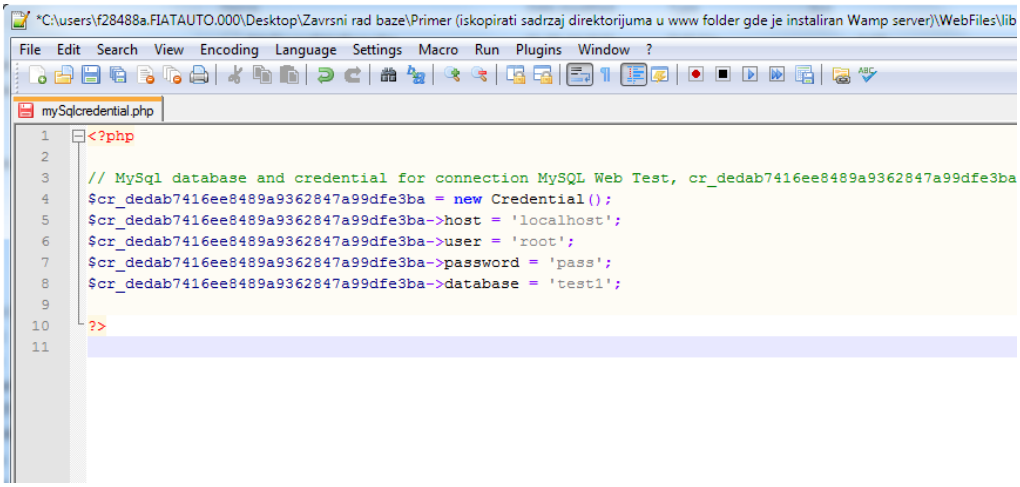
Слика 51 – Догађаји "AddNewRecord", "DeleteCurrentRecord" и "Update()Boolean

Сада на форми постоје сва дугмад потребна за покретање упита које извршава *"EasyDataSet"*.

- ***"MySQL Database Credential"***

Након завршених свих претходних корака, могуће је компајлирати пројекат. Сви Веб фајлови ће бити сачувани у *"WebFiles"* директоријуму.

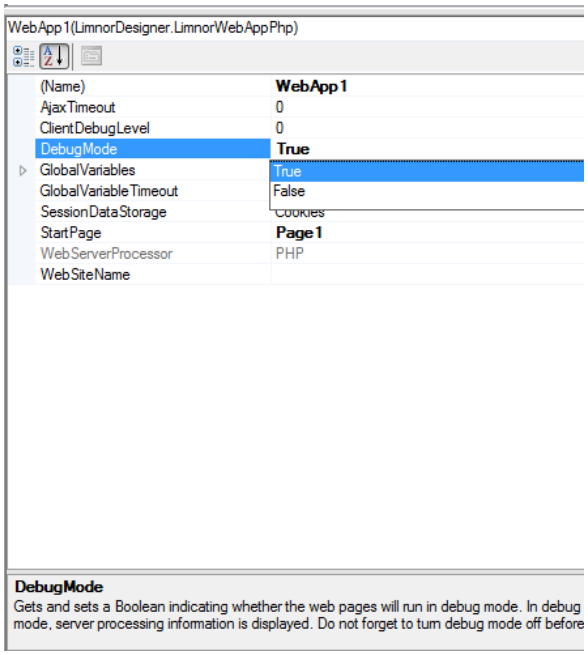
"Credentials", односно корисничко име и шифру за повезивање на базу је могуће пронаћи у фајлу који се зове *"mySqlCredentials.php"*. Могуће је изменити овај фајл користећи неки од едитора како би се изменили корисничко име и шифра за повезивање на базу података (слика 52):



Слика 52 – Фајл "mySqlCredentials.php" едитован у "Notepad++"

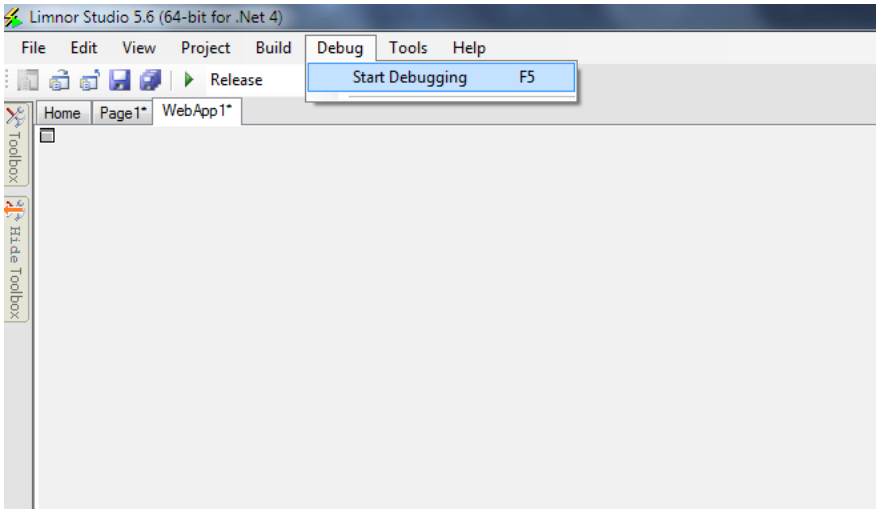
- **Тестирање Веб апликације и базе података**

На даље је неопходно укључити *"Debug Mode"* на вредност *"True"* у својствима *"WebApp1"* да би се испратило да ли постоје грешке у клијент/сервер комуникацијама (слика 53):



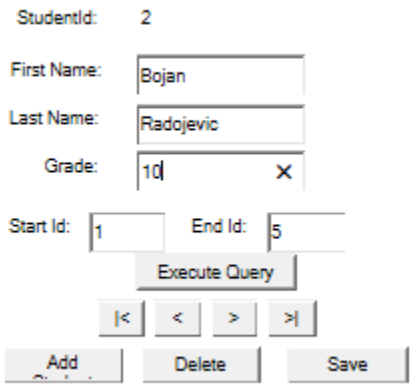
Слика 53 – "Debug Mode"

Даље, кликнути на дугме *"Start Debugging"* или *"F5"* на тастатури (слика 54):



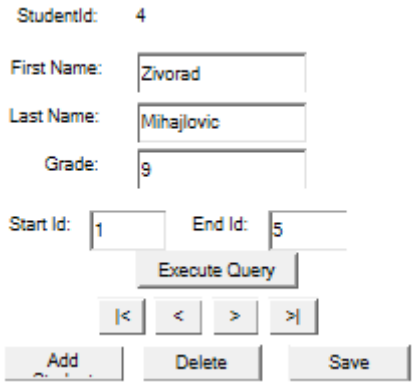
Слика 54 – Покретање стране

Након пар тренутака, отвориће се страна која је претходно креирана (слика 55):



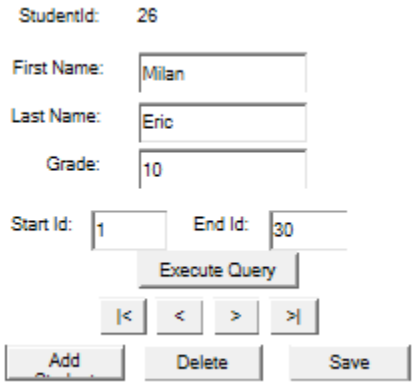
Слика 55 – Тест стране

Као што се из приложеног примера да видети, први студент који задовољава услове је студент са *"StudentID"* = 2. Након тога, кликтањем на дугмад за кретање на лево и десно, мењаће се подаци о студентима, све док *"StudentID"* вредност не буде мања од 5, што значи да је упит који је претходно направљен користећи *Limnor Studio* одрадио посао (слика 56):



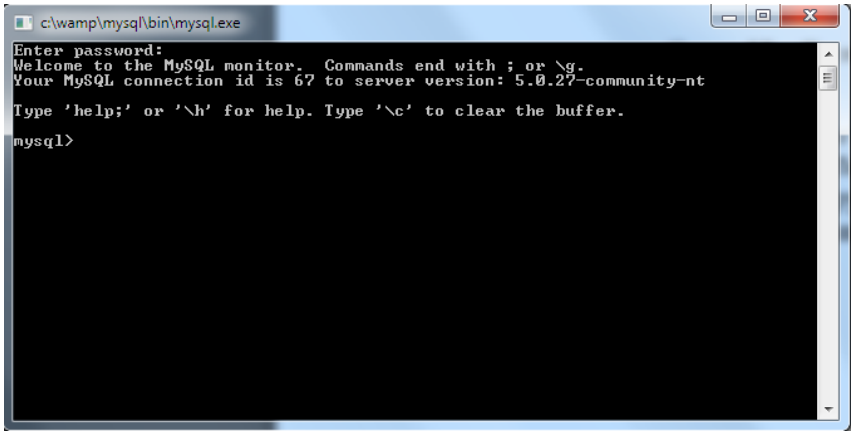
Слика 56 – Тест "SQL" упита

Може се такође испробати и додавање новог студента са оценом коју је добио како би се тестирала база на упис. Треба додати новог студента кликом на *"Add Student"* и уписивањем неопходних података (слика 57):



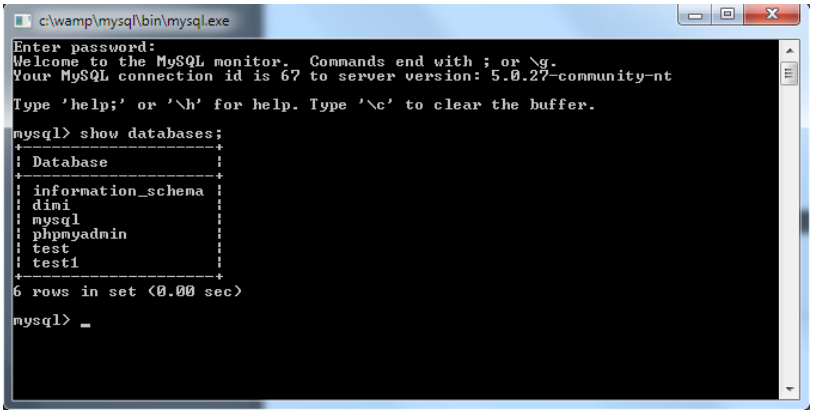
Слика 57 – Тест "SQL" упита за упис новог студента

а затим кликнути на дугме *"Save"*. Након тога, остаје само да покушамо да истог студента пронађемо у табели *"Student"* у бази *"test1"* користећи *"MySQL"*. Прво је потребно отворити *"MySQL"* конзолу и унети шифру, која у овом случају не постоји, јер при инсталацији није додељена. Након тога, појавиће се следећа конзола:



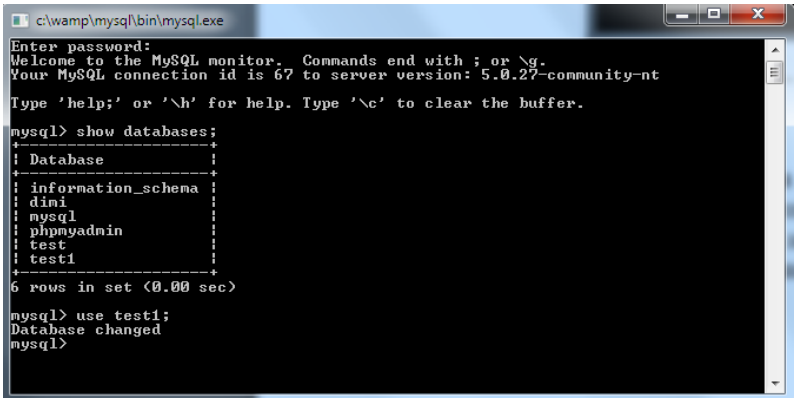
Слика 58 – "SQL" конзола

Укуцавањем кода: *"show databases;"* (без наводника), појавиће се списак свих тренутно креираних база (слика 59):



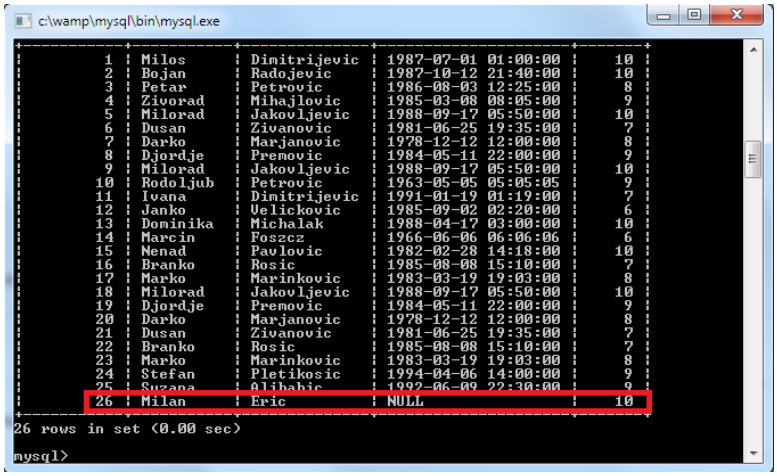
Слика 58 – "SQL" конзола – доступне базе

Као што се да видети, база *"test1"* је међу њима. Да би било могуће проверити табеле у њој, прво је неопходно селектовати базу која ће се користити. То је могуће учинити укуцавањем следећег кода: *"use test1;"* и притиском на тастер *Enter* на тастатури. *"SQL"* ће нам генерисати повратну информацију да је тражена база изабрана (слика 59):



Слика 59 - "SQL" конзола – одабир жељене базе

након чега је могуће изабрати да *"SQL"* извуче све податке из табеле *"Student"* кодом: *"Select * from Student;"*



Слика 59 - "SQL" конзола – провера уноса

Као што се види, упитом табеле *"Student"* директно у *"MySQL"*-у, проналази се податак који је претходно унешен коришћењем Веб стране. Податак *"BirthDate"* има вредност *"NULL"*, односно без вредности је, из разлога што на Веб страни није било омогућено уношење овог податка. Овим се може закључити да *"PHP"* и база *"test1"* комуницирају у оба смера и да је претходни пример у потпуности функционалан.

ЗАКЉУЧАК

Базе података на Веб-у могу бити веома користан извор и складиште информација за програмера Веб стране. Као што се види из наведених примера, исте могу бити веома корисне за складиштење велике количине података, коју би било веома тешко, чак и у неким случајевима немогуће чувати у радним променљивим у *"PHP/VB"* коду, а веома непрегледно у екстерним библиотекама и пропратним фајловима, из разлога што би за касниј евентуалну измену функционалности програмског пакета било потребно много више уложеног труда и изгубљеног времена ,што са собом носи и одређене трошкове.

Интеграцијом *"PHP/VB"* кода са *"MySQL"*-ом, добијају се софтверска решења која је на врло лак начин могуће мењати тако да буду привлачнија крајњим корисницима, ослобађа се радна меморија веће количине података која би се морала обезбедити без коришћења базе и добија се могућност складиштења истих на удаљеним серверима, рецимо, у *"Data Center"*-у фирме и истовремени приступ подацима са више клијената, односно више различитих Веб апликација уколико је то потребно, што даје веома широке могућности, нарочито у компанијама где је потребно користити исте податке у различите сврхе и где велик број корисника истовремено користи једну те исту Веб страну, која приступа бази ради преузимања података за крајњег корисника. Овакав приступ грађења инфраструктуре Веб страна је прихваћен од већине компанија широм света, јер пружа централизована и интегрисана решења, која као исход имају јефтиније пословање фирме уз повећање перформанси система, што је у суштини битно за крајњи кориснички утисак и опште пословање фирме.

ЛИТЕРАТУРА

1. Веиновић М., Франц И., Јевремовић А.: ***Базе података – Практикум***, Сингидунум Београд, Београд 2006.

2. Стојановски Ј., Шкварч Г.: ***Online базе података***, Хрватска академска и истраживачка мрежа *CARNet* Загреб, Загреб 2007.

3. *Longflow Enterprises Ltd.*: ***Web database programming***, *Longflow Enterprises Canada, Canada* 2011

4. Улман Л.: ***PHP и MySQL за динамичке Веб сајтове,*** *CET* Београд, Београд 2012.

5. Велинг Л.: ***PHP и MySQL: развој апликација за Веб,*** Микро Књига Београд, Београд 2013.

6. Дејвис Е. М., Филипс Ц.: ***Learning PHP & MySQL 2nd Edition***, *O'Reilly Media Sebastopol California, California* 2007.