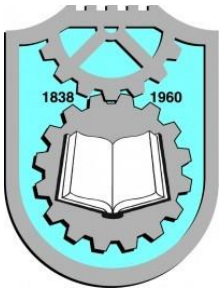
Факултет инжењерских наука Универзитета у Крагујевцу



Назив студијског програма: Машинско инжењерство
Ниво студија: Основне академске студије
Модул: Информатика у инжењерству
Предмет: Базе података
Број индекса: 139/2014

Иван М. Вучковић

Приступ базама података преко позива функција – основни концепти и примери коришћења

завршни рад

Комисија за преглед и одбрану:

1. Проф. Др Милан Ерић - ментор Датум
2._____ одбране: _____
3._____ Оцена:_____

У оквиру овог завршног рада са темом *"приступ базама података преко позива функција – основни концепти и примери коришћења"* **кандидат треба да:**

Проучи литературу и презентира основне концепте приступа базама података преко позива функција (SQL/CLI-SQL Call Level Interface). Упореди стандарде и конкретизује их на развојном веб окружењу. На крају рада дати закључна разматрања.

Препоручена литература:

1. Лазаревић Б. ….: **Базе података**, ФОН Београд, Београд 2003.

2. Павловић-Лажетић Г.: **Основе релационих база података**, Математички факултет, Београд, 2000.

3. James Wilson: **Microsoft .NET серверска решења**, ЦЕТ библиотека, Београд, 2013.

4. Welling L., Thomson L.: **PHP i MySQL развој апликација за веб**, Микро књига, Београд, 2009.

5. Jim Buyens: Развој база података на вебу: CET Computer Equipment and Trade Београд, 2001, Превод првог издања

Крагујевац,
2018.

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.

139/2014 Иван Вучковић

_____

(потпис)

САРДЖАЈ

Резиме

У овом раду детаљно је описан приступ базама података преко позива функција, као и основни концепти и примери коришћења истих, затим процедуре базе података. Такође и пројектовање базе података, које укључује детаљно сагледавање проблема почев од саме идеје за коју је потребно израдити базу података и усавршити је тако да долази до што мање редундансе како би само претраживање, унос и читање података било што брже и прецизније, па до израде сајта лако употребљивог, како за кориснике, тако и за администраторе.

Кључне речи: *база података, читање, уписивање, редунданса, сајт.*

Abstract

This study describes, database access through functions, main principles and concepts of use, and database procedures. It explains database projecting, which involves detailed analysis of the problem starting with the main idea so that in the end there is no redundancy, which leads to easier use of the database, for the administrators and users.

Key words: *database, reading, writing, redundancy, site.*

1. Уводна разматрања

Од самог почетка коришћења компјутера, обрада различитих врста података била је један од основних и најсложенијих задатака. Информације и подаци су постали снага покретања данашњег пословања у почетку на западу, а касније и у целом свету. Ако желимо да имамо квалитетне информације о свим деловима нашег пословног или пак приватног живота, најбоље је да на одређени начин организујемо све податке који могу да нам пруже информације у било ком тренутку када ми то пожелимо, а што је притом најбитније, можемо их увек имати код себе, на телефону, таблету, лап топ рачунару и увек их брзо прегледати, или на флеш меморији која може стати у новчаник, држати веома велике базе података. Ако је пак потребно да их брзо извучемо на било ком месту или чак да понудимо те информације одређеним корисницима, можемо их окачити на интернет, рецимо израдом веб сајта, на коме можемо и ажурирати те податке, онда је све те податке потребно сместити у табеле помоћу којих ће нама или неком кориснику бити лакше да пронађе жељену информацију. Веома често и готово увек је потребно направити више табела како би дошло до што мање преклапања у бази података. Те табеле се даље повезују помоћу веза са заједничким елементима двеју табела. Скуп више таквих међусобно повезаних табела које служе заједничком циљу се називају базе података. Њихов заједнички циљ се односи на свођење веома брзе и успешне информације о свим догађајима које се дешавају у оквиру једне целине.

Када куцамо текст у Word-u, радимо неке табеларне прорачуне у Excel-u у више табела, онда имамо додира са базом података. У случају да нам база постане толико компликована да нисмо више у стању да контролишемо ток и развој података, потребно је прећи на виши ниво и затражити помоћ од програма који служе лакшој организацији података и управљању базом података. Такви системи за рад са базама података постоје већ дуги низ година, а једни од њих су: Access – са којим ћемо се сусрести у даљем раду, затим Microsoft SQL, DB2, XML, FOXPRO, DBMC.

Најранија употреба термина база података потиче из 1963. године. База података као јединствена реч постала је уобичајена у Европи у раним 1970-тим.

2. О базама података

База података је алат за прикупљање и организовање информација. Базе података могу да складиште информације о особама, производима, поруџбинама или било чему другом. Многе базе података започну као листа у програму за обраду текста или унакрсној табели. Како листа расте, почињу да се појављују редундансе и недоследности у подацима. Подаци постају тешки за разумевање у облику листе и ограничени су начини претраживања или извлачења подскупова података за прегледање. Када проблеми почну да се јављају, добра идеја је пребацити податке у базу података направљену системом за управљање базом података (DMBS), као што је Access.

Компјутеризована база података представља контејнер објеката. Једна база података може да садржи више табела. На пример, систем за праћење инвентара који користи три табеле нису три базе података, него једна која садржи три табеле. Осим ако није нарочито дизајнирана тако да користи податке или кôд из другог извора, Access база података складишти табеле у једној датотеци, заједно са објектима, као што су обрасци, извештаји, макрои и модули. Базе података направљене у формату Access 2007 (који користе и Access 2016, Access 2013 и Access 2010) имају ознаку датотеке .accdb, а базе података направљене у старијим Access форматима имају ознаку типа датотеке .mdb. Access 2016, Access 2013, Access 2010 или Access 2007 могу се користити за прављење датотека у старијим форматима датотеке (на пример, Access 2000 и Access 2002 – 2003).

Помоћу програма Access могу се извршити следеће радње:

- Додавање нових података у базу података, као што је нова ставка у инвентару.
- Уређивање постојећих података у бази података, као што је промена тренутне локације ставке.
- Брисање информација, ако се нека ставка можда прода или одбаци.
- Организовање и приказ података на различите начине.
- Дељење података са другима путем извештаја, е-порука, интранета или интернета.

2.1. Делови Access базе података:

- Табеле
- Обрасци
- Извештаји
- Упити
- Макрои
- Модули

2.1.1. Табеле

База података изгледа слично као унакрсна табела због тога што се подаци складиште у редовима и колонама. Резултат тога јесте да је обично прилично лако увести унакрсну табелу у табелу базе података. Главна разлика између складиштења података у унакрсној табели и у бази података јесте начин организовања података.

Да би база података била што флексибилнија, подаци се морају организовати у табеле како се не би појављивала редунданса. На пример, ако складиштите информације о запосленима, сваки запослени би требало да се само једном унесе у табелу која је подешена да чува податке о запосленом. Подаци о производима складиштиће се у посебној табели, а подаци о пословницама складиштиће се у другој табели. Овај процес назива се нормализација.

Сваки ред у табели назива се запис. У записима се складиште појединачне информације. Сваки запис састоји се из једног или више поља. Поља одговарају колонама у табели. На пример, може постојати табела под именом „Запослени“ у којој сваки запис (ред) садржи информације о другом запосленом, а свако поље (колона) садржи други тип информација, као што су име, презиме, адреса и тако даље. Поља се морају назначити као одређени тип података, без обзира на то да ли је то текст, датум или време, број или неки други тип.

Други начин за описивање записа и поља јесте визуелизација старих каталога картица у библиотеци. Свака картица у картотеци одговара једном запису у бази података. Свака информација на појединачној картици (аутор, наслов и тако даље) одговара једном пољу у бази података.

2.1.2. Обрасци

Обрасци омогућавају прављење корисничког интерфејса у ком је могуће уносити и уређивати податке. Обрасци често садрже командну дугмад и друге контроле које извршавају разне задатке. Базу података могуће је направити без коришћења образаца тако што ће се једноставно уносити информације у листове са подацима табеле. Међутим, већина корисника базе података радије користи обрасце за приказивање, унос и уређивање података у табелама.

Командну дугмад је могуће програмирати тако да одређују који ће се подаци појављивати у обрасцу, отваратти друге обрасце или извештаје, односно извршавати разне друге задатке. На пример, можда постоји образац под именом „Образац клијента“ на коме се ради са подацима о клијенту. Образац клијента можда има дугме које отвара поруџбину у којој се може унети нова поруџбина за тог клијента.

Обрасци омогућавају и контролисање на који начин други корисници врше интеракцију са подацима у бази података. На пример, могуће је направити образац који приказује само одређена поља и омогућава извршавање само одређених операција. То доприноси заштити података и обезбеђује да се подаци исправно унесу.

2.1.3. Извештаји

Извештаји се користе за обликовање, дотеривање и представљање података. Извештај обично одговара на одређено питање, као што је „Колико новца смо примили од сваког клијента ове године?“ или „У којим се градовима налазе наши клијенти?“. Сваки извештај може да се обликује да представља информације тако да их је најлакше прочитати.

Извештај може да се покрене било када и увек ће представљати тренутне податке у бази података. Извештаји су обично обликовани за штампање, али могу и да се прикажу на екрану, извезу у други програм или пошаљу као прилог у е-поруци.

2.1.4. Упити

Упити могу да извршавају различите функције у бази података. Њихова најчешћа функција јесте да преузимају одређене податке из табела. Потребни подаци се углавном налазе у више табела, док упити омогућавају да све те податке сместимо у једну листу. Такође, пошто није потребно да се виде сви записи истовремено, упити омогућавају додавање критеријума за „филтрирање“ података само на жељене записе.

Одређени упити могу да се „ажурирају“, што значи да се могу уредити подаци у основним табелама помоћу листа са подацима упита. Ако је потребно радити у упиту који се може ажурирати, пожељно је запамтити да се промене заправо извршавају на табелама, а не само на листу са подацима упита.

Постоје две основне врсте упита: упити за издвајање и радни упити. Упит за издвајање само преузима податке и чини их доступним за коришћење. Резултати упита могу се видити на екрану, одштампани или копирани у остави. Односно, излаз упита може се користити као извор записа за образац или извештај.

Радни упит, као што му име говори, извршава задатак са подацима. Радни упити могу да се користе за прављење нових табела, додавање података у постојеће табеле, ажурирање или брисање података.

2.1.5. Макрои

Макрои у програму Access налик су на поједностављени програмски језик који се може користити за додавање функционалности бази података. На пример, макро се може приложити командном дугмету на обрасцу тако да се макро покреће кад год се кликне на дугме. Макрои садрже радње које извршавају задатке, као што је отварање извештаја, покретање упита или затварање базе података. Већина операција базе података које се извршавају ручно могу се аутоматизовати помоћу макроа, тако да они могу значајно уштедети време.

2.1.6. Модули

Модули, попут макроа, представљају објекте који могу да се користе за додавање функционалности бази података. Док се макрои праве у програму Access бирањем са листе радњи макроа, модули се пишу у програмском језику Visual Basic for Applications (VBA). Модул је колекција декларација, израза и процедура које су ускладиштене заједно као целина. Модул може да буде модул класе или стандардни модул. Модули класе приложени су обрасцима или извештајима и обично садрже процедуре које су специфичне за образац или извештај ком су приложени. Стандардни модули садрже опште процедуре које нису повезане са било којим другим објектом. Стандардни модули наведени су у оквиру ставке Модули у окну за навигацију, док модули класе нису.

2.2. Организација података

Организација података је од веома великог значаја када је потребно радити са базом података. Један од кључних аспеката доброг креирања базе података јесте како ће подаци бити организовани у бази података. Да би постигли добро креирану базу података, податке би требало организовати тако да су лако доступни и да омогућавају лако одржавање базе података.

Потребно је одредити који ће подаци улазити у базу података, затим који ће се подаци сместити у одређене табеле међу којима ће бити успостављен однос, те какав је однос међу тим подацима. Потребно је смањити могућност колико је могуће да се исти податак записује више пута (редунданса илити преклапање), јер вишеструким записивањем настају проблеми очувања стварне, јединствене вредности свих података при ажурирању. То утиче и на поузданост информација које се добију из тих података. Потребно је управљати смештањем података и очувања тих података од намерних и ненамерних уништења тј. да не дође до губитка интегритета података. Неке податке треба заштитити од тога да их неовлаштени корисници не мењају, што се зове тајност или приватност података.

2.3. Креирање базе података

У свакодневном животу, да би се почело нешто правити, потребно је унапред одредити дизајн, скицу. Ако је потребно да неко јело буде припремљено, потребан је рецепт, ако је потребно направити кућу, потребна је скица, као и документација како ће изгледати.

При креирању базе података, такође претходно треба организовати податке, и одредити циљеве.

Циљеви дизајнирања/креирања:

- Елиминисати сувишне податке;
- Омогућити брзо проналажење појединачних података;
- Сачувати једноставно одржавање базе података.

Кључне активности при креирању база података су:

- Моделирање апликације;
- Дефинисање података неопходних за апликацију;
- Организовање података у табелама;
- Успостављање међусобних веза између табела;
- Успостављање захтева индексирања и вредновања података;
- Израда и снимање свих потребних упита у вези са апликацијом.

Моделирање апликације се односи на поступак у којем се дефинишу задаци које апликација треба да обави. Било би добро да се дефинисани задаци спецификују у одређени документ који може помоћи усредсређивању на задатак самог програма. Приликом организовања података у табеле одредити да ли неки податак припада тој табели или не, на пример, ако је потребно пратити у исто време информације и о понуђачима и о самим корисницима сајта, онда ће у једну табелу бити стављени и понуђачи и сами потрошачи односно корисници. Обе групе захтевају информације о имену, адреси, телефонском броју, док би понуђач требало да има своју фирму, као и дате бројеве жиро рачуна како би се могла при куповини једноставно извршити трансакција новца.

Ако је направљено више табела где се сваки податак појављује само једном, приликом измена нова информација уписиваће се само једном.

При нормализацији података, међусобно повезане информације обично је потребно сместити у неколико табела. Међутим, када је потребно приступити подацима пожељно је видети информације из свих табела на једном месту. Да би се то постигло потребно је формирати скупове записа који обједињују међусобно повезане информације из неколико табела. Тај скуп записа се формира употребом SQL наредбе у којој се наводе жељена поља, локације поља и однос између табела и те наредбе може се сместити као упит у базу података.

Употреба упита има неколико предности:

- Премештање апликације у сервер окружење је једноставније;
- Уношење измена у SQL наредбу је једноставније;
- SQL наредбу је могуће лакше користити на више места у програму или више програма.

Ови упити раде брже од упита који се израђују задавањем наредбе програма кôда.

Добро креирана база података омогућава:

- Минимално време потребно за проналажење записа;
- Смештање података на најефикаснији начин тако да база не постане сувише велика;
- Најједноставније ажурирање података;
- Захваљујући својој флексибилности укључивање нових функција које би се од програма могле захтевати.

2.4. Приступи базама података

SQL (Structured Query Language) – омогућава пуни приступ подацима у релационим базама података (као што су: Oracle, SQL Server, ACCESS итд.) тако што корисник опише податке које жели да види. SQL омогућава и дефинисање и модификацију изгледа табела унутар базе података. Нити једна операција на базама података, извршена директно из DBMS-а или преко корисничке апликације, не би могла бити извршена без директне или индиректне употребе SQL језика. Често сам SQL није довољан уколико је потребно извршити неки упит у тачно одређено време или га поновити неколико пута. Због тога сваки DBMS поред SQL подржава бар још један програмски језик помоћу којег се могу извршити комплексне операције. Дакле, постоје још неки програмски језици за приступ подацима који се називају Database API, а то су: ODBC, OLE DB, JDBC, DAO, ADO…

ODBC (Open Database Conectivity) – помоћу њега програмери могу радити са табеларним подацима, као што су SQL базе података или са мултидимензионалним подацима, као што је OLAP коцке. Апликације за управљање базама података позивају функције у ODBC-у, а ODBC путем својих драјвера за базе података, апликацији враћа податке из базе.

OLE DB (Object Linking and Embendding Database)– се појавио као одговор проблему приступања комплексније организације података, као што су текст фајлови, е-маил системи и др. То је нова верзија ODBC-а.

JDBC (Java Database Conectivity)– је еквивалент ODBC технологије намењен употреби приликом развоја апликација.

DAO (Data access object)– се јавља као решење у прављењу објектног модела за приступ бази података који спречава да дође до евентуалних грешака при програмирању ODBC, OLE DB и JDBC. Он је саставни дио Visual Basic-а. Служи за приступ MS Access базама података. DAO такође омогућава приступ ODBC изворима података и ту лежи и његов највећи недостатак. Пошто се ослања на Access, приликом приступа ODBC базама података све наредбе за базу података и сви подаци из ње морају проћи кроз овај додатни слој, што може знатно угрозити перформансе апликације.

ADO (ActiveX Data Object)– нова технологија из Microsoft-а чији је циљ да замени DAO као стандардни објектни модел за приступ базама података.

Скуп функција за комуникацију преко позива функција са базом података назива се SQL/CLI (SQL Call-Level Interface). SQL/CLI је апликациони програмски интерфејс (API) за SQL приступ базама података из програмских језика. SQL/CLI даје библиотеку функција преко којих се клијент повезује са базом података и приступа подацима са SQL упитом. Пошто је SQL/CLI веома сличан Microsoft ODBC стандарду, који се врло широко примењује, овде ће бити приказане основе овог стандарда. Са појавом објектних језика појавиле су се и нове верзије оваквог начина комуникације програмских језика и базе података у оквиру ранијих и нових развојних окружења (J2EE, .NET). У овом поглављу биће приказани основни концепти и примери коришћења за ODBC, JDBC, ADO и ADO.NET.

2.4.1. Open Database Connectivity (ODBC)

ODBC представља библиотеку стандардизованих функција (функција које имају тачно дефинисано име, улазне и излазне аргументе) које су на располагању програмеру да из програмског кода приступа базама података. ODBC је API за приступ различитим, хетерогеним базама података без измене изворног програмског кода. [1]

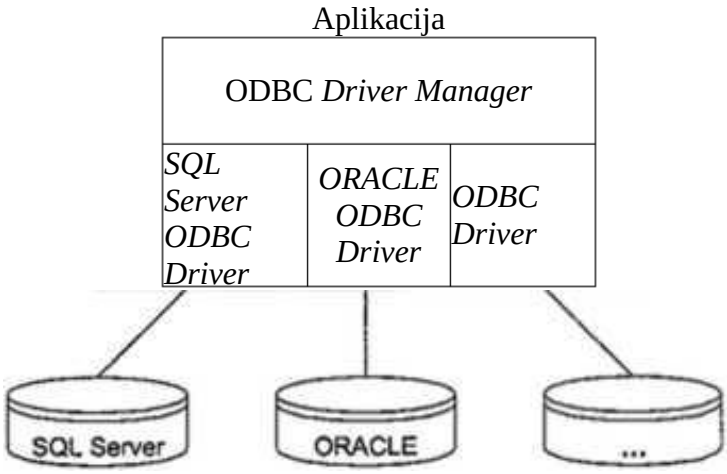
ODBC Архитектура података је на слици 1.

Све основне функције ODBC библиотеке биће објашњене коришћењем програмског језика C и следећим релационим моделом:

3d_stampa(email, stampac, dimenzije, materijal)

Materijal3DCENA(*stampacmat*, *PETG*, *NYLON*, *ABS*, *FLEXIBLE*, *PLA*, *HIPS*, *CARBNFIBREFIELD*, *ASA*, *WOODFIELD*, *PVA*)

Stampac3D(stampas, nazivstampaca, kolicinastampaca)



Слика 1. ODBC архитектура

Основне карактеристике ODBC API-ја

Да би се функције ODBC библиотеке могле користити у програму, на почетку је потребно прикључити неопходне *header* датотеке у којима се налазе дефиниције функција, типова података, структура и константи. [1]

```
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>
#include <odbcss.h>
```

Прикључивање ових *header* датотека обезбеђује креирање и манипулацију са следећим врстама рекорда (структура u C-у):

1. *Environment* (окружење) је структура која чува глобалне информације о ODBC окружењу као што су листа отворених конекција и тренутно активна конекција.
2. *Connection* (конекција) је структура која садржи податке о конекцији (назив базе података, корисничко име, статус трансакције).
3. *Statement* (наредба) је структура која садржи једну SQL наредбу. Апликација може креирати више оваквих структура.
4. *N-torke i parametri.* Ове структуре чувају информације било о параметрима упита било о n-торкама резултата.

Свака од ових структура у програму репрезентована је преко показивача на структуру који се назива handle. Ови показивачи се декларишу на следећи начин:

```
SQLHENV     henv  =   SQL_NULL_HENV;
SQLHDBC     con   =   SQL_NULL_HDBC;
SQLHSTMT    stmt  =   SQL_NULL_HSTMT;
```

Иницијализација показивача на структуру која представља ODBC окружење је увек први корак у апликацији. Без овог алоцирања меморијског простора није могуће користити ни једну функцију ODBC библиотеке. Требало би алоцирати један и само један handle на ODBC окружење. Алоцирање меморијског простора врши се позивом функције *SQLAIIocHandle(),* чији аргумент је врста показивача. Функција враћа вредност типа SQLRETURN (integer). Уколико није дошло до грешке вредност је 0 (нула), у противном вредност је различита од 0 и представља код грешке. За постављање вредности неких атрибута у структури ODBC окружења користи се функција *SQLSetEnvAttr().*Аргументи ове функције су приказани у примеру и дефинишу верзију ODBC драјвера који се користи. [1] На пример,

```
SQLRETURN retcode;
// Alociranje memorijskog prostora za ODBC okruzenje retcode = SQLAllocHandle
(SQL_HANDLE_ENV, NULL, Shenv);
// Postavljanje informacija o ODBC 3.0 aplikaciji retcode = SQLSetEnvAttr(henv,
SQL_ATTR_ODBC_VERSION,
                                    (SQLPOINTER)SQL_OV_ODBC3, SQL_IS_INTEGER);
// Alociranje memorijskog prostora za konekciju retcode =
SQLAllocHandle(SQL_HANDLE_DBC, henv, &con) ;
```

За конектовање на базу података користи се функција *SQLConnect().*

Да би се позвала функција за извршење само неке SQL наредбе неопходно је прво иницијализовати показивач на структуру која представља SQL наредбу и тиме резервисати потребан меморијски простор за податке о наредби.

// Alociranje memorijskog prostora za naredbu retcode = SQLAllocHandle(SQL_HANDLE_STMT, con, Sstmt);

Постоје два могућа начина специфицирања и извршења SQL наредбе. У многим књигама о ODBC стандарду ова два начина се називају *prepared execution* (припремљено извршење) і *direct execution* (директно извршење). [1]

Припремљено извршење се састоји из следећих корака:

1. Позив функције *SQLPrepare()* где се као аргумент прослеђује SQL наредба.
2. Позив функције *SQLBindParameter()* којом се постављају вредности параметара уколико наредба садржи маркере параметара.
3. Позив функције *SQLExecute()* за извршење SQL наредбе.

Следи пример који илуструје припрему и извршење SQL наредбе са једним параметром.

```
long brojstampaca;
brojstampaca=893;
retcode = SQLPrepare(stmt, (SQLTCHAR *)"DELETE FROM Stampac WHERE Brojstampaca = ?", SQL_NTS);
retcode = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_LONG, 0, 0, brojstampaca, 0, NULL);
retcode - SQLExecute(stmt);
```

Други начин (директно извршење) се састоји из следећих корака:

1. Позив функције *SQLBindParameter()* којом се постављају вредности параметара уколико наредба садржи маркере параметара
2. Позив функције *SQLExecDirect()* где се као аргумент прослеђује SQL наредба.

Следи пример који илуструје директно извршење SQL наредбе за ажурирање базе података.

```
retcode = SQLExecDirect(stmt, (SQLTCHAR *)"UPDATE Materijal3DCENA SET stampacmat = 'PETG' WHERE stampac=Ultimaker", SQL_NTS);
```

У случајевима када се користи SELECT наредба као резултат извршења се добија колекција n-торки које задовољавају задати услов претраживања. Резултујућа колекција n-торки (result set) смешта се у посебан меморијски простор (bafer). За приступ колонама и редовима (n-торкама) резултујуће колекције користе се *SQLBindCol()* и *SQLFetch()* ODBC функције. Позивом *SQLBindCol()* функције повезује се програмска променљива са колоном из резултујуће конекције. Функција *SQLFetch()* се користи за приступ наредном реду у резултујућој колекцији и преузимање података из колона тренутног реда у одговарајуће, претходно повезане програмске променљиве. Пример коришћења ових функција приказан је на следећем програмском коду. [1]

```
SQLCHAR     Strstampac[50];
SQLINTEGER stampacmat, indBroj, indNaziv; bool loop = true;

retcode = SQLExecDirect(stmt, (SQLTCHAR *)"SELECT
                    email,stampacmat FROM stampac", SQL_NTS);
```

```
if (retcode == SQL_SUCCESS I I retcode =- SQL_SUCCESS_WITH_INFO)
{
    SQLBindCol(stmt, 1, SQL_C_ULONG, Sstampacmat, 0, SindBroj); SQLBindCol(stmt, 2,
    SQL_C_CHAR, strstampac, 50, SindNaziv); while (loop == true)
    (
        retcode = SQLFetch(stmt) ;
        if (retcode == SQL_SUCCESS Ii retcode ==
SQL_SUCCESS_WITH_INFO)
        (
         printff'Broj stampaca: %-5d, 3d_stampa: %s \n", Broj stampaca, strstampac);
        )
        else { loop = false; break; )
    )
)
else
{
    Printf("Nije izvrsio upit\n");
    greska(stmt);
)
```

Позив процедура базе података

ODBC омогућава и стандардни начин за позивање процедура база података. Процедуре базе података позивају се преко функције *SQLExecDirect().*Знак питања се користи као маркер параметра процедуре. Синтакса за позив процедуре је следећа:
{ [? =) call *procedure_name[(?,* ?, ...)]}

За излазне параметре мора се поставити маркер параметра у позиву процедуре, а позивом функције *SQLBindParameter()* везати програмска променљива за излазни параметар процедура базе података. [1]

```
retcode = SQLBindParameter(stmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG, SQL_LONG, 0, 0,
brojStampaca, 0, NULL);

retcode = SQLExecDirect(stmt,
"(?=call DodajStampac?, '10.08.2018', 'Ultimaker', 762))", SQL_NTS);
```

Ажурирање табеле базе података из курсора

Претходно је описано како се врши ажурирање базе података преко директног извршења SQL наредбе. Ажурирање табеле базе података могуће је извршити коришћењем података из « текућег » реда именованог курсора дефинисаног и табелом која се ажурира. Текући курсор дефинише н-торку која се избацује или мења исказом *CURRENT OF* у WHERE *клаузули.* Назив курсора може се поставити из саме апликације позивом функције *SQLSetCursorName().*Уколико назив курсора није програмски постављен,ODBC *driver* аутоматски додељује назив који се касније може ишчитати коришћењем функције SQLGetCursorName().[1]

Sintaksa SQL naredbe za izmenu vrednosti u n-torci je:

```
UPDATE naziv-tabele
    SET identifikator-kolone = (izraz | NULL)
        [, identifikator-kolone = (izraz | NULL)] ...
    WHERE CURRENT OF naziv-kursora
```

Синтакса SQL наредбе за избацивање n-торке је:

```
DELETE FROM naziv-tabele WHERE CURRENT OF naziv-kursora
```

Управљање трансакцијама

За управљање трансакцијама у програму одговоран је програмер. Уколико се у програму позове функција за извршење SQL наредбе а не постоји отворена трансакција, ODBC *driver* ће аутоматски започети нову трансакцију. Трансакција остаје отворена све док се не потврди или поништи експлицитним позивом ODBC funkcije *SQLTransact(),*Где је вредност једног параметра COMMIT или ROLLBACK. [1]

```
// Potvrdjivanje transakcije
SQLTransact(SQL_HANDLE_DBC, con, SQL_COMMIT);
```

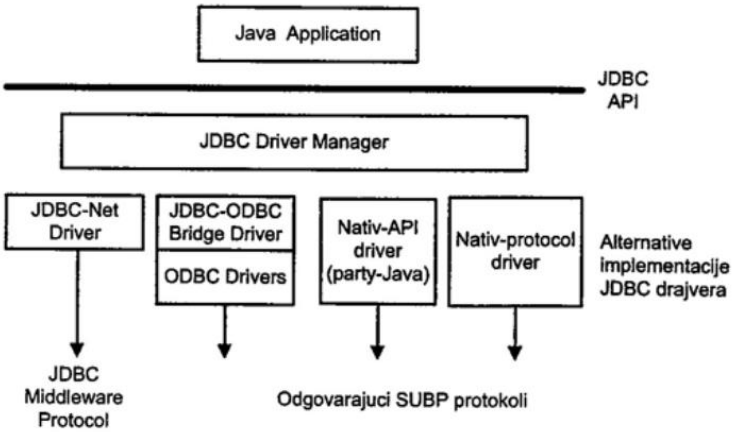
На крају сваког програма који користи *ODBC* библиотеку функција требало би ослободити све меморијске ресурсе алоциране за *ODBC* окружење и конекције. За ту сврху користе се функције *SQLDisconnect()* i *SQLFreeHandle().*

```
// Oslobadjanje zauzetog prostora SQLDisconnect(con);
SQLFreeHandle(SQL_HANDLE_DBC, con);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

2.4.2. Java Database Connectivity (JDBC)

JDBC (Java Database Connectivity) je standardni Java interfejs za povezivanje Jave sa relacionim bazama podataka definisan od strane Sun Microsystems. JDBC omogućava Java aplikacijama, apletima i servletima pristup bazama podataka kao i drugim tabelarnim izvorima podataka. [1]

JDBC архитектура дата је на Слици 2.



Слика 2. JDBC архитектура

JDBC архитектура сачињавају следеће компоненте: *JDBC Driver Manager* који има функцију посредника између Java програма і *дрaјвера* који представљају услужне програме између Јава програма и конкретних SUBP-ова. JDBC спецификацијом су дефинисана два типа итерфејса API-ја. Један је *JDBC Driver API* који је неоходан за имплементацију драјвера и стандардизује комуникацију између Driver Manager-а и драјвера. Други је кориснички *JDBC* ***API*** који дефинише стандардни приступ Java програма бази података. [1]

Постоје 4 типа **JDBC** driver-а. Сваки тип драјвера дефинише имплементацију са највишим нивоом платформске назависности, преформанси и администрације. Типови драјвера су следећи:

1. *JDBC-ODBC Bridge* - преводи све **JDBC** позиве у **ODBC** позиве и прослеђује их **ODBC** драјверу. Најчешће је **ODBC** дирвер присутан на клијентској машини па је и приступ бази на страни клијента.
2. *Native-API/partly Java driver* - преводи **JDBC** позиве у специфичне позиве функција дефинисане од произвођача SUBP-а као што су SQL Server, Informix, Oracle ili Sybase. Библиотека функција специфичног SUBP-а треба да се учита на свакој клијентској машини и није могуће овај тип драјвера користити за Web апликације.
3. *Net-protocol/all Java driver* - омогућава трослојну архитектуру система јер **JDBC** позиве прослеђује на средњи слој. Сервер на средњем слоју преводи захтеве клијента у позиве функција базе податаке, најчешће преко 1 и 2 типа драјвера.
4. *Native-protocol/all Java driver* - конвертује **JDBC** позиве у специфичан SUBP протокол тако да клијентска апликација може комуницирати директно са базом података. За овај тип драјвера није потребан специфичан софтвер јер се драјвер инсталира динамички. [1]

Параметризоване наредбе

JDBC омогућава дефинисање и параметризованих SQL наредби. Параметри се дефинишу у тексту SQL наредбе преко знака питања који чувају место за одговарајуће аргументе у тренутку извршавања наредбе. За креирање параметризоване наредбе користи се метода *preparedStatement(Q)* која као аргумент има текст наредбе Q. Ова метода омогућава парсирање наредбе пре њеног извршавања и креира објекат типа *PreparedStatement.* Након креирања одоговарајућег објекта а пре извршења наредбе потребно је доделити вредности одговарајућим параметрима. За то се користе методе чији је општи облик set<SQLTIP>(i,v) (нпр. setString(i.v), setDouble(i.v)). Први аргумент *i* методе дефинише редни број појављивања параметра у наредби док други *v* дефинише вредност парамтера који по типу одговара позваној методи. Извршавање наредбе омогућено је преко метода *executeUpdate()* и *executeQuery()*у зависности од типа наредбе. Овај начин се препоручује уколико се више пута жели извршавање исте SQL наредбе са различитим вредностима параметара. [1]

```
PreparedStatement pst = conn.prepareStatement(
"INSERT INTO Materijal3DCENA (PETG, NYLON, ABS, FLEXIBLE, PLA) VALUES <?,?,?,?)");
        pst.setBigDecimal(1, new BigDecimal(10); pst.setLong(2,(long)l);
pst.setLong(3,(long)55); pst.setDouble(4,(double)3*2.5); pst.executeUpdate();
```

Позив процедура базе података

JDBC омогућава и стандардни начин за позивање процедура база података. Процедуре база података позивају се преко објекта типа *CallableStatement.* Креирање овог објекта је омогућено преко методе *prepareCall.* Знак питања се користити као ознака за параметре процедуре. Синтакса за позив процеруре је следећа: [1]

{[? =] call *procedure_name*[(?, ?, ...)]}

Уколико процедура базе података враћа вредности тада их је неопходно регистровати пре позива. Регистровање се врши методом *registerOutParameter.*

```
CallableStatement cstmt =
                    con.prepareCall ("(call izracunajIznos (?, ?)}"); cstmt.registerOutParameter(1,
java.sql.Types.TINYINT); cstmt.registerOutParameter(2, java.sql.Types.DECIMAL);
ResultSet rs = cstmt.executeQuery();
```

Ажурирање табеле базе података преко резултујућег сета

Објекат типа *ResultSet* могуће је користити и за ажурирање табеле базе података. У *ResultSet*-у је могуће додати нови ред или обрисати постојећи, изменити вредности поља у некој n-торки. Да би се омогућило ажурирање потребно је приликом креирања наредбе поставити одоговарајуће вредности параметара, тј. омогућити навигацију кроз резултујући сет и његово ажурирање. [1]

```
Statement stmt = con.createStatement(
                        ResultSet,TYPE_SCROLL_SENSITIVE,
                        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = executeQuery("SELECT PETG, NYLON, ABS, FLEXIBLE, PLA as iznos FROM
Stampac");
```

Да би се креирала нова *n* -торка у резултујућем сету потребно је извршити позиционирање на нову *n* -торку а након попуњавања поља пренети податке у базу података позивом методе *insertRow().*Следи пример уноса нове n-торке.

```
rs.moveToInsertRow();
rs.updatelnt("brojStampaca", 15);
rs.updateDate("datum",'2018-04-08") ;
rs.updateString("nazivStampaca", "Ultimaker");
rs.updateFloat ("iznos", 10.50RSD);
rs.insertRow();
```

Као што је показано на претходном примеру за измену поља текуће n торке у резултујућем set им користе се методе *update<SQLTip>(i,v)* дефинисане по типу одговарајуће променљиве. Као први аргумент ових метода може се користити или назив или позиција колоне из SELECT наредбе за коју је краиран ResultSet. Да би се измене пренеле у базу података за постојеће n-торке у резултујућем сету користи се метода *updateRow()* за ажурирање, односно метода *deleteRow()* за брисање. [1]

Управљање трансакцијама

За транскацију је одговоран корисник. У општем случају connection објекат има постављену тзв. Auto-commit опцију којом се свака SQL наредба дефинише као трансакција и укључује имплицитан commit или rollback. Мењање дифолт карактеристика т.ј. укључивање трансакција врши се над Connection објектом и то преко метода *setTransactionlsolation* и *setAutoCommit.* У том случају морају се екплицитно дефинисати наредбе за контролу трансакција commit и rollback. [1]

На пример,

```
conn.setAutoCommit(false);
conn.setTransactionlsolation(Connection.TRANSACTION_READ_COMMITTED);
    // naredbe conn.commit();
```

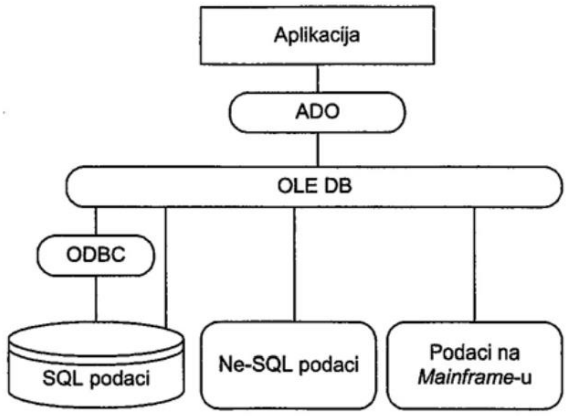
У верзији JDBC 3.0 API дефинисан је и интерфејс *Savepoint.* Овај интерфејс омогућава дефинисање контролних тачака којима се омогућава поништавање промена до дефинисане контролне тачке уместо поништавања ефеката целе трансакције.

```
Savepoint savel - conn. setSavepoint ("SAVEPOINT_1.") ;
// Rollback to kontrolne tačke save 1.
conn.rollback(save1);
```

2.4.3. ActiveX® Data Objects (ADO) i ADO.NET

Microsoft® ActiveX® Data Objects (ADO) омогућава клијентским апликацијама приступ и манипулацију са подацима из база података преко одговарајућег OLEDB *provider-a.* OLEDB *provider* представља специфичан сервис нижег нивоа за приступ конкретном систему за управљање базом података или неком другом извору података (датотека, електронска пошта итд.). Најзначајније особине ADO компоненте су: лакоћа коришћења, велика брзина приступа, мало заузеће меморије. ADO са својим језгровитим објектним моделом подржава основне концепте за изградњу клијент/сервер и вишеслојних Интернет/Интранет апликација. [1]

ADO архитктура дата је на Слици 3.



Слика 3. ADO архитектура

Основни објекти ADO компоненте су:

- Connection. Представља јединствену сесију ка извору података. Над овим објектом се може позвати метода *Execute*()која директно извршава задату SQL наредбу.
- Command. Овај објекат служи за припремање и извршавање SQL наредбе која може бити параметризована. Објекат Command поседује конекцију *Parameters* која садржи један или више параметара који су атрибути класе *Parameter*. Вредности ових атрибута задају се у току извршења програма.
- Recordset. Представља објекат ADO компоненте који садржи резултате SELECT упита над базом података и омогућава манипулацију са подацима редова из резултата. [1]

Основне карактеристике ADO компоненте

Коришћење ADO објеката за приступ и манипулацију подацима базе података биће објашњено коришћењем програмском језика Visual Basic над следећим релационим моделом:

3d_stampa(email, stampac, dimenzije, materijal)

Materijal3DCENA(*stampacmat, PETG, NYLON, ABS, FLEXIBLE, PLA, HIPS, CARBNFIBREFIELD, ASA, WOODFIELD, PVA*)

Stampac3D(stampas, nazivstampaca, kolicinastampaca)

Да би се објекти ADO компоненте могли користити у Visual Basic програму , на почетку је потребно у референцама пројекта укључити ADO biblioteku. Уколико се програм пише у програмском језику C++ потребно је прикључити неопходне *header* датотеке у којима се налазе дефиниције класа, функција, типова података, структура и константи.

Декларисање и инстанцирање објекта конекције је увек први корак у апликацији. На следећем примеру илустровано је инстанцирање објекта конекције, постављање параметара потребних за отварање конекције и само конектовање на базу података. [1]

```
'*** Deklaracija promenljivih Dim objKonekcija As ADODB.Connection
   Dim strKonekcija As String

'*** Instanciranje objekta konkecije
   Set objKonekcija = New ADODB.Connection
'*** Postavljanje parametara potrebnih za otvaranje konekcije strKonekcija = "Provider=sqloledb;Data Source=DBServer;Initial Catalog=Test;User Id=sa;Password=sa; "

'*** Otvaranje konekcije
   objKonekcija.Open strKonekcija
```

Да би се позвало извршење само неке SQL наредбе неопходно је прво инстанцирати објекат *Command* и тиме резервисати потребан меморијски простор за податке о наредби. Као што се у наредном примеру види, после инстанцирања објекта наредбе, објекат активне конекције се додељује атрибуту *ActiveConnection* посматране наредбе. тиме је инстанцирана наредба повезана за активну конекцију. Атрибуту *CommandText* додељује се текст SQL наредбе а позивом методе *Execute*() извршава се SQL наредба.

```
   Dim objKomandaStampac As ADODB.Command
   Set objKomandaStampac = New ADODB.Command
   objKomandaStampac.ActiveConnection = objKonekcija
```

```
objKomandaStampac.CommandType = adCmdText
objKomandaStampac.CommandText="UPDATE Stampac SET UkupnaVrednost=" &
        mUkupnaVrednost s " WHERE BrojStampaca=" & brojStampaca
objKomandaStampac.Execute
```

Оно што карактерише ADO компоненту је веома лако манипулисање са подацима из резултујуће афтер схаве (rekordset). За ту сврху користи се ADO Recordset објекат. Инстанцирање, постављање атрибута рекордсета и отварање самог рекордсета (извршавање SELECT наредбе) приказан је на следећем примеру. [1]

```
Dim rstStampac As ADODB.Recordset
Set rstStampac - New ADODB.Recordset Set rstStampac.ActiveConnection = objKonekcija
rstStampac.CursorLocation = adUseClient rstStampac.CursorType = adOpenStatic
rstStampac.LockType = adLockOptimistic
SQLupit = "SELECT * FROM Stampac"
rstStampac.Open SQLupit, , , , adCmdText
```

У приказаном примеру специфициран је рекордсет који је статички и клијентски, што значи да ће се све n- торке које су резултат упита налазити у баферу клијента. Атрибут *CursorType* узима једну од четири могуће вредности (*adOpenDynamic*, *adOpenKeySet, adOpenStatic, adOpenForwardOnly).* У случају да је избарана опција adOpenForwardOnly кроз рекордсет је могуће проћи само једном. У осталим случајевима ради се о скролујућем рекордсету (scrollable recordset), по којем се може кретати напред-назад, позиционирати на први или последњи ред. За позиционирање у рекордсету користе се методе *MoveFirst, MoveLast, MoveNext* и *MovePrevious.* За приступ вредностима поља тренутног рекорда у рекордсету користи се колекција *Fields.* Подразумева се да ће програмер декларисати променљиве у програму које су истог типа као ја колоне рекордсета. На следећем примеру илустровано је коришћење поменутих метода. [1]

```
rstStampac.MoveFirst While Not rstStampac.EOF
   brojStampaca = rstStampac.Fields("BrojStampaca").Value datum =
rstStampac.Fields("Datum").Value rstStampac.MoveNext Wend
```

Параметризоване наредбе

ADO објектни модел подржава извршавање параметризованих SQL наредби. Да би се означило место где треба да се убаци вредност параметра у време извршења наредбе користи се ознака параметра. И у ADO наредбама као маркер параметра користи се знак питања (?).*За дефинисање параметра користи се објекат Parameter.* Инстанцирани параметри се у колекцију параметара наредбе убацују у редоследу како су дефинисани маркери параметара у SQL наредби. Метода *CreateParameter()* објекта Command као аргументе има назив параметра, тип података поља табеле, тип параметра (улазни или излазни), као ја опционо вредност параметра. Да би се нагласило да се ради о параметризованој наредби атрибуту *Prepared* се додељује вредност True. На следећем делу кода илустровано је креирање параметара наредбе, њихово убацивање у колекцију параметара наредбе и извршавање SQL наредбе. [1]

```
Dim brojStampaca As Long
Dim objKomandaStampac As ADODB.Command
Dim param As ADODB.Parameter

Set objKomandaStampac = New ADODB.Command objKomandaStampac.ActiveConnection =
objKonekcija
objKomandaStampac.CommandText="DELETE FROM Stampac WHERE BrojStampaca = ?"
objKomandaStampac.CommandType = adCmdText objKomandaStampac.Prepared = True

Set param= objKomandaStampac.CreateParameter("pBrojStampaca", adBiglnt,
                                              adParamlnput)
objKomandaStampac.Parameters.Append param broj Stampaca=8 93
    objKomandaStampac.Parameters("pBrojStamp
aca").Value = brojStampaca obj
KomandaStampac.Execute
```

Позив процедура базе података

ADO омогућава позив процедура базе података коришћењем раније детаљно описаног објекта *Command.* Припрема наредбе за позив процедуре базе података састоји се из додељивања назива процедуре атрибуту *CommandText,* постављања константе adCmdStoredProc атрибуту *CommandType* и креирање потребног броја параметара, колико процедура базе података захтева. У наредном примеру приказан је позив процедура базе података коришћењем *Command* објекта.

```
Set objKomandaStampac = New ADODB.Command objKomandaStampac.ActiveConnection =
objKonekcija objKomandaStampac.CommandText = "IzracunajBroj"
objKomandaStampac.CommandType = adCmdStoredProc
Set param= objKomandaStampac.CreateParameter("pBrojStampaca", adBiglnt,
                                              adParamlnput)
objKomandaStampac.Parameters.Append param
objKomandaStampac.Parameters("pBrojStampaca").Value = 750
Set param= objKomandaStampac.CreateParameter("plznos", adCurrency,
                                              adParamOutput)
objKomandaStampac.Parameters.Append param obj KomandaStampac.Execute
```

Ажурирање табеле базе података преко резултујућег сета

Једна од особина ADO компоненте је могућност ажурирања табеле базе података преко рекордсета. У рекордсету је могуће додати нови ред, изменити вредности поља у неком реду или чак обрисати ред. Позивом методе *Update()* све измене извршене у рекордсету се преносе на табелу у бази података. Уколико је нарушен неки интегритет базе података биће изгенерисана грешка.

```
Set rstStampac = New ADODB.Recordset Set rstStampac.ActiveConnection =
objKonekcija rstStampac.CursorLocation = adUseClient rstStampac.CursorType =
adOpenStatic rstStampac.LockType = adLockOptimistic

rstStampac.Open "SELECT * FROM Stampac", , , , adCmdText
rstStampac.AddNew
rstStampac("BrojStampaca") = brojStampaca rstStampac{"Datum") = Datum
rstStampac("NazivKupca") = NazivKupca rstStampac("Napomena") = Napomena rstStampac.Update
```

Управљање трансакцијама

За управљање трансакцијама у програму одговоран је програмер. У апликацији се пре почетка извршавања наредби позива метода *BeginTrans()* над објектом *Connection* и тиме се отпочиње нова трансакција. Трансакција остаје отворена све док се не потврди или поништи експлицитним позивом методе *CommitTrans()* или *RollbackTrans()* чиме се потврђује односно поништава актуелна трансакција. [1]

```
'*** otpočinjanje transakcije objKonekcija.BeginTrans

... izvršavanje SQL naredbi ...

'*** Potvrđivanje transakcije objKonekci ja.ConunitTrans
ili
'*** Poništavanje transakcije
objKonekcija.RollbackTrans
```

Перзистентност рекордсета

Још једна корисна метода Recordset објекта је *SaveQ,* која омогућава снимање садржаја рекордсета у датотеку у XML формату. Касније се садржај те датотеке може учитати у било који други рекордсет коришћењем методе *Open().*

```
'*** Snimanje sadrzaja rekordseta u XML formatu rstSporniStampaci.Save
”C:\Temp\SporniStampaci.xml", adPersistXML

'*** Preuzimanje sadržaja XML datoteke u novi rekordset rstStampaci.Open
"C:\Temp\SporniStampaci.xml"
```

ADO.NET

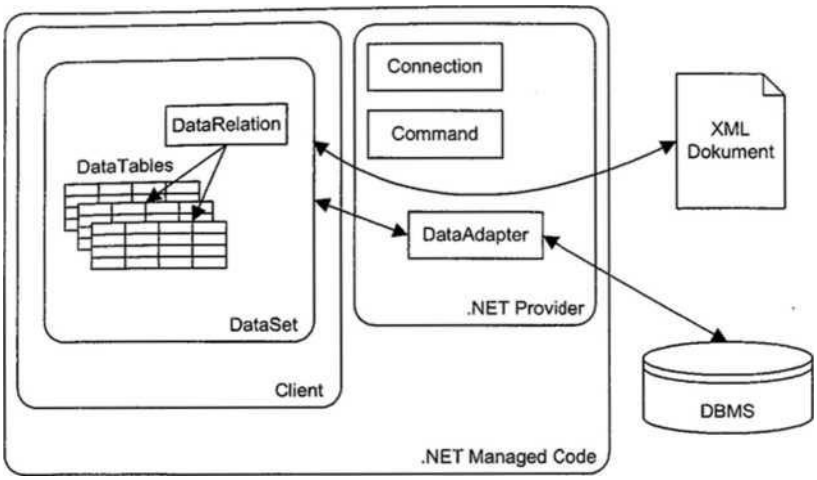
Са појавом новог развојног окружења.NET, као његов саставни део јавља се и ADO.NET. Иако именом и основним концептима подсећа на стару ADO технологију за приступ различитим изворима података, ради се о радикално измењеној технологији приступа базама података. Док се ADO користи у Windows апликацијама које комуницирају преко DCOM, ADO.NET је дизајниран за коришћење како у интерној мрежи тако и у Интернет апликацијама. Најприметнија разлика је између објеката ADO *Recordset*-а и .NET *DataSet-a.* [1]

Recordset садржи резултат једног упита и најчешће одржава отворену конекцију ка бази података над којом је извршен упит. Иако је могуће лако проследити *Recordset* DCOM клијенту, не постоји лак начин да се *Recordset* проследи неком броусеру преко Interneta, осим ако то није Internet Explorer. [1]

С друге стране, *DataSet* може садржати резултате више упита, чак из различитих извора података. DataSet је практично меморијски кеш за податке из различитих извора.

Као што се види на Слици 4. *DataSet садржи један или више објеката DataTable. DataTable* објекат садржи резултат неког упита. Између *DataTable* објеката, *DataSet* може успоставити и одржавати везе коришћењем *DataRelation објекта.* Такође, могуће је дефинисати нека ограничења над повезаним пољима (примарни кључ/спољни кључ) из два *DataTable* објекта , чиме се одржава интегритет података. [1]

DataSet је могуће лако серијализовати, трансформисати у XML формат што омогућава приказ у било којем броусеру и пренос преко Интернета коришћењем HTTP-а. Поред тога, могуће је шему *DataSet*-а снимити у XML шему, као и XML шему учитати у *DataSet.* Слично као и код ADO *Recordset-a,* могуће је снимање садржаја табела *DataSet-a* у XML документ и учитавање комплетног садржаја произвољно сложеног XML документа у *DataSet.* [1]



Слика 4. ADO.NET архитектура

Клијентска апликација у .NET окружењу приступа изворима података преко.NET *Data Provider-a,* софтверске компоненте која се пише за одређени извор података (нпр.SQL .NET *Data Provider за* SQL Server, OLE DB .NET *Data Provider* за Oracle itd.). У оквиру .NET *Data Provider-a* налазе се класе *Connection* і *Command,* Чија је намена идентична као и у ADO компоненти. Новина је класа *DataAdapter* која служи за приступ одређеном извору података, прослеђивање упита до базе података и пребацивање резултата упита у *DataSet.* За ту сврху *DataAdapter* користи атрибут *SelectCommand* којем је додељен објекат класе *Command* који садржи SQL SELECT наредбу. Један објекат *DataSet* може бити попуњен подацима из неколико различитих извора података, где се за сваки извор података инстанцира нови објекат класе *DataAdapter.* Такође, објекат класе *DataAdapter* врши додавање, измену и брисање n-торки у бази података, на основу атрибута *InsertCommand, UpdateCommand* и *DeleteCommand* којима су додељени објекти класе *Command* који садрже одговарајуће SQL наредбе (INSERT, UPDATE, DELETE). [1]

3. Процедуре базе података – Persistent Stored Module (PSM)

Сви комерцијални SUBP-ови нуде могућност смештаја процедура и функција у бази података. Ови делови процедуралног кода су писани у једноставном језику који омогућава процедурално коришћење SQL-а додавањем наредби контроле тока извршења, условних наредби, обраду изузетака и других програмских конструкција. SQL:1999 је дефинисао стандард за процедуре и функције базе података SQL/PSM. Треба напоменути да су произвођачи SUBP-а имплементирали ову могућност пре дефинисаног стандарда, те је и сам SQL/PSM стандард настао по узору на водеће комерцијалне SUBP-ove. Разлике између ових процедуралних надоградњи различитих SUBP- ова су претежно синтаксне.[1]

SQL/PSM дефинише синтаксу и семантику језика за рад са базом података који се односи на декларисање и манипулисање перзистентним рутинама у SQL- сервер модулима. Термином рутина обједињују се једним именом процедуре, функције и методе. PSM процедуре и функције представљају самосталне програмске целине које се у шеми базе података идентификују преко свог имена. [1]

3.1. Структура PSM рутина

SQL/PSM модули се дефинишу као колекције процедура и функција. Процедуре се дефинишу на следећи начин:

CREATE PROCEDURE <naziv> ((<vrstaParam nazivParam tipParam>, ...]) *lokalne_deklaracije*
BEGIN

izvrsne_naredbe
END;

Процедуре могу али не морају имати параметре. За сваки параметар наводи се врста параметра која може бити IN, OUT и INOUT, назив параметра и тип параметра који може бити неки од дефинисаних SQL типова података INTEGER, REAL, VARCHAR, DATE, BOOLEAN итд. Врсте параметара су дефинисане на следећи начин:

- IN параметри су параметри који се преносе као улазни, тј. приликом позива имају дефинисану вредност и не могу се мењати у телу процедуре.
- INOUT параметри су параметри који се преносе као улазни, а процедура их враћа као излазне, тј. приликом позива имају дефинисану вредност и могу се мењати у телу процедуре.
- OUT параметри су параметри које процедура враћа као излазне, тј. приликом позива немају дефинисану вредност и добијају вредност у телу процедуре. [1]

Функције и процедуре у PSM имају сличну структуру. За разлику од процедура, функције имају RETURNS клаузулу и сви параметри функције морају бити IN параметри, тј. не могу се користити остале ознаке врсте параметра. Декларација функције има следећу синтаксу:

CREATE FUNCTION <naziv> ([<nazivParam tipParam>, ...]) RETURNS
<SQLtip>
lokalne_deklaracije
BEGIN
izvrsne_naredbe
END;

Основну структуру PSM рутина (и функција и процедура) чине декларације SQL променљивих и извршни део који чине блок наредби дефинисан између *BEGIN* и *END*. Блокови наредби се могу угњеждавати. [1]

Све променљиве које се користе у извршном делу PSM рутина морају бити декларисане. Примери декларације променљивих:

```
DECLARE IBroj INTEGER;
DECLARE IKol        REAL = 0;
DECLARE IDatum DATE = NULL;
```

У Oracle језику за процедуре базе података, PL/SQL-у, може се декларисати променљива имплицитно:

```
DECLARE sifra Stampac.brojStampaca%TYPE;
DECLARE rac Stampac%ROWTYPE;
```

Исказ *%TYPE* значи да променљива *sifra* треба да буде истог типа и дужине као колона brojStampaca табеле Stampac. Исказ *%ROWTYPE* значи да је променљива *rac* запис (слог) који има поља која по броју, редоследу и типу одговарају колонама табеле *Stampac.* Овакво декларисање омогућава максималну независност програма од података. [1]

PSM променљиве се у SQL наредбама користе без икаквог префикса.

Основне наредбе које се користе у PSM рутинама су SQL наредбе.

Изрази се у PSM рутинама формирају на исти начин као у SQL- у, тј. коришћењем следећих релационих оператора: =, !=, <, >, <=, >=, аритметичких оператора: +, *, / и логичких оператора: AND, OR i NOT. У изразима који се пишу ван SQL наредби могу учествовати само променљиве декларисане у PSM рутини. Поред оператора, у изразима се могу користити и SQL функције, осим групних функција AVG, SUM, MIN, MAX, COUNT.

PSM модули садрже очигледне наредбе за управљање током извршавања програма као што су наредбе селекције, IF...THEN...ELSE i CASE, затим наредбе итерације LOOP, WHILE i REPEAT, наредбе за промену тока програма LEAVE и ITERATE.

На пример, функција којом се додељују различити проценти попуста у зависности од датог износа илуструје коришћење IF селекције. [1]

```
CREATE FUNCTION dajPopust(izn REAL) RETURNS REAL DECLARE popust
REAL;
BEGIN
   IF izn BETWEEN 0 AND 1000 THEN SET popust =      0;
   ELSEIF izn BETWEEN 1001 AND 2500 THEN SET popust = 10;
   ELSEIF izn BETWEEN 2501 AND 5000 THEN SET popust = 15;
   ELSE
          SET popust = 20;
   END IF;
   RETURN popust;
END;
```

Блок наредби дефинисан између кључних речи BEGIN i END може бити специфициран као *ATOMIC.* ATOMIC клаузула дефинише све SQL наредбе које се налазе у блоку као једну трансакцију. Блок наредби дефинисан преко калузуле ATOMIC не може садржати наредбе за управљање трансакцијама COMMIT и ROLLBACK. Блок наредби је имплицитно дефинисан као NOT ATOMIC, али се ова клаузула може и експлицитно навести.BEGIN [NOT] ATOMIC

3.2. Упити у PSM модулима

За SQL упите који враћају тачно једну n- торку може се користити наредба SELECT INTO којом се резултат упита смешта у променљиве декларисане у PSM рутини. На пример,

```
DECLARE pCena REAL;
BEGIN
    SELECT prodajnaCena INTO pCena FROM Proizvod WHERE
  sifraProizvoda = 10;
END;
```

Уколико SQL наредба враћа више n- торки користи се курсор који омогућава приступ n- торкама резултата запис по запис.

Синтакса за декларацију курсора је следећа:

```
DECLARE CURSOR <naziv> FOR select_naredba;
```

Манипулација курсором је омогућена преко наредби OPEN, FETCH i CLOSE. На пример,

```
CREATE FUNCTION dajUkupanPorez (pBroj INTEGER) RETURN REAL DECLARE
  suma REAL;
  DECLARE iznosPoreza REAL;
  DECLARE Not_Found CONDITION FOR SQLSTATE '02000' ;
  DECLARE stavkaCur CURSOR FOR
        SELECT brojStampaca, sifraProizvoda, iznosPoreza FROM StavkaStampaca
        WHERE brojStampaca = pBroj;
BEGIN
  OPEN stavkaCur; stPetlja: LOOP
    FETCH stavkaCur INTO iznosPoreza;
    IF Not_Found THEN LEAVE StPetlja; END IF;
    SET suma = suma + iznosPoreza;
  END LOOP;
  CLOSE stavkaCur;
  RETURN suma;
END;
```

У PSM- у је дефинисан и други начин коришћења курсора преко *for* конструкције. На овај начин се омогућава навигација кроз курсор без експлицитног отварања и затварања курсора. Треба нагласити да се SELECT наредба везана за курсор извршава приликом извршавања наредбе OPEN, односно првог извршавања наредбе FOR. Општи облик наредбе FOR је,

```
  FOR <naziv kursora> AS CURSOR FOR select_naredba DO
      i zvrsne_na redbe END FOR;
```

```
CREATE FUNCTION dajUkupanPorez (pBroj INTEGER) RETURN REAL DECLARE
  suma REAL;
BEGIN
  FOR stavkalter AS CURSOR FOR
        SELECT brojStampaca, sifraProizvoda, iznosPoreza FROM stavkaStampaca
        WHERE brojStampaca = pBroj;
  DO
    SET suma = suma + iznosPoreza;
  END FOR;
  RETURN suma;
END; [1]
```

3.3. Обрада грешака

За обраду грешака користи се посебан концепт који се зове изузетак (EXCEPTION). Механизам изузетака се може поделити у два дела: покретање изузетка и обрада изузетка.

Покретање изузетака најчешће обавља сам SUBP, а понекад и програмер, док је обрада изузетака искључиво задатак програмера. Сваки PSM блок може, али не мора, да има део за обраду изузетака. [1]

Ако у програму треба покренути изузетак користи се наредба *SIGNAL* иза које се пише назив изузетка.

SIGNAL <naziv izuzetka>;

Уколико се у оквиру акција обраде изузетка жели покренути нови изузетак тада се користи наредба RESIGNAL. [1]

У PSM рутини потребно је декларисати све изузетке који ће се обрађивати. Декларацијом изузетка се именује стање које представља неку изузетну ситуацију. Општа синтакса за декларацију изузетка је следећа,

DECLARE <naziv izuzetka> CONDITION [FOR SQLSTATE < vrednost>)

Уколико је при декларацији изузетка наведена клаузула FOR SQLSTATE тада се врши повезивање декларације изузетка са изузетком дефинисаним од стране SUBP- а који се идентификује преко вредности SQLSTATE променљиве. SQLSTATE променљива је представљена као низ од 5 карактера. Стандардом је дефинисана структура променљиве и неколико предефинисаних вредности док су остале имплементационо зависне. У оквиру истог опсега важења не могу постојати декларисана два изузетка са истом вредношћу променљиве SQLSTATE. [1]

За обраду изузетака потребно је декларисати *hendler* којим се дефинише акција обраде. Општа синтакса за декларацију хендлера је следећа,

DECLARE <tipHendlera> HANDLER FOR <lista izuzetaka>
<akcija hendlera>

Уколико се при извршавању наредби блока појави изузетак за који је дефинисана обрада, тада ће се извршити акција дефинисана преко хендлера, а након тога ће систем у зависности од типа hendlera радити следеће:

CONTINUE - наставити са радом, извршавајући следећу наредбу блока - *EXIT* - прекинути извршавање блока наредби, тј. прећи на прву наредбу која се налази иза блока

UNDO - поништити све ефекте блока наредби и прекида извршавање блока (може се користити само у оквиру ATOMIC блока наредби)

Преко истог хендлера се може обрадити већи број изузетака навођењем листе декларисаних изузетака, вредности SQLSTATE променљиве или предефинисаних изузетака SQLEXCEPTION, SQLWARNING i NOT FOUND. [1]

На пример,

```
DECLARE 1Datum DATE;
DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING;
BEGIN
    DECLARE pogresanDatum CONDITION;
    DECLARE CONTINUE HANDLER FOR pogresanDatum
```

```
            INSERT INTO greska(tekst) VALUES("pogresna vrednost datuma");
      BEGIN

        IF 1Datum > CURRENT_DATE THEN
         SIGNAL pogresanDatum;
         END IF;
        END;
END;
```

3.4. Коришћење PSM процедура и функција

PSM процедуре и функције представљају објекте шеме базе података. Као што смо видели, њихово креирање је омогућено преко наредбе CREATE. Измена ових објеката врши се наредбом ALTER а избацивање из шеме базе података преко наредбе DROP. [1]

SQL дозвољава постојање више рутина са истим називом - полиморфизам. Због тога све рутине имају два назива, други назив представља идентификатор рутине у шеми базе података. Додељивање назива рутине омогућено је преко клаузуле *SPECIFIC* Уколико се не наведе ова клаузула, систем сам додељује назив рутини.

```
CREATE PROCEDURE izstampaj( broj INT)
SPECIFIC proc2 BEGIN

END;

ALTER SPECIFIC proc2 PROCEDURE ...
DROP SPECIFIC proc2 PROCEDURE ...
```

Осим имена рутина за њену манипулацију могу се дефинисати и друге клаузуле. Једна од њих је индикација приступа SQL. Могуће опције су:

- не садржи SQL наредбе -*NO SQL*,
- садржи само SQL наредбе *CONTAINS SQL,*
- садржи само SQL наредбе које читају податке *READS SQL DATA*
- да садржи наредбе које мења податке у бази података *MODIFIES SQL DATA.*

Индикације приступа READS SQL DATA и MODIFIES SQL DATA искључују једна другу. [1]

При позиву SQL функције вредност неког од аргумената наведених у позиву те функције може имати нула вредност. Ако је она спецификована као CALLED ON NULL INPUT то значи да ће систем позвати функцију чак и када су вредности свих аргумената нула вредности. У супротном, ако смо навели RETURNS NULL ON NULL INPUT, тада ће систем једноставно вратити нула вредност као резултат функције, сваки пут када се као аргумент појави бар једна нула вредност.

Као карактеристике SQL рутина може се дефинисати и клаузула DETERMINISTIC. Уколико се рутина позива више пута са истом вредношћу параметера, а дефинисана је ова клаузула, онда ће систем вратити претходно израчунату вредност без поновног извршавања рутине. Уколико не специфицирамо ову клаузулу, подразумева се NOT DETERMINISTIC чиме се означава да систем сваки пут позива извршење рутине без обзира на то да ли је она већ позивана. [1]

Уколико дефинисана SQL/PSM функција не мења податке тада се може користити у упитним изразима, или у наредбама ажурирања базе података. На пример, уколико би раније описану функцију дајизрацунајПопуст декларисали са klauyyulo READ SQL DATA тада би је могли користити на следећи начин:

```
CREATE FUNCTION dajUkupanPorez (pBroj INTEGER) RETURN REAL
READS SQL DATA
RETURNS NULL ON NULL INPUT

BEGIN

END;


SELECT nazivKupca, ukupanlznos,
        dajUkupanPorez(brojStampaca) as porez FROM Stampac
WHERE EXTRACT(YEAR FROM DATUM) = 2018;

UPDATE Stampac
    SET ukupanlznos = ukupanlznos + dajUkupanPorez(brojStampaca) WHERE
EXTRACT(YEAR FROM DATUM) = 2018;
```

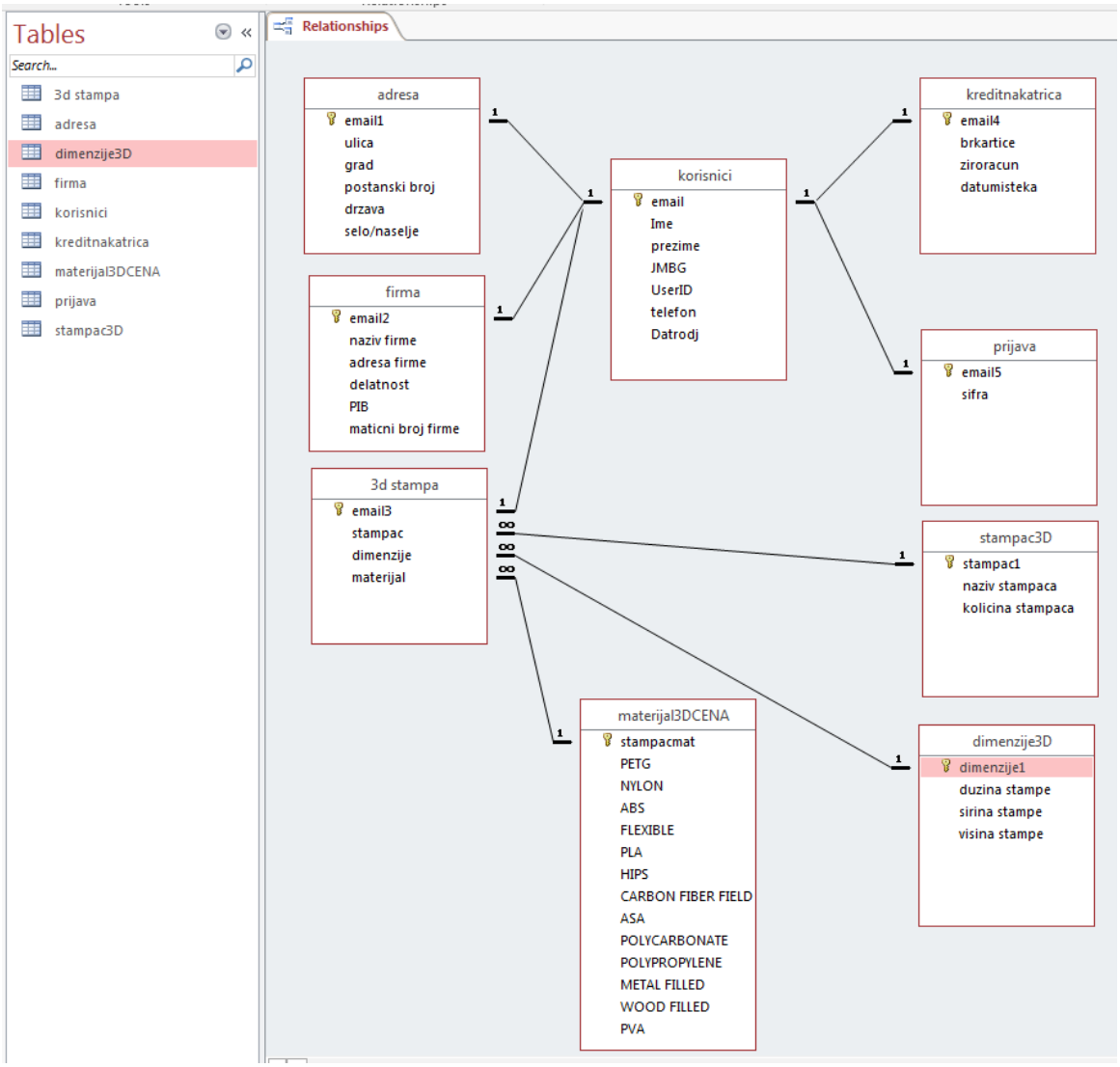
Позивање процедура базе података омогућено је преко CALL наредбе, док се функције позивају у оквиру израза SQL naredbi.

```
CALL kreirajStampac("Ivan");

SET iznos = dajUkupanPorez (pBroj);
```

Процедуре базе података могу се позивати са стране клијента. На овај начин се може дефинисати контрола приступа и сигурност BP јер се корисницима даје право да позову процедуру која, на пример, ажурира табелу , али им се онемогућава директан приступ тој табели. [1]

Дакле, поред података о регистованим корисницима, пожељно је имати и податке о регистрованим фирмама или пак приватним лицима које нуде услуге. Сваки корисник може да погледа све понуде које нуде други корисници овог сајта али може само своје понуде да мења. Све што је потребно да један корисник сајта види јесте место где се налази њему најближи 3Д штампач, број телефона понуђача, цену по којој може одштампати свој модел, као и димензије највећег предмета које може продавац одштампати на свом штампачу. Сваки власник може имати више штампача, али један штампач може имати само једног власника, осим када два власника имају исте штампаче, те би у том случају дошло до преклапања. Сваки штампач има своју дужину штампе, ширину штампе и висину штампе. Сваки власник може да зада да има више истих или више различитих штампача. Потребно је да власник може одабрати које материјале може штампати на свом штампачу и цену за сваки материјал посебно. Такође је потребно да власници имају унет број рачуна како би се могло извршити онлајн плаћање модела.

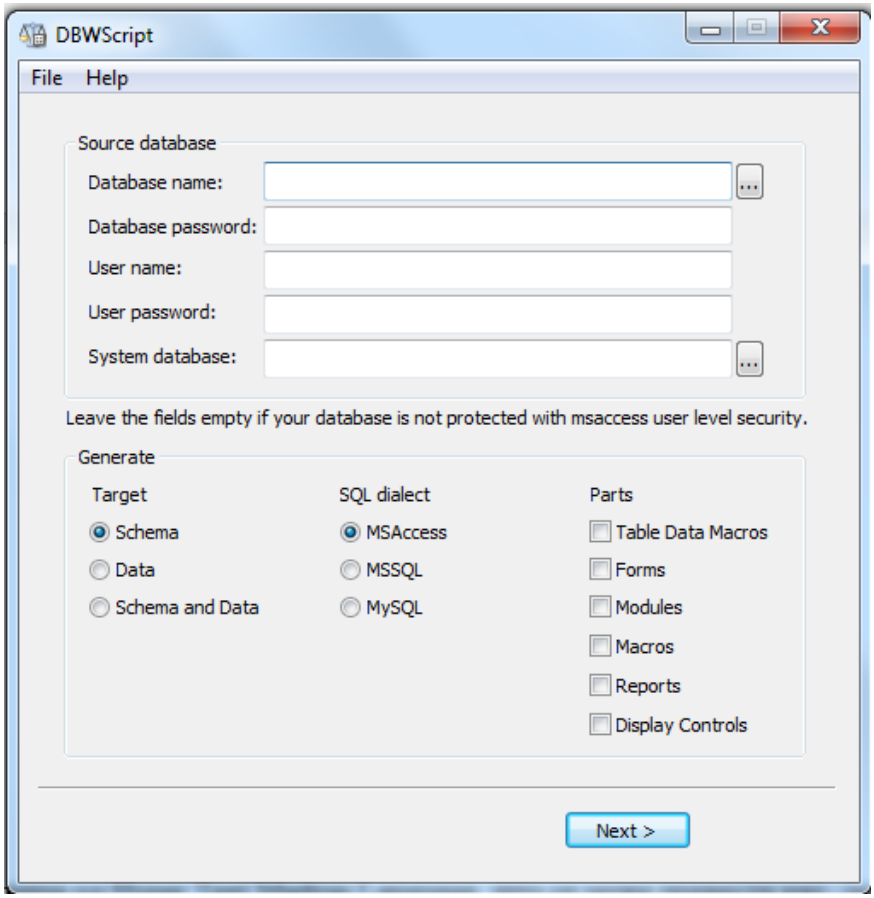
Дате табеле са потребним колонама и везама дате су на слици 6.



Слика 6. Везе између табела.

5. Израда погодне базе за спајање на виртуелни хост.

За спајање базе података коју смо креирали у Access-у са сајтом потребно је пребацити ту исту базу из формата који је креиран чувањем за сада празне базе а то је формат .accdb, у формат са екстензијом .sql. Тај поступак ћемо одрадити преко једног једноставног и пре свега веома квалитетног програма који одради посао максимално добро без икакве грешке, што је такође јако битно, а за разлику од других код којих се дешава да не ставе добро кардиналност, или да не споје табеле, односно да препакује само голе табеле без икаквих веза. Тај програм је DBWScript (*Слика 7.*).



Слика 7. Изглед програма DBWScript.

Функционисање програма је веома једноставно; у поље: Database name је потребно унети где нам се налази сачувана база података. Потребно је поред тога изабрати као SQL dialect: MySQL. Као Target бирамо Schema, из разлога јер нам је табела потпуно празна и податке ћемо први пут у њу унети путем сајта.

4. Израда Access базе података за дељење на веб-у

За прављење апликација за Веб базе података могу се користити Access 2016 и Access услуге.

Најпре ће бити направљен релациони модел а затим и све табеле у траженом програму.

S={3d stampa(email, stampac, dimenzije, materijal)

Adresa(email, ulica, grad, postanski broj, drzava, selo/naselje)

Dimenzije3D(dimenzije, duzina stampe, sirina stampe, visina stampe)

Firma(email, naziv firme, adresa firme, delatnost, PIB, maticni broj firme)

Korisnici(email, ime, prezime, JMBG, UserID, telefon, datrodj)

Kreditnakartica(email, brkartice, ziroracun, datumisteka)

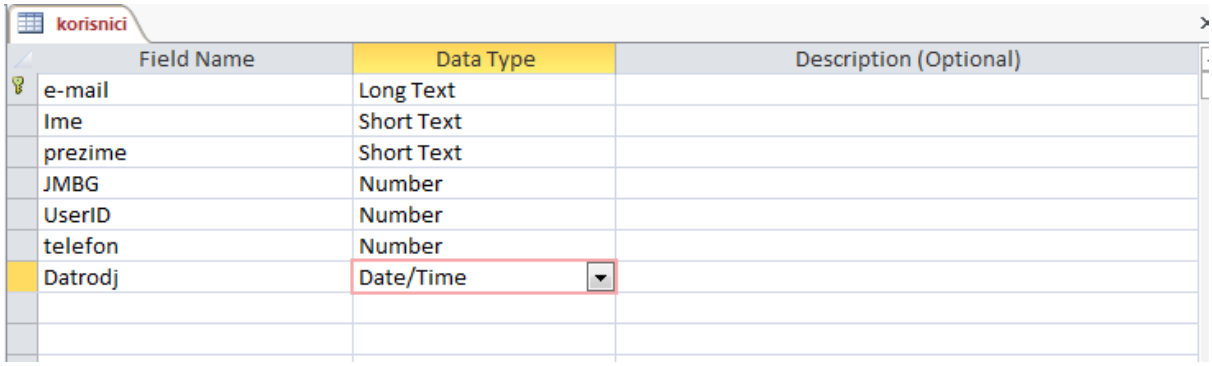
Material3DCENA(stampacmat, PETG, NYLON, ABS, PLA, FLEXIBLE, HIPS, CARBONFIBREFIELD, ASA, POLYCARBONATE, METALFIELD, WOODFIELD, PVA)

Prijava(email, sifra)

Stampac3D(stampac, nazivstampaca, kolicina stampaca)

}

Идеја је следећа: Биће направљена база података у којој ће се држати подаци о корисницима (име, презиме, ЈМБГ, телефон, датум рођења, кориснички број) (*Слика 5.*). Даље, у овом случају се ради о сајту на коме корисници могу наћи најближи себи или пак најповољнији сервис 3Д штампе, па чак и физичко лице или студент који има свој 3Д штампач и на њему жели одштампати некоме потребан модел, а за узврат добити одређену новчану накнаду.



Слика 5. Табела корисници

5.1. PHP

PHP – (PHP: Hypertext Preprocessor)

PHP је специјализовани скриптни програмски језик који се користи за израду динамичких веб страница, тј. за динамичко генерисање HTML кôда.

Помоћу PHP-а може се креирати HTML страница на серверу пре њеног слања на локални рачунар клијенту попуњену динамичким садржајем**.** На овај начин PHP кôд који је генерисао страну се не може видети и приказује се чисти HTML кôд.

По синтакси сличан је програмском језику C, чак има и исте функције. То значи да једну радњу можете извести коришћењем више различитих функција.

PHP је настао 1995. године. Направио га је Rasmus Lerdorf из PHP/FI. Користио је Perl скрипте на својим web страницама. Он је тај програм назвао "Алат за личну презентацију". Касније су додате функције из програмског језика C за комуникацију с базама података и постављања на сервер динамичких web страница.

Свака PHP датотека садржи знакове мање и веће са упитницима на почетку и на крају <?php , ?> између којих се налази PHP код. Поред тога, свака променљива има префикс $. У називу променљиве нема размака. Као размак може се користити знак "_" (подвучено). Још једна врло битна ствар код променљивих у PHP-у је да су имена case-sensitive. То значи да програм разликује велика и мала слова, а ево једног малог примера за илустрацију:

$ime_promenjive није исто што и $Ime_promenjive

Include ("naziv_datoteke") – ова функција омогућава уметање садржаја друге .PHP датотеке у тренутно активну .PHP датотеку.

Постоје две основне наредбе за исписивање текста и оне су: **echo** и **print**.

5.2. HTML

HTML је скраћеница од Hyper Text Markup Language, што се може превести као „језик за означавање хипер текстова“. Хипер текстови су текстови који поред речи садрже и слику, видео и аудио записе. HTML се користи за дефинисање изгледа World Wide Web докумената (страница) као и за успостављање веза (линкова) међу документима (документ садржи текст, слику, звук, графику).

HTML користи тагове да помоћу њих укаже претраживачу како неки текст или слика треба да буду приказани (исписани) на екрану. Тагови се стављају унутар угластих заграда <tag>. У већини случајева, тагови се постављају на почетак неког дела документа, а на крају тог документа поставља се таг завршетка </tag>.

Основни HTML тагови:

<html>, </html> – између ова два тага налазиће се сви остали тагови као и сам садржај документа.
<head>, </head> – служи да се унутар њега дефинишу неке карактеристике документа, као што је наслов. Све што се налази унутар овог тага неће се исписивати на екрану.
<title>,</title> – служи за дефинисање назива документа.
<body>,</body> – ови тагови дефинишу почетак и крај документа.
<h1>,</h1> – h таг се користи за избор величине слова, највећа слова су h1 а најмања h6.
, – **болдирано**
, – *италик*
, – користи се за центрирање
<hr> – Хоризонтална линија
<-- Komentar - -> – коментар који се неће видети у претраживачу.

HTML Команде за линковање:

Google pretraga

<a – означава почетак линка
href – означава “hypertext reference”
http://www.google.rs/ – ово је URL документа на који се повезујемо
Google pretraga – текст на који се појављује
/a> – крај линка

5.3. e-Commerce

e-Commerce (Electronic Commerce EC – електронска трговина) је куповина или продаја добара или услуга путем Интернета, нарочито путем сервиса World Wide Web. У пракси се овај термин често користи уместо новијег термина e-business, што значи пословање путем Интернета.

Термин e-commerce може се дефинисати и као процес управљања online финансијским трансакцијама од стране појединаца или компанија. Овај процес укључује како малопродајне, тако и велепродајне трансакције. Термин Online је синоним за Интернет (бити на линији = бити на Интернету).

Фокус e-Commerce-a је у системима и процедурама помоћу којих долази до размене различитих финансијских докумената и информација. Ови системи укључују трансакције кредитним картицама, e-cash (електронска готовина), e-billing (електронско плаћање), e-cheques (електронски чекови), electronic invoices (електронски рачуни), наруџбине и финансијске изјаве. e-Commerce практично омогућује наставак коришћења технологије електронских размена података (EDI).

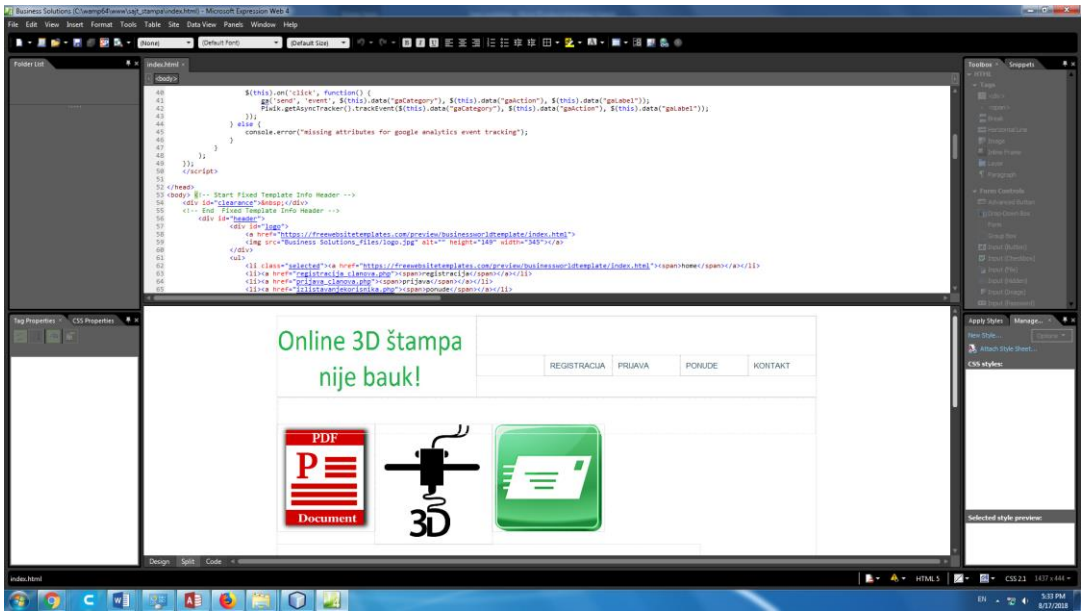
Electronic Data Interchange (EDI) је размена пословних података где се користи разумљиви формат ових података. Овај систем размене података је претходио појави Интернета и обично се користио у размени података између корисника који су међусобно већ били у контакту (систем 1 на 1).

На *Слици 8.* приказан је изглед e-commerce сајта који се базира на 3D штампи путем интернета.
Template је преузет са линка [7]

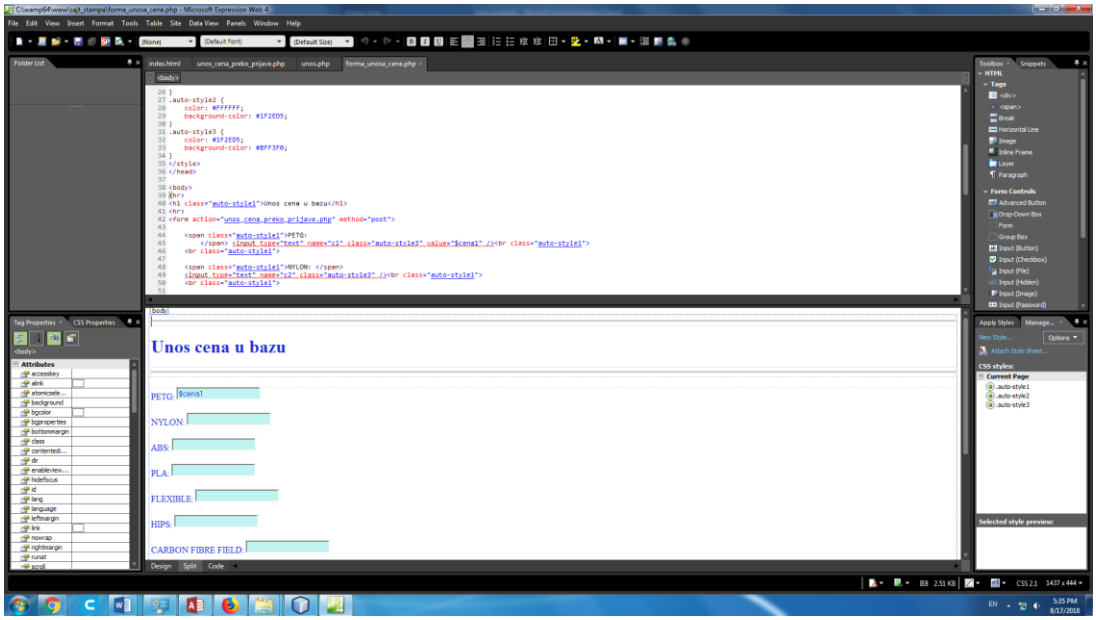
Слика 8. e-Commerce сајт

5.4. Microsoft Expression Web

Овај програм је намењен корисницима, како почетницима, тако и мало напреднијим програмерима. Савршен је за учење основних ствари у HTML-у због свог једноставног дизајна и сликовито описаних функција које се могу убацити у PHP код једноставним превлачењем, након чега ће се у другом екрану појавити читав HTML код који би морао у супротном да се испише за те функције које смо добили једноставним превлачењем.



Слика 9. Изглед радног окружења програма Microsoft Expression Web



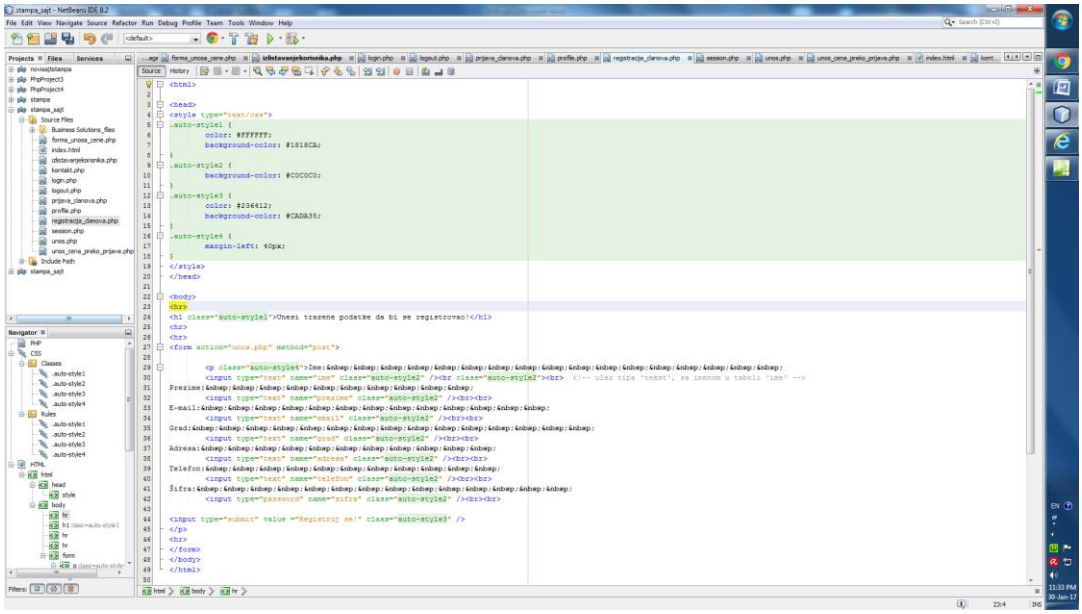
Слика 10. Измена боје поља, боје текста и убацивање хоризонталних линија

5.5. NetBeans IDE

Овај програм омогућава креирање различитих фајлова, како PHP-а тако и HTML-а и разних других, једноставно куцање, разне помоћне функције, као и библиотеке.

Омогућава бржи рад тако што у току куцања избацује могуће речи односно команде и објашњење шта та команда ради. Целу лабораторију је могуће видети притиском на RCTRL и SPACE, након чега ће се појавити падајући мени у коме се налази мноштво команди.

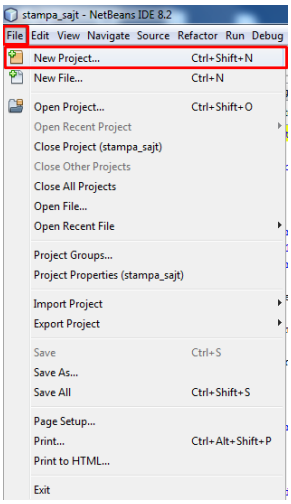
Помоћу овог програма (*Слика 11.*) је могуће повезивање са програмом wampserver и да погледавање како ће рад изгледати у web прегледачу.



Слика 11. Изглед радног окружења програма NetBeans IDE

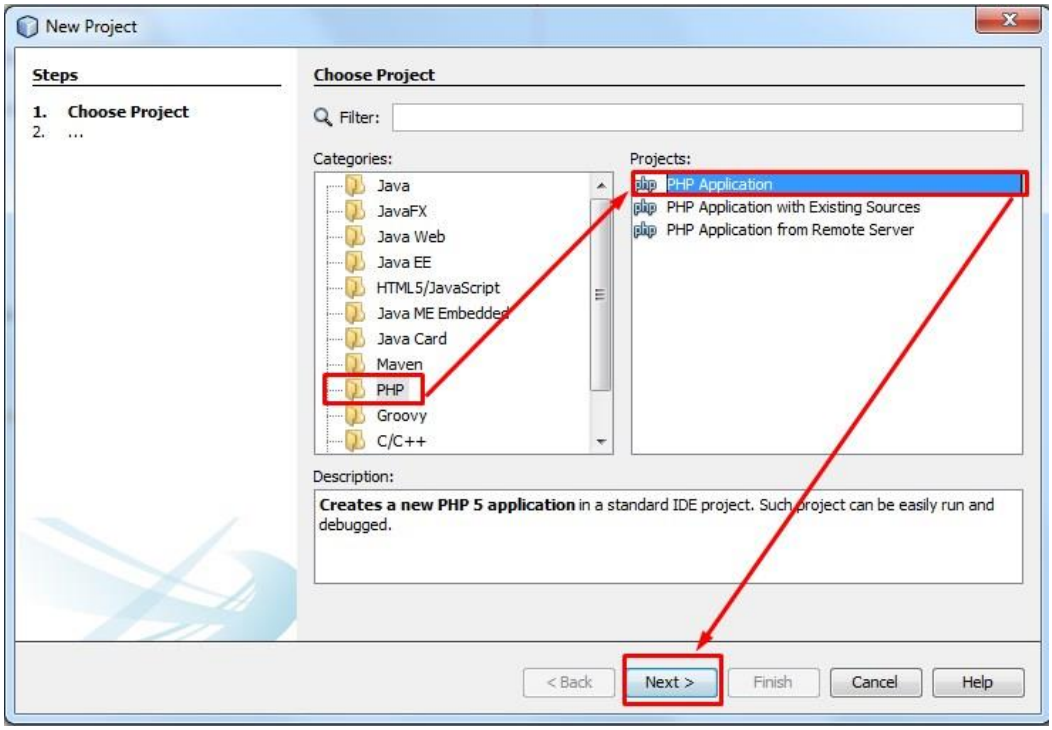
Праћењем следећих корака биће направљен нови Пројекат у програму са називом: stampa_sajt.

Најпре ће бити креиран нови пројекат (*Слика 12*).



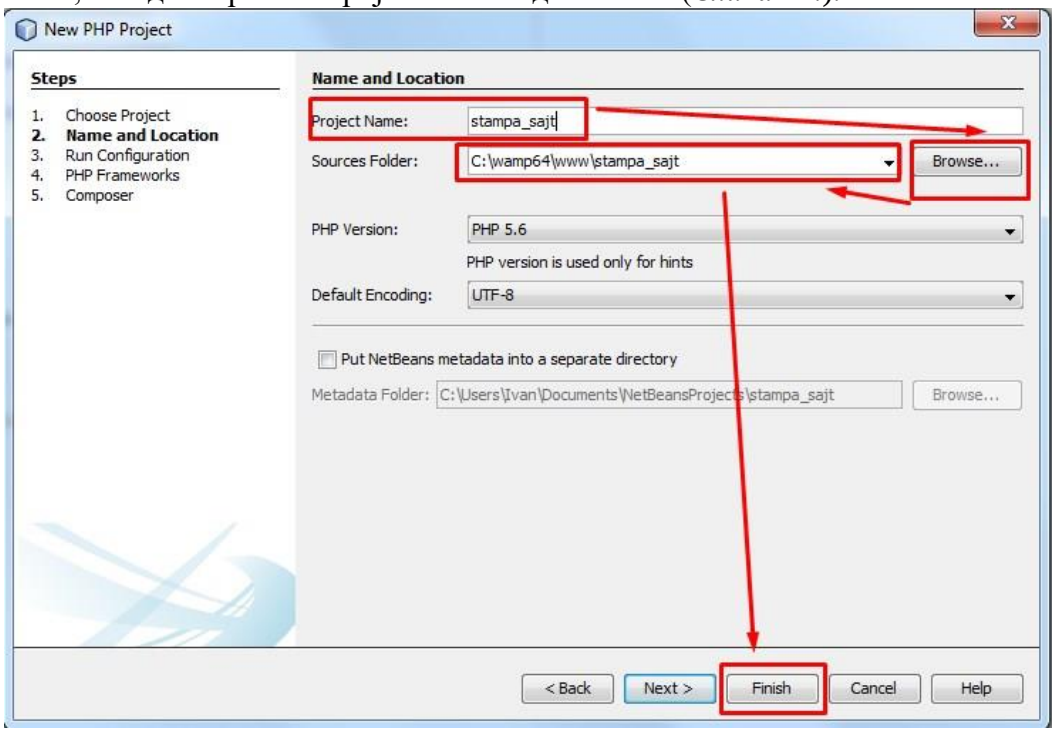
Слика 12. Нови пројекат

Затим оће бити одабрано PHP па PHP Application и онда је потребно прећи на следећу страну командом Next> (*Слика 13.*).



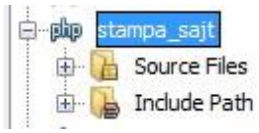
Слика 13. Креирање PHP Application

Одабиром жељеног имена пројекта, ксо пример биће назван: stampa_sajt (обавезно користити доњу црту уместо размака). Потребно је сместити тај пројекат командом Browse... у фолдер у рачунару где је инсталиран програм и обавезно као крајњи фолдер одабрати фолдер са називом "www", и онда завршити пројекат командом Finish (*Слика 14.*).



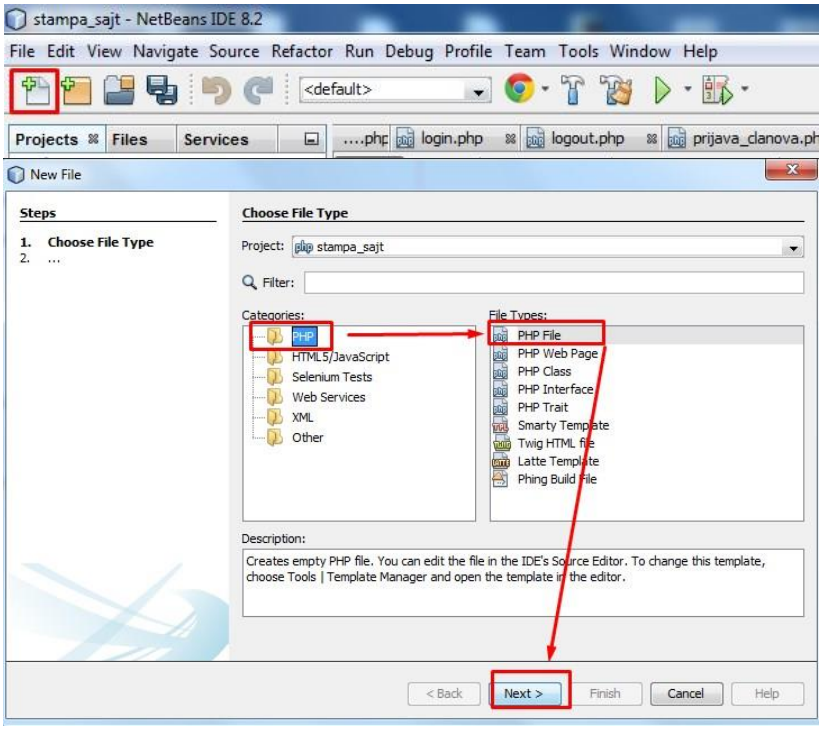
Слика 14.

У левом делу програма ће се приказати (*Слика 15*).



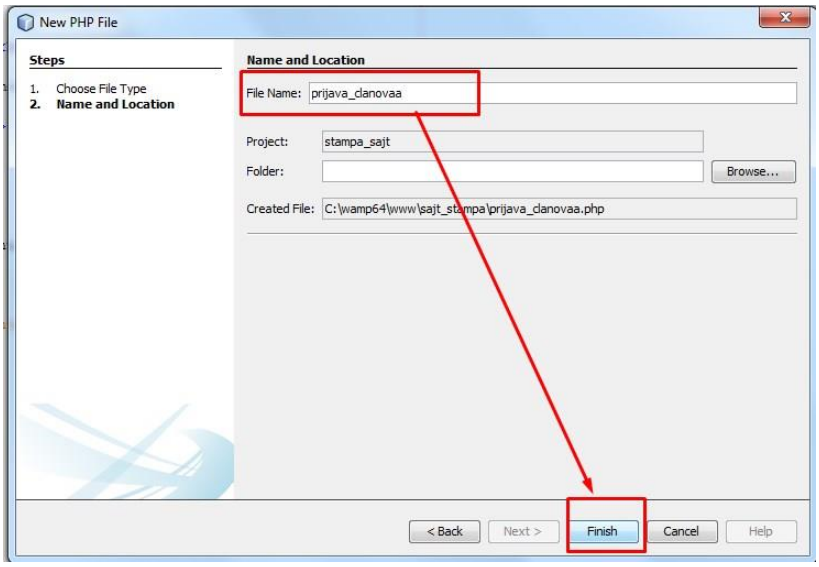
Слика 15.

Потребно је креирати нови PHP фајл у коме ће се касније исписивати команде (*Слика 16.*).



Слика 16.

Потребно је уписати назив фајла, нпр: prijava_clanova и завршити са прављењем истог командом Finish (*Слика 17.*).



Слика 17.

6. Убацивање креиране базе података у Wampserver

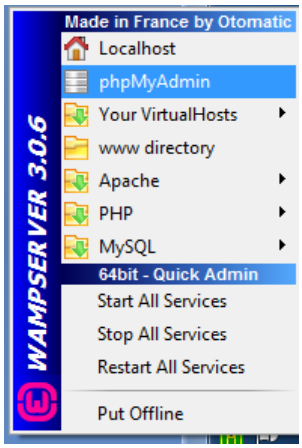
WampServer (*Слика 18.*) је пакет програма за рачунаре са *Windows* оперативним системом. У пакету се налази *Apache* server, MYSQL i PHP. *Wamp* је локални сервер сличан оном који се користи на хостинг рачунарима, у којима се могу инсталирати *wordpress, joomla!, Drupal, Woocommerce*, или било који други CMS и могуће је тестирање, учење и дизајнирање. Локални сервер је изванредан јер се може користити као место за тестирање *plugin*-ова, тема и модификација.

phpMyAdmin (*Слика 19.*) јесте у ствари део где се може креирати база података, или убацити већ постојећа која је направљена раније као у овом случају, уносити нове податке, брисати постојеће, вршити разне измене вредности и назива, поставити да нека вредност буде јединствена као што је на пример JMBG или ID.

Потребно је узети у обзир да је за сајт потребна база података у коју ће се уносити подаци о корисницима, цене за одређене производе (у овом случају за 3D штампу на 3D штампачима са разним димензијама, у разним бојама и са различитим материјалима),потребно је креирати базу података и направити одговарајућу табелу односно убацити већ направљену базу.

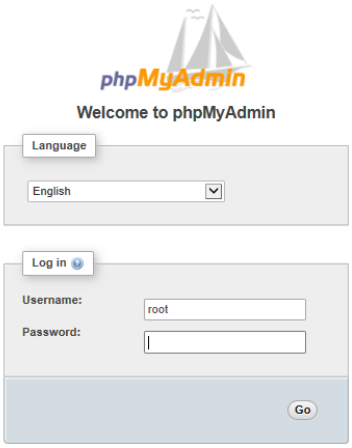
Левим кликом на икону која се налази у десном углу taskbar-а поред сата (*Слика 18.*) отвара се падајући мени као на коме ћемо одабрати phpMyAdmin као на (*Слика 19.*).

Слика18.

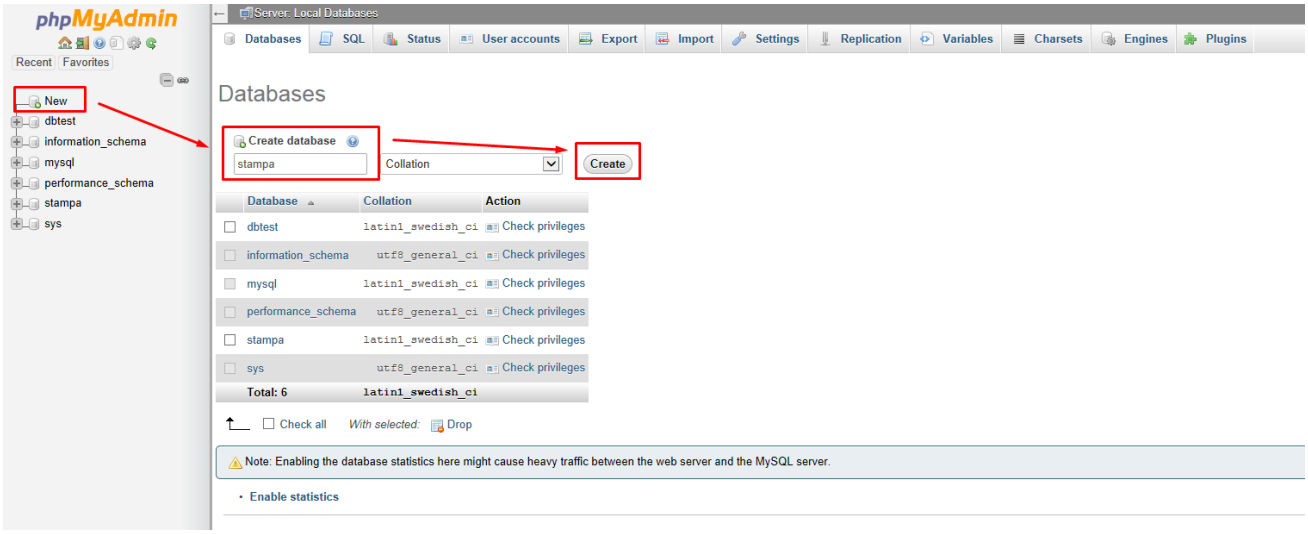


Слика 19.

Када је то обављено, отвориће се (*Слика 20.*) у претраживачу где је потребно уношење као корисничко име (Username) : root, а поље за шифру остаје празно и кликом на дугме ‘’Go’’ отвориће се идентично као на *Слици 21.*

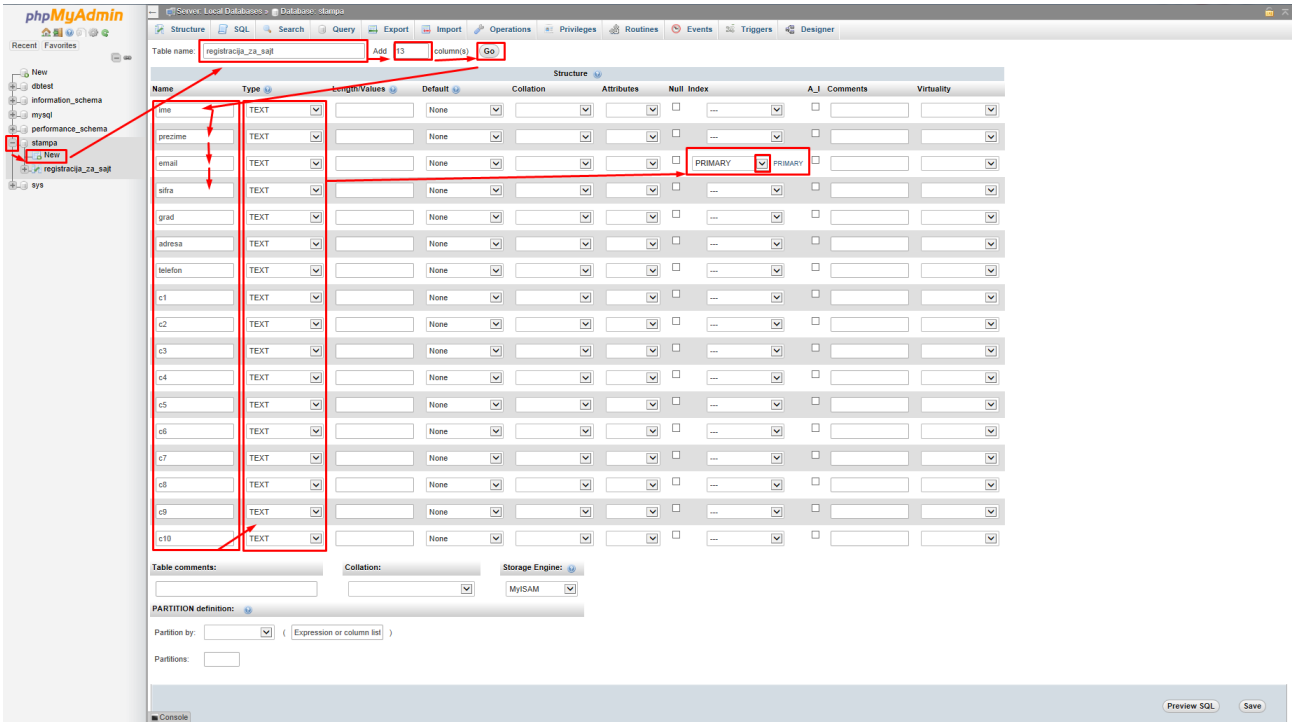


Слика20.



Слика 21.

Када је направљена нова база података са називом stampa потребно је убацити базу података коју смо раније направили у Access-u (*Слика 22.*).



Слика 22.

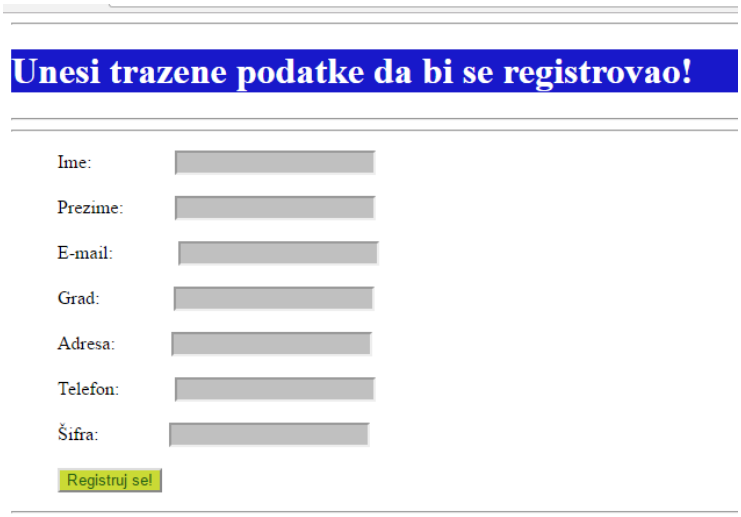
7. Потребни PHP фајлови

Након направљене базе података креирати PHP фајл чија је функција регистрација на сајт и самим тим уношење података у базу, са називом **unos.php**, и у њега је потребно унети следећи кôд:

```
<html>
<head>
<style type="text/css">
.auto-style1 {
	text-align: center;
}
</style>
</head>
<body>

<?php

$con = new mysqli("localhost","root","","stampa"); //konekcija na bazu podataka

if (!$con)
  {
  die('Nije moguca registracija, pokusajte kasnije: ' . mysql_error());
  }
$sql="INSERT INTO korisnici (ime, prezime, email, sifra, grad, adresa, telefon, PETG, NYLON, ABS, PLA, FLEXIBLE, HIPS, CARBONFIBREFIELD, ASA, POLYCARBONATE, WOODFIELD, PVA)
// INSERT INTO ime tabele(naziv, kolone, u, kojima, ce, redom, ispisivati, podatke, koje, smo, mu, zadali, u, daljem, kodu)

VALUES   // ispisivace sledece vrednosti

('$_POST[ime]','$_POST[prezime]','$_POST[email]','$_POST[sifra]','$_POST[grad]','$_POST[adresa]','$_POST[telefon]','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')";

if ($con->query($sql) === TRUE) {
   echo "Registracija uspesna!";
}
else {
   echo "Error: " . $sql . "<br>" . $con->error;
}
$con->close();

?>
<div id="profile" class="auto-style1">

   <b id="body"><a href="index.html">Vrati se na pocetnu stranu!</a></b>

</div>
</body>

</html>
```

Следећи корак је креирање новог PHP фајла са називом **registracija_clanova.php** у који је потребно уписати HTML код са текстуалним пољима из којих ће се узимати подаци и слати у unos.php.

```
<html>



<head>
<style type="text/css">
.auto-style1 {
	color: #FFFFFF;
	background-color: #1818CA;
}
.auto-style2 {
	background-color: #C0C0C0;
}
.auto-style3 {
	color: #236412;
	background-color: #CADA35;
}
.auto-style4 {
	margin-left: 40px;
}
</style>
</head>

<body>
<hr>
<h1 class="auto-style1">Unesi trazene podatke da bi se registrovao!</h1>
<hr>
<hr>
<form action="unos.php" method="post">

	<p class="auto-style4">
Ime:  <input type="text" name="ime" class="auto-style2" /><br class="auto-style2"><br>
			<!-- ulaz tipa 'tekst', sa imenom u tabeli 'ime' -->

Prezime: <input type="text" name="prezime" class="auto-style2" /><br><br>
E-mail: <input type="text" name="email" class="auto-style2" /><br><br>
Grad: <input type="text" name="grad" class="auto-style2" /><br><br>
Adresa: <input type="text" name="adresa" class="auto-style2" /><br><br>
Telefon: <input type="text" name="telefon" class="auto-style2" /><br><br>
Šifra: <input type="password" name="sifra" class="auto-style2" /><br><br>

<input type="submit" value ="Registruj se!" class="auto-style3" />     <!-- dugme -->
</p>
<hr>
</form>
</body>
</html>
```



Слика 23. Изглед у претраживачу

Затим је потребно креирати нове фајлове који ће нам бити потребни за пријављивање и сесију са називима:

-profile.php
-session.php

profile.php

```
<?php
include('session.php'); //obuhvatiti sesiju
?>
<!DOCTYPE html>
<html>
<head>
<title>Pocetna strana</title>
<style type="text/css">
.auto-style1 {
	text-align: center;
}
.auto-style2 {
	background-color: #25FF00;
}
.auto-style3 {
	color: #D51F1F;
	background-color: #FF004A;
}
.auto-style4 {
	background-color: #F5FF00;
}
.auto-style5 {
	color: #D51F1F;
	background-color: #F5FF00;
}
</style>
</head>
<body>
<div id="profile" class="auto-style1">
<b id="welcome">Dobrodošao : <i><?php echo $login_session; ?></i></b> <!-- ispisivanje
Dobrodosao:, i ubacivanje emaila ulogovanog korisnika iz sesije -->
	<br><br>
<b id="body"><a href="forma_unosa_cene.php"><span class="auto-style2">Klikni ovde za unos cena
u bazu!</span></a></b>
	<span class="auto-style2"> <!-- hiperlink koji salje na formu za unosenje cena u bazu podataka -
->
	</span><br><br>
<b id="logout"> <a href="logout.php"><span class="auto-style5">Odjavi se!</span></a></b>
	<span class="auto-style3"><span class="auto-style4"> <!-- hiperlink za odjavljivanje sa stranice
-->
	</span></span>
</div>
</body>
</html>
```

session.php

```
<?php
// konektovanje sa bazom
$connection = @mysql_connect('localhost', 'root', '');
// selektovanje baze podataka
$db = mysql_select_db("stampa", $connection);
session_start();// pocetak konekcije
// skladistenje sesije
$user_check=$_SESSION['login_user'];
// SQL upit za potpune informacije o korisniku
$ses_sql=mysql_query("select email from korisnici where email='$user_check'", $connection);
$row = mysql_fetch_assoc($ses_sql);
$login_session =$row['email'];
if(!isset($login_session)){
mysql_close($connection); // zatvaranje konekcije
header('Location: prijava_clanova.php'); // Preusmeravanje na pocetnu stranicu.
}
?>
```

Након креирања сесије потребно је направити PHP фајлове за пријаву, одјаву и форму пријаве. То ће бити одрађено на следећи начин:
Креирањем најпре logout.php и уписивањем следећег кода:

logout.php

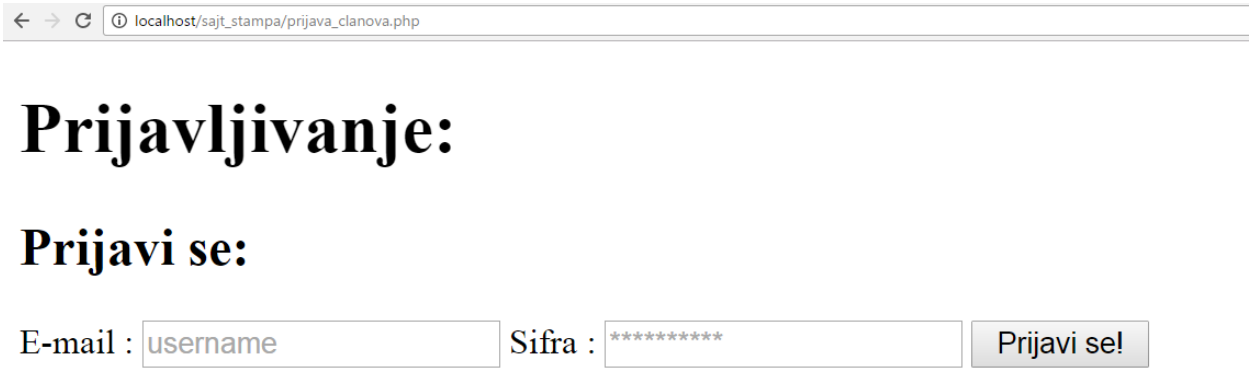
```
<?php
session_start();
if(session_destroy()) // zatvaranje, odnosno unistavanje svih sesija
{
header("Location: prijava_clanova.php"); // vracanje na stranu za prijavljivanje korisnika.
}
?>
```

login.php:

```
<?php
session_start(); // Startovanje sesije
$error=''; // Promenjiva koja salje podatke o gresci
if (isset($_POST['prijava'])) {  //klikom na dugme submit
if (empty($_POST['email']) || empty($_POST['sifra'])) {       //ako ne otkucate email ili sifru ili je neki
podatak netacan ispisace se sledece:
$error = "Email ili sifra nisu ispravni!";
}
else
{
        // Definisanje email-a i sifre
$username=$_POST['email'];
$password=$_POST['sifra'];
        // Uspostavljanje veze sa serverom koriscenjem imena servera, korisnickog id-a i sifre.
$connection = mysql_connect("localhost", "root", "");
        // Zastita Mysql ubacivanja za potrebe bezbednosti
$username = stripslashes($username);
$password = stripslashes($password);
$username = mysql_real_escape_string($username);
$password = mysql_real_escape_string($password);
        // Selektovanje baze podataka
$db = mysql_select_db("stampa", $connection);
        // SQL upit koji donosi informacije o registrovanim korisnicima
$query = mysql_query("select * from korisnici where sifra='$password' AND email='$username'",
$connection);
        // korisnika stavlja u promenjivu $rows
$rows = mysql_num_rows($query);
        //ispituje da li je korisnik postojeci.
if ($rows == 1) {
         //ako jeste stavlja u sesiju njegov email
$_SESSION['login_user']=$username; // Pokretanje sesije
header("location: profile.php"); // Preusmeravanje na drugu stranicu
} else { // u suprotnom ispisuje gresku.
$error = "email i sifra su neispravni!";
}
mysql_close($connection); // zatvaranje konekcije
}
}
?>
```

Креирати нови фајл са називом **prijava_clanova.php** и обухватити фајл login.php на следећи начин:

```
<?php
include('login.php'); // Obuhvatiti login.php

if(isset($_SESSION['login_user'])){
header("location: profile.php");
}
?>
<!DOCTYPE html>
<html>
<head>
<title>Forma Prijavljivanja u PHP-u preko sesije</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="main">
<h1>Prijavljivanje:</h1>
<div id="login">
<h2>Prijavi se:</h2>
<form action="" method="post">
<label>E-mail :</label>
<input id="name" name="email" placeholder="username" type="text">
<label>Sifra :</label>
<input id="password" name="sifra" placeholder="**********" type="password">
<input name="prijava" type="submit" value=" Prijavi se! ">
<span><?php echo $error; ?></span>
</form>
</div>
</div>
</body>
</html>
```



Слика 24: Изглед кôда за пријаву корисника у Интернет прегледачу.

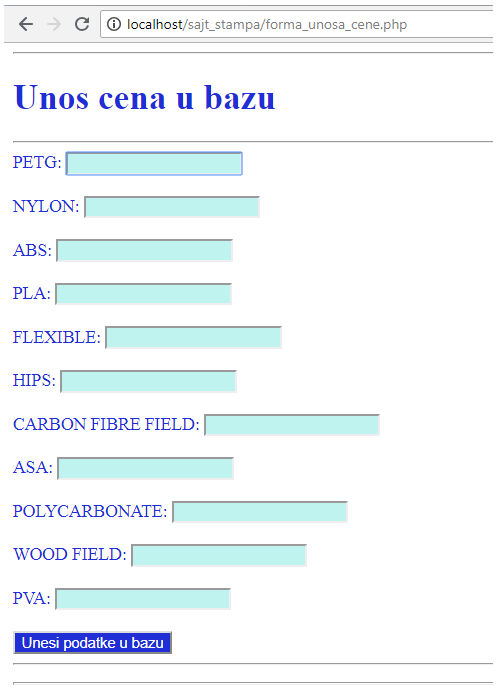
За сада је могуће извршити регистрацију на сајт и пријавити се на исти, тако да ће се е-маил адреса наћи запамћена у сесији, а затим је потребно направити унос цена у базу података преко сесије креирањем прво **unos_cena_preko_prijave.php** и уписивањем следећег кôда:

```
<?php
include('session.php'); //obuhvatiti sesiju
?>

<html>

<body>

<?php

$con = new mysqli("localhost","root","","stampa");

if (!$con)

  {

  die('Nije moguce unosenje u bazu: ' . mysql_error());

  }

$sql= "UPDATE materijal3Dcena
   SET   PETG = '$_POST[PETG]', NYLON = '$_POST[NYLON]', ABS = '$_POST[ABS]', PLA = '$_POST[PLA]', FLEXIBLE = '$_POST[FLEXIBLE]', HIPS = '$_POST[HIPS]', CARBON FIBRE FIELD = '$_POST[CARBON FIBRE FIELD]', ASA = '$_POST[ASA]', POLYCARBONATE = '$_POST[POLYCARBONATE]', WOODFIELD = '$_POST[WOODFIELD]', PVA= '$_POST[PVA]'
       WHERE email='$login_session'"
       ;


if ($con->query($sql) === TRUE) {
   echo "Izmena podataka uspesna!";
} else {
   echo "Error: " . $sql . "<br>" . $con->error;
}

$con->close();

?>
<div id="profile">

   <b id="body"><a href="index.html">Vrati se na pocetnu stranu!</a></b>

</div>
</body>

</html>
```

Креирати HTML форму са пољима за унос података који ће исте слати на адресу: unos_cena_preko_prijave.php а назваћемо га: **forma_unosa_cene.php**

```
<html>

<head>
<style type="text/css">
.auto-style1 {
	color: #1F2ED5;
}
.auto-style2 {
	color: #FFFFFF;
	background-color: #1F2ED5;
}
.auto-style3 {
	color: #1F2ED5;
	background-color: #BFF3F0;
}
</style>
</head>

<body>
<hr>
<h1 class="auto-style1">Unos cena u bazu</h1>
<hr>
<form action="unos_cena_preko_prijave.php" method="post">

	<span class="auto-style1">PETG:
      </span> <input type="text" name=" PETG " class="auto-style3"  /><br class="auto-style1">
	<br class="auto-style1">

	<span class="auto-style1"> NYLON: </span>
	<input type="text" name=" NYLON " class="auto-style3" /><br class="auto-style1">
	<br class="auto-style1">

	<span class="auto-style1"> ABS: </span>
	<input type="text" name=" ABS " class="auto-style3" /><br class="auto-style1">
	<br class="auto-style1">

	<span class="auto-style1"> PLA: </span>
	<input type="text" name=" PLA " class="auto-style3" /><br class="auto-style1">
	<br class="auto-style1">

	<span class="auto-style1"> FLEXIBLE: </span>
	<input type="text" name=" FLEXIBLE " class="auto-style3" /><br class="auto-style1">
	<br class="auto-style1">

	<span class="auto-style1"> HIPS: </span>
	<input type="text" name=" HIPS " class="auto-style3" /><br class="auto-style1">
	<br class="auto-style1">

	<span class="auto-style1"> CARBON FIBRE FIELD: </span>
```

```
        <input type="text" name=" CARBON FIBRE FIELD " class="auto-style3" /><br class="auto-style1">
        <br class="auto-style1">

        <span class="auto-style1"> ASA: </span>
        <input type="text" name=" ASA " class="auto-style3" /><br class="auto-style1">
        <br class="auto-style1">

        <span class="auto-style1"> POLYCARBONATE: </span>
        <input type="text" name=" POLYCARBONATE " class="auto-style3" /><br class="auto-style1">
        <br class="auto-style1">

        <span class="auto-style1"> WOODFIELD: </span>
        <input type="text" name=" WOODFIELD " class="auto-style3" /><br class="auto-style1">
        <br class="auto-style1">

        <span class="auto-style1"> PVA: </span>
        <input type="text" name=" PVA " class="auto-style3" /><br class="auto-style1">
        <br class="auto-style1">


<input type="submit" value ="Unesi podatke u bazu" class="auto-style2" />
<hr>
</form>

<hr>

</body>
</html>
```



Слика 25. Изгед уноса цена у претраживачу

Табеле су веома важне за креирање уређеног Веб интерфејса. Како би се могли видети сви корисници који су се регистровали на сајт, потребно је извршити читање свих података из базе података, изоставивши неке податке (у овом случају је то само шифра) и уписати их у табелу. Ћелија може садржати било који елемент, нпр. текст, слике, линкове, листе, формуларе итд.

Направити фајл **izlistavanjekorisnika.php** помоћу којег ће се очитавати подаци и исписивати у табели.

```
<?php
   // Nazovimo neku promenjivu link i dodelimo joj vrednosti konekcije.
   $link = mysqli_connect("localhost", "root", "", "stampa");
   // provera konekcije
   if($link === false){
      die("GRESKA: Nije moguca konekcija. " . mysqli_connect_error());
   }
   // Pokusaj da se selektuju svi podaci iz baze
   $sql = "SELECT * FROM korisnici";
   // ispitujemo da li je uspesno konektovan na bazu($link) i da li moze da selektuje podatke iz baze($sql) na koju se konektovao.
   if($result = mysqli_query($link, $sql)){
      // sada kazemo ako je rezultat veci od nule, da napravi tabelu i da ubaci vrednosti koje su trazene, tako sto ce u svaku kolonu da stavi na vrhu kolone ono sto se nalazi posle komande echo.
      if(mysqli_num_rows($result) > 0){
         echo "<table border>";
            echo "<tr>";  // <tr> oznacava pocetak kolona u tabeli i sa svakim novim redom odnosno oznakom ';' ce se praviti nova kolona.
               echo "<th>Ime:<vr></th>";     // <th>sve sto se ovde nalazi je boldirano</th>
               echo "<th>Prezime:</th>";
               echo "<th>E-mail:</th>";
               echo "<th>Grad:</th>";
               echo "<th>Adresa:</th>";
               echo "<th>Telefon:</th>";
               echo "<th>PETG:</th>";
               echo "<th>NYLON:</th>";
               echo "<th>ABS:</th>";
               echo "<th>PLA:</th>";
               echo "<th>FLEXIBLE:</th>";
               echo "<th>HIPS:</th>";
               echo "<th>CARBON FIBRE FIELD:</th>";
               echo "<th>ASA:</th>";
               echo "<th>POLYCARBONATE:</th>";
               echo "<th>WOODFIELD:</th>";
               echo "<th>PVA:</th>";
            echo "</tr>";
            // ako uhvatimo podatke stavicemo ih pod promenjivom ($row) i tako cemo ih vaditi iz tabele i u svaku kolonu redom izpisivati koliko god bude imalo podataka.
         while($row = mysqli_fetch_array($result)){
            echo "<tr>";
               echo "<td>" . $row['ime'] . "</td>";
               echo "<td>" . $row['prezime'] . "</td>";
               echo "<td>" . $row['email'] . "</td>";
               echo "<td>" . $row['grad'] . "</td>";
               echo "<td>" . $row['adresa'] . "</td>";
```

```
                echo "<td>" . $row['telefon'] . "</td>";
                echo "<td>" . $row['PETG'] . "</td>";
                echo "<td>" . $row['NYLON'] . "</td>";
                echo "<td>" . $row['ABS'] . "</td>";
                echo "<td>" . $row['PLA'] . "</td>";
                echo "<td>" . $row['FLEXIBLE'] . "</td>";
                echo "<td>" . $row['HIPS'] . "</td>";
                echo "<td>" . $row['CARBON FIBRE FIELD'] . "</td>";
                echo "<td>" . $row['ASA'] . "</td>";
                echo "<td>" . $row['POLYCARBONATE'] . "</td>";
                echo "<td>" . $row['WOODFIELD'] . "</td>";
                echo "<td>" . $row['PVA'] . "</td>";
            echo "</tr>";
        }     //ovde zavrsavamo nasu tabelu
        echo "</table>";
        mysqli_free_result($result);
    } else{     //ako nismo pronasli podatke ispisace se na ekranu:
        echo "Nije pronadjen sadrzaj";
    }
} else{   // u suprotnom ako nije moguca konekcija sa bazom podataka izpisace se:
    echo "Nije moguce izvrsiti: $sql. " . mysqli_error($link);
}
// Zatvaranje konekcije
mysqli_close($link);
?>
```

localhost/sajt_stampa/izlistavanjekorisnika.php

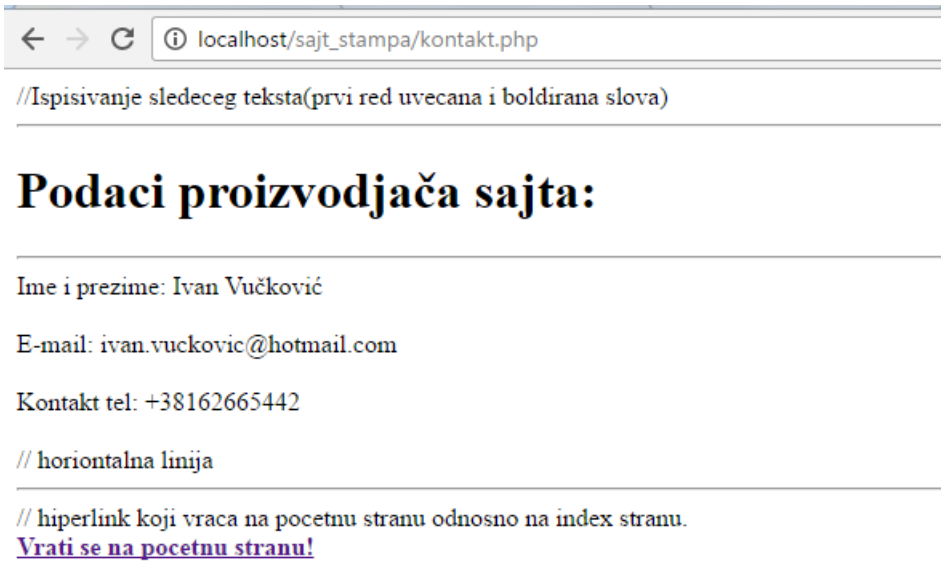
Ime:	Prezime:	E-mail:	Grad:	Adresa:	Telefon:	PETG:	NYLON:	ABS:	PLA:	FLEXIBLE:	HIPS:	CARBON FIBRE FIELD:	ASA:	POLYCARBONATE:	WOODFIELD:	PVA:
Ivan	Vučković	ivan.vuckovic@hotmail.com	Svilajnac	Dublje b.b. preko potoka	062665442	20	20	20	20	20	50	30	50	50	50	50
Ivan	ghdgh	dghdh	dhfghd	hdfgh	hhhhgf	0	0	0	0	0	0	0	0	0	0	0
Marija	Stojanovic	jojimjojim@gmail.com	Velika Plana	1.maj 9	06440	1	2	3	58	87	8	87	87	87	87	78
Marko	Zlatic	marko09k@gmail.com	Kragujevac	Bregalnicka 14	06633	20din	30din	130din	147.99din	2.54din	29.77din	30din	47din	78.99din	89.67din	0din
Bojana	Milosevic	bojanamilos	Jagodina	Milosevo bb	064	0	0	0	0	0	0	0	0	0	0	0
1	2	3	4	5	6	0	0	0	0	0	0	0	0	0	0	0

Слика 26: Изглед табеле из примера у Интернет прегледачу

Како би остао неки печат односно контакт на који је могуће контактирати админа, биће направљен следећи фајл са називом **kontakt.php** и укуцан следећи кôд:

```
<html>

<body>

   //Ispisivanje sledeceg teksta(prvi red sa najvecom velicinom slova)

<hr>

<h1>Podaci proizvodjača sajta:</h1>
```

```
<hr>
Ime i prezime: Ivan Vučković<br><br>
E-mail: ivan.vuckovic@hotmail.com<br><br>
Kontakt tel: +38162665442<br><br>
// horiontalna linija
<hr>
// hiperlink koji vraca na pocetnu stranu odnosno na index stranu.
<div id="profile">
    <b id="body"><a href="index.html">Vrati se na pocetnu stranu!</a></b>
</div>
</body>
</html>
```



Слика 27. Изглед кôда ‘’kontakt.php” у претраживачу

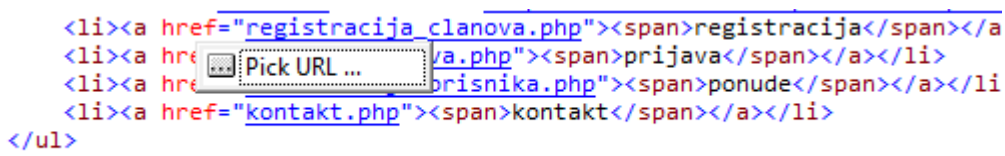
8. Template

Сачувани template за сајт је потребно дорадити и убацити све php фајлове у њега, затим је потребно ископирати цео фолдер template-а у www фолдер где нам се налазе и остали php фајлови. И цео измењен код копирати у нови фајл који ће бити назван : **index.html**

За измену кôда и убацивање хиперлинкова до наших php фајлова користићемо програм MicrosoftExpressionWeb4.

Као на *Слици 24*. клик на поље REGISTRACIJA одвешће нас до кôда који се налази у горњем делу програма, и веома једноставно можемо заменити назив дугмета и доделити му хиперлинк до кога ће одвести кликом на њега. Замена је веома једноставна – потребно је доћи курсором миша на хиперлинк кликнути леви клик, а затим командама на тастатури : LCTRL+SPACE појавиће се као Pick URL као на *Слици 25*. Дуплим кликом отвориће нам се мени у коме ћемо наћи php фајл до кога је потребно доћи кликом на дугме.

Слика 28.



Слика 29.

9. Закључак

У овом раду представљени су основни концепти приступу базама података преко позива функција и дат је пример израде базе за спајање на виртуелни хост. Посебна пажња посвећена је бази података Access. У првом делу рада дат је детаљан приказ карактеристика база података, са посебним освртом на Access базу података и њених делова, док је у другом и најзначајнијем делу дат пример праксе, односно принцип израде Access базе података за дељење на веб-у.

Као што се може закључити, база података је алат који служи за прикупљање и организовање најразличитијих информација, на пример о особама, производима, поруџбинама и томе слично. Добар пример је Access база података, која се састоји од табела, образаца, извештаја, упита, макроа и модула, који сви заједно чине да база података добро функционише. Да би добили добро креирану базу података, потребно је да се подаци организују тако да су лако доступни и да омогућавају лако одржавање саме базе. Када је једна база добро креирана, онда не треба много времена да се пронађу записи, подаци се смештају ефикасно и не долази до њиховог дуплирања, они се ажурирају врло једноставно и таква база је флексибилна, што даље значи да се могу лако укључити нове функције које би се од програма могле захтевати.

Како је циљ овог рада био да се бази података приступи преко позива функција, у раду је дато детаљно објашњење и подела функција. Наиме, скуп функција за комуникацију преко позива функција са базом података назива се SQL/CLI (SGL Call-Level Interface), а то је апликациони програмски интерфејс (API) за SQL приступ базама података из програмских језика. SQL/CLI даје библиотеку функција преко којих се клијент повезује са базом података и приступа подацима са SQL упитом. У раду су приказане основе овог стандарда, као и основни концепти и примери коришћења за ODBC, JDBC, ADO и ADO.NET.

10. Литература

[1] Лазаревић Б. …..: Базе података, ФОН Београд, Београд 2003.

[2] Павловић-Лажетић Г.: Основе релационих база података, Математички факултет, Београд,

[3] 2000. 3. James Wilson, Microsoft .NET серверска решења, ЦЕТ библиотека, Београд, 2013.

[4] Welling L., Thomson L.: PHP i MySQL развој апликација за веб, Микро књига, Београд, 2009.

[5] Jim Buyens: Развој база података на вебу: CET Computer Equipment and Trade Београд, 2001,

[6] Microsoft press: Microsoft Access 2000: CET Computer Equipment and Trade Београд, 1999,

[7] https://freewebsitetemplates.com/templates/page-3 јун 2018.

[8] https://www.formget.com/login-form-in-php/ јун 2018.

[9] http://www.w3schools.com/ мај 2018.

[10]http://registarfirmi.com/forum/e-poslovanje/odg-sta-je-e-commerce-a-sta-e-business/view-2.html јул 2018.

[11] http://www.draganmarkovic.net/weblog.php?file=sta-je-php.php фебруар 2018.

[12] http://www.vujke.com/web-design/html/ фебруар 2018.