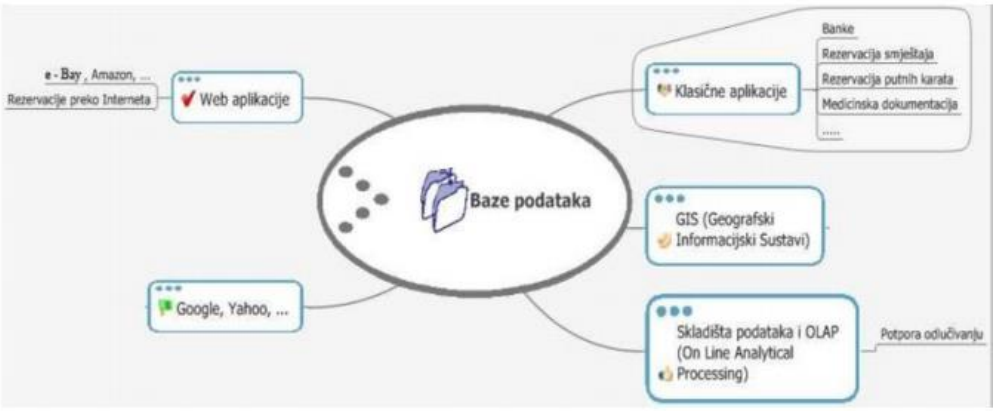
1. Историјски развој база података

Настанак база података се везује за Herman-a Holerith-a. Он је 1884. год пријавио патент – систем за аутоматску обраду података (АОП). АОП је служио за попис становништва у САД. Подаци су били на бушеним картицама који су се ручно убацивали у уређај за очитавање, а обрада података вршила се пребројавањем. Дотадашња обрада података пописа трајала је 10-ак година, а са овим изумом време обраде смањило се на шест недеља. Идеја АОП-а је била да се сваки становник САД преставња са низом од 80 карактера – име, годиште итд. Од Holerith -ове компаније настао је данашњи ИБМ. 1960'те Системи засновани на датотекама су имали доминантну улогу, али и поред тога први системи за управљањем базом података су уведени у овој деценији. Најпре су се користили само код великих пројеката, као што је био пројекат слетања Apollo-a на Месец. Такође први кораци ка стандардизацији предузети су у овој деценији , формирањем ДТБ Група (Data Base Task Group) . 1970'те Систем за управљањем базом података је постала комерцијална ствар, великим делом због потребе за системом који ће моћи да управља сложеним структурама података, као сто су рачуни фабрика при набавци сировина. Ови модели се сматрају првом генерацијом система за управљањем базом података. Многи од тих система су у употреби и дан данас, али имају неколико великих недостатака: • Тежак приступ подацима. За приступ и најједноставнијим подацима били су потребни изузетно сложени програми. • Веома ограничена независност података, тако да се програми нису могли изоловати од промена у формату података. Да би се превазишла ова ограничења Edgar.F.Codd је развио релациони модел. Овај модел одваја логички модел од физичког начина смештања података. 1980'те Релациони модел је доживео широку комерцијалну употребу у пословном свету. Са релационим моделом сви подаци су представљени у форми табеле. Релативно једноставан програмски језик четврте генерације, назван СQЛ, коришћен је за добијање информација. Овим моделом је обезбеђен једноставан приступ подацима и оним људима који нису били програмери. Овај модел је такође имао погодност за клијент/сервер обраду, паралелни пренос података и употребу графичког корисничког интерфејса (ГУИ). 1990'те Развој рачунарских мрежа и клијент/сервер обрада као и појава мултимедијалних података (графика, звук, слика и видео запис) постали су уобичајна ствар деведесетих. Самим тим што су подаци постали све сложенији морало је да се пронађе ново решење за системе база података. Тако су настали системи који су окренути ка објекту, и они се сматрају трећом генерацијом. 2000'те У овој деценији се јављају нови правци за развој система за базе података, као што су: • Могућност управљања све сложенијим типовима података. Ови типови укључују и мултидимензионалне податке, који су већ добили на важности у апликацијама складиштења података. • Децентрализоване базе података • Примена вештачке интелигенције ће олакшати приступ подацима и необученим корисницима • Развој нових техника и алгоритама за анализу података – анализа складишта података • Заштита података ИМДБ Систем база података у радној меморији (енг. in-memory database system, скраћено ИМДБ) је систем који целокупну базу података складишти у главну меморију рачунара. То се разликује од традиционалних система за управљање базама података који податке смештају на диск рачунара. Доња слика приказује разлику између традиционалних база података и база података смештеној у раној меморији. Будући да је рад са главном меморијом (РАМ) рачунара пуно бржи него писање и читање са диска, ИМДС омогућују пуно бржи рад самих апликација. Базе података у радној меморији убрзавају спремање информација, читање информација те сортирање информација тиме што све задржавају у главној меморији рачунара. Трансакције се изводе независно о тврдом диску чиме се отклања главни део времена обраде те се остварује добитак у ефикасност и брзини извршавања саме трансакције. Нагли развој Интернета, све веће брзине преноса података као и доступност више-језгрених процесора омогућавају и поспешују развој ИМДБ-а. Улога ИМДБ-а се увелико проширила током последњих неколико година због могућности адресирања већег броја РАМ-а (64 битних адресни простор) који је сада у могућности сместити базе података величине и до једног терабајта. Базе података у радној меморији своју примену све чешће налазе у складиштима података, специјалним справама великих брзина као што су медицински инструменти, мрежна и телекомуникациона опрема,

индустријски контролни уређаји, Web апликације... Т-Стабло Т-стабла су врста бинарних стабала које се користе у базама података у радној меморији. Т-стабло је структура података изведена као самобалансирајуће индексно стабло где су и индекси и сами подаци складиштени у примарној меморији рачунара. Ова стабла не чувају копије индексираних поља података унутар самог чвора индексног стабла. Уместо тога, користе чињеницу да су сами подаци увек у главној меморији као и индекс, тако да могу једноставно садржати показиваче на стварне податаке. Т-стабло садржи више елемената у самом чвору, а назив је доблио због облика чвора (слика доле).

2. Примјена база података



3. Типови база података

Хијерархијска стурктура

Хијерархијски тип заснива се на структури у којој су подаци уређени везом родитељ-дијете. Родитељ може да има више од једног дјетета, али дијете има само једног родитеља. Ова структура података има облик стабла. Састоји се од корјеног сегмента и њему подређених сегмената. Надређени сегменти могу имати више подређених сегмената, а подређени сегменти могу имати само један надређени. Нпр, представимо позиције ученика у клубу.(слику 3.) Групишемо чланове бејзбол клуба у основној школи за сваки разред. Ако желимо да представимо да су чланови Б и О обојица бацачи, такву везу не можемо додати у хијерархијскојструктури.

Мрежна структура

За разлику од хијерархијске базе података и хијерархијске структуре података, мрежна база података заснива се на мре#и података повезаних тако да не постоји ни корјени ни подређени сегменти. У основи је ово хијерархијска структура, међутим, не само да родитељ има више од једног дјетета, него и дијете има више од једног родитеља. С мрежним типом, вишеструке класе везе родитељ-дијете (сет) могу бити представљене. У условима када су везе између података једноставне и малобројне, мрежна и хијерархијска структура задовољавају потребе корисника.

Објектни модел

Објектни модел - инспирисан је објектно-оријентисаним програмским језицима. База је скуп трајно похрањених објеката који се састоје од својих интерних података и “метода” (операција) за руковање с тим подацима. Сваки објект припада некој класи. Између класа се успостављају везе насљеђивања, агрегације, односно међусобног кориштења операција.

Релациона база података

Представљање података дводимензионалним табелама представља релацијску структуру. Свака табела је еквивалентна и независна. У овој структури не постоје групе које се понављају и свака ћелија табеле представља један податак. Колоне имају свој назив, редови се разликују међу собом, а у једној колони постоји само једна

врста података. За хијерархијски и мрежни тип, сами подаци и веза између сваког податка представљене су линијама. Код релационе базе података, подаци се смјештају у табеле и повезују се релацијама.

4. **Описати модел ентитета и веза**

Бавимо се питањем: како обликовати шему за базу података, ускладену с правилима релацијског модела. У стварним ситуацијама доста је тешско директно погодити релацијску шему. Зато се служимо једном помоћном фазом која се зове моделирање ентитета и веза (Entity-Relationship Modelling). Ријечје о обликовању једне мање прецизне, концептуалне шеме, која представља апстракцију реалног свијета. Та тзв. ЕР-шема се даље, више-мање аутоматски, претвара у релацијску. Моделирање ентитета и веза захтијева да се свијет проматра преко три категорије: • ентитети: објекти или догадаји који су нам од интереса; • везе: односи међу ентитетима који су нам од интереса; • атрибути: својства ентитета и веза која су нам од интереса.

5. **Шта је ентитет**

Ентитет је нешто о чему желимо спремати податке, нешто што је у стању постојати или не постојати, те се може идентификирати. Ентитет може бити објект или биће (на примјер кућа, студент, ауто), односно догадај или појава (на примјер ногометна утакмица, празник, сервисирање аута). Ентитет је описан атрибутима (на примјер атрибути куће су: адреса, број катова, боја фасаде, . . .).

6. **Шта је атрибут**

Ентитет је описан атрибутима (на примјер атрибути куће су: адреса, број катова, боја фасаде, . . .). Уколико неки атрибут и сам захтијева своје атрибуте, тада га радије треба сматрати новим ентитетом (на примјер модел аута). Исто правило вриједи и ако атрибут може истовремено имати више вриједености (на примјер квар који је поправљен при сервисирању аута). Име ентитета, заједно са припадним атрибутима, заправо одредује тип ентитета. Може постојати много примјерака (појава) ентитета заданог типа (на примјер СТУДЕНТ је тип чији примјерци су Петровић Петар, Марковић Марко, . . .).

7. **Шта је примарни кључ**

Примарни кључ има двоструку улогу: једнозначно дефинише н-торке (ред у табели) и омогућава да се преко њега оствари веза са другим релацијама (табелама). Свака н-торка релације одређена је вриједношћу примарног кључа.

8. **Како одредити примарни кључ у табели**

Кандидат за кључ је атрибут, или скуп атрибута, чије вриједности једнозначно одредују примјерак ентитета заданог типа. Дакле, не могу постојати два различита примјерка ентитета истог типа с истим вриједностима кандидата за кључ. (На примјер за тип ентитета АУТО, кандидат за кључ је атрибут РЕГ БРОЈ). Уколико један тип ентитета има више кандидата за кључ, тада бирамо једног од њих и проглашавамо га примарним кључем. (На примјер примарни кључ за тип ентитета СТУДЕНТ могао би бити атрибут БРОЈ ИНДЕКСА.

9. **Шта је страни кључ**

Страни кључ (Foreign key) табеле је атрибут (скуп атрибута) који указује на зависност од неке друге табеле. Полазна табела се обично зове табела-дијете, а табела од које она зависи родитељска табела. Страни кључ мора задовољити услов да је скуп његових атрибута примарни кључ родитељске табеле.

10. **Степен везе између ентитета**

Везе се успостављају између два или више типова ентитета. Број ентитета у вези представља степен везе (Degree of Relationships). Пошто се у релационом моделу веза представља релацијом, може се рећи: Број домена на којима је дефинисана нека релација се назива степен релације. Слика 11. Приказ степена везе Слика 11. показује начин успостављања везе између два ентитета: Унарну (између двије инстанце једног типа ентитета, па се понекад назива рефлексивна), Бинарну (између инстанци два типа ентитета) и Тернарну (између инстанци три типа ентитета). Тернарна веза се остварује између два ентитета преко трећег (тако да се креира нова табела с примарним кључевима свих укључених ентитета), што ћемо касније разјаснити. Везе вишег степена су ријетке, али се темеље на истом принципу као тернарна веза

11. Дијаграм ентитета и веза

Већ смо дали једну репрезентацију представљања релација и веза. Пошто немамо много простора, само илустративно ћемо објаснити методу представљања помоћу дијаграма ентитета и веза познатих као E/P дијаграми (Ентиту Релатионсхип). Ово је једна од најприхваћенијих метода графичког представљања модела базе података, а увео ју је Петер Пин Шан Чен 1976. године и има неколико верзија и спецификација. E/P дијаграми се састоје од правоугаоника, који представљају ентитете, елипси, које представљају атрибуте и ромбова, који представљају везе између ентитета. На слици 10. можете видјети основне симболе који се користе за представљање E/P дијаграмом, као и један примјер модела базе представљен овом методом.

12. Кардиналност ентитета

Кардиналност (cardinality) представља однос броја објеката који се повезују. Кардиналност ентитета је уређени пар (а : б), који показује колики је број веза појединог елемента тог ентитета са елементима ентитета с којим је релацијски повезан. Слика 12. Типови кардиналности ентитета Постоје три типа кардиналности ентитета према броју веза: 1. Један-ка-један (1 : 1) Један примјерак првог типа ентитета може бити у вези с највише једним примјерком другог типа ентитета, те такође један примјерак другог типа може бити у вези с највише једним примјерком првог типа. 2. Један-ка-много (1 : н) Један примјерак првог типа ентитета може бити у вези с 0, 1 или више примјерака другог типа ентитета, но један примјерак другог типа може бити у вези с највише једним примјерког првог типа. 3. Много-ка-много (М : Н) Веза м:н значи да један примјерак првог типа ентитета може бити у вези с 0, 1 или више примјерака другог типа ентитета, те такође један примјерак другог типа може бити у вези с 0, 1 или више примјерака првог типа. На слици 12. дато је неколико илустрација које показују различите типове кардиналности и различите начине њиховог представљања. Веза м : н је веза (више према више) која се у моделима врло често јавља, а не може да се директно имплементира у релационом моделу базе.података. Проблем везе м:н између два ентитета се превазилази разбијањем ове везе на двије везе типа 1:н. Ова веза не може да се директно реализује у релационим базама података, већ мора посредно преко још једне табеле, као што је приказано на слици 12. Кардиналност је појам који означава бројност, па се говори и о кардиналности релације (ентитета), атрибута и везе, што може довести до забуне.

13. Особине релационог модела базе података

Релациони модел био је теоретски заснован још крајем 60-тих година 20. вијека, у радовима E.F. Codd -а. Модел се дуго појављивао само у академским расправама и књигама. Прве реализације на рачунару биле су сувише споре и неефикасне. Захваљујући интензивном истраживању, те напретку самих рачунара, ефикасност релационих база постепено се побољшавала. Средином 80-тих година 20. вијека релациони модел је постао превладавајући. И данас већина ДБМС користи тај модел. Релација, атрибут, н-торка, кључ Релациони модел захтијева да се база података састоји од скупа правоугаоних табела - тзв. релација. Свака релација има своје име по којем је разликујемо од осталих у истој бази. Једна колона релације обично садржи вриједност једног атрибута (за ентитет или везу) - зато колону поистовјећујемо с атрибутом и обратно. Атрибут има своје име по

којем га разликујемо од осталих у истој релацији. Вриједности једног атрибута су подаци истог типа. Дакле, дефинисан је скуп дозвољених вриједности за атрибут, који се зове домен атрибута. Вриједност атрибута мора бити једнострука и једноставна. Под неким условима толеришемо ситуацију да вриједност атрибута недостаје (није уписана). Један ред релације обично представља један примјерак ентитета, или биљежи везу између два или више примјерака. Ред називамо н-торка. У једној релацији не смију постојати двије једнаке н-торке. Број атрибута је степен релације, а број н-торки је кардиналност релације. Кључ К релације Р је подскуп атрибута од Р који има сљедећа "временски независна" својства: 1. Вриједности атрибута из К једнозначно одређују н-торку у Р. Дакле не могу у Р постојати двије нторке с истим вриједностима атрибута из К. 2. Ако избацимо из К било који атрибут, тада се нарушава својство 1. Будући да су све н-торке у Р медусобно различите, К увијек постоји. Наиме, скуп свих атрибута задовољава својство 1. Избацивањем сувишних атрибута доћи ћемо до подскупа који задовољава и својство 2.Дешава се да релација има више кандидата за кључ. Тада један он њих проглашавамо примарним кључем. Атрибути који састављају примарни кључ зову се примарни атрибути. Вриједност примарног атрибута не смије ни у једној н-торки остати неуписана. Грађу релације кратко описујемо тзв. шемом релације, која се састоји од имена релације и пописа имена атрибута у заградама. Примарни атрибути су подвучени.

14. Релациона алгебра

Релациона алгебра је алгебра са јасно дефинисаном логиком и семантиком, и за озбиљнији приступ потребно је далеко више простора него што ми имамо. Операција заједно са релацијама на које се примјењују образује релациони израз. Вриједност таквог израза је увијек релација. Сваки од алгебарских израза одговара упиту или претраживању. Релациони израз је израз облика РОп арг1 арг2 ... аргн гдје је са РОп представљен релациони оператор, а аргі су релације који су аргументи релационог оператора. Релациона алгебра представља скуп оператора чији су операнди и резултати релације. Чини је скуп од 8 операција које се називају основним (5 елементарних и 3 изведене). Према броју операнада операције се могу подијелити на: Унарне (1 операнд) и Бинарне (2 операнда). Преглед операција релационе алгебре дат је у табели 1. Операнд релационе алгебре је релација, а операнд операција са датотекама и класичним језицима је рекорд.

Сврха релационе алгебре код релационих база је двојака: 1. Писање релационих израза који се користе за • дефинисање простора за дохватање података, • дефинисње простора за ажурирање података, • дефинисање правила интегритета, • дефинисање изведених релација, • дефинисање правила заштите. 2. Основа за оптимизацију упита: Чак и једноставне дефиниције премашују зацртани оквир овог уџбеника, ра ћемо за већину операција дати само објашњења и графичке илустрације.

15. Операција селекције

Операција селекције (selection), позната и као операција ограничења или рестрикције (restriction), ознацава се са σ (сигма). Операцијом селекције се врши избор подскупа н-торки из релације и реализује се на основу услова селекције (избора). Услов селекције има улогу филтера, задржава само оне н-торке које испуњавају квалификацијски услов; док се остале н-торке одбијају (филтрирају). Услов селекције се састоји од чланова који су повезани операцијама (ознака оп): • and (^), • or (v), • not ($\neg$).

16. Опрерација пројекције

Операција пројекције (project) означава се са π или П (pi). Операцијом пројекције врши се вертикално партиционисање тако да се врши издвајање одређених колона (атрибута) из релације. Листа специфицираних колона (атрибута) се задржава у свакој нторки, а из релационе шеме се елиминишу постојећи атрибути, односно колоне релационе табеле, које из неког разлога више нису интересантне. Селекција Пројекција Резултат примјене пројекције је подскуп колона релације, уз уклањање дупликата. (Примјеном пројекције могуће је да

нова релација има више истих н-торки, па је потребно уклонити дупликате, јер математицки скупови не дозвољавају дуплиране елементе.)

17. Унија, пресјек и разлика

Унија, ознака U , елементарна је бинарна операција која из двије полазне релације формира нову, која садржи све н-торке из обје релације. Унија је могућа само у случају да полазне релације задовољавају услов унијске компатибилности који каже да: • Шеме релација имају исти број атрибута; • Атрибути релација респективно су исти по значењу и типу. И код уније се врши уклањање дупликата, ако постоје. Разлика, ознака -, (dffereace) елементарнаје бинарна операција која из двије полазне релације формира нову која садржи све н-торке прве релације које се не налазе у другој, а искључују се заједничке, што је приказано на слици 4.17. И разлика је могућа само између унијски компатибилних релација. Пресјек, ознака ∩, (intersect) елементарна је бинарна операција која из двије полазне релације формира нову која садржи све елементе прве који се налазе у другој (заједничке). И пресјек је могућ само измеду унијски компатибилних релација. Пресјек је изведена операција; уз мало пажње очигледно је да важи: R ∩ s =r- (r-s)

18. Нормализација базе података

Нормализација базе података је итеративни поступак којим се из модела базе података настоји отклонити вишеструко понављање истих података (редундација) и повећати стабилност базе. Редундантни записи, осим беспотребног заузимања простора, доводе и до проблема са промјенама садржаја, тј. до тзв. аномалија. Потпуно елиминисање редундације података у бази података је скоро немогуће остварити. Реални циљ при пројектовању базе података је контролисана редунданса података. При нормализацији је потребно да обезбиједимо реверзибилност, што значи да преуређивањем не смије доћи до губитка информација садржаних у полазној релацији и да на основу нормализованих релација, мора бити могућа реконструкција полазне ненормализоване релације.

19. Аномалије у бази података

Циљ нормализације је спречавање аномалија одржавања података. Под аномалијама одрзавања података подразумијевамо: • Аномалију додавања (инсерт) — јавља се у оним случајевима када су информације о атрибутима једног ентитета запамћене као дио неког другог ентитета; што има за посљедицу да је немогуће извршити додавање новог ентитета уколико не постоји информација о њему у неком другом ентитету. • Аномалију брисања (delete) — то је инверзија аномалије додавања; кад је немогуће избрисати један ентитет, уколико претходно не избришемо други. • Аномалију промјене (азурирања — update) — јавља се у случају када промјену података о једном ентитету треба извршити на више од једне копије података.

20. Нормалне форме

О нормализацији се обично говори у облику форми. Нормалне форме дефинишу начин како подаци треба да буду структурисани и уређени. Нормализација је према томе процес дефинисања структуре базе, односно релација које ју чине. Форме су хијерархијски поредане, тако да је прва најједноставнија. Дефинисано је шест нормалних форми. У пракси се сматра да је релациона шема (табела) нормализована, ако је у трећој нормалној форми. Стандардне нормалне форме су адитивне (види слику 4.26), дакле, ако је неки модел сведен на трећу нормалну форму, аутоматски је у другој и првој нормалној форми. 4.6.4.1. Прва нормална форма Релациона шема P налази се у 1НФ ако су сви некључни атрибути функцијски зависни о кључу од P. База података налази се у првој нормалној форми ако се свака релациона шема и њој налази у првој нормалној форми. Принцип 1НФ је да сваки атрибут торке може садржати само просту (атомску) вриједност. То значи да у табели треба уносити

појединачне вриједности, не групе или композитне објекте. За свођење релације на 1НФ, треба разлозити сваку неатомску вриједност.

Друга нормална форма Релација је у другој нормалној форми (2НФ), ако је у 1НФ и сви некључни атрибути релације у потпуности зависе од примарног кључа. Захтјеви 2НФ су: 1. Уклањање подскупова података који се налазе у више редова ј њихово смјештање у посебне табеле. 2. Креирање веза између нових табела и табела са којима су спојене коришћењем спољних кључева. То значи да се постојећа релација расчлани на нове релације (табеле) које зависе само од примарног кључа (у 2НФ није дозвољена парцијална зависност)

Трећа нормална форма Релација је у трећој нормалној форми, ако је у другој нормалној форми и ако су сви атрибути који нису дио ниједног кључа међусобно независни (ако сви њени некључни атрибути нетранзитивно зависе од примарног кључа). Основни захтјеви 3НФ: 1. Табела је у 1НФ и 2НФ. 2. Уклоњене су колоне које нису потпуно зависне од примарног кључа. Свођење на 3НФ се проводи уклањањем свих података у табелама који не зависе једино од примарног кључа. Практично, поставља се питање: Да ли постоје колоне које не зависе у потпуности од примарног кључа? Треба задржати само податке који су зависни од примарног кључа, а оне који нису треба премјестити у нове табеле и формирати примарни кључ за њих и везе између релација

21. Разлика између података и информације

Иако се речи податак и информација често користе као синоними, постоји разлика. Податакје чињеница која се чува у бази података. Информација се добија обрадом података. Примери података: 27 • назив књиге је Основи иорограмирања и прогамски језик C# • име аутора је Станка, • година рођења аутора је 1977, • врста повеза је тврд повез. Примери информација: • укупан број књига које је написао одређени аутор, • просечан број књига које одређена издавачка кућа изда годишње, • презимена свих аутора који су рођени пре 1950. године, • проценат издања са тврдим повезом у односу на укупан број издања. Примери који су дати су упрошћени и служе само као илустрација. Требало би бити обазрив приликом одлучивања о томе да ли је нешто податак или информација, зато што није ретко да нешто што је уједном контексту информација, у другом може да буде податак. На пример, укупан број књига које је написао одређени аутор је информација добијена из података о свим књигама које је написао тај аутор, али може да се посматра и као податак уколико га узимамо ради неке даље анализе и, на пример, поређења са бројем написаних књига других аутора. Информација, поред обраде, подразумева придруживање значења податку и његово тумачење у неком контексту.

22. Креирање табеле у Access-у

Приказати практично на рачунару.
Након покретања алата Access 2007, у горњем левом углу прозора налази се дугме Office Button, којеје присутно и у осталим апликацијама које су део пакета Microsoft Office2007. Кликом на ово дугме добија се падајући мени који садржи ставке за отварање постојеће, као и креирање нове БП. У неким верзијама се уместо овог дугмета налази падајући мени Fi1e са истим ставкама. Уколико желимо да креирамо нову БП, потребно је да изаберемо ставку New са менија. У десном делу екрана појављује се простор за креирање празне БП.

23. Постављање ограничења у табели – Access

Ограничења представљају скуп правила која морају да важе за податке у табели. Постоји више врста ограничења: примарни кључ, ограничењеје јединствености, обавезан податак, провера задатог правила, страни кључ.

Примарни кључ (енгл. Primary key - pk). - Вредност примарног кључа разликује се за сваки ред у табели. Табела може да има само један примарни кључ. Примарни кључ можеда будеједна колона или комбинација више њих. Ово ограничење подразумева и ограничење јединствености и обавезан податак, па се за колону/колоне које су

примарни кључ не морају посебно дефинисати ова два ограничења. Најчешће се уводи вештачки примарни кључ чије се вредности аутоматски попуњавају од стране СУБП, на пример:lD autora за табелу Autori, ID knjige за табелу Knjige, итд. Ово није неопходно и некада се користе и подаци других типова као примарни кључеви. На пример, у табели Autori примарни кључ би могао да буде ЈМБГ, који се чува као текстуални податак због операција које се над њим природно изводе и водећих нула које може да садржи. У том случају се вредности примарног кључа не би аутоматски попуњавале, већ би морале да се уносе као и други подаци. Осим примарног кључа који се састоји одједне колоне, табела може да има и сложени примарни кључ који се састоји од комбинације две или више колона. Ограничењејединствености (енгл. Uniquekey- uk). - Све вредности у колони над којомје дефинисано ово ограничење морају да буду различите. На пример, иако колона P1B није примарни кључ табеле lzdavaci, два издавача не могу да имају исту вредност броја ПИБ зато што је то јединствен број који се додељује сваком предузећу, и над том колоном мора да буде дефи- нисано ограничењејединствености. Требало би пажљиво да се размотре све могуће ситуације приликом одлуке о томе да ли би над неким пољем требало да се дефинише ограничење јединствености. На пример: иако се чини да су број телефона, адреса и имејл јединствени за сваког аутора, то не мора да буде случај уколико судвоје аутора супружници или чланови најуже породице који својом одлуком користе исти имејл, а живе заједно, па им је и фиксни контакт телефон исти. Две издавачке куће могу да имају канцеларије у истој пословној згради, па телефонске централе које обе користе и адреса могу да буду исти. Обавезан податак (енгл. Not nu11- nn). - Неки подаци морају да се унесу у БП, а неки не морају. На пример, обавезноје да се унесе назив издавачке куће, док не мора свака издавачка кућа да има факс, па тај податак мо- жемо да унесемо само за издавачке куће које имају број факса. Може да се деси и да издавачка куће има број факса, али да нам тај податак није доступан приликом уноса података о новој издавачкој кући, или није неопходно да тај број сачувамо у бази када већ имамо број телефона и мејл адресу издавача. Када у неком пољу нема податка, тада се у том пољу налази празан показивач - константа nu11. Уколикоје неки податак обавезан, онда у тој колони није дозвољен nu11. Провера задатог правила (енгл. Check constraint). - Често постоје пра- вила која подаци морају да задовољавају. На пример, уколико је Завод за уџбенике основан 1957. године, година издања било које књиге коју је издала ова издавачка кућа мора да буде та година или нека скорија. Страни кључ (енгл. Foreign key- fk). - Страни кључеви служе за повезивање табела. Сви подаци о књигама чувају се у табели Knjige. Поред осталих података (назив, број страна, цена итд.), за сваку књигује важно да знамо и која издавачка кућа ју је издала. Издавачке куће су важне и постоји потреба да се чува много података о њима. Из тог разлога постоји табела lzdavaci, у којој се чувају сви подаци о издавачима. Да би знали која изда- вачка кућа је издала коју књигу, потребно је да у табели књиге чувамо и кључ издавачке куће која ју је издала. Вредности кључа издавача у табели Knjige су вредности страног кључа. На тај начин, упоређивањем вредности кључа издавача, за сваку књигу можемо да дођемо до свих података везаних за њену издавачку кућу који се чувају у другој табели.

24. Типови података у Access-у

Типови података - *Access 2007*	
Тип	**Опис**
Attachment	*фајлови, као што* су, на пример, дигиталне фотографије
AutoNumber	бројеви који се аутоматски генеришу за сваки ред
Currency	валута, на пример, цена неке књиге или друге врсте производа
Date/Time	датум и време
Hyperlink	хиперлинкови, као што је, на пример, адреса електронске поште
Memo	дугачак текст са форматирањем. Често служи за опис. Може да буде, на пример, кратка биографија аутора у табели *Autori*.
Number	бројевне вредности, на пример, број страна књиге
OLE Object	*OLE* објекти, као што је, на пример, документ креиран у алату *Word*
Text	кратак текстуални податак који се састоји од слова и цифара, дужине до 255 знакова. Овај тип се много користи на пример, име, презиме, адреса, назив, ЈМБГ, тип повеза итд.
Yes/No	логичка вредност која може да буде тачно (ДА) или нетачно (НЕ).

25. Индексирање

Индекс је посебан објекат у БП који служи за лакше претраживање БП. Реализација индекса разликује се од система до система, и овдеје изнесена једна упрошћена прича о индексима 49 ради разумевања основног концепта. Индекс садржи уређену листу вредности колоне коју индексирамо и за сваку вредност колоне показиваче на меморијске локације где се налазе редови табеле чија одговарајућа колона има ту вредност (слично индекси- ма на крају књиге, појмови уређени у абецедном поретку и за сваки списак страница где се тај појам појављује). Индексирање може да буде везано за одређена ограничења, али и не мора. За сваку колону која је примарни кључ, или колону над којом је дефинисано ограничење јединствености, аутоматски се креира индекс, који можемо да креирамо и за друге колоне, уколико постоји потреба да се по њима врши претраживање. На пример, имена и презимена аутора никако не морајуда будујединствена (могуће је да две особе имају исто име, презиме, па чак и исто име и презиме), али постоји потреба да списак аутора ефикасно претражујемо по презимену. Уколико не постоји индекс, проналажење одређеног аутора захтевало би да се редом читају сва презимена из табеле Autori док се не би дошло до жељеног. Редослед аутора у табели је насумичан и одговара редоследу уноса у табелу, тако да може да се деси и да прочитамо стотине и стотине редова пре него што нађемо презиме које тражимо. Употреба индекса много скраћује време проналажења жељеног податка, јер он садржи уреће- ну листу презимена аутора. Уз свако презиме је показивач на локацију у меморији где се чувају подаци о том аутору. У уређеном списку се лако на- лази жељени податак. Постоје ефикасни алгоритми, као штоје, на пример, бинарна претрага, којом се у неколико корака долази до жељене позиције. Када се у индексу брзо пронађе презиме, помоћу сачуваног показивача се директно приступа локацији на којој се налазе остали подаци о том аутору.

26. Референцијални интегритет

Референцијални интегритетје правило које нам гарантује исправност података страног кључа, такозване референце (показивача) на примарни кључ друге табеле. Узмимо, на пример, колону ID_izdavaca која је страни кључ у табели Knjige u показује на примарни кључ ID_izdavaca табеле Izdavaci. На основу вредности страног кључа у табели Knjige, ми знамо ко је издао коју књигу. Ако је, на пример, вредност страног кључа ID_izdavaca неке књиге 1, знамо да је ту књигу издао Завод за уџбенике зато што је вредност његовог примарног кључа 1. Овај пример описан је на следећој слици. У колони која је страни кључ не сме да постоји ниједна вредност која се не налази у колони која је примарни кључ друге табеле. Шта би се десило када бисмо у колони која је страни кључ имали вредност 100, а не постоји издавач са шифром 100? Покушај да се пронађе издавач те књиге би био неуспешан, и била би нарушена исправност података у БП. Када се за неку колону дефинише да је страни кључ, приликом уноса података у њу систем врши проверу да ли постоји та вредност у колони која је одговарајући примарни кључ. Систем ће спречити, на пример, унос шифре издавача 100, уколико у табели Izdavaci не постоји издавач са том шифром. Постоје правила којима се гарантује очување референцијалног интегритета. Важно је дефинисати како би систем требало да реагује ако се покуша брисање реда у табели на који показују редови из друге табеле. На при- мер, шта би требало да се деси ако се покуша брисање издавача Завод за уџбенике из табеле Izdavaci, уколико постоје књиге у табели Knjige чији је то издавач? У теорији, постоје три могућа решења ове ситуације: • не дозволити брисање, • обрисати ред и каскадно све редове у другој табели који показују на тај ред, • обрисати ред и поставити null за страни кључ у свим редовима друге табеле који су показивали на овај ред

27. Обрасци у Access-у

За образац се често користи и назив форма, који је настао од одговарајуће енглеске речи (енгл. Form). Помоћу обрасца можемо да прегледамо садржај табела, мењамо и бришемо постојеће податке, мада првенствено служи за лакши унос података. Образац може да се креира на више различитих начина и да садржи различите

контроле. Образац прегледно приказује податке који се налазе уједном реду табеле са пратећим лабелама, које описују приказане податке.

28. Упити у Access-у

За рад са релационим базама користи се специјализовани језик за рад са подацима који се назива упитни језик SQL. Назив овог језика настаоје као скраћеница за „структуирани упитнијезик“ (енгл.Structured Query Language). Језик SQL није осетљив на велика и мала слова, тако да све команде могу да се пишу или малим (на пример, select), или великим словима (на пример, SELECT), или комбиновано (на пример, Select). Пракса је, међутим, да се у команди ипак неке речи пишу великим, а неке малим словима ради боље читкости целе команде. СУБП Access 2007 се у највећој мери придржава стандарда SQLјезика. Команде упитног језика SQL могу да се поделе у три категорије: • дефиниционе SQL команде, • манипулационе SQL команде, • контролне SQL команде. Дефиниционе SQL команде користе се за креирање, измену и укла- њање објеката у бази података. Контролне SQL команде користе се за управљање правима која имају корисници у системима за управљање базама података која омогућавају вишекориснички рад. Манипулационе SQL команде служе за рад са подацима, и то су ко- манде: INSERT, којом се подаци уносе у табеле, UPDATE, којом се подаци мењају, DELETE, којом се подаци бришу из базе, и SELECT, којом се добијају тражене информације из базе.

У СУБП Access сваки упит, и онај који узима податке изједне, и онај који узима податке из више табела, може да се креира на један од три начина: без чаробњака, употребом дизајналата или писањем SQL упита, или употребом чаробњака. На који год начин да се креира, упит касније може да се мења и допуњује у Design View погледу, или SQL View погледу на упит. Опције за креирање упита Query Wizard u QueryDesign налазе се на траци Create/Other.

29. Сортирање података у табели Access

Подаци у табели су неуређени. Сортирање нам служи да уредимо податке. Сортира се према вредностима у одређеној колони. Уколико је потребно да се сортирају књиге абецедно по наслову, потребноје да се прво кликне мишем на колону naziv. Након тога је потребно да се при- тисне десни тастер миша над овом колоном. Појављује се мени који нуди да се подаци уреде абецедно растуће SortA toZ, unu опадајуће SortZtoA. Кликните на SortA toZда би се књиге приказале уређено према називу. За колоне које не садрже текстуалне податке сортирање се врши на другачији начин. Колона која садржи бројеве може да се сортира растуће од најмањих ка највећим вредностима SortSmallest to Largest, unu опадајуће од највећих ка најмањим Sort Largest to Smallest. На следећој слици је приказано слева надесно како изгледају подаци у табели Knjige када нису сортирани, када су сортирани растуће по вредностима у ко- лони godina_izdanja u када су сортирани опадајуће по години издања. Уколико је садржај табеле Knjige сортиран растуће према години издања, прво су приказана најстарија издања, а затим редом све новија.

30. Филтрирање података у табели – Access

Филтрирање подразумева издвајање дела садржаја табеле који одговара неком услову. Претпоставимо да су нам потребна издања књига објављена између 2008. и 2012. године, укључујући и те две године. Креираћемо филтер који ће издвојити жељене редове. На траци Home/ Sort&Filter се налази дугме Filter. Као u за Find, важноје да прво обележимо мишем колону која нам је кључна за филтирање. У овом случају то је колона godina_izdanja. Након што се кликне на дугме Filter, појављује се мени над обележеном колоном. Изабрати филтер Number Filters/Between... Уместо свих редова у табели, сада могу да се виде само они који одговарају услову који смо поставили. У дну се види да се ради о филтрираним подацима - Filtered u да их има мање него што укупно има редова. На следећој слици се види да 8 редова одговара услову филтера. Уколико се мишем кликне на Filtered, прелази се на преглед свих података у табели, што је наглашено речју Unfiltered при дну екрана. На следећој слици се види да табела има укупно 32 реда. Кликом на Unfiltered може поново да се пређе на филтриране

податке. Филтер је врста једноставног упита за враћање информација из табеле. Када се налазимо у Design View погледу на филтер, филтриране податке можемо да видимо када активирамо Datasheet View поглед, или кликом на дугме Run на траци Design/Result

31. <u>Извјештаји у Access-у</u>

Извештај прегледно приказује информације које добијамо из базе података. Овако припремљене информације могу само да се прегледају, али и да се једноставно штампају уколико је то потребно. За извештај може да се чује и назив репорт, који је настао од одговарајуће енглеске речи (енгл. Report).

32. Шта је SQL?

SQL упитни језик орногућује да корисници могу ад хок формулисати и постављати питања и веома брзо добијати одговор, а да притом не зависе од сарадње са програмером. Програмери су на тај начин растерећени и посвећују се изради квалитетних програма за апликације, које захтевају много техницког знања (нпр. веома фреквентне трансакције, код којих је важно да се постигне сто краће време одзива). Непроцедурални језик SQL (Structured Query Language) дизајниран је тако да га са успехом могу користити и људи без техничких знања с подручја обраде података, такозвани крајњи корисници. SQL језик је непроцедуралан, јер специфицира операције у смислу ШТА треба урадити, а не КАКО. SQL омогућује повезивање са класичним вишим програмским језицима, као сто су: COBOL, PUI, PASCAL, FORTRAN, C и др. SQL језик је усвојио Комитет Америчког националног института за стандарде (АНСИ) као стандардни језик релационих база података. У релационим базама података се дефинисе структура података у облику табела. SQL релациони језик прави нове табеле, дефинишуци подскупове илили комбинујуци постојеће табеле. Дакле, SQL омогућује да се једном релационом наредбом могу читати, ажурирати или брисати редови меморисани у релационој бази података. SQL је језик за: • Интерактивно дефинисање базе података (Data Definition Language ili DDL), којим треба да се омогуће креирање (CREATE TABLE), додавање (ALTER TABLE), брисање табела (DROP TABLE), погледа (CREATE VIEW) и синонима (CREATE SYNONIM).

33. <u>Употребе наредбе DISTINCT</u>

<u>Елиминација дуплих редова</u> - DISTINCT

Како се у појединим колонама понављају подаци, то се опцијом DISTINCT приказују само различити подаци за изабрану колону. Доградена синтакса наредбе SELECT има следеци изглед:

SELECT [DISTINCT] колона {,колона ... } FROM табела

34. <u>Примјену наредбе WHERE</u>

Избор специфицираних редова WHERE клаузула

С обзиром на то да SELECT klauzula ornogućava da se dobiju željene kolone iz tabele, da bi se dobili odredeni redovi iz tabele, potrebno je dodati WHERE klauzulu u SELECT наредби. Доградена синтакса наредбе SELECT ima sledeci izgled: SELECT [DISTINCT] kolona [,kolona ...] FROM tabela [WHERE uslov_selekcije]

WHERE klauzula odgovara operatoru restrikcije u relacionoj algebri. U WHERE klauzuli upoređuju se vrednosti kolona, literal-vrednosti, aritmetički izrazi ili funkcije. Uslovi selekcije u WHERE klauzuli su: • operatori poredenja (kao sto su =, >, >=, <=), • operatori ranga (BETWEEN i NOT BETWEEN), • liste (IN, NOT IN), • uzorci (LIKE i NOT LIKE), • nepoznate vrednosti (IS NULL i IS NOT NULL), • висеструки услови претразивања (AND, OR).

35. <u>Примјена наредби ранга</u>

Operator BETWEEN omogućava da se izaberu redovi koji sadrže vrednosti u nekom rasponu, a koje je korisnik specificirao. Na primer, treba videti listu svih zaposlenih Osoba, čija je plata u rasponu 12000 i 14000.

SELECT Prezime, Plata FROM OSOBA WHERE Plata BETWEEN 12000 AND 14000

Operator NOT BETWEEN омогућава да се изаберу редови који су ван вредности у неком распону, а које је корисник специфицирао. На пример, треба видети листу свих запослених Особа, чија плата НИлЕ у распону 12000 и 14000.

SELECT Prezime, Plata FROM OSOBA WHERE Plata NOT BETWEEN 12000 AND 14000

36. Примјена наредбе ORDER BY

Коришћењем клаузуле ОРДЕР БУ контролише се редослед приказивања редова. Ова клаузула се додаје на крају СЕЛЕЦТ наредбе. Дограђена синтакса наредбе СЕЛЕЦТ има следећи изглед:

СЕЛЕЦТ [ДИСТИНЦТ] колона [,колона ...] ФРОМ табела [WXEPE услов-селекције] ОРДЕР БУ (израз | позиција) [АС] | ДЕСЦ]

На пример, ако се жели приказати листа запослених у одељењу 30, али да одговарајући редови буду приказани по ставци плате, у растућерн редоследу треба написати:

SELECT Plata, RadnomestoID, Prezime FROM OSOBA WHERE ODELjENjEID= '30' ORDER BY Plata

ORDER BY клаузула проузрокује сортирање редова у растућерн низу (АСЦ), тако да је најмања плата на првом месту листе. Растући низ (АСЦ) постављен је по дефаулту и ову клаузулу не треба експлицитно наглашавати.

37. Примјена наредбе GROUP BY

GROUP BY клаузула логицки дели табелу на групе н-торки тако да у оквиру једне групе све н-торке имају исту вредност задате колоне. Овим се омогућује да функције за добијање сумамих информација буду примењене на сваку овакву групу посебно, уместо на целу табелу. Синтакса ове клаузуле има следећи изглед:

SELECT [DISTINCT] kolona [,kolona ...] FROM tabela [WHERE uslov-selekcije] [GROUP BY izraz {,izrazl] ORDER BY (izraz|pozicija)[ASC I DESC]

38. Примјенуа наредбе HAVING

Klauzula HAVING koristi se zajedno sa GROUP BY и за групу н-торки и има исти ефекат као WHERE klauzula za pojedinacne n-torke. Klauzula HAVING има задатак да специфицира услове претраживања у оквиру GROUP BY клаузуле. Синтакса ове клаузуле има следећи изглед:

SELECT [DISTINCT] kolona [,kolona ...] FROM tabela [WHERE uslov-selekcije] [GROUP BY izraz {,izraz}] [HAVING grupni uslov] ORDER BY (izraz|pozicija)[ASC | DESC]

39. Примјена наредбе за рачунање средње вриједности –SQL

Функција AVG ([DISTINCT I ALL I ,n) израчунава средњу аритметицку вредност игноришући нулл вредности. Тако, ако се жели израчунати средња аритметицка вредност плата Особа, треба писати:.

SELECT AVG (Plata)AS SREDNJA_ARIT_VRED FROM OSOBA

40. Приказати примјену наредбе за рачунање суме –SQL

Функција SUM ([DISTINCT | ALL] expr) израчунава укупни збир вредности израза exпр. Тако, ако се жели израчунати укупан збир плата Особа за RADNOMESTOID = '02', треба писати:

SELECT SUM (Plata) FROM OSOBA WHERE RadnomestoID='02'

41. **Примјену наредбе за рачунање минимума и максимума-SQL**

Функција MIN ([DISTINCT I ALL I expr) израчунава минималну вредност израза exпр. Тако, ако се жели израчунати најмања плата Особа за RADNOMESTOID = '02', треба писати:

SELECT MIN (Plata) FROM OSOBA WHERE RadnomestoID='02'

Функција MAX ([DISTINCT | ALL] expr) израчунава максималну вредност израза exпр. Тако, ако се жели израчунати максимална плата Особа за RADNOMESTOID = '02', треба писати:

SELECT MAX (Plata) FROM OSOBA WHERE RadnomestoID='02'

42. **Упити над више табела – SQL**

Упити над више табела се реализују тзв. улагањем једног упита у други или преко упита спајања (engl.join queries). Упити који се могу реализовати улагањерн упита могу се добити и помоћу упита спајања (JOIN), па је нацин писања упита ствар избора корисника. JOIN кориснику дозвољава да бира податке из две или више табела и да комбинује изабране податке у једну резултујућу табелу. Дакле, ако се жели да у једном упиту буду издвојени подаци из више табела, потребно је табеле из којих се издвајају одређени подаци 137 навести у клаузули FROM. Са становишта процедуралног програмирања (коришћење језика FORTRAN, COBOL и др.), процедура повезивања две табеле, бирање специфицних поља из специфицних редова и сортирање резултата траже компликованији програм, поготову са повећавањем броја табела које се спајају. То није случај са SQL, јер он ради непрочедурално, тј. каже се КОЈИ се податак жели, а не КАКО да се добије. С обзиром на то, дефинисани су следећи типови спајања: • спајање на једнакост (Equal Join), • коришћење групних функцуја у JOIN -у, • спајање на основу неједнакости (Not-Equal Join), • спајање са самим собом (Self-Join), • спољно спајање (Other Join).

43. **Типови података у MYSQL бази података**

Tip podatka	Opis
INT	cijeli broj uobičajene veličine
TINYINT	vrlo malen cijeli broj (raspon -128 do 127)
SMALLINT	malen cijeli broj
MEDIUMINT	srednje velik cijeli broj
BIGINT	veliki cijeli broj
FLOAT	decimalni broj s pomičnim zarezom
DOUBLE	decimalni broj s pomičnim zarezom (dvostruka preciznost)
DECIMAL	decimalni broj
DATE	datum (format YYYY-MM-DD)
DATETIME	datum i vrijeme (format YYYY-MM-DD HH:MM:SS)
TIMESTAMP	vremenska oznaka (format YYYY-MM-DD HH:MM:SS)
TIME	vrijeme (format HH:MM:SS)
YEAR	godina (format YYYY ili YY)
CHAR	niz znakova fiksne duljine (maksimalno 255)
VARCHAR	niz znakova varijabilne duljine (također maksimalno 255)
TEXT	tekstualni podaci
TINYTEXT	tekstualni podaci manje veličine

MEDIUMTEXT	tekstualni podaci srednje veličine
LONGTEXT	veliki tekstualni podaci
BINARY	niz bajtova fiksne duljine (do 255)
VARBINARY	niz bajtova varijabline duljine (do 255)
BLOB	binarni podaci (Binary Large OBject)
TINYBLOB	binarni podaci manje veličine (do 255 bajtova)
MEDIUMBLOB	binarni podaci srednje veličine
LONGBLOB	veliki binarni podaci (do 4 GB)
ENUM	enumeracija – vrijednost može biti jedna od predefinisanih vrijednosti
SET	skup – podatak može poprimiti nijednu, jednu ili više predefinisanih vrijednosti

Најчешће кориштени типови података су INT за цијеле бројеве, DOUBLE или DECIMAL за децималне бројеве, DATETIME за вријеме, VARCHAR или TEXT за знаковне низове, те BLOB за бинарне податке.
Приликом дефиниције табеле, за поља се, уз тип података, наводи и величина поља. Ако се не наведе, користи се предефинисана величина за тај тип.
Свако поље, ако се не дефинише супротно, може попримити вриједност NULL, што је посебна константа која означава да поље не садржи податак. За поља чије вриједности требају бити обавезно постављене, потребно је приликом дефиниције навести *NOT NULL*.
Такође је могуће задати предефинирану вриједност поља коришћењем кључне ријечи *DEFAULT*.
Примарни кључ се поставља навођењем кључних ријечи PRIMARY KEY на крају дефиниције поља.

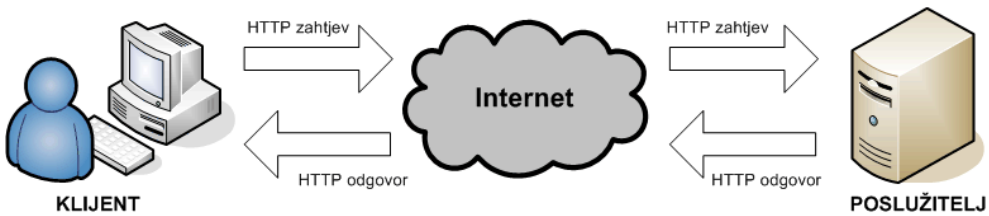
44. Шта је PHP?

PHP је скриптни језик који се извршава на послужитељу, а главна му је намјена динамичко стварање web страница. MySQL је систем за управљање релацијским базама података, и заједно с програмским језиком PHP представља једно од најпопуларнијих рјешења за израду динамичких web страница и web апликација темељених на базама података.

45. Клијентско и послужитељско скриптирање

У комуникацији путем Интернета судјелују двије стране: клијент и послужитељ (*сервер*). Клијент (корисников web прегледник) шаље захтјев, нпр. адресу неке странице, а послужитељ испоручује одговор, који садржи HTML кôд тражене странице (и заједно с њим бинарне датотеке, нпр. слике, видео...), што ће клијент интерпретирати и приказати кориснику као web страницу.

Комуникација између послужитеља и клијента одвија се путем HTTP протокола, који се састоји од HTTP захтјева и одговора. Комуникација преко Интернета је двосмјерна – и клијент може унутар HTTP захтјева послати неке податке послужитељу.



Послужитељске скрипте, као и клијентске скрипте, налазе се на послужитељу и, такође, могу бити убачене у HTML кôд, или се могу налазити у засебним датотекама. Кад стигне захтјев од клијента, скрипта се извршава на послужитељу, а као резултат извршавање добива се HTML кôд који се шаље клијенту. Највећа предност послужитељских скрипти је њихова могућност повезивања с базама података, што је омогућило појаву web

страница и web апликација с динамичким садржајем који се чита из базе података и који корисници могу мијењати преко web апликације.

46. Креирање базе података помоћу PHP-а.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";

// Kreiranje konekcije
$konekcija = mysqli_connect($servername, $username, $password);
// Provjera konekcije
if (!$konekcija) {
   die("Neuspješna konekcija: " . mysqli_connect_error());
}

// Kreiranje baze podataka
$sql = "CREATE DATABASE testiranje";
if (mysqli_query($konekcija, $sql)) {
   echo "Baza podataka je uspješno kreirana!";
} else {
   echo "Grepka u kreiranju baze podataka:" . mysqli_error($konekcija);
}

mysqli_close($konekcija);
?>
```

47. Креирање и попуњавање табела помоћу PHP-а.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
   echo "Table MyGuests created successfully";
```

```
} else {
   echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

48. Приказивање података из базе података – PHP

```
<?php
$servername="localhost";
$username="root";
$password="";
$baza="test";

//Usposatvljanje konekcije
$konekcija=mysqli_connect($servername,$username,$password,$baza);

//Provjera konekcije
if(!$konekcija)
{
        die("Neuspješna konekcija!".mysqli_error());
}
echo "Uspješna konekcija";

//sql nardba za popunjavanje tabele

 echo "<br/>";
$sql = "SELECT Ime, Prezime FROM Korisnici";
$rezultat = mysqli_query($konekcija, $sql);

 while ($red = mysqli_fetch_assoc($rezultat))
 {
 echo "Ime i prezime: ".$red["Ime"]."  ".$red["Prezime"];
 echo "<br/>";
 }
 echo "Dohvaćeno " . mysqli_num_rows($rezultat) . " redova";
 mysqli_free_result($rezultat);
;
?>
```

49. Аутентикација корисника– PHP

Аутентикација је провјера идентитета корисника, односно потврђивање да је он доиста онај за кога се представља. Најчешћи начин остваривања аутентикације је помоћу корисничког имена и лозинке. Корисник се представља својим корисничким именом, док се провјера је ли то стварно он извршава помоћу лозинке.
При првом приступању корисника апликацији, од корисника се тражи унос корисничког имена и лозинке. Након тога се провјерава постоји ли корисник с тим корисничким именом и том лозинком у бази података. Ако не постоји, приступ остатку апликације му се онемогућује док не упише тачно име и лозинку. (Дозвољени број покушаја уписивања имена и лозинке се често ограничава.)
Ако је корисник уписао одговарајуће име и лозинку, омогућит ће му се приступ остатку апликације. На овом мјесту ће се најчешће, коришћењем варијабле сједнице, записати да је аутентикација корисника извршена. Захваљујући томе, корисник неће морати поновно уписивати лозинку приликом сваког новог захтјева.
Кад сједница истекне, корисник ће се морати поновно аутентицирати уписивањем корисничког имена и лозинке.

Поставља се питање је ли могуће да корисник подастре лажни идентификатор сједнице (који може промијенити измјеном *колачића* или URL-а) и на тај начин се представи као већ аутентицирани корисник. Та је могућност математички врло мало вјеројатна, зато што свака PHP сједница добија насумични идентификатор, који се затим енкриптира.
И лозинке се у бази података често држе у криптираном облику, да нетко тко има приступ бази не би могао преузети лозинке других корисника.
За енкрипцију лозинки, (као и идентификатора сједнице), користи се посебна метода енкрипције – израчунавање сажетка поруке (*message digest*, *hash*) која је једносмјерна. То значи да се из записа у бази не може добити оригинална лозинка, али се из лозинке може лако добити њен енкриптирани облик и тако провјерити је ли уписана лозинка тачна.

Сједнице

Послужитељ остварује **сједницу** (спој, сесију) с појединим клијентом као низ HTTP захтјева и одговора између њих. Свака сједница добива свој јединствени идентификатор, на темељу којега се препознаје да поједини захтјеви припадају тачно одређеној сједници.
Идентификатор сједнице се ствара на послужитељу, и шаље клијенту унутар HTTP одговора. Клијент га потом просљеђује назад приликом сваког сљедећег захтјева.
Сједница ће трајати све док стижу захтјеви клијента. Ако од посљедњег захтјева протекне одређено вријеме (у PHP-у је предефинисана вриједност тог времена 30 минута), сједница ће се аутоматски затворити.
Основни начин за просљеђивање идентификатора сједнице и његово чување на клијентском рачунару су колачићи. Ако су колачићи онемогућени или нису подржани у корисниковом *web* прегледнику, идентификатор сједнице ће се просљеђивати преко URL-а. Примјер таквог URL -а:

`http://www.srce.hr?PHPSESSID=242c489fb4a1bc79f5cf365988167e4d`

Сједнице се могу употребити за спремање података који су везани за сједницу, дакле за појединог корисника. Такви подаци спремају се у варијабле сједнице које ће бити видљиве само за тог корисника и трајат ће само док траје и сједница.

50. Неке уграђене функције у PHP-у

Funkcija ***trim*** koristi se za micanje praznina s početka i kraja znakovnog niza. Pod prazninama se smatraju razmaci, tabulatori i znakovi za novi red. Funkcija vraća niz iz kojeg su maknute praznine ili zadani znakovi.

Funkcije ***strtoupper*** i ***strtolower*** služe za pretvaranje svih znakova u nizu u znakove napisane samo velikim, odnosno samo malim slovima.

Funkcija ***strlen*** vraća duljinu zadanog niza.

Funkcija ***substr*** služi za dobivanje dijela ulaznog niza. Argumenti koje ova funkcija prima su ulazni niz, položaj od kojeg počinje traženi podniz, i duljina podniza (opcionalno). Funkcija vraća dobiveni podniz.

Funkcija ***str_replace*** koristi se za zamjenu dijelova niza u ulaznom nizu. Argumenti koje prima traženi su podniz, zatim podniz kojim ga treba zamijeniti, i ulazni niz. Kao četvrti (opcionalni) argument može se predati broj zamjena (koliko puta treba obaviti zamjenu).

Funkcija ***explode*** služi za pretvaranje znakovnog niza u polje. Argumenti koje ova funkcija prima su niz po kojem se ulazni niz rastavlja, zatim ulazni niz i, kao opcionalni argument, maksimalni broj članova na koje se niz može rastaviti. Funkcija vraća dobiveno polje.

Funkcija ***implode*** ima obrnutu svrhu - služi za pretvaranje polja u znakovni niz. Argumenti koje prima su niz koji će se ubaciti između članova polja i polje koje treba pretvoriti u niz.

Funkcija ***count*** vraća broj članova polja.

Funkcija ***in_array*** provjerava nalazi li se zadani član u polju.

Funkcija ***array_sum*** će vratiti zbir svih članova polja.

Funkcija ***shuffle*** će nasumično promijeniti poredak članova polja.

Funkcija ***sort*** služi za sortiranje članova polja.

Funkcija ***asort*** sortira polje sa znakovnim ključem po vrijednostima, a funkcija ***ksort*** po ključevima

Funkcija ***each*** dohvaća trenutni član polja.

Funkcija ***mktime*** koristi se za stvaranje Unixove vremenske oznake. Argumenti koje funkcija prima su sat, minuta, sekunda, mjesec, dan i godina. Svi argumenti su opcionalni, a ako se izostave koriste se trenutne vrijednosti: trenutna godina, trenutni dan, trenutni mjesec, itd.

Funkcija ***date*** služi za pretvaranje *Unixove vremenske oznake* u željeni format. Argumenti koje prima su format i vremenska oznaka.

Funkcija ***getdate*** koristi se za dohvaćanje pojedinog podatka iz vremenske oznake. Funkcija prima vremensku oznaku kao argument, a ako se pozove bez argumenta, uzima se trenutno vrijeme.

Funkcija ***checkdate*** koristi se za provjeru je li zadani datum ispravan. Argumenti funkcije su mjesec, datum i godina.

Funkcija ***round*** koristi se za zaokruživanje decimalnog broja.

Funkcija ***ceil*** koristi se za zaokruživanje decimalnog broja na prvi veći cijeli broj.

Funkcija ***floor*** koristi se za zaokruživanje decimalnog broja na prvi manji cijeli broj.

Funkcija ***max*** vraća najveći u nizu brojeva.

Funkcija ***min*** vraća najmanji u nizu brojeva.

Funkcija ***sqrt*** računa drugi korijen iz zadanog broja. Kao argument prima broj čiji korijen treba izračunati.

Funkcija ***rand*** koristi se za dobivanje slučajno odabranog broja.

Funkcija ***exit*** koristi se za prekid rada skripte.

Funkcija ***isset*** provjerava je li nekoj varijabli pridijeljena vrijednost.