# Базе података

## Основи база података

Базе података (енгл. database) присутне су у великој мери у свакодневном животу, при чему олакшавају и убрзавају многе процесе. На пример, приликом резервације авионских карата користи се база података у којој се бележи колико је карата расположиво за продају, ко је већ резервисао карту на којем лету и слично. Такође, већина продавница рачунарске опреме има свој веб-сајт на којем се за описивање доступних производа користи база података. На тај начин се може добити списак производа по категоријама, списак оних производа који су на акцији, техничку спецификацију сваког појединачног производа, као и информацију о томе у којој је радњи доступан сваки понуђени производ. Чак се и у позадини блогова налазе базе података – све објаве на блогу организоване су коришћењем базе података и тиме се могу приказати објаве са сличном тематиком, могу се остављати коментари и слично. За неке од наведених примена кључно је да подаци буду добро вођени и да поштују правила из пословног домена на који се односе – каже се да буду конзистентни (или ваљани). На пример, важно је да иста авионска карта не може да се прода више пута. Из корисниковог угла важна је конзистентност података и у неким другим применама: на пример, списак продавница у којима може да се купи неки производ треба да се редовно ажурира како би корисник могао да изабере најближу продавницу у којој се сигурно производ налази. Исто тако, изузетно је важно да такви системи раде што ефикасније јер, на пример, и продавац авионских карата и купац желе да продаја карата буде што бржа, без беспотребног чекања. Базе података данас се налазе свуда: иза рачуна у банкама, књига у библиотекама, здравствених организација, друштвених мрежа...

Некада су се за овакве и сличне примене користили елементарни приступи рада са подацима: полазећи од описа тражене апликације, формирале би се датотеке за складиштење свих потребних података и писани су програми који обрађују податке из тих датотека. Недостаци оваквих система за чување и обраду података су следећи:

* сувишно понављање података – на пример, и датотека коју користи апликација за одржавање скупа расположивих производа у продавници рачунарске опреме и датотека коју користи апликација за продају тих производа обухватала би све податке о том производу,
* потенцијална неконзистентност података која се може јавити приликом промене вредности само неких појава једног истог податка – на пример, могуће је да само у једној од две наведене датотеке буде промењена цена производа, чиме долази до неконзистентности података,
* узајамна зависност програма за обраду података и начина на који су подаци структуирани, као и промена организације података подразумева и промену програма.

Базе података су настале 1960-их са циљем да се елиминишу наведени недостаци. Појам базе података означава скуп међусобно повезаних података који се чувају заједно и међу којима има само онолико понављања колико је потребно за угодан вишекориснички рад. Обично се ради о скупу података који се односе на исту тему. На пример, у бази се могу чувати подаци о школским активностима ученика једне школе: подаци о ученицима, професорима, наставним плановима, као и оцене ученика. Сви наредни концепти биће илустровани над базом података којом се реализује електронски дневник једне школе.

У суштини, добро испланиране и направљене базе података олакшавају рад програмеру јер о многим стварима не мора да води рачуна: на пример, не мора да води рачуна о томе да треба да ажурира све копије истог податка зато што је најпре број копија истог податка сведен на минимум, а постоји и механизам којим систем може да ажурира све потребне вредности, односно да забрани измену уколико је то потребно. Tакође, базе података раздвајају физичку репрезентацију података од начина на који се подаци приказују кориснику, тзв. логичког погледа на податке. Овај уграђен механизам омогућава прикривање детаља организације података, због чега се програми увек пишу на исти начин. За крајњег корисника то није важно колико за самог програмера базе података јер крајњи корисник види податке који на различите начине могу да буду представљени на диску и није неопходно да зна који се механизам складиштења и обраде података користи, да ли су у питању базе података или систем датотека.

Након сваке активности над базом података (читање података, измена података, упис података) проверава се стање базе које мора остати конзистентно – то значи да сви подаци у бази морају задовољавати неке услове који су унапред задати. За ефикасан рад са подацима и за одржавање конзистентног стања базе података користе се системи за управљање базама података, скраћено СУБП (енгл. DataBase Management System – DBMS). Ове системе одликују следеће карактеристике:

* омогућено је складиштење велике количине података уз минимално понављање,
* остварена је независност структура података од програма који раде над тим подацима и обратно,
* уведена је контрола приступа подацима и више корисника може истовремено да користи податке.

Поред наведених карактеристика, битан фактор представља и сигурност – с обзиром на то да је велики број база података активан по 24 часа дневно и да им истовремено приступа више корисника, неопходно је да СУБП буде толерантан на могуће грешке. Из тог разлога неопходно је да се копије базе података одржавају на више различитих места, тзв. репликација базе података. Такође, потребно је омогућити механизме којима се контролише којим подацима може да приступи корисник и које операције сме да изведе.

База података заједно са СУБП чини систем база података.

Централно место код база података заузимају упити – то су наредбе којима се издвајају подаци према одређеном скупу услова. На пример, упитом би се могла издвојити времена поласка и доласка свих летова из Београда за Барселону у наредна три дана или се могу издвојити шифре расположивих штампача у продавници уређени опадајуће према цени. Такође, све објаве на блогу којe се баве образовањем могу се посебно издвојити.

### Модели података

Током развоја система база података развијено је неколико генерација система за управљање базама података и њихова главна разлика лежи у моделу података. Моделом података представља се логичка структура података – то је начин на који се подаци из физичке базе података (са диска) представљају кориснику тако да њихова репрезентација буде погоднија за корисника од физичке репрезентације. Моделом података се дефинише и скуп операција које корисник може да изврши над тим подацима. Модел података чини основу за логичко пројектовање базе, тј. утиче на то како добро структуирати податке да би њихово одржавање било једноставно, а руковање њима ефикасно. Такође, он чини и основу за имплементацију базе података. Модел података се састоји из три компоненте:

* структурни део модела – њиме се задаје који су типови објеката подржани и која су њихова својства. На пример, релациони модел подржава релацију као основни тип објекта. Типовима објеката се описују објекти или ентитети – то су ствари, лица, појаве или догађаји које је могуће једнозначно идентификовати. Тако тип објекта у бази артикала продавнице рачунара може бити монитор и он се задаје својствима: модел, дијагонала, резолуција, фреквенција освежавања, тип осветљења и цена. Један примерак типа објекта монитор може бити монитор XY201 са дијагоналом од 60cm, резолуцијом 1920x1200, фреквенцијом освежавања од 60 херца, LED осветљењем и ценом 20000 динара или пак примерак објекта може бити догађај да је тај исти монитор купљен у продавници у Нишу.
* руковање подацима – овом компонентом се задаје скуп операција над објектима подржаних типова. Операције могу бити радње, попут „изменити цену одговарајуће рачунарске компоненте“, или упити, као, на пример, „који су монитори дијагонале 55cm расположиви у продавници у Београду?“.
* интегритет података – њиме се задаје скуп услова које подаци или односи међу подацима морају да задовоље. Један пример услова интегритета био би да дијагонала монитора мора бити између 48cm и 140cm.

Најпознатији класични модели података су хијерархијски, мрежни и релациони модел. Данас је доминантни модел релациони модел података, док се неретко користи и објектни (тј. чешће објектно-релациони) модел. У наставку ће бити детаљно изложен релациони модел података.

### Релациони модел података

Релациони системи су данас најзаступљенији системи база података. Релациони модел, на коме су ови системи засновани, теоријски је заснован још крајем шездесетих година двадесетог века.

#### Структура података

Основна идеја релационог модела је да се подаци представљају релацијама (у математичком смислу) табеларног изгледа. Свака релација има свој назив по коме је разликујемо од осталих релација у бази. У врстама (слоговима) табеле налазе се подаци о појединачним ентитетима релације, а у колонама су дата својства ентитета која ћемо називати атрибутима. Сваки атрибут такође има свој назив по коме ћемо га разликовати од осталих атрибута у тој релацији. На пример, скуп свих података о ученицима представићемо табелом под називом Ucenik. У њој ће се по врстама налазити подаци о ученицима, а по колонама ће бити дата својства ученика, као што су име, презиме, име оца, јединствени матични број грађана и датум рођења. Пример садржаја релације Ucenik приказан је у следећој табели:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ime | Prezime | ImeOca | JMBG | DatRodjenja |
| Petar | Marković | Milan | 2105005735322 | 21.05.2005. |
| Ana | Jovanović | Marko | 1101004758923 | 11.01.2004. |
| Milan | Marković | - | 2204005442534 | 22.04.2005. |
| Petar | Petrović | Marko | 0305004123123 | - |
| Ana | Jokić | Ivan | 0509004333111 | 05.09.2004. |

Сва наведена својства сматраћемо атрибутима ове релације и именоваћемо их редом као Ime, Prezime, ImeOca, JMBG и DatRodjenja, а релацију Ucenik записати као Ucenik(Ime, Prezime, ImeOca, JMBG, DatRodjenja). Скуп свих могућих вредности неког атрибута зваћемо доменом атрубута. То морају бити вредности истог типа, као што је, на пример, скуп свих личних имена. Тако, на пример, атрибути Ime и ImeOca узимају вредности на скупу свих личних имена, атрибут Prezime на скупу свих презимена, а атрибут DatRodjenja на скупу свих датума. Као што видимо у случају имена ученика и имена његовог оца, два атрибута исте релације могу имати исти домен.

Један од принципа релационог модела јесте да се табела посматра као скуп врста, те је редослед врста у табели небитан. Када је потребно да се подаци из врста табеле прикажу у жељеном поретку, неопходно је да тај поредак нагласимо експлицитно приликом формулисања упита. Такође, није ни значајан редослед колона у табели јер свака колона има јединствено име.

У релационом моделу важи да у табели не смеју постојати два идентична реда јер је релација представљена табелом скуп, тј. колекција различитих објеката. Притом треба имати у виду да је ова релација динамичког карактера. То значи да се с временом неки њени ентитети бришу, неки додају, а неки мењају У релацијама се често уводи неки јединствени идентификатор који не сме имати исту вредност на различитим ентитетима релације. На тај начин се кориснику олакшава руковање релацијама јер једино треба да води рачуна о томе да вредности овог атрибута у релацији буду међусобно различите.

Запис Ucenik(Ime, Prezime, ImeOca, JMBG, DatRodjenja) означава шему или схему (енгл. schema) релације Ucenik. Шемом се у општем случају задаје број атрибута релације, њихови називи (који морају бити различити) и њихови домени. У нашем случају шемом релације Ucenik задаје да се релација Ucenik састоји од пет атрибута и да су ти атрибути именовани са Ime, Prezime, ImeOca, JMBG, DatRodjenja(претпоставља се да је из назива атрибута лако одредити њихове домене). Битно је разликовати саме податке дате релације од шеме релације којом је задата њена структура.

На сличан начин може се задати и релација која садржи све предмете који се предају у школи. У релацији Predmet дат је јединствени идентификатор за сваки предмет, као и његов назив, разреди у којима се предаје, недељни фонд часова и информација да ли је предмет обавезан или није. Шема релације Predmet изгледала би на следећи начин:

Predmet(IdPredmeta, Naziv, Godina, Fond, Obavezan).

Поред ове две релације, разматраћемо и наредне релације:

Profesor(IdProfesora, Ime, Prezime)

Predaje(IdProfesora, IdPredmeta, SkolskaGod)

Ocena(JMBGUcenika, IdPredmeta, IdProfesora, Ocena, DatOcenjivanja)

Њима се редом задају скупови свих професора у школи, информација о томе који професор предаје који предмет у одређеној школској години, као и информација о томе да је неки ученик добио оцену из неког предмета код неког професора. Даћемо пример садржаја ових релација, при чему ћемо због прегледности овај садржај поједноставити у односу на ситуацију у пракси. У наставку текста, када будете стекли више знања и искуства у раду са базама података и када буде било потребе да се илуструју неки нови концепти, садржај ових релација биће проширен. Треба напоменути и да се овако задата база података дневника школе разликује од традиционалног школског дневника.

Нека је садржај ових релација следећи:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IdPredmeta | Naziv | Godina | Fond | Obavezan |
| 1 | Matematika | 1 | 5 | Da |
| 2 | Matematika | 2 | 5 | Da |
| 3 | Srpski jezik | 1 | 4 | Da |
| 4 | Računarstvo i informatika | 4 | 2 | Da |
| 5 | Veronauka | 4 | 1 | Ne |

|  |  |  |
| --- | --- | --- |
| IdProfesora | Ime | Prezime |
| 1 | Petar | Petrović |
| 2 | Marko | Marković |
| 3 | Anita | Antić |
| 4 | Jelena | Jelenković |

|  |  |  |
| --- | --- | --- |
| IdProfesora | IdPredmeta | SkolskaGod |
| 1 | 1 | 2019 |
| 1 | 2 | 2019 |
| 2 | 3 | 2019 |
| 4 | 4 | 2019 |
| 3 | 1 | 2020 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| JMBGUcenika | IdPredmeta | IdProfesora | Ocena | DatOcenjivanja |
| 2105005735322 | 1 | 1 | 5 | 21/03/2020 |
| 2105005735322 | 4 | 4 | 3 | 10/02/2020 |
| 2105005735322 | 1 | 3 | 4 | 03/04/2020 |
| 2204005442534 | 2 | 1 | 5 | 10/03/2020 |
| 0305004123123 | 1 | 1 | 4 | 03/12/2019 |

Битан принцип релационог модела јесте да све вредности у релацијама морају бити атомичне, односно да се не могу разложити на компоненте. Рецимо, ако бисмо уместо релације Ocena имали релацију која садржи атрибуте JMBGUcenika, IdPredmeta и SkupOcena, при чему би последњи атрибут имао као вредност скуп свих оцена датог ученика из одређеног предмета, онда та вредност не би била атомична. Такође, уколико бисмо у релацији Ocena уместо атрибута JMBGUcenika имали примерак ентитета Ucenik, опет вредност тог атрибута не би била атомична јер се она може разложити на компоненте – то би били атрибути релације Ucenik.

Релациона база података задаје се скупом оваквих релација. Скуп шема релација релационе базе података представља шему релационе базе података. На пример, претходно приказаним релацијама Ucenik, Predmet, Profesor, Predaje и Ocena задата је база података електронског дневника школе (такав дневник се не поклапа са традиционалним школским дневником јер, на пример, традиционални дневник не садржи јединствени идентификатор за сваки од предмета, а с друге стране, у горе задатој бази података недостају, на пример, подаци о изостанцима ученика).

Као што смо већ поменули, релација у релационом моделу података одговара појму табеле у бази података (слично, n-торка појму врста, атрибут појму колона, домен појму тип података, а шема релације појму опис табеле). Први термини се углавном користе у формалном опису релационог модела, а други у мање формалној терминологији релационих база података, као практичне реализације релационог модела. Упоредни приказ термина дат је у следећој табели:

|  |  |
| --- | --- |
| **Релациони модел** | **База података** |
| Релација | табела |
| *n*-торка | врста табеле |
| Атрибут | колона табеле |
| домен атрибута | тип податка |
| шема релације | опис табеле |

У наставку текста, наведени појмови биће коришћени као синоними.

#### Руковање подацима

Као што је већ наведено, у релационом моделу се сви подаци приказују табелама. Шта заправо може да се ради над подацима у табелама? Може се добити неки подскуп скупа података који ће задовољити одређене критеријуме. На пример, можемо добити све информације о ученицима који се презивају Петровић – тиме ће се заправо издвојити *n*-торке релације Ucenik које задовољавају дати услов. Такође се може добити скуп свих имена и презимена професора, због чега је неопходно да се издвоје атрибути Ime и Prezime релације Profesor. Поред тога, могу се издвојити имена и презимена професора са називима предмета које предају – у том случају је неопходно да се издвоје атрибути Ime и Prezime релације Profesor и атрибут Naziv релације Predmet, при чему је неопходно да се те две релације споје преко релације Predaje.

Формално посматрано, руковање подацима у релационом моделу података реализује се на два еквивалентна начина: релационом алгебром (енгл. relational algebra) и релационим рачуном (енгл. relational calculus). Релациона алгебра је алгебарски формални језик којим се задаје скуп операција над релацијама којима се долази до резултујуће релације, док се релационим рачуном описују својства која задовољава резултујућа релација. Операције издвајања атрибута (тј. колона), издвајања *n*-торки (тј. редова) и спајања две релације сматрају се елементарним и преко њих се могу изразити многе сложеније операције. Све три операције су подржане и у релационој алгебри и релационом рачуну, при чему се називају:

* пројекција – помоћу ње се издваја скуп жељених атрибута релације (на пример, релација Ucenik може се пројектовати само на имена и презимена ученика),
* рестрикција – помоћу ње се издвајају само *n*-торке релације које задовољавају неки логички услов (на пример, могу се издвојити само ученици чији се отац зове Марко),
* спајање – помоћу ње се врши комбиновање две релације (на пример, могу се спојити релације Profesor и Predaje тако да је идентификатор професора у релацији Profesor једнак идентификатору професора у релацији Predaje, чиме се у спојеним редовима добију сви подаци о професору заједно са информацијом који предмет предаје).

Постоје и неке додатне операције које су подржане само у релационој алгебри, као и неке друге операције које су подржане у релационом рачуну.

У релационој алгебри подржана је и операција дељења. Наиме, нека су дате две релације А и B и нека важи да постоји подскуп Y скупa атрибута релације A који има исти број елемената и исте домене као скуп атрибута релације B. У том случају кажемо да је резултат дељења релације A релацијом B максимални подскуп пројекције релације A на преостале атрибуте, који је у релацији са сваком n-торком релације B. Операција дељења представља аналогон универзалног квантификатора. Уз помоћ дељења можемо, на пример, издвојити предмете из којих су сви ученици добили неку оцену.

#### Интегритет података

Интегритет података код релационог модела односи се на услове који треба да буду задовољени да би стање базе података остало конзистентно. На пример, један од услова интегритета био би да се сваки ред табеле Ocena односи на неког од ученика из табеле Ucenik. Уколико би постојао ред табеле Ocena који се односи на ученика који не постоји у табели Ucenik, база података би се нашла у неконзистентном стању. Такође, као услов интегритета може се навести да оцена коју ученик може да добије мора да буде из скупа оцена {1,2,3,4,5}.

Разликујемо две врсте услова интегритета:

* општи услови – односе се на сваку табелу или базу података и везани су за концепте примарног и страног кључа,
* специфични услови – односе се на неку колону или на везу између више колона (пример таквог услова може да буде да разред у коме се предмет предаје има једну од вредности 1, 2, 3 и 4).

Правило које се назива интегритет ентитета (енгл. entity integrity) одређује да у табели у релационом моделу не могу постојати два иста реда. Кључ (енгл. key) табеле чини једна или више колона табеле које на јединствен начин одређују сваки ред те табеле и за њих важи да не постоји неки њен прави подскуп за који важи исто својство. Свака табела има минимално један кључ јер важи да комплетан скуп колона табеле у потпуности одређује сваки ред табеле, тако да у случају да не постоји неки подскуп за који важи исто својство, за кључ се може изабрати скуп свих колона табеле. Једна табела може имати више кључева, па се бира један од њих и проглашава се примарним кључем (енгл. primary key). На пример, за табелу Ucenik колона Prezime не би била добар кандидат за кључ јер може постојати више ученика с истим презименом. Чак и да га проширимо именом, може се догодити да постоје два ученика с истим именом и презименом. Добар избор за примарни кључ била би колона JMBG јер свака особа (па и сваки ученик) има различиту вредност матичног броја. Алтернативно, када бисмо табели Ucenik додали као колону број досијеа ученика под којим је заведен у дневнику који би био јединствен на нивоу школе, он би такође могао бити примарни кључ. Уколико је вредност неке колоне непозната или недефинисана, то се у релационим базама података може задати недостајућом вредношћу, о чему ће више речи бити у наставку уџбеника. Колоне табеле које улазе у састав примарног кључа не могу имати недостајућу вредност.

Релациони модел такође подразумева да подаци у бази сваког тренутка буду коректно повезани. На пример, уколико табела Ocena садржи информације о оцени ученика кога нема у табели Ucenik, дошло је до нарушавања услова интегритета. То се спречава на тај начин што се уоче колоне табела Ocena и Ucenik којима се успоставља веза ове две табеле – у конкретном случају то су само колоне JMBG и JMBGUcenika и поставља се услов да вредност колоне JMBGUcenika мора бити или у потпуности недостајућа или мора да одговара некој вредности колоне JMBG табеле Ucenik. Такав скуп колона табеле Ocena називамо страним кључем (енгл. foreign key) у односу на табелу Ucenik. У наведеном примеру, страни кључ би се састојао само из колоне JMBGUcenika. Страним кључевима задат је референцијални интегритет или интегритет обраћања (енгл. referential integrity). Табеле између којих се успоставља референцијални интегритет не морају бити нужно различите. На пример, у табели Ucenik могла би да постоји колона Tutor која би означавала тутора ученика (уколико га има) који је и сâм ученик, и у том случају би табела Ucenik имала страни кључ у односу на себе саму.

Од тренутка дефинисања примарних и страних кључева у бази података, гарантовање интегритета ентитета и референцијалног интегритета налази се у надлежности система за управљање базама података.

#### Недостајућа (NULL) вредност

У релационим базама података подржана је специјална вредност која означава непознату или недефинисану вредност. Назива се недостајућа (или NULL) вредност. Недостајућа вредност, поред значења непознате вредности (вредност која се не зна), може имати и значење неприменљивог својства (вредност која се никада не може одредити). На пример, вредност атрибута DatRodjenja за неког ученика може бити недостајућа, у смислу да та вредност није тренутно позната. Увођењем недостајуће вредности у домене, модел се доста усложњава јер поред вредности тачно (*T*) и нетачно (*F*) постоји и недостајућа истинитосна вредност (*NULL*), и потребно је дефинисати све логичке операције у тако проширеном моделу. Њиме се прелази на домен тровалентне логике са скупом истинитосних вредности {*T*,*F*,*NULL*}. Да би се дефинисао израз у оквиру тровалентне логике, потребно је дефинисати одговарајуће тровалентне логичке операције. Оне су задате следећим истинитосним таблицама:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **А** | **~A** | | **T** | F | | **F** | T | | **NULL** | NULL | | |  |  |  |  | | --- | --- | --- | --- | |  | **T** | **F** | **NULL** | | **T** | T | F | NULL | | **F** | F | F | F | | **NULL** | NULL | F | NULL | | |  |  |  |  | | --- | --- | --- | --- | |  | **T** | **F** | **NULL** | | **T** | T | T | T | | **F** | T | F | NULL | | **NULL** | T | NULL | NULL | |

На пример, пошто важи да је и важи да је , онда важи и да је јер коју год вредност имала недостајућа вредност, резултат логичког израза биће *F*.

Такође, потребно је дефинисати и све алгебарске операције у тако проширеном моделу, али се тиме нећемо бавити у овом делу уџбеника.

### Права приступа и ауторизација

Једна од значајних улога система за управљање базама података јесте и контрола приступа бази података. То подразумева дефинисање правила ко може да приступи бази података и којим њеним деловима, као и које операције може да извршава над базом података. Уколико се говори о општим задацима као што су повезивање са базом, прављење табела или администрирање система говори се о различитим нивоима ауторизације. Постоји потреба да корисници имају различита права јер неким подацима не смеју да имају приступ сви корисници. С друге стране, неки корисник сме само да чита одређене податке, док други корисник може да их мења и слично. У том случају говори се о различитим правима. Међу корисницима треба да постоји особа која је одговорна за одржавање и функционисање базе података, и њу називамо администратором базе података (ту говоримо о ауторизацији управљања базама података). Администратор базе података има право да додаје нове кориснике и да им додељује права приступа. Он је задужен за формирање и ажурирање шеме базе података, обезбеђивање подршке у случају да се подаци изгубе, њихов опоравак итд. Обични корисници немају право да додају нове кориснике, али могу да праве нове објекте унутар базе података и да над њима додељују права другим корисницима.

У данашње време, професионалне базе података праве дизајнери база података. Они су задужени за програмски аспект дизајна базе података (поделу података у табеле, поделу информација у колоне, задавање примарних кључева табела и успостављање односа између табела, проверу правила нормализације...), безбедност података и организацију најбоље презентације информација. Многи дизајнери база података тесно сарађују с администраторима база података и од њих преузимају нека задужења, као, на пример, одржавање резервних копија (енгл. backup) података и одржавање интегритета података. С друге стране, унос, одржавање и добијање података обављају корисници.

### Разлика између податка и информације. Информациони системи

Подаци представљају уређене колекције симбола: они могу бити текстуални, нумерички, графички, звучни итд. Тако, на пример, податак може бити опис својства неког ентитета, нека чињеница или запажање и слично (на пример: име ученика). Значење података се не подразумева само по себи, већ настаје када се подаци протумаче у неком контексту. Као резултат интерпретације података у одређеном контексту настају информације. Информација увек носи неко значење. На пример, подаци постају информације у запажању: „у одељењу постоје три ученика која се зову Петар”. Иста информација може бити искоришћена у различите сврхе. На пример, оцена коју је ученик добио на писменом задатку из неког предмета служи и за рачунање коначне оцене ученика из тог предмета, а и за рачунање просечне оцене коју су ученици добили на том писменом задатку. Као резултат прикупљања података, њихове анализе и логичког повезивања настаје знање и оно се стиче кроз образовање и искуство.

Информациони систем је систем развијен у циљу сакупљања, снимања, чувања, обраде, преношења и интерпретације информација. Циљ информационог система јесте обрада прикупљених података и њихова трансформација у информације, у циљу остваривања знања за неку специфичну намену. На пример, у здравству се користе информациони системи за чување информација о пацијенту, његове историје претходних посета тој здравственој установи, за праћење пацијента у сваком сегменту лечења и слично, као и за прављење статистичких извештаја на нивоу здравствене установе. Рачунари данас представљају неопходну компоненту информационих система и они су у многоме олакшали и унапредили њихов рад. Они су првобитно коришћени за електронску обраду велике количине података. Недуго затим је уочено да могућности рачунара у многоме превазилазе обраду података и да се они могу користити и у друге сврхе. Тако су најпре настали управљачки информациони системи који аутоматски анализирају податке и као резултат нуде корисне информације, а затим и системи за подршку у одлучивању. Такође, у области вештачке интелигенције развијани су експертски системи, тј. интелигентни рачунарски програми који емулирањем начина решавања проблема људи-експерата омогућавају решавање реалних проблема који би иначе захтевали људску експертизу.

Данас базе података представљају основу сваког информационог система.

## Прављење база података у конкретном окружењу

Од свих постојећих модела база података, релационе базе података данас су најраспрострањеније. У употреби је велики број популарних релационих система за управљање базама података: MS Access, MySQL, ORACLE, MS SQL Server, IBM DB2. У наставку уџбеника упознаћемо се са два система: MS Access и MySQL. Први је у широкој употреби, али је власнички, док је други систем отвореног кода. Сви концепти који буду уведени, биће илустровани у оквиру ова два система.

### Опис радног окружења

Према истраживањима, MySQL је најчешће коришћени релациони систем за управљање базама података отвореног кода (енгл. open source). MySQL подржава велики број различитих платформи, укључујући и Windows и главне Linux дистрибуције. Овај СУБП често се користи у изради веб-апликација, за пројекте који су отвореног кода и који се могу слободно преузети, као и за сајтове с великим бројем посетилаца, као што су, на пример, Facebook и Twitter. MySQL се може слободно преузети.

Прва верзија, која је носила ознаку 3.19, појавила се 1995. године и од тада је развијено неколико верзија. У примерима који следе користи се верзија 5.6.28 (актуелну верзију у тренутку писања уџбеника).

За употребу овог СУБП-а корисник може да користи уграђену конзолну апликацију, као и ГУИ (енгл. Graphical User Interface) клијенте, као што је, на пример, phpMyAdmin, које је потребно независно инсталирати. Коришћење алатки које нуди графички интерфејс за извршавање операција над базама података, уместо коришћења командне линије у те сврхе, у великој мери чини рад једноставнијим и удобнијим. У наставку уџбеника, сви концепти ће бити изложени коришћењем phpMyAdmin клијента. phpMyAdmin је бесплатни алат отвореног кода за администрацију MySQL база података коришћењем графичког корисничког интерфејса. Може се преузети са адресе: <http://www.phpmyadmin.net/home_page/downloads.php>. Да би се покренуо, потребно је да се у веб прегледачу отвори страница <http://localhost/phpmyadmin/index.php>. На тој страници је потребно да се корисник улогује на систем.

Када се корисник улогује на систем, приказује се прозор у којем се може радити са базама података. У левом делу прозора налази се списак свих база података које постоје на датом систему. Кликом на знак + може се добити списак табела које садржи дата база података, а затим се за сваку од табела може добити списак колона и индекса. Такође, кликом на дугме Recent може се изабрати неки од недавно коришћених објеката, а кликом на дугме Favorites добијају се они који су омиљени кориснику. Изнад та два дугмета налази се иконица Home којом се враћа на првобитни екран, иконица Log out којом се може излоговати са система и иконице којима се учитава документација. Тај део прозора апликације може се сакрити кликом на стрелицу лево, која се налази у оквиру насловне линије. Насловна линија садржи и путању до објекта над којим се тренутно ради. На пример, уколико се ради над табелом Ucenik базе elektronski\_dnevnik, у насловној линији биће исписан текст: Database: elektronski\_dnevnik >> Table: Ucenik.

На врху централног дела прозора апликације налазе се картице. Њихов списак зависи од типа објекта на коме је тренутно фокус. Картица Browse омогућава приказ информација о садржају табеле, картица Structure служи за приказ структуре објекта (базе података, табеле или колоне), картицом SQL омогућено је задавање упита над објектом који је тренутно у фокусу (на пример, над конкретном базом података или само једном табелом), картица Insert служи за додавање нових редова у табелу, Privileges за давање привилегија на изабраном објекту, Operations за уклањање, преименовање, копирање објекта који је у фокусу итд.

На дну прозора налази се иконица Console којом се може прегледати све до тада извршене операције у конзолном режиму.

### Прављење базе података

phpMyAdmin има једноставан интерфејс за прављење база података MySQL. Најпре је потребно да се корисник улогује као *root* корисник или као неки други корисник који има право да направи нову базу. Након логовања, кориснику ће бити приказан формулар помоћу којег може направити нову базу података. У оквиру формулара налазе се поље за унос текста Database name у које је потребно да се унесе назив нове базе података и падајућа листа из које треба изабрати жељени начин кодирања за дату базу. Након тога треба притиснути дугме Create и нова база ће бити излистана у списку база података на левој страни прозора апликације. Пошто у новој бази не постоји ниједна табела, кориснику ће бити понуђено да направи неку табелу у оквиру те базе података.

Да би се отворила постојећа база података, потребно је да се кликне на њен назив у листи свих база података на левој страни прозора. Информације о тој бази података биће приказане све док не кликне на неку другу базу података – тиме се претходна база података аутоматски затвара. Ако желимо да затворимо и саму апликацију, потребно је да кликнемо на иконицу Log out која се налази у горњем левом углу прозора апликације.

## Рад са табелама

Као што је већ напоменуто, у релационим базама података сви подаци се приказују на јединствен начин – табелама.

Садржај табеле може се приказати тако што се изабере жељена табела на левој страни прозора, а затим и картица Browse. Након тога ће се приказати садржај дате табеле на следећи начин: ред (заглавље) са називима сваке колоне, а затим следе редови табеле. Може се одредити број редова који ће се приказивати или се могу видети сви редови у табели обележавањем опције Show all. Може се задати филтер помоћу кога ће се приказати само они редови у којима се појављује одређена вредност. На пример, могу се тражити само они ученици који се презивају Петровић. Тада би било потребно да се означи табела Ucenik, а затим да се у поље Filter rows картице Browse унесе вредност „Petrović”. Наравно, не морају се излистати вредности свих атрибута табеле, већ се могу приказати само оне које желимо. Уколико, на пример, желимо да прикажемо само имена и презимена ученика, то може да се уради тако ће се у левом крају заглавља кликнути на стрелицу надоле и означити атрибути Ime и Prezime.

Списак свих табела у жељеној бази података може се видети на левој страни прозора апликације, када се разграна стабло за дату базу података. Уколико желимо да видимо структуру табеле, то се може урадити одабиром табеле на левој страни прозора, а затим позиционирањем на картицу Structure. На тај начин се добија назив за сваки од атрибута, тип података, кодирање, инофрмација да ли може да има недостајућу вредност, да ли постоји подразумевана вредност, као и листа додатних опција.

SQL је програмски језик посебне намене који се користи за управљање подацима (чување, издвајање и манипулацију подацима) у релационим системима за управљање базама података. Сви релациони системи за управљање базама података користе SQL као стандардни језик база података.

Уколико желимо да користимо конзолну апликацију, садржај табеле може се добити следећом наредбом:

SELECT \* FROM <naziv\_tabele>

На пример, уколико желимо да видимо садржај табеле Ucenik, то се може урадити следећом наредбом:

SELECT \* FROM Ucenik

Уколико желимо да се прикажу само имена и презимена ученика, то се може урадити следећом наредбом:

SELECT Ime, Prezime FROM Ucenik

Списак свих табела у бази података може се добити следећом наредбом:

SHOW TABLES

Наредбом DESCRIBE може се приказати структура табеле. На пример, ако желимо да сазнамо каква је структура табеле Ucenik, то може да се уради следећом наредбом:

DESCRIBE Ucenik

Као резултат те наредбе добиће се информације о колонама те табеле. Те информације укључују: назив колоне, њен тип, информације да ли може имати недостајућу вредност, да ли је колона индексирана, као и подразумевану вредност колоне уколико је задата.

### Прављење табела

Да би могло да се ради над базом података, неопходно је да се на почетку направе све потребне табеле. Као пример ће бити направљене табелe Ucenik и Predmet.

У оквиру окружења phpMyAdmin, табела може да се направи кликом на назив базе података у којој корисник жели да направи табелу. Након тога, у оквиру секције Create table потребно је да се зада назив табеле и број колона табеле, а на крају да се кликне дугме Go.

Слика: 017\_kreiranje\_ucenik\_mysql\_1.png

Потпис: Прављење табеле Ucenik коришћењем окружења phpMyAdmin (први корак)

На тај начин отвара се дијалог и у оквиру њега је потребно да се за сваку колону задају њен назив, тип, као и низ додатних опција попут подразумеване вредности, да ли колона може да садржи недостајућу вредност и друго. За наш пример ће бити значајни следећи типови података:

* char(n) – ниске фиксне дужине *n*,
* varchar(n) - ниске променљиве дужине, максималне дужине *n*,
* smallint(n), int(n) – двобајтни односно четворобајтни означени цео број (опциони аргумент *n* означава број цифара које се приказују),
* float, double – реални бројеви у једнострукој односно двострукој прецизности приказани помоћу четири односно осам бајтова,
* date – тип података којим се представља датум (подразумевани формат приказа вредности типа датум је YYYY-MM-DD).

Након уноса детаља за све колоне, потребно је да се зада жељено кодирање и изабере алат за складиштење (изабраћемо подразумевани InnoDB). На крају је потребно да се притисне дугме Save. Нова табела ће бити излистана у листи свих табела на левој страни прозора апликације.

Слика: 018\_kreiranje\_ucenik\_mysql\_2.png

Потпис: Прављење табеле Ucenik коришћењем окружења phpMyAdmin (други корак)

У оквиру конзолног окружења, табела може да се направи коришћењем наредбе CREATE TABLE након које иде назив табеле која се прави, а затим унутар малих заграда следе дефиниције колона табеле међусобно раздвојене запетама и, опционо, дефиниције примарног кључа, индекса и страних кључева. Између кључне речи CREATE TABLE и назива табеле може се наћи конструкт IF NOT EXISTS који спречава појаву грешке ако већ постоји табела с истим називом. Дефиниција колоне састоји се од назива колоне, типа података у који ће бити смештене вредности те колоне и списка додатних опција. Пример једне од могућих додатних опција за колону јесте NOT NULL. Та опција означава да колона не може имати недостајућу вредност. Остале могуће опције разматраћемо накнадно.

Коришћењем окружења phpMyAdmin, табели се може додати нови ред означавањем табеле и уношењем жељених података у оквиру картице Insert за сваку од колона. Поред константних вредности, могу се користити и функције одговарајућег типа. Уколико се у неко поље може унети недостајућа вредност, приказаће се поље за штиклирање које треба штиклирати ако се недостајућа вредност уноси. На крају треба притиснути дугме Go.

Слика: 019\_unos\_podataka\_mysql.png

Потпис: Унос редова у табелу Ucenik

Редови могу да се додају табели из командне линије следећом наредбом:

INSERT INTO <naziv\_tabele> <podaci>

Притом, <podaci> означавају скуп редова који може бити:

* константна табела – она се наводи коришћењем кључне речи VALUES након које следи списак редова, међусобно раздвојених запетом,
* резултат упита SELECT.

На пример, уколико желимо да у табелу Ucenik додамо два нова ученика, то се може урадити на следећи начин:

INSERT INTO Ucenik VALUES

('Marko','Marković','Milan','2605005123456','2005-05-26'),

('Petar','Petrović','Marko','1109005555444','2005-09-11')

Неки постојећи ред табеле може да се обрише тако што се на датој табели отвори картица Browse, означи се ред за брисање и притисне се опција Delete. Уколико се истовремено брише више редова, потребно је да се да се пре активирања опције Delete означе сви редови за брисање.

Редови табеле могу се обрисати и из командне линије следећом наредбом:

DELETE FROM <naziv\_tabele> [WHERE <uslov>]

Претходном наредбом биће обрисани сви редови табеле који задовољавају дати услов. Уколико се не наведе кључна реч WHERE, том наредбом се бришу сви редови табеле. У супротном, биће обрисани они редови који задовољавају дати услов. На пример, ако желимо да обришемо све ученике који се зову Марко, то можемо да постигнемо следећом наредбом:

DELETE FROM Ucenik WHERE ime = 'Marko'

Уколико желимо да обришемо све постојеће оцене свих ученика у школи, то се може извршити следећом наредбом:

DELETE FROM Ocena

Притом се не брише табела већ само њен садржај.

Табела се може уклонити тако што се означи база података са датом табелом, а затим се у оквиру картице Structure притисне опција Drop уз наведену табелу. Други начин уклањања табеле је њено означавање и избор опције Delete the table (DROP)у оквиру картице Operations. Обрисана табела више се не може повратити, због чега ће корисник пре брисања бити упитан да ли заиста жели да обрише табелу.

Табела се брише из командне линије наредбом DROP TABLE. Ако желимо да обришемо табелу Ocena, то се може постићи следећом наредбом:

DROP TABLE Ocena

Између конструкта DROP TABLE и назива табеле коју бришемо може се додати конструкт IF EXISTS којом се спречава појава грешке уколико не постоји табела са датим називом.

### Подешавање својстава атрибута

Поред одабира типа података за сваки од атрибута, могу се задати и неке од додатних опција: величина поља, формат приказа податка, подразумевана вредност, као и правило за проверу исправности унетог податка.

Приликом дефинисања табеле у оквиру окружења phpMyAdmin, свакој колони се могу задати додатне опције, од којих су најважније следеће:

* Length/Values – за ниску је њена максимална дужина, за цео број је број приказаних цифара, а за набројиви тип је списак дозвољених вредности,
* Default – подразумеванa вредност за колону,
* Collation – начин кодирања,
* Null – одређивање да ли је подржана недостајућа вредност,
* Index – одређивање да ли је потребно направити индекс по тој колони,
* A\_I – одређивање да ли да се вредност поља аутоматски повећа за 1 за сваки наредни ред табеле.

Уколико користимо командну линију за прављење табеле у MySQL-у, за сваку од колона табеле, приликом њеног дефинисања, могу се додати нека додатна својства навођењем одговарајућих конструката након њене дефиниције. Неки могући конструкти су:

* NOT NULL – означава да колона не може имати недостајућу вредност,
* DEFAULT <vrednost> – колони се задаје подразумевана вредност,
* CHECK (<naziv\_kolone> IN (<skup vrednosti>)) – вредност колоне мора бити из датог скупа дозвољених вредности,
* PRIMARY KEY – означава да та колона представља примарни кључ табеле.

На пример, приликом прављења табеле Predmet може се подесити да ниједан атрибут не може имати недостајућу вредност, да атрибут Fond мора имати једну од вредности 1, 2, 3, 4 или 5 и да је подразумевана вредност 2.

CREATE TABLE IF NOT EXISTS Predmet (

IdPredmeta int(4) COLLATE utf8\_unicode\_ci NOT NULL,

Naziv varchar(50) COLLATE utf8\_unicode\_ci NOT NULL,

Godina smallint(4) NOT NULL,

Fond smallint(4) NOT NULL DEFAULT 2 CHECK (Fond IN (1,2,3,4,5)),

PRIMARY KEY (IdPredmeta)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;

У оквиру MS Access може се задати правило како мора да изгледа податак који се уноси у неко поље. На пример, за поље JMBG текстуалног типа препоручљиво је да се подеси да мора да се уноси тачно 13 цифара (не сме бити мање од 13 карактера и сви карактери морају одговарати цифрама). За то се могу користити маске који се уносе у поље Input Mask одговарајуће колоне. Наредни резервисани карактери су наведени као пример и дата су њихова објашњења:

* 0 – корисник мора да унесе цифру,
* 9 – корисник може да унесе цифру,
* L – корисник мора да унесе слово,
* ? – корисник може да унесе слово.

У складу са конвенцијом, маска за атрибут JMBG обухватала би 13 узастопних карактера 0. На пример, маску за шифру која се састоји од две цифре за којима следе два или три слова имала би облик 00LL?.

### Примарни кључ

Као што је већ наведено, примарни кључ табеле чини скуп колона табеле који на јединствен начин одређује сваки ред те табеле и за који не постоји прави подскуп с истим својством. Требало би дефинисати примарни кључ за сваку табелу у оквиру базе података. То је битно, између осталог, да би се формирале везе између табела.

Примарни кључ унутар окружења phpMyAdmin може се задати отварањем табеле којој желимо да се дода примарни кључ и отварањем картице Structure. Потребно је да се означе колоне које улазе у састав примарног кључа, а затим да се притисне опција Primary.

Примарни кључ се може додати из командне линије на два начина: уколико се састоји само из једне колоне, може се навести конструкт PRIMARY KEY након дефиниције колоне. Ако се примарни кључ састоји од више колона, након дефиниција колона наводи се конструкт PRIMARY KEY, а након њега, унутар малих заграда, наводе се називи колона од којих се примарни кључ састоји.

### Измена постојећих табела

Постојећој табели може се променити структура тако што ће се додати или уклонити нека колона, или се може додати или уклонити примарни кључ.

Уколико је потребно да се табели дода једна или више нових колона, то се може урадити одабиром жељене табеле, позиционирањем на картицу Structure и уносом броја нових колона између речи Add и columns. Поред тога се може задати место на коме треба да се додају нове колоне: на почетку табеле, на њеном крају или након неке друге колоне. Након тога је потребно да се наведу додатне опције за све новододате колоне, на исти начин као и приликом прављења табеле.

У оквиру командне линије у MySQL-у, структура табеле може се изменити коришћењем наредбе ALTER после које се наводи листа промена раздвојених запетама. Промена може бити додавање неког објекта (нпр. колоне, примарног кључа итд.), што се означава кључном речју ADD, или уклањање неког објекта, што се означава кључном речју DROP.

На пример, ако табели Ucenik желимо да додамо колону MestoRodjenja као ниску променљиве дужине али с максималном дужином 20, то се може урадити на следећи начин:

ALTER TABLE Ucenik

ADD COLUMN MestoRodjenja varchar(20)

Колона се може уклонити одабиром опције Drop након назива колоне на картици Structure за дату табелу.

Колону MestoRodjenja можемо уклонити из табеле Ucenik следећом наредбом:

ALTER TABLE Ucenik

DROP COLUMN MestoRodjenja

Уколико желимо да табели додамо примарни кључ који се састоји из само једне колоне, довољно је да се притисне опција Primary након назива одговарајуће колоне. Ако се примарни кључ састоји из више колона, потребно је да се колоне означе, а затим да се притисне опција Primary која се налази након списка свих колона те табеле.

Примарни кључ се може додати табели из командне линије на следећи начин:

ALTER TABLE <naziv\_tabele>

ADD PRIMARY KEY (<skup kolona>)

Из командне линије може се уклонити на следећи начин:

ALTER TABLE <naziv\_tabele>

DROP PRIMARY KEY

На пример, уколико желимо да променимо примарни кључ табеле Predmet и да поставимо назив предмета за примарни кључ, то се може урадити тако што се најпре обрише постојећи примарни кључ, а затим се на следећи начин дода нови примарни кључ:

ALTER TABLE Predmet

DROP PRIMARY KEY,

ADD PRIMARY KEY (naziv)

### Индекси

Индекси у базама података настали су на основу индекса који се користи у књигама: претрага према појмовима који се појављују у књизи омогућава брже проналажење појединих страна. На пример, претпоставимо да у овом уџбенику желимо да пронађемо део у којем се говори о погледима. Пошто уџбеник не садржи индекс, морали бисмо да листамо књигу све док не дођемо до појма који нас интересује. Јасно је да таква секвенцијална претрага у већини случајева није задовољавајућа. Када би, пак, постојао индекс на крају уџбеника, најпре бисмо у индексу потражили одредницу „погледи”, а затим бисмо према бројевима страна које стоје уз одредницу пронашли дати појам. У базама података индекси табела су физички представљени датотеком чије странице образују специфичну структуру (тзв. B-стабло) које омогућује ефикасни несеквенцијални приступ врсти табеле према вредностима оних атрибута над којима је изграђен индекс.

Над којим атрибутима табеле треба направити индексе? Ако се табела често претражује или сортира према неком одређеном атрибуту, добра идеја би била да се направи индекс за тај атрибут. На тај начин се убрзавају те операције. Наравно, са сваким уносом у табелу или брисањем из табеле неопходно је да се ажурирају сви индекси табеле, па ће се све операције знатно успорити када постоји велики број индекса.

Такође се може направити индекс над већим бројем атрибута. На пример, за табелу Ucenik потребно је да се омогући брзо проналажење особе са датим именом и презименом. Због тога би било корисно да се сви ученици сортирају према презимену, а затим према имену у оквиру истог презимена.

Зашто је значајно уређење по колонама у индексима? Када се резултат сортира на посебан начин, уређење може да буде од значаја. На пример, ако се жели сортирање резулата растуће према имену, а затим растуће према презимену, док је код индекса уређење растуће према имену, али опадајуће према презимену, тада база података мора да изврши додатно сортирање. То је значајно за индексе који садрже више од једне колоне, јер код индекса који садрже само једну колону промена уређења доводи само до обрнутог уређења.

Индекс се може направити на следећи начин: у оквиру картице Structure означе се колоне табеле које улазе у састав индекса, а затим се притисне опција Index. Такође, у оквиру секције Indexes може се направити индекс кликом на дугме Go поред текста Create an index. Потребно је да се из падајуће листе изабере број колона од којих се састоји индекс. Након тога, отвара се дијалог Add Index у коме је потребно да се унесу назив индекса и нека додатна опција за индекс, а затим треба одабрати колоне од којих ће се индекс састојати.

Индекс се може направити из командне линије на следећи начин:

CREATE [UNIQUE] INDEX <naziv\_indeksa> ON <naziv\_tabele>(<spisak kolona>)

Кључна реч UNIQUE означава да све вредности у индексираним колонама морају бити различите. Списак колона се састоји из назива колона раздвојених запетама, при чему сваки од назива може да прати нека од ознака уређења:

* ASC – означава растуће уређење и подразумева се уколико се не наведе уређење,
* DESC – означава опадајуће уређење.

На пример, ако желимо да направимо индекс на табелу Ucenik који се састоји од колона Ime и Prezime, при чему је уређење по имену растуће а по презимену опадајуће, то се може урадити на следећи начин:

CREATE INDEX ind\_ime\_i\_prezime ON Ucenik(ime ASC, prezime DESC)

## Везе између табела

При логичком пројектовању база података, подаци се смештају у различите табеле на основу логичких веза које важе међу атрибутима. Притом треба водити рачуна да се минимизује њихово понављање. На пример, уколико не би постојала табела Ucenik него би се у табели Ocena налазили сви подаци о ученику (као што су матични број, име и презиме), не би се могли унети подаци о ученику док се не унесу подаци о барем једној оцени. Такође, брисањем неке оцене изгубиле би се информације и о самом ученику. Приликом измене неког податка о ученику (на пример, презимена), то би се морало урадити у свим редовима табеле који се односе на његове оцене.

Због тога је неопходно да се након прављења табела код којих се не јављају наведене неправилности обезбеди механизам за повезивање одговарајућих података када је то потребно. На пример, како може да се уз свако име и презиме ученика прикажу његове оцене? У табели Ocena постоји информација о томе који ученик има које оцене, али су у њој ученици идентификовани својим матичним бројем, а не именом и презименом. С друге стране, подаци о имену и презимену ученика налазе се у табели Ucenik. Да бисмо објединили податке о имену и презимену ученика са његовим оценама, потребно је да се повежу редови табеле Ocena са редовима табеле Ucenik тако да се односе на истог ученика. Из тог разлога је потребно да се дефинишу везе (енгл. relationship) између табела. Неопходно је да поља табела која се повезују буду истог типа, док им се називи могу разликовати. У нашем случају потребно је повезати атрибут JMBG табеле Ucenik са атрибутом JMBGUcenika табеле Ocena чиме ће се обезбедити да се односе на истог ученика.

Између табела постоје следећа три типа веза:

* један према више (енгл. one-to-many) – посматрајмо табелу Ocena и табелу Ucenik. Један ученик може имати више оцена, док се свака оцена односи на само једног ученика. Стога је тип везе између ове две табеле „један према више”. Овај тип везе се реализује у дизајну базе тако што се колоне које припадају примарном кључу прве табеле додају другој табели. У примеру који смо видели, атрибут са вредношћу ЈМБГ потребно је додати табели Ocena.
* више према више (енгл. many-to-many) – овај тип везе може се илустровати на примеру табела Predmet и Profesor. Наиме, један професор може да предаје више различитих предмета (на пример, математику и физику), а такође један исти предмет може предавати више професора (различитим одељењима). Овај тип везе назива се „више према више” и реализује се увођењем треће везне табеле која разбија ову везу на две везе типа „један према више“. У случају табела Predmet и Profesor, то би била табела Predaje и њој се додају примарни кључеви обе табеле, као и потребни додатни атрибути.
* један према један (енгл. one-to-one) – овај тип везе ретко се користи јер ако један ред једне табеле одговара највише једном реду друге табеле и обратно, ти подаци се најчешће чувају заједно у истој табели. Овај тип везе се некада користи да би се из безбедоносних разлога поједине личне информације сакриле од неке групе корисника у другу табелу. На пример, могли бисмо неке приватне информације о ученику (адресу, број телефона и држављанство) заједно са атрибутом JMBG сместити у нову табелу UcenikPrivPodaci. Тада би било потребно да се табеле Ucenik и UcenikPrivPodaci повежу преко атрибута JMBG и то би онда била веза типа „један према један”.

### Формирање веза између табела

Да би се направиле везе између табела једне базе података, потребно је кликнути на ту базу података, а затим одабрати картицу Designer (уколико се не приказује, потребно је кликнути на More). Након тога, требало би да се у оквиру прозора прикажу табеле те базе података. Уколико се прикаже само назив табеле а не и скуп атрибута те табеле, потребно је да се кликне на стрелицу у горњем левом углу табеле. Табеле се могу померати унутар датог прозора ради проналажења њиховог погодног положаја. Након тога се може снимити задати положај табела притиском на дугме Save position на линији с алаткама.

Уколико се притисне дугме Create relation са линије с алаткама, појавиће се порука Select referenced key и тада треба притиснути примарни кључ једне стране у вези. Затим ће се појавити порука Select the foreign key која одговара страни „више” у вези. Након тога се приказује дијалог у коме је потребно одабрати жељене акције које се врше приликом брисања реда табеле која одговара страни „један” у вези, као и приликом ажурирања неке вредности која улази у састав примарног кључа табеле која одговара страни „један” у вези и притиснути OK. Тиме би требало да се појави линија која одговара формираној вези између ове две табеле.

### Референцијални интегритет

Референцијални интегритет (енгл. referential integrity) представља скуп услова који се намећу приликом прављења базе података и његов смисао лежи у томе да у сваком моменту подаци у бази података морају бити коректно повезани, тј. да не постоји веза постојећих података на непостојеће. На пример, у бази електронског дневника референцијални интегритет био би нарушен ако би у табели Ocena постојао ред који се односи на неког ученика који се не налази у табели Ucenik. Референцијални интегритет се реализује страним кључевима.

Услови референцијалног интегритета проверавају се након сваке измене у бази. На пример, уколико је потребно да се обрише неки ученик из табеле Ucenik, није добро да његове оцене остану у табели Ocena јер се не би знало на кога се те оцене односе. Потребно је применити један од следећих поступака:

* забранити такво брисање,
* обрисати и одговарајуће редове табеле Ocena (табеле над којом је дефинисан страни кључ),
* поставити одговарајуће вредности табеле над којом је дефинисан страни кључ на NULL, уколико је то дозвољено.

Слично томе, уколико је потребно да се промени ЈМБГ неког ученика у табели Ucenik, није добро да вредност ЈМБГ тог ученика остане непромењена у табели Ocena јер би се редови са том вредношћу односили на непостојећег ученика.

Приликом формирања веза у оквиру окружења phpMyAdmin кориснику се приказује дијалог у коме је потребно да изабере акцију приликом брисања реда у табели на коју се односи страни кључ друге табеле, као и приликом ажурирања неке вредности. Следеће вредности су дозвољене:

* CASCADE – приликом брисања реда табеле на коју се односи страни кључ друге табеле бришу се и сви редови табеле над којом је дефинисан страни кључ,
* SET NULL – приликом брисања реда табеле на коју се односи страни кључ друге табеле поставља се вредност NULL на скупу атрибута страног кључа уколико је то дозвољено (колоне нису дефинисане као NOT NULL),
* RESTRICT – дозвољено је брисање само оног реда табеле на коју се односи страни кључ друге табеле за који не постоји одговарајућа вредност у табели над којом је дефинисан страни кључ, а у супротном систем пријављује да је дошло до грешке,
* NO ACTION – у програму MySQL ово правило је исто као и RESTRICT.

Задавање страних кључева табелама представља механизам којим се намећу услови референцијалног интегритета. Дефиниција страног кључа има следећи облик:

[CONSTRAINT <naziv\_stranog\_kljuca>] FOREIGN KEY (<lista\_kolona\_odnosne>)

REFERENCES <bazna\_tabela> (<lista\_kolona\_bazne>)

[ON DELETE <pravilo>] [ON UPDATE <pravilo>]

У зависности од типа измене у програму MySQL постоји неколико правила.

* Уос новог реда у табелу над којом је дефинисан страни кључ – ако вредност страног кључа унетог реда није NULL, онда она мора да одговара вредности примарног кључа неког реда из табеле на коју се односи страни кључ.
* Брисање реда из табеле на коју се односи страни кључ друге табеле – подржана су следећа правила:
  + RESTRICT – дозвољено је брисање само оног реда табеле за који не постоји одговарајућа вредност у табели над којом је дефинисан страни кључ; у супротном систем пријављује да је дошло до грешке,
  + SET NULL – приликом брисања реда у табели поставља се NULL вредност на скупу атрибута страног кључа уколико је то дозвољено (колоне нису дефинисане као NOT NULL),
  + CASCADE – приликом брисања реда табеле бришу се и сви редови табеле над којом је дефинисан страни кључ.
* Измена постојећег реда табеле на коју се односи страни кључ друге табеле – подржана су следећа правила:
  + RESTRICT – није дозвољено мењати вредности атрибута који улазе у примарни кључ табеле уколико се та вредност јавља у табели над којом је дефинисан страни кључ,
  + SET NULL – приликом ажурирања вредности примарног кључа у табели поставља се NULL вредност на одговарајућим атрибутима страног кључа уколико је то дозвољено (колоне нису дефинисане као NOT NULL),
  + CASCADE – аутоматски се ажурирају и одговарајуће вредности у табелама над којима је дефинисан страни кључ,
  + RESTRICT је подразумевано правило и за брисање и за ажурирање.

Није неопходно да се именује страни кључ, али је добра пракса да постоји могућност да се уклони ако је то потребно.

На пример, уколико желимо да приликом дефинисања табеле Predaje додамо именоване стране кључеве на табелу Predmet и табелу Profesor са правилом CASCADE и приликом брисања и приликом ажурирања, то се може урадити на следећи начин:

create table if not exists Predaje (

IdProfesora int(4) NOT NULL,

IdPredmeta int(4) NOT NULL,

SkolskaGod int(4) NOT NULL,

PRIMARY KEY (IdProfesora, IdPredmeta),

KEY IdProfesora (IdProfesora),

KEY IdPredmeta (IdPredmeta),

CONSTRAINT fkprof FOREIGN KEY (IdProfesora) REFERENCES Profesor (IdProfesora) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT fkpred FOREIGN KEY (IdPredmeta) REFERENCES Predmet (IdPredmeta) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

За прављење страног кључа неопходно је постојање одговарајућег индекса у табели над којом се дефинише страни кључ, како би провере вредности страног кључа биле брзе. Из тог разлога је табели Predaje додат индекс на колону IdPredmeta и индекс на колону IdProfesora.

### Измена веза између табела

Табели се може накнадно додати страни кључ наредбом ALTER. На пример, табели Ocena може се додати страни кључ на табелу Ucenik на следећи начин:

ALTER TABLE Ocena

ADD CONSTRAINT fkou FOREIGN KEY (JMBGUcenika) REFERENCES Ucenik (JMBG)

ON DELETE CASCADE ON UPDATE CASCADE

Наредбом ALTER може се и уклонити страни кључ. Додати страни кључ може се обрисати на следећи начин:

ALTER TABLE Ocena

DROP FOREIGN KEY fkou

Ако желимо да за постојећи страни кључ само променимо правило које се примењује приликом брисања или ажурирања, то се мора урадити у два корака: најпре се постојећи страни кључ обрише, а затим се дода страни кључ идентичан претходном али са жељеним правилом за брисање или ажурирање. На пример, ако је табели Ocena додат страни кључ fkou на табелу Ucenik са правилом CASCADE и при брисању и при ажурирању, а желимо да правило при брисању променимо у SET NULL, то се може урадити на следећи начин:

ALTER TABLE Ocena

DROP FOREIGN KEY fkou,

ADD CONSTRAINT fkou2 FOREIGN KEY (JMBGUcenika)

REFERENCES Ucenik (JMBG)

ON DELETE SET NULL ON UPDATE CASCADE;

Подразумевани тип спајања табела јесте унутрашње спајање (енгл. INNER JOIN) које обухвата спајање две табеле по заједничким редовима. На пример, уколико спајамо табелу Ucenik са табелом Ocena по услову да је исти ЈМБГ, у ситуацији када постоји неки ученик без иједне оцене, он се неће појавити у резултату. Некада је корисно да се у резултату добију сви редови једне табеле, чак и ако не постоји ред друге табеле са којим се може упарити. У том случају, за неупарене редове добија се вредност NULL на осталим колонама друге табеле. Овај тип спајања назива се спољашње спајање (енгл. OUTER JOIN), при чему се разликују следеће три врсте:

* лево спољашње спајање (енгл. LEFT OUTER JOIN) – када се у резултату добијају сви редови леве табеле и упарени редови десне табеле,
* десно спољашње спајање (енгл. RIGHT OUTER JOIN) – када се у резултату добијају сви редови десне табеле и упарени редови леве табеле,
* пуно спољашње спајање (енгл. FULL OUTER JOIN) – када се у резултату добијају сви редови леве табеле и сви редови десне табеле.

У програму Access, тип спајања табела за неку везу може се изабрати кликом на ту везу, а затим у дијалогу Edit Relationships треба кликнути на дугме Join Type. Прва опција подразумева унутрашње спајање, друга опција лево спољашње спајање, а трећа опција десно спољашње спајање. Пуно спољашње спајање не може се добити без употребе упита. У програму MySQL тип спајања табела може се одабрати приликом формулисања упита.

## 1.5 Упити

У претходним поглављима упознали смо се прављењем појединих објеката базе података – табела, индекса, форми и слично. Након прављења табела у оквиру базе података и њиховог попуњавања одређеним садржајем, често је потребно да се испитају и анализирају вредности у тим табелама.

Као што је већ поменуто, под упитом се подразумева захтев за претраживањем и сакупљањем података из једне или више табела. Упити представљају ефикаснији алат од једноставне претраге и филтрирања података из табеле јер се упитима могу издвајати подаци из више табела. На пример, могли бисмо да претражујемо табелу Ucenik како би пронашли ученике који се зову Марко. Такође, могли бисмо да применимо филтер за издвајање само оних оцена које су ученици добили данас, али ни претрага нити филтрирање података не би нам омогућили да у исто време видимо све оцене које су ученици с именом Марко добили данас. Зато се једноставна претрага и филтрирање могу третирати као специјални случај упита, па се могу и изразити коришћењем упита.

Упити се могу користити у различите сврхе: за добијање одговора на неко питање, за извођење рачунских операција, за комбиновање података из више табела итд. Такође, упити се користе за прикупљање података које ћемо представити у виду форме или извештаја.

Постављање упита је најчешћа операција над базом података. Резултат упита се представља табеларно. Најпопуларнији упитни језик је SQL и подржава га већина окружења база података. За постављање упита у језику SQL користи се наредба SELECT и њом се „дохватају” подаци из једне или више међусобно повезаних табела. Овом наредбом се задају услови које резултат упита треба да задовољи, а сам СУБП је задужен да те услове преведе у појединачне операције које су потребне за добијање тражених резултата. Наредбом SELECT никако се не могу изменити подаци у бази података.

За сваки упит може се задати:

* које податке желимо да издвојимо, тј листу колона које ће бити укључене у резултат упита,
* из којих табела треба издвојити податке,
* који скуп услова треба да задовоље подаци,
* у ком поретку треба издвојити добијене податке.

Синтакса наредбе SELECT у SQL-у је следећа:

SELECT <lista\_kolona>

FROM <spisak\_tabela>

[WHERE <skup\_uslova>]

[ORDER BY <naziv\_kolone [uredjenje]>|{ <naziv\_kolone [uredjenje]>}]

У оквиру ове наредбе у угластим заградама задато је опционо појављивање фрагмената наредбе, а у витичастим заградама део наредбе који се не јавља ниједном или се јавља више пута. Упити се могу писати у једном или више редова.

Поред захтева за претраживањем, често се јављају и захтеви за ажурирањем података које називамо радњама. Њима се могу додавати, мењати или брисати подаци.

И oкружењe phpMyAdmin подржава SQL. У овом окружењу упит се може задати у оквиру картице SQL, уносом SQL упита у одговарајуће поље.

За сваки од наредних упита даћемо резултат добијен над попуњавањем базе података која је дата у почетном поглављу.

Упитни језици су у значајној мери декларативни, тј. њима се задаје оно шта желимо да добијемо, без прецизног навођења како треба да се добију жељени подаци. СУБП је тај који одређује како се долази до тих података.

### Упити над једном табелом

Jеднорелациони упити представљају SQL облик задавања претраге односно филтрирања података. Једине обавезне ставке у оквиру наредбе SELECT јесу ставке SELECT и FROM, тј. неопходно је да се задају подаци који се издвајају и из којих се табела издвајају. Након кључне речи WHERE може се навести скуп услова које треба да задовољавају подаци, тј. биће издвојени само они редови резултујуће табеле за које скуп услова има логичку вредност „тачно”. Жељени редослед приказа података наводимо након кључне речи ORDER BY, а притом се може навести већи број колона и уз сваку од њих смер уређења.

Некада не желимо да прикажемо све колоне дате табеле, већ нас интересују само неке од њих. На пример, уколико желимо да издвојимо само имена и презимена свих ученика, то се може урадити SQL упитом:

SELECT Ime, Prezime

FROM Ucenik

Приликом покретања овог упита у конзолном окружењу у резултату упита добили бисмо следеће редове:

Слика: 071\_mysql\_upit\_select\_rez.png

Потпис: Резултат упита задатог из командне линије

Резултат одговарајућег упита формулисаног у окружењу Access 2010 приказује се у виду табеле на следећи начин:

Слика: 072\_access2010\_upit\_select\_rez.png

Потпис: Резултат упита формулисаног у оквиру окружења Access

Надаље ћемо резултате упита приказивати у окружењу Access.

Редослед колона у резултујућој табели одговара редоследу навођења колона у оквиру ставке SELECT. У оквиру SQL-a не прави се разлика између малих и великих слова, али ће у наставку текста кључне речи SQL-a бити писане великим словима због прегледности.

Уколико се издвајају све колоне неке табеле, потребно је да се наведу називи свих колона у оквиру ставке SELECT:

SELECT Ime, Prezime, ImeOca, JMBG, DatRodjenja

FROM Ucenik

Други начин је навођење симбола \* иза кључне речи SELECT.

SELECT \*

FROM Ucenik

Тиме би се у резултату упита добили следећи редови:

Слика: 073\_upit\_select\_zvezda\_rez.png

Потпис: Резултат упита

У наставку ћемо видети како се задаје упит над табелом. Најпре је потребно позиционирање на ту табелу. На пример, ако желимо само да излистамо имена и презимена свих ученика, у левом углу прозора позиционирали бисмо се на табелу Ucenik, а затим бисмо у оквиру поља за унос текста у оквиру картице SQL унeли текст упита и притиснули дугме Go.

Списак колона у табели налази се у десном углу прозора у оквиру листе Columns. Неку од колона са тог списка можемо уметнути у упит двоструким кликом на назив те колоне. Такође, на дну прозора дате су наредбе које се могу извршити: SELECT, INSERT, UPDATE, DELETE итд. Кликом на неко од наведених дугмади аутоматски се генерише костур те наредбе. Постоји и дугме Format којим се написани упит форматира тако да буде преломљен у више редова.

Приликом писања упита подржано је аутоматско допуњавање преосталог текста у речи (енгл. autocomplete). Такође, уколико упит садржи грешку, линије са грешком биће означене црвеним кружићем. Кликом на тај кружић може се добити пропратна порука.

Кликом на одговарајуће наредбе и имена колона генерише се наредни упит:

SELECT 'Ime', 'Prezime'

FROM 'Ucenik'

Као што можемо да видимо, овде се сва имена (табела, колона и сл.) уносе као низови карактера, под једноструким наводницима. Иначе, имена у оквиру SQL упита могу се наводити и без наводника.

Слика: 079\_query\_phpmyadmin.png

Потпис: Прављење упита у оквиру окружења phpMyAdmin

Резултат упита приказује се табеларно у оквиру централног дела прозора. На врху прозора приказује се упит који је извршен, а након тога резултат упита у табеларној форми. На дну прозора постоје различите могућности за штампу резултата упита, извоз у неки други формат, приказ графика (ако су у питању нумеричке вредности), као и могућност да се направи поглед са тим садржајем. Упит се, такође, може обележити (енгл. bookmark).

Слика: 080\_query\_phpmyadmin\_rez.png

Потпис: Приказ резултата упита у оквиру окружења phpMyAdmin

Упит се може изменити командом Edit којом се враћамо на унос упита у оквиру картице SQL или командом Edit inline којом се допушта измена упита у оквиру стране са резултатима. Након измене довољно је да се притисне дугме Go како би се видели измењени редови у резултату упита.

Уколико желимо да прикажемо све колоне табеле, можемо изабрати наредбу SELECT \* из листе наредби.

Само имена очева свих ученика могу се издвојити SQL упитом.

SELECT ImeOca

FROM Ucenik

Слика: 081\_upit\_select\_imeoca.png

Потпис: Резултат упита који садржи поновљене вредности

Можемо приметити да се у резултату упита нека имена јављају више пута – то је зато што неки ученици имају исто име оца. Понављања у резултатима упита елиминишу се навођењем кључне речи DISTINCT испред листе колона која се издваја.

SELECT DISTINCT ImeOca

FROM Ucenik

Слика: 082\_upit\_select\_distinct\_imeoca.png

Потпис: Резултат упита којим се елиминишу поновљене вредности

Поред назива колона, у оквиру SELECT ставке могу се наћи и скаларни изрази (константне вредности или рачунски изрази у којима се као аргумент јавља колона). На пример, за сваки месец могу да се издвоје његов назив и месечни фонд, под претпоставком да месец има четири радне недеље.

SELECT Naziv, 'Mesecni Fond: ', Fond\*4

FROM Predmet

Овим упитом се, поред колоне Naziv, издваја и једна константна колона (која има вредност Mesecni Fond у свакој врсти резултата), као и колона добијена као резултат рачунског израза.

Слика: 083\_upit\_select\_izrazi.png

Потпис: Резултат упита који садржи скаларне изразе

Може се приметити да су називи друге и треће колоне редом 2 и 3. То је зато што су у питању колоне које су настале као резултат рачунања неког израза, па су системски добиле назив редног броја колоне у упиту. Уколико желимо да овим колонама доделимо назив, односно да некој колони променимо назив, то се може урадити навођењем новог назива непосредно након израза који желимо именовати, односно након постојећег назива колоне коју желимо да преименијемо. Новоуведеном имену може да опционо претходи кључна реч AS.

SELECT Naziv, 'Mesecni Fond: ' Tekst, Fond\*4 MesecniFond

FROM Predmet

Или се може представити на следећи начин:

SELECT Naziv, 'Mesecni Fond: ' AS Tekst, Fond\*4 AS MesecniFond

FROM Predmet

Слика: 084\_upit\_select\_izrazi.png

Потпис: Резултат упита који садржи скаларне изразе и којим се именују колоне

У оквиру линије за издвајање података могу се задати рачунски изрази, као и константни низови карактера под једноструким наводницима. Уколико не именујемо колоне у резултату упита, оне ће бити именоване у резултату изразом по коме се рачунају. Ако желимо да их именујемо, ново име можемо навести у оквиру SQL упита непосредно након тог израза или константног низа карактера.

Некада нас не занимају сви редови табеле, већ само они који задовољавају неки скуп услова. Ставка WHERE омогућава задавање услова које редови резултата упита треба да задовољавају. На пример, имена и презимена само оних ученика чије је име оца Марко могу се издвојити на следећи начин:

SELECT Ime, Prezime

FROM Ucenik

WHERE ImeOca = 'Marko'

Слика: 086\_upit\_where.png

Потпис: Резултат упита који садржи рестрикцију

У оквиру картице SQL подразумевани упит садржи услов WHERE 1 који је увек тачан, па се приказују сви редови табеле. Ако желимо да издвојимо само оне редове који задовољавају неки услов, то се ради на претходно поменути начин у оквиру SQL упита.

У оквиру ставке WHERE могу се јавити стандардни релациони оператори (=, <>, <, >, <=, >=) и стандардни логички оператори (AND за конјункцију услова, OR за дисјункцију услова, NOT за негацију). На пример, уколико желимо да издвојимо само информације о школским предметима који се не изучавају у првом разреду и чији је фонд барем три часа недељно, то бисмо могли да урадимо следећим упитом:

SELECT \*

FROM Predmet

WHERE Godina<>1 AND Fond>=3

То се може урадити и на следећи начин:

SELECT \*

FROM Predmet

WHERE NOT(Godina=1) AND Fond>=3

У резултату ових упита добили бисмо наредне редове:

Слика: 088\_upit\_select\_log\_rel\_op\_rez.png

Потпис: Резултат упита који укључује логичке и релационе операторе

Поред поменутих оператора, у оквиру логичког израза у WHERE се могу пронаћи и следећи релациони оператори:

* IN (NOT IN)
* BETWEEN (NOT BETWEEN)

Оператор IN проверава да ли се вредност израза налази у наведеној листи вредности. Његова синтакса је следећа:

<izraz> IN (<lista vrednosti razdvojena zarezima>)

На пример, ако је потребно да се прикажу све информације о предметима чији је фонд два, три или четири часа недељно, то се може урадити следећим упитом:

SELECT \*

FROM Predmet

WHERE Fond IN (2,3,4)

Oво је еквивалентно следећем упиту:

SELECT \*

FROM Predmet

WHERE Fond=2 OR Fond=3 OR Fond=4

У комбинацији с оператором NOT добија се предикат који му је супротан. На пример, уколико је потребно да се прикажу све информације о предметима чији фонд није ни два, ни три, нити четири часа, то се може добити следећим упитом:

SELECT \*

FROM Predmet

WHERE Fond NOT IN (2,3,4)

Њему је еквивалентан следећи упит:

SELECT \*

FROM Predmet

WHERE NOT (Fond IN (2,3,4))

Такође је и следећи упит еквивалентан:

SELECT \*

FROM Predmet

WHERE Fond<>2 AND Fond<>3 AND Fond<>4

Оператор BETWEEN проверава да ли се вредност израза налази у задатом опсегу. Његова синтакса је следећа:

<izraz> BETWEEN <pocetna\_vrednost> AND <krajnja\_vrednost>

Притом су укључене и граничне вредности. На пример, ако је потребно да се прикажу све информације о предметима чији је фонд између два и четири часа недељно, укључујући и те вредности, то може да се уради на следећи начин:

SELECT \*

FROM Predmet

WHERE Fond BETWEEN 2 AND 4

Oво је еквивалентно следећем упиту:

SELECT \*

FROM Predmet

WHERE Fond>=2 AND Fond<=4

У комбинацији с оператором NOT добија се предикат NOT BETWEEN који му је супротан. На пример, ако је потребно да се прикажу све информације о предметима чији се фонд не налази у опсегу од два до четири, то се може добити упитом:

SELECT \*

FROM Predmet

WHERE Fond NOT BETWEEN 2 AND 4

Њему је еквивалентан следећи упит:

SELECT \*

FROM Predmet

WHERE NOT (Fond BETWEEN 2 AND 4)

Такође је и следећи упит еквивалентан:

SELECT \*

FROM Predmet

WHERE Fond<2 OR Fond>4

Оператор LIKE служи за поређење низа карактера са задатом маском. Притом се прави разлика између малих и великих слова. Синтакса овог оператора изгледа на следећи начин:

<izraz> LIKE <maska>

Маска се задаје као константни низ карактера. У оквиру маске симбол ' ' означава појављивање тачно једног произвољног знака, а симбол % означава појаву нула или више произвољних знакова. Уколико је потребно да се у оквиру маске наведу баш ови симболи, неопходно је да се испред њих наведе тзв. прекидачки симбол. Његова подразумевана вредност је \, а ако из неког разлога желимо да га променимо, потребно је да се на крају овог оператора, иза кључне речи ESCAPE, наведе жељени прекидачки симбол. На пример, уколико желимо да издвојимо имена ученика чије се име завршава словом „а”, то можемо да учинимо следећим упитом:

SELECT Ime

FROM Ucenik

WHERE Ime LIKE '%a'

Као резултат упита добили бисмо наредне редове у резултату:

Слика: 091\_upit\_maska\_like.png

Потпис: Резултат упита којим се издвајају ниске које се завршавају на дато слово

Уколико је потребно да име садржи слово „a”, одговарајућа маска била би %a%.

Kомбинацијом са оператором NOT добија се супротан оператор. На пример, ако желимо да издвојимо имена ученика која немају на другом месту слово „a”, то се може добити следећим упитом:

SELECT Ime

FROM Ucenik

WHERE Ime NOT LIKE '\_a%'

Тиме бисмо у резултату добили наредне редове:

Слика: 092\_upit\_maska\_not\_like.png

Потпис: Резултат упита којим се издвајају ниске које се не завршавају на дато слово

Ако бисмо, на пример, хтели да добијемо сва имена ученика чије име садржи симбол % на првој позицији, а завршава се са \, то се може добити поређењем имена ученика са маском \%%\\.

Оператор IS NULL проверава да ли је дата вредност недостајућа. Комбинацијом са оператором NOT добија се супротан предикат IS NOT NULL. На пример, ако желимо да прикажемо све податке о ученицима чије име оца није познато, то се може урадити следећим упитом:

SELECT \* FROM Ucenik

WHERE ImeOca IS NULL

Овим упитом издвојили бисмо наредне редове:

Слика: 094\_upit\_null.png

Потпис: Резултат упита којим се издвајају редови код којих је вредност ниске недостајућа

Наредним упитом приказују се подаци о ученицима чије је име оца познато:

SELECT \* FROM Ucenik

WHERE ImeOca IS NOT NULL

У резултату би се добили следећи редови:

Слика: 095\_upit\_not\_nul1.png

Потпис: Резултат упита којим се издвајају редови код којих вредност ниске није недостајућа

Резултат упита може се уредити према једној или више колона, у растућем или опадајућем редоследу. То се постиже коришћењем ставке ORDER BY на крају упита, за којом следи списак колона по којима се резултат упита уређује. За сваку колону може се задати уређење навођењем кључне речи ASC за растуће уређење а DESC за опадајуће. Подразумевано уређење је растуће, због чега није неопходно да се наводи експлицитно. На пример, желимо да прикажемо имена и презимена свих ученика уређених растуће према презимену, а затим растуће према имену (уколико имају исто презиме) – овај редослед одговара редоследу ученика у дневнику. Ово бисмо могли да постигнемо следећим упитима:

SELECT Ime,Prezime

FROM Ucenik

ORDER BY Prezime Asc, Ime Asc

SELECT Ime,Prezime

FROM Ucenik

ORDER BY Prezime, Ime

На тај начин бисмо добили следеће уређење редова табеле Ucenik:

Слика: 096\_upit\_order\_by.png

Потпис: Резултат упита којим се издвајају редови табеле Ucenik уређени према презимену растуће, па према имену растуће

Понекад је потребно да се изврши упит који је једноставна модификација постојећег упита: на пример, имали смо упит којим се издвајају имена и презимена ученика чије је име оца Марко. Уколико бисмо сад желели да издвојимо оне ученике чије је име оца Петар, могли бисмо да модификујемо претходни упит. Ипак, уколико су нам често потребне варијације истог упита, корисно је да се размотри идеја параметарских упита у Access-у. Приликом покретања параметарског упита корисник од корисника се тражи да унесе вредност неког поља и на основу критеријума за унесену вредност упит се извршава. На пример, у претходном примеру било би корисно да се име оца прогласи за параметар, а у моменту извршавања упита да се унесе вредност за име оца. На тај начин овакав упит би могао да се користи за произвољно име оца ученика. Параметри се уносе у реду Criteria у оквиру угластих заграда. Тачније речено, у оквиру угластих заграда потребно је да се напише питање које ће бити постављено кориснику, на пример, „унесите име оца ученика”.

Слика: 098\_access2010\_query\_design\_parametric\_criteria.png

Потпис: Прављење упита који укључује параметарске критеријуме

### Упити над више табела

Уколико желимо да издвојимо информације које се налазе у различитим табелама, потребно је да формулишемо упит над више табела. На пример, ако желимо да издвојимо имена и презимена ученика који су из неког предмета добили оцену 5, јасно је да се информације о имену и презимену ученика налазе у табели Ucenik, а информације о добијеној оцени у табели Ocena. Дакле, потребно је да се издвоје тражени подаци из ове две табеле, тако да спојимо сваки ред табеле Ocena са редом табеле Ucenik који одговара том ученику, односно по услову да je вредност атрибута JMBGUcenika табеле Ocena иста као и вредност атрибута JMBG табеле Ucenik. Наредба за спајање табела у SQL-у је JOIN и њена синтакса је следећа:

<tabela1> JOIN <tabela2> ON <uslov\_spajanja>

Ова конструкција може да се нађе у оквиру ставке FROM наредбе SELECT. Уколико желимо да издвојимо имена и презимена ученика који су добили оцену 5 из неког предмета, то се може урадити следећим упитом:

SELECT Ime, Prezime

FROM Ucenik JOIN Ocena

ON Ucenik.JMBG = Ocena.JMBGUcenika

WHERE Ocena.Ocena = 5

Овим упитом издвојили бисмо следећа имена и презимена ученика:

Слика: 099\_upit\_spajanje\_dve\_tabele.png

Потпис: Резултат упита који укључује спајање две табеле

Можемо уочити да су у овом упиту називи колона квалификовани називом табеле из које потичу. То је потребно да се ради увек када постоји атрибут с истим називом у табелама које спајамо у оквиру упита. Уколико је назив атрибута јединствен, није неопходно да се квалификује називом табеле (али није ни грешка да се то уради, као што смо ми урадили у претходном упиту). Некада је заморно да се више пута пише назив табеле, нарочито ако је дугачак. Због тога се често прибегава прављењу алијаса – надимка за табелу. Табела може да се преименује навођењем новог назива непосредно након назива табеле у оквиру линије FROM. Тако, на пример, у претходном упиту могу да се уведу краћа имена за табеле: за табелу Ucenik – U, а за табелу Ocena – O. Претходни упит могуће је написати и на следећи начин:

SELECT Ime, Prezime

FROM Ucenik U JOIN Ocena O

ON U.JMBG = O.JMBGUcenika

WHERE Ocena = 5

Или пошто се у овом случају јединствени називи атрибута по којима вршимо спајање, не морамо имена атрибута квалификовати називима табела:

SELECT Ime, Prezime

FROM Ucenik JOIN Ocena

ON JMBG = JMBGUcenika

WHERE Ocena = 5

Уколико је потребно да се зада упит над више табела, најпре се у оквиру левог дела прозора морамо позиционирати на базу података у којој се налазе те табеле. Након тога је потребно да се одговарајући упит унесе у оквиру поља за унос текста на картици SQL. Након уноса упита и клика на дугме Go приказује се нови прозор који садржи текст упита и након тога резултат упита приказан у виду табеле.

Уколико бисмо хтели да надоградимо претходни упит и да поред имена и презимена ученика прикажемо и називе предмета из којих је ученик добио оцену 5, било би потребно да извршимо и спајање са табелом у којој се налазе називи предмета, а то је табела Predmet. Спојили бисмо табелу Predmet с табелом Ocena по критеријуму да је у питању исти предмет, тј. да буде иста вредност поља IdPredmeta. SQL упит којим би се ово издвајало изгледао би на следећи начин:

SELECT Ime, Prezime, Naziv

FROM Ucenik U JOIN Ocena O

ON U.JMBG = O.JMBGUcenika

JOIN Predmet P

ON P.IdPredmeta = O.IdPredmeta

WHERE Ocena = 5

Овим упитом издвојили бисмо спојено у једном реду име и презиме ученика и назив предмета из кога је добио оцену 5.

Слика: 101\_upit\_spajanje\_tri\_tabele.png

Потпис: Резултат упита који укључује спајање три табеле

Претпоставимо да је потребно да се издвоје сви парови матичних бројева оних ученика који се исто зову. Тада је потребно да се размотре две копије табеле Ucenik и да се међусобно споје. Наравно, овде је концептуални проблем како да се разликују те две копије и како да се међусобно споје. Овде се као згодан механизам показало увођење алијаса за табеле. Наиме, ове две копије именоваћемо на различит начин у оквиру ставке FROM и затим ћемо их спојити коришћењем имена колона која су квалификована новим именима табела.

SELECT Prva.JMBG, Druga.JMBG

FROM Ucenik Prva JOIN Ucenik Druga

ON Prva.Ime = Druga.Ime

У резултату упита добили бисмо наредне редове:

Слика: 103\_upit\_spajanje\_iste\_tabele.png

Потпис: Резултат упита који укључује спајање табеле са самом собом

Треба скренути пажњу на то да ћемо овим упитом добити и све парове ученика са самим собом. То се може елиминисати додатним условом да су матични бројеви ова два ученика различити. И поред тог услова добиће се неки редундантни подаци јер за два ученика који се исто зову у резултату ће се добити два реда, по један за сваку пермутацију. И ово се може избећи наметањем уређења на редове у резултату упита, у овом случају додавањем услова да је, на пример, матични број првог ученика већи од матичног броја другог ученика.

SELECT Prva.JMBG, Druga.JMBG

FROM Ucenik Prva JOIN Ucenik Druga

ON Prva.Ime = Druga.Ime

WHERE Prva.JMBG > Druga.JMBG

Овим бисмо у резултату добили само наредне редове:

Слика: 104\_upit\_spajanje\_iste\_tabele\_bez\_ponavljanja.png

Потпис: Резултат упита који укључује спајање табеле са самом собом без понављања

Уколико је потребно да се, на пример, прикажу сви парови матичних бројева ученика, опет су нам потребне две копије табеле Ucenik, али их овај пут не треба спајати јер не постоји критеријум спајања. Сада је потребно да се наведу све табеле из којих издвајамо колоне у оквиру ставке FROM и да се међусобно раздвоје запетама. Тиме се заправо задаје Декартов производ две табеле, тј. сваки ред прве табеле је у релацији са сваким редом друге табеле. Да бисмо избегли претходно поменуте редудантне парове, додајемо услов да је матични број првог ученика већи од матичног броја другог ученика.

SELECT Prva.JMBG, Druga.JMBG

FROM Ucenik Prva, Ucenik Druga

WHERE Prva.JMBG > Druga.JMBG

Резултат овог упита је следећи:

Слика: 106\_upit\_Dekartov\_proizvod\_tabela.png

Потпис: Резултат упита који укључује Декартов производ табела

### Упити који укључују агрегатне функције

Некада је потребно да се изброје редови у резултату упита или да се, на пример, израчуна збир свих вредности у некој колони. За овакве потребе користе се агрегатне функције. То су функције које извршавају неку операцију над колоном вредности и као резултат враћају скаларну вредност. У SQL-у су подржане следеће агрегатне функције:

* COUNT – враћа број вредности у датој колони (или број редова у резултату упита),
* SUM – враћа збир вредности у датој колони,
* AVG – враћа средњу вредност вредности у датој колони,
* MAX – враћа максималну вредност у датој колони,
* MIN – враћа минималну вредност у датој колони.

Агрегатне функције имају као аргумент назив колоне над којом рачунају вредности (или израз у коме учествује колона) и могу се наћи само у линији SELECT, а никако у услову упита датим ставком WHERE.

Ако се агрегатној функцији COUNT проследи као аргумент назив колоне, онда се пребројавају редови у којима вредност ове колоне није недостајућа. На пример, ако желимо да израчунамо за колико ученика је познат датум рођења, то се може урадити коришћењем агрегатне функције COUNT, тако што ћемо јој као аругмент проследити колону DatRodjenja.

SELECT COUNT(DatRodjenja) poznat\_datrodj

FROM Ucenik

Овим упитом добили бисмо следећу вредност као резултат:

Слика: 108\_access2010\_upit\_count.png

Потпис: Резултат упита који укључује пребројавање познатих вредности у колони

Aко називу колоне претходи кључна реч DISTINCT, онда се елиминишу дупликати из колоне пре примене агрегатне функције.

Када се агрегатној функцији COUNT као аргумент проследи \*, тада се њоме пребројавају редови у резултату упита. На пример, ако желимо да избројимо колико има професора у школи, то се може урадити на следећи начин:

SELECT COUNT(\*) ukupno\_prof

FROM Profesor

Као резултат упита добили бисмо следеће:

Слика: 109\_upit\_count\_zvezda.png

Потпис: Резултат упита који укључује пребројавање редова у резултату упита

Такође, ако желимо да издвојимо просечну оцену ученика са матичним бројем 2105001735322 из предмета са шифром 201, то се може урадити следећим упитом:

SELECT AVG(Ocena+0.0) prosek

FROM Ocena

WHERE JMBG = 2105001735322 AND IdPredmeta = 201

Као резултат упита добили бисмо следеће:

Слика: 110\_upit\_avg.png

Потпис: Резултат упита који укључује рачунање просечне вредности колоне

Потребно је обратити пажњу на чињеницу да се просек целобројних вредности рачуна као целобројна вредност, због чега је неопходно да се пре примене агрегатне функције аргументи преведу у реалне бројеве, што се може учинити додавањем вредности 0.0.

Ако бисмо желели да за овог ученика израчунамо просечну оцену из неког другог предмета, могли бисмо само да променимо шифру предмета у упиту. Шта ако бисмо хтели да за овог ученика израчунамо просечну оцену из сваког предмета? У том случају може се применити оператор груписања GROUP BY. Тако се редови деле на групе да редови у истој групи имају исту вредност на колонама из ставке GROUP BY, а редови из различитих група различиту вредност. На тај начин може се применити нека агрегатна функција на сваку од група посебно, као у следећем примеру:

SELECT AVG(Ocena+0.0) prosek

FROM Ocena

WHERE JMBG = 2105001735322

GROUP BY IdPredmeta

У резултату упита добили бисмо следеће редове:

Слика: 111\_upit\_avg\_group\_by.png

Потпис: Резултат упита који укључује рачунање просечне вредности колоне на групама

Овим упитом се добијају просечне оцене за сваки школски предмет, али нећемо знати која се просечна оцена односи на који предмет. Правило је да се уз агрегатну функцију у линији SELECT додатно могу наћи само колоне по којима се врши груписање јер је вредност тих колона иста на целој групи. У овом случају имало би смисла да се уз просечну оцену издвоји и идентификатор предмета.

SELECT IdPredmeta, AVG(Ocena+0.0) prosek

FROM Ocena

WHERE JMBG = 2105001735322

GROUP BY IdPredmeta

Сада бисмо у резултату упита добили и информацију о томе на који се предмет односи која просечна оцена.

Слика: 112\_upit\_avg\_group\_by\_1.png

Потпис: Резултат упита који укључује рачунање просечне вредности колоне на групама са идентификатором групе

Уколико су нам значајне само неке групе, оне се могу изабрати ставком HAVING – то су оне групе на којима је вредност агрегатне функције у неком опсегу. На пример, ако нас интересују само просечне оцене овог ученика, под условом да је просечна оцена већа или једнака 4, то се може урадити на следећи начин:

SELECT IdPredmeta, AVG(Ocena+0.0) prosek

FROM Ocena

WHERE JMBG = 2105001735322

GROUP BY IdPredmeta

HAVING AVG(Ocena+0.0)>=4

Слика: 113\_upit\_avg\_having.png

Потпис: Резултат упита који укључује рачунање просечне вредности колоне само на групама од интереса

### Упити који укључују унију

Понекад је потребно да се заједно прикажу резултати два различита упита. На пример, ако желимо да прикажемо обједињено имена и презимена свих ученика и свих професора у једној табели, могло би да се посебно издвоје имена и презимена ученика, а посебно имена и презимена професора, а затим да се направи унија резултујућих редова. За прављење уније у SQL-у користи се кључна реч UNION. Упити које желимо да искомбинујемо оператором уније морају да задовољавају неке услове: да враћају једнак број колона и да одговарајуће колоне буду истих или компатибилних типова (може се, на пример, унирати колона која узима вредности на скупу целих бројева и колона која узима вредности на скупу реалних бројева).

Поменути упит може се реализовати на следећи начин:

SELECT Ime, Prezime

FROM Ucenik

UNION

SELECT Ime, Prezime

FROM Profesor

Слика: 115\_upit\_union.png

Потпис: Резултат упита који укључује рачунање уније два скупа вредности

### Наредбе којима се мењају подаци

Измена података у бази података укључује:

* додавање нових података,
* брисање података,
* ажурирање вредности података.

Нови подаци додају се наредбом INSERT и детаље синтаксе ове наредбе и начина њене употребе упознали смо у поглављу које се односи на прављење табела. Наредба за брисање података је DELETE и њу смо обрадили у истом поглављу.

Вредности у оквиру табеле могу се мењати наредбом UPDATE, а она има следећу синтаксу:

UPDATE <tabela>

SET <dodele>

[WHERE <uslov>]

Притом, <dodele> могу бити изрази следећег облика:

<kolona> = <izraz>

Међусобно су раздвојени запетама или може да буде групна додела следећег облика:

(<lista\_kolona>) = (<lista izraza>)

Пре него што научимо како да изменимо податке, погледајмо како је тренутно попуњена табела Predmet. То се може постићи следећом наредбом:

SELECT \* FROM Predmet

Њен резултат изгледа овако:

Слика: 117\_upit\_update\_all.png

Потпис: Резултат упита који укључује ажурирање вредности свих редова табеле

Ако желимо да предмету Математика изменимо фонд на пет часова недељно, то може да се уради на следећи начин:

UPDATE Predmet

SET Fond = 5

WHERE Naziv = 'Matematika'

Вредности реда тебеле Predmet које се односе на школски предмет под називом Математика можемо да излистамо следећом наредбом:

SELECT \* FROM Predmet

WHERE Naziv = 'Matematika'

Након тога се добија следећи резултат:

Слика: 118\_upit\_pre\_update.png

Потпис: Резултат упита који укључује прегледање иницијалног попуњавања табеле

Можемо приметити да је вредност колоне Fond предмету Математика постављена на 5. Ако бисмо хтели да свим предметима смањимо фонд за 1, то се може учинити следећом наредбом:

UPDATE Predmet

SET Fond = Fond-1

Излистајмо сада све вредности реда табеле Predmet следећом наредбом:

SELECT \* FROM Predmet

Након тога ће се добити следећи резултат:

Слика: 119\_upit\_update\_all.png

Потпис: Резултат упита који укључује ажурирање вредности свих редова табеле

Можемо приметити да је вредност колоне Fond свим предметима смањена за 1. Aко бисмо, пак, хтели да предмету Рачунарство и информатика истовремено повећамо фонд за 1 и да поставимо да је обавезан, то бисмо могли да постигнемо на следећи начин:

UPDATE Predmet

SET Fond = Fond+1

Obavezan = 'Da'

WHERE Naziv = 'Računarstvo i informatika'

Као што смо већ напоменули, приликом позиционирања на неку од табела и одабира картице SQL, у дну прозора се приказују наредбе које се могу извршити. Међу њима су и наредбе INSERT за додавање нових редова, UPDATE за измену вредности и DELETE за брисање редова. Кликом на неко од ових дугмади аутоматски се генерише костур одговарајуће наредбе, па је само потребно да се попуне потребне вредности на одговарајућим местима.

Слика: 122\_phpmyadmin\_insert\_update\_delete.png

Потпис: Подршка за наредбе додавања, измене и брисања редова у окружењу phpMyAdmin